8-7-2018

DEEP LEARNING IN CHEMISTRY AND COMPUTER-GO

Mengyuan Zhu

DEEP LEARNING IN CHEMISTRY AND COMPUTER-GO

by

MENGYUAN ZHU

Under the Direction of Yi Pan, PhD

ABSTRACT

Deep learning a research field in artificial intelligence and also a fast-growing technology in helping human in different directions. This thesis will focus on two of its usages: chemistry and computer go. In the two fields, deep learning achieves state of art accuracy in prediction and game playing ability.

INDEX WORDS: Deep learning, Chemistry, Computer-Go

Deep Learning in Chemistry and Computer-Go

by

MENGYUAN ZHU

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science

in the College of Arts and Sciences

Georgia State University

2018

DEEP LEARNING IN CHEMISTRY AND COMPUTER-GO

by

MENGYUAN ZHU

Committee Chair: Yi Pan

Committee: Yanqing Zhang

Zhipeng Cai

Electronic Version Approved:

Office of Graduate Studies

College of Arts and Sciences

Georgia State University

August 2018

DEDICATION

This study is wholeheartedly dedicated to our beloved parents, who let me grow up freely without any limitations.

ACKNOWLEDGEMENTS

Firstly, I would like to express my sincere gratitude to my advisor Prof. Yi Pan for the continuous support of my deep learning research. Besides, I would like to thank the rest of my thesis committee Dr. Yanqing Zhang and Dr. Zhipeng Cai.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AI – Artificial Intelligence

DL – Deep Learning

CNN – convolution Neural Network

RNN – Recurrent Neural Network

LSTM – Long short term memory

SMILES – Simplified Molecular-Input Line-Entry System

NLP – Natural Language Processing

GPU – Graphics Processing Unit

1 DEEP LEARNING IN CHEMISTRY

After 2012 ImageNet competition, artificial intelligence (AI) technologies have become very useful tools in solving scientific and life problems. Specifically, deep learning (DL) especially convolutional neural network (CNN) shows its powerful ability in many fields including image, text, etc. Thus in my research, I conducted deep learning studies in chemistry to see how better we can utilize the state-of-the-art deep learning into real problems.

1.1 Purpose of the Study

In the field of machine learning in chemistry there is a long history. People have tried to use different machine learning methods in solving drug discovery problems. Especially in the area of small molecules properties prediction, there is a famous rule called Lipinski Rule of 5 in the beginning. After that, people tried to use machine learning to solve properties prediction called ADMET. ADMET are 5 items for prediction. They are adsorption, distribution, metabolism, excretion and toxicity. Medicinal chemists used nearly all machine learning methods including support vector machine (SVM), neural network (NN), decision tree (DT), random forest etc. They all give good results. However, the most important part in machine learning for drug discovery is feature extraction and selection.

In chemistry, all inputs are small molecules. So it would be quite difficult compared with other fields of which all features are numbers. In chemistry, we have to extract features. Thus it would be interesting if there is a feature extraction method. Traditionally, medicinal chemists extract features by counting. For example, people can count how many atoms in a molecule, how many carbon atoms in a molecule, how many bonds in a molecule, molecular weight and son. Such as this, there are a bunch of software to extract feature even thousands of features. By using this method, people use these features inputs to feed machine learning models. Many models give good

results. However, there are many problems because of these methods. The reasons are that it takes time to get these features because we have to develop a program to calculate features. Some features are difficult calculate in terms of time, especially 3D features and quantum calculation based features. Because if you want to get a 3D feature you have to generate a 3D model and for quantum chemistry calculation it may take seconds. Thus it would be very time consuming if there are thousands of molecules. It is not a problem maybe 20 years ago because for that time people used QSAR to do such work in which there are only small datasets. Nowadays we go to big data era and there are so many molecules. Thus there is a need to extract features very fast. The second reason is the feature selection part. Because there are thousands of feature, they are not easy for machine learning. Such a big amount of features will cause dimension curse because models can't learn from a small dataset of molecules. Another problem of these many features is underfitting. So there is a need to develop a good method to find good a good way to extract and select features. That's why I plan to use deep learning to help solve this problem (Figure 1.1) by using SMILES.

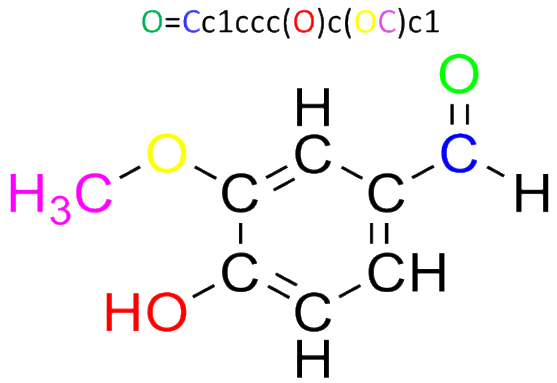

Figure 1.1 SMILES representation for one molecule.

1.2 Experimental Design

1.2.1 Data preparation

To prepare the dataset the SMILES data of CYP1A2 inhibitor was used. The dataset its self is SMILES data of a list of small molecules with negative or positive properties. Negative means they are not CYP1A2 inhibitors while positive means they are CYP1A2 inhibitors. All

molecules data were preprocessed by adding protonation states on them because for CYP1A2 is very sensitive with positive charged molecules. For CYP1A2 inhibition negative molecules they will be labeled 0 while for positive molecules they will be labeled 1 (Figure 1.2).

```
Cc1cccc(c1NC(=O)CSc2nc(c(c(c2C#N)C)CC(=O)c3ccc(cc3)Br)C)C 0
COc1cccc(c1)c2cc(n3ncc(c3n2)S(=O)(=O)c4ccccc4)C(F)(F)F 0
Clc1nccccc1NS(=O)(=O)c2ccc3c(c2)OCCO3 0
Cc1cccc(c1NC(=O)COc2cccc(c2)C=NNC(=O)C(=O)NCc3ccccc3)C 0
CC(=NOCC(O)COCc1occc1)CCN2CCCc3c2cc(c(n3)C)C 0
Oc1nc(c2[nH]c(=O)n(c2n1)c3ccccc3)[S-] 0
CC1=NN(C(=O)C1N=Nc2ccc(c(c2)O)[As](=O)([O-])[O-])c3ccccc3 1
CC1=NN(C(=O)C1N=Nc2ccc(cc2)S(=O)(=O)N)c3ccccc3 1
Nc1nc(c(c(n1)N)N=Nc2ccc(c(c2)O)[As](=O)([O-])[O-])N 1
O=C1CC(=[NH+]N1c2ccccc2)Nc3ccccc3 1
CC(=O)N(N=C1Sc2ccccc2C1=O)c3ccccc3 1
Cn1n(c(=O)c(c1C)NCc2cc(ccc2O)Cl)c3ccccc3 1
CN(C)c1c(n(n(c1=O)c2ccccc2)C)C 1
CC1=NN(C(=O)C1N=Nc2ccc3ccccc3c2)C(=O)c4ccncc4 1
OCCN(CCO)C=[NH+]C1=[NH+]N(CC1)c2cccc(c2)C(F)(F)F 1
CC(Sc1cc(c2ccccc2n1)C)C(=O)Nc3c(n(n(c3=O)c4ccccc4)C)C 1
CCCC[c-]1c(=O)n(n(c1=O)c2ccc(cc2)O)c3ccccc3 1
CCCn1c2c(c(c(c1=O)C(=O)NCc3ccccc3)[O-])CCCC2 1
NS(=O)(=O)c1ccc(cc1)N=Nc2cnc(c(c2)O)O 1
```

Figure 1.2 SMILES representation with labels for molecules.

1.2.2 Data indexing

Because computer can't understand molecules in SMILES format. There is a need to treat them as text to prepare the data. Everything is similar to word2vec. So firstly all SMILES data were converted to 1D vectors based on indexing.

Figure 1.3 Preprocessing for SMILES data.

1.2.3 Data splitting

The dataset was split with a training set and a test set with a ratio 4:1

1.2.4 Deep learning model building

All deep learning models were built with Keras functions. Layers include embedding, lstm, conv, dense, pooling, activations, etc. Parameters were searched with different configuration to try to find the best one.

1.2.5 Deep learning model training

Adam was used to optimize the model with binary cross entropy as a loss function. All calculations are conducted on GPUs. Early stopping was used when the validation set accuracy does not move up by 10 epochs. Totally 100 epochs training was conducted with a batch size of 100.

1.3 Experimental Results

1.3.1 Data preparation for deep learning

Data was prepared with SMILES data (Figure 1.3) with a Python script below. SMILES is short for Simplified Molecular-Input Line-Entry System. SMILES is easy compared with other molecules structure like mol2 or SDF because SMILES is a one-line representation of molecules and It does not store the 3D information. It is a linear notation of chemical structures and molecules in this format can be easily translated to many 3D structures. The reason for me to use SMILES is that it saves a lot of space because of the linear notation feature. Otherwise for other formats like mol2 I have to prepare a line for each atom. Thus it would be easier to define a reader to read these sequences.

THE SMIELS format is very easy to understand. For atoms, they are represented using standard abbreviation of atom names. For example, carbon atom be written as C. For bonds,

commonly there are 3 kinds of bonds: single bonds written as “-”, double bonds written as “=” and triple bonds written as “#”. Usually for single bond the “-” sign is usually omitted. For rings, people can label them with numbers for the two atoms connected. Thus if there are more than 1 rings people can just label more atoms. For branches, parentheses can be added after an atom. The first atom in the parentheses is the atom connected.

SMILES is so easy to represent a molecule thus we can use it for deep learning inputs. Because SMILES representation for a molecule is a string of characters, we can consider them as texts. So it is easy to use simple natural language processing (NLP) method to build machine learning models. In NLP, there is an interesting concept called word2vec. This term is to translate a word into a vector. An interesting example is that king – man + woman equals to queen. Thus to implement a similar work, I tried to index all characters first with a simple code below:

```
#Find characters of sequences and build a library
for index, SMILESsequence in X_SMILES.iterrows():
    for letter in SMILESsequence[0]:
        if not letter in char_lib:
            char_lib.append(letter)

X=[]
#SMILES sequence to an array
for index, SMILESsequence in X_SMILES.iterrows():
    sequenceArray=[]
    for letter in SMILESsequence[0]:
        sequenceArray.append(char_lib.index(letter))
    X.append(sequenceArray)
```

By calculating the results totally there are 11,922 data items. The longest length for SMILES is 226 which is used as the input length. For SMILES data with length less than 226 they are filled with zeros. Totally there are 43 different characters after indexing.

All the data was split randomly as a training set and a test set with ratio 4:1. Training set will be used for machine learning models training while test set will be used for validate the model

robustness. Finally, the feature contains only numbers of the index of characters and the label is 0 and 1 depending whether they are CYP1A2 inhibitors or not.

1.3.2 Deep learning study

It is critical to build a deep learning model because for shallow machine learning models the parameters tuning part is easy. While for deep learning there many architectures available. Deep learning is a kind of neural network but regularly with more layers. Neural network was a very hot idea 20 years ago. However, because during that time the computational resource is not power and the data is small, neural network has a lot of problems and other machine learning methods are widely used like SVM and RF. Interestingly, with the advanced technology improvement in graphics processing unit (GPU) and the storage/collection method. We can easily computer things more quickly and collect and store more data. Thus, previous shallow machine learning methods are not able to achieve good result because of underfitting.

Neural network borrows the idea of biological neurons. So in a neural network the basic components are neurons and connections. A neural network has many layers. Each layer has many neurons. For the simple neural network, a neuron only connects with all neurons in its previous layer and next layer. Each neuron is a number and the connections are weights and biases. The first layer is also called input layer and the final layer is also called output layer. Backpropagation is the method to update weights by calculating errors from the output and the real result. Some kinds of layers are important to know before conducting deep learning study.

Fully connected layer: it is a layer with all neurons connect directly from the previous layer to its layer and next layer to this layer. This is the most time-consuming layer because there is a matrix multiplication during this process. This is most common layer in the early version of neural networks.

Activation layer: it is a layer to introduce nonlinearity. There are many kinds of activation layers such as sigmoid and relu.

Convolution layer: it is a layer to extract features with many small filters. It is very useful in extracting high-level information from images. It also shows its power in NLP.

Pooling layer: it is a layer to reduce the number of features and computation. People also think it will help if there are small changes to the photo as they are not sensitive to pooling layer. Whether this concept is true is still under question.

Dropout layer: it is a layer to reduce the overfitting problems by randomly cut some connections between neurons.

Batch normalization layer: it is a layer to reduce the overfitting problem by changing the distribution of values.

Embedding layer: it is a very useful layer in NLP because it converts a number to a vector of numbers. It saves time and has more information compared with 1 hot representation.

After 200 experiments with different hyperparameter study, I finally determined to use an architecture with convolutional neural network shown in Figure 1.4.

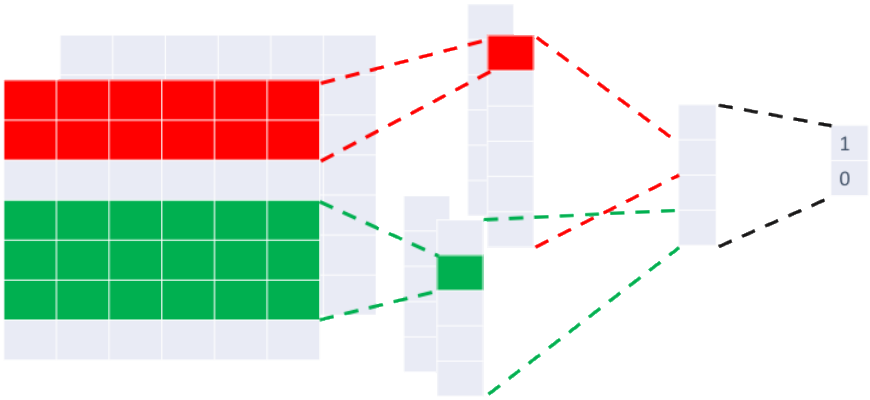

Figure 1.4 CNN architecture of SMILES prediction.

The details architecture can be found in Figure 1.5. The first layer is an input layer with length 226 because the longest sentence has 226 characters. The second layer tis an embedding layer to convert the inputs into vectors with each vector length 50. A dropout layer was then used to reduce overfitting problem. Then it connects with different convolutional layers. Each convolutional layer has different number of kernel size. They are 3, 5 and 7 respectively. For filters the numbers are same: 128. Then leaky relu activation function was applied to each convolutional layer with an alpha value of 0.3. A dropout layer was again used to connect the activation layer to global max-pooling layer. After then, all layers were added together following by a dense layer with 500 neurons. A dropout layer was again used with a dropout rate 50%. A leaky relu activation was used with alpha value of 0.3. Another dense function with a sigmoid activation layer was used to predict the label.

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 226)	0	
embedding_1 (Embedding)	(None, 226, 50)	2150	input_1[0][0]
dropout_1 (Dropout)	(None, 226, 50)	0	embedding_1[0][0]
conv1d_1 (Conv1D)	(None, 220, 128)	44928	dropout_1[0][0]
conv1d_2 (Conv1D)	(None, 224, 128)	19328	dropout_1[0][0]
conv1d_3 (Conv1D)	(None, 222, 128)	32128	dropout_1[0][0]
batch_normalization_1 (BatchNorm	(None, 220, 128)	512	conv1d_1[0][0]
batch_normalization_2 (BatchNorm	(None, 224, 128)	512	conv1d_2[0][0]
batch_normalization_3 (BatchNorm	(None, 222, 128)	512	conv1d_3[0][0]
leaky_re_lu_1 (LeakyReLU)	(None, 220, 128)	0	batch_normalization_1[0][0]
leaky_re_lu_2 (LeakyReLU)	(None, 224, 128)	0	batch_normalization_2[0][0]
leaky_re_lu_3 (LeakyReLU)	(None, 222, 128)	0	batch_normalization_3[0][0]
global_max_pooling1d_1 (GlobalMa	(None, 128)	0	leaky_re_lu_1[0][0]
global_max_pooling1d_2 (GlobalMa	(None, 128)	0	leaky_re_lu_2[0][0]
global_max_pooling1d_3 (GlobalMa	(None, 128)	0	leaky_re_lu_3[0][0]
concatenate_1 (Concatenate)	(None, 384)	0	global_max_pooling1d_1[0][0] global_max_pooling1d_2[0][0] global_max_pooling1d_3[0][0]
dense_1 (Dense)	(None, 250)	96250	concatenate_1[0][0]
dropout_2 (Dropout)	(None, 250)	0	dense_1[0][0]
leaky_re_lu_4 (LeakyReLU)	(None, 250)	0	dropout_2[0][0]
dense_2 (Dense)	(None, 1)	251	leaky_re_lu_4[0][0]
activation_1 (Activation)	(None, 1)	0	dense_2[0][0]

Figure 1.5 Detailed architecture.

1.3.3 Results and conclusion

The deep learning model achieves a good result with accuracy 0.819 and ROC_AUC 0.884 on the validation set. It is better than published data PKCSM and ADMETSAR shown in Table

1.1. It is the first to use convolutional neural network to solve chemistry problem. This result suggests that deep learning is good choice when applying machine learning in predicting drug properties. However, there is still a need to use deep learning to implement other chemistry models. This is not done by me because I don't have time to work on that. And I believe other people will also do a good job by applying more novel technologies.

Table 1.1 Performance of Deep SMILES and published results

METHOD	**ACCURACY**	**ROC_AUC**
DEEP-SMILES	0.819	0.884
PKCSM [1]	0.802	0.876
ADMETSAR [2]	0.815	0.815

2 DEEP LEARNING IN COMPUTER-GO

2.1 Introduction

Go is a traditional China chess game. It is so interesting so that I learnt it when I was 10 years old. When I was a child, I can hardly believe it is true that a computer can beat a professional Go player because it is so hard and complex. It is well-known that the number of steps is more than the number of atoms in the universe. It will use a lot of computational power so most of the go computer programs can't even win me when I was 14 years old, the time I stopped learning the game of go.

It is amazing that Google DeepMind developed a computer-go program AlphaGo[3] which beat professional go players. One of the methods behind is deep learning. It uses convolutional neural network to treat a go play as an image. Thus it would be very interesting whether we can use the latest convolutional neural network to give better result.

2.2 Background

2.2.1 Deep learning architectures in computer vision

In the field of image recognition, deep learning is currently the most successfully method. Deep learning is kinds of a neural network but with more hidden layers than regular artificial neural network. Previously, there are only dense layers as hidden layers. Dense layers are all connected layers. Because people use backpropagation to update errors, all connected layers will consume a lot of computational power. Thus, this method is suitable for machine learning in the early days when the hardware is not that powerful.

2.2.1.1 LeNet-5

However, in 1998, LeCun Yann developed LeNet-5[4] as shown in Figure 1. In this neural network, LeCun introduced two different kinds of layers. One is convolution layer, which used filter to generate features. The other is pooling layer, which is used for reducing the size of generated features. So for LeNet-5, there are totally 8 layers: input layer, convolution layer, maxpooling layer, convolution layer, maxpooling layer, two dense layers and 1 output layer which is also a dense layer. This neural network is very good in written number recognition with accuracy 99.3%.

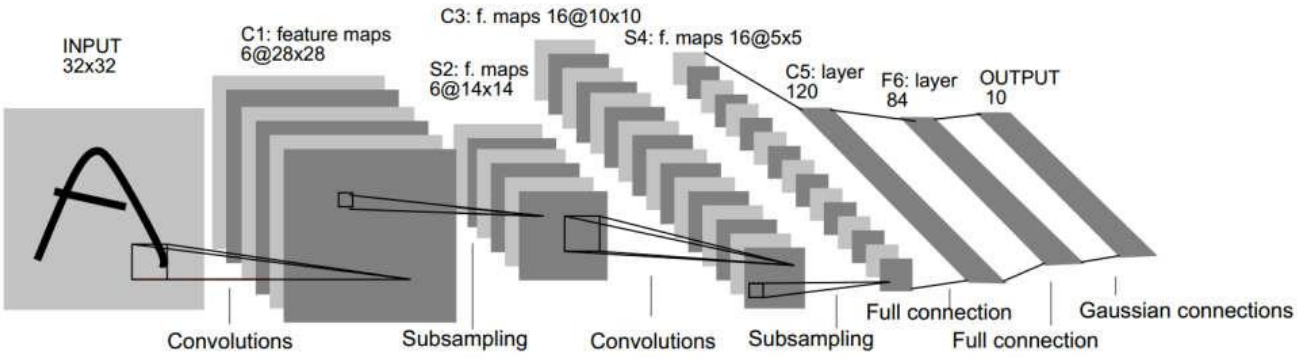

Figure 2.1 Architecture of LeNet-5

2.2.1.2 AlexNet

Then neural network "died" for about 14 years because of the computational power limitation. In 2012, AlexNet [5] was shown to be very successful in ImageNet, which is a competition for image classification, because it beat the number 2 player with 10% more accuracy. The major reasons for the success of AlexNet are: 1. It used ReLu for non-linearity; 2. Data augmentation was used such as image translations, horizontal reflections; 3. Dropout was used reduce overfitting. The architecture can be seen in Figure 2.

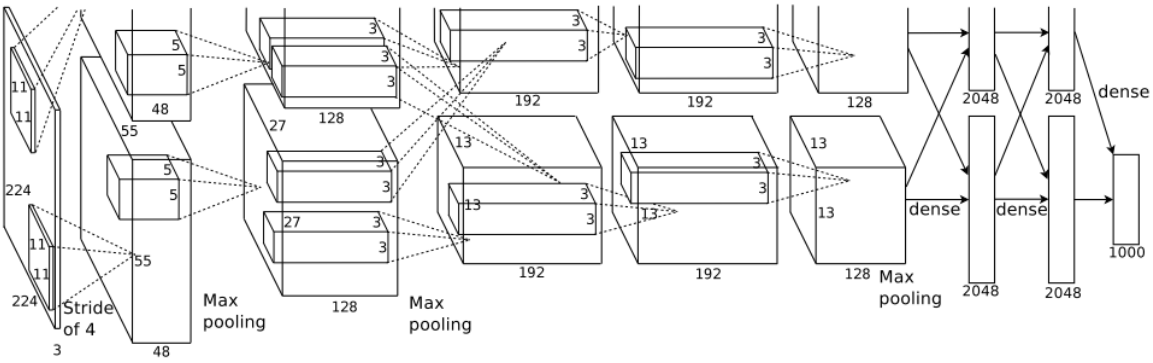

Figure 2.2 Architecture of AlexNet

2.2.1.3 VGG

Different from AlexNet, VGG[6] used very small filters in the convolution layer. As in LeNet- 5 or Alexnet, their size is 11X11 in the first layer. However, VGG used 3X3 for all layers. The is because 2 3X3 layer is equal to 1 5X5 layer. This in turn simulates a larger filter while keeping the benefits of smaller filter sizes. This method achieves better accuracy with 7.3% error rate.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 2.3 Architecture of VGG

2.2.1.4 Inception

Then winner for year 2014 is actually GoogleLenet[7] with error rate 6.7%. Their innovation is that they developed inception module. The naïve model has 4 options for machine to pick. They are 1x1 convolution, 3x3 convolution, 5x5 convolution and 3x3 maxpooling. However, this will have a lot of parameters for training. Thus the authors developed a new idea of 1x1 convolution layer before 3x3 and 5x5. By using this method, they can reduce the number of parameters.

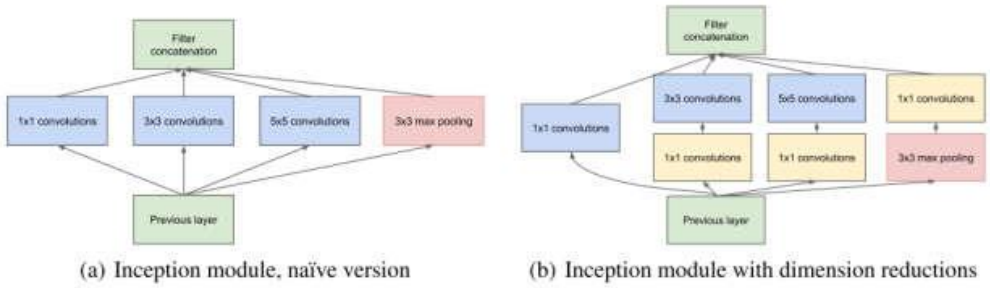

Figure 2.4 Architecture of inception

2.2.1.5 Network in Network

For Network in Network[8] authors, they introduced mlp layer, which is used to increase the ability of convolution layer. Then they used global maxpooling to reduce the number of parameters. This method has less parameters, making it faster.

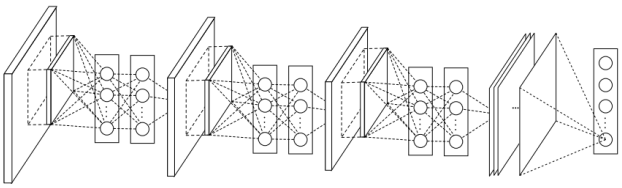

Figure 2.5 Architecture of "network in network"

2.2.1.6 ResNet

ResNet[9] is the new winner of ImageNet with 3.6 error rate. In their structure, they used skip connection to avoid gradient vanishing. As shown in Figure 5, there are 5 different architectures for the skip connection implementation. As in their paper, they found that full pre- activation is the best. In the bottleneck, they should follow these steps: BN, ReLU, weight, BN, ReLU and

weight. They best model has more than 1000 layers.

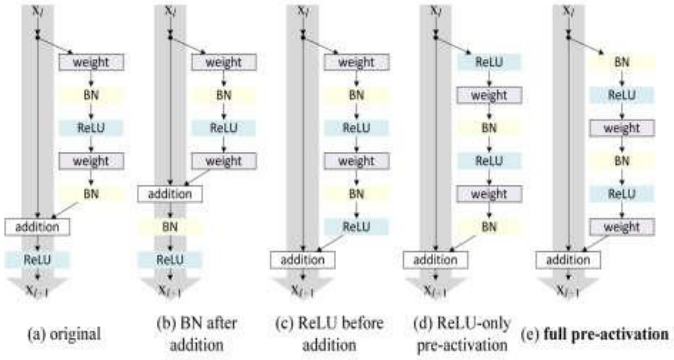

Figure 2.6 Architecture of Residual Network.

2.2.1.7 Wide Residue Network

Residual network is very long. So Wide Residue Network[10] authors came up with a new network which is short but wide[8]. What they mean by wide is that the k value in neural network is larger than 1. This gives better accuracy: 3.89% error rate.

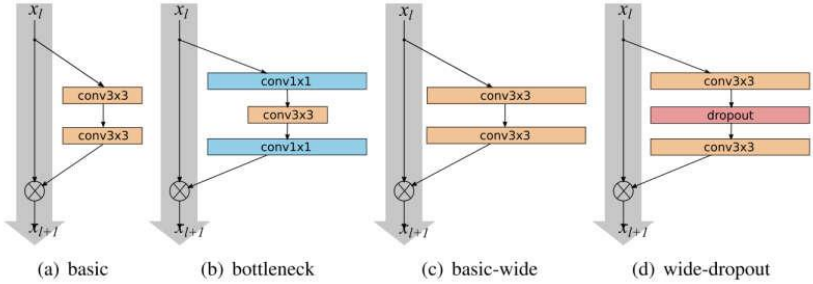

Figure 2.7 Architecture of Wide Residual Network.

2.3 Experimental design

2.3.1 Dataset preparation

Gogod database was used to get go plays with SGF file format. Only data after 1980 was used because of the improvement of go theory in the last 26 years. To extract the "image" information from the go plays. Same as AlphaGo, each step is treated as one image to prepare input files. 1% of the dataset was used as a validation set.

2.3.2 Feature preparation

Same as AlphaGo, 48 features were used to prepare as inputs. They are called planes. The planes are:

1. Stone color, which has 3 features for player stone, enemy stone and empty, respectively.
2. Ones feature, which is 1 plane of ones.
3. Turns since feature, which is a list of the last 8 steps.
4. Liberties feature, which is a list of liberties of black stones of while stones. The maximum liberty is 8. If it has more than 8 stones it was treated 8 liberties.
5. Capture feature, which is the number of stones to be captured. The maximum number of stones is 8 and if the more than 8 stones it will be treated as there are 8 stones to be captures
6. Self-atari feature, which is the like 5 but it is the number of stones to be captures in our side.
7. Liberties after move feature, it is a feature of number of stones after the move
8. Ladder capture feature, it is to see whether a position can capture enemies' stones using ladder method.
9. Ladder escape feature, it is also to see whether a position can leave from enemies' ladder tactics.
10. Sensibility feature, whether a step is legal and does not fill its eyes.
11. Zeros feature, same as ones feature but they are all zeros.

2.3.3 Deep learning model building

The policy network borrows ideas from ResNet. The first layer is the same as AlphaGo, it pads zeros to make 23X23 images and the convolve with 5X5 kernel size filters with stride 1. Then there is an activation layer plus a batch normalization layer. After that they are resnet blocks which is same as ResNet. For all ResNet variations, the full pre-activation architecture was adopted. Totally there are 19 residue blocks. The last layer is a softmax layer to map all possible next steps.

2.3.4 Deep learning model training

Gradient descent method was used to optimize the model. The initial learning rate is 0.01. Learning rate decay was applied for every 1milllion steps. The batch size is 32. 1 GPU card NVIDIA Tesla P100 was used to train the model for 18 days.

2.4 Results

Because ResNet shows very good result in images recognition tasks and it is very computation efficient, it was used to replace the old architectures of AlphaGo to train the network. The next of each go play was used to label. So, it is purely a supervised learning in the AlphaGo project. The reason more studies were not conducted is because I don't have enough computers to work on that.

2.4.1 Feature test

To understand how the feature planes affect the prediction accuracy, different features based calculation was conducted. As we can see from features 2.8 to feature 2.15, the more features, the better the result. Ladder features do not improve the results but helps because it is very difficult to find this tactic. The new AlphaGo zero does not have feature. They train theirs using a pretty long time.

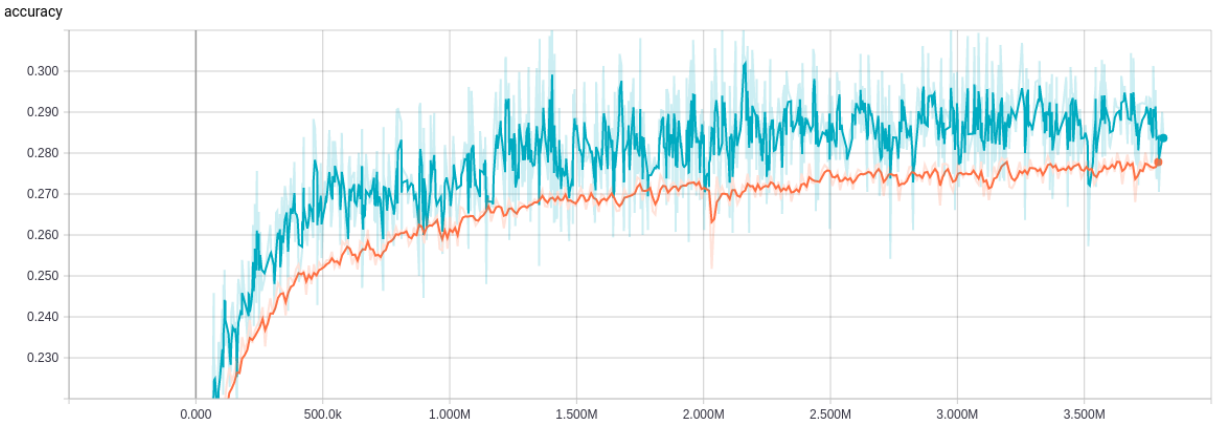

Figure 2.8 Performance of Shouzhuo with final test-set accuracy 0.277.

Parameters: 5 layers' convolutional neural network and 10 epochs. Features and number feature planes: stone color and ones: 3 and 1.

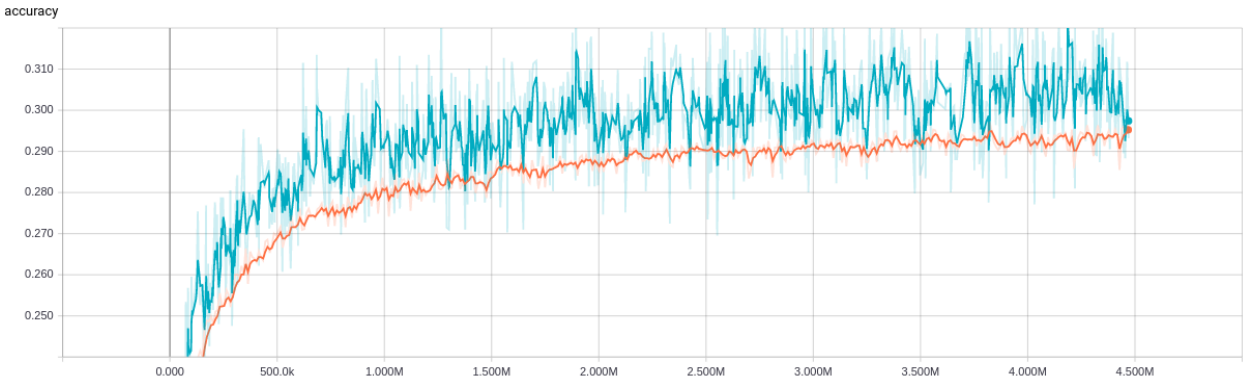

Figure 2.9 Performance of Shouzhuo with final test-set accuracy 0.295.

Parameters: 5 layers' convolutional neural network and 10 epochs. Features and number feature planes: stone color, ones, liberties and capture size: 3,1, 8 and 8.

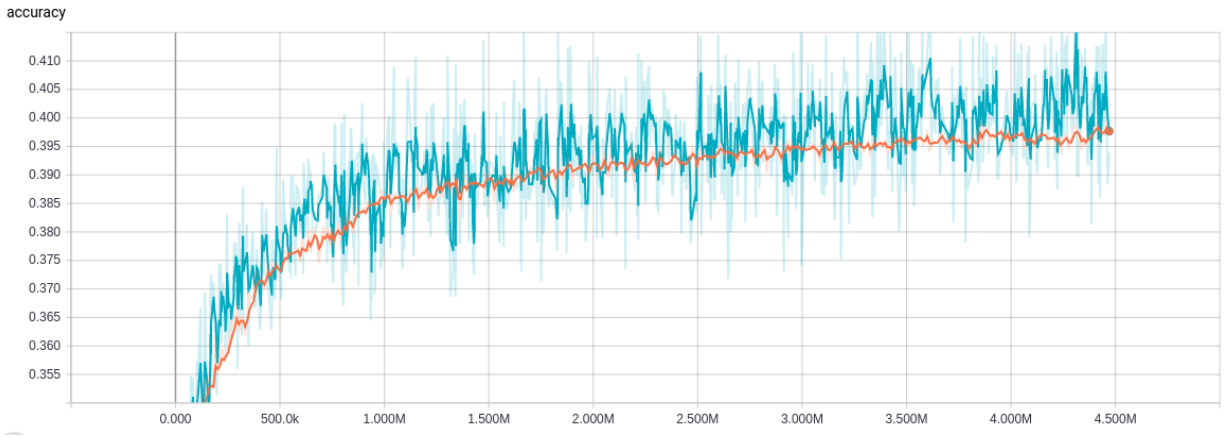

Figure 2.10 Performance of Shouzhuo with final test-set accuracy 0.398.

Parameters: 5 layers' convolutional neural network and 10 epochs. Features and number feature planes: stone color, ones, liberties, capture size and turns since: 3,1, 8, 8 and 8.

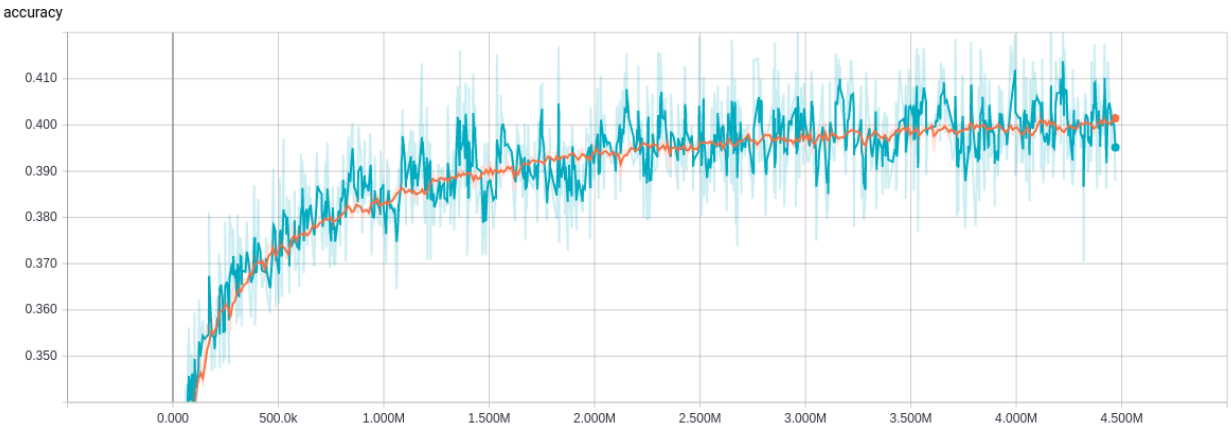

Figure 2.11 Performance of Shouzhuo with final test-set accuracy 0.401.

Parameters: 5 layers' convolutional neural network and 10 epochs. Features and number feature planes: stone color, ones, liberties, capture size, self-atari size, and turns since: 3,1, 8, 8, 8 and 8.

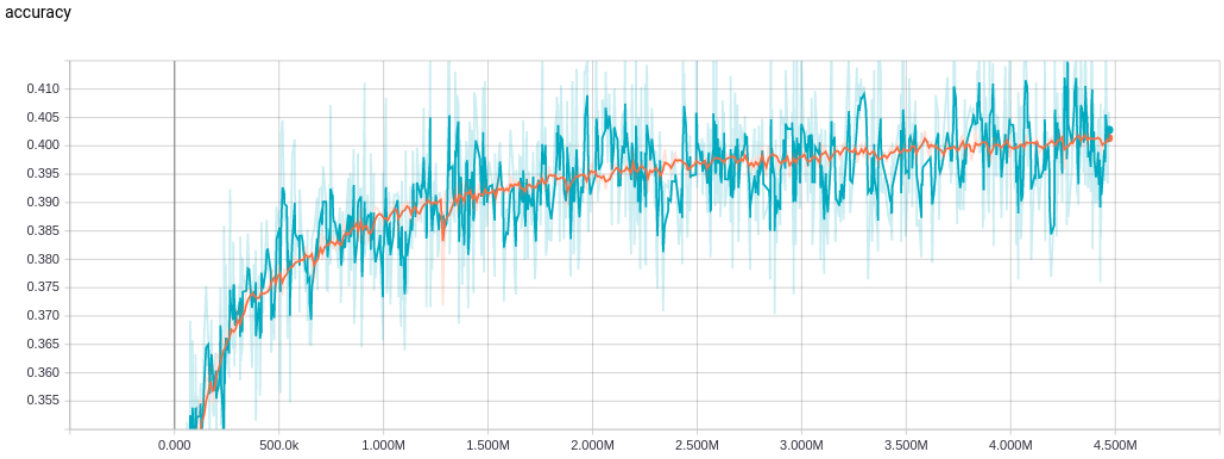

Figure 2.12 Performance of Shouzhuo with final test-set accuracy 0.402.

Parameters: 5 layers' convolutional neural network and 10 epochs. Features and number feature planes: stone color, ones, liberties, capture size, self-atari size, liberties after move and turns since: 3,1, 8, 8, 8, 8 and 8.

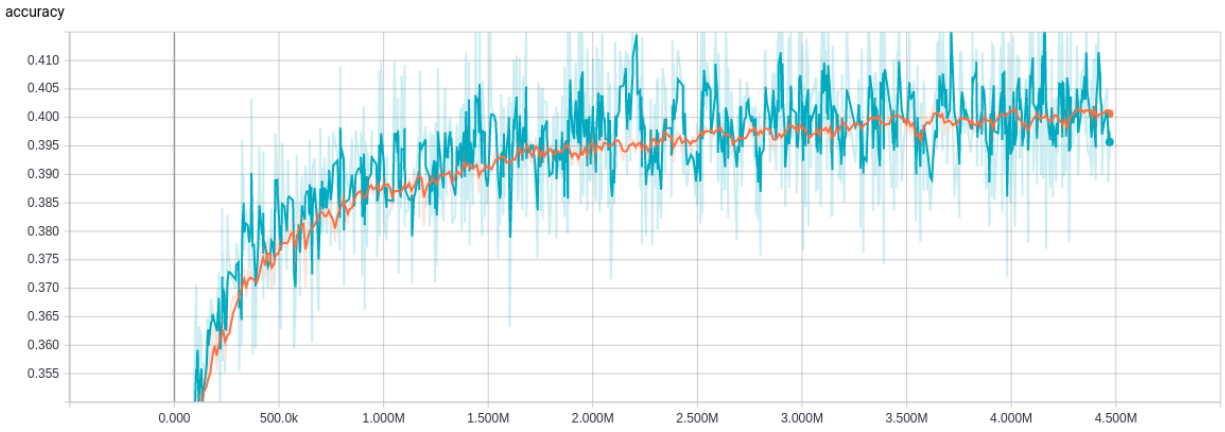

Figure 2.13 Performance of Shouzhuo with final test-set accuracy 0.401.

Parameters: 5 layers' convolutional neural network and 10 epochs. Features and number feature planes: stone color, ones, liberties, capture size, self-atari size, liberties after move, sensibility and turns since: 3,1, 8, 8, 8, 8, 1 and 8.

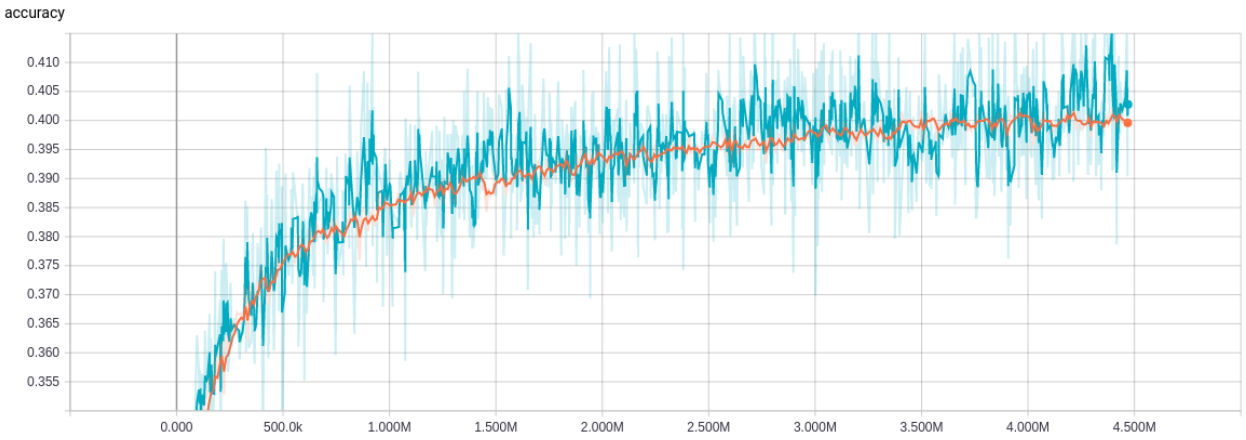

Figure 2.14 Performance of Shouzhuo with final test-set accuracy 0.401.

Parameters: 5 layers' convolutional neural network and 10 epochs. Features and number feature planes: stone color, ones, liberties, capture size, self-atari size, zeros and turns since: 3,1, 8, 8, 8, 1 and 8.

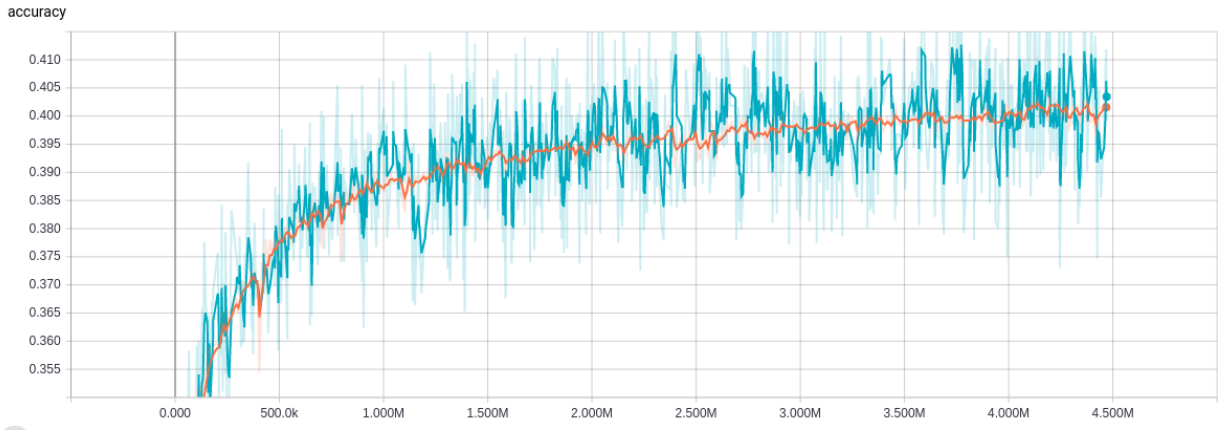

Figure 2.15 Performance of Shouzhuo with final test-set accuracy 0.402.

Parameters: 5 layers' convolutional neural network and 10 epochs. Features and number feature planes: stone color, ones, liberties, capture size, self-atari size, liberties after move, sensibility, zeros and turns since: 3,1, 8, 8, 8, 8, 1, 1 and 8.

2.4.2 ResNet in computer go

ResNet was developed in 2015. The basic model was shown in Figure 2.16. The best model is called full pre-activation model with structure: BN-Relu-Weight-BN-Relu-Weight to connect layers. It solves the vanishing gradients problem so that in my example I can use more than 13 layers, which AlphaGo used for its first version.

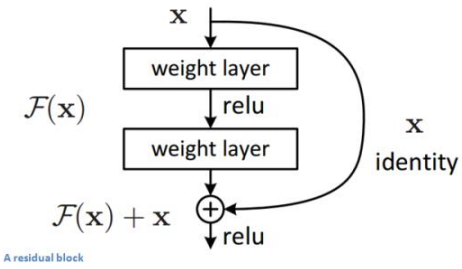

Figure 2.16 A residue block

2.4.3 Deep learning model training

With this architecture, I trained it on 1 GPU for 18 days. It is very interesting play with it when it is under training. Its accuracy reached 54% on the validation set.

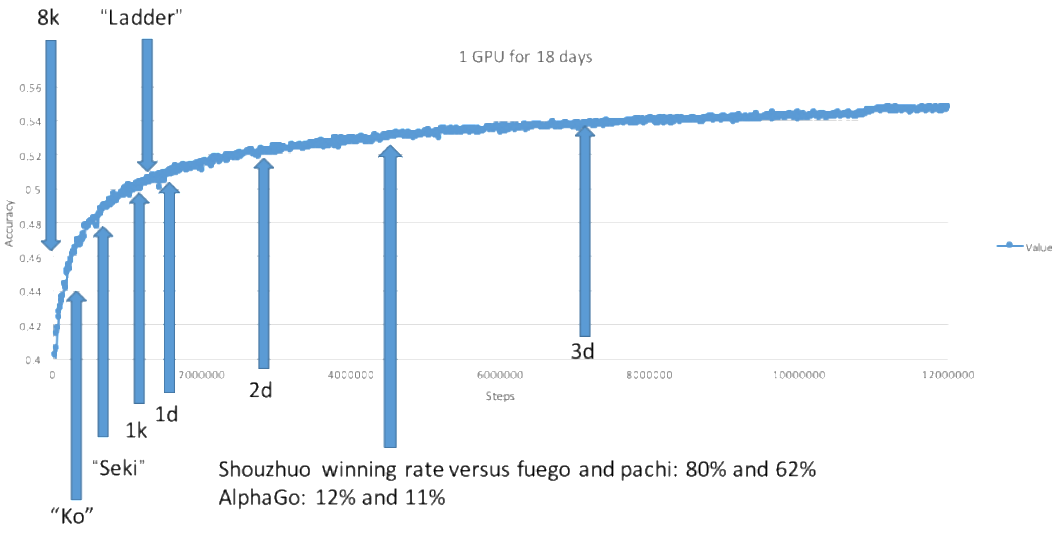

Figure 2.17 Performance of Shouzhuo

Within half a day, it knows ko. After about one day it began to know seki. After 1 days it reaches 1 k level and begin to know the idea of ladder. After 2 days it reaches 1d level. To reach 2d level it takes more time, about 5 days.

Beause AlphaGo supervised model also compared with fuego and pachi, I tried to use it to play with my model. As we can see from Figure 3.6, it does better than AlphaGo after 1-week training.

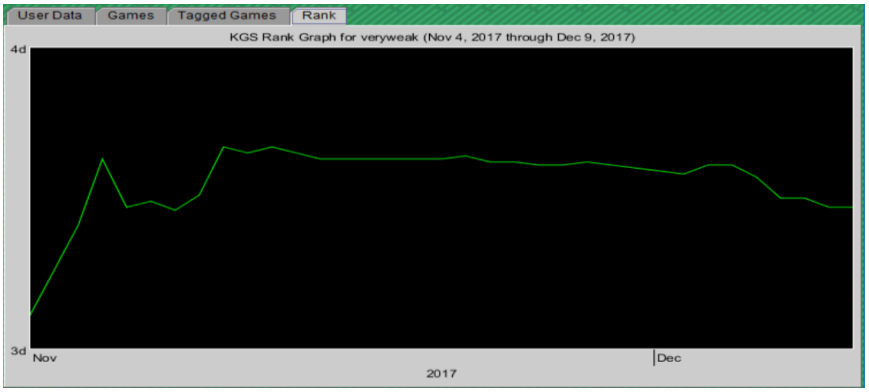

Figure 2.18 An online match between Shouzhuo and a 3D level player on KGS

Within 2 weeks it reaches 3d level on KG8S and it did not improve a lot after that. I tried to use it play 3D level players. It can win 50%. Figure 3.7 shows a play of its paly with a 3D level player on KGS.

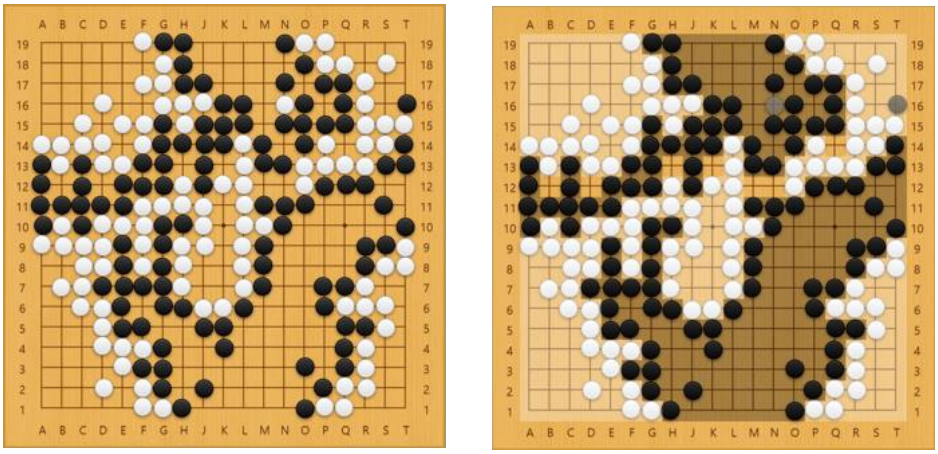

Figure 2.19 An online match between Shouzhuo and a 3D level player on KGS

2.4.4 Conclusion

Deep learning is a very powerful tool in terms of prediction. In computer-go, ResNet shows it ability to predict the next move with high accuracy. Thus, it can be very strong if combined with reinforcement learning for self-training.

REFERENCES

[1] D. E. V. Pires, T. L. Blundell, and D. B. Ascher, "pkCSM: Predicting small-molecule pharmacokinetic and toxicity properties using graph-based signatures," *J. Med. Chem.*, vol. 58, no. 9, pp. 4066–4072, 2015.

[2] F. Cheng, W. Li, Y. Zhou, J. Shen, Z. Wu, G. Liu, P. W. Lee, and Y. Tang, "AdmetSAR: A comprehensive source and free tool for assessment of chemical ADMET properties," *J. Chem. Inf. Model.*, vol. 52, no. 11, pp. 3099–3105, 2012.

[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[4] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2323, 1998.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Adv. Neural Inf. Process. Syst.*, pp. 1–9, 2012.

[6] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *International Conference on Learning Representations*, 2015.

[7] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 07–12–June, pp. 1–9, 2015.

[8] M. Lin, Q. Chen, and S. Yan, "Network In Network," in *International Conference on Learning Representations*, 2014, pp. 1–10.

[9] K. He, X. Zhang, S. Ren, and J. Sun, “Identity mappings in deep residual networks,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics),* vol. 9908 LNCS, pp. 630–645, 2016.

[10] S. Zagoruyko and N. Komodakis, “Wide Residual Networks,” in *Computing Research Repository*, 2015.

APPENDICES

Appendix A

Appendix A.1 Code for early stopping and AUC calculating

```
class CustomCallbacks(Callback):
    def __init__(self, filename="best_weight.hdf5", monitor='val_acc', pat
ience=10, test_input=None, test_label=None):
        super(Callback, self).__init__()
        self.filename = filename
        self.monitor = monitor
        self.patience = patience
        self.test_input=test_input
        self.test_label=test_label

        self.wait = 0

        if 'acc' in self.monitor or 'val_acc' in self.monitor:
                self.monitor_op = np.greater
                self.best = -np.Inf
        else:
                self.monitor_op = np.less
                self.best = np.Inf
        self.losses = []
        self.acc = []
        self.val_acc =[]
        self.val_losses=[]

    def on_epoch_end(self, epoch, logs={}):
        self.losses.append(logs.get('loss'))
        self.acc.append(logs.get('acc'))
        self.val_acc.append(logs.get('val_acc'))
        self.val_losses.append(logs.get('val_loss'))

        y_score=self.model.predict(X_test)
        y_score_get_value=[]
        for item in y_score:
            y_score_get_value.append(item[0])
        fpr, tpr, _ = roc_curve(y_test, y_score_get_value)
        val_auc=auc(fpr,tpr)

        print("acc: %0.3f" % logs.get('acc'),
              ", loss: %0.3f" % logs.get('loss'),
              ", val_acc: %0.3f" % logs.get('val_acc'),
              ", val_loss: %0.3f" % logs.get('val_loss'),
              ", val_auc: %0.3f" % val_auc)

        current = logs.get(self.monitor)
        if current is None:
            warnings.warn('Can save best model only with %s available, '
                                'skipping.' % (self.monitor), RuntimeWar
ning)
```

```
        else:
            if self.monitor_op(current, self.best):
                print('Epoch %05d: %s improved from %0.5f to %0.5f,'
                                  ' saving model to %s'
                                  % (epoch, self.monitor, self.best,
                                     current, self.filename))
                self.best = current
                self.wait = 0
                self.model.save(self.filename, overwrite=True)
            else:
                print('Epoch %05d: %s did not improve' %  (epoch, self.mon
itor))
                if self.wait >= self.patience:
                    self.model.stop_training = True
                self.wait += 1

    def on_train_end(self, logs=None):
        if self.wait >= self.patience:
            print('Warning: early stopping')
        self.wait=0
        if 'acc' in self.monitor or 'val_acc' in self.monitor:
                self.monitor_op = np.greater
                self.best = -np.Inf
        else:
                self.monitor_op = np.less
                self.best = np.In

my_callbacks=CustomCallbacks(filename="CYP1A2_conv1.hdf5", monitor='val_ac
c', patience=np.Inf, test_input=X_test, test_label=y_test)
```

Appendix A.2 Code for model training

```
inputs = Input(shape=(maxlen,))
embedding_1=Embedding(input_dim=max_features,output_dim=50, input_length=m
axlen)(inputs)
dropout_1=Dropout(0.5)(embedding_1)

conv1d_kernel_size=[7,3,5]
conv1d_filters=[128,128,128]
my_globalmaxpoolings=[]
for i in range(0,len(conv1d_kernel_size)):
    conv1d_1=Conv1D(filters=conv1d_filters[i],
                    kernel_size=conv1d_kernel_size[i],
                    padding='valid',
                    strides=1)(dropout_1)

    leakyrelu_1=LeakyReLU(0.3)(conv1d_1) #better than relu with 0.004 more
 accuracy...
    dropout_conv=Dropout(0.5)(leakyrelu_1)
    globalmaxpooling1d_1=GlobalMaxPooling1D()(leakyrelu_1)
    my_globalmaxpoolings.append(globalmaxpooling1d_1)

concatenate_1=Concatenate()(my_globalmaxpoolings)
```

```
#model.add(MaxPooling1D(pool_size=2))

dense_1=Dense(500)(concatenate_1)
dropout_2=Dropout(0.5)(dense_1)
leakyrelu_hidden=LeakyReLU(0.3)(dropout_2)

dense2=Dense(1)(leakyrelu_hidden)
prediction=Activation('sigmoid')(dense2)


model = Model(inputs=inputs, outputs=prediction)
adam=optimizers.Adam(lr=0.001, beta_1=0.99, beta_2=0.999, epsilon=1e-08, d
ecay=0.0)
model.compile(loss='binary_crossentropy',
              optimizer=adam,
              metrics=['accuracy'])
```