Move Evaluation in Go Using Deep Convolutional Neural Networks

Chris J. Maddison
University of Toronto
cmaddis@cs.toronto.edu

Aja Huang[1], Ilya Sutskever[2], David Silver[1]
Google DeepMind[1], Google Brain[2]
{ajahuang,ilyasu,davidsilver}@google.com

Abstract

The game of Go is more challenging than other board games, due to the difficulty of constructing a position or move evaluation function. In this paper we investigate whether deep convolutional networks can be used to directly represent and learn this knowledge. We train a large 12-layer convolutional neural network by supervised learning from a database of human professional games. The network correctly predicts the expert move in 55% of positions, equalling the accuracy of a 6 dan human player. When the trained convolutional network was used directly to play games of Go, without any search, it beat the traditional-search program *GnuGo* in 97% of games, and matched the performance of a state-of-the-art Monte-Carlo tree search that simulates two million positions per move.

1 Introduction

The most frequently cited reason for the difficulty of Go, compared to games such as Chess, Scrabble or Shogi, is the difficulty of constructing an evaluation function that can differentiate good moves from bad in a given position. The combination of an enormous state space of 10^{170} positions, combined with sharp tactics that lead to steep non-linearities in the optimal value function, has led many researchers to conclude that representing and learning such a function is impossible (Müller, 2002). In previous years, the most successful methods have sidestepped this problem altogether using Monte-Carlo search, which dynamically evaluates a position through random sequences of self-play. Such programs have led to strong amateur level performance, but a considerable gap still remains between top professional players and the strongest computer programs. The majority of recent progress has been due to increased quantity and quality of prior knowledge, which is used to bias the search towards more promising states in both the search tree and during rollouts (Coulom, 2007; Gelly & Silver, 2011; Enzenberger et al., 2010; Baudiš & Gailly, 2012; Huang et al., 2011), and it is widely believed that this knowledge is the major bottleneck towards further progress (Huang & Müller, 2013). However, this knowledge again is ultimately compiled into an evaluation function or distribution that expresses a preference over moves.

In this paper we address these fundamental questions of representation and learning of Go knowledge, by using a deep convolutional neural network (CNN). Although CNNs have previously been applied to the game of Go, with modest success (Schraudolph et al., 1994; Enzenberger, 1996; Sutskever & Nair, 2008), previous architectures have typically been limited to one hidden layer of relatively small size, and have not exploited recent advances in computational power. In this paper we use much deeper and larger CNNs of 12 hidden layers and several billion connections to represent and learn Go knowledge. We find that this increase in depth and size leads to a qualitative jump in performance, suggesting that contrary to previous beliefs, a strong move evaluation function for Go can indeed be represented and learnt by such architectures.

We focus on a supervised learning setup, in which the network is trained to predict expert human moves, using a large database of professional 19×19 Go games. The predictive accuracy of the

CNN on a held-out set of positions reaches 55%, which is a significant improvement over the 35% and 39% predictive accuracy reported for some of the strongest Go programs, and comparable to the performance of the 6 dan author on the same data set. Furthermore, when the CNN was used to play games by directly selecting the move recommended by the network output, without any search, it equalled the performance of state-of-the-art Monte-Carlo search programs, such as *Pachi* (Baudiš & Gailly, 2012), that are given 10,000 rollouts per move (i.e., programs that combine handcrafted or shallow prior knowledge with a search that simulates two million positions), and the first strong Monte-Carlo search program *MoGo* with 100,000 rollouts per move (Gelly & Silver, 2007). In addition, direct move selection using the CNN beat *GnuGo* (a traditional search program) in 97% of games.[1]

Finally, we demonstrate that the Go knowledge embodied by the CNN can be effectively combined with Monte-Carlo tree search, by using a *delayed prior knowledge* procedure. In this approach, the CNN is evaluated asynchronously on a GPU, and results are incorporated into the main search procedure once available. Using 100,000 rollouts per move, the overall search defeats the raw CNN in 87% of games.

2 PRIOR WORK

Convolutional neural networks have a long history in the game of Go. Schraudolph Schraudolph et al. (1994) trained a simple CNN (exploiting rotational, reflectional, and colour inversion symmetries) to predict final territory, by reinforcement learning from games of self-play. The resulting program beat a simplistic handcrafted program called *Wally*. *NeuroGo* (Enzenberger, 1996) used a more sophisticated architecture to predict final territory, eyes, and connectivity, again exploiting symmetries; and used a connectivity pathfinder to propagate information across weakly connected groups of stones. Enzenberger's program also used reinforcement learning from self-play. When combined with an alpha-beta search, *NeuroGo* equalled the performance of *GnuGo* on 9×9 Go, and reached around 13 kyu on 19×19 Go. Sutskever & Nair (2008) applied convolutional networks to supervised learning of expert moves, but using a small 1 hidden layer CNN; this matched the state-of-the-art prediction performance, achieving 34.6% accuracy, but this was not sufficient to play Go at any reasonable level.

The most successful current programs in Go are based on Monte-Carlo tree search (Kocsis & Szepesvári, 2006). The basic algorithm was augmented in *MoGo* to use prior knowledge to bootstrap value estimates in the search tree (Gelly & Silver, 2007); and to use abstractions over subtrees to accelerate the search (Gelly & Silver, 2011). The strongest current programs such as *CrazyStone* apply supervised learning to construct a move selection policy; this is then used to bias the exploration during search; a faster policy is also learned that selects moves during rollouts (Coulom, 2007). CrazyStone achieved a 35% move prediction accuracy by extracting a large database of common patterns from expert games, and combining them into a large linear softmax.

Recent work in image recognition has demonstrated considerable advantages of deep convolutional networks over alternative architectures. Krizhevsky et al. (2012) were the first to achieve a very large performance gain with large and deep convolutional neural networks over traditional computer vision systems. Improved convolutional neural network architectures (primarily in the form of deeper networks) (Simonyan & Zisserman, 2014) provided another substantial improvement, culminating with Szegedy et al. (2014), who reduced the error rate of Krizhevsky et al. (2012) from 15.3% top-5 error to 7.0%. The power and generality of large and deep convolutional neural networks suggests that they may do well on other "visual" domains, such as computer Go.

3 DATA

The dataset used in this work comes from the KGS Go Server. It consists of sequences of board positions s_t for complete games played between humans of varying rank. Board state information includes the position of all stones on the 19x19 board and the sequence allows one to determine the

---

[1] Since we performed this research, we have learned that Clark & Storkey (2014) independently adopted a similar approach using a smaller 8-layer CNN to achieve 44% move prediction accuracy; and defeated GnuGo in 86% of games.

Feature	Planes	Description
Black / white / empty	3	Stone colour
Liberties	4	Number of liberties (empty adjacent points)
Liberties after move	6	Number of liberties after this move is played
Legality	1	Whether point is legal for current player
Turns since	5	How many turns since a move was played
Capture size	7	How many opponent stones would be captured
Ladder move	1	Whether a move at this point is a successful ladder capture
KGS rank	9	Rank of current player

Table 1: Features used as inputs to the CNN.

sequence of moves; a move a_t is encoded as a 1 of 361 indicator for each position on the 19x19 board. We collected 29.4 million board-state next-move pairs (s_t, a_t) corresponding to 160,000 games.

Each position s_t was preprocessed into a set of 19×19 feature planes $\phi(s_t)$, that serve as input to the neural network. The features that we use come directly from the raw representation of the game rules (stones, liberties, captures, legality, turns since). In addition, we have one simple tactical feature representing a basic common pattern in Go known as ladders; in practice this adds a small performance benefit, but the results that we report would be qualitatively similar even without these features. Many of the features are split into multiple planes of binary values, for example in the case of liberties there are separate binary features representing whether each intersection has 1 liberty, 2 liberties, 3 liberties, $>= 4$ liberties. The feature planes are listed in Table 1.[2]

Finally, we used the following minor innovation. Our dataset consists of games from players of different strengths. Specifically, the KGS data contains more games by lower dan players, and fewer games by higher dan players. As a result, a naive approach to training on the KGS data will result in a network that primarily imitates weaker players. Alternatively, training only on games by stronger players would result in a massive reduction of training data.

To mitigate this, we provided the network with an additional global inputs indicating the player's rank. Specifically we add 9 feature planes each indicating a specific rank. This is like a 1 of 9 encoding that represents the strength of the current player. That is, if the network is learning to predict a move made by a d dan player, the dth rank feature plane is filled with 1s and the remaining 8 planes are filled with 0s. This has the effect of providing a dynamic bias to the network that depends on rank.

Because every Go game is symmetric under reflections and rotations, we augmented the dataset by sampling uniformly from one of the 8 symmetric boards as we filled minibatches in gradient descent. The dataset was split into a training set of 27.4 million board-state next-move pairs and a test set of 2 million. This split was done before shuffling, so this corresponds to a test set with distinct games.

4 ARCHITECTURE & TRAINING

In this section we describe the precise network architecture and the details of the training procedure.

We used a deep convolutional neural network with 12 weight matrices for each of 12 layers and rectified linear non-linearities. The first hidden layer's filters were of size 5×5 and the remainder were of size 3×3, with a stride of 1. Every layer operated on a 19×19 input space, with no pooling; outputs were zero-padded back up to 19×19. The number of filters in each layer ranged from 64 to 192. In addition to convolutions, we also used *position-dependent biases* (following Sutskever & Nair (2008)). Our best model has 2.3 million parameters, 630 million connections, and 550,000 hidden units.

The output layer of the CNN was also convolutional with position dependent biases, but with only two filters. Each produced a 19×19 plane, corresponding to inputs to two softmax distributions of size 361. The first softmax is the distribution over the next move if it is the black player's turn, and

---

[2]Due to the computational cost of running extensive experiments, it is possible that some of these features are unnecessary or redundant.

the second softmax is the distribution over the next move if it is the white player's move. Although both players may often prefer the same move, in general the optimal policy may select different moves for each player.

We also experimented with weight symmetries Schraudolph et al. (1994). Given that the board is symmetric, it makes sense to force the filters and biases to be rotationally and reflectionally symmetric, by aggregating weight updates over the 8-fold symmetry group between connections. This type of symmetry is stronger than the symmetric data augmentation described above, since it enforces local symmetry of all filters at all locations on the board, not just global symmetry of the entire board.

For training the network, we used asynchronous stochastic gradient descent (Dean et al., 2012) with 50 replicas each on its own GPU. All parameters were initialized randomly from a uniform[-0.05, 0.05]. Each replica was trained for 25 epochs with a batchsize of 128, a fixed learning rate of 0.128 normalized by batchsize, and no momentum. The network was then fine-tuned on a single GPU with vanilla SGD for 3 epochs with an annealed learning rate, beginning at half the learning rate for the asynchronous setting and halved again every epoch. After augmenting the dataset with random symmetries overfitting was very minor — our 10 layer network overfit by under 1% achieving 55% on the training set and 54.5% on the test set. Even at the end of training errors on the test set did not increase. This suggests that we are currently operating in an underfitting regime suggesting that further improvement is possible. All reported accuracies are on a held out test set.

5 RESULTS

5.1 INVESTIGATION OF WEIGHT SYMMETRIES

We evaluated the effect of weight symmetries on a smaller CNN with 3 and 6 layers respectively. These networks were trained on a reduced feature set, excluding rank, liberties after move, capture size, ladder move, and only including a history of one move. The results are given in the table below:

model	% Accuracy
3 layer, 64 filters	43.3
3 layer, 64 filters, symmetric	44.3
6 layer, 192 filters	49.6
6 layer, 192 filters, symmetric	49.4

These results suggest that, perhaps surprisingly, weight symmetries have a strong effect on move prediction for small and shallow networks, but the effect appeared to disappear completely in larger and deeper networks.

5.2 ACCURACY AND PLAYING STRENGTH

To understand how the performance depends on network depth, we trained several networks of different depths. Each CNN used the same architecture as described above, except that the number of 3×3 layers was restricted to 3, 6, 10 and 12 respectively. We measured the prediction accuracy on the test set, and also the playing strength of the CNN when it was used to directly select moves. This was achieved by inputting the current position into the network, and selecting the action with maximum probability in the softmax output for the current player. Unless otherwise specified the *KGS rank* feature was set to its maximum setting.

Performance was evaluated against the benchmark program *GnuGo* 3.8, running at its highest level 10. Comparisons are given with reported values for the 3 dan Monte-Carlo search program *Aya*[3]; simultaneously published results on a somewhat shallower CNN Clark & Storkey (2014)[4]; and also with the prediction accuracy of a 6 dan human (the second author) on randomly sampled positions

---

[3]http://computer-go.org/pipermail/computer-go/2014-December/007018.html

[4]It should be noted that Clark & Storkey (2014) did not use the highly-predictive *turns since* feature, because they believed that it would hurt the network's play. This is an interesting hypothesis, which this work does not address.
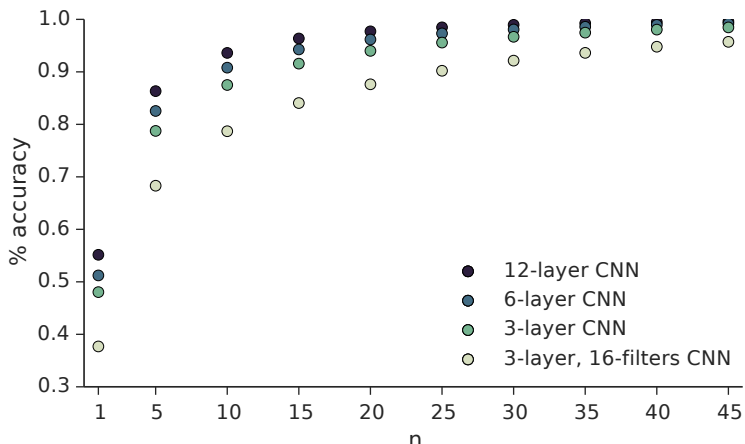
Figure 1: Probability that the expert's move is within the top-n predictions of the network. The 10 layer CNN was omitted for clarity, but it's performance is only slightly worse than 12 layer. Note y-axis begins at 0.30.

from the test set. All games were scored using Chinese rules, refereed by *GnuGo*; duplicate games were excluded from results.

It is apparent from the results that larger and deeper networks have qualitatively better performance than shallow networks, reaching 97% winning rate against GnuGo for a large 12-layer network compared to 3.4% for a small 3-layer network. Furthermore, the accuracy on the supervised learning task is clearly strongly correlated with playing performance, demonstrating that the knowledge learnt by the network generalises effectively to the real task of evaluating moves.

Depth	Size	% Accuracy	% Wins vs. *GnuGo*	stderr
3 layer	16 filters	37.5	3.4	$\pm$ 1.1
3 layer	128 filters	48.0	61.8	$\pm$ 2.6
6 layer	128 filters	51.2	84.4	$\pm$ 1.9
10 layer	128 filters	54.5	94.7	$\pm$ 1.2
12 layer	128 filters	55.2	97.2	$\pm$ 0.9
8 layer (Clark & Storkey, 2014)[4]	$\leq$ 64 filters	44.4	86	$\pm$ 2.5
Aya 2014		38.8	6	$\pm$ 1.0
Human 6 dan		52 $\pm$5.8	100	

It is also valuable to know that the correct move is within the network's n most confident predictions. If n can be kept small, then this knowledge can be used to reduce the program's effective search space. We find that the top-n performance of our network is quite strong; in particular, the network is able to predict the correct expert move 94% of the time when $n = 10$.

Next, we compared how the CNN performed when asked to imitate players of different strengths. We used the same CNN, trained on KGS data of all ranks, and asked it to select moves as if it was playing according to a specified rank. The opponent was a fixed 10 layer, 128 filter CNN trained without the *KGS rank* feature. The results clearly show that the network plays significantly better when it is asked to imitate a stronger player.

KGS rank	% wins vs. *10-layer CNN*	stderr
1 dan	49.2	$\pm$ 3.6
5 dan	60.1	$\pm$ 1.6
9 dan	67.9	$\pm$ 5.0

Finally, we evaluated the overall strength of the 12-layer CNN when used for move selection, by playing against several publicly available benchmark programs. All programs were played at the strongest available settings, and a fixed number of rollouts per move, as specified in the table.

Opponent	Rollouts per move	Games won by CNN	stderr
GnuGo		97.2	± 0.9
MoGo	100,000	45.9	± 4.5
Pachi	100,000	11.0	± 2.1
Fuego	100,000	12.5	± 5.8
Pachi	10,000	47.4	± 3.7
Fuego	10,000	23.3	± 7.8

The neural network is considerably stronger than the traditional search-based program GnuGo, and its performance is on a par with *MoGo* with 100,000 rollouts per move (Gelly & Silver, 2007), and *Pachi* running a somewhat reduced search of 10,000 rollouts per move (a search that visits approximately 2 million positions). It wins more than 10% of games against *Fuego* (latest svn revision 1966) (Enzenberger et al., 2010) and *Pachi* 10.99 playing at a strong level (using 100,000 rollouts per move over 16 threads).[5]

6 SEARCH

The overarching goal of this work is to build a strong Go playing program. To this end, we attempted to integrate our move prediction network with Monte Carlo Tree Search (MCTS).

Combining MCTS with a large deep neural network is far from trivial, since the CNN is slower than the natural speed of the search, and it is not feasible to evaluate every node with the neural network. The 12-layer network takes 0.15s to evaluate a minibatch of size 128.[6]

We address this problem by using *asynchronous* node evaluation. In asynchronous node evaluation, MCTS builds its search tree and tracks the new nodes that are added into the search tree. When the number of new nodes equals the minibatch size, all these new positions are submitted to the CNN for evaluation on a GPU. The GPU computes the move recommendations, while the search continues in parallel. Once the GPU computation is complete, the prior knowledge in the new nodes is updated to contain move evaluations from the CNN. The network evaluates the nodes in a FIFO order, in order to maximally influence the search tree. By using a single machine with Intel® Xeon® CPU E5-2643 v2 @ 3.50GHz and GeForce GTX Titan Black GPU, we are able to maintain a MCTS search at approximately 47,000 rollouts per second, without dropping CNN evaluations. However, it should be noted that the performance of asynchronous node evaluation is significantly less than a fully synchronous and serial implementation, since new information from the search is only utilised after a significant lag (around 0.15s in our case), due to the GPU computation.

In addition, the MCTS engine utilised standard heuristics for computer Go: RAVE (Gelly & Silver, 2011), a UCT exploration strategy similar to Chaslot et al. (2008), and very simple rollouts based solely on 3×3 patterns (Huang et al., 2011).

We measured the performance of the search-based program by playing games between the 12-layer CNN with MCTS, and a baseline 12-layer CNN without any search. Using 100,000 rollouts per move, the search-based program beats the baseline CNN in 87% of games.

Rollouts per move	% wins against baseline	stderr
100,000	86.7	± 3.5
10,000	67.6	± 2.6

7 DISCUSSION

In this work, we showed that large deep convolutional neural networks can predict the next move made by Go experts with an accuracy that exceeds previous methods by a large margin, approximately matching human performance. Furthermore, this predictive accuracy translates into much

---

[5]The 8-layer network of Clark & Storkey (2014) won 12% of games against the older version Fuego 1.1 at 10 seconds per move on 2×1.6 GHz cores. We tested our 12-layer CNN against Fuego 1.1 at 5 and 10 seconds per move on 2×3.1GHz cores, winning 56% and 33% respectively.

[6]Reducing the minibatch size does not significantly speed up end-to-end computation time in our GPU implementation.
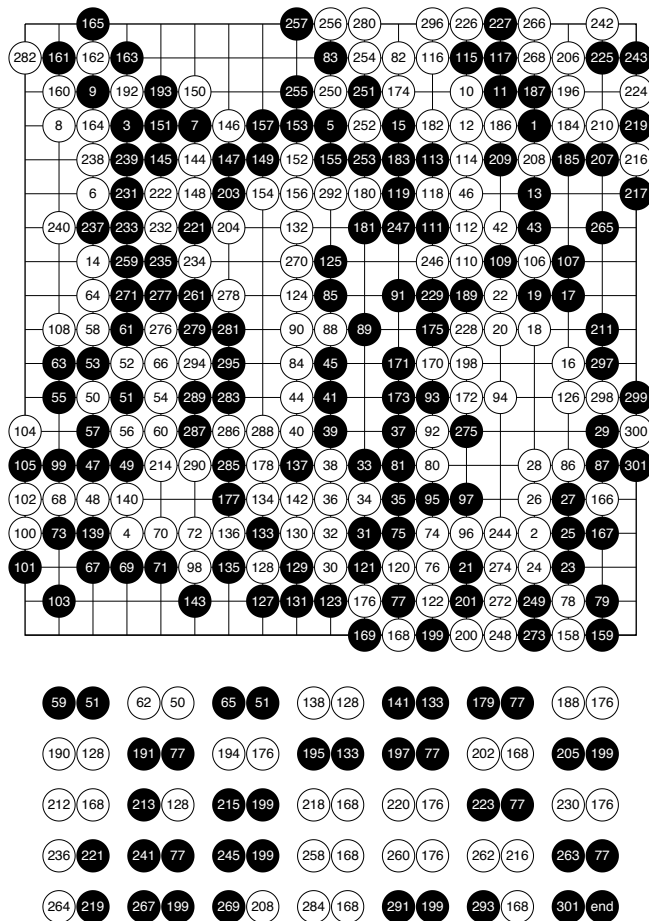
Figure 2: A game played between the 12-layer CNN (without any search) and Fuego (using 100k roll-outs/move). The CNN plays white.

stronger move evaluation and playing strength than has previously been possible. Without any search, the network is able to outperform traditional search based programs such as GnuGo, and compete with state-of-the-art MCTS programs such as *Pachi* and *Fuego*.

In Figure 2 we present a sample game played by the 12-layer CNN (with no search) versus *Fuego* (searching 100K rollouts per move) which was won by the neural network player. It is clear that the neural network has implicitly understood many sophisticated aspects of Go, including good shape (patterns that maximise long term effectiveness of stones), Fuseki (opening sequences), Joseki (corner patterns), Tesuji (tactical patterns), Ko fights (intricate tactical battles involving repeated recapture of the same stones), territory (ownership of points), and influence (long-term potential for territory). It is remarkable that a single, unified, straightforward architecture can master these elements of the game to such a degree, and without any explicit lookahead.

On the other hand, we note that the network still has weaknesses: notably it sometimes fails to under-stand the global picture, behaving as if the life and death status of large groups has been incorrectly assessed. Interestingly, it is precisely these global aspects of the game for which Monte-Carlo search excels, suggesting that these two techniques may be largely complementary. We have provided a preliminary proof-of-concept that MCTS and deep neural networks may be combined effectively. It appears that we now have two core elements that scale effectively with increased computational re-source: scalable planning, using Monte-Carlo search; and scalable evaluation functions, using deep neural networks. In the future, as parallel computation units such as GPUs continue to increase in performance, we believe that this trajectory of research will lead to considerably stronger programs than are currently possible.

7

REFERENCES

Baudiš, Petr and Gailly, Jean-loup. Pachi: State of the art open source go program. In *Advances in Computer Games*, pp. 24–38. Springer, 2012.

Chaslot, Guillaume M. J-B., Winands, Mark H. M., van den Herik, H. Jaap, Uiterwijk, Jos W. H. M., and Bouzy, Bruno. Progressive strategies for Monte-Carlo tree search. *New Mathematics and Natural Computation*, 4:343–357, 2008. doi: 10.1142/S1793005708001094.

Clark, Christopher and Storkey, Amos. Teaching deep convolutional neural networks to play Go. *arXiv preprint arXiv:1412.3409*, 2014.

Coulom, Rémi. Efficient selectivity and backup operators in Monte-Carlo tree search. In *Computers and games*, pp. 72–83. Springer, 2007.

Dean, Jeffrey, Corrado, Greg, Monga, Rajat, Chen, Kai, Devin, Matthieu, Mao, Mark, aurelio Ranzato, Marc', Senior, Andrew, Tucker, Paul, Yang, Ke, Le, Quoc V., and Ng, Andrew Y. Large scale distributed deep networks. In Pereira, F., Burges, C.J.C., Bottou, L., and Weinberger, K.Q. (eds.), *Advances in Neural Information Processing Systems 25*, pp. 1223–1231. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4687-large-scale-distributed-deep-networks.pdf.

Enzenberger, Markus. The integration of a priori knowledge into a Go playing neural network. *URL: http://www. markus-enzenberger. de/neurogo. html*, 1996.

Enzenberger, Markus, Müller, Martin, Arneson, Broderick, and Segal, R. Fuego - an open-source framework for board games and Go engine based on monte carlo tree search. *IEEE Trans. Comput. Intellig. and AI in Games*, 2(4):259–270, 2010.

Gelly, S. and Silver, D. Combining online and offline learning in UCT. In *17th International Conference on Machine Learning*, pp. 273–280, 2007.

Gelly, S. and Silver, D. Monte-Carlo tree search and rapid action value estimation in computer Go. *Artificial Intelligence*, 175:1856–1875, 2011.

Huang, Shih-Chieh and Müller, Martin. Investigating the limits of Monte-Carlo tree search methods in computer Go. In *Computers and Games - 8th International Conference, CG 2013, Yokohama, Japan, August 13-15, 2013, Revised Selected Papers*, pp. 39–48, 2013.

Huang, Shih-Chieh, Coulom, Rémi, and Lin, Shun-Shii. Monte-Carlo simulation balancing in practice. In *Proceedings of the 7th International Conference on Computers and Games*, pp. 81–92. Springer-Verlag, 2011.

Kocsis, Levente and Szepesvári, Csaba. Bandit based Monte-Carlo planning. In *Machine Learning: ECML 2006*, pp. 282–293. Springer, 2006.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Müller, Martin. Computer Go. *Artif. Intell.*, 134(1-2):145–179, 2002.

Schraudolph, Nicol N, Dayan, Peter, and Sejnowski, Terrence J. Temporal difference learning of position evaluation in the game of Go. *Advances in Neural Information Processing Systems*, pp. 817–817, 1994.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Sutskever, Ilya and Nair, Vinod. Mimicking Go experts with convolutional neural networks. In *Artificial Neural Networks-ICANN 2008*, pp. 101–110. Springer, 2008.

Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014.