Vision Transformers for Computer Go

Amani Sagri[1], Tristan Cazenave[1], Jérôme Arjonilla[1], and Abdallah Saffidine[2]

[1] LAMSADE, Université Paris Dauphine - PSL, CNRS, Paris, France
[2] The University of New South Wales, Sydney, Australia

Abstract. Motivated by transformers' success in diverse fields like language understanding and image analysis, our investigation explores their potential in the game of Go. Specifically, we focus on analyzing Transformers in Vision. Through a comprehensive examination of factors like prediction accuracy, win rates, memory, speed, size, and learning rate, we underscore the significant impact transformers can make in the game of Go. Notably, our findings reveal that transformers outperform the previous state-of-the-art models, demonstrating superior performance metrics. This comparative study was conducted against conventional Residual Networks.

Keywords: Computer Go · Transformer · Games

1 Introduction

Due to a huge game tree complexity, the game of Go has been an important source of work in the perfect information setting. In 2007, search algorithms have been able to increase drastically the performance of computer Go programs [11, 12, 20, 16]. In 2016, a groundbreaking achievement occurred when AlphaGo became the first program to defeat a skilled professional Go player [26]. Currently, the level of play of such algorithms is far superior to those of any human player [26–28].

Over the years, various significant advances have been made to improve performance in the game of Go [3, 34, 32, 31, 33]. Many of these innovations find their roots in other domains, notably in computer vision, where the recognition and interpretation of the Go board's image serve as fundamental inputs. Algorithms such as ResNet [18, 3] and MobileNet [19, 7, 5] have demonstrated exceptional performance by harnessing groundbreaking developments in computer vision. However, it is worth noting that one remarkable advancement in the realm of computer vision remains relatively untapped for Computer Go: *transformers* [30].

Transformers represent a groundbreaking leap in deep learning, reshaping how various tasks in natural language processing (NLP), computer vision, and beyond are approached. Initially developed for NLP tasks, transformers introduce a departure from conventional sequential methods by employing self-attention mechanisms. These mechanisms simultaneously capture intricate interdependencies among all elements in a sequence. This ability to understand

nuanced relationships over long distances, without relying on recurrent or convolutional structures, has propelled transformers to the forefront of AI research. Notably, transformers have not only advanced language understanding, exemplified by models like BERT [13], but have also expanded their utility to image analysis, as seen in Vision Transformers (ViTs) [15] and other transformer-based models. EfficientFormer [21], a transformer-based model, achieves high performance and matches MobileNet's speed on mobile devices, proving that well-designed transformers can deliver low latency in computer vision tasks.

In this paper, we propose to analyze the impact of using Transformer methods in the game of Go. To do this, we use the EfficientFormer architecture. Our study analyses were done in comparison with other state-of-the-art vision architectures in Go such as Residual Networks on a wide range of criteria including prediction accuracy, win rates, memory, speed, architecture size, and even learning rate. We tune the learning rate and the size of the network for each network and we find that EfficientFormer improves better than Residual Network with longer training times. Both the policy accuracy, the Mean Squared Error (MSE), and the Mean Absolute Error (MAE) are better with longer training time. Other important properties of the tested networks are their latency and their memory use. To take into account the latency in the performance of the networks, we make them play using the same Monte Carlo Tree Search search time at every move and record their winning rates. We observe that EfficientFormer of size 'l9' with a learning rate of 0.0005 with 1,000 epochs of 100,000 states and a batch of 32 is better than Residual Networks on CPU and on GPU with the same number of epochs.

In Section 2, we present Computer Go, while Section 3 introduces the various algorithms and network architectures employed in this paper. Our results are detailed in Section 4, and the final section provides a comprehensive summary of our work along with insights into future avenues.

2 Computer Go

The game of Go is a turn-taking strategic board game of perfect information, played by two players. One player adds black stones to a vacant intersection of the board and the opponent adds white stones. After being placed, a player's stones cannot move. A group of contiguous stones is removed if and only if the opponent surrounds the group on all orthogonally adjacent points. The players aim at capturing the most territory and the game ends when no player wishes to move any further. There exist multiple rules for scoring. We have used the Chinese rule in our experiments: the winner of the game is defined by the number of stones that a player has on the board, plus the number of empty intersections surrounded by that player's stones and komi (bonus added to the second player as compensation for playing second).

Even though the rules are relatively simple, the game of Go is known as an extremely complex one in comparison to other board games such as Chess. On the standard board of size 19×19, the number of legal positions has been

estimated to be 2.1×10^{170}. Algorithms based on Monte Carlo Tree Search (MCTS) [1] have been achieving excellent performance in the game of Go for many years. Combining deep reinforcement learning and MCTS as introduced in the *AlphaGo* series programs [26, 28, 27] has been widely applied. The neural network takes an image of the board as input and produces two outputs: a probability distribution over moves (policy head) and a scalar of score prediction (value head) (see Fig. 1).

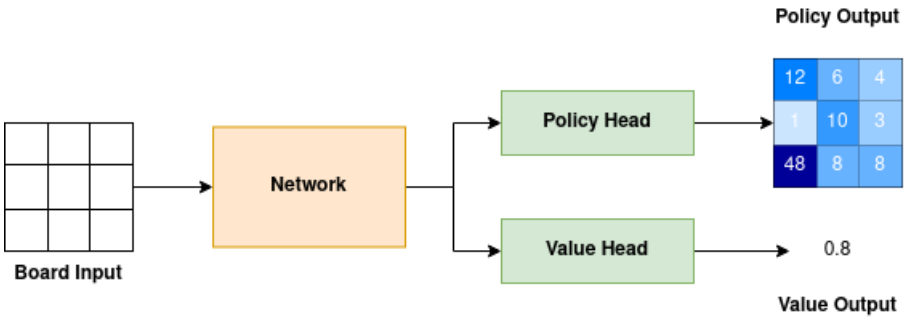


Fig. 1: AlphaZero network architecture

3 Network Architectures

3.1 Residual Network

Residual Networks are the standard networks for games [3, 28]. They are used in combination with MCTS to evaluate the leaves of the search tree and to give a prior on the possible moves. To speed up the computation of the evaluation and of the prior the networks are usually run on a batch of states [6].

The employed residual layer in image classification integrates the input of the layer with its output, incorporating two convolutional layers before the addition. ReLU layers are applied following the first convolutional layer and after the addition. Figure 2 illustrates the structure of the residual layer. We will experiment with this kind of residual layer for our Go networks.

3.2 Transformer

Transformers are advanced neural network architectures that leverage the concept of self-attention to process and understand complex sequences of data, such

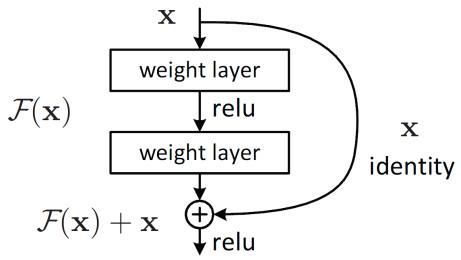


Fig. 2: The residual block.

as language. Self-attention allows a transformer model to analyze different elements within a sequence and determine their relative importance in relation to one another. By calculating attention scores based on the similarity of these elements, the model can dynamically weigh their significance and understand how they interrelate. Transformers employ multiple self-attention mechanisms (multihead self-attention) operating in parallel, enabling them to capture intricate patterns, dependencies, and contextual nuances across the entire input sequence.

Transformer was originally proposed as a sequence-to-sequence model [29] for machine translation. Later works show that Transformer-based pre-trained models (PTMs) [23] can achieve state-of-the-art performances on various tasks. In addition to language-related applications, Transformer has also been adopted in computer vision [22, 2, 15], audio processing [14, 17, 10] and even other disciplines, such as chemistry [25] and life sciences [24]. In natural Language Processing, the mechanism of attention of the Transformers tried to capture the relationships between different words of the text to be analyzed, in Computer Vision the Vision Transformers try instead to capture the relationships between different portions of an image.

3.3 Efficient Former

The EfficientFormer model is a big step forward in making transformer architectures work better for tasks that need real-time results, especially on devices with not much computing power. Adding a dimension-consistent plan allows the model to easily switch between different ways of organizing its parts, like in 4D and 3D setups. This way of thinking helps the EfficientFormer model to make the time it takes for predictions much shorter. By focusing on making predictions happen fast, a set of EfficientFormer models emerges, each achieving a careful equilibrium between performance and latency. This change in approach reaches its peak with models like EfficientFormer-l1, which impressively demonstrates outstanding top-1 accuracy on benchmarks like ImageNet-1K. At the same time, it manages to keep inference latency remarkably low on mobile devices, aligning closely with the efficiency of optimized versions of MobileNet.

In Figure 3, we illustrate the network architecture of the EfficientFormer. The network begins its operations with a convolution stem. This part usually

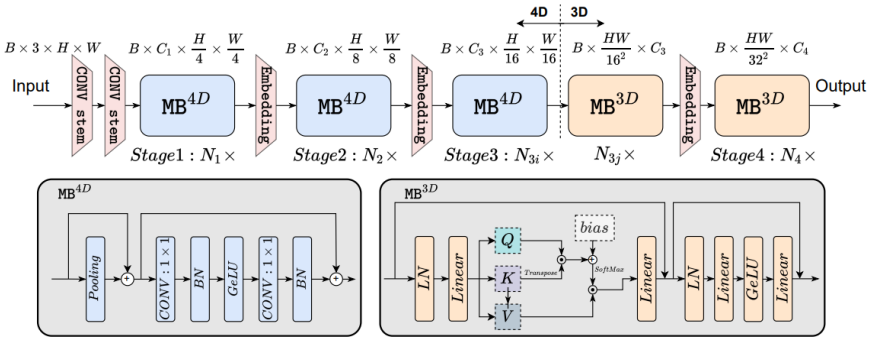


Fig. 3: Overview of EfficientFormer architecture [21].

carries out a set of actions to pull out basic features from the input data. These actions typically involve using convolutional layers, pooling layers, and normalization layers. Following that, there is a set of MetaBlocks (MB). There are two types of MetaBlocks: MB^{4D} and MB^{3D}. The layers that make up each block are illustrated in Figure 3. In between these blocks, we have the embedding layers. These layers break down the input into patches of a fixed size and assign each patch to a high-dimensional vector representation. These patch representations are then sent into the transformer blocks (within MB^{3D}) for further processing.

EfficientFormer is available in various sizes, denoted as l1, l3, l7, and l9. Each size is linked to a tuple of information where the first information is the width and the second information is the depth. The width is a list designing different dimensionalities (number of channels) of the feature vectors processed by different layers and blocks within the neural network. The width represents the number of blocks in different levels of the EfficientFormer architecture. The sizes tested throughout the paper are the following:

- 'l1' : ([48, 96],[3, 4])
- 'l3' : ([64, 128],[4, 6])
- 'l7' : ([96, 192],[6, 8])
- 'l9'; ([128,256],[8,10])

In order to clarify the notations used for transformers, we are going to explain in detail the parameters of the first example 'l1'. The entire structure comprises 7 MetaBlocks (MBs): 3 MBs^{4D} and 4 MBs^{3D}. The count of these blocks is determined by examining the second list [3, 4]. Let's denote the set of the 3 MBs^{4D} as X and the set of the 4 MBs^{3D} as Y. The number 48 from the first list represents the number of channels in all layers of X. while 96 is the number of channels in all layers of Y. Between the set X and the set Y, we have an embedding layer. This embedding layer allows the transition of the number of channels from 48 in X to 96 in Y.

3.4 Adaptation for the game of Go

In this paper, we took the same work on EfficientFormer model of Li *et al.* [21] and adapt the transformer mechanism to the Go game prediction. This necessitated modifying the final layers, which were originally designed for tasks like classification or segmentation, and instead, replacing them with layers tailored for policy and value (*i.e.*, the probability of winning the game) prediction.

This modification transformed the tasks into a dual output setting, combining multiclass classification and regression functionalities. The value head uses Global Average Pooling followed by two Dense layers [4, 8]. The policy head uses a 1x1 convolution to a single plane that defines the convolutional policy [8].

Another significant adjustment involved the downsampling and embedding layers commonly used in image classification tasks to detect features by reducing the image size before feeding it into the transformer. However, in the context of Go, the input board's dimensions were fixed at 19×19, and it was imperative to preserve this size throughout the training process to avoid losing critical information. Therefore, to retain the richness of the board data, the height and width of the board were maintained during training, ensuring that no valuable details were lost in the process. This tailored architectural approach played a pivotal role in optimizing the models for Go game prediction.

4 Experimental Results

4.1 Dataset

The data used for training comes from the Katago Go program self played games [31]. KataGo is one of the strongest open-source Go bots available online. There are 1,000,000 different games in total in the training set. Input data consists of 31 planes with a size of 19x19 each, encompassing information such as the color to play, ladders, the current state on two planes, and two previous states on four planes. The output targets are the policy (a vector of size 361 with 1.0 for the move played, 0.0 for the other moves), and the value (close to 1.0 if White wins, close to 0.0 if Black wins).

4.2 Experimental Information

To compare the different network architectures we trained them on multiple epochs. One epoch uses $100,000$ states randomly selected from the Katago dataset with two labels: a one-hot encoding of the Katago move and an evaluation between 0 and 1 by Katago of the winrate for White. The training is done with Adam and cosine annealing [9] without restarts. Cosine annealing leads to better convergence by modifying the learning rate of Adam.

In the next tables, we denote Residual(X,Y), the Residual Network of X blocks of Y planes and we denote Efficient(lX), the lX architecture of EfficientFormer. Among the different metrics used, we compute the accuracy, mean

squared error (MSE), mean absolute error (MAE), and when possible the winning rate against an opponent. The winning rate is more informative than the other because it combines the impact of improving policy and value network. It also takes into account the latency of the networks. Accuracy measures the closeness of strategies between the policy network and Katago data. In the following experiment, we present the result on the validation set.

4.3 Training and Playing

Our research paper focuses on the study of transformers and their impact on several performance criteria, as well as speed and memory. In our experiments, we first analyze the best hyperparameters obtained on performance for models of varying sizes (l1/l3/l5/l9) and with varying lengths of training (100/500/1000 epochs). Our results on Efficient are compared against Residual Network, which is the current state of the art in the field. After that, we delve into a comprehensive analysis with a specific focus on assessing memory consumption and processing speed for each model.

In Table 1, we analyze the impact of the learning rate on the Accuracy, MSE, and MAE for multiple Residual networks across 1000 epochs in comparison to Efficient(l9). We will then use the best learning rate in Table 2 where we analyze the Accuracy, MSE, and MAE for both Residual and Transformer networks across 500 epochs. We found that Efficient(l9) performs better than the other Efficient Former and Residual networks. Additionally, Figure 4 provides a visual comparison of the training curves for Accuracy, MAE, and MSE between Efficient (l9) and Residual (20,256) over 1000 epochs.

In Tables 3 and 4, an analysis of winning rates among multiple algorithms compared to Efficient (l9) over 500 and 1000 epochs is presented. It is noteworthy that Efficient (l9) exhibits superiority over the majority of algorithms, demonstrating a remarkable performance trend. An exception is observed in the case of the 500-epoch scenario 3, where Efficient (l7) achieves comparable performance on CPU, and Residual (20,256) attains equivalent performance on GPU. The longer training time and the better GPU are in favor of the transformer network.

In Table 5, we conduct an analysis of latency and peak memory utilization on GPU and CPU for the various network architectures under examination. The experiments on GPU were carried on a *RTX 2080 Ti* with 11 Gb of Memory and *Epyc* server for the CPU. On GPU, it is noteworthy that both Residual Networks and Transformers exhibit similar latency characteristics, however, Transformers incur a memory usage approximately three times greater than that of Residual Networks. On CPU (Table 5), for smaller network, we observe that Efficient architecture achieve a lower evaluation rate per second than Residual. Nevertheless, this trend shifts as network sizes increase, with the Efficient architecture ultimately surpassing Residual Networks in evaluation per second.

Lastly, Table 6 charts the evolution of GPU latency concerning batch size variation for the different network configurations. Large batch sizes are relevant to self-play in Alpha Zero style [28, 31]. Smaller batch sizes are relevant to normal

play with batch parallel MCTS [6]. This analysis sheds light on how network performance scales with batch size changes. The Residual Networks use relatively more playouts since they parallelize better with current GPU hardware and software.

Table 1: Comparaison of large network architectures for 1000 epochs of 100,000 states per epoch.

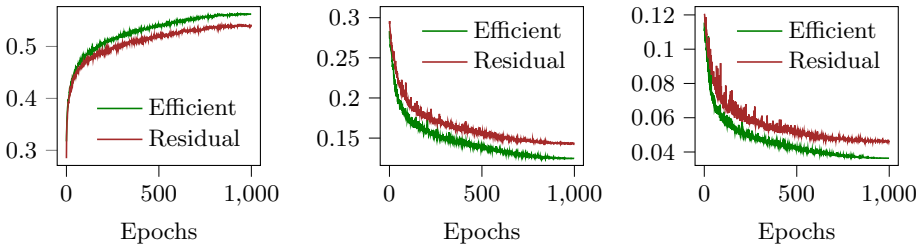
Network	Size	Learning Rate	Batch	Accuracy	MSE	MAE
Residual	20,256	0.0002	64	52.13%	0.0496	0.1510
	20,256	0.0001	64	53.30%	**0.0458**	0.1456
	20,256	0.00005	64	**53.82**%	0.0465	**0.1434**
	20,256	0.00002	64	53.28%	0.0471	0.1483
	20,256	0.00001	64	51.23%	0.0516	0.1595
	40,256	0.0002	64	45.68%	0.0717	0.1990
	40,256	0.0001	64	51.88%	0.0513	0.1599
	40,256	0.00005	64	52.64%	**0.0475**	**0.1493**
	40,256	0.00002	64	**52.90**%	0.0486	0.1497
	40,256	0.00001	64	51.10%	0.0551	0.1625
Efficient	l9	0.0005	32	**56.22**%	**0.0360**	**0.1240**

Table 2: Comparison of networks for 500 epochs of 100,000 states per epoch and a batch size of 64. The Accuracy, MSE and MAE were computed on a set of 50,000 states sampled from 50,000 games that were never seen during training.

Network	Size	Learning Rate	Accuracy	MSE	MAE
Residual	10,128	0.0002	49.12%	0.0534	0.1649
	20,128	0.0002	50.29%	0.0516	0.1618
	20,256	0.00005	**52.50**%	**0.0476**	**0.1518**
	40,256	0.00005	51.27%	0.0499	0.1586
Efficient	l1	0.002	49.35%	0.0553	0.1659
	l3	0.002	51.28%	0.0484	0.1519
	l7	0.002	53.01%	0.0440	0.1422
	l9	0.001	**54.29**%	**0.0405**	**0.1351**

5 Conclusion

This paper investigates the impact of the Vision Transformer architecture on the game of Go. Building upon the proven success of the Transformer architecture



(a) Accuracy of the policy (b) MAE of the value (c) MSE of the value

Fig. 4: Efficient(l9) versus Residual(20,256) comparisons on Accuracy, MAE and MSE over 1,000 epochs of 100,000 states. All values are computed with the test set that contains games that were never seen in the training set.

Table 3: Comparison of networks for 500 epochs of 100,000 states per epoch and a batch size of 64. The winrate WinCPU is the result of 800 randomized matches on CPU against Efficient(l9) with 10 seconds of CPU per move for both sides. The GPU winrate is calculated by using the same RTX 2080 Ti GPU time for both networks with a batch of 64. The learning rate is fixed to 0.0005 for the Residual network and 0.001 for the Efficient network.

Network	Size	vs Efficient(l9)	
		WinCPU	WinGPU
Residual	10,128	33.5%	20.4%
	20,128	31.6%	25.8%
	20,256	30.6%	46.4%
	40,256	8.9%	29.7%
Efficient	l1	11.6%	8.1%
	l3	31.0%	19.4%
	l7	50.4%	38.3%

Table 4: Comparison of the winning rate for multiple networks trained during $1,000$ epochs of $100,000$ states per epoch. The winrate WinCPU is the result of 400 randomized matches on CPU with 10 seconds of CPU per move for both sides. The GPU winrate is calculated by using the same A6000 GPU time for both networks with a batch of 32 for the inference.

Network	Size	vs Efficient(l9)	
		WinCPU	WinGPU
Residual	20,256	30.5%	39.0%
	40,256	15.0%	33.0%

Table 5: Latency and number of evaluations per second on CPU/GPU for different architectures and networks of different sizes and peak memory on GPU. The CPU used is *Epyc* server. The GPU used is a RTX 2080 Ti GPU with 11 Go. The latency and the peak memory on the GPU are measured using a batch of 64 states. They are averaged over 100 calls to predict after a warmup of 100 previous calls. The latency is the average time in seconds to make a forward pass.

Network	Size	CPU		GPU		
		Latency	Evals/s	Latency	Evals/s	Peak Memory
Residual	10,128	0.043	23.07	0.0890	719	436,656,640
	20,128	0.082	12.24	0.0943	679	350,025,728
	20,256	0.304	3.29	0.1185	540	452,578,816
	40,256	0.455	2.20	0.1580	405	529,187,072
Efficient	l1	0.065	15.27	0.0958	668	1,101,474,048
	l3	0.074	13.52	0.1106	579	1,148,030,976
	l7	0.092	10.90	0.1307	490	1,159,418,368
	l9	0.159	6.30	0.1700	376	1,179,129,088

Table 6: Evolution of the A6000 GPU latency with the size of the batch. The latency and the peak memory are the median values of 7 runs. Each run is the average over 100 forwards after a warmup of 100 forwards.

Network	Size	Batch	GPU Latency	Evals/s on GPU	Peak Memory
Residual	20,256	32	0.111	288	253,801,472
	20,256	64	0.126	508	548,938,240
	20,256	128	0.159	805	800,936,192
	20,256	256	0.227	1,128	1,566,134,528
	20,256	512	0.368	1,391	2,954,716,416
	20,256	1,024	0.667	1,535	4,793,448,960
Efficient	l9	32	0.128	250	589,454,592
	l9	64	0.168	381	1,141,404,672
	l9	128	0.224	571	2,297,159,168
	l9	256	0.346	740	4,359,236,608
	l9	512	0.583	878	8,672,660,992
	l9	1,024	1.062	964	17,121,701,376

across diverse domains, our investigation seeks to explore its potential in the context of Go. Our analysis traverses a multitude of critical dimensions, ranging from prediction accuracy and win rates to memory utilization, processing speed, and learning rates. Through this examination, we underscore the significant role that Transformers can play in enhancing performance in the game of Go.

Significantly, our findings highlight the benefits of the EfficientFormer architecture, showcasing remarkable performance enhancements on both CPU and GPU platforms. Notably surpassing the benchmarks set by the previous state-of-the-art algorithms, this superiority becomes particularly pronounced in the context of larger networks, underscoring the scalability and efficiency of the EfficientFormer.

In addition, it is essential to emphasise that the impact and adaptability of the EfficientFormer architecture goes beyond the boundaries of the game of Go, extending its applicability to a wide range of games and domains. This versatility positions the EfficientFormer as a promising candidate for pushing the boundaries of artificial intelligence not only in strategic board games but also in various other complex decision-making scenarios.

References

1. Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of Monte Carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in Games **4**(1), 1–43 (Mar 2012)
2. Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., Zagoruyko, S.: End-to-end object detection with transformers. In: European conference on computer vision. pp. 213–229. Springer (2020)
3. Cazenave, T.: Residual networks for computer go. IEEE Trans. Games **10**(1), 107–110 (2018)
4. Cazenave, T.: Spatial average pooling for computer go. In: Computer Games: 7th Workshop, CGW 2018, IJCAI 2018, Stockholm, Sweden, July 13, 2018, Revised Selected Papers 7. pp. 119–126. Springer (2019)
5. Cazenave, T.: Improving model and search for computer go. In: 2021 IEEE Conference on Games (CoG). pp. 1–8. IEEE (2021)
6. Cazenave, T.: Batch Monte Carlo Tree Search. In: Browne, C., Kishimoto, A., Schaeffer, J. (eds.) Computers and Games 2022. Lecture Notes in Computer Science, vol. 13865, pp. 146–162. Springer (2022)
7. Cazenave, T.: Mobile networks for computer go. IEEE Trans. Games **14**(1), 76–84 (2022)
8. Cazenave, T., Chen, Y., Chen, G., Chen, S., Chiu, X., Dehos, J., Elsa, M., Gong, Q., Hu, H., Khalidov, V., Li, C., Lin, H., Lin, Y., Martinet, X., Mella, V., Rapin, J., Rozière, B., Synnaeve, G., Teytaud, F., Teytaud, O., Ye, S., Ye, Y., Yen, S., Zagoruyko, S.: Polygames: Improved zero learning. J. Int. Comput. Games Assoc. **42**(4), 244–256 (2020)
9. Cazenave, T., Sentuc, J., Videau, M.: Cosine annealing, mixnet and swish activation for computer go. In: Browne, C., Kishimoto, A., Schaeffer, J. (eds.) Advances in Computer Games - 17th International Conference, ACG 2021, Virtual Event,

November 23-25, 2021, Revised Selected Papers. Lecture Notes in Computer Science, vol. 13262, pp. 53–60. Springer (2021)
10. Chen, X., Wu, Y., Wang, Z., Liu, S., Li, J.: Developing real-time streaming transformer transducer for speech recognition on large-scale dataset. In: ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 5904–5908. IEEE (2021)
11. Coulom, R.: Efficient selectivity and backup operators in Monte-Carlo tree search. In: van den Herik, H.J., Ciancarini, P., Donkers, H.H.L.M. (eds.) Computers and Games, 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers. Lecture Notes in Computer Science, vol. 4630, pp. 72–83. Springer (2006)
12. Coulom, R.: Computing elo ratings of move patterns in the game of Go. ICGA Journal **30**(4), 198–208 (2007)
13. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. pp. 4171–4186 (2019)
14. Dong, L., Xu, S., Xu, B.: Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition. In: 2018 IEEE international conference on acoustics, speech and signal processing (ICASSP). pp. 5884–5888. IEEE (2018)
15. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al.: An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929 (2020)
16. Gelly, S., Silver, D.: Monte-Carlo tree search and rapid action value estimation in computer Go. Artif. Intell. **175**(11), 1856–1875 (2011)
17. Gulati, A., Qin, J., Chiu, C.C., Parmar, N., Zhang, Y., Yu, J., Han, W., Wang, S., Zhang, Z., Wu, Y., et al.: Conformer: Convolution-augmented transformer for speech recognition. arXiv preprint arXiv:2005.08100 (2020)
18. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
19. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017)
20. Kocsis, L., Szepesvári, C.: Bandit based Monte-Carlo planning. In: 17th European Conference on Machine Learning (ECML'06). LNCS, vol. 4212, pp. 282–293. Springer (2006)
21. Li, Y., Yuan, G., Wen, Y., Hu, J., Evangelidis, G., Tulyakov, S., Wang, Y., Ren, J.: Efficientformer: Vision transformers at mobilenet speed. Advances in Neural Information Processing Systems **35**, 12934–12949 (2022)
22. Parmar, N., Vaswani, A., Uszkoreit, J., Kaiser, L., Shazeer, N., Ku, A., Tran, D.: Image transformer. In: International conference on machine learning. pp. 4055–4064. PMLR (2018)
23. Qiu, X., Sun, T., Xu, Y., Shao, Y., Dai, N., Huang, X.: Pre-trained models for natural language processing: A survey. Science China Technological Sciences **63**(10), 1872–1897 (2020)
24. Rives, A., Meier, J., Sercu, T., Goyal, S., Lin, Z., Liu, J., Guo, D., Ott, M., Zitnick, C.L., Ma, J., et al.: Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. Proceedings of the National Academy of Sciences **118**(15), e2016239118 (2021)

25. Schwaller, P., Laino, T., Gaudin, T., Bolgar, P., Hunter, C.A., Bekas, C., Lee, A.A.: Molecular transformer: a model for uncertainty-calibrated chemical reaction prediction. ACS central science **5**(9), 1572–1583 (2019)
26. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of go with deep neural networks and tree search. Nature **529**, 484–489 (2016)
27. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T.P., Simonyan, K., Hassabis, D.: Mastering chess and shogi by self-play with a general reinforcement learning algorithm. CoRR **abs/1712.01815** (2017), http://arxiv.org/abs/1712.01815
28. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al.: A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science **362**(6419), 1140–1144 (2018)
29. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. Advances in neural information processing systems **27** (2014)
30. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: 31st International Conference on Neural Information Processing Systems (NIPS 2017). p. 6000–6010 (2017)
31. Wu, D.J.: Accelerating self-play learning in go. arXiv preprint arXiv:1902.10565 (2019)
32. Wu, I.C., Wu, T.R., Liu, A.J., Guei, H., Wei, T.: On strength adjustment for MCTS-based programs. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33, pp. 1222–1229 (2019)
33. Wu, T.R., Wei, T.H., Wu, I.C.: Accelerating and improving AlphaZero using population based training. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 34, pp. 1046–1053 (2020)
34. Wu, T.R., Wu, I.C., Chen, G.W., Wei, T.H., Wu, H.C., Lai, T.Y., Lan, L.C.: Multilabeled value networks for computer go. IEEE Transactions on Games **10**(4), 378–389 (2018)