Effect of ensemble size on performance for chess position classification

Thijs Bertram
Deep Learning, Tilburg Univeristy
05-04-2019

Abstract

This paper examines the influence of Ensemble size on task performance for a binary classification problem. The task consists of classifying chess positions as winning for either white or black. Different sizes and architectures of neural network ensembles are compared against a baseline model. A positive relationship between ensemble size and performance is found, but this effect diminishes as ensembles get bigger. The best ensemble correctly predicts 87,1% of the 28.148 positions. The effect of ensemble size on prediction time has been studied as well and appears to be of linear nature.

1 Introduction

Chess is an extensively-studied problem in artificial intelligence, mostly due to the complex yet well-defined nature of the game. Probably thousands of scientific studies on computer chess have been written since 1950 (Lai 2015). One could even argue that the task is solved: Already in 1997 IBM's Deep Blue defeated the world champion Garry Kasparov. Since then chess engines have been beating the best human players consistently. The task for chess engines is not an easy one though and can be defined as follows: Given a chess position, come up with the move that maximizes the difference in probability of winning the game. Thus, all chess engines need a way to evaluate a chess position, as well as a way to efficiently compare lots of possible positions. There is still a lot of room for improvement, even in the state-of-the-art chess engines.

Modern chess engines use some sort of minimax tree-search in combination with alpha beta pruning to compare a wide range of possible positions. The evaluation function is either hardcoded (Lai 2015), or estimated via extensive reinforcement learning (Silver et al. 2017). AlphaZero is considered to be the most powerful engine around at the moment. It uses deep neural networks in combination with reinforcement learning to estimate an evaluation function. AlphaZero had to play 44 million games against itself, which were than processed on 64 second generation TPU chips during training.

This paper will explore the possibility of using deep supervised learning to evaluate positions, instead of relying on costly hardcoding or reinforcement learning processes. An ensemble of neural networks will be trained on a relatively small dataset. Deep forward neural networks are universal function approximators (Hornik, Stinchcombe, and White 1989), so they should theoretically be able to represent evaluation scores based on positional input. The next section will briefly cover the opportunities and pitfalls of deep learning models.

1.1 Deep neural networks

Deep learning algorithms are a subset of machine learning algorithms. They learn abstract, high-level representations of data by using a hierarchical architecture of non-linear functions. The theory behind neural networks stems from the 1950's, but the recent surge in computing power and available data has enabled viable applications of this technique. Especially neural network architectures with many hidden layers, also called *deep* learning models, have shown promising results on many traditional machine-learning tasks.

In fact, deep learning models are regarded as state-of-the-art in many domains: Deep Convolutional Neural Networks (CNNs) dominate the field of computer vision (Guo et al. 2016), Deep Long Short Term Memory networks (LSTMs) are at the core of encoder and decoder networks which are widely used in machine translation, and Deep Generative Networks (GANs) have proven hugely successful at unsupervised learning (Mao et al. 2017).

Deep learning networks are clearly very powerful, but that does not mean that it comes without limitations.

1.2 Overfitting

One of the main limitations of deep learning is their tendency to overfit (Livingstone, Manallack, and Tetko 1997). Overfitting happens when the model fits the training data too closely; leading to a drop in performance on the test set. This can have all sorts of reasons; sometimes the model is simply too deep for the task at hand (especially the case for relatively simple tasks). It could also be the effect of training the model for too many epochs; in this case, the training examples are seen too many times which causes the weights to be too specifically tailored towards samples it has already seen, leaving less room for generalizing to unseen examples. Generalizing to unseen examples also becomes a problem when the training dataset is too small, leading to an internal representation

that is too narrow for the task at hand (Srivastava et al. 2014).

Finding ways to combat overfitting has been a popular research topic within the field of Deep learning for decades already, so it may come as no surprise that lots of so-called regularization methods have already been developed (Srivastava et al. 2014). Perhaps the most well-known regularization technique is that of Dropout, which temporarily removes nodes from the network, along with all of its incoming and outgoing connections (Srivastava et al. 2014). L2 regularization (weight penalties) and Batch-Normalization (normalizing layer outputs for every batch) are other popular regularization techniques. The subject of this paper could also be seen as regularization method; ensembles are collections of models which are known to reduce variance, resulting in higher generalizability. It is known that larger ensembles correlate with higher results, but the exact relationship is hard to determine (DelSole, Nattala, and Tippett 2014) and has been an active research question in the field for the last two decades (Rokach 2010). It may be interesting to contribute to this line of research by studying the effect of ensemble size on performance for an new type of dataset. The next section of this paper will cover the different types of ensembles, their algorithms and the effect of ensemble size on performance in more detail. For now, the research question of this paper could be defined as:

To what extent does the size of ensemble methods influence the performance on the binary classification task of classifying chess positions?

2 Architecture and Methods

I will construct ensembles of different sizes, all consisting of feed forward neural networks (FFNN). A visual representation of the model architecture can be seen in figure 1. Weight kernels were initialized as a random uniform distribution, except for weights in boosted methods. Weights are not initialized for these models, since that lead to an value greater than 0.5 which made training enough models impossible.

Dropout layers are added after every Dense layer to make sure that there will at least be *some* variance between individual models of the ensemble. A relatively small value was picked after some initial testing: bigger values lead to a drop of performance for the baseline model. The number of models that an ensemble will contain varies from 1 (acting as a baseline) up to 25.

2.1 Ensembles

An ensemble can be described as a *collection of models* and is based on the notion of 'wisdom of the crowds'. Instead of relying on one model, a group of models is trained after which their individual output is combined into one, final prediction. A wide variety of ensemble architectures has been developed which mostly differ in either 1) the way models are trained or 2) the way individual outputs are aggregated into the final output. The most bare-bones ensemble consists of models which are trained in parallel and combined by means of averaging

Layer (type)	Nr. of nodes	Output Shape	Param #
dense_1	130	(None, 130)	100100
dropout_1		(None, 130)	0
dense_2	130	(None, 130)	17030
dropout_2		(None, 130)	0
dense_3	260	(None, 260)	34060
dropout_3		(None, 260)	0
dense_4	260	(None, 260)	67860
dropout_4		(None, 260)	0
dense_5	130	(None, 130)	33930
dropout_5		(None, 130)	0
dense_6	130	(None, 130)	17030
dropout_6		(None, 130)	0
dense_7	65	(None, 65)	8515
dense_8	2	(None, 2)	132
Total params: 278, 657			
Trainable params: 278, 657			
Non-trainable params: 0			

Table 1: The architecture of individual FFNN models

the predictions. This paper will study the effect of ensemble size on accuracy for such a a bare-bones ensemble, as well as more complex ensemble architectures. The next section will cover these architectures and their respective algorithms in more detail.

Bagging and boosting

Bagging and boosting are ways of training the individual models belonging to an ensemble. In the bare-bones example above, models are trained in parallel on all available training data. Bagging and boosting work a little differently; individual models are trained on a subset of the training data to make sure that there is variance between individual models.

Bagging selects a bootstrapped subset of data for every classifier. This bootstrapped set S_t is chosen randomly by drawing $R\%$ samples of training data S. The result is n number of subsets for n number of ensembles. Samples can occur multiple times in one subset, or can be absent in a subset altogether (Zhang and Ma 2012).

Boosting also creates unique subsets of data for every classifier, but does not sample according to an equal probability distribution. Instead, it updates its probability distribution D_t based on the performance of previous models. The sampling probabilities for miss classified examples are increased, so the next subset $S_t + 1$ contains more samples that are 'difficult to classify' (Zhang and Ma 2012). The AdaBoost.M1 algorithm will be used as implementation of Boosting.

Mean vs. Weighted voting

We can also apply different techniques for aggregating predictions of individual models. The simplest method is ag-

gregate by means of 'majority vote'. A more sophisticated method however, is that of mean voting. Mean voting calculates the mean of all classes over all models, outputting the class with the highest average probability. The drawback is obviously that every model is treated as equally important, while some models may be better than others.

Weighted voting aims to improve the aggregation process by assigning weights to individual models that represent their individual fitness. These weights are based on validation accuracy (Zhang and Ma 2012) and can be either absolute, or based on rank. Validation accuracy can be similar among networks, so weights may not be different enough when validation scores are used as absolute weights. Another possibility is to create a uniform distribution that is based on the 'ranking' of validation scores (Rokach 2010). This paper will test both of the above mentioned weighing methods.

2.2 Ensemble size and performance

Ensemble size is known to positively impact task performance (Mitchell, Houtekamer, and Pellerin 2002), but at the same time also incurs a higher computational cost (Partalas, Tsoumakas, and Vlahavas 2008). It is a well-studied trade-off, but no one-size-fits-all answer has been found. It appears that increasing ensemble size yields better results because of an increase in diversity. So, one should aim towards constructing ensembles that are as small as possible, while maintaining the level of diversity of the original ensemble (Liu, Mandvikar, and Mody 2004a).

Zhou and colleauges indeed proved that smaller ensembles may be equally powerful as the original ensemble (Zhou, Wu, and Tang 2002). How much an original ensemble can be constringed without losing performance is, for the most part, dependent on the type and complexity of the dataset (Liu, Mandvikar, and Mody 2004b). It is especially interesting to study the relation between ensemble size and performance on a chess position classification task, because such a dataset has not been used before. The insights of this paper can therefore hopefully contribute to understanding this relationship a little better.

3 Experimental setup

This section contains a formal description of the experiment so that future studies are able to reproduce the results.

3.1 Data

The dataset contains 112.592 chess positions, represented by a 1D-array of shape (769,) and is specifically scraped for this experiment. Twenty percent of the data is split off for testing, resulting in a training set of 84.444 samples and a test set of 28.148 samples. Class labels are slightly imbalanced: white is winning in 59,8% and 60.1% of the positions in the training and test set respectively. This is not a big issue, since it is ... to the game of chess: white simply has a greater chance at winning the game than black (Rowson 2005). I will not report on other statistical properties, since mean and standard deviation do not make sense for bitmap data.

Representing chess positions

The vector of training samples is made up of two parts: a bitmap of length 768 representing board position, as well as one additional bit that represents the player to move. The bitmap representation of chess positions has been used extensively in previous research (Lai 2015). It represents all of the 64 squares by 12 binary features that indicate whether a piece of each of 12 piece types is present on that square or not. This results in the $64 * 12 = 768$ bitmap. I decided to add the final bit that represent if it is whites turn to move or not, because that information is vital information when it comes to scoring chess positions.

Scraping

The dataset has been scraped by inputting Portable Game Notation (PGN) into the online analysis tool of Lichess. A selenium chrome web-driver is used to input PGN. After that, the html code of Lichess' board representation is extracted and converted to a bitmap representation using a custom function. Finally the positional score is extracted from Lichess' html code as well. These positional scores are than transformed into an array of two binary values: the first bit represents a winning position for black, the second represents a winning position for white.

The PGN's are randomly generated from a collection of amateur and professional games. I deliberately added amateur games to the dataset because they tend to end up in more unconventional positions, leading to a higher variance of the training data. A hashmap was used to make sure that every unique position would only occur once in the training data. I chose to scrape the data from Lichess because their scoring system is based on Stockfish 10+, one of the strongest chess engines around. On top of that they have a very structured HTML code, making it an easy and reliable source to scrape from.

All the code for scraping and pre-processing can be found on my Github.

3.2 Experiment

Five experiments have been conducted to study the relationship between ensemble size and performance. For each experiment 25 models were trained. Than 25 ensembles of size 1 up to 25 were constructed based on random sampling of trained models. The five experiments are based on the following ensemble architectures:

- No subsetting of training data with mean voting
- No subsetting of training data with weighted voting based on rank
- No subsetting of training data with weighted voting based on absolute scores
- Bagging with mean voting
- bagging with weighted voting based on rank
- bagging with weighted voting based on absolute scores
- Boosting

It was not possible to vary aggregation methods for ensembles that are trained by boosting. Boosting has its own way of

aggregating models which is based on the β parameter. I refer readers to this article (Rokach 2010) for more information about this parameter.

3.3 Hyperparemter settings

After some initial experimenting with the baseline model the following hyperparameters were chosen:

- Adam optimizer with
  - Learning rate: 0.001
  - beta_1: 0.9
  - beta_2: 0.999
- Number of epochs: 15
- Batch size: 150
- Shuffle: True
- Validation split: 0.2

These settings seem to be performing best within the limitations of the available hardware.

3.4 Performance metrics

Performance of the different ensemble models is evaluated by means of a comparison of test scores. The metric used is Keras' binary accuracy metric. A table, as well as several plots describing the findings can be found in the result section of the paper.

4 Results

The result section is split up into two parts. In the first part I will present the results of ensembles which are trained on the whole dataset, i.e. without any subsetting. The second part presents the results of ensembles which are trained on a subset of data, either by means of boosting or bagging.

4.1 No subset

Table 2 presents the results for ensemble models that have been trained without subsetting any data. Let us first look at the general impact of ensemble size on performance. The performance of small ensembles seems to grow rapidly when ensemble size is increased. Using an ensemble of 3 models results in an average increase of 3,2% accuracy, compared to using just one classifier. This steady increase in performance continues up until an ensemble size of around 7 or 8, from than on it seems to depend on which models are added. This is unfortunately heavily reliant on the random sampling.

An other interesting observation is that the accuracy for the maximum ensemble size is equal across all three aggregation methods. Let us take a closer look at the difference between aggregation methods for other ensemble sizes. We can see that differences in accuracy between aggregation methods are quite substantial for the small ensemble sizes, but seem to diminish as the ensemble size gets bigger.

Figure 1 is included to give the reader an intuitive feel as to how the influence of aggregation method on accuracy relates to ensemble size. Aggregating by weights based on ranking yields the worst performance: the base classifier is the strongest, but performance for almost every ensemble is worse than the other two methods. Aggregating by mean absolute weights seem to be equal; both oscillating but in an opposite phase.

	mean	weighted rank	weighted absolute
1	0,825	0,829	0,812
2	0,847	0,837	0,852
3	0,855	0,853	0,855
4	0,859	0,857	0,863
5	0,862	0,860	0,861
6	0,864	0,859	0,861
7	0,865	0,864	0,865
8	0,863	0,864	0,866
9	0,865	0,864	0,865
10	0,868	0,865	0,865
11	0,866	0,865	0,867
12	0,868	0,867	0,866
13	0,868	0,866	0,867
14	0,867	0,868	0,867
15	0,868	0,867	0,868
16	0,868	0,868	0,868
17	0,868	0,868	0,869
18	0,868	0,868	0,869
19	0,869	0,869	0,868
20	0,869	0,869	0,869
21	0,868	0,869	0,869
22	0,869	0,868	0,868
23	0,869	0,869	0,869
24	0,870	0,870	0,869
25	0,869	0,869	0,869

Table 2: Binary accuracy on the test set for ensembles trained without any subsetting of training data

4.2 Bagging and boosting

Table 3 shows the results for ensemble models that have been trained on a subset of data. Most of the findings from table 1 also apply to the Bagging section of table 2; performance of small ensembles grows rapidly when increasing their size and accuracy for maximum ensemble size is more or less equal across aggregation methods.

The results of boosted models are substantially different compared to the other training methods. Performance is lower for virtually every ensemble size and only increases by a marginal amount. Another interesting observation is that there are few models that actually increase accuracy on the test set. The models that are added in ensemble twelve (from ensemble size twelve to thirteen) and nineteen are

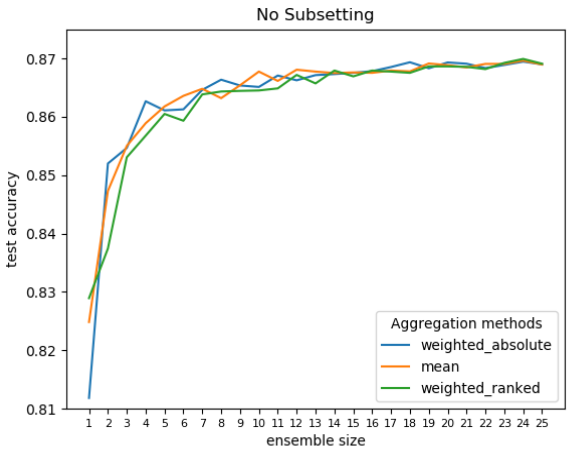

Figure 1: Relation between ensemble size and test accuracy for different aggregation methods

clearly adding some value to the ensemble. Some models, on the other hand, actually decrease performance by a marginal amount. Again, figure 2 is added to get represent above described relationships.

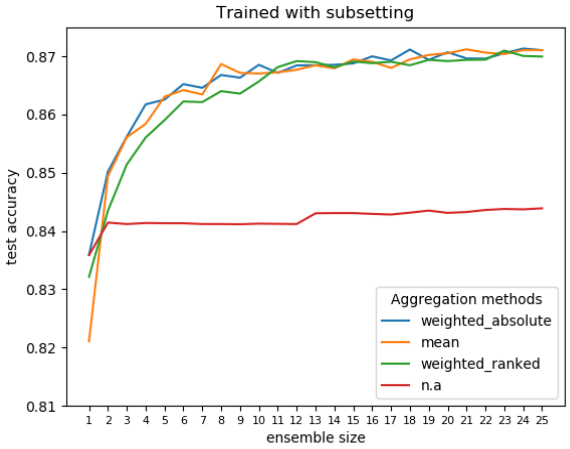

Figure 2: Relation between ensemble size and test accuracy for different aggregation methods

4.3 Prediction time

Figure 3 shows the prediction time for all different ensemble types. The relationship between prediction time and ensemble size is one of linear nature. Let us now compare the differences in prediction time for different aggregation methods in more detail. Aggregating by absolute weights has the highest time-cost, followed by aggregating by ranked weights. Weighing for Boosted models, which is based on β scores is by far the most time efficient.

	Bagging			Boosting
	mean	weighted rank	weighted absolute	n.a
1	0,821	0,832	0,836	0,836
2	0,849	0,843	0,850	0,841
3	0,856	0,851	0,856	0,841
4	0,858	0,856	0,862	0,841
5	0,863	0,859	0,863	0,841
6	0,864	0,862	0,865	0,841
7	0,863	0,862	0,865	0,841
8	0,869	0,864	0,867	0,841
9	0,867	0,864	0,866	0,841
10	0,867	0,866	0,869	0,841
11	0,867	0,868	0,867	0,841
12	0,868	0,869	0,868	0,841
13	0,868	0,869	0,868	0,843
14	0,868	0,868	0,869	0,843
15	0,869	0,869	0,869	0,843
16	0,869	0,869	0,870	0,843
17	0,868	0,869	0,869	0,843
18	0,869	0,868	0,871	0,843
19	0,870	0,869	0,869	0,844
20	0,871	0,869	0,871	0,843
21	0,871	0,869	0,870	0,843
22	0,871	0,869	0,870	0,844
23	0,870	0,871	0,871	0,844
24	0,871	0,870	0,871	0,844
25	0,871	0,870	0,871	0,844

Table 3: Binary accuracy on the test set for ensembles trained without any subsetting of training data

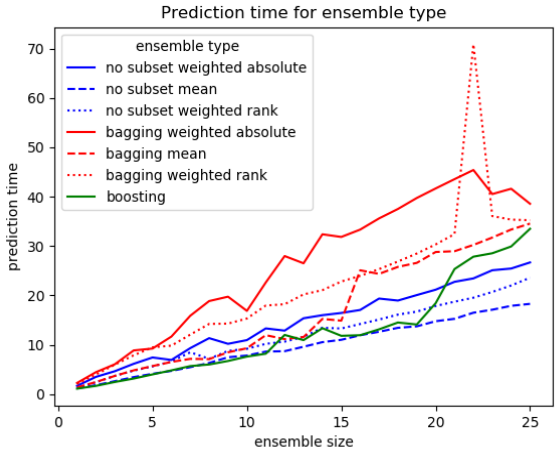

Figure 3: Relation between ensemble size and prediction time

For models trained with bagging or no subsetting of data, mean aggregation is the most efficient method. The difference in prediction times for different methods of training are unexpected and may be caused by hardware performance. I will address this in more detail in the next chapter of this paper.

5 Conclusion and Discussion

Chess engines are a well studied phenomenon in machine learning literature. The evaluation of positions is the backbone of these engines, but training such a model is often time and computationally intensive. This paper explored the possibility of using ensembles of feed forward neural networks without the need of any reinforcement learning for evaluating positions. The results are promising; the best ensemble of classifiers was able to correctly classify 87,1% percent of 28.148 chess positions as winning for either black or white.

I specifically studied the relationship between ensemble size and classification accuracy. Bigger ensembles result in better accuracy, just as expected. This positive effect of ensemble size on performance diminishes the larger the size of ensembles. That is in line with previous literature and can be explained by variance of the ensemble models: a model is less likely to add variance to to bigger ensembles.

I also conducted experiments to study the relation between time and ensemble size, which appears to be of linear nature. The aspect of time is especially important, since time is a limited resource in chess which causes a player to lose if they run out of it (Gobet and Simon 1996). Ensembles that are trained with bagging and aggregated by mean vote seem to be the best fit for this specific task. This ensemble architecture is the most time-efficient and among the highest scoring architectures.

The ideal ensemble size is hard to determine, since it involves a trade-off between time and performance. An ensemble of size 8 seems to be fine (relatively high performance and low prediction time), but we can not be sure due to limitations of this study. The next section will briefly cover those limitations and recommend directions for future work.

5.1 Limitations

Unfortunately the limitations of this paper are abundant. Firstly, the bitmap representation of chess positions is outdated (Hyatt 1999). In the bitmap representation three positions could have a similar distance score, while the positional implications for the game are huge. The feature space thus is insensitive to some specific positional advantages. More sophisticated techniques have been developed (Lai 2015), but were not implemented due to time constraints.

Secondly, one could argue that approaching position evaluation as a binary task is unrepresentative of the problem at hand. Chess engines almost always work with a floating point representation of a positions, positive values indicating a winning position for white. Approaching it as a binary classification task boosts model performance tremendously, but that is not the reason I chose to do so. Individual models in an ensemble would need to be a lot deeper and more sophisticated for it to work. This would unfortunately make the task too demanding for the hardware at hand.

Finally, the results could be significantly influenced by stochasticity. Every experiment is only run once so every stochastic process can theoretically have an impact on the results. There is randomness involved in initializing the weights of individual models, as well as in selecting samples of models when constructing ensembles. There exist quite a few methods to select models based on performance instead of random sampling (Saeys, Abeel, and Van de Peer 2008; Caruana et al. 2004). In table 2 and 3 you can actually see this: some models cause a drop in performance.

The relation between time and ensemble size also seems to be influenced by stochasticity in hardware performance. The model architecture of models that are trained on the whole dataset are the same as models that are trained by means of bagging. The aggregation methods are the exact same as well, so you would not expect such a big difference in prediction time for models trained on the whole dataset and models trained with bagging (as can be seen in figure 3).

5.2 Future work

Future work could eliminate the above mentioned influence of stochasticity by repeating the experiment multiple times and averaging the results. It could also be interesting to experiment with different ensemble architectures. More sophisticated algorithms for training ensemble models or aggregating their results could be implemented and compared with the ones that are used in this paper.

Another interesting idea is to experiment with different model architectures by changing the number of layers, number of nodes or other hyperparameters. Completely different types of neural networks could even be used, like Convolutional Neural Netowrks which have shown promising results when it comes to positional evaluation (Oshri and Khandwala 2016).

The experimental setup of this paper can be applied to predicting chess positions as a floating point value as well. This task will be a lot harder so a better dataset would be recommended.

The ultimate test of position evaluation though, is its performance in an actual chess engine. It would be valuable to be able to compare an engine that uses an ensemble based approach to other existing engines. Only than we will know how the performance versus time trade-off plays out and how viable an ensemble based approach really is.

References

Caruana, Rich et al. (2004). "Ensemble selection from libraries of models." In: *Proceedings of the twenty-first international conference on Machine learning*. ACM, p. 18.

DelSole, Timothy, Jyothi Nattala, and Michael K Tippett (2014). "Skill improvement from increased ensemble size and model diversity." In: *Geophysical Research Letters* 41.20, pp. 7331–7342.

Gobet, Fernand and Herbert A Simon (1996). "The roles of recognition processes and look-ahead search in time-constrained expert problem solving: Evidence from grandmaster-level chess." In: *Psychological science* 7.1, pp. 52–55.

Guo, Yanming et al. (2016). "Deep learning for visual understanding: A review." In: *Neurocomputing* 187, pp. 27–48.

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White (1989). "Multilayer feedforward networks are universal approximators." In: *Neural networks* 2.5, pp. 359–366.

Hyatt, Robert M (1999). "Rotated bitmaps, a new twist on an old idea." In: *ICGA Journal* 22.4, pp. 213–222.

Lai, Matthew (2015). "Giraffe: Using deep reinforcement learning to play chess." In: *arXiv preprint arXiv:1509.01549*.

Liu, Huan, Amit Mandvikar, and Jigar Mody (2004a). "An Empirical Study of Building Compact Ensembles." In: *Advances in Web-Age Information Management*. Ed. by Qing Li, Guoren Wang, and Ling Feng. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 622–627. ISBN: 978-3-540-27772-9.

– (2004b). "An empirical study of building compact ensembles." In: *International Conference on Web-Age Information Management*. Springer, pp. 622–627.

Livingstone, David J, David T Manallack, and Igor V Tetko (1997). "Data modelling with neural networks: advantages and limitations." In: *Journal of computer-aided molecular design* 11.2, pp. 135–142.

Mao, Xudong et al. (2017). "Least Squares Generative Adversarial Networks." In: *The IEEE International Conference on Computer Vision (ICCV)*.

Mitchell, Herschel L, Pieter L Houtekamer, and Gérard Pellerin (2002). "Ensemble size, balance, and model-error representation in an ensemble Kalman filter." In: *Monthly weather review* 130.11, pp. 2791–2808.

Oshri, Barak and Nishith Khandwala (2016). *Predicting moves in chess using convolutional neural networks*.

Partalas, Ioannis, Grigorios Tsoumakas, and Ioannis P Vlahavas (2008). "Focused Ensemble Selection: A Diversity-Based Method for Greedy Ensemble Selection." In: *ECAI*, pp. 117–121.

Rokach, Lior (2010). "Ensemble-based classifiers." In: *Artificial Intelligence Review* 33.1-2, pp. 1–39.

Rowson, Jonathan (2005). "Chess for Zebras." In: *Gambit, London*.

Saeys, Yvan, Thomas Abeel, and Yves Van de Peer (2008). "Robust feature selection using ensemble feature selection techniques." In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, pp. 313–325.

Silver, David et al. (2017). "Mastering chess and shogi by self-play with a general reinforcement learning algorithm." In: *arXiv preprint arXiv:1712.01815*.

Srivastava, Nitish et al. (2014). "Dropout: a simple way to prevent neural networks from overfitting." In: *The Journal of Machine Learning Research* 15.1, pp. 1929–1958.

Zhang, Cha and Yunqian Ma (2012). *Ensemble machine learning: methods and applications*. Springer.

Zhou, Zhi-Hua, Jianxin Wu, and Wei Tang (2002). "Ensembling neural networks: many could be better than all." In: *Artificial intelligence* 137.1-2, pp. 239–263.