Deep Dive on Checkers Endgame Data

Jiuqi Wang
Department of Computing Science
University of Alberta
Edmonton, Canada
jiuqi@ualberta.ca

Martin Müller
Department of Computing Science
University of Alberta
Edmonton, Canada
mmueller@ualberta.ca

Jonathan Schaeffer
Department of Computing Science
University of Alberta
Edmonton, Canada
jonathan@ualberta.ca

***Abstract*—For games such as checkers and chess, large endgame databases/tablebases have been constructed to capture the perfect win/loss/draw value for positions near the end of the game. Such databases/tablebases can be used to enhance game-playing performance. However, this approach quickly runs into computational and storage resource limitations. An enticing alternative is to *learn* from such data and apply the learned evaluation to even larger data sets through transfer learning. This paper reports on research that uses deep learning to a) correctly learn a high percentage of checkers endgame positions; b) learn patterns that can be used for transfer learning; c) demonstrates that learning from a small sample of a large data set is an efficient way to compute a neural net evaluation that achieves most of the benefits; and d) shows that dynamically choosing between neural network prediction and using it in a one-ply search yields about 96% prediction accuracy.**

***Index Terms*—deep learning, transfer learning, endgames, tablebases, checkers, board games**

I. Introduction

Over the past decade, deep learning has achieved impressive successes for many real-world applications, including the development of high-performance game-playing programs. By using neural networks and deep learning principles, strong evaluation functions have been trained for complex games such as Go [1], Hex [2], and curling [3]. AlphaGo and its successors are the most famous examples [1].

The essential ingredient in deep learning is data. One area where extensive data repositories have been created is endgame databases (or tablebases). In games where the number of pieces is reduced over time, exhaustive enumeration of states and the computation of their game-theoretic (optimal) value is feasible. For example, for the game of checkers (8×8 draughts), the win/loss/draw value has been computed for all roughly 10^{13} positions with 10 or fewer pieces on the board [4], [5]. In chess, all seven-piece positions had been computed by 2018, and positions with eight pieces are under construction (up to 10^{16} positions [6]). For the game of Awari, the entire 10^{12} state search space was computed and stored [7].

What might a deep-learning-based algorithm learn from all this data? We start by training a neural net on the win/loss/draw values of database positions, to create a heuristic evaluation function. Based on experience with other domains, the algorithm should achieve generalizations by uncovering patterns in the data. Some of these patterns might reflect the game knowledge known to humans, while others could lead to completely new insights, as has happened in games such as backgammon and Go. The patterns might represent general knowledge about a game that would transfer to other data sets, such as checkers positions with more pieces, or with pieces in different locations.

This paper reports on our experiences in applying deep learning on data sets containing the game result of checkers positions. The insights gleaned from this work include:

1) A deep learning solution can be implemented with a fraction of the intellectual effort and computational time that was required for the custom solution. Our results indicate a tradeoff of accuracy and space versus effort and resources.
2) The amount of data available is enormous and would challenge any neural-net-based learning algorithm in terms of memory usage and computing resources required. However, we show that even learning from tiny subsets of the data is sufficient to achieve high performance.
3) Deep learning can discover patterns that are general enough to be applicable to positions that are significantly different from those in the training set. This is a strong demonstration of transfer learning in a complex domain.
4) The neural network can achieve good results as a predictor of a game position's outcome. Combining that with a one-ply search often gets better results. However, dynamically choosing between using the model or the search yields even better results.

II. Background

A. Checkers

The popular game of checkers (8×8 draughts) has simple rules that give rise to deep and subtle play. The human World Championship has been contested since 1840. In 1994, the Chinook program became the (non-human) World Champion [8]. The game was weakly solved to be a draw in 2007 [5].

Checkers is a two-player, zero sum, perfect information game. It is played on an 8×8 checkered board using only

We gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC), from Müller's Canada CIFAR AI chair, and from UAHJIC.

979-8-3503-2277-4/23/$31.00 ©2023 IEEE

the dark squares [9]. Each side has 12 checkers that can move forward diagonally. When a *checker* reaches the far end of the board, it becomes a *king* and can move diagonally forward and backward. Pieces are captured by jumping over them. If a capture move is present in a position, then it must be played. A player with no pieces on the board or no legal moves loses.

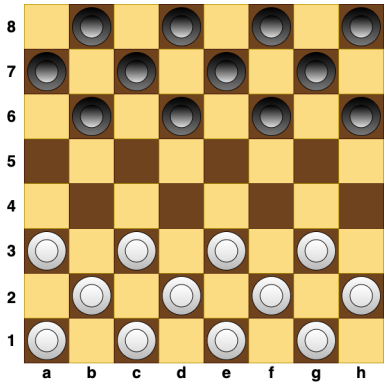

Fig. 1. Initial Checkers Position. Black to move.

Fig. 1 shows a checkerboard at the beginning of the game. Columns are labelled 'a' to 'h' and rows '1' to '8'. An example of a first move for Black is moving the *checker* on 'f6' to 'e5'.

B. Checkers Endgame Database

The CHINOOK endgame database (or tablebase) was computed using multiple computers for over a decade [4], [10]. The retrograde analysis algorithm iteratively solves all positions with one piece (black or white, *king* or *checker*) on the board, then computes all the two-piece positions, then three-piece, and so on. Capture moves take a position with N pieces on the board to one with less than N pieces, whose value has already been computed. Using this methodology, all positions with 2-10 pieces on the board have been computed, as summarized in Table I. The 120 positions with one piece on the board consist of 28 positions with one black *checker*, 28 positions with one white *checker*, 32 positions with one black *king*, and 32 positions with one white *king*.

TABLE I
CHECKERS ENDGAMES.

# of Pieces	# of Positions
1	120
2	6,972
3	261,224
4	7,092,774
5	148,688,232
6	2,503,611,964
7	34,779,531,480
8	406,309,208,481
9	4,048,627,642,976
10	34,778,882,769,216

The program computes only the win/loss/draw result (in chess, they also compute a distance-to-win metric). A custom algorithm compresses the 4×10^{13} position values into 2.5×10^{11} bytes. The data is used during games by CHINOOK, meaning the data organization has to support real-time decompression, doing thousands of database queries per second.

C. Deep Learning and Residual Neural Network

Designing an endgame database that is storage and lookup efficient is not a trivial task. It often requires designers to leverage application-specific knowledge and invent special tricks to make it work. The rise of deep learning [11] provides us with a promising tool, deep artificial neural networks (DNN), as an alternative approach to using large custom-designed data storage. Position-value lookup tables can be replaced by a function computed by a DNN, as they are universal function approximators [12]. In addition, they are particularly good at generalization, implying that one could harvest most of the benefits of an endgame database by training the neural network on a small subset of all possible states. In terms of access speed, modern deep-learning libraries and hardware are sophisticated in parallelizing the neural network inference – speeding up batched access significantly.

A residual neural network (ResNet) [13] is a deep learning architecture that achieves state-of-the-art performance in computer vision tasks such as image classification [13], object detection [14], and semantic segmentation [15], as well as in game-playing programs such as ALPHAZERO [16]. ResNet uses skip connections to avoid the vanishing gradient problem, where gradients become very small as they propagate through many layers. This allows the training of very deep neural networks.

III. METHOD

A. Board Representation

Inspired by ALPHAZERO [16], we represent each board position as a stack of binary planes where each plane represents a specific piece type. For each plane, the dimension is 8×8 – same as the checkerboard. The locations of pieces of that type are encoded as 1, while the rest are 0. With four types of pieces in checkers *(black king, white king, black checker, white checker)*, the representation for one board position is of dimension $(4, 8, 8)$. Fig. 2 shows an example of a board position and its representation.

B. Neural Network Architecture

We build a residual neural network, shown in Fig. 3, consisting of an input convolution layer, five residual blocks with two convolution layers each, and a classifier composed of two fully connected layers. We make this decision based on experiments with different numbers of residual blocks, fully-connected layers, channels of the convolution layers, and neurons in each fully-connected layer. The chosen structure represents a "sweet spot" between performance and computational cost.

We run experiments on a subset of the 2-5 piece data set to choose the best activation function from a number of candidates. We chose Mish [17] as our activation function for the neural network after comparing with rectified linear units

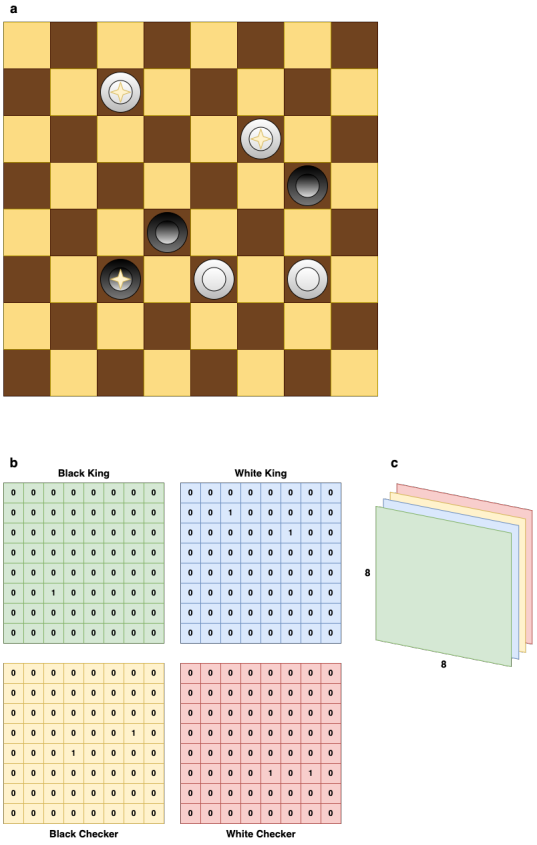

Fig. 2. Board Representation Example. **a**. The original board position. **b**. The values of each plane corresponding to the piece type. **c** The representation of the original position as a stack of planes.

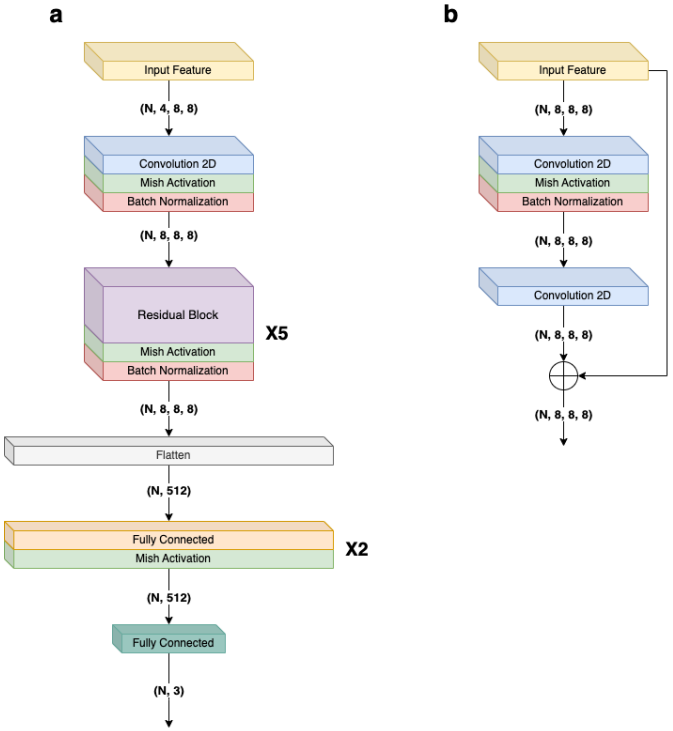

Fig. 3. Residual Neural Network Architecture. **a**. The overall structure. **b**. Structure of one residual block.

(ReLU) [18], SELU [19], and tanh [11]. The Mish activation function is defined as:

$$\text{Mish}(x) = x \tanh(\text{softplus}(x)) = x \tanh(\ln(1 + e^x)) \quad (1)$$

We find batch normalization [20] necessary to stabilize and speed up the convergence of the neural network. We run experiments with batch normalization applied to the whole neural network except for the output layer, to the residual blocks only, and without batch normalization. The result suggests that batch normalization on the residual blocks exclusively yields the most robust learning curves and best final accuracy.

C. Loss Function

Our loss function is the weighted cross entropy loss (CE) [21]. Given a predicted probability distribution $[p_1, p_2, p_3]$, a true distribution $[y_1, y_2, y_3]$, a weight vector $[w_1, w_2, w_3]$, and a weight exponent α, the loss is:

$$\text{Weighted CE Loss} = -\sum_{i=1}^{3} w_i^{\alpha} y_i \log(p_i) \quad (2)$$

Here we use vectors of dimension three because there are three possible outcomes in a game of checkers.

The weight vector is important to remedy the effect of unbalanced data sets. Let N_1, N_2, N_3 denote the number of positions for each outcome class in the data set, respectively. We calculate the weight vector as follows:

$$w_i = \frac{N_1 + N_2 + N_3}{N_i} \quad (3)$$

where we assume $0 < N_1, N_2, N_3$. Whenever $N_i = 0$, we let $w_i = 0$. α controls the degree of the weights – larger exponents emphasize the weighting.

D. Optimizer

For the optimizer, we use rectified Adam (RAdam) [22], a more robust variant of Adam [23]. We use the default values for the two hyperparameters of RAdam: the exponential decay rates of the first ($\beta_1 = 0.9$) and the second ($\beta_2 = 0.999$) moment of past gradients. In our testing, the performance of RAdam is comparable to or superior to the alternatives Adamax [23], Adagrad [24], and Adamw [25].

E. Hyperparameters

As shown in Table II, the hyperparameters include the learning rate, the mini-batch size, the decay rate for batch normalization, the weight exponent α for the loss function, and the decay rates β_1, β_2 for the optimizer. The large number of combinations prevents us from using an exhaustive grid search due to the computational cost. As a result, we have treated the hyperparameters independently and optimized them one by one.

TABLE II
HYPERPARAMETERS.

Primary	Learning rate = 0.0002 Mini-batch size = 1024
Batch normalization	Decay rate = 0.99
Loss function	α = 0.5
Optimizer	β_1 = 0.9 β_2 = 0.999

F. Technology

We use the deep learning framework JAX [26] and the neural network library Haiku [27] to implement and train our ResNet. Haiku transforms our neural network into a pure function compatible with JAX. Then we use JAX's automatic differentiation transformation to compute the gradients and JIT (just-in-time compilation) transformation to significantly accelerate forward and backward propagation – speeding up the training process. We implement our optimizer using Optax [28], a library of optimizers compatible with JAX. During training, we use TensorBoard [29] to visualize and monitor the progress. In addition, we save checkpoints every five epochs in the pickle format [30] to preserve the progress.

G. One-ply Search

Although the deep neural network is good at extracting patterns and generalizing them to unseen data, it is not perfect. Therefore, we propose to use a one-ply search algorithm to improve the prediction accuracy once the neural network is trained. For a position *p*, procedure *findChildren* generates the children of *p*. That is, each child is a position (standardized to Black-to-move) reachable from *p* after taking a move as Black. If *findChildren* returns nothing (Black has no legal moves), then the value is a loss. If the returned set is not empty, we then apply a *predict* function that predicts the outcome of each child. *predict* uses the neural network as the oracle except for those positions where one side has no pieces (an immediate loss). We identify the outcome of *p* through those of its children. If any child position is losing then *p* is a win for Black; If that does not hold, then it's a draw for Black in *p* if any child is a draw; Finally, if neither is present, then Black must be losing. Algorithm 1 describes the one-ply search algorithm that predicts the outcome of *p*.

Algorithm 1 One-ply Search

Require: p
  children $\leftarrow$ *findChildren*(p)
  if *children* is empty **then**
    return LOSS
  end if
  predictions $\leftarrow$ *predict*$($*children*$)$
  if LOSS is in *predictions* **then**
    return WIN
  else if DRAW is in *predictions* **then**
    return DRAW
  else
    return LOSS
  end if

H. Evaluation

To monitor the progress of training, we record the overall accuracy (% of correct predictions with respect to the total number of samples) and the recall (% of the correct predictions for outcome i with respect to the number of positions of outcome i) on the training set and the validation set.

For testing, we generate a normalized confusion matrix where each row contains the percentages of the true outcome and each column records the percentages of the prediction. Specifically, the (i, j)-entry of the confusion matrix denotes the percentage of the instances of class i predicted as class j over the test set. Furthermore, we produce a table that correlates the neural network performance with that of the search. It partitions the set of evaluation results into four disjoint categories – the neural network and search are both correct, only the neural network is correct, only the search is correct, and both are wrong.

IV. EXPERIMENTS

A. Sampling

The sampling experiment investigates if our ResNet can generalize to unseen data within the same databases by training it on a subset of all data. Our data set consists of 10% of the entire 2-6 piece databases and 0.1% of the 4 pieces vs. 3 pieces partition of the 7-piece database (referred to as the 7-piece data set). Beginning with a split of 70% for training, 10% for validation and 20% for testing, we shrink the training set by a factor of ten for each consecutive experiment, down to 0.1% of the original size. The size of the validation and test sets are kept constant. For the 2-7 piece data set, Table III summarizes the number of positions for each experiment. We train the neural networks until the validation accuracy stops improving.

Fig. 4 displays the normalized confusion matrices for the test set. The left column lists the neural network's confusion matrices, while the right column shows those of the corresponding one-ply search. Most errors occur for the drawing positions. Table IV displays the correlation tables. The generalization capability of the neural network is surprising – the model has achieved 95.46% accuracy and the search 93.30% accuracy on the test set when trained with a mere 10% of all data in the 2-7 piece database. Moreover, for the 0.1% case over 90% accuracy on a test set of roughly 58 million positions is achieved with only 202,229 samples. Furthermore, the one-ply search improves upon the ResNet in three out of the four tasks, demonstrating its capability to increase the robustness of the prediction when the neural network is imperfect.

B. Inter-database Generalization

Here we explore how the neural network trained on one database can generalize to different databases: First, we train our ResNet on a data set of 70% of the 4-piece database and

TABLE III
SUMMARY OF 2-7 PIECE SAMPLING DATA SETS.

	Baseline	10%	1%	0.1%
Training	202,238,136	20,223,804	2,022,368	202,229
Validation	28,891,154	-	-	-
Test	57,782,317	-	-	-

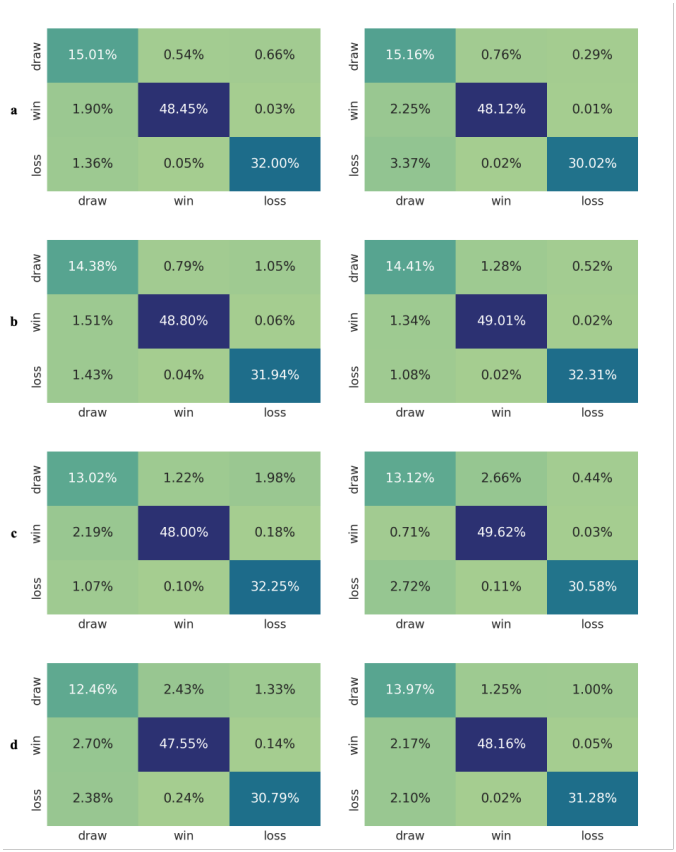

Fig. 4. Normalized Confusion Matrices of the Sampling Experiment. The rows are ground truths and the columns are predictions. Each cell is the percentage of positions over the test set. Left Column: neural network. Right Column: one-ply search. **a.** baseline **b.** 10% of baseline **c.** 1% of baseline **d.** 0.1% of baseline. Darker colours indicate dominant table values.

test it on 6% of the 6-piece database. Then we train another model on 70% of the 5-piece database and test it on 20% of the 7-piece database. Our choice to train the neural network on an even/odd-piece database and test it on another even/odd-piece database is based on the concern of material advantage. For all the positions in odd-piece databases, one of the players has a material advantage, resulting in a different distribution of outcomes from the even ones.

Table V summarizes the number of positions in each data set of each task. Fig. 5 displays the normalized confusion matrices of the neural network and the search for the aforementioned two tasks. They demonstrate that our ResNet is able to generalize to unseen databases to some degree. Nevertheless, both the neural network's and the one-ply search's performance degrade on the drawing positions, where they predict Draw to be Win

TABLE IV
SAMPLING EXPERIMENT CORRELATION TABLES.

Baseline	Model Correct	Model Incorrect	
Search Correct	52,321,877 (90.55%)	1,591,546 (2.75%)	93.30%
Search Incorrect	2,839,778 (4.91%)	1,029,116 (1.78%)	6.69%
	95.46%	4.53%	
10%	Model Correct	Model Incorrect	
Search Correct	53,309,487 (92.26%)	2,002,055 (3.46%)	95.72%
Search Incorrect	1,648,779 (2.85%)	821,996 (1.42%)	4.27%
	95.11%	4.88%	
1%	Model Correct	Model Incorrect	
Search Correct	51,183,270 (88.58%)	2,745,657 (4.75%)	93.33%
Search Incorrect	2,708,708 (4.69%)	1,144,682 (1.98%)	6.67%
	93.27%	6.73%	
0.1%	Model Correct	Model Incorrect	
Search Correct	50,455,079 (87.32%)	3,515,282 (6.08%)	93.40%
Search Incorrect	2,004,385 (3.47%)	1,807,571 (3.13%)	6.60%
	90.79%	9.21%	

or Loss, and vice-versa. Table VI shows the correlation tables on the 6-piece database and 7-piece database test sets. The model trained on the 4-piece database achieves an accuracy of approximately 82.68% on the 6-piece test set, while the search scores 83.28%. The model trained on the 5-piece database reaches an accuracy of 82.93% on the 7-piece test set, and the search 80.56%. These are good results, but not impressive.

TABLE V
SUMMARY OF INTER-DATABASE GENERALIZATION DATA SETS.

	4 Piece	5 Piece	6 Piece	7 Piece
Training	4,346,521	97,615,565	N/A	N/A
Validation	620,931	13,945,080	N/A	N/A
Test	N/A	N/A	153,162,235	38,110,526

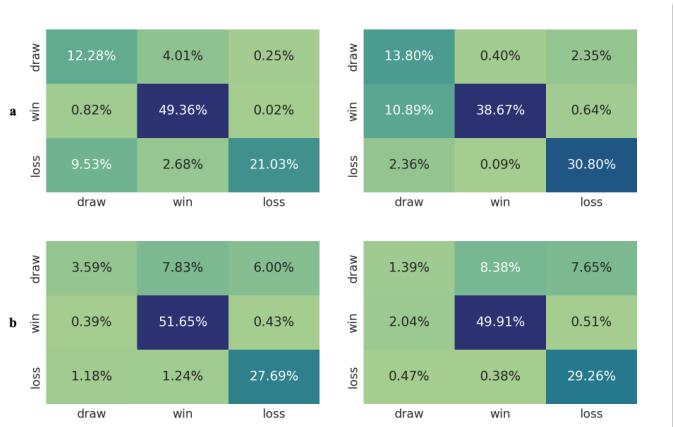

Fig. 5. Inter-database Generalization Normalized Confusion Matrices. The rows are ground truths and the columns are predictions. Each cell is the percentage of positions over the test set. Left Column: neural network. Right Column: one-ply search. **a.** train on the 4-piece database and test on the 6-piece database **b.** train on the 5-piece database and test on the 7-piece database. Darker colours indicate dominant table values.

TABLE VI
INTER-DATABASE GENERALIZATION EXPERIMENT CORRELATION TABLES.

4 Piece → 6 Piece	Model Correct	Model Incorrect	
Search Correct	107,115,684 (69.94%)	20,430,421 (13.34%)	83.28%
Search Incorrect	19,514,041 (12.74%)	6,102,089 (3.98%)	16.72%
	82.68%	17.32%	
5 Piece → 7 Piece	Model Correct	Model Incorrect	
Search Correct	29,301,935 (76.89%)	1,397,825 (3.67%)	80.56%
Search Incorrect	2,302,548 (6.04%)	5,108,218 (13.40%)	19.44%
	82.93%	17.07%	

C. Transfer Learning

The experiments so far demonstrate two main points: a network trained on one database can attain reasonable accuracy on another database, and a model trained on a tiny fraction of all data can generalize to the unseen data with high accuracy. This is evidence that the neural network is learning generally useful features that help to predict the outcome of checkers positions. Nevertheless, we know little about the nature of the learned features. To what extent are they specific to the databases that the model has seen? Does the network actually learn features that are applicable to other types of checkers positions? In a *transfer learning* experiment, we first pre-train two models from the 2-5 piece and 2-6 piece databases, then test their performance on the 7-piece database, as in the inter-database generalization experiment.

The first half of Table VII shows the correlation tables for this model. The network achieves an accuracy of 80.22% when pre-trained on 2-5 pieces and 75.36% for 2-6 piece pre-training (but 80.92% for using search). The drop in performance by adding the 6-piece positions is an artifact of checkers. The 5-piece database is dominated by wins – one side has a material advantage – which is a good match for the 7-piece (4 pieces versus 3) database. The 6-piece database is dominated by draws – balanced material – meaning the network is learning mostly on positions that are unlikely to help predict 7-piece positions.

Next, we freeze (stop gradient) the residual blocks of the two pre-trained networks, and fine-tune their classifier parts for only five epochs on the 7-piece database. After only one epoch, both pre-trained models achieve a validation accuracy above 94%! Over the remaining four epochs, they improve by another 0.5% (Fig. 6), to a final accuracy after transfer learning of 94.9% (2-5 pre-training) and 95.2% (2-6 pre-training) on the 7-piece database. The second half of Table VII displays the correlation tables of the fine-tuned models on the test set. This result strongly indicates that the knowledge gained on a small piece count database transfers to checkers positions with more pieces. The neural network clearly discovers some general checkers patterns, and does not merely memorize a position-outcome mapping.

As further evidence, Fig. 7 and Table VIII shows the results where, instead of pre-training, we freeze the residual blocks of a *randomly initialized* ResNet, then tune only the classifier for 60 epochs on the 7-piece database. This network (*random model*) has a validation accuracy below 90.5% after 60 epochs and a similar accuracy on the test set. This eliminates the possibility that the network achieves high accuracy simply by mapping the input to a higher-dimensional space.

Finally, as a reference we train a ResNet directly on the 7-piece database until convergence. Table IX shows results for the best checkpoint. The trained neural network is only marginally better than the transfer learning models in Table VII. The pre-trained models can learn features that are at least similarly strong as those learned directly on the target database.

TABLE VII
CORRELATION TABLES OF PRE-TRAINED NEURAL NETWORKS BEFORE AND AFTER FINE-TUNING.

2-5 Piece Pre-train	Model Correct	Model Incorrect	
Search Correct	29,313,893 (76.92%)	1,682,679 (4.42%)	81.34%
Search Incorrect	1,258,510 (3.30%)	5,855,444 (15.36%)	18.66%
	80.22%	19.78%	
2-6 Piece Pre-train	Model Correct	Model Incorrect	
Search Correct	25,383,593 (66.61%)	5,454,699 (14.31%)	80.92%
Search Incorrect	3,334,511 (8.75%)	3,937,723 (10.33%)	19.08%
	75.36%	24.64%	
2-5 Piece → 7 Piece	Model Correct	Model Incorrect	
Search Correct	34,695,990 (91.04%)	1,019,743 (2.68%)	93.72%
Search Incorrect	1,463,083 (3.84%)	931,710 (2.44%)	6.28%
	94.88%	5.12%	
2-6 Piece → 7 Piece	Model Correct	Model Incorrect	
Search Correct	34,326,432 (90.07%)	969,666 (2.54%)	92.61%
Search Incorrect	1,971,422 (5.17%)	843,006 (2.21%)	7.38%
	95.24%	4.75%	

TABLE VIII
RANDOM MODEL CORRELATION TABLE.

Random Model	Model Correct	Model Incorrect	
Search Correct	26,906,527 (70.60%)	1,926,654 (5.06%)	75.66%
Search Incorrect	7,515,449 (19.72%)	1,761,896 (4.62%)	24.34%
	90.32%	9.68%	

TABLE IX
REFERENCE MODEL CORRELATION TABLE.

Reference Model	Model Correct	Model Incorrect	
Search Correct	30,257,729 (79.39%)	816,636 (2.14%)	81.53%
Search Incorrect	6,268,606 (16.45%)	767,555 (2.01%)	18.46%
	95.84%	4.15%	

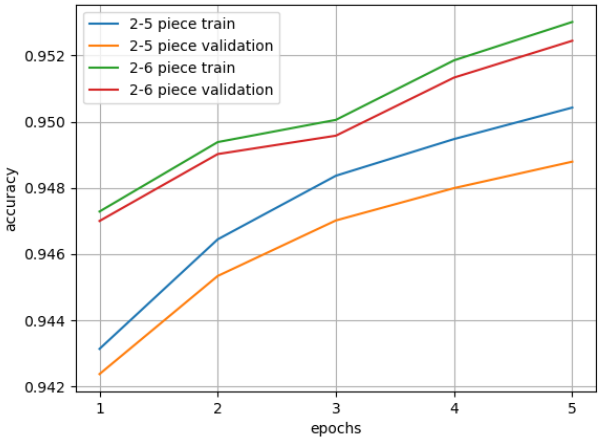

Fig. 6. Learning Curves of the Pre-trained Models Fine-tuned on the 7-piece Database.

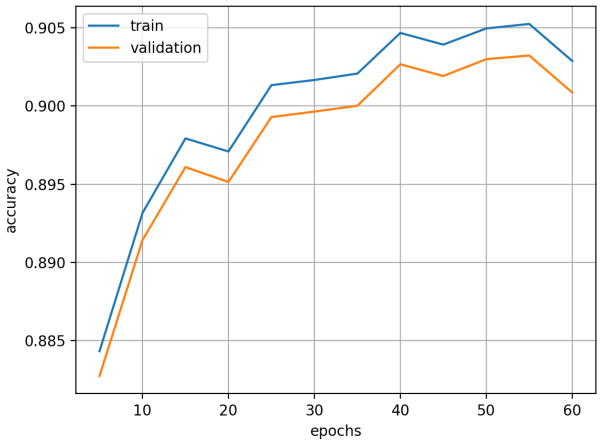

Fig. 7. Learning Curves of the (Stop Gradient) Randomly Initialized Model Training on the 7-piece Database.

D. Combining the Neural Network with One-Ply Search

The previous experiments demonstrate that the neural network inference and the one-ply search do not always agree. While in some tasks, the one-ply search can get an overall better accuracy at the end, in other cases, using the model alone is better, or the advantage is negligible. Therefore, using only one for inference implies a considerable opportunity cost. Is there a way to dynamically switch between the model and the search based on some criteria such that the ultimate classifier is more accurate than either?

As a context for our next experiment, we first define the highest probability from the distribution output by the neural network given a position as its confidence for that position. For instance, if the model outputs a distribution [0.80 (Draw), 0.15 (Win), 0.05 (Loss)] given an input board representation, its confidence for this board position is 80%.

We discover a correlation between the neural network's quality of prediction and confidence – the higher the confidence, the more likely the ResNet's inference is correct. We reuse the best baseline model from the sampling experiment, whose correlation table is in Table IV. Using this neural network, we generate Table X, a summary of the mean confidences of the correctly and incorrectly classified positions in the 2-7 piece validation set with respect to the outcome class. One can realize a significant difference between the mean confidence between the correct and incorrect predictions. The difference across the outcome class is not as striking. We also plot the distribution of the number of correct/incorrect positions over bins of confidence levels in Figure 8. The higher the confidence, the higher the ratio of the number of correct to incorrect inferences.

Exploiting this property of the neural network, we propose using a confidence threshold to dynamically switch between the neural network and the one-ply search for each position. That is – if the confidence of the neural network for a position is higher than the threshold, then we accept the model's decision. Otherwise, the result of a one-ply search is used.

Among thresholds 70%, 80% and 90%, 80% gives the best performance. Thus, we use 80% as the confidence threshold to test our algorithm on the 2-7 piece test set. Table XI displays the statistics of using the model only, the search only and the threshold algorithm that merges the two dynamically. Interestingly, the search-only method performs worse than the model-only method but complementing the model inference with search according to the threshold results in 55,441,847 correct mappings out of 57,782,317 positions in the test set – 280,192 (0.5%) more than model-only and 1,528,424 (2.65%) more than search-only. This evidence supports the effectiveness of our threshold algorithm.

TABLE X
MEAN CONFIDENCE TABLE.

Outcome	Mean Confidence Correct	Mean Confidence Incorrect
Draw	94.26%	74.86%
Win	98.54%	77.31%
Loss	97.71%	73.88%

TABLE XI
STATISTICS OF THE NEURAL NETWORK, THE ONE-PLY SEARCH, AND THE CONFIDENCE THRESHOLD METHOD.

Method	Correct	Total	Accuracy
Model-only	55,161,655	57,782,317	95.46%
Search-only	53,913,423	57,782,317	93.30%
Above threshold - Model used	52,782,198	53,899,336	
Below threshold - Search used	2,659,649	3,882,981	
Threshold	55,441,847	57,782,317	**95.95%**

V. CONCLUSIONS

This paper investigates using deep learning to approximate the value of positions for the game of checkers. We discuss

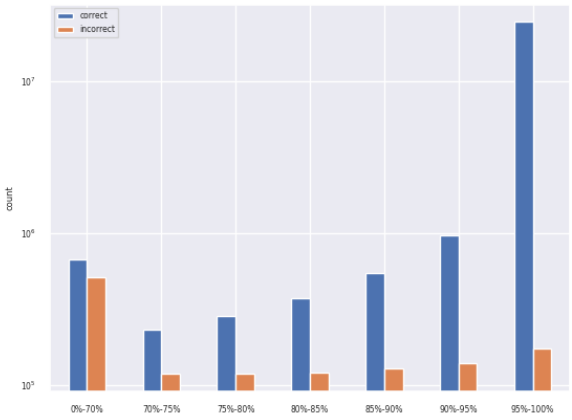

Fig. 8. Confidence Distribution.

three ways that neural networks can be used in this context: the network model, combining the model with a one-ply search, and dynamically choosing between the model and search. The latter technique is novel and holds great promise for exploiting the information generated by a neural network – the prediction and the confidence. Surprisingly, high performance can be achieved even with a tiny sample of positions, allowing the technology to be scaled up to even larger data sets.

For endgame databases, how do learned models compare to using a database/tablebase? There are many factors to consider, however the essence of the two approaches can be summarized in two key differences. First, the tablebase approach gives 100% accuracy, something that a learned model cannot do in general. If accuracy is essential (e.g., you are trying to prove the game-theoretic result of checkers), then this is the limiting factor for learning. But the tablebase approach does not generalize; the data is only applicable to the set of positions that have been computed. The learned approach creates patterns that can be applied to a broader set of positions – transfer learning. Ultimately, the success of AI depends upon programs being able to learn and apply that learning in situations that extend and/or differ from the learning environment. This work is a step in this direction.

REFERENCES

[1] D. Silver and *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[2] K. Takada, H. Iizuka, and M. Yamamoto, "Reinforcement learning for creating evaluation function using convolutional neural network in hex," in *Conference on Technologies and Applications of Artificial Intelligence*, 2017, pp. 96–201.

[3] Y.Han, Q. Zhou, and F. Duan, "A game strategy model in the digital curling system based on nfsp," *Complex and Intelligent Systems*, vol. 8, p. 1857–1863, 2022.

[4] J. Schaeffer, Y. Björnsson, N. Burch, R. Lake, P. Lu, and S. Sutphen, *Building the Checkers 10-Piece Endgame Databases*. Boston, MA: Springer US, 2004, pp. 193–210.

[5] J. Schaeffer, N. Burch, Y. Björnsson, A. Kishimoto, M. Müller, R. Lake, P. Lu, and S. Sutphen, "Checkers is solved," *Science*, vol. 317, no. 5844, pp. 1518–1522, 2007.

[6] "Syzygy endgame tablebases," 2023, syzygy-tables.info.

[7] J. W. Romein and H. E. Bal, "Awari is solved," *ICGA Journal*, vol. 25, no. 3, pp. 162–165, 2002.

[8] J. Schaeffer, *One Jump Ahead: Challenging Human Supremacy at Checkers*. Springer-Verlag, 1997, expanded edition 2009.

[9] "Checkers," 2023, en.wikipedia.org/wiki/Checkers.

[10] R. Lake, J. Schaeffer, and P. Lu, "Solving large retrograde analysis problems using a network of workstations," in *Advances in Computer Chess VII*, 1994, pp. 135–162.

[11] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[12] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 06 2016.

[14] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask r-cnn," in *IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[15] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *European Conference on Computer Vision (ECCV)*, September 2018.

[16] D. Silver and *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[17] D. Misra, "Mish: A self regularized non-monotonic neural activation function," *arXiv preprint arXiv:1908.08681*, 2019.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, p. 84–90, 5 2017.

[19] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," *Advances in neural information processing systems*, vol. 30, 2017.

[20] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15. JMLR.org, 2015, pp. 448–456.

[21] Y. S. Aurelio, G. M. de Almeida, C. L. de Castro, and A. P. Braga, "Learning from imbalanced data sets with weighted cross-entropy function," *Neural Processing Letters*, vol. 50, no. 2, pp. 1937–1949, 2019.

[22] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, "On the variance of the adaptive learning rate and beyond," in *International Conference on Learning Representations*, 2020.

[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[24] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of Machine Learning Research*, vol. 12, no. 7, 2011.

[25] I. Loshchilov and F. Hutter, "Fixing weight decay regularization in adam," *CoRR*, vol. abs/1711.05101, 2017.

[26] J. Bradbury and *et al.*, "JAX: Composable transformations of Python+NumPy programs," 2018. [Online]. Available: http://github.com/google/jax

[27] T. Hennigan, T. Cai, T. Norman, and I. Babuschkin, "Haiku: Sonnet for JAX," 2020. [Online]. Available: http://github.com/deepmind/dm-haiku

[28] I. Babuschkin and *et al.*, "The DeepMind JAX Ecosystem," 2020. [Online]. Available: http://github.com/deepmind

[29] M. Abadi and *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," 2015. [Online]. Available: http://download.tensorflow.org/paper/whitepaper2015.pdf

[30] G. Van Rossum, *The Python Library Reference, release 3.11.2*. Python Software Foundation, 2023.