Predicting Moves in Chess using Convolutional Neural Networks

Barak Oshri
Stanford University
boshri@stanford.edu

Nishith Khandwala
Stanford University
nishith@stanford.edu

Abstract

We used a three layer Convolutional Neural Network (CNN) to make move predictions in chess. The task was defined as a two-part classification problem: a piece-selector CNN is trained to score which white pieces should be made to move, and move-selector CNNs for each piece produce scores for where it should be moved. This approach reduced the intractable class space in chess by a square root.

The networks were trained using 20,000 games consisting of 245,000 moves made by players with an ELO rating higher than 2000 from the Free Internet Chess Server. The piece-selector network was trained on all of these moves, and the move-selector networks trained on all moves made by the respective piece. Black moves were trained on by using a data augmentation to frame it as a move made by the white side.

The networks were validated against a dataset 20% the size of the training data. Our best model for the piece selector network produced a validation accuracy of 38.3%, and the move-selector networks for the pawn, rook, knight, bishop, queen, and king performed at 52.20%, 29.25%, 56.15%, 40.54%, 26.52% and 47.29%. The success of the convolutions in our model are reflected in how pieces that move locally perform better than those that move globally. The network was played as an AI against the Sunfish Chess Engine, drawing with 26 games out of 100 and losing the rest.

We recommend that convolution layers in chess deep learning approaches are useful in pattern recognition of small, local tactics and that this approach should be trained on and composed with evaluation functions for smarter overall play.

1. Introduction

Convolutional neural networks have been shown to be successful in various longstanding AI challenges that can be reduced to classification problems. Clark and Storkey have reported a 44.4% accuracy in predicting professional moves in Go, a game known for its abstract logical reasoning that experts often describe as being motivated by faithful intuition [2]. This is an exciting result, indicating that CNNs trained with appropriate architectures and a valid dataset can catch up with much of the experience-based human reasoning in complex logical tasks.

The success of CNN-Go can be attributed to smooth arrangements of positions that are approximately continuous through and between games. Additionally, since each move in Go adds a single piece to the board, essentially flipping the value of one pixel, the difference in board representations before and after a move is smooth, constant, and almost always linked to the important patterns observed by the network, which contributes to the consistency of Go classification algorithms.

1.1. Challenges of CNN Approaches to Chess

Unlike Go, chess is more motivated by heuristics of many kinds of pieces in diverse and short-term tactics that build into longer-term strategies. This is essentially because the advantage of a position is always rooted in the relationships between the rules of the pieces. This makes pattern identification of chess more reliant on understanding how the nuanced and specific positioning of pieces leads to their advantages. Chess boards also do not shift smoothly, as each move causes a shift in two pixels in the 8×8 board, a factor of $1/32$, which is more significant than a change in one pixel out of a 19×19 board ($1/361$) in Go.

For these reasons, it is less clear that the logical patterns in chess can be described in activation layers of a neural network. Important concepts such as defending or pawn chains are often times best expressed by heuristic methods and logic information systems, such as ”if pawn diagonally behind piece” or ”if bishop on central diagonal” conditionals. That is, chess understanding is more characterized by domain knowledge. Therefore, we already predict that ConvChess, as we termed our intelligence, should be supported by and combined with other methods and approaches in chess intelligence to produce maximal results, such as lookahead and coupling with an evaluation function.

For example, Sebastiun Thrun's *NeuroChess* learns an evaluation function using domain-specific knowledge in an *explanation-based neural network learning* model that maps temporal dependencies between a chess board and the corresponding board two moves later [4]. The changes to the board are used to bias the evaluation function to estimate the slope of the function given any move. This approach, therefore, uses move-predictions as domain knowledge to influence an on-model evaluation function.

1.2. Chess Reasoning as Pattern Recognition

Indeed, the approach of directly classifying moves is a reflexive, off-model approach that makes moves without understanding *why* those moves are made but instead what patterns inspire moves to be made given the situation. However, it uses a precomputed model to predict chess moves in very little time and with high accuracy.

Traditional approaches to chess intelligences are comprised of two parts: an evaluation function and a search function. The evaluation function scores a board in a relative assessment of how likely it is to lead to a win, and the search function is a lookahead implementing minimax using the evaluation function. Since chess is a finite state hence solvable game, this approach is first limited by computational needs and second by the success of the evaluation function. Leaps in chess AIs therefore improve on either of these limitations, by cleverly navigating the search space or incorporating chess principles into the board evaluation.

It is thus not surprising that machine learning approaches to chess capitalized on the challenge of producing a successful evaluation function by attempting pattern recognition on data points labeled with a 1 if white is the winning side and 0 if white is the losing side [3]. The data then is just considered as "samples" of boards seen in real plays that led to an eventual outcome, with the hope that optimal moves were played ahead and that the advantage of the board at that state manifested in the correct turnout of the game (ie the player continues to behave optimally). Although such an approach is principally correct, it is severely compromised by the weak labeling of the data set, and little can be done to overcome this reward system.

1.3. Convolutional Neural Networks in Chess

Critics of CNNs argue that neural networks cannot adequately explain such tactical advantages because the forms of these conditions are too global across the board and affected by extraneous variables [1]. However, we claim that these shortcomings are mostly a result of the ill-formed task of training to binary labels of win and loss. Such an algorithm labors at developing an oracle intuition for whether small local patterns correspond to a winning or losing state, the association of which is likely weak in most chess situations.

However, the task of using small, local features to make chess moves is different and situated well for the task of a CNN. Such features are activated on arrangements that serve as heuristics and intuitive patterns made by real players. For this reason, we eschewed the one-sided labeling of chess boards and modelled incremental tactical choices by labeling each board state with the move made from it. This philosophy better captures the nature of chess moves in an experienced chess game: almost every move played by a high-ELO chess game is a reasonable move, especially when averaged over the entire training set.

In this approach, the patterns that matter in the raw image are those that encourage human actors to make certain moves. The cost of increasing the information content of our labels is that the class space has significantly grown. Also interesting to note is that classifying for the next best move acts as a precomputation of the lookahead for further board states involved in the search function, as the needs for the search are now met with an understanding of which move was played for a given board representation. A lookahead in this model is now relevant to making consistent strategic plans as opposed to stronger evaluative claims of a board using minimax.

1.4. Approach

The greatest challenge to this approach to training is that the space of possible moves is unwieldy large. The class space for the next move in Go is always some subset of $19 \times 19 = 361$ possible positions; but in chess, although there are generally an average of fifty possible moves given any position, there are $(8 \times 8)^2 = 4096$ possible classes which a CNN would have to score for.

For this reason, we divided the classification challenge into two parts using a novel approach (a very novel approach...). The first part is training a CNN to predict which coordinate a piece needs to be moved *out* of. This captures the notion of *escape* when a piece is under attack or the king needs to move. The network takes as input a board representation and then outputs a probability distribution over the grid for how desirable it is for a piece to leave a square, with all squares without pieces or with opponent pieces clipped to 0. The second part is training *six* other CNNs to encode which coordinates would be advantageous to put each of the six possible piece on. For example, this includes a bishop neural network that takes as input a chess board and outputs a probability distribution on the whole grid for how desirable it is to have a bishop in each square, with all squares that the bishop cannot move to given that board state clipped to 0.

We obtain the optimal move by composing the piece selector network (pCNN) with all move selector networks (mCNNs) by multiplying the values in the pCNN by the highest value in the corresponding mCNN and taking the argmax over the entire composition to obtain two coordinates for which piece is moved off the board and where it is placed. The pCNN clips to zero probabilities at positions that have no friendly piece and the mCNN clips to zero probabilities at positions where the move with the current piece is illegal.

Note that each CNN now only has a class size of 64 (a square root of the original absurdity!) for a cost of doubling the training time, since it only has to decide on a single position. Interestingly, though, this approach captures much of the human intuition behind chess thinking: sometimes a move is made in the spirit of protecting a piece under attack (the piece selector network outputting high probabilities) and other times in the spirit of seeing a positional advantage (the move selector network outputting high probabilities). The downside to this approach is that highly specific move combinations between both nets are not learnt, although we deemed that there are enough representations to learn in each net that are sufficiently hard.

Since the image type of this project is unique to an image classification task, we had few baselines for how varying architectures fit on the situation. Our experimentation involved starting with small models and increasing their sizes to find its limits and experiment with potential factors for expansion.

2. Technical Approach

2.1. Data structures

A chess board is represented in our model as an $8 \times 8 \times 6$ image, with two dimensions covering the chess board and six channels corresponding to each possible piece; a different representation is used for each piece since its value is of discrete worth - the difference between two pieces is not continuous and so cannot be measured on the same channel. Also we opted to use one layer for both colors, +1 to denote friendly piece and -1 to denote opponent pieces; using 12 layers to represent each piece of both colors would make the data unnecessarily sparse.

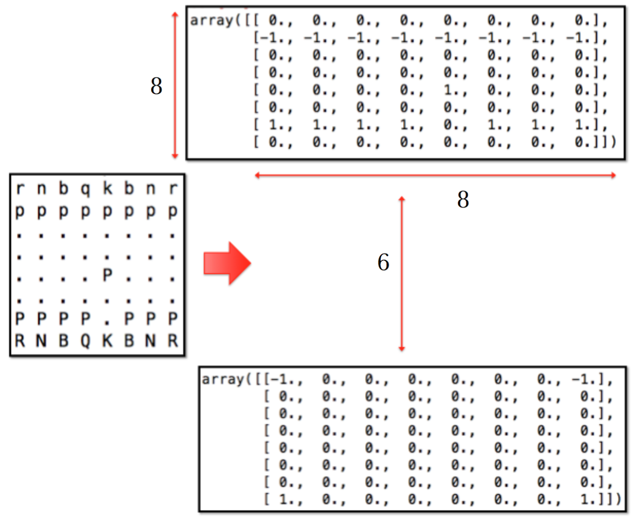


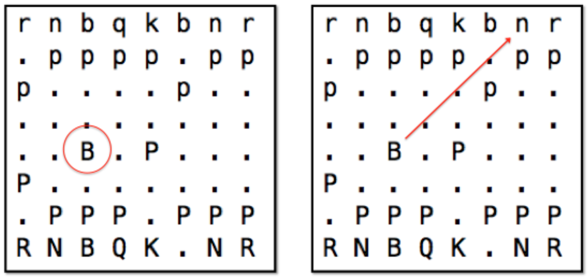
Figure 1. Conversion of Board to Input Image

There were seven datasets that the networks were trained on. The first is the move selector set, with images labelled on coordinates that a piece was moved from. The last six are piece selector sets, with images of boards labelled on coordinates where the class piece moved to. Note that the move selector dataset is the largest and that the sizes of the piece selector datasets sum up to the size of the move selector's since each each move is associated with only one of the move selector networks while each move associated with the piece selector.

Although it does not matter which color the algorithm is training from, we must ensure that the side of the board the algorithm is training from is the same. For this reason, we performed a data augmentation so that the algorithm is able to train on both white and black moves: when the algorithm trains from black we reflect the board vertically and horizontally (including the label associated with the board) to preserve the orientation that white plays from even when encoding black's move so that the data point "appears" like a white move. Using this data augmentation the net is thus also able to play on Black's side when using in real-time.

2.2. CNN architecture

All seven networks take as input the chess representation described above and output an 8×8 probability distribution representing the scores of each position. We use a three layer convolutional neural network of the form [conv-relu]-[affine]x2-softmax with 32 and 128 features. We found that training on larger networks (particularly ones with more convolutional layers) became increasingly impossible, with a grid-search on a five layer network leading to no learn-



Piece Selector Move Selector

Figure 2. Model Overview

ing in any of the hyperparameters and a three layer network leading to minimal differences. We suspect that this is because too many parameters are used on small and sparse data that becomes hard to train on the higher layers. We emphasize the need for two affine layers at the top so that low level features can be accompanied by stronger global logic evaluations, and we suspect that the model is saturated on convolutional layers and would be improved most by including further affine layers.

We tested the networks with both relu and tanh activation functions. The reason for incorporating the tanh function is that we suspected that because relu discriminates against negative values it might harm the signal of the enemy pieces which are initially represented as negatives. However, the relu performed marginally better than tanh in all tests and so it was used in our final model.

2.3. Preprocessing

The FICS dataset is comprised of chess games in PGN format, which enlists every move played by both players in Algebraic Chess Notation (SAN). In order for this dataset to be relevant to our project, the SAN notation had to be converted to two coordinates representing the position a piece was moved from and the position a piece was moved to. These moves were then played out on a chess board to obtain the board representation at each position, encoded in the data structure described above. The two sets of labels used are then the coordinate a piece was moved out from and the coordinate the piece was moved into.

2.4. Training

2.4.1 Sampling

We train the networks for 245,000 moves over 20,000 games, with the piece selector CNN trained on every piece and each move selector network training on every move made by the respective piece of the network. The training data is assumed to sample a wide range of chess situations and strategies.

2.4.2 Pooling

We do not use pooling layers to preserve as much data as possible during training. Pooling to learn transformations is also not relevant as any transformation on the chess image makes a huge impact on the outcome of the board.

2.4.3 Weight initialization

Crucially, the weights had to be initialized to very low values to match the small values of the data made up of -1, 0 and 1 in the input layer. When training at first with high initializations, the input data had no bearing on the final class scores with their overall effect on the forward propagation depressed by the high weights. We cross-validated the order of magnitude of the initializations and determined that 10^{-7} is the optimal initialization for the parameters in the first layer, using *larger* initializations in the deeper layers of 10^{-6} when the data is less sparse and sensitive to bad initial forward propagations.

2.4.4 Regularization

We use a minimal amount of regularization. Encouraging the smoothing of parameters does not immediately appear to be applicable to this task because chess exhibits more entropy than image recognition; however, we found that some regularization initially increases the performance.

2.4.5 Dropout

As in regularization, dropout was deemed to not conform well to this task, and this was supported in our results. The image is small enough that all the features must be interacting with each other on some level such that dropping out some of the activations is bound to eliminate crucial patterns in the forward propagation. Also, since the data is already sparse, dropout systematically removes pieces of data in training that are much needed in this task.

2.4.6 Loss Function

A softmax loss function was used so that moves could be interpreted as being made with a probability as opposed to a score with an arbitrary scale. This is especially important when composing the piece-selector with the move-selector for two arbitrary scales cannot be composed together in any meaningful fashion. Probabilities as output are also useful in interpreting second- and third-best moves to observe the algorithm's other intended strategies.

2.4.7 Parameter Update

We used the RMSProp parameter update to emphasize the concept of "confidence" in training. Since the RMSProp update strength is influenced by a running average of magnitudes of recent gradients, it allows repeated moves to influence the model more strongly. This also encourages the final distribution of scores to have a higher standard deviation which reflects a greater confidence in a few moves, which is ideal (moves that are made less consistently among players - idiosyncrasies - should be filtered out and have less influence on the training).

2.4.8 Software

The data was trained on a quad-core CPU using a custom library designed by instructors and TAs of a class taught at Stanford. An optimized library was not deemed necessary because the training time on the data was not significant.

2.5. Testing

Testing in this project was done in validation and in pitting the algorithm real-time against a Sunfish chess engine.

2.5.1 Validation

In the validation setting, we compared the move prediction accuracies with the real-life moves. When predicting correctly, it shows that the model has a sophisticated understanding of how players are making moves. When it does not predict the real-life moves correctly it is not necessarily indicative that it is making less-than-ideal moves or bad moves. Firstly, the validation accuracy does not measure how "bad" the incorrect predictions were. This is alikened to a hierarchical classifier that predicts a "specific" class wrongly but a higher level class correctly (like "cat" without designating what kind of cat). Such a "hierarchical" approach to measuring how close a move is to the labelled outcome is impossible to make as there is no metric for "how far" the success of moves are from each other.

Secondly, in comparing predictions with other players' moves, it doesn't account for differences in strategies among the players. That is, by computing averages of the players, the network itself learns a "mixed and averaged" playstyle of the players it learnt from. This represents one of the broader issues with this algorithm: real-player moves are sometimes made with a playout in mind for several moves, but the network is trained on only a one-layer lookahead. The network thus performs best in validation in situations with unambiguous strategies. A more advanced implementation of the model would implement a "bigram" model where it learns for two (or more) moves at once for greater lookahead.

2.5.2 Against Computer

The other testing mode is playing the algorithm against a computer intelligence, such as the Sunfish chess engine. To make the algorithm playable, the probability distribution over the 8×8 board outputted from the networks need to be clipped to represent valid choices of pieces in the piece-selector network and legal moves in the move-selector. The piece-selector network is clipped by searching for all the white pieces in the image and the legal moves are filtered using the in-build chess logic algorithms in the Python-Chess module.

3. Experiment

Our best model was found by doing a grid-search over the parameters of learning-rate, regularization, and weight initialization with fine tuning afterwards. We found that models in the grid-search either performed really poorly (got stuck at 5% validation) or overtrained without generalization. Successful models were in the few and they were very sensitive to tuning around those values.

The parameters of the best model were a regularization of 0.0001, learning rate of 0.0015, batch size of 250, a learning rate decay of 0.999 on RMSprop with no dropout trained in 15 epochs.

3.1. Validation

3.1.1 Results

The best result obtained with the piece selector was 38.30%. The results for the move selector varied. Pieces with local movements performed significantly better than pieces with global movement around the chess board. The pawn, knight, and king performed at accuracies of 52.20%, 56.15%, 47.29% respectively. Pieces with global movements, the queen, rook, and bishop, were predicted with significantly less accuracy at 26.53%, 26.25%, and 40.54% respectively.

Table 1. Piece Selection Accuracy

Metric	Accuracy
Piece Selection	38.30%

Table 2. Move Prediction Accuracy

Move Selector	Accuracy
Pawn	52.20%
Rook	29.25%
Knight	56.15%
Bishop	40.54%
Queen	26.53%
King	47.29%

This can be attributed to both the success of the convolution layers in producing relevant features allowing the algorithm to evaluate the local situation around the piece and the fact that local pieces have fewer positions they can move to. The first statement is likely. Removing the second convolution layer reduces the accuracy of the local pieces to between 20% to 34% but does not affect the global pieces. Conversely, removing the second affine layer does similarly by decreasing the accuracy of the global pieces to between 15% and 21% but the local pieces to 32% and 38%.

3.1.2 Clipping

The accuracies above do not reflect the most faithful prediction of the validation accuracies of the model because they do not clip to making sure a valid piece is chosen or legal move is made. This is because we wanted to test our network completely off-model and with no external influence at all on the game rules or parameters - as purely a classification task.

Clipping to force the algorithm to choose a correct piece or legal move of course can only increase the validation accuracy. Naturally, we found that when we clipped the piece-selector network the algorithm predicted correctly three times as much in the first several epochs with the effects levelling off by the 3rd epoch and completely converging by the 4th or 5th. The non-clipped accuracies unfailingly converged with the clipped accuracies for any dataset trained on, which indicates that the network learns to classify based on the rules of the game. This observation has made convergence time between clipped rates and non-clipped rates a useful metric and criterion for deciding when training has

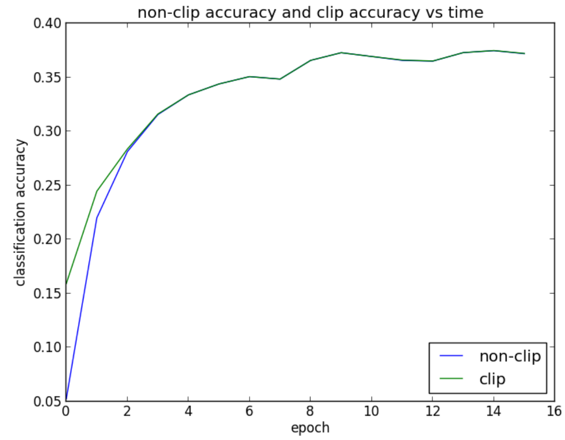


Figure 3. Clipping Vs. Non-Clipping of Illegal Moves

finished (when the accuracies are equivalent for five epochs in a row).

3.1.3 Trade-off between Move Legality and Optimality

The success of the ability of the network to classify into legal chess moves leads to an interesting question: How does the network classify to both move legality *and* move optimality? Since it unfailingly makes moves that are legal, is there some trade-off between a prediction that is legal and one that is advantageous? That is, since the move-selector network is without context of which piece has already been "picked" from the piece-selector, the network must also have features and architectures dedicated to ensuring that a legal move is chosen, the activations of which may "counteract against" those which compute the optimal move. Further work on this could develop approaches to perform the classification task with rule-based understanding of the game (such as having a loss function that incorporates rule-based violations), so that the network doesn't have to devote computational resources to ensuring move validity and so that it trains purely on move optimality. This relates to the idea at the beginning of this paper that the dynamics of chess are often the result of the interactions of the rules of the pieces - it seems like a deep learning model doing move prediction emulates this philosophy very faithfully.

3.1.4 Performance, Saturation, and Limitations

Dropout, as predicted, produced no positive measurable changes in the performance. Its linear decrease explains that

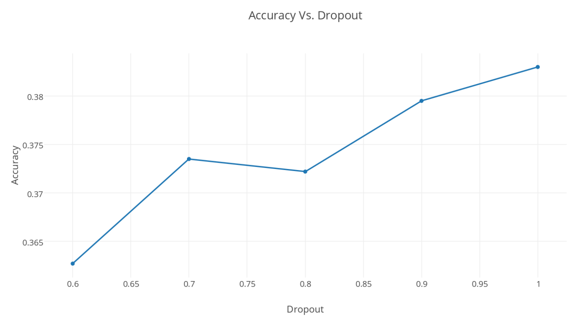


Figure 4. Accuracy Vs. Dropout Rate

all the data in the board is linearly important in contributing to the overall accuracies. Piece dependencies ensure that as much information about the activations and the original input are needed for better predictions.

Table 3. Dropout Rate Vs. Accuracy

Dropout Rate	Accuracy
0.6	0.362700
0.7	0.373500
0.8	0.372200
0.9	0.379500
1.0	0.383000

One of the most surprising results in our research is that increasing the size of the network with both affine layers and convolution layers, when the resulting net was trainable, did not lead to any increases in performance. Pooling did not decrease the performance as much as we thought it would, indicating that perhaps only a minority of the features are most influential on performance. This was confirmed by the fact that doubling the filters used in our best model produced the same results. The most likely reason for this happening is that the extra features aren't contributing new information to the computation which could mean that the features learnt are repeating themselves and that there is a saturation point for how many local features can be observed. Even increasing the size of the depth of field to 5×5 (making the convolution layers more fully connected) and changing the size of the local features reduced the performance accuracy by 3.2%.

So unlike traditional image recognition problems, we predict that improving the results of this experiment are not going to be made by expanding the CNN but by interweaving chess-based rules that guide and influence the training.

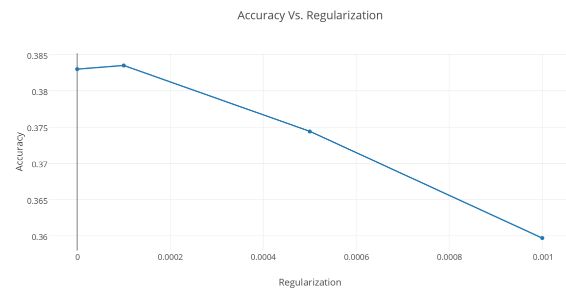


Figure 5. Accuracy Vs. Regularization

3.2. Against a Computer

The AI was played against the Sunfish chess engine in 100 games. 26 of the games drew and the rest were lost to the engine.

The AI makes good judgments with piece captures. This is likely because piece captures involve pawn defenses that are strongly activated when a capture is made with them. There is evidence that the AI picked up on basic attacking strategies. For example, in games played against the researchers, the AI was noted to frequently attempt to "fork" the rook and the queen, but it always failed to see that a king, knight, or bishop was defending that square. These "single piece mishaps" are made frequently, where a 1-piece difference from other examples seen in the past make an enormous difference on the dynamics of the move choice; that is, the AI makes a move choice based on generalities of positions as opposed to specific combinations. We believe that if the network is trained with an evaluation function that would strongly criticize such move choices it would play significantly better.

The fact that the AI draws 26% of the time and loses the rest is not disheartening. A move-prediction AI in any situation cannot be expected to understand faultlessly the dynamic of every situation; this task is more suited to evaluation AIs whose explicit purpose is to generalize to new situations. The sheer number of combinations in endgame and even more sparse representation of the image (once pieces are captured) mean the AI is troubled at making choices. In fact, the games where the AI draws happen mostly in the middle game in crowded positions where the convolutions reveal the complex patterns needed to push the opponent into a draw.

3.3. Conclusion and further work

By a CNN learning on the end result of chess thinking, it is essentially precomputing an evaluation function to make directly on a given situation. Training on this approach while useful for quick and moderately successful training of a chess algorithm, falls short of teaching innovation and creativity, and convolutional layers, as we have demonstrated in this paper, are adept at characterizing small, local objectives, such as short-term piece captures, local piece movement, and creating draw scenarios. In chess, creative move choices come all too difficulty with a highly adept and logical eye for how the position of one piece or the dynamic between two pieces on the board completely reshapes the environment, and a chess CNN is not made for predicting moves in these kinds of situations. A further study in this topic could examine the use of an evaluation function to bias the move selector or compose the two approaches.

References

[1] Erik Bernhardsson. Deep learning for... chess. 2014.

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

[3] Johannes Frnkranz. Machine learning in computer chess: The next generation. *International Computer Chess Association Journal*, 19:147–161, 1996.

[4] Sebastian Thrun. Learning to play the game of chess. *Advances in neural information processing systems*, 7, 1995.

The codebase of this project is publicly available at **http://github.com/BarakOshri/ConvChess**.