Learning to Play Othello with Deep Neural Networks

Paweł Liskowski, Wojciech Jaśkowski, Krzysztof Krawiec

arXiv:1711.06583v1 [cs.AI] 17 Nov 2017

Abstract—**Achieving superhuman playing level by AlphaGo corroborated the capabilities of convolutional neural architectures (CNNs) for capturing complex spatial patterns. This result was to a great extent due to several analogies between Go board states and 2D images CNNs have been designed for, in particular translational invariance and a relatively large board. In this paper, we verify whether CNN-based move predictors prove effective for Othello, a game with significantly different characteristics, including a much smaller board size and complete lack of translational invariance. We compare several CNN architectures and board encodings, augment them with state-of-the-art extensions, train on an extensive database of experts' moves, and examine them with respect to move prediction accuracy and playing strength. The empirical evaluation confirms high capabilities of neural move predictors and suggests a strong correlation between prediction accuracy and playing strength. The best CNNs not only surpass all other 1-ply Othello players proposed to date but defeat (2-ply) Edax, the best open-source Othello player.**

Index Terms—**deep learning, convolutional neural networks, Othello**

I. INTRODUCTION

Neural networks, particularly the deep and convolutional ones (CNNs), excel at recognizing virtually all kinds of patterns [32]. They naturally generalize to arbitrary dimensionality, which makes them suitable for analyzing raster images, time series, video sequences, and 3D volumetric data in medical imaging [38]. These properties make CNNs powerful for solving a wide range of challenging tasks at unprecedented performance level, ranging from image classification [12], to object localization [10], object detection [27], image segmentation [23], and even visual reinforcement learning in 3D environments [19].

The structural analogy between 2D raster images and board states in games is natural, so no wonder that it has been exploited within the neural paradigm in the past (e.g., [42]) . In most cases however, neural nets served there as *board evaluation functions* employed for searching game trees. It is only recently that massive computing and efficient learning algorithms for deep CNNs enabled their use as direct *move predictors* capable of achieving the level of play in the most challenging games previously thought to be exclusive only to human players. This gave rise to a line of Go-playing programs, including the paramount achievement of AlphaGo, the superhuman-level Go-playing program [33].

The authors are with the Institute of Computing Science, Poznan University of Technology, Poznan, Poland. W. Jaśkowski did this work while staying at IDSIA Dalle Molle Institute for Artificial Intelligence Research, Manno TI, Switzerland; e-mails: {pliskowski, wjaskowski, kkrawiec}@cs.put.poznan.pl.

CNNs' success in Go was possible because it has been known for a long time that Go players, rather than performing a "mental" tree search of future game states, rely heavily on detecting patterns in board states. This, together with a high branching factor of 250, and a large board renders tree search approaches ineffective for Go.

Here we ask whether CNNs have a practical potential for games of a small branching factor and much smaller board size, for which the minimax-style tree search algorithms perform well and achieve human-level performance. To this aim, we consider learning CNN-based move predictors for the game of Othello, a long-standing benchmark for AI [18]. This problem not only diverges from Go, but is also very different from analysis of images in computer vision: the input rasters are tiny (8×8 ternary board states), free from noise and distortions typical for real-world settings (sensor noise, lighting, perspective, etc.), and the desirable invariances involve axial symmetries rather than translation and scaling. Last but not least, every single piece on the board matters and influences the decision. CNNs have been hardly ever evaluated in such settings.

Our contributions include i) an experimental study of different CNN-based architectures of deep neural networks for Othello; ii) state-of-the-art move prediction accuracy on the French Othello league game dataset WThor; iii) state-of-the-art 0-ply policy for Othello (no look ahead) that significantly improves over the previous approaches; iv) an in-depth analysis of the characteristics of trained policies, including the relationship between move prediction accuracy and winning odds, strength and weaknesses at particular game stages, and confrontation with opponents that employ game tree search at different depths. The gathered evidence suggests that several of the proposed neural architectures are best-to-date move predictors for Othello.

II. RELATED WORK

CNNs have been introduced by Fukushima in Neocognitron [9] and occasionally used for selected image analysis tasks for the next two decades [22]. The attempts of applying them to broader classes of images remained however largely unsuccessful. The more recent advent of deep learning brought substantial conceptual breakthroughs, enabling training of deeper and more complex networks. This, combined with cheap computing power offered by GPUs revolutionized machine perception and allowed achieving unprecedented classification accuracy, exemplified by the chain of ever-improving neural architectures assessed on the ImageNet database [21], [12].

As signaled in the Introduction, the structural analogy between natural images and game boards is quite obvious for humans and as such could not remain unnoticed for long. This led to early attempts of applying CNNs to board games [8]. Soon it became clear that the deep learning paradigm may be powerful enough to be used together with game tree search techniques. This claim led to DeepMind's AlphaGo, the first superhuman-level Go playing system [33], which combined supervised learning, policy gradient search and Monte Carlo Tree Search.

Othello has for a long time been a popular benchmark for computational intelligence methods [29], [35], [5], [25], [26], [42], [40], [28], [16], [30], [18]. All strong Othello-playing programs use a variant of the minimax search [7] with a board evaluation function. Past research suggested that more can be gained by improving the latter than the former; that is why recently the focus was mostly on training 1 look-ahead (a.k.a. 1-ply) agents using either self-play [26], [18], fixed opponents [20], [16], or expert game databases [30]. Multiple ways of training the agents have been proposed: value-based temporal difference learning [25], [42], [34], (co)evolution [26], [31], [17], [15], and hybrids thereof [39], [40].

Designing a board evaluation function involves finding a good function approximator. One of the most successful function approximators for board games are *n-tuple networks*. Interestingly, they were originally proposed for optical character recognition [4], and their reuse for board games is yet another sign that the analogy between images and piece arrangements on a board is feasible. They were employed for the first time for Othello under the name of *tabular value functions* by Buro [7] in his famous Logistello program and later popularized by Lucas [24]. Neural networks have also been used for Othello as function approximators, e.g., in [42], but, to the best of our knowledge, CNNs have never been used for this purpose.

As we are not aware of any published move predictors for Othello, in the experimental part of this paper we compare our method to a representative sample of 1-ply strategies. A comparison of the state-of-the-art 1-ply players for Othello can be found in [18]. Among the published works, the best 1-ply player to date was obtained by Coevolutionary CMA-ES (covariance matrix adaptation evolution strategies) [18] and systematic n-tuple networks. Among the multi-ply Othello strategies, one of the contemporary leaders is Edax[1], an open source player that uses tabular value functions, one for each stage of the game. Edax is highly optimized (bitboard move generator) and uses deep tree minimax-like search (negasout with multi-probcut tree search [6]). Unfortunately, a description of how its evaluation function was obtained is not publicly available.

One of the other top-5 Othello players to date is RL14_iPrefN that has been trained using preference learning on the French Othello League expert games dataset [30], reportedly obtaining 53.0% classification accuracy. In this paper, we significantly improve over this baseline.
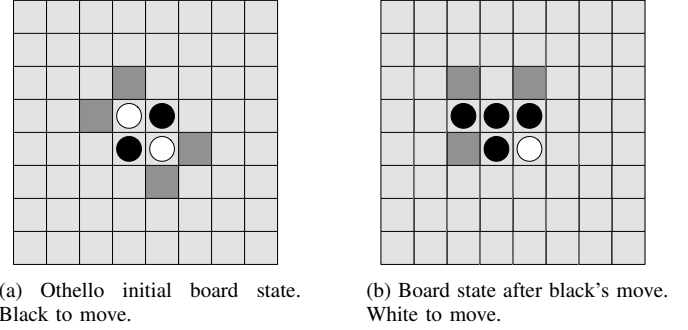

(a) Othello initial board state. Black to move.

(b) Board state after black's move. White to move.

Figure 1: Othello boards with legal moves marked as shaded locations.

III. OTHELLO

Othello is a perfect information, zero-sum, two-player strategy game played on an 8×8 board. There are 64 identical pieces which are white on one side and black on the other. The game begins with each player having two pieces placed diagonally in the center of the board (Fig. 1a). The black player moves first, placing a piece on one of four shaded locations, which may lead to the board state in Fig. 1b. A move is legal if the newly placed piece is adjacent to an opponent's piece and causes one or more of the opponent's pieces to become enclosed from both sides on a horizontal, vertical or diagonal line. The enclosed pieces are then flipped. Players alternate placing pieces on the board. The game ends when neither player has a legal move, and the player with more pieces on the board wins. If both players have the same number of pieces, the game ends in a draw.

Despite simple rules, the game of Othello is far from trivial. The number of legal positions is approximately 10^{28} and the game tree has in the order of 10^{58} nodes [1], which precludes any exhaustive search method. Othello is also characterized by a high temporal volatility: a high number of pieces can be flipped in a single move, dramatically changing the board state. These features and the fact that the game has not yet been solved makes Othello an interesting test-bed for computational intelligence methods.

IV. LEARNING TO PREDICT EXPERT MOVES

A. The Classification Problem

Since Othello is a deterministic and fully-observable environment, a policy $\pi : X \rightarrow Y$ is here a mapping from the space of board states X to the space of actions $Y = \{1, \ldots, K\}$ that correspond one-to-one to the $K = 60$ board positions. $\pi(x)$ represents the desired move for board state $x \in X$. Given a training set of N examples of good (e.g., experts') moves in $X \times Y$, we can formulate the problem of finding π as a *multi-class classification* problem. A typical approach to such problems is training a parameterized probability function $p(y|x; \theta)$, that represents the confidence

[1] http://abulmo.perso.neuf.fr/edax/

Othello board $2 \times 8 \times 8$ input $3 \times 8 \times 8$ feature maps $3 \times 8 \times 8$ feature maps   output layer
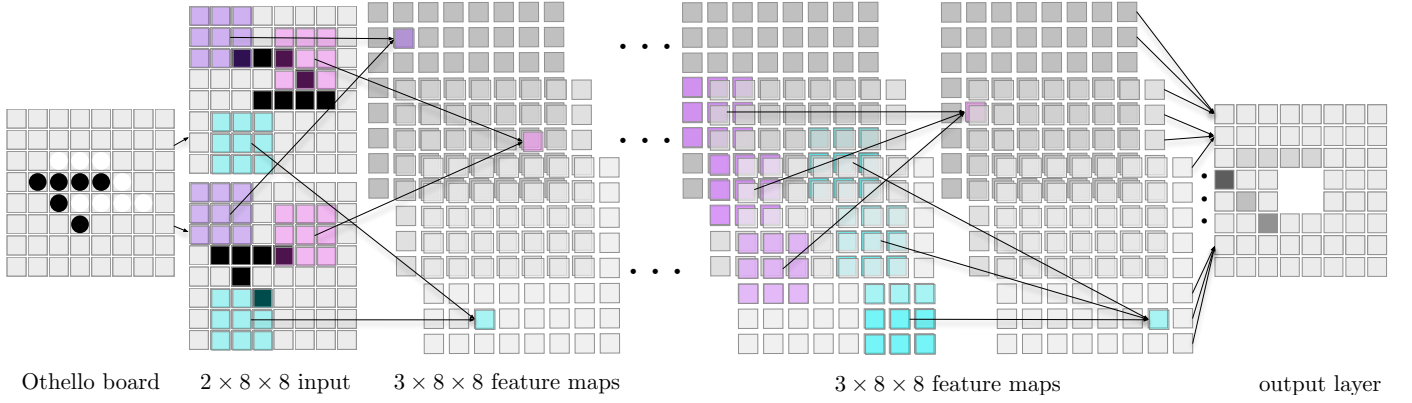
Figure 2: Exemplary CNN for move prediction. The board is fed into the network as an 8×8 two-channel binary image, where ones indicate the locations of the player's (first channel) and opponent's pieces (second channel). Units in a feature map (colored squares) receive data from 3×3 RFs (in the same color) in the image (or in a previous layer). The network uses zero-padding on the edges (not shown), so maps in all layers are 8×8. For clarity, each hidden layer has only 3 feature maps.

of making a move y in state x. To estimate p, one needs to minimize the cross-entropy loss:

$$L(\theta) = -\sum_{i=1}^{N} \sum_{k=1}^{K} y_i^{(k)} \ln p(y_i^{(k)}|x_i; \theta),$$

where $y^{(k)} = 1$ if $y = k$ and 0 otherwise. The task of a training algorithm is to estimate the optimal θ and so find $\hat{\theta}$, a policy parameterization that brings $L(\theta)$ sufficiently close to its global optimum. Once found, the player chooses the moves of highest confidence:

$$\pi_{\hat{\theta}}(x) = \arg\max_{y \in Legal(x)} p(y|x; \hat{\theta}),$$

where $Legal(x) \subset Y$ is the subset of moves that are legal in board state x.

If π determines the move to make without explicit simulation of the future game course (using, e.g., minimax-style algorithms), we refer to it as *move predictor*. One-to-one correspondence between board locations and moves in Othello greatly facilitates the design of move predictors. For other board games that involve heterogeneous pieces that move around the board, the space of moves Y is more complex and requires a more sophisticated representation.

B. CNNs for Move Prediction

In the simplest scenario, the board states in X could be encoded as 64 ternary variables and fed into any probability estimator. However, generic probability estimators are not biased towards the kind of structure that relates board state variables in Othello. To see that such a structure exists, it is enough to imagine the board variables being randomly permuted, which would completely obfuscate the game for human players (while being still the same policy learning task under the formulation in the previous section). This shows that capturing spatial patterns of the player's and opponent's pieces is essential, and makes it particularly appropriate to take advantage of learning systems that have such capabilities, i.e., treat a board state as an image.

Convolutional neural networks (CNNs) are nowadays the leading machine learning tool for image analysis (cf. the work

cited in the introduction). In the following, we briefly describe the CNN architecture as we tailored it to move prediction in Othello, referring readers interested in this topic to other comprehensive studies [2], [32]. For clarity, we present here only the core CNN architecture and augment it with extensions in the subsequent experimental section.

A CNN is a composite of multiple elementary processing units, each equipped with weighted inputs and one output, performing convolution of input signals with weights and transforming the outcome with some form of nonlinearity. The units are arranged in rectangular *feature maps* (grids) and spatially aligned with the raster of the input image. The spatial arrangement of units is the primary characteristic that makes CNNs suitable for processing visual information. Feature maps are arranged into *layers*, where maps in a given layer fetch signals only from the previous layer (see also Fig. 2).

Our CNNs take as input the representation of the current board state, encoded in two 8×8 binary matrices. The 1s indicate the locations of the player's pieces in the first matrix and opponent's pieces in the second matrix. We refer to this encoding as *pieces*. In the encoding called *vmoves*, we add a third matrix in which we mark with 1s the player's legal moves, to facilitate picking the valid moves during game playing. Following recent research conducted in Go [33], we define also the *ones* encoding, which includes an additional regularizing channel in *pieces* encoding that contains only ones.

A board state represented in the above way forms an 8×8 binary image comprising two (for *pieces*), or three (for *vmoves* and *ones*) channels, and is fetched by the feature maps in the first layer of the CNN. Each unit in a feature map receives data only from its *receptive field* (RF), a small rectangle of neighboring board locations, however from all channels, and uses a separate weight set for each channel (see exemplary RFs in Fig. 2). Analogously in the subsequent layers, a unit fetches data from the corresponding RFs in all feature maps in the previous layer, and treats them with separate weight sets.

The RFs of neighboring units in a map are offset by a *stride*. Board size, RF size and stride together determine the dimensions of a feature map. For instance, a feature map with

Table I: CNN architectures used in the experiments. Layer names are followed by numbers of feature maps.

Conv4	$conv64 \rightarrow conv64 \rightarrow conv128 \rightarrow conv128 \rightarrow fc128 \rightarrow fc60$
Conv6	$conv64 \rightarrow conv64 \rightarrow conv128 \rightarrow conv128 \rightarrow conv256 \rightarrow conv256 \rightarrow fc128 \rightarrow fc60$
Conv8	$conv64 \rightarrow conv64 \rightarrow conv128 \rightarrow conv128 \rightarrow conv256 \rightarrow conv256 \rightarrow conv256 \rightarrow conv256 \rightarrow fc128 \rightarrow fc60$

3×3 RFs with stride 1 needs only 36 units when applied to an 8×8 board, because six RFs of width 3 each, overlapping by two rows, span the entire board width (and analogously for height). The local connectivity substantially reduces the number of weights and facilitates capturing of spatial patterns.

Units within the same feature map share their weights, and so calculate the same local feature, albeit from a different part of the board. This reduces the number of parameters even further and makes the extracted features equivariant. For instance, the 6×6 feature map mentioned above has only $9 \cdot b + 1$ parameters for a b-channel input ($3 \times 3 = 9$ plus one threshold per feature map).

We use rectified linear units (ReLU) to process convolution outcomes, with the nonlinearity defined as $f(x) = max(x, 0)$. ReLUs make networks learn faster than squeezing-function units (e.g., sigmoid or $\tanh$) and are particularly effective in networks with many layers, effectively alleviating the vanishing gradient problem, because gradient backpropagates through them undistorted for all positive excitations, in contrast to a small interval of excitations for the squeezing units.

Our basic architectures are the networks composed of 4, 6, and 8 convolutional layers shown in Table I. In each architecture, the input is passed through a stack of convolutional layers composed of 64, 128, or 256 feature maps, with 3×3 RFs and the stride set to 1. Since the input is very small, we pad it with zeros on the grid border (so that the RFs can reach also beyond the input grid) in order to preserve the spatial resolution after convolution. For the same reason, we do not use any form of *pooling*, i.e., aggregating the outputs of multiple units by other means than convolution, as there is little rationale for that for small input grids [36]. Consequently, all feature maps in all layers have 8×8 units.

In all architectures, the stack of convolutional layers is followed by two fully-connected (FC) layers. The hidden layer consists of 128 units with ReLU activations, while the output layer has $K = 60$ units corresponding to 60 possible move candidates. Each output $y^{(k)}$ corresponds to a single board location, and the desired moves are encoded using the binary 1-of-K scheme (a.k.a. *one-hot encoding*). For the output values to be interpreted as probabilities they must sum to unity, which is achieved using the softmax transformation:

$$o_k = \frac{e^{y^{(k)}}}{\sum_j e^{y^{(j)}}},$$

The classifier selects the move with the highest probability, i.e., $\hat{o} = \arg\max_{k \in K} o_k$.

The architectures presented in Table I resulted from a series of preliminary experiments, where we found out that increasing network depth beyond eight convolutional layers does not improve the performance while significantly lengthening the training time. Also, small 3×3 RFs and deeper networks turn out to be more efficient in terms of trading-off runtime and accuracy than using fewer but larger RFs.

V. EXPERIMENTS IN MOVE PREDICTION ACCURACY

A. Experimental Setup

In this section, we consider networks as move predictors and do not engage them in actual games. The key performance indicator is prediction accuracy, i.e. the percentage of correctly predicted moves. We compare various configurations of the neural move predictors described in Section IV trained on examples of moves extracted from games played by humans (Section V-A2) .

1) Datasets: Created in 1985, the WThor database[2] contains the records of 119,339 games played by professional Othello players in various French tournaments. From those records, we extracted all board states accompanied with the color of the player to move and the move chosen by the player in that particular state. The resulting set of 6,874,503 (*board, color, move*) triples is referred to as *Original*. We then removed the duplicates, obtaining 4,880,413 unique (*board, color, move*) triples, referred to as *Unique* dataset in the following. Note that *both* datasets may contain inconsistent examples, i.e. the same board state associated with different moves. A perfect classifier can achieve accuracy of 91.16% on *Original* and 97.49% on *Unique*. The color attribute is used only to invert the pieces on the boards when the white player is to move, so that all states in the dataset are seen from the black player's perspective.

Deep neural networks are known to perform particularly well when trained on large volumes of data. In order to augment the datasets, we take advantage of Othello's invariance to board symmetries. For each board state in the *Original* and *Unique* datasets, we produce its all seven reflections and rotations. This results respectively in two *symmetric datasets*: *Original-S* (54,996,024 examples) and *Unique-S* (39,019,056 examples), where the latter was cleared also of the duplicates resulting from symmetric transformations.

To assess the generalization capabilities of move predictors, we partition each dataset into disjoint training and testing sets. For the asymmetric datasets (*Original* and *Unique*), we allocate 25% of examples for testing. The symmetric datasets are much larger, so we allocate only 5% of examples for testing, so that all test sets are similar in size.

2) Training: For training, we use stochastic gradient descent with 0.95 momentum. Units' weights are initialized using He's method [11] while the biases (offsets) are initialized with zeroes. The learning rate is initially set to 0.1, and then halved twice per epoch. The loss function is regularized by the L_2 norm with the weight of $5 \cdot 10^{-4}$. Learning is stopped after

Table II: Prediction accuracy of networks trained on four different training sets and evaluated on the *Original* test set. *Pieces* encoding was used.

Architecture	*Original*	*Original-S*	*Unique*	*Unique-S*
Conv4	55.1	56.9	55.9	**57.2**
Conv6	57.2	58.7	57.9	**59.1**
Conv8	58.1	60.1	58.9	**60.5**

Table III: Prediction accuracy on *Unique-S* test set for different board encodings and number of convolutional layers.

Architecture	*pieces*	*ones*	*vmoves*
Conv4	57.2	**57.3**	**57.3**
Conv6	**59.2**	59.1	59.0
Conv8	60.5	**60.6**	59.0

Table IV: Probability [%] of choosing a valid move on *Unique-S*.

Architecture	*pieces*	*ones*	*vmoves*
Conv4	99.95	99.95	100.0
Conv6	99.99	99.99	100.0
Conv8	99.99	99.99	100.0

Table V: The influence of dropout, batch normalization, and bagging on prediction accuracy [%]. *Pieces* encoding was used.

Architecture	Prediction accuracy	Prediction time [$\times 10^{-4}$s]
Conv8 (baseline)	60.5	1.6
Conv8+dropout	58.8	1.6
Conv8+BN	**62.7**	1.9
Conv8+BN+bagging	**64.0**	19.4

24 epochs of training for the asymmetric datasets ($171,600$ batches of examples) and 6 epochs ($434,400$ batches) for the symmetric datasets.

The implementation is based on the Theano framework [41], which performs all computation on a GPU in single-precision arithmetic. The experiments were conducted on an Intel Core i7-4770K CPU with NVIDIA GTX Titan GPU. Training time varied from single hours to dozens of hours depending on configuration, e.g. nearly 62 hours for Conv8 trained on *Unique-S*.

B. Results

1) Effect of Data Augmentation: Table II presents the test-set prediction accuracy of three basic network architectures trained on particular training sets with the *pieces* encoding. For fairness, the networks are compared on the same data, i.e. the *Original* test set. Training on the datasets augmented by symmetries systematically leads to a higher prediction accuracy. The impact of removing duplicates is also consistently positive, albeit not so strong. These two trends together render *Unique-S* most promising, so in the following experiments we use this dataset only.

2) Impact of Board Encoding and the Number of Layers: Table III presents the prediction accuracy of the three architectures when trained on *Unique*-S with various encodings of the board state. This time, the networks are tested on the *Unique-S* test set, which explains the minor differences w.r.t. Table II, where *Original* test set was used for testing. The accuracy clearly correlates with the number of layers, so it is tempting to reach for even deeper architectures. However, a 10-layer architecture analogous to Conv8 failed to further improve the prediction accuracy.

Concerning the comparison of encodings, *pieces* and *ones* seem to perform on par, so the former should be preferred as it does not require an additional input layer/channel. The relative underperformance of *vmoves* is more puzzling as it is hard to see how giving a learner a hint about moves' validity could deteriorate its performance. In an attempt to investigate this outcome, we evaluate the networks with respect to their ability to discern the valid and invalid moves. In Table IV, we present the test-set accuracy of particular networks, where 100% indicates that the network's most excited output

is always among the valid moves. All considered networks and encodings bring this indicator very close to perfection, with the networks using *vmoves* making no mistakes at all in that respect. This suggests that telling apart the valid moves from the invalid ones is very easy regardless of configuration, and providing this information explicitly in *vmoves* is not only unnecessary but may distract the training process. Therefore, we use the *pieces* encoding in all experiments that follow.

3) Impact of Regularization and Bagging: In Table V, we report the impact of a few techniques that tend to improve generalization performance of CNNs.

Dropout consists in disabling a number of (here: 50 percent) randomly selected units for the individual batches of training examples, which forces a network to form multiple alternative pathways from inputs to outputs. This usually makes networks more robust and reduces overfitting, particularly in domains where input data are inherently redundant, like natural images, typically composed of at least thousands of pixels. The states of Othello board are however very different: there are only 64 inputs, and every single piece counts when choosing the move to make. There is virtually no redundancy in the input. As Table V suggests, dropout prevents the Conv8 network from capturing the board state in full and so hampers its predictive accuracy.

The purpose of *batch normalization* (BN) is to address the *internal covariate shift* [14] by standardizing the outputs of individual network units over batches of examples, which was demonstrated to be beneficial in many contexts. This is corroborated in our case with over two percent gain in accuracy, so we consider this configuration as a baseline throughout the rest of this paper.

Finally, we attempt to reduce the classifier's variance by employing bootstrap aggregation (*bagging*). We train 10 networks on bootstrapped samples drawn from the training set and average their corresponding outputs to determine the move to make. When applied to the Conv8+BN network (Table V), bagging boosts the performance to 64%. The price paid for that is a 10-fold increase in not only the training time[3], but

---

[3]The mean prediction times were computed using batch size of 256. Making decision for a *single* board state takes on average 5-20 times more GPU time.

Table VI: Prediction accuracy of residual networks.

Architecture	Accuracy	Prediction time [$\times 10^{-4}$s]
Conv8+BN (baseline)	**62.7**	1.9
ResNet-32	57.4	1.0
ResNet-32-np	59.2	1.0
ResNet-32-np-p	59.5	1.0
ResNet-56	58.3	1.8
ResNet-56-np	60.1	1.8
ResNet-56-np–p	60.5	1.8
DenseNet-40	59.9	4.8
DenseNet-100	**61.9**	37.2
WideNet-40-4	**62.2**	9.4

also the prediction time, which will become relevant in the later parts of the study, when confronting our strategies with other opponents.

4) Architectures with Skip-Connections: The more recent DL research brought very deep network architectures containing dozens of layers [37], [12], which became the state-of-the-art in image classification. Training such networks is only possible due to *skip connections* that form shortcuts in the pathways between layers, typically adding the representation (feature map) learned in a given layer to the representation learned in one of the consecutive layers. This, particularly when repeated multiple times across the network, opens new ways to achieving the training goal: rather than learning a mapping H such that $y(x) = H(x)$, a network learns a residual F such that $y(x) = F(x) + x$. Skip connections alleviates the vanishing gradient problem as the gradient can backpropagate unchanged through many layers.

Table VI presents the predictive accuracy of several skip-connection architectures. A ResNet-n network [12] consists of $\frac{n}{2}$ *residual blocks*, each comprising a nonlinear layer (ReLU activation) followed by a linear layer, shortcut with a skip connection that adds the input of the first layer to the output of the second one. Originally designed for image analysis, ResNets perform downsampling that reduces the input dimensionality; as there is arguably little need for that for our very small 8×8 input, we consider a variant stripped off that feature, ResNet-n-np. As expected, maintaining the dimensionality of the input throughout learning improves the accuracy of the smaller (ResNet-32) and larger (ResNet-56) architecture.

Simple adding of layer's outputs is possible only when they have the same dimensions. Otherwise, the authors of ResNets propose to implement the skip connections as linear mappings. The ResNet-n-np-p configurations in Table VI implement that feature, which slightly improves the predictions.

ResNets' skip connections can be considered first-order as each of them spans the inputs and outputs of the same residual block. In DenseNets [13], skip connections are being introduced between each pair of layers, so in a network of l layers (or residual blocks) there are $\frac{l(l-1)}{2}$ of them. We trained two DenseNet architectures. The smaller one (DenseNet-40) managed to perform on par with the other networks considered in this section, while the larger one (DenseNet-100) significantly outperformed them, almost reaching the performance level of the baseline.
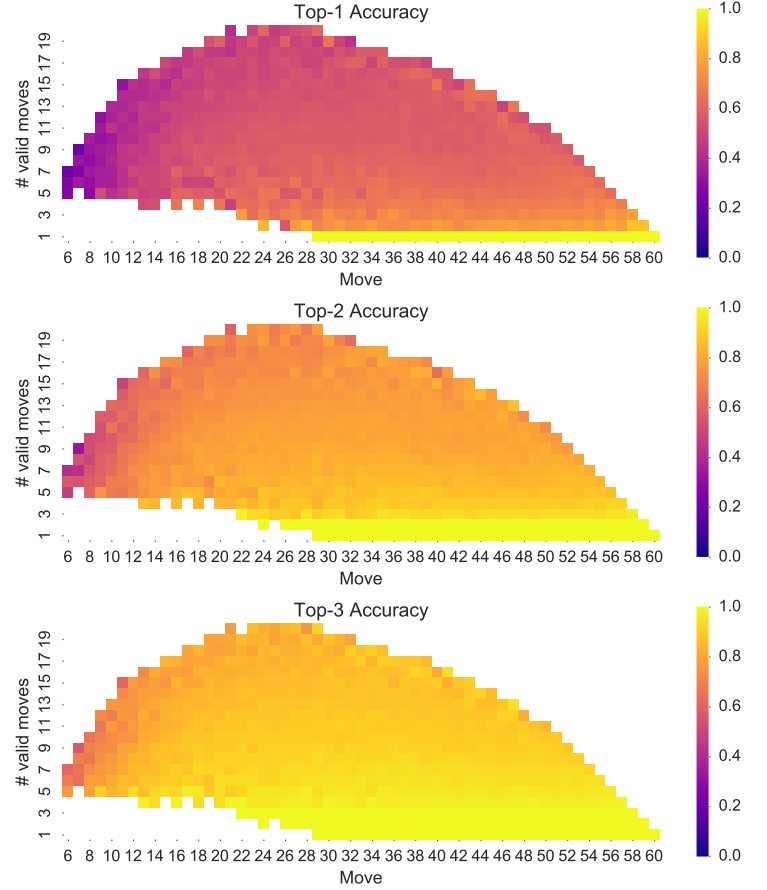

Figure 3: Top-k prediction accuracy factored w.r.t. the number of move in a game and the number of valid moves.

Finally, a very recent work on architectures dubbed WideNet [43] revealed that the number of layers (and thus parameters) in residual networks can be significantly reduced with little or no harm for accuracy by making the residual blocks more expressive. This can be achieved by increasing the number of convolution layers within a block, increasing the number of feature maps, or both. Our instance of this architecture comprises 3 stacks of residual blocks, each stack composed of 6 blocks. Each residual block contains two 3×3 convolutional layers. The layers in consecutive stacks are equipped with respectively 64, 128, and 256 feature maps (cf. Table 1 in [43] for $k = 4$), so the network is four times wider than the original ResNet. As the last row of Table VI demonstrates, this WideNet configuration fares the best among the skip-connections architectures considered here, which suggests that aggregating multiple features at various stages of processing is essential for successful move prediction. However, WideNet-40-4 is also one of the slowest networks when queried.

Overall however, contrary to the expectations, neither of the very-deep skip-connection networks did improve over the Conv8+BN baseline.

5) Prediction Accuracy Analysis: The move prediction task is inherently multimodal: to play well, correct predictions have to be made along the entire course of the game, for

the board states typical for the initial game stages as well as those occurring in the endgame. It is thus interesting to ask how difficult it is for a network to perform well at particular stages of the game. Here, we answer this question in terms of prediction accuracy. In Fig. 3, we present the top-k prediction accuracy for our baseline network Conv8+BN, factored with respect to the number of a move in a game and the number of legal moves in a given state. Given four pieces in the initial board state, the former factor may vary from 5 to 60; however we start with 6 as the fifth move is always the same under the Othello symmetries (which we take into account here). The color of a datapoint reflects the probability that the correct move (as per the WThor test set) is among the top k indicated by the network.

The figure reveals that the prediction accuracy remains largely consistent throughout the game. Only in the game's beginning the network's predictions are less accurate, which we attribute to a relatively small amount of training data for *Unique-S* at this game stage (due to many identical states). The other aspect at play, suggested by the differences between the top-1 and top-3 graphs, is the large fraction of inconsistent examples for the early game states, when there are many alternative paths to success and players tend to follow them accordingly to their personal habits and preferences. The high quality of predictions in the later game stages (also when the number of available moves is high) is particularly encouraging, as even a single move towards the end of a game can drastically change the state of the board – the feature that Othello owes its name to.

VI. PLAYING STRENGTH ANALYSIS

Prediction accuracy considered in the previous section is only a surrogate measure of performance, since we are ultimately interested in the playing strength. In this section, we let our predictors play against other players.

A. Experimental Setup

We consider a suite of thirteen 1-ply players proposed in previous research, gathered and compared in our former study [18]. The suite consists of players with board evaluation functions encoded by weighted piece counter and n-tuple networks, trained by different methods including hand-design, temporal difference learning, evolution, and coevolution (cf. Section II). In addition to that suite of opponents, we consider also Edax ver. 4.3.2[4], a strong, open source, highly optimized Othello program. Edax can be configured to play at a given ply (search depth) n, which we denote as Edax-n.

Since our move predictors as well as all 14 opponents (including Edax) are deterministic, we resort to the following technique in order to obtain robust estimates of the odds of winning. Rather than playing a single game that starts from the initial board, we use a set of 1000 6-ply opening positions prepared by Runarsson and Lucas [30]. Starting from each such state, two games are played, with the players switching roles. For a single game, a player can obtain 1, 0.5 or 0 points

[4]http://abulmo.perso.neuf.fr/edax/4.3/edax.4-3-2.zip

Table VII: The winning rates of the best performing move predictors against the 1-ply opponent players. Note that while CoCMAES is included in the 13 opponents suite, Edax-1 is not.

Architecture	Prediction accuracy	Winning rate		
		Average (13 opponents)	CoCMAES -4+2x2	Edax-1
ResNet-56-np-p	60.5	94.3	87.7	80.7
DenseNet-sym-100	61.9	94.1	87.5	79.0
WideNet	62.2	96.3	92.2	**88.1**
Conv8+BN	62.7	**96.7**	93.2	87.5
Conv8+BN+bagging	**64.0**	96.6	**93.5**	87.6

on the win, draw, or loss, respectively. We report the *winning rate*, i.e. the percentage of points possible to score in 2000 games.

B. Playing Strength Evaluation

We selected a representative sample of the five best predictors from Section V and confronted them with all 13 opponents from our suite as well as with Edax-1. Table VII presents their average winning rates against the 13 players, and separately the winning rates against CoCMAES-4+2x2 that proved the best in [18] and against Edax-1.

All networks turn out to be significantly stronger than any of the existing 1-ply look-ahead players, achieving around 90% winning rate against them. We find this interesting, given that move predictors are essentially 0-ply strategies, as they do not perform any look-ahead. Conv8+BN achieves the highest average winning rate, and is only slightly worse than Conv8+BN+bagging when playing against CoCMAES-4+2x2 and than WideNet when playing against Edax-1. This overall performance corroborates its high prediction accuracy.

The roughly monotonous relationship between the prediction accuracy and the winning rate observed in Table VII lets us ask whether such a tendency holds in general. In Fig. 4, we plot the latter against the former for all networks evaluated in this paper, when confronted with the models in the suite and with Edax-1. The graph reveals a very strong linear dependency (determination coefficient $r^2 = 0.997$ and $r^2 = 0.999$, respectively). Fitting linear models suggests that in the considered range of prediction accuracy (57-64 percent), each percent point of the prediction accuracy brings 1.7 percent points of the winning rate against the suite of 13 players, and 3.4 percent points against Edax-1. Obviously, this trend cannot hold globally, as illustrated by Conv8+BN+bagging (the rightmost two data points). Although this might be an outlier, it may also suggest that 63 percent is approximately the point at which further increases of prediction accuracy do not translate into better winning rates anymore.

C. Effect of Data Augmentation on Playing Strength

In Section V-B1, we found out that the training set stripped of duplicates and augmented with symmetries (*Unique-S*) gives rise to the best move prediction accuracy. However, prediction accuracy is only a proxy of what we really care about – the winning rate. It is thus not obvious whether an analogous claim holds for playing strength, so, in this

factor of Othello is 10, so for search depth 5 Edax's decision are based of tens of thousands of board states each, while our neural predictor performs quite well with just one.

Search depth impacts the mean time required by Edax to make a move, which we report in the last column of the table. As Edax is highly optimized, it is able to make moves at a rate comparable to the prediction time of the Conv8 CNN (1.9×10^{-4}s) for search depths up to 3. Deeper search takes Edax longer. On the other hand, however, the times reported for neural predictors have been assessed in *batches* of 256 board states, the common efficiency means in GPU computing. When asked to predict a move for a single board state, CNNs are 5-20 times slower.

Table IX includes also Edax prediction accuracy on the *Original* dataset, which clearly grows with the depth of search. However, increasing the depth from 4 to 5 does not improve the accuracy, despite the associated increase in head-to-head matches against Conv8+BN. Also, Edax's prediction accuracy (59% at $n = 5$) is significantly lower than the 64% demonstrated by Conv8+BN. It seems thus that Edax plays in a different way than human players, whom CNNs try to mimic.

Figure 4: Networks' move prediction accuracy vs. winning rate against all 13 opponent players (red dots) and against Edax (blue triangles).

Table VIII: The winning rates against the 13 players from the suite for the basic neural predictors trained on different datasets

Architecture	*Original*	*Original-S*	*Unique*	*Unique-S*
Conv4	77.9	85.2	80.5	**86.0**
Conv6	84.6	89.0	86.5	**90.0**
Conv8	85.4	91.8	88.6	**93.1**

section we let the basic convolutional predictors from Section V-B play against the opponents from the suite. Other settings remain as in the previous section.

The winning rates presented in Table VIII corroborate the outcomes in terms of prediction accuracy presented earlier in Table II. Both duplicate removal and augmentations have significantly positive effects on the playing strength. This correlation shows again that, at least in certain intervals, move prediction accuracy is a reliable predictor of the playing strength (cf. Fig. 4).

D. Playing against Game Tree Search

Section VI-B revealed that the 0-ply neural move predictors outperform the 1-ply opponents, and do so by a large margin (~ 90 percent vs. the 50 percent tie point). It is interesting to see how that margin decreases when the opponents are allowed for deeper game-tree searches. To investigate this, we confront Conv8+BN with Edax-n for search depth $n \in [2, 5]$ (using the same protocol involving 2000 games), and gather the results in Table IX. Remarkably, our predictor maintains superiority when Edax is allowed for 2-ply lookahead, though the odds of its winning are now only roughly 6:4. Starting from search depth 3, Conv8+BN is more likely to lose than to win, and for 5-ply search its odds for winning drop to 1:9. This is however still impressive, given that the average branching

Table IX: The performance of Conv8+BN vs. Edax-n.

n	Winning rate	Edax move time [$\times 10^{-4}$s]	Edax prediction accuracy
1	87.5	1.0	47.9
2	59.4	1.1	53.3
3	34.7	1.6	55.2
4	19.8	5.0	59.8
5	10.3	10.0	59.7

E. Analysis of Game Stages

As much as we find the performance of our 0-ply move predictors impressive, the previous experiments show that deeper lookaheads are essential for achieving high winning rates. In our last experiment, we seek to find out which game stage may benefit the most when supplementing our move predictor with game tree search (and conversely, which game stage is most problematic for neural predictors). To this aim, we split the game into four stages according to intervals of move numbers: $[1, 15]$, $[16, 30]$, $[31, 45]$, and $[46, 60]$. For each stage, we design a hybrid strategy that employs Edax-5 in that stage, while using the Conv8+BN policy to make decisions in the remaining stages. We let those hybrid strategies play against Edax-1..5.

The results summarized in Fig. 5 confirm that letting Edax make moves at any game stage improves the performance of the hybrid. The gains are largest for $n = 3$; as demonstrated in Section VI-D, for smaller n Edax is relatively easy to beat already by pure Conv8+BN, so extending it with Edax-5 at a single game stage does not improve the odds for winning much. Conversely, for $n > 3$ the opponent Edax-n becomes so strong that it is hard to increase the odds of winning either.

The graphs reveal also that there is more to gain at later stages of the game regardless of the opponent's depth. We suspect that this is due to the combinatorial nature of the game, which makes the CNN's pattern recognition capabilities less relevant for success in the last game stage. Also, given that Othello games last *at most* 60 moves, a search depth 5 in the last game stage enables Edax to often reach the final game states and so obtain robust assessments of potential moves. Last but not least, certain fraction of moves derived from WThor can be suboptimal and so might have 'deceived' the networks.
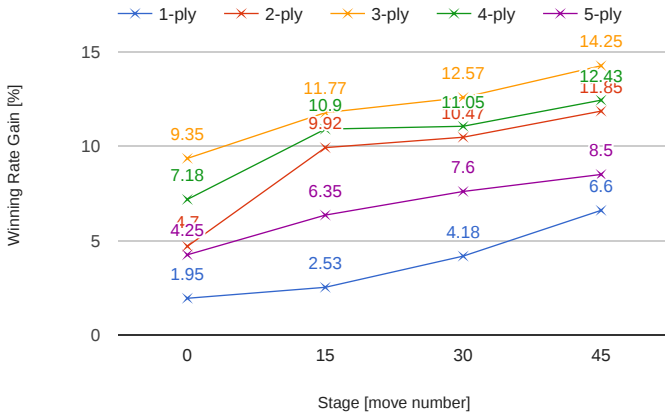
Figure 5: The gain of winning rate of Conv8+BN when hybridized with Edax-5 for a particular stage of the game. Winning rates are assessed independently against Edax-1..5.

VII. DISCUSSION

The deep CNN-based move predictor with its nearly 63% of accuracy turned to be much better than n-tuples-based preference learning [30], which achieved only 53% on the same dataset. Our Conv-8-BN wins against it in 93.9% of games. We obtained a similar winning rate of 93.2% also against the CoCMAES player [18], the strongest to-date 1-ply Othello player (cf. Table VII).

Despite Othello's relatively low branching factor and small board, CNNs proved to be supreme move predictors, managing to efficiently capture the expertise materialized in millions of humans' moves. We find that impressive given that small, discrete-valued board states are arguably very different from natural images[5].

Seeing a *convolutional* network to perform so well in a board game that is characterized with little translation invariance is intriguing, as the main motivation for using convolution is precisely the translation invariance. This suggests that certain patterns observed in Othello boards can be to some extent translation-invariant, or at least can be detected using 'higher-order convolutions' implemented by a stack of several convolutional layers intertwined with nonlinearities.

Concerning limitations of the approach, we could not improve upon the result obtained with the Conv-8-BN network by using deeper networks with skip connections. This suggests that what works for high-dimensional image data does not always succeeds for low-dimensional problems such as Othello.

An interesting observation concerning training is that the large networks hardly overfit to the data. The classification accuracy on the training set is typically within one percent point of the accuracy on the test set, suggesting that the networks we considered here cannot memorize the training data entirely. We may conclude thus that there is still room for improving their architectures. In particular, this may be a sign of an inherent limitation of convolutional processing, where the limited in size RFs may be insufficient to capture

arbitrary complex combinatorial patterns occurring in entire board states. A natural follow-up step is thus considering fully-connected networks that do not suffer from that limitation; however, they may be also much more costly in training and tend to overfit. In relation to that, an interesting study regarding fully-connected networks for Othello with up to 3 hidden layers was performed in [3].

VIII. CONCLUSIONS

Our study brings evidence that deep neural networks are a viable methodology for training Othello agents. They do not require explicit knowledge on game rules or strategies, nor extensive game-tree searches, nor explicitly compare individual future board states or actions. Rather than that, they achieve a high level of playing by learning the associations between the patterns observed in board states and the desired moves. We may claim thus that, at least to some extent, the trained neural predictors embody the knowledge of the Othello domain – a statement that would be hard to defend for game-tree search techniques, which owe their power largely to algorithmic efficiency.

On the other hand, the supervised approach proposed here cannot be deemed as 'knowledge-free' in the same sense as for instance evolutionary or reinforcement learning methods. Here, the essential know-how is hidden in the expert games used for training. CNNs power lies in the ability to extract and incorporate this knowledge into a system that is able to generalize beyond the training examples, something no other method in the past has been shown to do well enough to achieve the expert level of play in Othello.

Given the supreme performance of CNNs, this work may be worth extending in several directions. The networks that rely more on the fully-connected layers, mentioned in Section VII, are one such possibility. Logistello [7] and Edax use different evaluation functions for each stage of the game (cf. SectionVI-E), and such 'factorization by game course' could also lead to a better CNN agent. Another, arguably simple extension is to add the information about the player to move, which we expect to be beneficial given that Othello is a non-symmetric game. Last but not least, given the usefulness of convolutional features evidenced here, it would be interesting to verify their capabilities in other contexts than supervised learning of move predictors, e.g. as state or state-action evaluation functions trained with reinforcement learning methods.

REFERENCES

[1] Victor L Allis. *Searching for solutions in games and artificial intelligence*. PhD thesis, University of Limburg, Maastricht, The Netherlands, 1994.

---

[5]Note that the images considered in computer vision hardly ever contain fewer than 100 pixels.

[2] Yoshua Bengio. Learning deep architectures for AI. *Foundations and trends in Machine Learning*, 2(1):1–127, 2009.

[3] Kevin J Binkley, Ken Seehart, and Masafumi Hagiwara. A study of artificial neural network architectures for Othello evaluation functions. *Information and Media Technologies*, 2(4):1129–1139, 2007.

[4] Woodrow Wilson Bledsoe and Iben Browning. Pattern recognition and reading by machine. In *Proc. Eastern Joint Comput. Conf.*, pages 225–232, 1959.

[5] M Buro. An evaluation function for Othello based on statistics. Technical report, NEC, Princeton, NJ, 1997.

[6] Michael Buro. Experiments with multi-probcut and a new high-quality evaluation function for othello. *Games in AI Research*, pages 77–96, 1997.

[7] Michael Buro. Experiments with Multi-ProbCut and a new high-quality evaluation function for Othello. In H.J. van den Herik and H. Iida, editors, *Games in AI Research*, pages 77–96. University Maastricht, 2000.

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

[9] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[10] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on Imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[13] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.

[14] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[15] Wojciech Jaśkowski. Systematic n-tuple networks for othello position evaluation. *ICGA Journal*, 37(2):85–96, June 2014.

[16] Wojciech Jaśkowski. Systematic n-tuple networks for position evaluation: Exceeding 90% in the othello league. Technical Report RA-06/2014, arXiv:1406.1509, Institute of Computing Science, Poznan University of Technology, Poznań, Poland, 2014.

[17] Wojciech Jaśkowski, Paweł Liskowski, Marcin Szubert, and Krzysztof Krawiec. Improving coevolution by random sampling. In Christian Blum, editor, *GECCO'13: Proceedings of the 15th annual conference on Genetic and Evolutionary Computation*, pages 1141–1148, Amsterdam, The Netherlands, July 2013. ACM.

[18] Wojciech Jaśkowski and Marcin Szubert. Coevolutionary CMA-ES for knowledge-free learning of game position evaluation. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(4):389–401, 2016.

[19] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning. In *IEEE Conference on Computational Intelligence and Games*, pages 341–348, Santorini, Greece, Sep 2016. IEEE.

[20] Krzysztof Krawiec and Marcin Grzegorz Szubert. Learning n-tuple networks for Othello by coevolutionary gradient search. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation - GECCO '11*, pages 355–362, New York, New York, USA, jul 2011. ACM Press.

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

[22] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[23] Paweł Liskowski and Krzysztof Krawiec. Segmenting retinal blood vessels with deep neural networks. *IEEE transactions on medical imaging*, 35(11):2369–2380, 2016.

[24] Simon M. Lucas. Learning to play Othello with N-tuple systems. *Australian Journal of Intelligent Information Processing Systems, Special Issue on Game Technology*, 9(4):01–20, 2007.

[25] Simon M. Lucas and Thomas P. Runarsson. Temporal difference learning versus co-evolution for acquiring Othello position evaluation. In *IEEE Symposium on Computational Intelligence and Games*, pages 52–59. IEEE, 2006.

[26] Edward P Manning. Using Resource-Limited Nash Memory to Improve an Othello Evaluation Function. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(1):40–53, 2010.

[27] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016.

[28] Michiel Van Der Ree and Marco Wiering. Reinforcement learning in the game of Othello: Learning against a fixed opponent and learning from self-play. In *Adaptive Dynamic Programming And Reinforcement Learning (ADPRL), 2013 IEEE Symposium on*, pages 108—115, 2013.

[29] P.S. Rosenbloom. A world-championship-level othello program. *Artificial Intelligence*, 19(3):279–320, 1982.

[30] Thomas Runarsson and Simon Lucas. Preference Learning for Move Prediction and Evaluation Function Approximation in Othello. *Computational Intelligence and AI in Games, IEEE Transactions on*, 6(3):300–313, 2014.

[31] Spyridon Samothrakis, S Lucas, TP Runarsson, and David Robles. Coevolving Game-Playing Agents: Measuring Performance and Intransitivities. *IEEE Transactions on Evolutionary Computation*, 17(2):213–226, 2013.

[32] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[33] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[34] Ioannis E. Skoulakis and Michail G. Lagoudakis. Efficient Reinforcement Learning in Adversarial Games. In *2012 IEEE 24th International Conference on Tools with Artificial Intelligence*, pages 704–711. IEEE, nov 2012.

[35] R. Smith and B. Gray. Co-adaptive genetic algorithms: An example in othello strategy. In *7th Florida AI Research Symposium*, 1994.

[36] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[37] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[38] Maciej Szkulmowski, Daniel Ruminski, Paweł Liskowski, Bartosz Wieloch, Krzysztof Krawiec, Bartosz Sikorski, and Maciej D Wojtkowski. Oct retinal angiography using neural networks. *Investigative Ophthalmology & Visual Science*, 57(12):454–454, 2016.

[39] Marcin Szubert, Wojciech Jaśkowski, and Krzysztof Krawiec. Learning Board Evaluation Function for Othello by Hybridizing Coevolution with Temporal Difference Learning. *Control and Cybernetics*, 40(3):805–831, 2011.

[40] Marcin Szubert, Wojciech Jaśkowski, and Krzysztof Krawiec. On scalability, generalization, and hybridization of coevolutionary learning: a case study for othello. *IEEE Transactions on Computational Intelligence and AI in Games*, 5(3):214–226, 2013.

[41] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[42] Sjoerd van den Dries and Marco A. Wiering. Neural-Fitted TD-Leaf Learning for Playing Othello With Structured Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 23(11):1701–1713, November 2012.

[43] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.