Learning to Play *Othello* With Deep Neural Networks

Paweł Liskowski, Wojciech Jaśkowski, and Krzysztof Krawiec

***Abstract*—Achieving a superhuman playing level by AlphaGo corroborated the capabilities of convolutional neural network (CNN) architectures for capturing complex spatial patterns. This result was, to a great extent, due to several analogies between *Go* board states and 2-D images that 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*—Convolutional neural networks (CNNs), deep learning, *Othello*.**

I. Introduction

NEURAL networks, particularly deep and convolutional neural networks (CNNs), excel at recognizing virtually all kinds of patterns [1]. They naturally generalize to arbitrary dimensionality, which makes them suitable for analyzing raster images, time series, video sequences, and 3-D volumetric data in medical imaging [2]. These properties make CNNs powerful in solving a wide range of challenging tasks at an unprecedented performance level, ranging from image classification [3], to object localization [4], object detection [5], image segmentation [6], and even visual reinforcement learning in 3-D environments [7].

The structural analogy between 2-D 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., [8]). In most cases however, neural nets served there as board evaluation functions employed for searching game trees. It has been 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 [9].

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 [10]. This problem not only diverges from *Go*, but is also very different from the 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 hardly ever been evaluated in such settings.

Our contributions include the following.

1) An experimental study of different CNN-based architectures of deep neural networks for *Othello*.
2) State-of-the-art move prediction accuracy on the French *Othello* league game data set WThor.
3) State-of-the-art 0-ply policy for *Othello* (no lookahead) that significantly improves over the previous approaches.
4) An in-depth analysis of the characteristics of trained policies, including the relationship between move prediction accuracy and winning odds, strengthes 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 the best-to-date move predictors for *Othello*.

Manuscript received April 21, 2017; revised August 23, 2017 and October 15, 2017; accepted November 13, 2017. Date of publication January 31, 2018; date of current version December 13, 2018. The work of P. Liskowski was supported by the National Science Centre, Poland, under Grant 2014/15/N/ST6/04572. The work of W. Jaśkowski was supported by the Ministry of Science and Higher Education Grant "Mobility Plus" 1296/MOB/IV/2015/0. The work of K. Krawiec was supported by the National Science Centre, Poland, under Grant 2014/15/B/ST6/05205. The computations were performed at Poznań Supercomputing and Networking Center. *(Corresponding authors: Paweł Liskowski.)*

P. Liskowski and K. Krawiec are with the Institute of Computing Science, Poznań University of Technology, Poznań 60-965, Poland (e-mail: pliskowski@cs.put.poznan.pl; krawiec@cs.put.poznan.pl).

W. Jaśkowski was with IDSIA Dalle Molle Institute for Artificial Intelligence Research, Manno CH-6928, Switzerland and with the Institute of Computing Science, Poznań University of Technology, Poznań 60-965, Poland. He is now with NNAISENSE, Lugano, Switzerland (e-mail: wjaskowski@cs.put.poznan.pl).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TG.2018.2799997

II. Related Work

CNNs have been introduced by Fukushima in Neocognitron [11] and occasionally used for selected image analysis tasks for the next two decades [12]. 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 [3], [13].

As signaled in Section I, 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 at applying CNNs to board games [14]. 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 [9], 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 [8], [10], [15]–[23]. All strong *Othello*-playing programs use a variant of the minimax search [24] 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-ply lookahead (also known as 1 ply) agents using either self-play [10], [19], fixed opponents [22], [25], or expert game databases [23]. Multiple ways of training the agents have been proposed: value-based temporal difference learning [8], [18], [26], (co)evolution [19], [27]–[29], and hybrids thereof [20], [30].

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 [31], 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 [24] in his famous Logistello program and later popularized by Lucas [32]. Neural networks have also been used for *Othello* as function approximators, e.g., in [8], 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 [10]. Among the published works, the best 1-ply player to date was obtained by coevolutionary covariance matrix adaptation evolution strategies (CMA-ES) [10] 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.

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

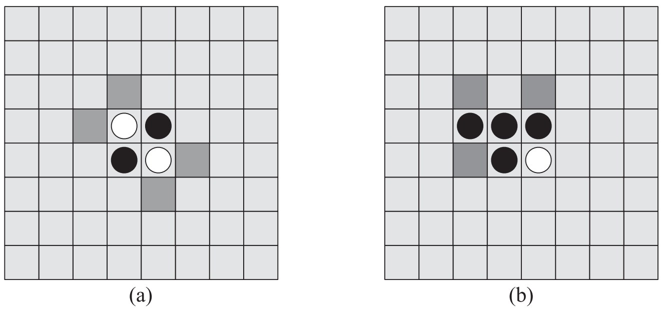
Fig. 1. *Othello* boards with legal moves marked as shaded locations. (a) *Othello* initial board state. Black to move. (b) Board state after black's move. White to move.

Edax is highly optimized (bitboard move generator) and uses deep tree minimax-like search (negasout with multiprobcut tree search [33]). Unfortunately, a description of how its evaluation function was obtained is not publicly available.

One of the other top-five *Othello* players to date is RL14_iPrefN that has been trained using preference learning on the French *Othello* League expert games data set [23], reportedly obtaining 53.0% classification accuracy. In this paper, we significantly improve over this baseline.

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 [see Fig. 1(a)]. The black player moves first, placing a piece on one of four shaded locations, which may lead to the board state in Fig. 1(b). 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 [34], 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. *Classification Problem*

Since *Othello* is a deterministic and fully observable environment, a policy $\pi : X \rightarrow Y$ is here a mapping from the space

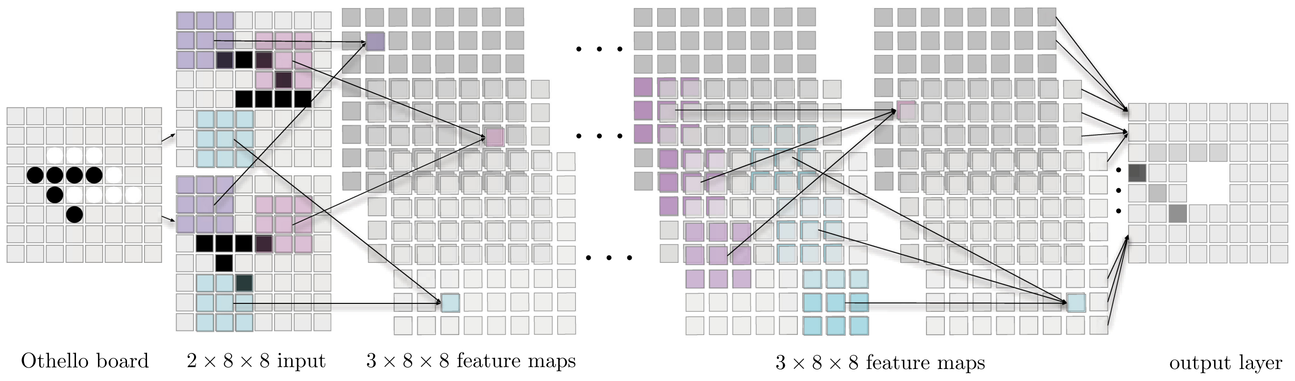
Othello board
$2 \times 8 \times 8$ input
$3 \times 8 \times 8$ feature maps
$3 \times 8 \times 8$ feature maps
output layer

Fig. 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 three feature maps.

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 multiclass classification problem. A typical approach to such problems is training a parameterized probability function $p(y|x;\theta)$, which represents the confidence 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

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

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 toward 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 still being 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.

CNNs are nowadays the leading machine learning tool for image analysis (cf., the work cited in Section I). In the following, we briefly describe the CNN architecture as we tailored it to move prediction in *Othello*, while referring readers interested in this topic to other comprehensive studies [1], [35]. 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* [9], we also define the ones encoding, which includes an additional regularizing channel in pieces encoding that contains only ones.

A board state represented in the above-mentioned 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 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 previously 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 (ReLUs) 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 also reach 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, whereas 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 (also known as 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 (see Section V-A2) .

1) Data sets: 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* data set in the following. Note that *both* data sets may contain inconsistent examples, i.e., the same board state associated with different moves. A perfect classifier can achieve the 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 data set 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 data sets, we take advantage of *Othello*'s invariance to board symmetries. For each board state in the *Original* and *Unique* data sets, we produce its all seven reflections and rotations. This results, respectively, in two symmetric data sets: *Original-S* (54 996 024 examples) and *Unique-S* (39 019 056 examples), where the latter was also cleared of the duplicates resulting from symmetric transformations.

To assess the generalization capabilities of move predictors, we partition each data set into disjoint training and testing sets. For the asymmetric data sets (*Original* and *Unique*), we allocate 25% of examples for testing. The symmetric data sets 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 [37], whereas 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 24 epochs of training for the asymmetric data sets (171 600 batches of examples) and 6 epochs (434 400 batches) for the symmetric data sets.

The implementation is based on the Theano framework [38], which performs all computation on a GPU in single-precision arithmetic. The experiments were conducted on an Intel Core

[2]http://www.ffothello.org/informatique/la-base-wthor/

TABLE I
CNN Architectures Used in the Experiments

Conv 4	conv64 → conv64 → conv128 → conv128 → fc128 → fc60
Conv 6	conv64 → conv64 → conv128 → conv128 → conv256 → conv256 → fc128 → fc60
Conv 8	conv64 → conv64 → conv128 → conv128 → conv256 → conv256 → conv256 → conv256 → fc128 → fc60

Layer names are followed by numbers of feature maps.

TABLE II
Prediction Accuracy of Networks Trained on Four Different Training Sets and Evaluated on the *Original* Test Set

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**

Pieces encoding was used.

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
Influence of Dropout, Batch Normalization, and Bagging on Prediction Accuracy [%]

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

Pieces encoding was used.

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 h for Conv8 trained on *Unique-S*. The code we used to conduct the experiments is available online.[3]

B. Results

1) Effect of Data Augmentation: Table II presents the test-set prediction accuracy of three basic network architectures trained on the 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 data sets 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 data set 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 the *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 ten-layer architecture analogous to Conv8 failed to further improve the prediction accuracy.

[3]https://github.com/pliskowski/deep-othello

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 the generalization performance of CNNs.

Dropout consists in disabling a number of (here: 50%) 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 the *Othello* board are however very different: there are only 64

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

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 [39] 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 more than 2% 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 ten 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 (see Table V), bagging boosts the performance to 64%. The price paid for that is a tenfold increase in not only the training time,[4] but 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 deep learning (DL) research brought very deep network architectures containing dozens of layers [3], [40], 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 achieve 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 alleviate 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 [3] consists of $n/2$ 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 [41], skip connections are being introduced between each pair of layers, so in a network of l layers (or residual blocks) there are $l(l-1)/2$ 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, whereas the larger one (DenseNet-100) significantly outperformed them, almost reaching the performance level of the baseline.

Finally, a very recent work on architectures dubbed WideNet [42] revealed that the number of layers (and thus parameters) in residual networks can be significantly reduced with little or no harm on 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 three stacks of residual blocks, each stack composed of six 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., [42, Table I] 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 improved 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 moves 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.

[4]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.

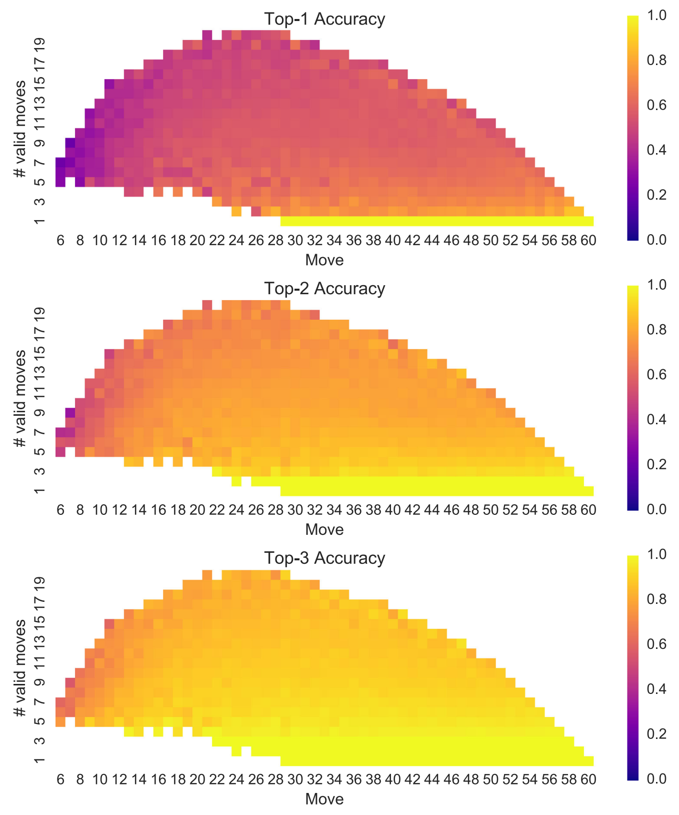

Fig. 3. Top-k prediction accuracy factored w.r.t. the number of moves in a game and the number of valid moves.

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 according 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 toward 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 [10]. The suite consists of players with board evaluation functions encoded by a 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 also consider Edax ver. 4.3.2,[5] 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 [23]. 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 on the win, draw, or loss, respectively. We report the winning rate, i.e., the percentage of points possible to score in 2000 games.

TABLE VII
Winning Rates of the Best Performing Move Predictors Against the 1-ply Opponent Players

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

Note that while CoCMAES is included in the 13 opponents suite, Edax-1 is not.

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 [10] and against Edax-1.

All networks turn out to be significantly stronger than any of the existing 1-ply lookahead 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 lookahead. Conv8+BN achieves the highest average winning rate, and is only slightly better than Conv8+BN+bagging when playing against CoCMAES-4+2x2 and better 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%), each percent point

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

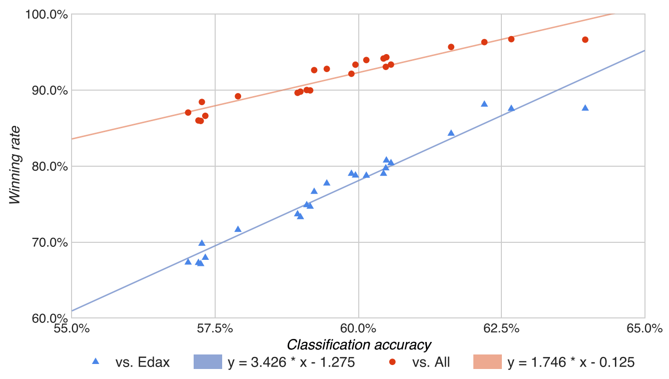

Fig. 4. Networks' move prediction accuracy versus winning rate against all 13 opponent players (red dots) and against Edax (blue triangles).

TABLE VIII
WINNING RATES AGAINST THE 13 PLAYERS FROM THE SUITE FOR THE BASIC NEURAL PREDICTORS TRAINED ON DIFFERENT DATA SETS

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**

TABLE IX
PERFORMANCE OF CONV8+BN VERSUS 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

of the prediction accuracy brings 1.7% points of the winning rate against the suite of 13 players, and 3.4% 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% 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 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 ($\sim 90\%$ versus the 50% 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 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 also includes Edax prediction accuracy on the *Original* data set, 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 thus seems that Edax plays in a different way than human players, whom CNNs try to mimic.

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

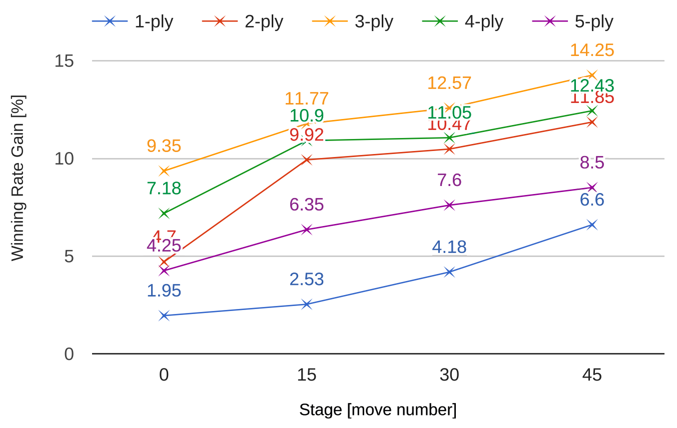

Fig. 5. Gain of winning rate of Conv8+BN when hybridized with Edax-5 for a particular stage of the game. Winning rates are assessed independently against Edax-1.5.

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 also reveal 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.

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 [23], which achieved only 53% on the same data set. Our Conv-8-BN wins against it in 93.9% of games. We obtained a similar winning rate of 93.2% also against the CoC-MAES player [10], 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.[6]

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.

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

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 succeed 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 1% point of the accuracy on the test set, suggesting that the networks we considered here cannot memorize the training data entirely. We may thus conclude 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 FC networks that do not suffer from that limitation; however, they may also be much more costly in training and tend to overfit. In relation to that, an interesting study regarding FC networks for *Othello* with up to three hidden layers was performed in [43].

VIII. Conclusion

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 thus claim 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 paper may be worth extending in several directions. The networks that rely more on the FC layers, mentioned in Section VII, are one such possibility. Logistello [24] 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 nonsymmetric 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] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural Netw.*, vol. 61, pp. 85–117, 2015.

[2] M. Szkulmowski *et al.*, “Oct retinal angiography using neural networks,” *Investigative Ophthalmology Vis. Sci.*, vol. 57, no. 12, pp. 454–454, 2016.

[3] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[4] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2014, pp. 580–587.

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 779–788.

[6] P. Liskowski and K. Krawiec, “Segmenting retinal blood vessels with deep neural networks,” *IEEE Trans. Med. Imag.*, vol. 35, no. 11, pp. 2369–2380, Nov. 2016.

[7] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski, “ViZDoom: A doom-based AI research platform for visual reinforcement learning,” in *Proc. IEEE Conf. Comput. Intell. Games*, Santorini, Greece, Sep. 2016, pp. 341–348. [Online]. Available: http://www.cs.put.poznan.pl/wjaskowski/pub/papers/Kempka2016ViZDoom.pdf

[8] S. van den Dries and M. A. Wiering, “Neural-fitted TD-leaf learning for playing Othello with structured neural networks,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 11, pp. 1701–1713, Nov. 2012. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6291798

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

[10] W. Jaśkowski and M. Szubert, “Coevolutionary CMA-ES for knowledge-free learning of game position evaluation,” *IEEE Trans. Comput. Intell. AI Games*, vol. 8, no. 4, pp. 389–401, Dec. 2016. [Online]. Available: http://www.cs.put.poznan.pl/wjaskowski/pub/papers/Jaskowski2015CoCMAES.pdf

[11] K. Fukushima and S. Miyake, “Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition,” in *Competition and Cooperation in Neural Nets*. New York, NY, USA: Springer-Verlag, 1982, pp. 267–285.

[12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Proc. 25th Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[14] C. Clark and A. Storkey, “Training deep convolutional neural networks to play go,” *Int. Conf. Mach. Learning*, pp. 1766–1774, 2015.

[15] P. Rosenbloom, “A world-championship-level Othello program,” *Artif. Intell.*, vol. 19, no. 3, pp. 279–320, 1982.

[16] R. Smith and B. Gray, “Co-adaptive genetic algorithms: An example in Othello strategy,” in *Proc. 7th Florida AI Res. Symp.*, 1994.

[17] M. Buro, “An evaluation function for Othello based on statistics,” NEC Res. Inst., Princeton, NJ, USA, Tech. Rep. 31, 1997. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.49.7258&rep=rep1&type=pdf

[18] S. M. Lucas and T. P. Runarsson, “Temporal difference learning versus co-evolution for acquiring Othello position evaluation,” in *Proc. IEEE Symp. Comput. Intell. Games*, 2006, pp. 52–59.

[19] E. P. Manning, “Using resource-limited nash memory to improve an Othello evaluation function,” *IEEE Trans. Comput. Intell. AI Games*, vol. 2, no. 1, pp. 40–53, Mar. 2010.

[20] M. Szubert, W. Jaśkowski, and K. Krawiec, “On scalability, generalization, and hybridization of coevolutionary learning: A case study for Othello,” *IEEE Trans. Comput. Intell. AI Games*, vol. 5, no. 3, pp. 214–226, Sep. 2013. [Online]. Available: http://www.cs.put.poznan.pl/mszubert/pub/szubert2013tciaig.pdf

[21] M. V. D. Ree and M. Wiering, “Reinforcement learning in the game of Othello: Learning against a fixed opponent and learning from self-play,” in *Proc. IEEE Symp. Adapt. Dyn. Program. Reinforcement Learn.*, 2013, pp. 108–115. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6614996

[22] W. Jaśkowski, “Systematic n-tuple networks for position evaluation: Exceeding 90% in the Othello league,” Inst. Comput. Sci., Poznan Univ. Technol., Poznań, Poland, Tech. Rep. RA-06/2014, 2014. [Online]. Available: http://arxiv.org/abs/1406.1509

[23] T. Runarsson and S. Lucas, “Preference learning for move prediction and evaluation function approximation in Othello,” *IEEE Trans. Comput. Intell. AI Games*, vol. 6, no. 3, pp. 300–313, Sep. 2014.

[24] M. Buro, “Experiments with multi-ProbCut and a new high-quality evaluation function for Othello,” in *Games in AI Research*, H. van den Herik and H. Iida, Eds. Maastricht, The Netherlands: Maastricht University, 2000, pp. 77–96.

[25] K. Krawiec and M. Szubert, “Learning n-tuple networks for Othello by coevolutionary gradient search,” in *Proc. 13th Annu. Conf. Genetic Evol. Comput.* New York, NY, USA, Jul. 2011, pp. 355–362. [Online]. Available: http://dl.acm.org/citation.cfm?id=2001576.2001626

[26] I. E. Skoulakis and M. G. Lagoudakis, “Efficient reinforcement learning in adversarial games,” in *Proc. IEEE 24th Int. Conf. Tools Artif. Intell.*, Nov. 2012, pp. 704–711. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6495112

[27] S. Samothrakis, S. Lucas, T. Runarsson, and D. Robles, “Coevolving game-playing agents: Measuring performance and intransitivities,” *IEEE Trans. Evol. Comput.*, vol. 17, no. 2, pp. 213–226, Apr. 2013. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6242396

[28] W. Jaśkowski, P. Liskowski, M. Szubert, and K. Krawiec, “Improving coevolution by random sampling,” in *Proc. 15th Annu. Conf. Genetic Evol. Comput.*, Amsterdam, The Netherlands, Jul. 2013, pp. 1141–1148. [Online]. Available: http://www.cs.put.poznan.pl/wjaskowski/pub/papers/jaskowski2013improving.pdf

[29] W. Jaśkowski, “Systematic n-tuple networks for Othello position evaluation,” *ICGA J.*, vol. 37, no. 2, pp. 85–96, Jun. 2014. [Online]. Available: http://www.cs.put.poznan.pl/wjaskowski/pub/papers/jaskowski2014ICGAsystematic.pdf

[30] M. Szubert, W. Jaśkowski, and K. Krawiec, “Learning board evaluation function for Othello by hybridizing coevolution with temporal difference learning,” *Control Cybern.*, vol. 40, no. 3, pp. 805–831, 2011. [Online]. Available: http://www.cs.put.poznan.pl/wjaskowski/pub/papers/szubert2011learning.pdf

[31] W. W. Bledsoe and I. Browning, “Pattern recognition and reading by machine,” in *Proc. Eastern Joint Comput. Conf.*, 1959, pp. 225–232.

[32] S. M. Lucas, “Learning to play Othello with N-tuple systems,” *Aust. J. Intell. Inf. Process. Syst.*, vol. 9, no. 4, pp. 1–20, 2007.

[33] M. Buro, “Experiments with multi-probcut and a new high-quality evaluation function for Othello,” in *Games in AI Research*. pp. 77–96, 1997.

[34] V. L. Allis, “Searching for solutions in games and artificial intelligence,” Ph.D. dissertation, , Univ. Limburg, Maastricht, The Netherlands, 1994.

[35] Y. Bengio, “Learning deep architectures for AI,” *Found. Trends Mach. Learn.*, vol. 2, no. 1, pp. 1–127, 2009.

[36] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, “Striving for simplicity: The all convolutional net,” *arXiv preprint arXiv:1412.6806*, 2014.

[37] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on Imagenet classification,” in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 1026–1034.

[38] Theano Development Team, “Theano: A Python framework for fast computation of mathematical expressions,” *arXiv e-prints*, vol. arXiv:abs/1605.02688, May 2016. [Online]. Available: http://arxiv.org/abs/1605.02688

[39] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *Int. Conf. Mach. Learning*, pp. 448–456, 2015.

[40] R. K. Srivastava, K. Greff, and J. Schmidhuber, “Highway networks,” *arXiv preprint arXiv:1505.00387*, 2015.

[41] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, “Densely connected convolutional networks,” in *Proc. IEEE Conf. Comput. Vision Pattern Recog.*, vol. 1, no. 2, 2017, p. 3.

[42] S. Zagoruyko and N. Komodakis, “Wide residual networks,” *arXiv preprint arXiv:1605.07146*, 2016.

[43] K. J. Binkley, K. Seehart, and M. Hagiwara, “A study of artificial neural network architectures for Othello evaluation functions,” *Inf. Media Technol.*, vol. 2, no. 4, pp. 1129–1139, 2007.

Paweł Liskowski received the B.Eng. and M.Sc. degrees in computing science from the Institute of Computing Science, Poznań University of Technology, Poznań, Poland, in 2011 and 2012, respectively, where he is currently working toward the Ph.D. degree in computer science at the Laboratory of Intelligent Decision Support Systems.

He is currently a Research Assistant at the Laboratory of Intelligent Decision Support Systems, Institute of Computing Science, Poznań University of Technology. His research focuses on artificial intelligence with a focus on machine learning, deep learning, and evolutionary computation. Recently, his efforts have been devoted to researching deep neural networks for medical imaging and program synthesis. He also actively works on coevolution for test-based problems and genetic programming.

Wojciech Jaśkowski received the Ph.D. degree in computing science from the Institute of Computing Science, Poznań University of Technology, Poznań, Poland, in 2011.

He is currently a Research Scientist at NNAISENSE, Lugano, Switzerland. He is the author of more than 30 publications in computational intelligence. His research interests include (deep) reinforcement learning, competitive coevolution, and learning strategies for games, but has done work in evolutionary computations, genetic programming, and visual learning.

Krzysztof Krawiec (M'06) received the Ph.D. and Habilitation degrees in computing science from Poznań University of Technology, Poznań, Poland, in 2000 and 2004, respectively.

He is currently an Associate Professor at Poznań University of Technology. His recent work includes: semantics and program behavior in genetic programming; coevolutionary algorithms and test-based problems; evolutionary computation for machine learning, primarily for learning game strategies and for synthesis of pattern recognition systems; and modeling of complex phenomena using genetic programming. He is the author of more than 100 publications on the above-mentioned and related topics, including *Evolutionary Synthesis of Pattern Recognition Systems* (2005).

Dr. Krawiec is an Associate Editor for *Genetic Programming and Evolvable Machines*, and the President of the Polish Chapter of IEEE Computational Intelligence Society for the term 2013–2014.