

---

# Transfer of Fully Convolutional Policy-Value Networks Between Games and Game Variants

---

Dennis J. N. J. Soemers<sup>\* 1</sup> Vegard Mella<sup>2</sup> Éric Piette<sup>1</sup> Matthew Stephenson<sup>1</sup> Cameron Browne<sup>1</sup>  
Olivier Teytaud<sup>2</sup>

## Abstract

In this paper, we use fully convolutional architectures in AlphaZero-like self-play training setups to facilitate transfer between variants of board games as well as distinct games. We explore how to transfer trained parameters of these architectures based on shared semantics of channels in the state and action representations of the Ludii general game system. We use Ludii’s large library of games and game variants for extensive transfer learning evaluations, in zero-shot transfer experiments as well as experiments with additional fine-tuning time.

## 1. Introduction

AlphaGo (Silver et al., 2016) and its successors (Silver et al., 2017; 2018) have inspired a significant amount of research (Anthony et al., 2017; Tian et al., 2019; Morandini et al., 2019; Wu, 2019; Cazenave et al., 2020; Cazenave, 2020) on combinations of self-play, Monte-Carlo Tree Search (MCTS) (Kocsis & Szepesvári, 2006; Coulom, 2007; Browne et al., 2012) and Deep Learning (LeCun et al., 2015) for automated game-playing. Originally, AlphaGo used distinct value and policy networks, each of which have convolutional layers (LeCun et al., 1989) followed by fully-connected layers. Silver et al. (2017) demonstrated that the use of residual blocks (He et al., 2016), alongside merging the policy and value networks into a single network with two output heads, significantly improved playing strength in AlphaGo Zero. Other modifications to neural network architectures were also explored in subsequent research (Wu, 2019; Cazenave, 2020).

In the majority of prior research, spatial structures present in the state-based inputs for board games are exploited by the

inductive bias of convolutional layers, but the policy head – which has one output for every distinct possible move in a board game – is preceded by one or more fully-connected layers which do not leverage any spatial semantics. Various architectures that also account for spatial semantics in outputs have been proposed in computer vision literature (Ronneberger et al., 2015; Shelhamer et al., 2017), and in the context of games can also handle changes in board size (Lin et al., 2014; Wu, 2019; Cazenave et al., 2020).

The primary contribution of this paper is an approach for transfer learning between variants of games, as well as distinct games. We use fully convolutional networks with global pooling from the Polygames framework (Cazenave et al., 2020) for their ability to handle changes in spatial dimensions during transfer. We focus on transfer between games implemented in the Ludii general game system (Browne et al., 2020; Piette et al., 2020). Its consistent state and action representations (Piette et al., 2021) and game-independent manner of constructing tensor representations for states and actions (Soemers et al., 2021) enables the identification of shared semantics between the non-spatial dimensions (channels) of inputs and outputs for different (variants of) games. This facilitates transfer of trained parameters of fully convolutional networks. Despite previous publications of benchmarks (Nichol et al., 2018), transfer learning in games has remained a challenging problem with limited successes. We propose that Ludii’s large library of board games can be used as a new benchmark for transfer learning in games, and provide extensive baseline results that include various cases of successful zero-shot transfer and transfer with fine-tuning, for transfer between variants of board games as well as distinct board games. Our experiments include transfer between domains with differences in board sizes, board shapes, victory conditions, piece types, and other aspects – many of these have been recognised as important challenges for learning in games (Marcus, 2018).

## 2. Background

This section first provides background information on the implementation of AlphaZero-like (Silver et al., 2018) self-play training processes for games in Polygames (Cazenave

---

<sup>\*</sup>Work done during an internship at Facebook AI Research.

<sup>1</sup>Department of Data Science and Knowledge Engineering, Maastricht University, Maastricht, the Netherlands <sup>2</sup>Facebook AI Research, Paris, France. Correspondence to: Dennis Soemers <dennis.soemers@maastrichtuniversity.nl>.

et al., 2020), and the architectures of neural networks used for this purpose. Secondly, we briefly describe tensor representations in the Ludii general game system (Browne et al., 2020; Piette et al., 2020; Soemers et al., 2021), which we use for all transfer learning experiments. Finally, we discuss background information on transfer learning in games.

### 2.1. Learning to Play Games in Polygames

Similar to AlphaZero (Silver et al., 2018), game-playing agents in Polygames (Cazenave et al., 2020) use a combination of MCTS and deep neural networks (DNNs). Experience for training is generated in the form of self-play games between MCTS agents that are guided by the DNN. Given a tensor representation of an input state  $s$ , the DNN outputs a value estimate  $V(s)$  of the value of that state, as well as a discrete probability distribution  $\mathbf{P}(s) = [P(s, a_1), P(s, a_2), \dots, P(s, a_n)]$  over an action space of  $n$  distinct actions. Both of these outputs are used to guide the MCTS-based tree search. The outcomes (typically losses, draws, or wins) of self-play games are used as training targets for the value head (which produces  $V(s)$  outputs), and the distribution of visit counts to children of the root node by the tree search process is used as a training target for the policy head (which produces  $\mathbf{P}(s)$  outputs).

For board games, input states  $s$  are customarily represented as three-dimensional tensors of shape  $(C, H, W)$ , where  $C$  denotes a number of channels,  $H$  denotes the height of a 2D playable area (e.g., a game board), and  $W$  denotes the width. The latter two are interpreted as spatial dimensions by convolutional neural networks. It is typically assumed that the complete action space can be feasibly enumerated in advance, which means that the shape of  $\mathbf{P}(s)$  output tensors can be constructed such that every possibly distinct action  $a$  has a unique, matching scalar  $P(s, a)$  for any possible state  $s$  in the policy head. A DNN first produces logits  $L(s, a)$  for all actions  $a$ , which are transformed into probabilities using a softmax after masking out any actions that are illegal in  $s$ .

In some general game systems, it can be difficult or impossible to guarantee that there will never be multiple different actions that share a single output in the policy head (Soemers et al., 2021) without manually incorporating additional game-specific domain knowledge. We say that distinct actions are *aliased* if they are represented by a single, shared position in the policy head’s output tensor. In Polygames, the MCTS visit counts of aliased actions are summed up to produce a single shared training target for the corresponding position in the policy head. In the denominator of the softmax, we only sum over the distinct logits that correspond to legal actions (i.e., logits for aliased actions are not counted more than once). All aliased actions  $a$  receive the same prior probability  $P(s, a)$  to bias the tree search – because the DNN cannot distinguish between them – but the tree

search itself can still distinguish between them.

### 2.2. The Ludii General Game System

Ludii (Browne et al., 2020; Piette et al., 2020) is a general game system with over 500 built-in games, many of which support multiple variants with different board sizes, board shapes, rulesets, etc. It automatically constructs suitable object-oriented state and action representations for any game described in its game description language, and these can be converted into tensor representations in a consistent manner without the need for additional game-specific engineering effort (Soemers et al., 2021). All games in Ludii are modelled as having one or more “containers”, which can be viewed as areas with spatial semantics (such as boards) that contain relevant elements of game states and positions that are affected by actions. This means that all games in Ludii are compatible with fully convolutional architectures.

### 2.3. Transfer Learning in Games

AlphaZero-like training approaches have produced superhuman agents for a variety of board games (Silver et al., 2018; Cazenave et al., 2020), and hence been shown to have fairly general applicability, but models are generally trained from scratch for every distinct game. Transfer learning (Taylor & Stone, 2009; Lazaric, 2012; Zhu et al., 2020) may allow for significant savings in computation costs by transferring trained parameters from a *source domain* (i.e., a game that we train on first) to one or more *target domains* (i.e., variants of the source game or different games altogether).

To the best of our knowledge, research on transfer learning between distinct games has been fairly limited. Kuhlmann & Stone (2007) investigated the automated discovery of certain types of relations between source and target games, and transferred trained value functions in specific ways for specific relations. Banerjee & Stone (2007) proposed to transfer value functions based on game-independent features of a game tree’s shape. Both approaches were evaluated, and the former is also restricted to, games defined in the Stanford Game Description Language (Love et al., 2008).

In this paper, we focus on transferring complete policy-value networks – with policy as well as value heads – as they are commonly used in modern game AI literature, using the Ludii general game system to provide easy access to a large and diverse number of games and game variants. Crucially, the transfer of trained parameters for a network with a policy head requires the ability to construct a mapping between action spaces of source and target domains, which we propose a method for. This is in contrast to approaches that only transfer value functions, which also only require the ability to create a mapping between state spaces.

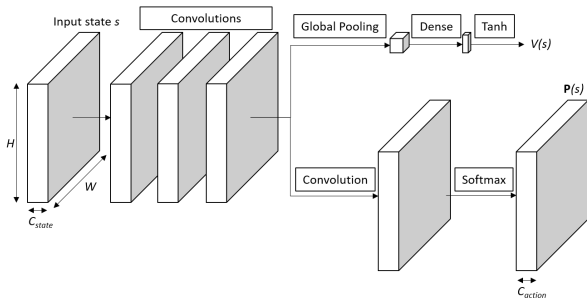


Figure 1. Example of a fully convolutional architecture for game playing. Input states  $s$  are provided as tensors of shape  $(C_{state}, H, W)$ . All convolutions construct hidden representations of shape  $(C_{hidden}, H, W)$ , where  $C_{hidden}$  may differ from  $C_{state}$ . The policy output  $\mathbf{P}(s)$  is a tensor of shape  $(C_{action}, H, W)$ . The value output  $V(s)$  is a scalar. Global pooling is used to reduce the dimensionality for the  $V(s)$  output in a manner that can transfer to different board sizes (Wu, 2019; Cazenave et al., 2020).

## 2.4. Fully Convolutional Architectures

Following Subsection 2.1, we assume that game states  $s$  are represented as tensors of shape  $(C_{state}, H, W)$ , and that the action space can be represented by a shape  $(C_{action}, H, W)$  – we may think of actions as being represented by an action channel, and coordinates in the 2D space of  $H$  rows and  $W$  columns. A network’s policy head therefore also has a shape of  $(C_{action}, H, W)$ . We focus on 2-player zero-sum games, which means that a single scalar suffices as output for the value head. Note that different games may have different numbers of channels and different values for  $H$  and  $W$ .

It is common to use architectures that first process input states  $s$  using convolutional layers, but at the end use one or more non-convolutional layers (such as fully-connected layers) preceding the outputs (Anthony et al., 2017; Silver et al., 2017; 2018). This leads to networks that cannot handle changes in the spatial dimensions (i.e., changes in  $H$  or  $W$ ), and have no inductive biases that leverage any spatial structure that may be present in the action representation. Fully convolutional architectures with global pooling, such as those implemented in Polygames (Cazenave et al., 2020), can address both of those concerns. In particular the ability to handle changes in spatial dimensions is crucial for transfer between distinct games or game variants, which is why we restrict our attention to them in the majority of this paper.

A simplified version of such an architecture is depicted in Figure 1. Note that some details, such as the use of ReLU activations (Nair & Hinton, 2010), batch normalization (Ioffe & Szegedy, 2015), and residual connections (He et al., 2016) have been omitted for brevity. Full source code for Polygames (Cazenave et al., 2020), including its

architectures, is available online.<sup>1</sup>

## 3. Transferring Parameters Between Games

The fully convolutional architectures as described in the previous section allow for DNNs trained in a source task  $\mathcal{S}$  to be directly used in any target task  $\mathcal{T}$  if it only has different values for one or both of the spatial dimensions  $H$  and  $W$ . However, it is also possible that different tasks have different numbers of channels  $C_{state}$  or  $C_{action}$  for state or action representations (or the same number of channels, but significantly different semantics for those channels). This section describes how we identify channels – for actions as well as states – that are likely to be semantically “equivalent” for *any* pair of games in Ludii’s tensor representations (Soemers et al., 2021), and how to transfer trained parameters accordingly. We use a relatively simplistic, binary notion of equivalence; a pair of a channel in  $\mathcal{S}$  and a channel in  $\mathcal{T}$  will either be considered to have identical semantics, or completely different semantics. Furthermore, we only consider the raw data that is encoded by a channel, and do not account for any differences in game rules in this notion of equivalence. For example, the two channels that encode presence of the two players’ pieces in many games such as *Hex*, *Go*, *Tic-Tac-Toe*, etc., are considered to have identical semantics. A link to all the source code for identifying these mappings will be provided after double-blind peer review.

### 3.1. Mapping State Channels

For the majority of channels in Ludii’s state tensor representations (Soemers et al., 2021), semantically equivalent channels are straightforward to identify. For example, a channel that encodes whether player  $p$  is the current player, or whether a position was the destination of the previous move in a source domain  $\mathcal{S}$ , will always have a semantically equivalent channel that encodes identical data and can easily be identified as such in a target domain  $\mathcal{T}$ . These cases are listed in detail in Appendix A of the supplementary material. The first non-trivial case is that of binary channels which encode, for every pair of spatial coordinates, whether or not the corresponding position exists in a container represented by that channel. For example, the game of *Shogi* in Ludii has three separate containers; a large container for the game board, and two smaller containers representing player “hands”, which hold captured pieces. These different containers are assigned different sections of the 2D space, and every container has a binary channel that tells the DNN which positions exist in which containers. In all the built-in games available in Ludii, containers are ordered in a “semantically consistent” manner; the first container is always the main game board, always followed by player hands (if

<sup>1</sup><https://github.com/facebookincubator/Polygames>

there are any), etc. Therefore, we use the straightforward approach of mapping these container-based channels simply in the order in which they appear. It may be possible to remove the reliance on such domain knowledge with techniques that automatically analyse the rules of a game in more detail (Kuhlmann & Stone, 2007; Bou Ammar, 2013; Bou Ammar et al., 2014), but is outside the scope of this work.

The second case that warrants additional explanation is that of binary channels that encode the presence of pieces; for every distinct piece type that is defined in a game, Ludii creates a binary channel that indicates for every possible position whether or not a piece of that type is placed in that position. Many games (such as *Hex*, *Go*, *Tic-Tac-Toe*, etc.) only have a single piece type per player, and for these we could easily decide that a channel indicating presence of pieces of Player  $p$  in one game is semantically equivalent to a channel indicating presence of pieces of the same player in another game. For cases with more than a single piece type per player, we partially rely on an unenforced convention that pieces in built-in games of Ludii tend to be named consistently across closely-related games (e.g., similar piece type names are used in many variants of *Chess*). If we find an exact name match for piece types between  $\mathcal{S}$  and  $\mathcal{T}$ , we treat the corresponding channels as semantically equivalent. Otherwise, for any piece type  $j$  in  $\mathcal{T}$ , we loop over all piece types in  $\mathcal{S}$ , and compute the Zhang-Shasha tree edit distances (Zhang & Shasha, 1989) between the trees of “ludemes” that describe the rules for the piece types in Ludii’s game description language (Piette et al., 2020).

### 3.2. Transferring State Channel Parameters

For the purpose of determining how to transfer parameters that were trained in  $\mathcal{S}$  to a network that can play  $\mathcal{T}$ , we make the assumption that  $\mathcal{S}$  and  $\mathcal{T}$  are *exactly the same game, but with different state tensor representations*; some state channels may have been added, removed, or shuffled around. We make this assumption because it enables us to build a more rigorous notion of what it means to “correctly” transfer parameters without accounting for differences in rules, optimal strategies, or value functions. In practice,  $\mathcal{S}$  and  $\mathcal{T}$  can end up being different games, and we intuitively still expect this transfer to be potentially beneficial if the games are sufficiently similar, but the notion of “correct” transfer cannot otherwise be made concrete without detailed domain knowledge of the specific games involved.

Let  $s$  denote the tensor representation, of shape  $(C_{state}^{\mathcal{S}}, H, W)$ , for any arbitrary state in  $\mathcal{S}$ . Let  $s'$  denote the tensor representation, of shape  $(C_{state}^{\mathcal{T}}, H, W)$ , for the same state represented in  $\mathcal{T}$  – this must exist by the assumption that  $\mathcal{S}$  and  $\mathcal{T}$  are the same game, modelled in different ways. Let  $h_1^{\mathcal{S}}(s)$  denote the hidden representation obtained by the application of the first convolutional operation on

$s$ , in a network trained on  $\mathcal{S}$ . For brevity we focus on the case used throughout all our experiments, but most if not all of these assumptions can likely be relaxed; a `nn.Conv2d` layer as implemented in PyTorch (Paszke et al., 2019), with  $3 \times 3$  filters, a stride and dilation of 1, and a padding of 1 (such that the spatial dimensions do not change). Let  $\Theta^{\mathcal{S}}$  denote this layer’s tensor of parameters trained in  $\mathcal{S}$ , of shape  $(k_{out}, C_{state}^{\mathcal{S}}, 3, 3)$ , and  $B^{\mathcal{S}}$  – of shape  $(k_{out})$  – the corresponding bias. This leads to  $h_1^{\mathcal{S}}$  having a shape of  $(k_{out}, H, W)$ . Similarly, let  $h_1^{\mathcal{T}}(s')$  denote the first hidden representation in the network after transfer, for the matching state  $s'$  in the new target domain’s representation, with weight and bias tensors  $\Theta^{\mathcal{T}}$  and  $B^{\mathcal{T}}$ .

Under the assumption of source and target games being identical, we could obtain correct transfer by ensuring that  $h_1^{\mathcal{S}}(s) = h_1^{\mathcal{T}}(s')$ . Achieving this would mean that the first convolutional layer would handle any changes in the state representation, and the remainder of the network could be transferred in its entirety and behave as it learned to do in the source domain.  $B^{\mathcal{T}}$  is simply initialised by copying  $B^{\mathcal{S}}$ . Let  $i \cong j$  denote that the  $i^{th}$  channel in a source domain  $\mathcal{S}$  has been determined (as described in Subsection 3.1) to be semantically equivalent to the  $j^{th}$  channel in a target domain  $\mathcal{T}$ . Channels on the left-hand side are always source domain channels, and channels on the right-hand side are always target domain channels. For any channel  $j$  in  $\mathcal{T}$ , if there exists a channel  $i$  in  $\mathcal{S}$  such that  $i \cong j$ , we initialise  $\Theta^{\mathcal{T}}(k, j, \cdot, \cdot) := \Theta^{\mathcal{S}}(k, i, \cdot, \cdot)$  for all  $k$ . If there is no such channel  $i$ , we initialise these parameters using the default approach for initialising untrained parameters (or initialise them to 0 for zero-shot evaluations, where these parameters do not need to remain trainable through backpropagation).

If  $\mathcal{T}$  contains channels  $j$  such that there are no equivalent channels  $i$  in  $\mathcal{S}$ , i.e.  $\{i \mid i \cong j\} = \emptyset$ , we have no transfer to the  $\Theta^{\mathcal{T}}(k, j, \cdot, \cdot)$  parameters. This can be the case if  $\mathcal{T}$  involves new data for which there was no equivalent in the representation of  $\mathcal{S}$ , which means that there was also no opportunity to learn about this data in  $\mathcal{S}$ .

If  $\mathcal{S}$  contained channels  $i$  such that there are no equivalent channels  $j$  in  $\mathcal{T}$ , i.e.  $\nexists j (i \cong j)$ , we have no transfer from the  $\Theta^{\mathcal{S}}(k, i, \cdot, \cdot)$  parameters. This can be the case if  $\mathcal{S}$  involved data that is no longer relevant or accessible in  $\mathcal{T}$ . Not using them for transfer is equivalent to pretending that these channels still are present, but always filled with 0 values.

If  $\mathcal{S}$  contained a channel  $i$  such that there are multiple equivalent channels  $j$  in  $\mathcal{T}$ , i.e.  $|\{j \mid i \cong j\}| > 1$ , we copy a single set of parameters multiple times. This can be the case if  $\mathcal{T}$  uses multiple channels to encode data that was encoded by just a single channel in  $\mathcal{S}$  (possibly with a loss of information). In the case of Ludii’s tensor representations, this only happens when transferring channels representing the presence of piece types from a game  $\mathcal{S}$  with fewer types



of pieces, to a game  $\mathcal{T}$  with more distinct piece types. In the majority of games in Ludii, such channels are “mutually exclusive” in the sense that if a position contains a 1 entry in one of these channels, all other channels in the set are guaranteed to have a 0 entry in the same position. This means that copying the same parameters multiple times can still be viewed as “correct”; for any given position in a state, they are guaranteed to be multiplied by a non-zero value at most once. The only exceptions are games  $\mathcal{T}$  that allow for multiple pieces of distinct types to be stacked on top of each other in a single position, but these games are rare and not included in any of the experiments described in this paper.

If  $\mathcal{T}$  contains a single channel  $j$  such that there are multiple equivalent channels  $i$  in  $\mathcal{S}$ , i.e.  $|\{i \mid i \cong j\}| > 1$ , there is no clear way to correctly transfer parameters without incorporating additional domain knowledge on how the single target channel summarises – likely with a loss of information – multiple source channels. This case never occurs when mapping channels as described in Subsection 3.1.

### 3.3. Mapping Action Channels

Channels in Ludii’s action tensor representations (Soemers et al., 2021) have three broad categories of channels; a channel for pass moves, a channel for swap moves, and one or more channels for all other moves. Channels for pass or swap moves in one domain can easily be classified as being semantically equivalent only to channels for the same type of moves in another domain.

We refer to games where, in Ludii’s internal move representation (Piette et al., 2021), some moves have separate “from” and “to” (or source and destination) positions as *movement games* (e.g. *Amazons*, *Chess*, *Shogi*, etc.), and games where all moves only have a “to” position as *placement games* (e.g. *Go*, *Hex*, *Tic-Tac-Toe*, etc.). In placement games, there is only one more channel to encode all moves that are not pass or swap moves. In movement games, there are 49 additional channels, which can distinguish moves based on any differences in  $x$  and  $y$  coordinates between “from” and “to” positions in  $\{\leq -3, -2, -1, 0, 1, 2, \geq 3\}$ .

If both  $\mathcal{S}$  and  $\mathcal{T}$  are placement games, or if both are movement games, we can trivially obtain one-to-one mappings between all move channels. If  $\mathcal{S}$  is a movement game, but  $\mathcal{T}$  is a placement game, we only treat the source channel that encodes moves with equal “from” and “to” positions as semantically equivalent (in practice, this channel remains unused in the vast majority of movement games, which means that we effectively get no meaningful transfer for moves due to the large discrepancy in movement mechanisms). If  $\mathcal{S}$  is a placement game, and  $\mathcal{T}$  is a movement game, we treat the sole movement channel from  $\mathcal{S}$  as being semantically equivalent to *all* the movement channels in  $\mathcal{T}$ .

### 3.4. Transferring Action Channel Parameters

Similar to Subsection 3.2, we make the assumption that source and target games  $\mathcal{S}$  and  $\mathcal{T}$  are identical games, but with *different action tensor representations*, such that we can define a clear notion of correctness for transfer. Let  $s$  and  $a$  denote any arbitrary state and action in  $\mathcal{S}$ , such that  $a$  is legal in  $s$ , and let  $s'$  and  $a'$  denote the corresponding representations in  $\mathcal{T}$ . We assume that the state representations have been made equivalent through transfer of parameters for the first convolutional layer, as described in Subsection 3.2. Let  $h_n(s)$  be the hidden representation that, in a fully convolutional architecture trained in  $\mathcal{S}$ , is transformed into a tensor  $L(h_n(s))^{\mathcal{S}}$  of shape  $(C_{action}^{\mathcal{S}}, H, W)$  of logits. Similarly, let  $L(h_n(s'))^{\mathcal{T}}$  of shape  $(C_{action}^{\mathcal{T}}, H, W)$  denote such a tensor of logits in the target domain. By assumption, we have that  $h_n(s) = h_n(s')$ .

If the action representations in  $\mathcal{S}$  and  $\mathcal{T}$  are equally powerful in their ability to distinguish actions, we can define a notion of correct transfer of parameters by requiring the transfer to ensure that  $L(h_n(s))^{\mathcal{S}} = L(h_n(s'))^{\mathcal{T}}$  after accounting for any shuffling of channel indices. If we have one-to-one mappings for all action channels, this can be easily achieved by copying parameters of the final convolutional operation in a similar way as for state channels (see Subsection 3.2).

If the action representation of  $\mathcal{T}$  can distinguish actions from each other that cannot be distinguished in  $\mathcal{S}$ , we have a reduction in *move aliasing* (see Subsection 2.1). This happens when transferring from placement games to movement games. Since these actions were treated as identical when training in  $\mathcal{S}$ , it is sensible to continue treating them as identical and give them equal probabilities in  $\mathcal{T}$ . This is achieved by mapping a single source channel to multiple target channels, and copying trained parameters accordingly.

If the action representation of  $\mathcal{S}$  could distinguish actions from each other that can no longer be distinguished in  $\mathcal{T}$ . This happens when transferring from movement games to placement games. As described in the previous subsection, we handle this case conservatively by only allowing transfer from a single source channel – the one that is arguably the “most similar” – and discarding all other parameters.

## 4. Experiments

This section discusses experiments used to evaluate the performance of fully convolutional architectures, as well as several transfer learning experiments between variants of games and between distinct games. We used the training code from Polygames (Cazenave et al., 2020). For transfer learning experiments, we used games as implemented in Ludii v1.1.6 (Browne et al., 2020). Appendix B of the supplementary material provides details on hyperparameters.

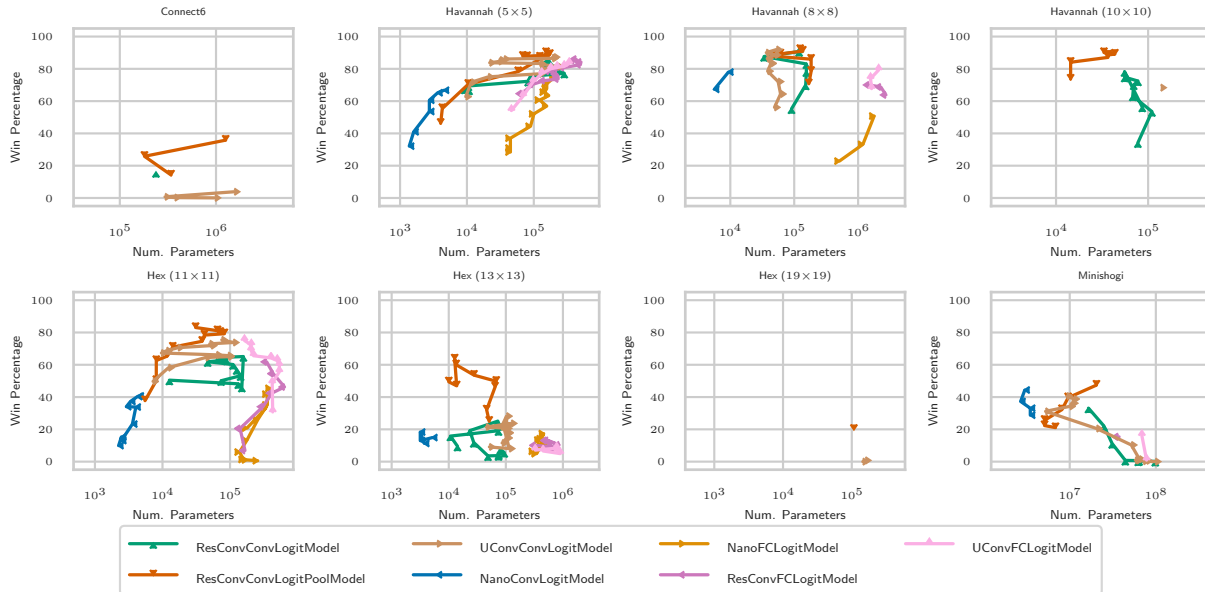


Figure 2. Win percentages of a trained MCTS with 40 iterations/move vs. UCT with 800 iterations/move for a variety of architectures. Nano: shallow architectures. ConvConv: deep fully convolutional. ConvFC: fully connected layers after convolutional ones. Pool: adding global pooling; U: adding U-connections (Ronneberger et al., 2015). Deep is better than shallow, U-nets are slightly better than their classical counterparts, and deep nets are greatly improved by (i) using fully convolutional policy heads (ii) using global pooling.

#### 4.1. Evaluation of Fully Convolutional Architectures

We selected a variety of board games as implemented in Polygames (Cazenave et al., 2020), and trained networks of various sizes and architectures, using 24 hours on 8 GPUs and 80 CPU cores per model. Models of various sizes – measured by the number of trainable parameters – have been constructed by randomly drawing choices for hyperparameters such as the number of layers, blocks, and channels for hidden layers. After training, we evaluated the performance of every model by recording the win percentage of an MCTS agent using 40 iterations per move with the model, versus a standard untrained UCT (Browne et al., 2012) agent with 800 iterations per move. These win percentages are depicted in Figure 2. In the majority of cases, ResConvConvLogitPoolModel – a fully convolutional model with global pooling – is among the strongest architectures. Fully convolutional models generally outperform ones with dense layers, and models with global pooling generally outperform those without global pooling. This suggests that using such architectures can be beneficial in and of itself, and their use to facilitate transfer learning does not lead to a sacrifice in baseline performance.

#### 4.2. Evaluation of Transfer Learning

All transfer learning experiments discussed below used the ResConvConvLogitPoolModelV2 architecture from Polygames (Cazenave et al., 2020). All models were trained

for 20 hours on 8 GPUs and 80 CPU cores, using 1 server for training and 7 clients for the generation of self-play games.

##### 4.2.1. TRANSFER BETWEEN GAME VARIANTS

We selected a set of nine different board games, as implemented in Ludii, and for each of them consider a few different variants. The smallest number of variants for a single game is 2, and the largest number of variants for a single game is 6. In most cases, the different variants are simply different board sizes. For example, we consider *Gomoku* played on  $9 \times 9$ ,  $13 \times 13$ ,  $15 \times 15$ , and  $19 \times 19$  boards as four different variants of *Gomoku*. We also include some cases where board shapes change (i.e., *Breakthrough* played on square boards as well as hexagonal boards), “small” changes in rules (i.e., *Broken Line* played with a goal line length of 3, 4, 5, or 6), and “large” changes in rules (i.e., *Hex* with the standard win condition and *Misère Hex* with an inverted win condition). Details on all the games and game variants used are provided in Appendix C of the supplementary material.

We trained a separate model for every variant of each of these games, and within each game, transferred models from all variants to all other variants. We evaluate zero-shot transfer performance for a source domain  $\mathcal{S}$  and target domain  $\mathcal{T}$  by reporting the win percentage of the model trained in  $\mathcal{S}$  against the model that was trained in  $\mathcal{T}$ , over 300 evaluation games per  $(\mathcal{S}, \mathcal{T})$  tuple running in  $\mathcal{T}$ .

Figure 3 and Figure 4 depict scatterplots of all these zero-

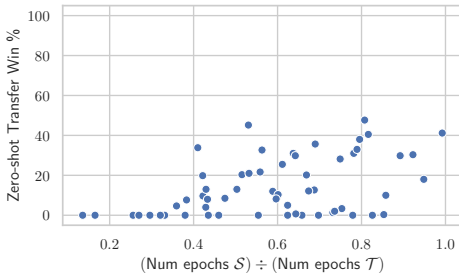


Figure 3. Zero-shot transfer from  $\mathcal{S}$  with larger board sizes to  $\mathcal{T}$  with smaller board sizes, for several board games and board sizes.

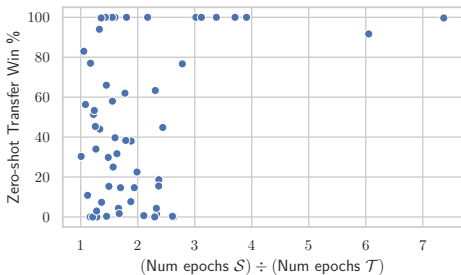


Figure 4. Zero-shot transfer from  $\mathcal{S}$  with smaller board sizes to  $\mathcal{T}$  with larger board sizes, for several board games and board sizes.

shot transfer evaluations for the cases where  $\mathcal{S}$  has a larger board size than  $\mathcal{T}$ , and where  $\mathcal{S}$  has a smaller board size than  $\mathcal{T}$ , respectively. The  $y$ -axis represents win percentages of the transferred model against the baseline model, and the  $x$ -axis represents the ratio of the number of training epochs of the source model to the number of training epochs of the target model. Models trained on larger board sizes tend to have a lower number of training epochs for three reasons; the neural network passes are more expensive, the game logic in Ludii is more expensive, and episodes often tend to last for a higher number of turns when played on larger boards. Hence, all data points in Figure 3 have a ratio  $\leq 1.0$ , and all data points in Figure 4 have a ratio  $\geq 1.0$ .

When transferring a model that was trained on a large board to a small board (Figure 3), zero-shot win percentages tend to be below 50%, but frequently still above 0%. This suggests that training on a larger board than the one we intend to play on does not outperform simply training on the correct board directly, but it often still produces a capable model that can win a non-trivial number of games. When transferring a model that was trained on a small board to a large board (Figure 4), we also frequently obtain win percentages above 50%, even reaching up to 100%, against models that were trained directly on the board used for evaluation.

Zero-shot transfer between variants with larger differences, such as modified board shapes or changes in win conditions, only leads to win percentages significantly above 0% in a

few cases. These results, as well as more detailed tables, are presented in Appendix D of the supplementary material.

For every model transferred from a source domain  $\mathcal{S}$  to a target domain  $\mathcal{T}$  as described above, we train it under identical conditions as the initial training runs – for an additional 20 hours – to evaluate the effect of using transfer for initialisation of the network. Figure 5 depicts scatterplots of win percentages for four distinct cases; transfer to variants with smaller board sizes, with larger board sizes, with different board shapes, and with different win conditions. There are many cases of win percentages close to 50%, which can be interpreted as cases where transfer neither helped nor hurt final performance, and many cases with higher win percentages – which can be interpreted as cases where transfer increased final performance in comparison to training from scratch. We observe a small number of cases, especially when  $\mathcal{S}$  has a larger board than  $\mathcal{T}$ , or has different win conditions, in which there is clear negative transfer (Zhang et al., 2020) and the final performance is still closer to 0% even after fine-tuning on  $\mathcal{T}$ . More detailed results are provided in Appendix E of the supplementary material.

#### 4.2.2. TRANSFER BETWEEN DIFFERENT GAMES

For our final set of experiments, we collected four sets of games, and within each set carried out similar experiments as described above – this time transferring models between distinct games, rather than game variants. The first set consists of six different **Line Completion Games**; in each of these games the win condition is to create a line of  $n$  pieces, but the games differ in aspects such as the value of  $n$ , board sizes and shapes, move rules, loss conditions, etc. We evaluate transfer from each of those games, to each of these games. The second set consists of four **Shogi Variants**: we include *Hasami Shogi*, *Kyoto Shogi*, *Minishogi*, and *Shogi*, and evaluate transfer from and to each of them. In the third set we evaluate transfer from each of four variants of *Broken Line*, to each of the six line completion games. *Broken Line* is a custom-made line completion game where only diagonal lines count towards the win condition, whereas the standard line completion games allow for orthogonal lines. In the fourth set, we evaluate transfer from each of five variants of *Diagonal Hex*, to each of six variants of *Hex*. *Diagonal Hex* only considers diagonal connections for the win condition of *Hex*, whereas the *Hex* variants only consider orthogonal connections. Appendix C of the supplementary material provides more details on all the games and variants.

In most cases, zero-shot win percentages for models transferred between distinct games are close to 0%. We observe some success with zero-shot win percentages greater than 30% for transfer from several different line completion games to *Connect 6*, zero-shot win percentages between 20% and 50% for transfer from three different Shogi vari-

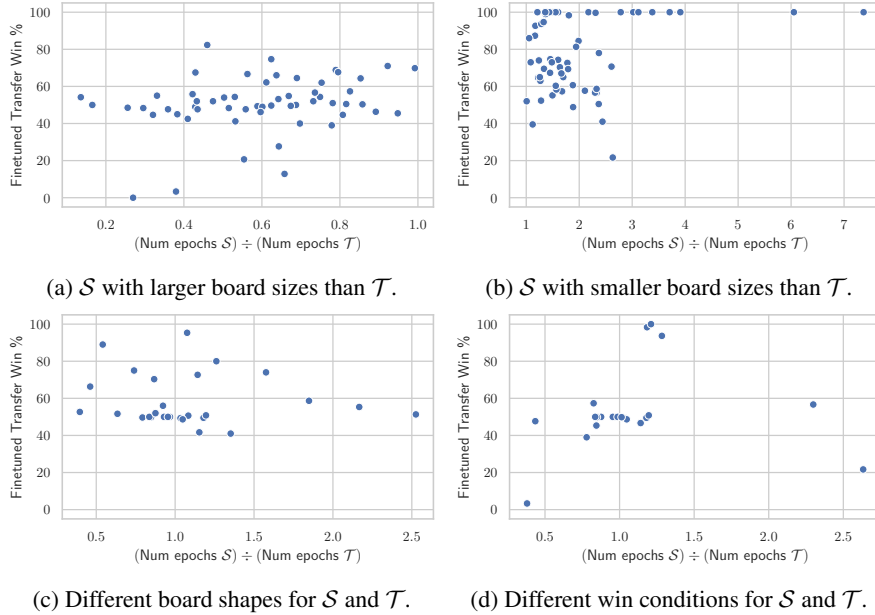


Figure 5. Win percentages of models trained on  $\mathcal{S}$  and subsequently fine-tuned on  $\mathcal{T}$ , against models trained only on  $\mathcal{T}$  – evaluated on  $\mathcal{T}$ .

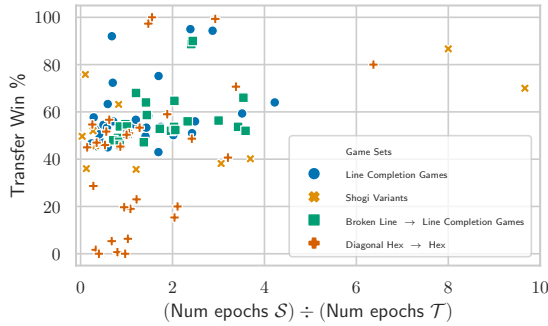


Figure 6. Win percentages of models that were trained in  $\mathcal{S}$ , transferred to  $\mathcal{T}$ , and fine-tuned in  $\mathcal{T}$ , evaluated in  $\mathcal{T}$  against models trained directly in  $\mathcal{T}$ .  $\mathcal{S}$  and  $\mathcal{T}$  are different games.

ants to *Hasami Shogi*, as well as a win percentage of 97% for zero-shot transfer from *Minishogi* to *Shogi*. Appendix F of the supplementary material contains more detailed results.

Figure 6 depicts win percentages for transferred models after they received an additional 20 hours of fine-tuning time on  $\mathcal{T}$ . Most notably for various cases of transfer from *Diagonal Hex* to *Hex*, there is a high degree of negative transfer, with many win percentages far below 50% even after fine-tuning. It may be that the differences in connectivity rules are too big to allow for consistently successful transfer. The difficulties in transfer may also be due to large differences in the distributions of outcomes, resulting in large mismatches for the value head; ties are a common result in some variants of *Diagonal Hex*, but impossible in *Hex*. In particular for line completion games, transfer appears to be generally

successful; there are no severe cases of negative transfer, and a significant amount with strong positive transfer.

### 5. Conclusions

In this paper, we explored the ability to transfer fully convolutional networks with global pooling, trained using an AlphaZero-like approach, between variants of board games, and distinct board games. Firstly, we compared the performance of such architectures to various others outside of a transfer learning setting, and demonstrated them to be among the top-performing architectures in a variety of board games: fully convolutional nets bring a strong improvement in particular for large board games, and global pooling and U-nets provide slight improvements (Fig. 1). Secondly, we explored how to transfer parameters of such networks between different games with different state and action representations in the Ludii general game system. We evaluated zero-shot transfer performance, as well as the performance of transferred models after additional fine-tuning in the target domain, for a wide variety of source and target games and game variants in Ludii. We find several cases where even zero-shot transfer is highly successful – especially when transferring from smaller games to larger ones. We also observe a significant number of cases of beneficial transfer after fine-tuning, even when the source and target domains have more significant differences than just changes in board size or shape (see Figure 6). Finally, we find some cases with clear negative transfer, even after fine-tuning, which point to avenues for future research.



## Acknowledgements

The authors would like to thank Nicolas Usunier for comments on an earlier version of this work. This work was partially supported by the European Research Council as part of the Digital Ludeme Project (ERC Consolidator Grant #771292), led by Cameron Browne at Maastricht University’s Department of Data Science and Knowledge Engineering.

## References

- Anthony, T., Tian, Z., and Barber, D. Thinking fast and slow with deep learning and tree search. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 5360–5370. Curran Associates, Inc., 2017.
- Banerjee, B. and Stone, P. General game learning using knowledge transfer. In *The 20th International Joint Conference on Artificial Intelligence*, pp. 672–677, 2007.
- Bou Ammar, H. *Automated transfer in reinforcement learning*. PhD thesis, Maastricht University, Maastricht, the Netherlands, 2013.
- Bou Ammar, H., Eaton, E., Taylor, M. E., Mocanu, D. C., Driessens, K., Weiss, G., and Tuyls, K. An automated measure of mdp similarity for transfer in reinforcement learning. In *Proceedings of the Interactive Systems Workshop at the American Association of Artificial Intelligence (AAAI)*, pp. 31–37, 2014.
- Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–49, 2012.
- Browne, C., Stephenson, M., Piette, É., and Soemers, D. J. N. J. A practical introduction to the ludii general game system. In Cazenave, T., van den Herik, J., Saffidine, A., and Wu, I.-C. (eds.), *Advances in Computer Games. ACG 2019*, volume 12516 of *Lecture Notes in Computer Science (LNCS)*. Springer, Cham, 2020.
- Cazenave, T. Mobile networks for computer go. *IEEE Transactions on Computational Intelligence and AI in Games*, 2020. To appear.
- Cazenave, T., Chen, Y.-C., Chen, G., Chen, S.-Y., Chiu, X.-D., Dehos, J., Elsa, M., Gong, Q., Hu, H., Khalidov, V., Li, C.-L., Lin, H.-I., Lin, Y.-J., Martinet, X., Mella, V., Rapin, J., Roziere, B., Synnaeve, G., Teytaud, F., Teytaud, O., Ye, S.-C., Ye, Y.-J., Yen, S.-J., and Zagoruyko, S. Polygames: Improved zero learning. *ICGA Journal*, 2020. To appear.
- Coulom, R. Efficient selectivity and backup operators in Monte-Carlo tree search. In van den Herik, H. J., Ciancarini, P., and Donkers, H. H. L. M. (eds.), *Computers and Games*, volume 4630 of *LNCS*, pp. 72–83. Springer Berlin Heidelberg, 2007.
- He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778. IEEE, 2016.
- Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Bach, F. and Blei, D. (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 448–456, 2015.
- Kocsis, L. and Szepesvári, C. Bandit based Monte-Carlo planning. In Fürnkranz, J., Scheffer, T., and Spiliopoulou, M. (eds.), *Machine Learning: ECML 2006*, volume 4212 of *Lecture Notes in Computer Science (LNCS)*, pp. 282–293. Springer, Berlin, Heidelberg, 2006.
- Kuhlmann, G. and Stone, P. Graph-based domain mapping for transfer learning in general games. In Kok, J., Koronacki, J., Mantaras, R., Matwin, S., Mladenič, D., and Skowron, A. (eds.), *Machine Learning: ECML 2007*, volume 4071 of *Lecture Notes in Computer Science (LNCS)*, pp. 188–200. Springer, Berlin, Heidelberg, 2007.
- Lazaric, A. Transfer in reinforcement learning: a framework and a survey. In Wiering, M. and van Otterlo, M. (eds.), *Reinforcement Learning*, volume 12 of *Adaptation, Learning, and Optimization*, pp. 143–173. Springer, Berlin, Heidelberg, 2012.
- LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.
- LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *Nature*, 521(7553):436–444, 2015.
- Lin, M., Chen, Q., and Yan, S. Network in network. *CoRR*, abs/1312.4400, 2014. URL <https://arxiv.org/abs/1312.4400>.
- Love, N., Hinrichs, T., Haley, D., Schkufza, E., and Genereth, M. General game playing: Game description language specification, 2008.
- Marcus, G. Innateness, alphazero, and artificial intelligence. *CoRR*, abs/1801.05667, 2018. URL <https://arxiv.org/abs/1801.05667>.

- Morandini, F., Amato, G., Gini, R., Metta, C., Parton, M., and Pascutto, G.-C. SAI: a sensible artificial intelligence that plays Go. In *Proceedings of the 2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019.
- Nair, V. and Hinton, G. Rectified linear units improve restricted boltzmann machines. In Fürnkranz, J. and Joachims, T. (eds.), *Proceedings of the 27th International Conference on Machine Learning*, pp. 807–814. Omnipress, 2010.
- Nichol, A., Pfau, V., Hesse, C., Klimov, O., and Schulman, J. Gotta learn fast: A new benchmark for generalization in rl. *CoRR*, 2018. URL <https://arxiv.org/abs/1804.03720>.
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.
- Piette, É., Soemers, D. J. N. J., Stephenson, M., Sironi, C. F., Winands, M. H. M., and Browne, C. Ludii – the ludemic general game system. In Giacomo, G. D., Catala, A., Dilkina, B., Milano, M., Barro, S., Bugarín, A., and Lang, J. (eds.), *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI 2020)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pp. 411–418. IOS Press, 2020.
- Piette, É., Browne, C., and Soemers, D. J. N. J. Ludii game logic guide. *CoRR*, abs/2101.02120, 2021. URL <https://arxiv.org/abs/2101.02120>.
- Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In Navab, N., Hornegger, J., Wells, W. M., and Frangi, A. F. (eds.), *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pp. 234–241, Cham, 2015.
- Shelhamer, E., Long, J., and Darrell, T. Fully convolutional networks for semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):640–651, 2017.
- Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.
- Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. Mastering the game of Go without human knowledge. *Nature*, 550:354–359, 2017.
- Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018.
- Soemers, D. J. N. J., Mella, V., Browne, C., and Teytaud, O. Deep learning for general game playing with ludii and polygames. *CoRR*, 2021. URL <https://arxiv.org/abs/2101.09562>.
- Taylor, M. E. and Stone, P. Transfer learning for reinforcement learning domains: A survey. In Mahadevan, S. (ed.), *Journal of Machine Learning Research*, volume 10, pp. 1633–1685, 2009.
- Tian, Y., Ma, J., Gong, Q., Sengupta, S., Chen, Z., Pinkerton, J., and Zitnick, C. L. ELF OpenGo: An analysis and open reimplement of AlphaZero. In *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, pp. 6244–6253, 2019.
- Wu, D. J. Accelerating self-play learning in go. *CoRR*, abs/1902.10565, 2019. URL <http://arxiv.org/abs/1902.10565>.
- Zhang, K. and Shasha, D. Simple fast algorithms for the editing distance between trees and related problems. In *SIAM Journal on Computing*, volume 18, pp. 1245–1262, 1989.
- Zhang, W., Deng, L., Zhang, L., and Wu, D. Overcoming negative transfer: A survey. *CoRR*, abs/2009.00909, 2020. URL <https://arxiv.org/abs/2009.00909>.
- Zhu, Z., Lin, K., and Zhou, J. Transfer learning in deep reinforcement learning: A survey. *CoRR*, abs/2009.07888, 2020. URL <https://arxiv.org/abs/2009.07888>.

## A. Directly Transferable Ludii State Channels

Most state channels in Ludii’s tensor representations (Soemers et al., 2021) are directly transferable between games, in the sense that they encode semantically similar data for any game in which they are present, and can be modelled as channels that always have only values of 0 in any game that they are not present in. These channels are briefly described here:

- A channel encoding the height of a stack of pieces for every position (only present in games that allow for pieces of more than a single piece type to stack on the same position).
- A channel encoding the number of pieces per position (only present in games that allow multiple pieces of the same type to form a pile on the same position).
- Channels encoding the (typically monetary) “amount” value per player.
- Binary channels encoding whether a given player is the current player to move.
- Channels encoding “local state” values per position (for instance used to memorise whether pieces moved to determine the legality of castling in *Chess*).
- A channel encoding whether or not players have swapped roles.
- Channels encoding “from” and “to” positions of the last and second-to-last moves.

## B. Details on Experimental Setup

For all training runs for transfer learning experiments, the following command-line arguments were supplied to the `train` command of Polygames (Cazenave et al., 2020):

- `--num_game 2`: Affects the number of threads used to run games per self-play client process.
- `--epoch_len 256`: Number of training batches per epoch.
- `--batchsize 128`: Batch size for model training.
- `--sync_period 32`: Affects how often models are synced.
- `--num_rollouts 400`: Number of MCTS iterations per move during self-play training.
- `--replay_capacity 100000`: Capacity of replay buffer.
- `--replay_warmup 9000`: Minimum size of replay buffer before training starts.
- `--model_name "ResConvConvLogitPoolModelV2"`: Type of architecture to use (a fully convolutional architecture with global pooling).
- `--bn`: Use of batch normalization (Ioffe & Szegedy, 2015).
- `--nsize 2`: A value of 2 means that hidden convolutional layers each have twice as many channels as the number of channels for the state input tensors.
- `--nb_layers_per_net 6`: Number of convolutional layers per residual block.
- `--nb_nets 10`: Number of residual blocks.
- `--tournament_mode=true`: Use the tournament mode of Polygames to select checkpoints to play against in self-play.
- `--bsfinder_max_bs=800`: Upper bound on number of neural network queries batched together during inference (we used a lower value of 400 to reduce memory usage in *Breakthrough*, *Hasami Shogi*, *Kyoto Shogi*, *Minishogi*, *Shogi*, and *Tobi Shogi*).

All evaluation games in transfer learning experiments were run using the following command-line arguments for the `eval` command of Polygames:

- `--num_actor_eval=1`: Number of threads running simultaneously for a single MCTS search for the agent being evaluated.
- `--num_rollouts_eval=800`: Number of MCTS iterations per move for the agent being evaluated.
- `--num_actor_opponent=1`: Number of threads running simultaneously for a single MCTS search for the baseline agent.
- `--num_rollouts_opponent=800`: Number of MCTS iterations per move for the baseline agent.

Any parameters not listed were left at their defaults in the Polygames implementation.

### C. Details on Games and Game Variants

This section provides additional details on all the games and variants of games used throughout all the experiments described in the paper. A game with name `GameName` is selected in Polygames by providing `--game_name="GameName"` as command-line argument. For games implemented in Ludii, non-default variants are loaded by providing `--game_options "X" "Y" "Z"` as additional command-line arguments, where X, Y, and Z refer to one or more option strings.

#### C.1. Polygames Games

For the evaluation of fully convolutional architectures, we used games as implemented directly in Polygames. Table 1 lists the exact game names used. Note that all versions of *Havannah* and *Hex* included use of the pie rule (or swap rule).

Table 1. Game implementations from Polygames used for evaluation of fully convolutional architectures. The right column shows the names used in command-line arguments.

| <b>Game</b>      | <b>Game Name Argument</b> |
|------------------|---------------------------|
| Connect6         | Connect6                  |
| Havannah (5×5)   | Havannah5pie              |
| Havannah (8×8)   | Havannah8pie              |
| Havannah (10×10) | Havannah10pie             |
| Hex (11×11)      | Hex11pie                  |
| Hex (13×13)      | Hex13pie                  |
| Hex (19×19)      | Hex19pie                  |
| Minishogi        | Minishogi                 |

#### C.2. Ludii Game Variants

For the transfer learning experiments between variants of games, we used nine games – each with multiple variants – as implemented in Ludii: *Breakthrough*, *Broken Line*, *Diagonal Hex*, *Gomoku*, *Hex*, *HeXentafl*, *Konane*, *Pentalath*, and *Yavalath*. For each of these games, Tables 2-10 provide additional details. In each of these tables, the final column lists the number of trainable parameters in the Deep Neural Network (DNN) that is constructed for each game variant, using hyperparameters as described in Appendix B.



---

**Transfer of Fully Convolutional Policy-Value Networks Between Games and Game Variants**

---

Table 2. Details on *Breakthrough* variants. This implementation of Breakthrough is loaded in Polygames using “LudiiBreakthrough.lud” as game name. By default, Breakthrough is played on an 8×8 square board.

| <b>Variant</b> | <b>Options</b>                    | <b>Description</b>  | <b>Num. Params DNN</b> |
|----------------|-----------------------------------|---------------------|------------------------|
| Square6        | "Board Size/6x6" "Board/Square"   | 6×6 square board    | 188,296                |
| Square8        | "Board Size/8x8" "Board/Square"   | 8×8 square board    | 188,296                |
| Square10       | "Board Size/10x10" "Board/Square" | 10×10 square board  | 188,296                |
| Hexagon4       | "Board Size/4x4" "Board/Hexagon"  | 4×4 hexagonal board | 188,296                |
| Hexagon6       | "Board Size/6x6" "Board/Hexagon"  | 6×6 hexagonal board | 188,296                |
| Hexagon8       | "Board Size/8x8" "Board/Hexagon"  | 8×8 hexagonal board | 188,296                |

Table 3. Details on *Broken Line* variants. This implementation of Broken Line is loaded in Polygames using “LudiiBroken Line.lud” as game name.

| <b>Variant</b>  | <b>Options</b>                                      | <b>Description</b>                  | <b>Num. Params DNN</b> |
|-----------------|---|-------------------------------------|------------------------|
| LineSize3Hex    | "Line Size/3"<br>"Board Size/5x5"<br>"Board/hex"    | 5×5 hexagonal board, lines of 3 win | 222,464                |
| LineSize4Hex    | "Line Size/4"<br>"Board Size/5x5"<br>"Board/hex"    | 5×5 hexagonal board, lines of 4 win | 222,464                |
| LineSize5Square | "Line Size/5"<br>"Board Size/9x9"<br>"Board/Square" | 9×9 square board, lines of 5 win    | 222,464                |
| LineSize6Square | "Line Size/6"<br>"Board Size/9x9"<br>"Board/Square" | 9×9 square board, lines of 6 win    | 222,464                |

Table 4. Details on *Diagonal Hex* variants. This implementation of Diagonal Hex is loaded in Polygames using “LudiiDiagonal Hex.lud” as game name.

| <b>Variant</b> | <b>Options</b>     | <b>Description</b>  | <b>Num. Params DNN</b> |
|----------------|--------------------|---------------------|------------------------|
| 7×7            | "Board Size/7x7"   | 7×7 hexagonal board | 222,464                |
| 9×9            | "Board Size/9x9"   | 9×9 hexagonal board | 222,464                |
| 11×11          | "Board Size/11x11" | 11×11 square board  | 222,464                |
| 13×13          | "Board Size/13x13" | 13×13 square board  | 222,464                |
| 19×19          | "Board Size/19x19" | 19×19 square board  | 222,464                |

Table 5. Details on *Gomoku* variants. This implementation of Gomoku is loaded in Polygames using “LudiiGomoku.lud” as game name. By default, Gomoku is played on a 15×15 board.

| <b>Variant</b> | <b>Options</b>     | <b>Description</b> | <b>Num. Params DNN</b> |
|----------------|--------------------|--------------------|------------------------|
| 9×9            | "Board Size/9x9"   | 9×9 square board   | 180,472                |
| 13×13          | "Board Size/13x13" | 13×13 square board | 180,472                |
| 15×15          | "Board Size/15x15" | 15×15 square board | 180,472                |
| 19×19          | "Board Size/19x19" | 19×19 square board | 180,472                |

---

**Transfer of Fully Convolutional Policy-Value Networks Between Games and Game Variants**

---

Table 6. Details on *Hex* variants. This implementation of Hex is loaded in Polygames using “LudiiHex.lud” as game name. By default, Hex is played on an 11×11 board.

| Variant      | Options                                  | Description                         | Num. Params DNN |
|--------------|--|-------------------------------------|-----------------|
| 7×7          | "Board Size/7x7"                         | 7×7 board, standard win condition   | 222,464         |
| 9×9          | "Board Size/9x9"                         | 9×9 board, standard win condition   | 222,464         |
| 11×11        | "Board Size/11x11"                       | 11×11 board, standard win condition | 222,464         |
| 13×13        | "Board Size/13x13"                       | 13×13 board, standard win condition | 222,464         |
| 19×19        | "Board Size/19x19"                       | 19×19 board, standard win condition | 222,464         |
| 11×11 Misere | "Board Size/11x11"<br>"End Rules/Misere" | 11×11 board, inverted win condition | 222,464         |

Table 7. Details on *HeXentafl* variants. This implementation of HeXentafl is loaded in Polygames using “LudiiHeXentafl.lud” as game name. By default, HeXentafl is played on a 4×4 board.

| Variant | Options          | Description         | Num. Params DNN |
|---------|------------------|---------------------|-----------------|
| 4×4     | "Board Size/4x4" | 4×4 hexagonal board | 231,152         |
| 5×5     | "Board Size/5x5" | 5×5 hexagonal board | 231,152         |

Table 8. Details on *Konane* variants. This implementation of Konane is loaded in Polygames using “LudiiKonane.lud” as game name. By default, Konane is played on an 8×8 board.

| Variant | Options            | Description        | Num. Params DNN |
|---------|--------------------|--------------------|-----------------|
| 6×6     | "Board Size/6x6"   | 6×6 square board   | 188,296         |
| 8×8     | "Board Size/8x8"   | 8×8 square board   | 188,296         |
| 10×10   | "Board Size/10x10" | 10×10 square board | 188,296         |
| 12×12   | "Board Size/12x12" | 12×12 square board | 188,296         |

Table 9. Details on *Pentalath* variants. This implementation of Pentalath is loaded in Polygames using “LudiiPentalath.lud” as game name. By default, Pentalath is played on half a hexagonal board.

| Variant         | Options                 | Description            | Num. Params DNN |
|-----------------|-------------------------|------------------------|-----------------|
| HexHexBoard     | "Board/HexHexBoard"     | A full hexagonal board | 180,472         |
| HalfHexHexBoard | "Board/HalfHexHexBoard" | Half a hexagonal board | 180,472         |

Table 10. Details on *Yavalath* variants. This implementation of Yavalath is loaded in Polygames using “LudiiYavalath.lud” as game name. By default, Yavalath is played on a 5×5 board.

| Variant | Options          | Description         | Num. Params DNN |
|---------|------------------|---------------------|-----------------|
| 3×3     | "Board Size/3x3" | 3×3 hexagonal board | 222,464         |
| 4×4     | "Board Size/4x4" | 4×4 hexagonal board | 222,464         |
| 5×5     | "Board Size/5x5" | 5×5 hexagonal board | 222,464         |
| 6×6     | "Board Size/6x6" | 6×6 hexagonal board | 222,464         |
| 7×7     | "Board Size/7x7" | 7×7 hexagonal board | 222,464         |
| 8×8     | "Board Size/8x8" | 8×8 hexagonal board | 222,464         |

### C.3. Ludii Line Completion Games

For the evaluation of transfer between different line completion games, we used six different line completion games: *Connect6*, *Dai Hasami Shogi*, *Gomoku*, *Pentalath*, *Squava*, and *Yavalath*. Several properties of these games are listed in Table 11.

Table 11. Details on different line completion games.

|                     | Connect6    | Dai Hasami Shogi | Gomoku    | Pentalath  | Squava     | Yavalath    |
|---------------------|-------------|------------------|-----------|------------|------------|-------------|
| Board Shape         | Square      | Square           | Square    | Hexagonal  | Square     | Hexagonal   |
| Board Size          | 19×19       | 9×9              | 9×9       | 5×5        | 5×5        | 5×5         |
| Win Line Length     | 6           | 5                | 5         | 5          | 4          | 4           |
| Loss Line Length    | -           | -                | -         | -          | 3          | 3           |
| Max Win Line Length | -           | -                | 5         | -          | -          | -           |
| Can Move Pieces?    | ×           | ✓                | ×         | ×          | ×          | ×           |
| Can Capture Pieces? | ×           | ✓                | ×         | ✓          | ×          | ×           |
| Uses Swap Rule?     | ×           | ×                | ×         | ×          | ✓          | ✓           |
| Moves per Turn      | 2*          | 1                | 1         | 1          | 1          | 1           |
| State Tensor Shape  | (9, 19, 19) | (9, 9, 9)        | (9, 9, 9) | (9, 9, 17) | (10, 5, 5) | (10, 9, 17) |
| Policy Tensor Shape | (3, 19, 19) | (51, 9, 9)       | (3, 9, 9) | (3, 9, 17) | (3, 5, 5)  | (3, 9, 17)  |
| Num. Params DNN     | 180,472     | 188,296          | 180,472   | 180,472    | 222,464    | 222,464     |

\*The first turn in Connect6 consists of only 1 move.

### C.4. Ludii Shogi Games

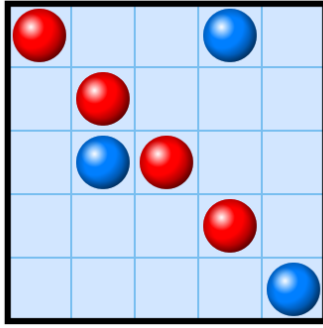
For the evaluation of transfer between different variants of Shogi, we used four games: *Hasami Shogi*, *Kyoto Shogi*, *Minishogi*, and *Shogi*. Several properties of these games are listed in Table 12.

Table 12. Details on variants of Shogi.

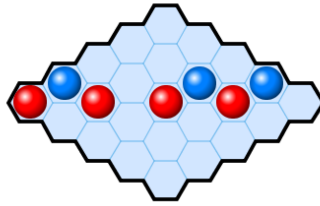
|                             | Hasami Shogi | Kyoto Shogi | Minishogi  | Shogi       |
|-----------------------------|--------------|-------------|------------|-------------|
| Board Size                  | 9×9          | 5×5         | 5×5        | 9×9         |
| Num. Piece Types per Player | 1            | 9           | 10         | 14          |
| Can Drop Captured Pieces?   | ×            | ✓           | ✓          | ✓           |
| State Tensor Shape          | (9, 9, 9)    | (28, 8, 5)  | (30, 8, 5) | (38, 12, 9) |
| Policy Tensor Shape         | (51, 9, 9)   | (51, 8, 5)  | (51, 8, 5) | (51, 12, 9) |
| Num. Params DNN             | 188,296      | 1,752,908   | 2,009,752  | 3,212,648   |

### C.5. Broken Line and Diagonal Hex

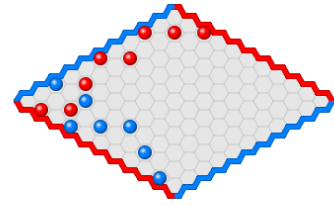
*Broken Line* and *Diagonal Hex* are variations on line completion games, and *Hex*, respectively, which only take into consideration diagonal connections for the line completion and connection win conditions. On hexagonal grids, two cells are considered to be “diagonally connected” if there exists an edge that connects exactly one vertex of each of the cells. Figure 7 depicts examples of winning game states for the red player in Broken Line on a square board, Broken Line on a hexagonal board, and Diagonal Hex.



(a) A diagonal line of 4 on the square board is a win for the red player in *Broken Line*.



(b) A “diagonal” line of 4 on the hexagonal board is a win for the red player in *Broken Line*.



(c) A chain of “diagonally” connected pieces on the hexagonal board is a win for the red player in *Diagonal Hex*.

Figure 7. Examples of winning game states for the red player in *Broken Line* (on a square and hexagonal board), and *Diagonal Hex*. In both examples for *Broken Line*, the target line length was set to 4.

## D. Detailed Results – Zero-shot Transfer Between Game Variants

Tables 13-21 provide detailed results for all evaluations of zero-shot transfer between variants within each out of nine different games.

Table 13. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain (zero-shot transfer). Source and target domains are different boards in Breakthrough.

| Game: Breakthrough | Target Domain |         |         |          |          |          |
|--------------------|---------------|---------|---------|----------|----------|----------|
|                    | Source Domain | Square6 | Square8 | Square10 | Hexagon4 | Hexagon6 |
| Square6            | -             | 0.00%   | 7.33%   | 0.00%    | 0.00%    | 0.67%    |
| Square8            | 10.00%        | -       | 77.00%  | 2.67%    | 0.00%    | 1.00%    |
| Square10           | 1.33%         | 0.33%   | -       | 0.67%    | 0.00%    | 0.33%    |
| Hexagon4           | 0.00%         | 0.00%   | 0.00%   | -        | 0.33%    | 1.33%    |
| Hexagon6           | 0.67%         | 0.00%   | 0.00%   | 12.67%   | -        | 39.67%   |
| Hexagon8           | 0.00%         | 0.00%   | 0.00%   | 4.00%    | 5.00%    | -        |

Table 14. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain (zero-shot transfer). Source and target domains are different boards in Broken Line.

| Game: Broken Line | Target Domain |              |              |                 |
|-------------------|---------------|--------------|--------------|-----------------|
|                   | Source Domain | LineSize3Hex | LineSize4Hex | LineSize5Square |
| LineSize3Hex      | -             | 5.67%        | 0.00%        | 0.00%           |
| LineSize4Hex      | 19.33%        | -            | 0.00%        | 0.17%           |
| LineSize5Square   | 7.00%         | 0.00%        | -            | 49.67%          |
| LineSize6Square   | 3.67%         | 0.00%        | 47.17%       | -               |



**Transfer of Fully Convolutional Policy-Value Networks Between Games and Game Variants**

---

Table 15. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain (zero-shot transfer). Source and target domains are different boards in Diagonal Hex.

| <b>Game: Diagonal Hex</b> | Target Domain |        |        |         |         |       |
|---------------------------|---------------|--------|--------|---------|---------|-------|
|                           | Source Domain | 7×7    | 9×9    | 11×11   | 13×13   | 19×19 |
| 7×7                       | -             | 38.00% | 22.50% | 100.00% | 99.67%  |       |
| 9×9                       | 45.17%        | -      | 83.00% | 100.00% | 100.00% |       |
| 11×11                     | 13.00%        | 18.00% | -      | 100.00% | 100.00% |       |
| 13×13                     | 0.00%         | 0.00%  | 0.00%  | -       | 44.83%  |       |
| 19×19                     | 0.00%         | 0.00%  | 0.00%  | 33.83%  | -       |       |

Table 16. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain (zero-shot transfer). Source and target domains are different boards in Gomoku.

| <b>Game: Gomoku</b> | Target Domain |        |        |        |       |
|---------------------|---------------|--------|--------|--------|-------|
|                     | Source Domain | 9×9    | 13×13  | 15×15  | 19×19 |
| 9×9                 | -             | 44.00% | 31.67% | 18.67% |       |
| 13×13               | 28.17%        | -      | 51.33% | 62.00% |       |
| 15×15               | 25.50%        | 40.50% | -      | 66.00% |       |
| 19×19               | 19.83%        | 32.67% | 35.67% | -      |       |

Table 17. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain (zero-shot transfer). Source and target domains are different variants of Hex.

| <b>Game: Hex</b> | Target Domain |        |        |         |         |       |              |
|------------------|---------------|--------|--------|---------|---------|-------|--------------|
|                  | Source Domain | 7×7    | 9×9    | 11×11   | 13×13   | 19×19 | 11×11 Misere |
| 7×7              | -             | 38.33% | 14.67% | 76.67%  | 91.67%  | 0.00% |              |
| 9×9              | 21.67%        | -      | 56.33% | 100.00% | 100.00% | 0.00% |              |
| 11×11            | 20.33%        | 30.33% | -      | 100.00% | 100.00% | 0.00% |              |
| 13×13            | 4.67%         | 0.67%  | 0.00%  | -       | 100.00% | 0.00% |              |
| 19×19            | 0.00%         | 0.00%  | 0.00%  | 0.00%   | -       | 0.00% |              |
| 11×11 Misere     | 0.00%         | 0.00%  | 0.00%  | 0.00%   | 0.00%   | -     |              |

Table 18. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain (zero-shot transfer). Source and target domains are different boards in HeXentafl.

| <b>Game: HeXentafl</b> | Target Domain |        |     |
|------------------------|---------------|--------|-----|
|                        | Source Domain | 4×4    | 5×5 |
| 4×4                    | -             | 15.50% |     |
| 5×5                    | 9.67%         | -      |     |

Table 19. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain (zero-shot transfer). Source and target domains are different boards in Konane.

| Game: Konane | Target Domain |       |        |         |       |
|--------------|---------------|-------|--------|---------|-------|
|              | Source Domain | 6×6   | 8×8    | 10×10   | 12×12 |
| 6×6          | -             | 3.00% | 14.67% | 63.33%  |       |
| 8×8          | 31.00%        | -     | 94.00% | 100.00% |       |
| 10×10        | 12.00%        | 3.33% | -      | 99.67%  |       |
| 12×12        | 8.00%         | 0.00% | 2.00%  | -       |       |

Table 20. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain (zero-shot transfer). Source and target domains are different boards in Pentalth.

| Game: Pentalth  | Target Domain |             |                 |
|-----------------|---------------|-------------|-----------------|
|                 | Source Domain | HexHexBoard | HalfHexHexBoard |
| HexHexBoard     | -             | 26.67%      |                 |
| HalfHexHexBoard | 18.00%        | -           |                 |

Table 21. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain (zero-shot transfer). Source and target domains are different boards in Yavalath.

| Game: Yavalath | Target Domain |        |        |        |        |        |     |
|----------------|---------------|--------|--------|--------|--------|--------|-----|
|                | Source Domain | 3×3    | 4×4    | 5×5    | 6×6    | 7×7    | 8×8 |
| 3×3            | -             | 10.83% | 4.33%  | 1.67%  | 0.67%  | 0.33%  |     |
| 4×4            | 29.83%        | -      | 29.83% | 15.33% | 7.67%  | 4.33%  |     |
| 5×5            | 10.33%        | 12.17% | -      | 30.33% | 34.00% | 25.00% |     |
| 6×6            | 8.17%         | 20.17% | 41.17% | -      | 45.33% | 58.00% |     |
| 7×7            | 8.50%         | 21.00% | 33.00% | 38.00% | -      | 53.33% |     |
| 8×8            | 7.67%         | 13.00% | 31.00% | 29.83% | 47.67% | -      |     |

### E. Detailed Results – Transfer Between Game Variants With Fine-tuning

Tables 22-30 provide detailed results for all evaluations of transfer performance after fine-tuning, for transfer between variants within each out of nine different games. Models are trained for 20 hours on the source domain, followed by 20 hours on the target domain, and evaluated against models trained for 20 hours only on the target domain. Tables 31-39 provide additional results for a similar evaluation where we reinitialised all the parameters of the final convolutional layers before policy and value heads prior to fine-tuning. The basic idea behind this experiment was that it would lead to a more random, less biased policy generating experience from self-play at the start of a fine-tuning process, and hence may improve fine-tuning transfer in cases where full transfer produces a poor initial policy. Overall we did not observe many major changes in transfer performance.

**Transfer of Fully Convolutional Policy-Value Networks Between Games and Game Variants**

---

Table 22. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain after fine-tuning. Source and target domains are different boards in Breakthrough.

| Source Domain | Target Domain |         |          |          |          |          |
|---------------|---------------|---------|----------|----------|----------|----------|
|               | Square6       | Square8 | Square10 | Hexagon4 | Hexagon6 | Hexagon8 |
| Square6       | -             | 87.33%  | 99.00%   | 50.67%   | 74.00%   | 51.33%   |
| Square8       | 50.33%        | -       | 92.67%   | 50.00%   | 41.00%   | 55.33%   |
| Square10      | 52.00%        | 64.33%  | -        | 49.67%   | 41.67%   | 58.67%   |
| Hexagon4      | 56.00%        | 95.33%  | 80.00%   | -        | 74.67%   | 56.67%   |
| Hexagon6      | 51.67%        | 75.00%  | 70.33%   | 50.00%   | -        | 74.33%   |
| Hexagon8      | 52.67%        | 66.33%  | 89.00%   | 49.00%   | 74.67%   | -        |

Table 23. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain after fine-tuning. Source and target domains are different boards in Broken Line.

| Source Domain   | Target Domain |              |                 |                 |
|-----------------|---------------|--------------|-----------------|-----------------|
|                 | LineSize3Hex  | LineSize4Hex | LineSize5Square | LineSize6Square |
| LineSize3Hex    | -             | 46.67%       | 50.00%          | 50.00%          |
| LineSize4Hex    | 50.00%        | -            | 49.83%          | 50.00%          |
| LineSize5Square | 49.33%        | 49.50%       | -               | 50.00%          |
| LineSize6Square | 48.67%        | 50.83%       | 49.83%          | -               |

Table 24. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain after fine-tuning. Source and target domains are different boards in Diagonal Hex.

| Source Domain | Target Domain |        |        |         |         |
|---------------|---------------|--------|--------|---------|---------|
|               | 7×7           | 9×9    | 11×11  | 13×13   | 19×19   |
| 7×7           | -             | 48.83% | 84.50% | 100.00% | 100.00% |
| 9×9           | 54.33%        | -      | 86.00% | 100.00% | 100.00% |
| 11×11         | 54.00%        | 45.50% | -      | 100.00% | 100.00% |
| 13×13         | 55.00%        | 49.67% | 12.83% | -       | 41.00%  |
| 19×19         | 54.17%        | 48.50% | 0.00%  | 42.50%  | -       |

Table 25. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain after fine-tuning. Source and target domains are different boards in Gomoku.

| Source Domain | Target Domain |        |        |        |
|---------------|---------------|--------|--------|--------|
|               | 9×9           | 13×13  | 15×15  | 19×19  |
| 9×9           | -             | 69.50% | 70.33% | 78.00% |
| 13×13         | 54.33%        | -      | 64.67% | 72.67% |
| 15×15         | 62.17%        | 50.50% | -      | 67.33% |
| 19×19         | 55.50%        | 66.67% | 64.50% | -      |

---

**Transfer of Fully Convolutional Policy-Value Networks Between Games and Game Variants**

---

Table 26. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain after fine-tuning. Source and target domains are different variants of Hex.

| <b>Game: Hex</b> | Target Domain |        |        |         |         |         |              |
|------------------|---------------|--------|--------|---------|---------|---------|--------------|
|                  | Source Domain | 7×7    | 9×9    | 11×11   | 13×13   | 19×19   | 11×11 Misere |
| 7×7              | -             | 69.33% | 81.33% | 100.00% | 100.00% | 100.00% | 56.67%       |
| 9×9              | 47.67%        | -      | 73.00% | 100.00% | 100.00% | 100.00% | 93.67%       |
| 11×11            | 48.33%        | 71.00% | -      | 100.00% | 100.00% | 100.00% | 98.33%       |
| 13×13            | 47.67%        | 27.67% | 40.00% | -       | 100.00% | 100.00% | 57.33%       |
| 19×19            | 50.00%        | 48.33% | 44.67% | 82.33%  | -       | -       | 3.33%        |
| 11×11 Misere     | 47.67%        | 39.00% | 45.33% | 100.00% | 21.67%  | -       | -            |

Table 27. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain after fine-tuning. Source and target domains are different boards in HeXentafl.

| <b>Game: HeXentafl</b> | Target Domain |        |     |
|------------------------|---------------|--------|-----|
|                        | Source Domain | 4×4    | 5×5 |
| 4×4                    | -             | 50.50% | -   |
| 5×5                    | 55.83%        | -      | -   |

Table 28. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain after fine-tuning. Source and target domains are different boards in Konane.

| <b>Game: Konane</b> | Target Domain |        |        |         |       |
|---------------------|---------------|--------|--------|---------|-------|
|                     | Source Domain | 6×6    | 8×8    | 10×10   | 12×12 |
| 6×6                 | -             | 52.33% | 65.00% | 99.67%  | -     |
| 8×8                 | 51.00%        | -      | 94.67% | 98.33%  | -     |
| 10×10               | 49.33%        | 62.00% | -      | 100.00% | -     |
| 12×12               | 52.00%        | 20.67% | 56.67% | -       | -     |

Table 29. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain after fine-tuning. Source and target domains are different boards in Pentalath.

| <b>Game: Pentalath</b> | Target Domain |             |                 |
|------------------------|---------------|-------------|-----------------|
|                        | Source Domain | HexHexBoard | HalfHexHexBoard |
| HexHexBoard            | -             | 72.67%      | -               |
| HalfHexHexBoard        | 52.00%        | -           | -               |



**Transfer of Fully Convolutional Policy-Value Networks Between Games and Game Variants**

Table 30. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain after fine-tuning. Source and target domains are different boards in Yavalath.

| Game: Yavalath<br>Source Domain | Target Domain |        |        |        |        |        |
|---------------------------------|---------------|--------|--------|--------|--------|--------|
|                                 | 3×3           | 4×4    | 5×5    | 6×6    | 7×7    | 8×8    |
| 3×3                             | -             | 39.50% | 67.00% | 57.33% | 57.67% | 70.67% |
| 4×4                             | 46.33%        | -      | 73.00% | 55.17% | 60.67% | 58.67% |
| 5×5                             | 49.00%        | 49.50% | -      | 52.00% | 63.00% | 58.33% |
| 6×6                             | 46.17%        | 54.83% | 69.83% | -      | 65.00% | 60.33% |
| 7×7                             | 52.00%        | 41.17% | 68.83% | 67.67% | -      | 74.00% |
| 8×8                             | 45.00%        | 67.50% | 66.00% | 53.17% | 44.67% | -      |

Table 31. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain after fine-tuning with reinitialised final layers. Source and target domains are different boards in Breakthrough.

| Game: Breakthrough<br>Source Domain | Target Domain |         |          |          |          |          |
|-------------------------------------|---------------|---------|----------|----------|----------|----------|
|                                     | Square6       | Square8 | Square10 | Hexagon4 | Hexagon6 | Hexagon8 |
| Square6                             | -             | 92.67%  | 96.33%   | 48.33%   | 66.67%   | 65.33%   |
| Square8                             | 57.00%        | -       | 88.33%   | 49.67%   | 60.33%   | 65.33%   |
| Square10                            | 52.33%        | 53.33%  | -        | 49.33%   | 42.33%   | 38.00%   |
| Hexagon4                            | 47.67%        | 77.67%  | 95.00%   | -        | 84.33%   | 73.00%   |
| Hexagon6                            | 53.00%        | 86.67%  | 68.67%   | 49.67%   | -        | 74.00%   |
| Hexagon8                            | 52.33%        | 66.00%  | 93.33%   | 52.00%   | 74.00%   | -        |

Table 32. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain after fine-tuning with reinitialised final layers. Source and target domains are different boards in Broken Line.

| Game: Broken Line<br>Source Domain | Target Domain |              |                 |                 |
|------------------------------------|---------------|--------------|-----------------|-----------------|
|                                    | LineSize3Hex  | LineSize4Hex | LineSize5Square | LineSize6Square |
| LineSize3Hex                       | -             | 49.00%       | 50.00%          | 50.00%          |
| LineSize4Hex                       | 49.00%        | -            | 50.00%          | 49.83%          |
| LineSize5Square                    | 50.00%        | 50.50%       | -               | 50.00%          |
| LineSize6Square                    | 49.67%        | 49.67%       | 49.67%          | -               |

Table 33. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain after fine-tuning with reinitialised final layers. Source and target domains are different boards in Diagonal Hex.

| Game: Diagonal Hex<br>Source Domain | Target Domain |        |        |         |         |
|-------------------------------------|---------------|--------|--------|---------|---------|
|                                     | 7×7           | 9×9    | 11×11  | 13×13   | 19×19   |
| 7×7                                 | -             | 51.00% | 86.67% | 100.00% | 100.00% |
| 9×9                                 | 50.33%        | -      | 89.67% | 100.00% | 100.00% |
| 11×11                               | 51.33%        | 46.83% | -      | 100.00% | 100.00% |
| 13×13                               | 54.83%        | 49.67% | 15.00% | -       | 53.00%  |
| 19×19                               | 52.83%        | 49.33% | 6.50%  | 45.17%  | -       |

**Transfer of Fully Convolutional Policy-Value Networks Between Games and Game Variants**

---

Table 34. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain after fine-tuning with reinitialised final layers. Source and target domains are different boards in Gomoku.

| <b>Game: Gomoku</b> | Target Domain |        |        |        |       |
|---------------------|---------------|--------|--------|--------|-------|
|                     | Source Domain | 9×9    | 13×13  | 15×15  | 19×19 |
| 9×9                 | -             | 68.00% | 65.33% | 70.00% |       |
| 13×13               | 61.83%        | -      | 65.33% | 74.00% |       |
| 15×15               | 59.33%        | 58.33% | -      | 71.67% |       |
| 19×19               | 55.33%        | 55.17% | 57.17% | -      |       |

Table 35. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain after fine-tuning with reinitialised final layers. Source and target domains are different variants of Hex.

| <b>Game: Hex</b> | Target Domain |        |        |         |         |       |              |
|------------------|---------------|--------|--------|---------|---------|-------|--------------|
|                  | Source Domain | 7×7    | 9×9    | 11×11   | 13×13   | 19×19 | 11×11 Misere |
| 7×7              | -             | 68.00% | 74.00% | 100.00% | 100.00% |       | 87.33%       |
| 9×9              | 49.00%        | -      | 72.67% | 99.67%  | 100.00% |       | 96.33%       |
| 11×11            | 49.00%        | 50.33% | -      | 100.00% | 100.00% |       | 93.00%       |
| 13×13            | 51.33%        | 41.33% | 40.00% | -       | 100.00% |       | 73.33%       |
| 19×19            | 49.67%        | 43.00% | 36.00% | 99.00%  | -       |       | 0.67%        |
| 11×11 Misere     | 47.00%        | 37.67% | 41.00% | 100.00% | 84.00%  |       | -            |

Table 36. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain after fine-tuning with reinitialised final layers. Source and target domains are different boards in HeXentafl.

| <b>Game: HeXentafl</b> | Target Domain |        |     |
|------------------------|---------------|--------|-----|
|                        | Source Domain | 4×4    | 5×5 |
| 4×4                    | -             | 52.17% |     |
| 5×5                    | 43.17%        | -      |     |

Table 37. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain after fine-tuning with reinitialised final layers. Source and target domains are different boards in Konane.

| <b>Game: Konane</b> | Target Domain |        |        |        |       |
|---------------------|---------------|--------|--------|--------|-------|
|                     | Source Domain | 6×6    | 8×8    | 10×10  | 12×12 |
| 6×6                 | -             | 54.00% | 76.33% | 98.33% |       |
| 8×8                 | 51.67%        | -      | 95.67% | 99.33% |       |
| 10×10               | 50.67%        | 36.00% | -      | 99.00% |       |
| 12×12               | 51.33%        | 14.67% | 38.00% | -      |       |

Table 38. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain after fine-tuning with reinitialised final layers. Source and target domains are different boards in Pentalath.

| Game: Pentalath | Target Domain |                 |
|-----------------|---------------|-----------------|
|                 | HexHexBoard   | HalfHexHexBoard |
| Source Domain   |               |                 |
| HexHexBoard     | -             | 65.67%          |
| HalfHexHexBoard | 51.67%        | -               |

Table 39. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain after fine-tuning with reinitialised final layers. Source and target domains are different boards in Yavalath.

| Game: Yavalath | Target Domain |        |        |        |        |        |
|----------------|---------------|--------|--------|--------|--------|--------|
|                | 3×3           | 4×4    | 5×5    | 6×6    | 7×7    | 8×8    |
| Source Domain  |               |        |        |        |        |        |
| 3×3            | -             | 50.50% | 56.17% | 65.67% | 72.00% | 70.67% |
| 4×4            | 48.33%        | -      | 69.00% | 68.50% | 63.00% | 47.33% |
| 5×5            | 50.33%        | 51.67% | -      | 44.00% | 57.67% | 44.33% |
| 6×6            | 53.17%        | 60.50% | 68.17% | -      | 68.33% | 53.67% |
| 7×7            | 49.33%        | 54.83% | 68.83% | 56.67% | -      | 59.00% |
| 8×8            | 51.17%        | 43.33% | 57.67% | 45.83% | 68.00% | -      |

## F. Detailed Results – Zero-shot Transfer Between Games

Tables 40-43 provide detailed results for zero-shot transfer evaluations, where source domains are different games from target domains (not just different variants).

Table 40. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain (zero-shot transfer). Source and target domains are different line-completion games.

| Source Domain    | Target Domain |                  |        |           |        |          |
|------------------|---------------|------------------|--------|-----------|--------|----------|
|                  | Connect6      | Dai Hasami Shogi | Gomoku | Pentalath | Squava | Yavalath |
| Connect6         | -             | 0.00%            | 2.33%  | 0.00%     | 1.00%  | 0.33%    |
| Dai Hasami Shogi | 0.67%         | -                | 1.33%  | 0.00%     | 0.67%  | 1.67%    |
| Gomoku           | 36.67%        | 0.00%            | -      | 0.33%     | 2.67%  | 1.33%    |
| Pentalath        | 11.67%        | 0.00%            | 4.33%  | -         | 2.00%  | 1.33%    |
| Squava           | 16.00%        | 0.00%            | 0.33%  | 0.00%     | -      | 2.00%    |
| Yavalath         | 0.00%         | 0.00%            | 0.00%  | 0.33%     | 1.67%  | -        |

Table 41. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain (zero-shot transfer). Source and target domains are different Shogi variants.

| Source Domain | Target Domain |             |           |        |
|---------------|---------------|-------------|-----------|--------|
|               | Hasami Shogi  | Kyoto Shogi | Minishogi | Shogi  |
| Hasami Shogi  | -             | 1.33%       | 0.33%     | 52.67% |
| Kyoto Shogi   | 39.83%        | -           | 3.00%     | 44.67% |
| Minishogi     | 47.17%        | 16.17%      | -         | 97.00% |
| Shogi         | 23.83%        | 1.67%       | 0.00%     | -      |

Table 42. Win percentage of MCTS with final checkpoint from *Broken Line* variants against MCTS with final checkpoint trained in target domain, evaluated in target domain (zero-shot transfer). Target domains are different line completion games.

| Source (Broken Line) | Target Domain |                  |        |           |        |          |
|----------------------|---------------|------------------|--------|-----------|--------|----------|
|                      | Connect6      | Dai Hasami Shogi | Gomoku | Pentalath | Squava | Yavalath |
| LineSize3Hex         | 0.00%         | 0.00%            | 0.00%  | 0.00%     | 0.67%  | 1.67%    |
| LineSize4Hex         | 0.00%         | 0.00%            | 0.00%  | 0.00%     | 0.33%  | 1.67%    |
| LineSize5Square      | 31.33%        | 0.00%            | 1.00%  | 0.33%     | 0.67%  | 1.33%    |
| LineSize6Square      | 32.00%        | 0.00%            | 1.00%  | 1.67%     | 0.33%  | 2.00%    |

Table 43. Win percentage of MCTS with final checkpoint from *Diagonal Hex* variants against MCTS with final checkpoint trained in target domain, evaluated in target domain (zero-shot transfer). Target domains are different variants of *Hex*.

| Source (Diagonal Hex) | Target (Hex) |       |       |              |       |        |
|-----------------------|--------------|-------|-------|--------------|-------|--------|
|                       | 7×7          | 9×9   | 11×11 | 11×11 Misere | 13×13 | 19×19  |
| 7×7                   | 0.00%        | 0.00% | 0.00% | 0.00%        | 0.00% | 10.33% |
| 9×9                   | 0.00%        | 0.00% | 0.00% | 0.00%        | 0.00% | 15.67% |
| 11×11                 | 0.00%        | 0.00% | 0.00% | 0.00%        | 0.00% | 28.33% |
| 13×13                 | 0.00%        | 0.00% | 0.00% | 0.00%        | 0.00% | 9.00%  |
| 19×19                 | 0.00%        | 0.00% | 0.00% | 0.00%        | 0.00% | 24.33% |

### G. Detailed Results – Transfer Between Games With Fine-tuning

Tables 44-47 provide detailed results for evaluations of transfer performance after fine-tuning, where source domains are different games from target domains (not just different variants). Note that in these cases, the two models that play against each other do not always have exactly the same number of trainable parameters. For hidden convolutional layers, we always use twice as many channels as the number of channels in a game’s state tensor representation, and this is not modified when transferring to a new domain. This means that if a source domain has a greater number of channels in its state tensor representation than the target domain, the transferred model will also still use more channels in its hidden convolutional layers than the baseline model, and vice versa when the source domain has fewer state channels. Tables 48-51 provide additional results where we adjust the number of channels of hidden convolutional layers when transferring models, prior to fine-tuning, for a more “fair” evaluation in terms of network size.

**Transfer of Fully Convolutional Policy-Value Networks Between Games and Game Variants**

Table 44. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain (after fine-tuning). Source and target domains are different line-completion games.

| Source Domain    | Target Domain |                  |        |           |        |          |
|------------------|---------------|------------------|--------|-----------|--------|----------|
|                  | Connect6      | Dai Hasami Shogi | Gomoku | Pentalath | Squava | Yavalath |
| Connect6         | -             | 54.00%           | 53.17% | 58.33%    | 45.50% | 63.33%   |
| Dai Hasami Shogi | 54.67%        | -                | 54.50% | 53.67%    | 48.00% | 72.33%   |
| Gomoku           | 95.00%        | 50.17%           | -      | 59.33%    | 48.50% | 49.67%   |
| Pentalath        | 92.00%        | 53.33%           | 57.67% | -         | 46.67% | 50.50%   |
| Squava           | 94.33%        | 51.00%           | 56.67% | 64.00%    | -      | 75.17%   |
| Yavalath         | 43.00%        | 53.33%           | 56.00% | 56.00%    | 45.00% | -        |

Table 45. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain (after fine-tuning). Source and target domains are different Shogi variants.

| Source Domain | Target Domain |             |           |        |
|---------------|---------------|-------------|-----------|--------|
|               | Hasami Shogi  | Kyoto Shogi | Minishogi | Shogi  |
| Hasami Shogi  | -             | 38.17%      | 40.17%    | 89.00% |
| Kyoto Shogi   | 45.67%        | -           | 35.67%    | 70.00% |
| Minishogi     | 52.00%        | 63.17%      | -         | 86.67% |
| Shogi         | 49.67         | 75.83%      | 36.00%    | -      |

Table 46. Win percentage of MCTS with final checkpoint from *Broken Line* variants against MCTS with final checkpoint trained in target domain, evaluated in target domain (after fine-tuning). Target domains are different line completion games.

| Source (Broken Line) | Target Domain |                  |        |           |        |          |
|----------------------|---------------|------------------|--------|-----------|--------|----------|
|                      | Connect6      | Dai Hasami Shogi | Gomoku | Pentalath | Squava | Yavalath |
| LineSize3Hex         | 56.00%        | 52.00%           | 54.83% | 53.67%    | 49.00% | 47.17%   |
| LineSize4Hex         | 64.67%        | 52.83%           | 53.83% | 56.33%    | 48.00% | 68.00%   |
| LineSize5Square      | 88.67%        | 53.67%           | 53.83% | 66.00%    | 47.33% | 64.00%   |
| LineSize6Square      | 90.00%        | 52.33%           | 50.67% | 52.00%    | 46.00% | 58.67%   |

Table 47. Win percentage of MCTS with final checkpoint from *Diagonal Hex* variants against MCTS with final checkpoint trained in target domain, evaluated in target domain (after fine-tuning). Target domains are different variants of Hex.

| Source (Diagonal Hex) | Target (Hex) |        |        |              |         |        |
|-----------------------|--------------|--------|--------|--------------|---------|--------|
|                       | 7×7          | 9×9    | 11×11  | 11×11 Misere | 13×13   | 19×19  |
| 7×7                   | 51.00%       | 59.00% | 15.33% | 48.67%       | 99.33%  | 80.00% |
| 9×9                   | 51.67%       | 50.33% | 19.00% | 53.33%       | 100.00% | 70.67% |
| 11×11                 | 46.00%       | 19.67% | 6.33%  | 23.00%       | 97.33%  | 40.67% |
| 13×13                 | 47.00%       | 56.67% | 5.33%  | 0.67%        | 0.00%   | 20.00% |
| 19×19                 | 45.00%       | 54.67% | 28.67% | 1.67%        | 0.00%   | 45.33% |

**Transfer of Fully Convolutional Policy-Value Networks Between Games and Game Variants**

Table 48. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain (after fine-tuning). Source and target domains are different line-completion games.

| Source Domain    | Target Domain |                  |        |           |        |          |
|------------------|---------------|------------------|--------|-----------|--------|----------|
|                  | Connect6      | Dai Hasami Shogi | Gomoku | Pentalath | Squava | Yavalath |
| Connect6         | -             | 53.83%           | 57.50% | 69.00%    | 51.00% | 74.00%   |
| Dai Hasami Shogi | 55.33%        | -                | 57.67% | 59.00%    | 48.67% | 68.83%   |
| Gomoku           | 93.00%        | 52.00%           | -      | 60.33%    | 46.00% | 62.67%   |
| Pentalath        | 76.67%        | 48.33%           | 59.83% | -         | 47.33% | 58.33%   |
| Squava           | 40.67%        | 50.00%           | 58.17% | 58.67%    | -      | 69.50%   |
| Yavalath         | 70.33%        | 52.00%           | 51.67% | 53.00%    | 49.00% | -        |

Table 49. Win percentage of MCTS with final checkpoint from source domain against MCTS with final checkpoint trained in target domain, evaluated in target domain (after fine-tuning, with number of channels in hidden convolutional layers adjusted to be equal). Source and target domains are different Shogi variants.

| Source Domain | Target Domain |             |           |        |
|---------------|---------------|-------------|-----------|--------|
|               | Hasami Shogi  | Kyoto Shogi | Minishogi | Shogi  |
| Hasami Shogi  | -             | 35.83%      | 34.67%    | 67.33% |
| Kyoto Shogi   | 48.00%        | -           | 33.67%    | 63.33% |
| Minishogi     | 50.00%        | 58.00%      | -         | 65.33% |
| Shogi         | 49.67%        | 45.67%      | 45.67%    | -      |

Table 50. Win percentage of MCTS with final checkpoint from *Broken Line* variants against MCTS with final checkpoint trained in target domain, evaluated in target domain (after fine-tuning, with number of channels in hidden convolutional layers adjusted to be equal). Target domains are different line completion games.

| Source (Broken Line) | Target Domain |                  |        |           |        |          |
|----------------------|---------------|------------------|--------|-----------|--------|----------|
|                      | Connect6      | Dai Hasami Shogi | Gomoku | Pentalath | Squava | Yavalath |
| LineSize3Hex         | 46.67%        | 50.00%           | 48.17% | 56.00%    | 45.67% | 74.17%   |
| LineSize4Hex         | 45.67%        | 54.33%           | 52.17% | 60.00%    | 48.33% | 66.67%   |
| LineSize5Square      | 94.00%        | 49.33%           | 54.33% | 63.00%    | 48.33% | 70.00%   |
| LineSize6Square      | 82.67%        | 49.67%           | 50.17% | 47.33%    | 47.00% | 72.00%   |

Table 51. Win percentage of MCTS with final checkpoint from *Diagonal Hex* variants against MCTS with final checkpoint trained in target domain, evaluated in target domain (after fine-tuning, with number of channels in hidden convolutional layers adjusted to be equal). Target domains are different variants of Hex.

| Source (Diagonal Hex) | Target (Hex) |        |        |              |        |         |
|-----------------------|--------------|--------|--------|--------------|--------|---------|
|                       | 7×7          | 9×9    | 11×11  | 11×11 Misere | 13×13  | 19×19   |
| 7×7                   | 50.67%       | 31.67% | 56.33% | 8.67%        | 99.67% | 100.00% |
| 9×9                   | 47.00%       | 48.33% | 42.00% | 40.00%       | 98.67% | 66.67%  |
| 11×11                 | 47.67%       | 25.67% | 42.33% | 20.67%       | 87.00% | 12.00%  |
| 13×13                 | 49.33%       | 44.33% | 8.33%  | 1.33%        | 0.00%  | 56.67%  |
| 19×19                 | 50.00%       | 50.67% | 28.33% | 2.00%        | 0.00%  | 10.67%  |