2018 Conference on Technologies and Applications of Artificial Intelligence (TAAI)

# Interpreting Neural-Network Players for Game 2048

Kiminori Matsuzaki
*School of Information, Kochi University of Technology*
Kami, Kochi, Japan
Email: matsuzaki.kiminori@kochi-tech.ac.jp

Madoka Teramura
*School of Information, Kochi University of Technology*
Kami, Kochi, Japan
190341z@ugs.kochi-tech.ac.jp

***Abstract*—Game 2048 is a stochastic single-player game and development of strong computer players for 2048 has been based on N-tuple networks trained by reinforcement learning. In our previous study, we developed computer players for game 2048 based on convolutional neural networks (CNNs), and showed by experiments that networks with three or more convolution layers performed much better than that with two convolution layers. In this study, we analyze the inner working of our CNNs (i.e. white box approach) to identify the reasons of the performance. Our analyses include visualization of filters in the first layers and backward trace of the networks for some specific game states. We report several findings about inner working of our CNNs for game 2048.**

***Index Terms*—Game 2048, Convolutional Neural Network, Visualization, Deconvolution**

## I. Introduction

Machine learning algorithms are now widely used in the development of computer game players. Among them, neural networks (NNs), especially (deep) convolutional neural networks (CNNs) have been studied actively in recent years and served an important role in the development of master-level computer players, for example, for Go (AlphaGo [1] and AlphaGo Zero [2]), Chess (Giraffe [3] and DeepChess [4]), Shogi (AlphaZero [5]), Poker (Poker-CNN [6] and DeepStack [7]), and Atari games [8].

The target in this study is game "2048" [9], a stochastic single-player game. Game 2048 is one of slide-and-merge games and its "easy to learn but hard to master" characteristics attracted quite a few people[1].

Several computer players have been developed for game 2048. Among them, the most successful approach is to use N-tuple networks (NTNs) as evaluation functions and apply a reinforcement learning method to adjust the weights of NTNs. This approach was first introduced to game 2048 by Szubert and Jaśkowski [10], and then several studies were based on it. The state-of-the-art computer player developed by Jaśkowski [11] combined several techniques to improve NTN-based players, and achieved average score 609,104 under the time limit of 1 second per move.

In our previous study [12], we developed CNN-based players for game 2048, extending a prior study by Guei et al. [13]. Two important changes in our CNN-based players were: (1) we increased the number of convolution (conv2d) layers from two up to nine[2] while keeping the number of weights almost the same; (2) we trained our networks by supervised learning with play logs of existing strong players. After training with $6 \times 10^8$ game actions, the networks with three or more conv2d layers outperformed the network with two conv2d layers, and the best network with five conv2d layers achieved average score 86,030 without lookahead.

In this paper, we analyze the inner working of our CNNs (i.e. white box approach) to identify the reason of poor performance of the 2-conv2d network and nice performance of deeper networks. We first visualize weights in the first layers to check that the first layers successfully extracted features of states. We then trace the networks in the backward direction to see how outputs are computed for some particular states.

Important findings in this paper are as follows.

- The first layers of CNNs work to extract several features of states. We can also find several corresponding pairs of filters between the first layers.
- Both 2- and 3-conv2d networks have some generalized knowledge. In particular, such an instance of knowledge was encoded in some paths near the end of networks.
- There are some cases that our networks select a bad move while human players can easily does a good move.

The rest of the paper is organized as follows. Section II briefly introduces the rule of game 2048 and important heuristics for the game. Section III reviews the design of our CNN-based players for game 2048. Section IV analyzes the first layers of the networks by visualizing the weights of filters. Section V analyzes the generalization ability of our CNNs by backward trace from specific states. Section VI further analyzes moves of our CNN-based players. We discuss related work in Section VII, and conclude the paper in Section VIII.

## II. Game 2048

The game 2048 is a slide-and-merge game played on a $4 \times 4$ grid (Fig. 2). In an initial state, two tiles are put randomly with numbers 2 ($p_2 = 0.9$) or 4 ($p_4 = 0.1$). On each turn, the player selects a direction (either up, right, down, or left), and then all the tiles will slide in the selected direction. When two tiles of the same number collide in the direction of slide, they create a tile with the sum value and the player gets the

[1]According to its author, during the first three weeks after the release, people spent a total time of over 3000 years on playing the game.

[2]The whole structure of the networks consists of $k$ conv2d layers and a full-connect layer followed by a softmax as shown in Fig. 1. To avoid confusion, we call them $k$-conv2d networks instead of $(k+1)$-ply networks.

2376-6824/18/$31.00 ©2018 IEEE
DOI 10.1109/TAAI.2018.00038

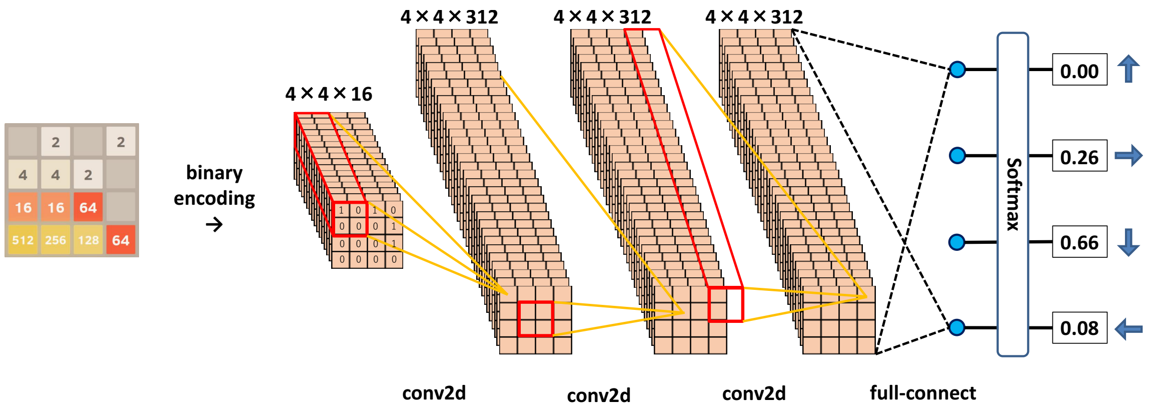
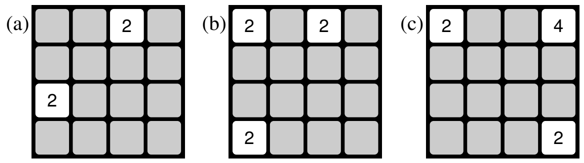


Fig. 1. Overview of our convolutional neural networks (3-conv2d network)



(a) An initial state. Two tiles are put randomly.
(b) After the first move: *up*. A new 2-tile appears at the lower-left corner.
(c) After the second move: *right*. Two 2-tiles are merged to a 4-tile, and score 4 is given. A new tile appears at the upper-left corner.

Fig. 2. Process of game 2048

sum value as the score. Here, the merges occur from the far side and newly created tiles do not merge again on the same turn: slide to the right from `222␣`, `␣422` and `2222` results in `␣␣24`, `␣␣44`, and `␣␣44`, respectively. Note that the player cannot select a direction in which no tiles slide nor merge. After each slide (and merge), a new tile appears randomly at an empty cell with number 2 ($p_2 = 0.9$) or 4 ($p_4 = 0.1$). If the player cannot move the tiles, the game ends. The objective of the original 2048 game is to reach a 2048 tile by sliding and merging tiles according to the rules. In this paper, we continue the game after reaching a 2048 tile, and try to obtain as high score as possible.

Here, we show two important heuristics for the game 2048, to help readers to understand our analyses in the following sections.

*Heuristic 1:* Put the largest-numbered tile at a corner and keep it stationary.

*Heuristic 2:* Put large-numbered tiles at an edge, and they should be aligned in the descending order.

## III. Neural Network Players for Game 2048

In this section, we review our CNN-based players for game 2048 developed in our previous work [12].

TABLE I
Numbers of convolution layers, channels and weights

| Layers $k$ | Channels $Ch(k)$ | Weights |
|---|---|---|
| 2 | 436 | 817,068 |
| 3 | 312 | 819,628 |
| 4 | 256 | 820,228 |
| 5 | 224 | 832,612 |

### A. Structure

We designed networks with $k$ convolution layers and a full-connect layer followed by a softmax. Figure 1 depicts the structure of CNNs for the case of three convolution layers.

The input board is encoded to a $4 \times 4$ 16-channel binary image[3] where the channels correspond to empty cells, 2-tiles, 4-tiles, ..., 32768-tiles, respectively. In the first channel, for instance, we put ones for the places of empty cells and zeros for the others.

Then, we have $k$ convolution layers (conv2d). We chose the number of channels of intermediate images, $C_k$, so that the networks have similar numbers of weights (Table I). We have $C_k$ filters of size $2\times2\times16$ in the first layer, and $C_k$ filters of size $2 \times 2 \times C_k$ in the latter layers. For each convolution, we set the stride width to be one and enable zero padding[4] so that the result images keep the size to be $4 \times 4$. After each convolution, a bias is added for each channel and ReLU (rectified linear units) is applied.

After the convolution layers, all the pixel values are processed in a full-connect layer that outputs four values. The last softmax transforms the four values to probabilities of selecting the four directions.

### B. Supervised Learning

We applied supervised learning to train the networks. Since we developed strong computer players for 2048 in our prior

[3]The encoding of the input board is the same as that by Guei et al. [13].
[4]Convolution is asymmetric due to the padding on the right and bottom.

TABLE III
VISUALIZING FILTERS IN THE FIRST LAYERS OF CNNS.

| 2-conv2d player | | – | $2^1$ | $2^2$ | $2^3$ | $2^4$ | $2^5$ | $2^6$ | $2^7$ | $2^8$ | $2^9$ | $2^{10}$ | $2^{11}$ | $2^{12}$ | $2^{13}$ | $2^{14}$ | $2^{15}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| #174 | NW | -0.53 | -0.68 | -0.58 | 0.65 | -0.27 | -0.77 | -0.78 | -0.79 | -0.83 | -0.44 | -0.47 | -0.30 | -0.44 | -0.12 | -0.03 | 0.08 |
| | NE | 0.00 | -0.44 | 0.04 | 0.84 | -0.13 | -0.60 | -0.56 | -0.60 | -0.61 | -0.49 | -0.19 | -0.25 | -0.06 | -0.07 | -0.23 | 0.10 |
| | SW | -0.27 | -0.39 | -0.03 | 0.96 | -0.31 | -0.42 | -0.30 | -0.29 | -0.32 | -0.20 | -0.05 | 0.01 | 0.16 | 0.12 | -0.04 | 0.19 |
| | SE | 0.28 | 0.12 | 0.42 | 1.09 | 0.24 | -0.52 | -0.27 | -0.33 | -0.24 | -0.19 | -0.04 | -0.09 | -0.05 | 0.27 | 0.21 | 0.55 |
| # 97 | NW | 0.31 | -0.47 | 0.16 | 0.24 | 0.23 | -0.24 | -0.64 | 0.70 | -1.02 | -0.26 | -0.24 | -0.14 | -0.08 | -0.05 | -0.05 | -0.05 |
| | NE | -0.47 | -0.07 | -0.46 | -0.54 | -0.47 | -0.01 | 0.18 | -1.03 | 0.44 | 0.30 | 0.31 | 0.24 | 0.36 | 0.39 | 0.31 | 0.55 |
| | SW | -0.38 | 0.09 | -0.41 | -0.46 | -0.56 | -0.21 | 0.10 | -1.05 | 0.33 | 0.18 | 0.25 | 0.15 | 0.27 | 0.31 | 0.28 | 0.32 |
| | SE | -0.55 | -0.34 | -0.23 | 0.03 | 0.12 | -0.09 | -0.97 | 0.17 | -0.99 | -0.26 | -0.12 | -0.10 | -0.10 | -0.04 | -0.06 | -0.06 |
| # 17 | NW | -1.18 | -0.41 | -0.10 | 0.05 | -0.06 | -0.19 | -0.58 | 0.00 | 0.32 | 0.51 | 0.45 | 0.50 | 0.52 | 0.39 | 0.36 | 0.42 |
| | NE | -1.37 | -0.68 | -0.83 | -0.16 | -0.28 | -0.12 | 0.08 | 0.16 | 0.48 | 0.42 | 0.50 | 0.42 | 0.41 | 0.49 | 0.39 | 0.44 |
| | SW | 0.60 | 0.41 | 0.24 | -0.06 | -0.12 | -0.06 | -0.55 | -0.68 | -0.76 | 0.21 | 0.04 | -0.01 | 0.06 | 0.11 | 0.01 | 0.05 |
| | SE | 0.51 | 0.47 | 0.42 | -0.28 | -0.17 | -0.13 | 0.07 | -0.14 | -0.14 | -1.12 | -0.37 | 0.01 | -0.13 | -0.13 | -0.10 | -0.19 |
| #432 | NW | -0.18 | -0.47 | -0.03 | -0.07 | -0.03 | -0.05 | 0.15 | -0.15 | -0.11 | -0.31 | -0.05 | 0.25 | 0.25 | 0.44 | 0.45 | 0.50 |
| | NE | -0.54 | -0.43 | -0.30 | 0.14 | 0.14 | -0.06 | 0.33 | 0.34 | 0.58 | 0.70 | 0.71 | 0.49 | 0.51 | 0.17 | 0.20 | 0.30 |
| | SW | 0.56 | 0.13 | -0.13 | -0.46 | -0.39 | -0.21 | 0.03 | 0.02 | 0.23 | 0.24 | 0.32 | 0.29 | 0.47 | -1.62 | -0.94 | -0.40 |
| | SE | 0.85 | 0.14 | 0.15 | 0.34 | 0.09 | 0.03 | -0.21 | -1.27 | -1.29 | -1.30 | -0.57 | -0.63 | -0.31 | -0.12 | -1.07 | -0.99 |
| #120 | NW | 0.43 | 0.59 | -0.79 | 0.03 | -0.49 | -0.51 | -0.56 | -0.45 | -0.40 | -0.34 | 0.08 | 0.18 | 0.08 | -0.11 | -0.21 | -1.13 |
| | NE | -0.15 | -0.25 | -0.30 | -0.43 | -0.01 | 0.10 | 0.33 | 0.32 | 0.47 | 0.57 | 0.48 | 0.40 | 0.45 | 0.34 | 0.29 | 0.09 |
| | SW | -0.85 | 0.37 | -0.07 | -0.09 | -0.37 | -0.42 | -1.32 | -1.35 | -1.23 | -1.34 | -1.28 | -0.30 | -0.43 | -0.38 | -0.41 | -0.41 |
| | SE | 0.56 | 0.19 | -0.40 | -0.29 | -0.13 | -0.54 | -0.58 | -0.33 | 0.00 | 0.04 | 0.14 | 0.27 | 0.05 | 0.13 | 0.15 | -0.02 |
| #154 | NW | -1.00 | 0.04 | 0.03 | 0.16 | 0.08 | -0.49 | -0.44 | -1.05 | -1.02 | -0.66 | -0.01 | 0.66 | 0.35 | -0.52 | -1.05 | -0.74 |
| | NE | 0.00 | 0.38 | 0.33 | -0.27 | -0.09 | -0.16 | -1.17 | -0.65 | -1.15 | 0.07 | 0.42 | 0.60 | 0.56 | -1.14 | -1.14 | -1.09 |
| | SW | 0.28 | 0.41 | -0.04 | 0.12 | -0.06 | -0.44 | -1.23 | -1.12 | -1.21 | -1.17 | -1.19 | 0.21 | 0.51 | -0.07 | -0.41 | -0.02 |
| | SE | 0.03 | -0.34 | 0.12 | 0.20 | 0.50 | 0.23 | 0.24 | 0.16 | 0.16 | 0.29 | 0.43 | 0.08 | -1.13 | -0.22 | -1.05 | 0.04 |

| 3-conv2d player | | – | $2^1$ | $2^2$ | $2^3$ | $2^4$ | $2^5$ | $2^6$ | $2^7$ | $2^8$ | $2^9$ | $2^{10}$ | $2^{11}$ | $2^{12}$ | $2^{13}$ | $2^{14}$ | $2^{15}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| #118 | NW | -0.23 | -0.39 | -0.47 | 0.48 | 0.13 | -0.55 | -0.53 | -0.60 | -0.52 | -0.57 | -0.70 | -0.50 | -0.45 | -0.30 | -0.51 | -0.50 |
| | NE | -0.10 | -0.13 | 0.15 | 0.62 | 0.11 | -0.40 | -0.45 | -0.39 | -0.16 | -0.11 | -0.10 | -0.03 | -0.05 | 0.02 | -0.13 | 0.06 |
| | SW | 0.04 | -0.13 | -0.43 | 0.71 | -0.02 | -0.59 | -0.44 | -0.37 | -0.10 | -0.12 | 0.01 | -0.00 | 0.02 | 0.01 | 0.07 | -0.02 |
| | SE | -0.06 | -0.12 | 0.09 | 0.78 | -0.04 | -0.08 | -0.05 | 0.03 | 0.05 | 0.01 | -0.01 | 0.01 | 0.13 | 0.17 | 0.14 | 0.17 |
| #261 | NW | -0.84 | -0.34 | 0.22 | 0.28 | 0.17 | -0.06 | -0.90 | 0.40 | -0.89 | -0.21 | 0.04 | -0.03 | 0.14 | 0.18 | 0.12 | 0.24 |
| | NE | -0.25 | 0.00 | -0.32 | -0.39 | -0.21 | 0.15 | 0.14 | -0.52 | 0.47 | 0.42 | 0.07 | 0.13 | 0.06 | 0.11 | 0.14 | 0.12 |
| | SW | -0.02 | -0.01 | -0.24 | -0.35 | -0.18 | 0.16 | 0.27 | -0.47 | 0.44 | 0.40 | 0.17 | 0.24 | 0.18 | 0.30 | 0.27 | 0.22 |
| | SE | 0.10 | 0.17 | 0.26 | 0.30 | 0.27 | 0.03 | -0.95 | 0.30 | -0.37 | 0.02 | 0.08 | 0.11 | 0.12 | 0.03 | 0.07 | -0.01 |
| #196 | NW | -0.95 | -0.47 | -0.04 | 0.09 | -0.31 | -0.04 | 0.00 | 0.07 | 0.20 | 0.11 | 0.27 | 0.28 | 0.14 | 0.08 | -0.10 | 0.22 |
| | NE | -0.85 | -0.07 | -0.31 | -0.52 | 0.17 | 0.26 | 0.15 | 0.28 | 0.30 | 0.31 | 0.40 | 0.30 | 0.35 | 0.25 | 0.19 | 0.30 |
| | SW | 0.02 | 0.19 | 0.48 | 0.56 | 0.21 | -0.23 | -0.06 | -0.14 | -0.26 | -0.65 | -0.75 | -0.05 | -0.46 | -0.73 | -0.76 | -0.67 |
| | SE | -0.14 | 0.47 | 0.66 | 0.18 | 0.40 | 0.28 | -0.07 | -0.59 | -0.62 | -0.57 | -0.65 | -0.04 | -0.28 | 0.20 | 0.19 | 0.22 |
| # 27 | NW | -0.05 | -0.44 | -1.11 | -0.22 | 0.24 | 0.17 | 0.20 | 0.20 | 0.16 | 0.15 | 0.16 | 0.16 | 0.09 | 0.04 | 0.08 | 0.05 |
| | NE | -0.14 | -0.38 | -0.89 | -0.54 | 0.11 | 0.31 | 0.23 | 0.46 | 0.49 | 0.60 | 0.57 | 0.56 | 0.45 | 0.28 | 0.16 | 0.37 |
| | SW | 0.38 | 0.12 | 0.11 | -0.02 | -0.94 | -0.08 | 0.03 | -0.04 | -0.07 | -0.10 | -0.02 | -0.18 | -0.06 | -0.32 | -0.87 | -0.51 |
| | SE | 0.63 | 0.40 | 0.62 | 0.35 | -0.73 | -0.16 | 0.05 | -0.38 | -0.30 | -0.63 | -0.70 | -0.66 | -0.23 | -0.74 | -0.72 | -0.71 |
| # 10 | NW | -0.80 | 0.31 | -0.21 | 0.27 | -0.07 | 0.08 | -0.09 | -0.07 | 0.02 | 0.02 | 0.30 | 0.18 | 0.25 | -0.04 | -0.04 | 0.15 |
| | NE | -0.48 | 0.39 | -0.10 | 0.04 | -0.26 | 0.16 | 0.12 | 0.21 | 0.18 | 0.17 | 0.24 | 0.25 | 0.24 | 0.17 | 0.24 | 0.20 |
| | SW | 0.36 | 0.23 | -0.80 | 0.02 | -0.78 | -0.75 | -0.79 | -0.81 | -0.72 | -0.67 | -0.73 | -0.12 | -0.10 | 0.04 | 0.18 | -0.09 |
| | SE | -0.39 | 0.33 | -0.10 | 0.15 | -0.10 | 0.00 | 0.14 | 0.04 | 0.08 | 0.01 | 0.03 | 0.02 | -0.03 | 0.01 | 0.10 | -0.13 |
| # 77 | NW | -0.95 | 0.13 | 0.12 | 0.00 | -0.21 | -0.03 | 0.09 | -0.04 | -0.11 | -0.21 | -0.02 | -0.79 | -0.07 | -0.83 | -0.85 | -0.87 |
| | NE | 0.05 | 0.29 | 0.27 | 0.41 | -0.64 | -0.64 | -0.63 | -0.62 | -0.20 | -0.28 | -0.09 | -0.31 | -0.54 | -0.62 | -0.59 | -0.62 |
| | SW | 0.45 | 0.50 | -0.63 | -0.62 | -0.11 | -0.61 | -0.56 | -0.62 | -0.65 | -0.62 | -0.21 | -0.06 | -0.15 | -0.55 | -0.62 | -0.36 |
| | SE | -0.06 | -0.49 | 0.31 | 0.51 | 0.37 | 0.32 | 0.27 | 0.26 | 0.44 | 0.46 | 0.47 | 0.43 | 0.37 | 0.21 | 0.16 | -0.05 |

TABLE II
AVERAGE SCORE AND MAXIMUM SCORE OF CNN-BASED PLAYERS

| layers $k$ | average | maximum |
|---|---|---|
| 2 | 25,669 | 175,628 |
| 3 | 69,840 | 332,868 |
| 4 | 80,284 | 343,496 |
| 5 | 86,030 | 385,560 |

work [14], we used play records of those players as the training data.

We selected three players from the artifacts of our prior work [14] to generate the training data. These players utilized multi-staged N-tuple networks as the evaluation functions: the networks consisted of four 6-tuples and the game was split to eight stages based on the maximum number of tiles (The networks had 536,870,912 weights in total). The weights of the networks were adjusted by a reinforcement learning method, namely backward temporal coherent learning with restart strategy. The players selected moves by the 3-ply expectimax search. The average scores were 459,455, 463,660 and 460,069. Note that the difference of the players was only in the feature weights.

For the training data, we extracted $6 \times 10^8$ actions from the play records (from about 33,000 games). Each action consists of a state and the move that the player selected, which we used as the best (correct) move. The training data were shuffled before fed to supervised learning. Note that the play records were not biased in terms of symmetry since the N-tuple networks were trained in a completely symmetric manner.

Table II shows the average score and the maximum score (out of 10,000 test plays) of our CNN-based players after training with $6 \times 10^8$ actions. As we can see from the table, the players with three or more conv2d layers performed much better than the player with two conv2d layers.

## IV. ANALYSIS OF FIRST LAYER

In many CNN applications the first layer of CNNs works to extract features of the input. In this work, we first analyze the filters of the first conv2d layers by visualizing the weights.

We show some particular filters in Table III. The left column includes six filters in the first layer of the 2-conv2d network, and the right column includes ones of the 3-conv2d network. It is worth noting that we chose filters with similar weights form both networks and aligned them side by side. Each filter in the first layer is of size $2 \times 2 \times 16$ and the weights are put in two dimension in the figures: vertically by the position, i.e. upper left (NW), upper right (NE), lower left (SW), and lower right (SE), in order; horizontally by the corresponding value of tiles, i.e, the leftmost one for empty cell, and then 2, 4, . . . , 32678 to the right. Each cell is colored blue if the weight

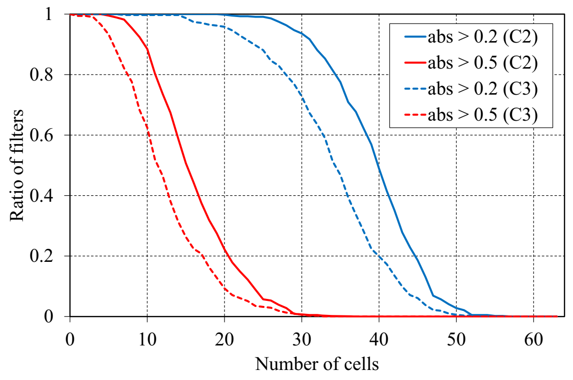


Fig. 3. Comparing the first layers with those weight values (C2 stands for 2-conv2d network and C3 for 3-conv2d network)

is positive and red if it is negative, where the color strength depends on the absolute value of the weight. The numbers after # denote the indices of filters, which will be referred to in later analyses.

As we can see in the table, we can confirm that some filters work for feature extraction.

- The filters #174 in the 2-conv2d network and #118 in the 3-conv2d network judge whether a target area of convolution has (multiple) 8-tiles.
- The filters #97 in the 2-conv2d network and #261 in the 3-conv2d network judge whether the upper left is a 128-tile and 128-tiles are not on the right or below of it.
- The filter #17 in the 2-conv2d network judges whether the upper row has tiles with large numbers and the lower row has empty cells or tiles with small numbers.
- The filter #432 in the 2-conv2d network judges a bit more particular situation where the lower row has empty cells.

On the other hand, we also have some filters for which it is hard to explain the extracted feature(s). The filter #154 in the 2-conv2d network returns a positive value if upper left, upper right, and lower left cells have large-numbered tiles (2048 or 4096) or small-numbered tiles (2, 4 or 8). Those large-numbered tiles and small-numbered tiles, however, work in a very different manner in game 2048. We expect that the filter mixes up two (or more) independent filters. In the 3-conv2d network (or more deeper ones), we have less instances of such a mix up.

We can find some more interesting facts from the comparison of the 2-conv2d and 3-conv2d networks. The most significant difference is that the filters in the 3-conv2d network include whiter cells, i.e., tiles with smaller absolute values, and extract more local features. To analyze it quantitatively, we plot in Fig. 3 the number of filters with respect to the number of weights with absolute value being larger than some thresholds. From the plot, the 3-conv2d network has more filters with smaller weights than those in the 2-conv2d network. It is worth noting that we also plotted similar graphs for the 4-conv2d and 5-conv2d networks and the results were almost the same as that of the 3-conv2d network.

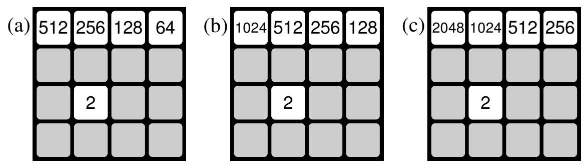


Fig. 4. States used in generalization analysis

## V. Analysis of Generalization Ability

Regardless of the success of N-tuple networks for the game 2048, we study NN-based approach for its possible generalization ability. In this section we try to analyze the generalization ability of our networks.

There could be several aspects of generalization in game 2048. What we want to inspect in this work is robustness to doubling (or halving) consecutive tiles. Consider the three states shown in Fig. 4 (a)–(c). Human players can easily capture a common pattern in these states: that is, the largest tile is at a corner and other large tiles are put on an edge in descending order. However, it is not easy for computer players to capture the pattern unless specifically programmed, and usual N-tuple networks deal with them independently. "Whether our CNNs compute probabilities in a generalized manner for these states?" is the question here.

To answer the question, we apply a backward analysis from the results of CNNs. Note that we should not choose "down" for the three states from Heuristics 1 and 2 in Section II. Therefore, starting from the results (before the softmax) for "down", we deconvolute computations toward the input. The deconvolution is calculated in the following step:

1) For each corresponding pair in the convolution computation, we calculate the product of the value and the weight.
2) Compute the absolute sum of those products.
3) Compute the contribution as the ratio of absolute value against the absolute sum, and pick up indices with the highest contribution.

For instance, consider the case with the 2-conv2d network for the input being Fig. 4 (b). Starting from the result (before the softmax) for "down", we deconvolute the full-connect layer and pick up top five indices before the full-connect layer (middle column in Fig. 5). The top one is as follows.

index: (0,0)#163, contribution ratio: 15.4%

We then continue the deconvolution of the second conv2d layer from the indices. Here we compute the contribution by the product of those in the second conv2d layer and in the full-connect layer. Since an input of the conv2d layer can contribute to two or more outputs, the contribution relation forms a DAG and we compute the sum of contribution ratio for such cases. In the deconvolution of the second conv2d layer, we pick up top six indices (left column in Fig. 5).

We conducted similar computation for the three states in Fig. 4 (a)–(c) for both 2-conv2d and 3-conv2d networks. We

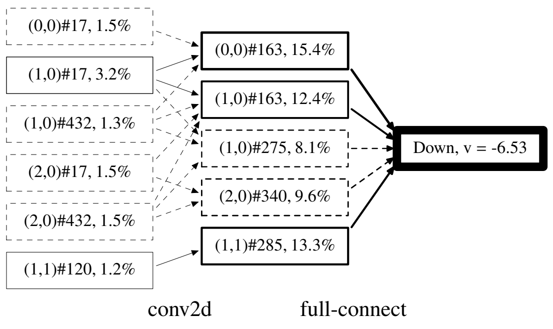


Fig. 5. Backward analysis for the 2-conv2d network

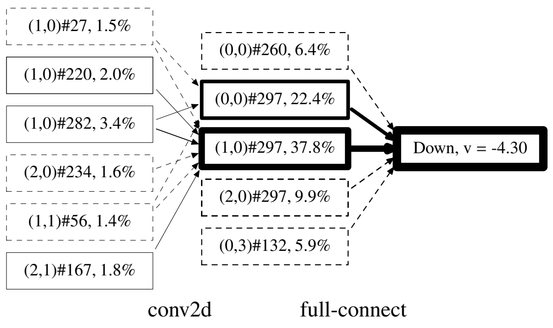


Fig. 6. Backward analysis for the 3-conv2d network

then found an important fact in the results of deconvolution: despite the difference of numbers of tiles, both networks share some paths in the computation. In particular, those shared paths have rather high contribution ratios. In Fig. 5, we draw the DAG of computation and the boxes or arrows in solid line are those shared for all the three states. Similarly, we draw the DAG of computation for the 3-conv2d network in Fig. 6.

At the full-connect layer of the 2-conv2d network, the top three inputs have about 40% contribution for all the three states. This means that the 2-conv2d network obtains some general knowledge about the game. Looking at the conv2d layer, large amount of contribution comes from channels #17 and #432, both of which judge large tiles on the upper row and small tiles or empty cells on the lower row. This basically coincides with our understanding to the states.

At the full-connect layer of the 3-conv2d network, the top two inputs have about 60% contribution for all the three states. This fact that a small number of inputs contribute more means that the network captures the features of the states well. We tried to trace the network to the first conv2d layer, but a large number of inputs went through the conv2d layers and we found it too hard to explain the whole network as we did for the 2-conv2d network.

Finally, we would like to note another important difference between the 2-conv2d and 3-conv2d networks. In the 2-conv2d network, more outputs of the first conv2d layer are connected to different inputs of the full-connect layer. We may

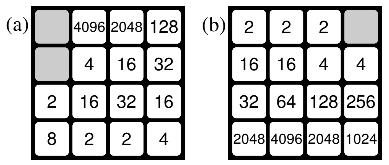


Fig. 7. Hand-designed and real-play states

understand it that a feature can be used for several places, or it failed to decoupling features. We will see in the next section that the insufficient decoupling could lead to selection of an inappropriate move.

## VI. Inappropriate Plays by CNNs

### A. For Hand-designed State

Consider a hand-designed state in Fig. 7 (a). Following Heuristic 1 in Section II, human players would first take the move "left" into account. However, the 2-conv2d and 3-conv2d networks returned

2-conv2d: [up: 0.243, right: 0.405, down: 0.104, left: 0.248]
3-conv2d: [up: 0.226, right: 0.524, down: 0.000, left: 0.250]

and both chose "right" as the best move.

We consider two reasons of these results. One reason is the limited features extracted in our CNNs. When human players make a judgment, they know that the 4096-tile is the maximum one and apply heuristics to move it to a corner. However, the 2-conv2d or 3-conv2d networks are too shallow to judge a tile being the maximum one. Another reason is the bias in the training data. It is quite rare that sliding toward empty cells is the best move. Therefore, our CNNs did not learn that moving toward empty cells could be a good move.

### B. For State from Real Play

We collected 77,253 states from 20 games of the 5-conv2d player (we removed the first 100 states for each game). For each state, we applied 2-, 3-, 4-, and 5-conv2d networks to compute moves, and looked for states for which only the 2-conv2d network selected a different move. For 5,915 (7.7%) states, the 2-conv2d network returned a different move from the other networks[5].

Figure 7(b) is an instance. For this state, the best move is "left" and the 3-, 4-, and 5-conv2d networks returned the move. The 2-conv2d network, however, returned "up", which is definitely the worst move.

We investigated the reason of selecting "up" with the backward analysis similar to the previous section, and found that the value at index (0,3)#154 after the first conv2d layer contributed most to the result. The filter #154 in the first conv2d has, as seen in Table III, two possibility for positive outputs: with 2048- and/or 4096-tiles, and smaller 2-, 4-, or 8-tiles. If the tiles in the third row were small-numbered ones, selecting "up" would be reasonable. However, in this case,

[5]All the selected moves were the same for 22,807 states (29.5%).

the 2048- and 4096-tiles activated the path, which yielded an inappropriate move.

This could be considered as a kind of overfitting. As we have seen in Section V, the 2-conv2d network involves several intermediate values to contribute in the full-connect layer. This means, in the training phase, several weights are adjusted at a time. In the case of networks with three or more conv2d layers, not so many intermediate values contribute in the full-connect layer, and thus the updates are localized.

## VII. Related Work

The most successful approach to computer 2048 players is based on N-tuple networks (NTNs) and reinforcement learning methods. Thanks to the simple design and implementation of the NTNs, we can easily increase the number of weights to improve the performance of players [15], [16]. We can also extend NTNs by so-called multi-staging technique [17].

Though NTNs have worked fine, a weakness remains: missing generalization. Since the weights are basically independent from each other, NTNs do not obtain some important property of the game. Weight promotion [11], [14], which initializes a first-accessed weight with a certain existing one, can be considered as a human-aided solution to this issue. A more affirmative reuse of feature weights could achieve even a 65536-tile [18].

Behind the success of NTNs, (deep) neural networks have been less studied or utilized for the development of 2048 players. Guei et al. [13] first tried to develop 2048 players based on convolutional neural networks. They developed two networks, one with $2 \times 2$ filters and the other with $3 \times 3$ filters. The weights in these networks are adjusted by reinforcement learning methods. The best average score achieved with the $2 \times 2$ network was about 11,400 and with the $3 \times 3$ network about 5,300.

The neural network developed by tjwei [19] consists of two convolution layers followed by a full-connected layer. The differences were in the shape ($2 \times 1$ and $1 \times 2$) and the number ($512 \times 2$ for the first layer and $4096 \times 4$ for the second layer) of filters. The average score achieved was 85,351 when trained with a supervised learning method.

## VIII. Conclusion

In this paper, we analyzed the inner working of our CNN-based players for 2048, which was developed in our previous study [12]. We first visualized weights in the first layers to check that the first layers successfully extracted features of states. We then traced the networks in the backward direction to see how outputs were computed for some particular states.

We obtained several findings in this work. The most important one was about generalization ability: Both 2- and 3-conv2d networks had some generalized knowledge, which was encoded in some computation paths. Some findings on the 2-conv2d network could partially explain the poor performance of the 2-conv2d player.

Our future work includes extension of our CNN-based player in terms of the size of networks as well as the structure itself. We hope the analyses in this paper could help to extend the NN-based computer players for 2048.

## References

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[2] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, pp. 354–359, 2017.

[3] M. Lai, "Giraffe: Using deep reinforcement learning to play Chess," Master's thesis, Imperial College London, 2015, *arXiv*, vol. 1509.01549v1.

[4] O. E. David, N. S. Netanyahu, and L. Wolf, "DeepChess: End-to-end deep neural network for automatic learning in Chess," in *International Conference on Artificial Neural Networks and Machine Learning (ICANN 2016)*, 2016, pp. 88–96.

[5] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering Chess and Shogi by self-play with a general reinforcement learning algorithm," *arXiv*, vol. 1712.01815, 2017.

[6] N. Yakovenko, L. Cao, C. Raffel, and J. Fan, "Poker-CNN: A pattern learning strategy for making draws and bets in Poker games," *arXiv*, vol. 1509.06731, 2015.

[7] M. Moravcík, M. Schmid, N. Burch, V. Lisý, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. H. Bowling, "DeepStack: Expert-level artificial intelligence in heads-up no-limit poker," *Science*, vol. 356, no. 6337, pp. 508–513, 2017.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," in *NIPS Deep Learning Workshop*, 2013.

[9] G. Cirulli, "2048," http://gabrielecirulli.github.io/2048/, 2014.

[10] M. Szubert and W. Jaśkowski, "Temporal difference learning of N-tuple networks for the game 2048," in *2014 IEEE Conference on Computational Intelligence and Games*, 2014, pp. 1–8.

[11] W. Jaśkowski, "Mastering 2048 with delayed temporal coherence learning, multi-stage weight promotion, redundant encoding and carousel shaping," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 10, no. 1, pp. 3–14, 2018.

[12] N. Kondo and K. Matsuzaki, "Playing game 2048 with deep convolutional neural networks trained by supervised learning," *Journal of Information Processing*, 2018, under revision after conditional accept.

[13] H. Guei, T. Wei, J.-B. Huang, and I.-C. Wu, "An early attempt at applying deep reinforcement learning to the game 2048," in *Workshop on Neural Networks in Games*, 2016.

[14] K. Matsuzaki, "Developing 2048 player with backward temporal coherence learning and restart," in *Proceedings of Fifteenth International Conference on Advances in Computer Games (ACG2017)*, 2017, pp. 176–187.

[15] K. Oka and K. Matsuzaki, "Systematic selection of N-tuple networks for 2048," in *Proceedings of 9th International Conference on Computers and Games (CG2016)*, ser. Lecture Notes in Computer Science, vol. 10068. Springer, 2016, pp. 81–92.

[16] K. Matsuzaki, "Systematic selection of N-tuple networks with consideration of interinfluence for game 2048," in *Proceedings of the 2016 Conference on Technologies and Applications of Artificial Intelligence (TAAI 2016)*, 2016, pp. 186–193.

[17] I.-C. Wu, K.-H. Yeh, C.-C. Liang, C.-C. Chang, and H. Chiang, "Multi-stage temporal difference learning for 2048," in *Technologies and Applications of Artificial Intelligence*, ser. Lecture Notes in Computer Science, vol. 8916, 2014, pp. 366–378.

[18] H. Guei and I.-C. Wu, personal communication, 2018.

[19] tjwei, "A deep learning ai for 2048," https://github.com/tjwei/2048-NN.