# W którą stronę ruszyć?
# Sieć neuronowa przewidująca następny ruch w grze 2048

Zuzanna Maciszewska, Aleksandra Piłat, Jędrzej Sarna

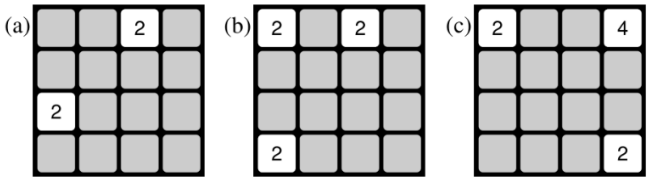
1 kwietnia 2025

## 1 Cel projektu

2048 to gra typu przesuń i połącz, która po wypuszczeniu zyskała wyjątkową popularność. Według twórcy gry, w pierwszych trzech tygodniach od jej wypuszczenia ludzie spędzili łącznie 3000 lat na rozgrywkach.

Wielu "graczy komputerowych" zostało opracowanych na przestrzeni ostatnich lat, większość oparta na uczeniu przez wzmacnianie. Naszym celem jest stworzenie modelu przewidującego następny ruch stosując uczenie nadzorowane. W dużej mierze opieramy się o badania prowadzone przez prof. Kiminori Matsuzaki, w szczególności o artykuł [**1**].

## 2 Gra 2048




Rysunek 1: Stany planszy gry

(a) Stan początkowy planszy. Dwie płytki umieszczane są w losowych polach.

(b) Plansza po wykonaniu pierwszego ruchu *w górę*, w lewym dolnym rogu pojawia się nowa płytka 2.

(c) Plansza po wykonaniu drugiego ruchu *w prawo*. Dwie płytki o numerze 2 łączą się tworząc płytkę o numerze 4, otrzymujemy wynik 4. W lewym górnym rogu pojawia się nowa płytka 2.

Gra 2048 jest rozgrywana na siatce 4×4. Celem oryginalnej gry 2048 jest dotarcie do płytki 2048 poprzez przesuwanie i łączenie płytek na planszy zgodnie z powyższymi zasadami. W stanie początkowym (rys. 1 (a)), losowo umieszczane są dwie płytki o numerach 2 (prawdopodobieństwo pojawienia się liczby 2 to $P(2) = 0,9$) lub 4 (prawdopodobieństwo pojawienia się liczby 4 to $P(4) = 0,1$). Gracz wybiera kierunek (w górę, w prawo, w dół lub w lewo), a następnie wszystkie płytki przesuwają się w wybranym kierunku. Gdy zderzą się dwa pola o tym samym numerze, powstaje nowy kafelek o wartości sumy tych dwóch pól, a wartość tej sumy wlicza się do ogólnego wyniku gry. Po każdym ruchu w pustej komórce pojawia się płytka o numerze 2 ($P(2) = 0,9$) lub 4 ($P(4) = 0,1$).

Jeśli gracz nie może przesunąć płytek, gra się kończy. Gdy osiągniemy pierwsze pole o numerze 1024, wynik w grze wynosi około 10 000. Podobnie, wynik wynosi około 21 000 dla płytki 2048, około 46 000 dla płytki 4096, około 100 000 dla płytki 8192, około 220 000 dla płytki 16384 i około 480 000 dla płytki 32768.

## 3 Opis i przygotowanie danych

Dane treningowe pochodzą z artykułu [**1**] i zostały udostępnione przez prof. Matsuzaki. Zostały wygenerowane przez trzech istniejących graczy komputerowych opracowanych w poprzednich badaniach [**3**] którzy uzyskali bardzo

dobre wyniki. Do danych treningowych zostało wybrane $6 \cdot 10^8$ ruchów z zapisanych rozgrywek (ruchy zostały wyodrębnione z 54000 rozgrywek). Każdy ruch składa się ze stanu planszy do zagrania, kierunku wybranego przez gracza oraz miejsca powstania nowej płytki. Dane treningowe zostały przetasowane przed przekazaniem ich do procesu uczenia. Ponieważ w udostępnionym zbiorze danych każdy ruch jest niezależny od pozostałych, nie miałoby sensu implementowanie sieci rekurencyjnych.

Dane mają postać plików tekstowych, które program czyta linijka po linijce. W każdej linijce znajduje się odpowiednio:

- Litera r oddzielająca każdą kolejną zapisaną akcję.
- 16 liczb, każda oznacza wartość na danym polu w tablicy 4x4, zapisana jako odpowiednia potęga dwójki (przy czym 0 oznacza tutaj puste pole).
- Kierunek ruchu (0: w górę, 1: w prawo, 2: w dół, 3: w lewo).
- Pozycja ostatnio umieszczonej płytki.

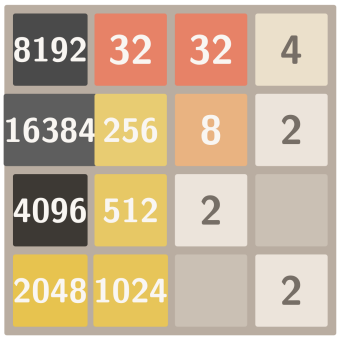
Warto również podkreślić, że w zbiorze danych nie ma tablic symetrycznych (powstałych jako obrót danej planszy), zamiast generować je i dokładać do zbioru treningowego uwzględnimy to w trakcie rozgrywki prowadzonej przez wytrenowany model.

## Przykład kodowania planszy gry 2048 dla sieci neuronowej

Weźmy pod uwagę pewien przykład ze zbioru treningowego (przypomnijmy, że są one potasowane), aby pokazać, jak zostaje kodowana plansza dla sieci neuronowej. Oto przykład jednego wiersza z pliku .txt:

```
r 13 5 5 2 14 8 3 1 12 9 1 0 11 10 0 1 : 3 15
```

Odpowiadająca plansza w grze będzie miała następującą postać:




Przystępujemy teraz do przedstawienia tej planszy jako tensor o wymiarach $16 \times 4 \times 4$. Każdy kanał tensora będzie odpowiadał jednej z możliwych wartości pól na planszy: puste pola, liczba 2, liczba 4 itd. aż do liczby 32768.

Przyjmijmy, że:

- Kanał 0 odpowiada pozycjom pustych pól (wartość 0)
- Kanał 1 odpowiada pozycjom pól z wartością 2 (wartość 2)
- Kanał 2 odpowiada pozycjom pól z wartością 4 (wartość 4)
- Kanał 3 odpowiada pozycjom pól z wartością 8 (wartość 8)
- $\vdots$
- Kanał 15 odpowiada pozycjom pól z wartością 32768 (wartość 32768)

W ten sposób otrzymujemy reprezentację planszy jako tensor o wymiarach $16 \times 4 \times 4$, gdzie każdy kanał jest macierzą $4 \times 4$, zawierającą informacje o obecności odpowiednich wartości na poszczególnych polach planszy.Poniżej znajduje się poglądowa reprezentacja planszy w formie tensora:

$$\left[\begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix},\begin{bmatrix}0&0&0&0\\0&0&0&1\\0&0&1&0\\0&0&0&1\end{bmatrix},\begin{bmatrix}0&0&0&1\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix},\begin{bmatrix}0&0&0&0\\0&0&1&0\\0&0&0&0\\0&0&0&0\end{bmatrix}\right]$$

$$\left[\begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix},\begin{bmatrix}0&1&1&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix},\begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix},\begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix}\right]$$

$$\left[\begin{bmatrix}0&0&0&0\\0&1&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix},\begin{bmatrix}0&0&0&0\\0&0&0&0\\0&1&0&0\\0&0&0&0\end{bmatrix},\begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&1&0&0\end{bmatrix},\begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\1&0&0&0\end{bmatrix}\right]$$

$$\left[\begin{bmatrix}0&0&0&0\\0&0&0&0\\1&0&0&0\\0&0&0&0\end{bmatrix},\begin{bmatrix}1&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix},\begin{bmatrix}0&0&0&0\\1&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix},\begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix}\right]$$

Również dokonujemy kodowania ruchu gracza. W omawianym przykładzie ze zbioru treningowego, gracz dokonał ruchu zakodowanego jako liczba 3 (w lewo). Dla sieci neuronowej zostaje to zakodowane jako tensor rozmiaru $1 \times 4$, dokładniej dla przykładu tensor będzie postaci

$$\begin{bmatrix}0&0&0&1\end{bmatrix}$$

# 4 Architektura sieci

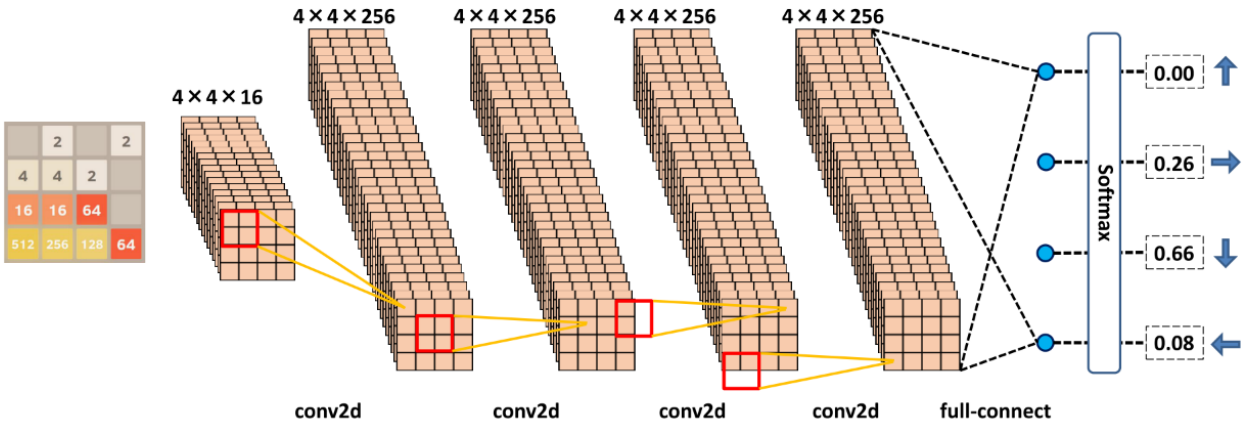
## Sieć konwolucyjna

W pierwszej kolejności wybraliśmy architekturę sieci konwolucyjnych. Chociaż sieci konwolucyjne są najczęściej stosowane przy analizie obrazów, w omawianym problemie również znajdują zastosowanie. Warstwy konwolucyjne umożliwiają sieci neuronowej wykrywanie wzorców na różnych skalach, co jest przydatne do identyfikacji zarówno małych, jak i dużych struktur na planszy. Dzięki temu sieć może zrozumieć zarówno pojedyncze płytki, jak i większe bloki, co jest kluczowe dla podejmowania decyzji dotyczących ruchów.

Plansza wejściowa jest kodowana w sposób opisany wcześniej w pracy. Następnie k razy (gdzie k to liczba warstw konwolucyjnych) stosujemy filtry $2 \times 2$ a szerokość kroku wynosi jeden. Aby utrzymać rozmiar planszy $4 \times 4$ stosujemy asymetryczny padding (dodajemy kolumnę zer z prawej strony planszy i na dole planszy), w PyTorchu implementujemy to dodając w metodzie forward `x = F.pad(x, (1, 0, 1, 0))` przed każdą warstwą konwolucyjną. Dzięki małemu rozmiarowi planszy oraz paddingowi, który nie zmienia wymiarów, nie stosujemy żadnej z operacji typu pooling. Między warstwami konwolucyjnymi stosujemy funkcję aktywacji `ReLu`. Kryterium optymalizacji `nn.CrossEntropy()` w PyTorchu przyjmuje jako input surowe wyjście z ostatniej warstwy liniowej, stosując `Softmax()` domyślnie, dlatego nie używamy w architekturze sieci funkcji aktywacji po ostatniej w pełni połączonej warstwie liniowej. Liczbę kanałów warstw konwolucyjnych dobieramy tak, aby otrzymane modele miały podobną liczbę wag, co pozwoli nam porównać jak istotna jest głębokość sieci CNN. Liczbę wag obliczamy definiując funkcję `count_parameters(model)` która iteruje przez wszystkie parametry modelu i zwraca ich sumę.

| k | Liczba kanałów wejściowych | Liczba wag |
|---|---|---|
| 4 | 256 | 820228 |
| 5 | 222 | 818074 |

Tabela 1: Tabela przedstawiająca liczbę warstw konwolucyjnych, liczbę kanałów wejściowych i liczbę wag dla danego modelu.

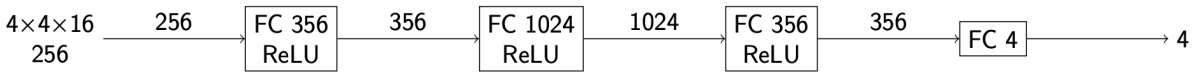



Rysunek 2: Architektura sieci DCNN z czterema warstwami konwolucyjnymi i asymetrycznym paddingiem

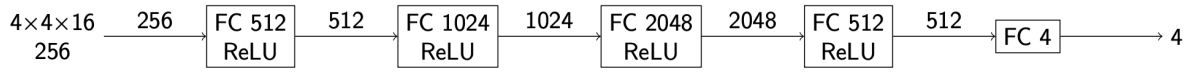
Dla pięciu warstw konwolucyjnych postanowiliśmy również rozważyć zastosowowanie symetrycznego paddingu, pozwalając aby rozmiar tablic nie był utrzymany na poziomie 4 x 4. Stosując symetryczny padding oraz długość kroku jeden, po przejściu przez każdą następną warstwę konwolucyjną otrzymamy na wyjściu tablice rozmiaru o jeden większe (czyli po pierwszej warstwie dostaniemy tablice 5 x 5, po drugiej 6 x 6 itd.). Liczba wag w tym modelu jest równa 875794 (około $5 \cdot 10^4$ więcej niż stosując padding asymetryczny).

### Wielowarstwowy perceptron

Ponieważ tablice wejściowe nie są zbyt duże, postanowiliśmy zbudować również sieć MLP, stosując uczenie nadzorowane (sieć MLP trenowana przez wzmacnianie otrzymywała bardzo dobre wyniki [**2**], jednak długość procesu uczenia i ilość danych była dużo większa). Parametry czterowarstwowej sieci MLP dobraliśmy tak, aby liczba wag była porównywalna z wcześniej zbudowanymi modelami CNN. Dla architektury z rysunku 3 liczba wag w modelu jest równa 823388. Postanowiliśmy również odtworzyć sieć MLP (rys. 4) która osiągnęła dobre wyniki w eksperymencie wykorzystującym uczenie przez wzmacnianie. Jest to szersza sieć z dodatkową warstwą ukrytą, a liczba wag w tym modelu jest równa 3807236.




Rysunek 3: Architektura 4-warstwowej sieci MLP

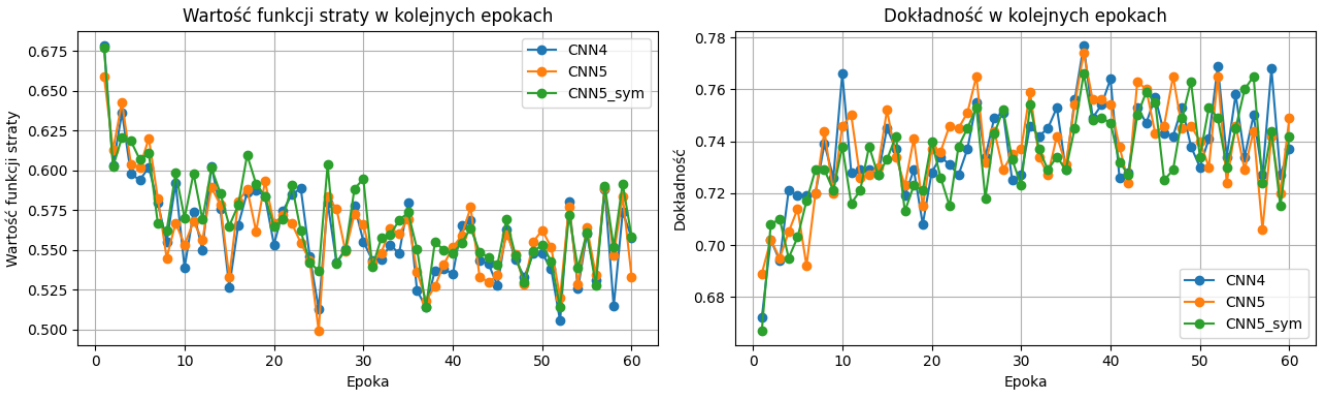



Rysunek 4: Architektura 5-warstwowej sieci MLP

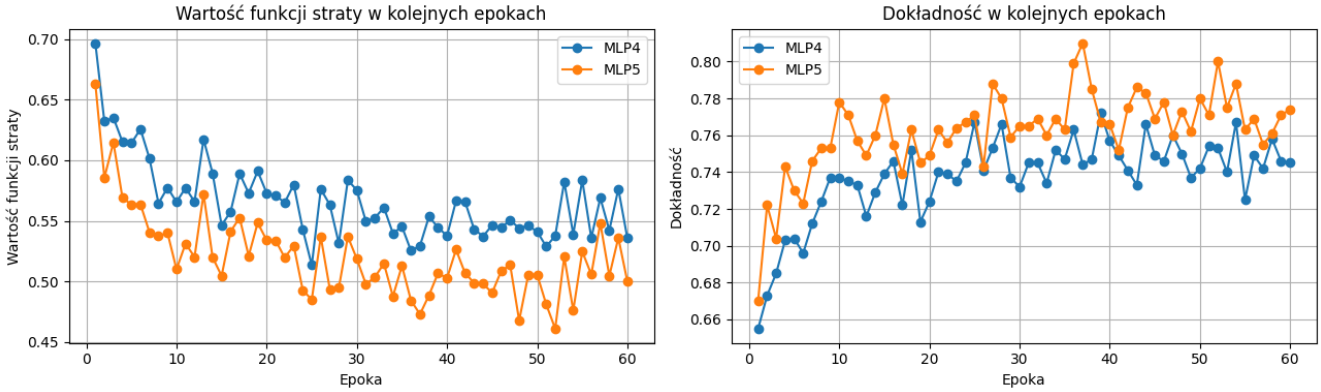
## 5 Ocena wyników

### Wyniki na zbiorze treningowym

Podczas uczenia modeli, zapisywaliśmy co epokę wartości funkcji straty oraz dokładność modelu. Poniżej przedstawione są wykresy zmiany tych miar wraz ze wzrostem liczby epok. Można zauważyć, że wartości te nie spadają gładko i momentami nawet wzrastają. Zauważając ten problem, zmienialiśmy współczynnik uczenia (learning rate) na inny, jednak wyniki były tylko gorsze. Jednym z powodów tak dużej wariancji błędów i dokładności może być zbyt mały rozmiar batcha. W zbiorze treningowym mamy bardzo dużo różnych stanów tablic, ponieważ bierzemy wymieszane ruchy z różnych gier, a jedna gra ma często do 20 000 ruchów, podczas gdy rozmiar batcha wynosi 1000.



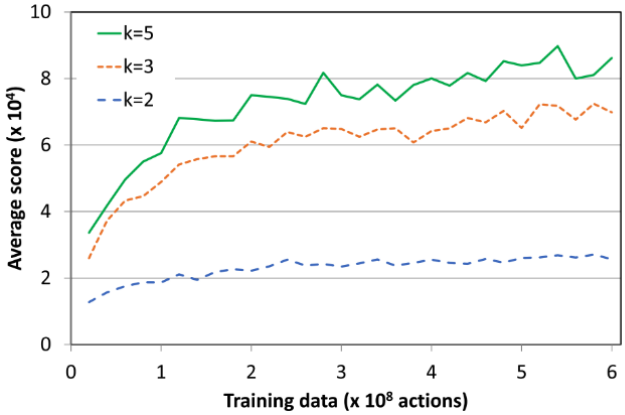

Rysunek 5: Wykresy wartości funkcji straty oraz dokładności w kolejnych epokach dla sieci CNN




Rysunek 6: Wykresy wartości funkcji straty oraz dokładności w kolejnych epokach dla sieci MLP

Wyniki dokładności mogą nie wydawać się zadowalające. Należy jednak zwrócić uwagę na naturę problemu. Postawiony problem klasyfikacji nie jest zero-jedynkowy. Choć chcemy, aby model dobrze odwzorowywał ruchy wcześniej opracowanych "graczy komputerowych", same te modele nie są idealne. Wysoka dokładność na zbiorze treningowym (przykładowo na poziomie 90-100%) nie jest zatem wymagana, aby model był "dobry". Należy dodatkowo sprawdzić, jak model zachowuje się na zbiorze testowym, czyli w praktycznym zastosowaniu, jakim jest gra.

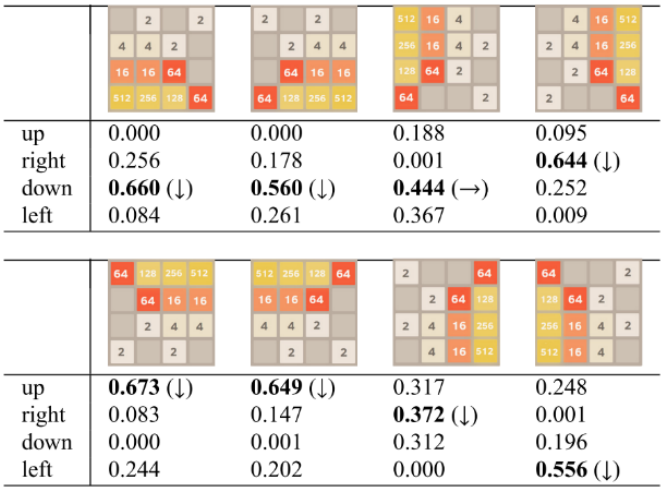
Dodatkowo, choć powyższe wykresy nie do końca wskazują na to, do gier wybieramy modele po 60. epoce. Jest to uzasadnione eksperymentami przeprowadzonymi w literaturze [**1**], gdzie najlepsze wyniki w samej grze uzyskały modele wytrenowane na maksymalnej ilości dostępnych danych.




Rysunek 7: Wykres z artykułu [**1**] średniego wyniku w grze w zależności od ilości danych treningowych dla sieci CNN z k-warstwami konwolucyjnymi

## Wyniki w grze

Ponieważ w procesie uczenia w zbiorze treningowym nie uwzględnialiśmy tablic symetrycznych, uwzględniamy to w trakcie rozgrywki. Tworzymy plansze które powstały jako symetryczna transformacja planszy wejściowej, dla każdej planszy dokonujemy predykcji wybierając maksymalną wartość outputu zwróconego przez model i wybieramy ruch który uzyskał największą ilość głosów. W przypadku gdy dwa lub więcej ruchów ma taką samą liczbę głosów, predykcji dokonujemy na podstawie sumy outputów zwróconych przez model.



| | | | | |
|---|---|---|---|---|
| up | 0.000 | 0.000 | 0.188 | 0.095 |
| right | 0.256 | 0.178 | 0.001 | **0.644** (↓) |
| down | **0.660** (↓) | **0.560** (↓) | **0.444** (→) | 0.252 |
| left | 0.084 | 0.261 | 0.367 | 0.009 |

| | | | | |
|---|---|---|---|---|
| up | **0.673** (↓) | **0.649** (↓) | 0.317 | 0.248 |
| right | 0.083 | 0.147 | **0.372** (↓) | 0.001 |
| down | 0.000 | 0.001 | 0.312 | 0.196 |
| left | 0.244 | 0.202 | 0.000 | **0.556** (↓) |

Rysunek 8: Rysunek przykładowego stanu planszy i jej symetrii z artykułu [**1**].

Biorąc pod uwagę naturę problemu, jakość wytrenowanych modeli będziemy oceniać na podstawie wyników osiągniętych w grze. Po wytrenowaniu modelu zapisujemy go, a następnie przeprowadzamy 1000 gier testowych. Porównamy modele na podstawie średniego osiągniętego wyniku, najwyższego osiągniętego wyniku oraz frakcji wygranych (czyli w ilu grach modelowi udało się dojść do płytki z numerem 2048). Zaimplementowaliśmy również gracza wybierającego ruch całkowicie losowo, aby porównać czy predykcje modeli rzeczywiście prowadzą do lepszych wyników w grze.

Tabela 2: Porównanie wyników różnych modeli

| Model | Średni wynik | Najwyższy wynik | Frakcja wygranych | Najwyższa płytka |
|---|---|---|---|---|
| Random | 1089.22 | 3440 | 0.0 | 256 |
| CNN4 | 82657.752 | 342928 | 85.6 | 16384 |
| CNN5 | **85494.56** | **374528** | **86.4** | 16384 |
| CNN5_sym | 67870.68 | 330528 | 76.1 | 16384 |
| MLP4 | 31339.144 | 298908 | 40.9 | 16384 |
| MLP5 | 46047.164 | 296492 | 48.5 | 16384 |

Tabela 3: Maksymalna wartość płytki osiągnięta w 1000 gier

| Model | $\leq 256$ | 512 | 1024 | 2048 | 4096 | 8192 | 16384 |
|---|---|---|---|---|---|---|---|
| CNN4 | 38 | 44 | 62 | 183 | 391 | 271 | 11 |
| CNN5 | 32 | 31 | 73 | 174 | 395 | 279 | **16** |
| CNN5_sym | 95 | 65 | 79 | 163 | 414 | 176 | 8 |
| MLP4 | 299 | 136 | 156 | 176 | 174 | 57 | 2 |
| MLP5 | 320 | 93 | 102 | 123 | 224 | 132 | 6 |

Jak możemy zauważyć w tabeli 2 5-cio warstwowa sieć konwolucyjna z asymetrycznym paddingiem radzi sobie najlepiej w rozgrywkach pod kątem wszystkich rozważanych wyników. Widzimy również, że zastosowanie asyme-

trycznego paddingu w sieciach konwolucyjnych znacznie poprawia wyniki, frakcja wygranych modelu CNN5 jest dużo większa, niż w przypadku modelu CNN5_sym.

Wyniki uzyskane przez obie sieci MLP wypadają znacznie gorzej niż dla sieci CNN, co sugeruje, że sieci konwolucyjne dużo lepiej generalizują wiedzę zdobytą w procesie uczenia. Ich struktura pozwala lepiej wyłapywać zależności w danych. Porównując MLP4 i MLP5 między sobą, szeroka sieć uzyskała znacznie lepszy średni wynik w rozgrywkach, zarówno jak i większą frakcję wygranych gier. Większą liczba neuronów w każdej warstwie prawdopodobnie pozwala sieci na lepszą generalizację.

Niestety żadnemu modelowi nie udało się osiągnąć płytki 32768, z drugiej strony każdy wytrenowany model jest dużo lepszy niż "gracz losowy".

# 6 Podsumowanie

W naszym projekcie opracowaliśmy graczy komputerowych korzystając z biblioteki PyTorch dla gry 2048 w oparciu o głębokie konwolucyjne sieci neuronowe oraz Wielowarstwowy Perceptron, wykorzystując uczenie nadzorowane. Dla sieci CNN sprawdziliśmy różne liczby warstw konwolucyjnych jak i różne typy paddingu, a dla sieci MLP porównaliśmy węższą sieć z liczbą wag podobną do rozważanych sieci CNN, do szerokiej sieci z dużo większą liczbą wag. Sieci te zostały wytrenowane na podstawie zapisów istniejących silnych graczy komputerowych. Najlepszym stworzonym przez nas graczem okazał się model CNN5, uzyskując najwyższy wynik równy 374528 i najczęściej uzyskując płytkę 16384 w symulowanych rozgrywkach. Gracze oparci o sieci MLP uzyskiwali dużo gorsze wyniki niż sieci CNN, wskazując na to, że sieci konwolucyjne są w stanie dużo lepiej generalizować wiedzę w tak złożonym problemie.

# Literatura


[1] Naoki Kondo, Kiminori Matsuzaki, Playing Game 2048 with Deep Convolutional Neural Networks Trained by Supervised Learning, Journal of Information Processing, 2019, Volume 27, Pages 340-347

[2] Kiminori Matsuzaki, Developing Value Networks for Game 2048 with Reinforcement Learning, Journal of Information Processing, 2021, volume 29, p. 336-346

[3] Matsuzaki, K.: Developing 2048 Player with Backward Temporal Co- herence Learning and Restart, Proc. 15th International Conference on Advances in Computer Games (ACG2017), pp.176–187 (2017).

[4] Guei, H., Wei, T., Huang, J.-B. and Wu, I.-C.: An Early Attempt at Applying Deep Reinforcement Learning to the Game 2048, Workshop on Neural Networks in Games (2016).