UiA University of Agder

Master

A Supervised Attention-Based Multiclass Classifier for Tile Discarding in Japanese Mahjong

TRONG DUC TRUONG

SUPERVISOR
Morten Goodwin

University of Agder, 2021
Faculty of Engineering and Science
Department of Engineering and Sciences

Abstract

Japanese Mahjong, an imperfect-information, multiplayer, multi-round game, has become an area of interest for the AI community due to its immense imperfect-information space and complex playing and scoring rules. In 2020 Microsoft unveiled the Mahjong AI Suphx that managed to outdo most of the top human players in Tenhou.net, the most popular platform for Japanese Mahjong. With supervised learning, Suphx's discard model reached a prediction accuracy of 76.7% when tested on game logs from Tenhou.net.

Two recurring problems with state-of-the-art Mahjong AIs, including Suphx, are their heightened architecture complexity and the sizeable data structure. These problems make it less feasible to replicate these experiments with limited hardware. We propose two models: MHA-B and MHA-S. Both use the same architecture with a multi-head attention layer but are trained on different training sets. Furthermore, we suggest a data structure that is a fraction of the size of contemporary alternatives. A smaller data structure implies fewer data values for the models to focus on making the models converge faster.

When tested on game logs from Tenhou.net, MHA-B and MHA-S reach a prediction accuracy of 66.7% and 65.2%, respectively. Although somewhat subpar compared to state-of-the-art models' results, our approach yields notable results considering the non-complex model architecture and the restricted data structure.

Acknowledgements

I am indebted to all who have supported me throughout this project.

I am grateful to the University of Agder for providing me with the resources to write this thesis and a place to meet new people that have become very important to me.

I want to thank Morten Goodwin for his invaluable guidance and support throughout this project; I was never alone.

I want to thank my older brother, Tam, for helping me in a pinch yet again; my partner in crime, Jiu-Wai, for his eloquent way of words; my childhood friend, An, for his helpful review; my selfless friend, Tinh, for sharing his computer for my cause; and Ingunn for providing me coffee mugs.

Words cannot begin to describe how grateful I am for my family, so I'll stop here.

Contents

Figures

Tables

Code Listings

Chapter 1

Introduction

Past Artificial Intelligence (AI) in games seems to suggest that humans have an innate drive to be challenged. Over these past two decades, we have seen the rise of machines in two-player games such as Chess, Shogi, and Go [1, 2]. The games mentioned are all perfect-information games, meaning all the game information is visible for all players. In recent times, the spotlight has shifted towards more complex games, including imperfect-information multiplayer games such as Texas Hold'em [3], Dota 2 [4], and StarCraft II [5], where some information is unavailable or hidden.

One such game is Japanese Mahjong, a multi-round tile-based game where up to four players race to complete their hands by acquiring specific combinations of tiles. The game utilises an imperfect-information system, prompting players to carefully evaluate which tiles to collect and discard to avoid giving the opponents an advantage. Due to the nature of the game, designing an AI for Japanese Mahjong is problematic for multiple reasons. First, the action space will change depending on the current board state, which dictates the set of available actions a player may choose from. Second, it is a multi-round game; each game consists of multiple rounds. Winning a single round does not ensure victory. Consequently, this leads to cases where players must consider whether folding is more beneficial than risk trying to win the round. Third, the average size of Mahjong's hidden information set is large compared to other imperfect-information games such as Texas Hold'em and Bridge [6].

Multiple AIs have been implemented for Japanese Mahjong with varying degrees of success. In 2020, Microsoft unveiled *Suphx* [7], the first Japanese Mahjong AI that managed to reach 10th dan, a rank that would place Suphx above 99.99% of Tenhou.net's human players. At the time of writing, Suphx is recognised as the strongest of its kind.

The performance of Suphx does not come without a cost. The architecture behind Suphx is both massive and complex. Additionally, the data structure proposed alongside Suphx is extensive, befitting the size of the AI's architecture. These characteristics make it infeasible to repeat the experiments with limited hardware. Other contemporary solutions suffer from similar complications.

This thesis seeks to find an alternative solution to the above problems without compromising too much performance. This solution is two-fold: First, a simpler model architecture is vital. Second, we must scale down the data structure to be used with the proposed architecture to a more accessible size.

Suphx and other similar implementations [8–11] have separate models for each of the possible actions within Japanese Mahjong. For example, an independent Pon model handles board states where the option to perform a Pon call is available for the player. Similarly, a Riichi model handles cases where the option to declare riichi is permitted. Suphx consists of five different

models: The Chi model, Pon model, Kan model, Riichi model and discard models[1].

An important note is that instead of implementing a fully working Japanese Mahjong AI agent, we will only concern ourselves with the action of tile discarding in the game. In other words, we propose an alternative model to Suphx's discard model. As a result, the final implementation cannot be used in an actual Japanese Mahjong match against human players as it cannot handle the complete set of possible actions.

1.1 Problem Statement

The aim for this thesis is to find a model architecture that can perform sufficiently when it comes to predicting the next discarded tile at Japanese Mahjong with a lower demand of hardware compared to contemporary solutions. Our main approach is the incorporation of an attention-mechanism in our proposed model architectures. Thus, the research question is:

RQ: *Can attention-based models learn to predict the next tile to be discarded in Japanese Mahjong at a level comparable to Suphx, the current state-of-the-art Mahjong AI?*

We are particularly interested in whether the models will learn to distinguish tiles in the observed player hand from tiles that are not, as failing to do so will lead to invalid predictions. Furthermore, as we are using attention-based models, we want to see which part of a given board state the models chooses to prioritise when predicting the next tile to discard.

1.1.1 Hypotheses

We will try to aim our experiments and discussion towards the following hypotheses:

Hypothesis 1: *An attention-based model will yield a higher prediction accuracy the non-attention-based version of the same model when presented with the same tile discarding task.*

We assume that incorporating an attention mechanism enables the model to learn to prioritise patterns found in the given data. If this is true, it follows that the model should achieve a higher prediction accuracy.

Hypothesis 2: *The attention-based models will yield a higher prediction accuracy when trained on a larger training set than a smaller one.*

For this hypothesis we go with the naive assumption that more training data is better than less.

Hypothesis 3: *The proposed models will yield higher accuracy on board states that is nearing its end than board states that are closer to the onset of the round.*

Throughout a Mahjong game, more and more tiles are revealed. In other words, games nearing their end should have more information available than the initial stages of the rounds.

Hypothesis 4: *The proposed attention-based models will predict less invalid classes than non-attention based model when presented with the same task.*

During the classification, our models have the option to predict any of the 34 classes. Due to the nature of Japanese Mahjong, some of these classes represent tiles that are impossible to discard in the given board state. We call such classes for invalid classes, and predicting fewer invalid classes is preferable. We assume that attention-based models will learn to avoid predicting invalid classes better than non-attention-based models.

[1] In the original paper [7], the Chi model, Pon model and Kan model are instead named Chow model, Pong model and Kong model, respectively. The Suphx paper uses the Chinese Mahjong terms for melds, whereas we choose to use the equivalent Japanese terms instead.

1.1.2 Scope

The following points defines the scope of this project:

- **Only Japanese Mahjong:** There exists multiple variants of Mahjong, such as Competition Mahjong, Hong Kong Mahjong, American Mahjong, and many others[2]. We focus solely on *Japanese Mahjong*.
- **Using the ruleset from Tenhou.net:** Within Japanese Mahjong, there exist a plethora of different rulesets[3]. For this thesis, we will use the most common ruleset played with in Tenhou.net, which will be explored in Chapter 4.
- **Only high-level game logs are used:** Tenhou.net publishes game logs of games played on their platform at regular intervals which can be retrieved from `http://tenhou.net/sc/raw/`. We only use a subset of the archived game logs from Tenhou.net, mainly games played by top ranked players. We detail the exact specifications on how the selection of game logs is performed in Chapter 3.
- **Solely focus on tile discarding:** In Japanese Mahjong, players can perform various actions, such as claiming tiles with Chi or Pon and declaring Riichi, depending on if the current board state allows for it. This thesis, however, will solely focus on tile discarding, one of the most integral parts of the game.
- **Only supervised learning is used:** When it comes to state-of-the-art Mahjong AIs, supervised learning seems to be the most popular approach when training the models [8–10, 12]. Suphx [7] used reinforcement learning in addition to supervised learning. For model assessment, some opted for online evaluation and tested them against real players on the Tenhou platform. [7–9, 12]. Our approach is more limited — the models used are trained solely with supervised learning and assessed with offline evaluation alone.

1.1.3 Limitations

Some limitations for this thesis should be noted:

- **Data Collection Method:** Papers on Mahjong AI often point out how many game states they use, and in some cases, also which year the game logs are from. However, seldom do they elaborate on inclusion/exclusion rules for these game states. This raises questions such as:
  - Do all included games use the same ruleset?
  - Are player rating taken into consideration?
  - Are games with disconnected players included?

  The absence of answers to the above questions makes it difficult to recreate identical datasets. Naturally, this will affect the final results and the integrity of the comparisons. We choose here to define custom inclusion/exclusion rules, which are elaborated in Chapter 3.
- **Lack of previous studies:** At the time of writing, the usage of multi-head attention layers in Japanese Mahjong AIs seems to be lacking. This thesis is one of the first explorations in this area. Naturally, having more similar contemporary attention-based models around would make it easier to compare architectures directly.
- **Hardware Limitations:** We perform all experiments using a single NVIDIA Tesla V100. Having access to better or more hardware would allow for more sizeable experiments. For our purposes, more GPU memory would allow for bigger datasets to be used during model fitting.

[3] A list of more variations can be found here: `https://en.wikipedia.org/wiki/Mahjong#Variations`.
[3] An exhaustive list over popular Mahjong rule sets can be found here: `https://ooyamaneko.net/mahjong/rratw/`.

1.2 Contributions

In this thesis, we show that adding an attention mechanism can improve the performance of a model when tasked to classify which tile to discard in Japanese Mahjong. Furthermore, we show that a more confined data structure can be used to achieve respectable results. Although our results fall short in comparison to those achieved by state-of-the-art models such as Suphx, the outcome is deemed satisfactory considering the limited size of the data structure used. Moreover, the thesis presents the following, more tangible contributions as well:

- **Processed Datasets:** The datasets used in the following chapters are published along the written thesis, as a means to motivate future works to continue in the steps where our work falls short. A thorough description of the dataset can be found in Chapter 3. The processed datasets can be retrieved from: `https://www.kaggle.com/trongdt/japanese-mahjong-board-states`.
- **Source Code and Models:** The source code used in the experiments can be found here: `https://github.com/TrongTheAlpaca/mahjong_project`. This includes checkpoints of the proposed models as well, of which can be found here `https://github.com/TrongTheAlpaca/mahjong_project/tree/main/model_checkpoints`. A snippet of the attention-based model source code can be found in Appendix C.
- **Experiments and results:** The reported results of the performed experiments serve as a benchmark for future works to compare with.

1.3 Overview

The thesis is structured as follows:

- **Chapter 2** establishes the theoretical framework needed for the subsequent chapters, including an introduction to Japanese Mahjong and the current state of Mahjong AIs.
- **Chapter 3** introduces the proposed models and explains how the experiments are carried out.
- **Chapter 4** details how the necessary data was gathered and how it will be used with the proposed models. Furthermore, specifications of the various datasets are presented.
- **Chapter 5** showcases the results from the experimentation with the proposed models. The results are discussed in the same chapter.
- **Chapter 6** revisits the objectives established in the initial chapter and presents ideas for future work.

Chapter 2

Background

2.1 Japanese Mahjong

Japanese Mahjong is a 4-player multi-round zero-sum game where players aim to collect tiles to build their hands [13], as seen in Figure 2.1. Each round, players race to form their hand into certain arrangements of tiles, where the fastest player is rewarded points depending on the arrangement. The player with the most points at the end of the game, wins the game.

Figure 2.1: People enjoying a game of Japanese Mahjong (Picture from Pixabay)

2.1.1 Tenhou.net

Tenhou.net is one of the most popular platform to play Japanese Mahjong on [13]. Tenhou.net periodically archives Japanese Mahjong game logs from games played on their platform, which one can retrieve from `http://tenhou.net/sc/raw/`. Although there exist many variants of Mahjong [6], this thesis will solely focus on the variant most commonly played on the *Tenhou* platform.

Figure 2.2 shows how the Mahjong is presented on the Tenhou platform.

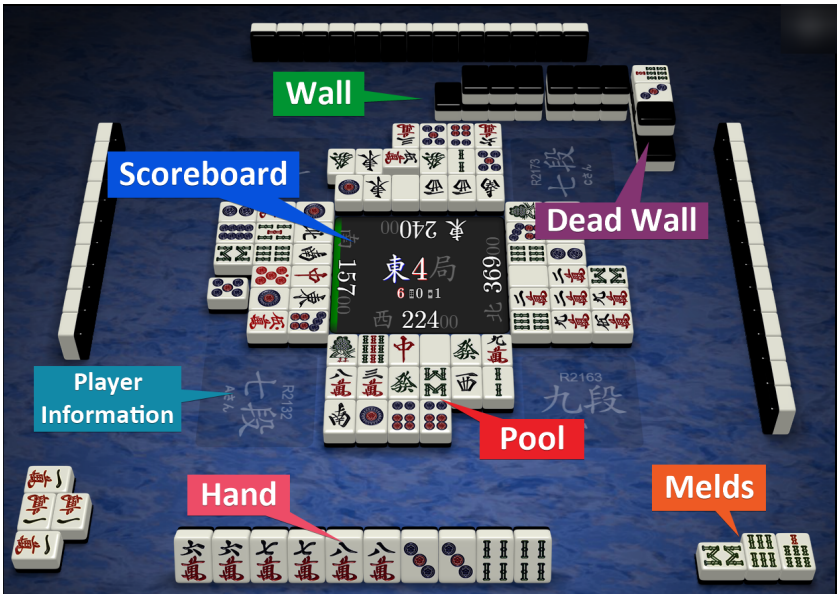


Figure 2.2: A Mahjong board as seen on the Tenhou platform, annotated. Note that the picture depicts a game in Tenhou's replay system which makes the players' rating visible unlike during play.

The black screen in the middle of the board is the *scoreboard*.

Figure 2.3: The scoreboard in the middle of the board.

Figure 2.3 shows the scoreboard in details with numbers pointing to the following stats:

1. **The current round wind:** In this example the current round wind is **East-4**. The number **4** here indicates that the current round wind is in its fourth rotation, which is also the last rotation for current round wind. If the game proceeds to the next rotation, the round wind will change into **South-1**.
2. **Various counters:** The left-most number (here: 9) indicates how many tiles there are left in the wall. The two next numbers display the number of honba sticks and number of riichi sticks, respectively.
3. **The current seat wind:** Each player has a seat wind corresponding to the seat position. In the current rotation depicted in the example: The POV player is **West**, the player to the right is **North**, the player across is **East**, and finally the player the left is **South**. Sitting with the **East** seat wind means that you are the current dealer.
4. **The player score:** Player scores are always rounded to nearest 100.

2.1.2 Core Gameplay

Initial Setup of Tiles

The Japanese variant plays with 136 tiles in total, consisting of 4 copies for each of the 34 tile classes. The following setup is performed at the start of each round: Each player begins with 13 tiles each. 70 of the 84 remaining tiles are reserved for the *live wall*, which are the tiles players draw from. The last 14 tiles are reserved for what is called the *dead wall*, which are tiles that will not see play for the current round.

2.1.3 Game Concepts

General Procedure

Each turn a player draws a tile and then discards a tile, where the dealer draws the first tile per round. Through selective discarding, a player can develop their hand into arrangements. Discarded tiles are placed in the discarding player's corresponding *pool* in front of them, for all players to see.

When a player's hand reaches a ready state, said player can declare victory on the next occurrence of the final tile. A player winning a round means cashing out the final hand into points, and the player is rewarded points according to the hand's arrangement. Generally, the rarer the arrangement is, the more points are rewarded.

Mahjong is a zero-sum game, which means that points won by one player is lost by the other players. It follows that when the total player gains are added up, and total player loss are subtracted, will sum to zero [14].

Tiles

The Japanese variant plays with 136 tiles in total, consisting of 4 copies for each of the 34 tile classes. The tile classes in Mahjong are separated into two categories:

- **Simples:** Tiles of suits, which comprise of *Pin* (Circles), *Sou* (Bamboos), and *Man* (Characters). Suit tiles are numbered from 1 to 9, as seen in Table 2.1. As there are four identical copies of each tile, there is a total of 108 simples.
- **Honour tiles:** Picture tiles, which comprise the dragon tiles and wind tiles as seen in Table 2.2 and Table 2.3, respectively. The dragon tiles include the tile classes *Chun* (Red dragon), *Hatsu* (Green dragon) and *Haku* (White dragon). The wind tiles comprise of the tile classes *East*, *South*, *West* and *North*. There are in total 12 dragon tiles and 16 wind tiles.

	1	2	3	4	5	6	7	8	9
Pin									
Sou									
Man									

Table 2.1: Simples

	Chun	Hatsu	Haku
Dragon			

Table 2.2: Dragon tiles

	East	South	West	North
Wind				

Table 2.3: Wind tiles

Melds

Melds are tile groups and are a central part of the game. There are three forms of melds in the game:

- **Sequence**: Three sequential numbered tiles, e.g. 4-5-6 Sou ()
- **Triplet**: Three of the same kind, e.g. 7-7-7 Pin ()
- **Quad**: Four of the same kind, e.g. 4x East tiles ()

Although *pairs*, two of the same kind, is an additional tile group important to the game, it is *not* regarded as a meld. The reason for this is because it is not a tile group you can form from *claiming* tiles.

Claiming tiles

In Mahjong, players have the ability to *claim* other player's discards. This action is only available at the time of discard. If no one claims the discard, the tile will forever be in the discard pool. Claiming other player's discards is an optional action. Claiming a tile requires the player to declare the action publicly. Furthermore, melds formed through claiming must be revealed to all players. There are four ways this can be done:

- **Chii**: **Player A** can only declare "Chii" if the player to the left, **Player B**, discards a tile that can be used to complete a sequence in **Player A**'s hand.
- **Pon**: Can be declared any time a player discards a tile that can be used to complete a *triplet* in **Player A**'s hand.
- **(Open) Kan**: Can be declared any time a player discards a tile that can be used to complete a *quad* in **Player A**'s hand.

Players are incentivised to claim tiles because of the following advantages:

- **Typically faster**: Claiming tiles from other player's discards to build melds typically leads to a ready state faster.
- **It skips the game to your turn**: If you claim a tile, the game skips to your turn regardless of who's turn it currently was. As claiming tiles adds an additional tile to your hand, you do not draw a new tile, but instead skips to the discarding action.

Although melds formed through tile claiming are put aside for all players to see, the melds are still regarded as part of your hand. However, a disadvantage is important

- **Expose strategy**: As you are obligated to reveal melds created through claimed tiles, you expose parts of your hand to your opponents.

- **Your hand becomes open**: A hand in Mahjong is either *closed* or *open* at any given time. Your hand start each round closed, and will remain closed until you claim a tile discarded by another player. Whether you have an open or closed hand plays a major role, as certain winning conditions become unavailable with an open hand. Additionally, winning with open hands is generally less rewarding.

Riichi

Declaring "riichi" is a yaku unique to closed hands. If your hand is closed and in tenpai, you can declare riichi on your next draw, which puts you in a state called *riichi*. If you manage to win while in this state, you are rewarded more points. However, this declaration means revealing to the other players that you are a single tile away from winning. Declaring riichi may prove to be detrimental as other players will be more cautious of which tiles they discard next. Furthermore, while in riichi, you cannot change the arrangement of your hand anymore: You must discard all subsequent tile draws if it is not the winning tile. Thus, declaring riichi is a high-risk but high reward wager.

Winning Conditions and Yakus

To win a game of Mahjong, you must end up as the player with the most points after the game concludes. The game concludes if one of the following conditions are met: a player reaches a negative score, or two wind rounds have passed.

Points are awarded each round to the player who completes his or her hand first. To achieve this, you need to develop your hand towards a "ready state", called *tenpai*. Being in tenpai means that your hand is one tile away from being complete. If you are in tenpai, you can complete your hand and declare victory by either drawing the last tile yourself or claim another player's if they discard it.

How many points you get when you declare victory depends on multiple criteria, such as which *yakus* your hand consists of. A yaku is either a particular arrangement of tiles or a special condition under which a victory is declared. An example of a yaku is forming a hand that consists entirely of pairs, or a hand made up of four triplets and a pair.

In some cases, you can combine multiple yakus, resulting in a more rewarding hand. We will not delve into how the scoring system works as it is somewhat complex and out of scope for this thesis. The rule of thumb is that the rarer or more demanding the arrangement of your hand is, the more points you are rewarded.

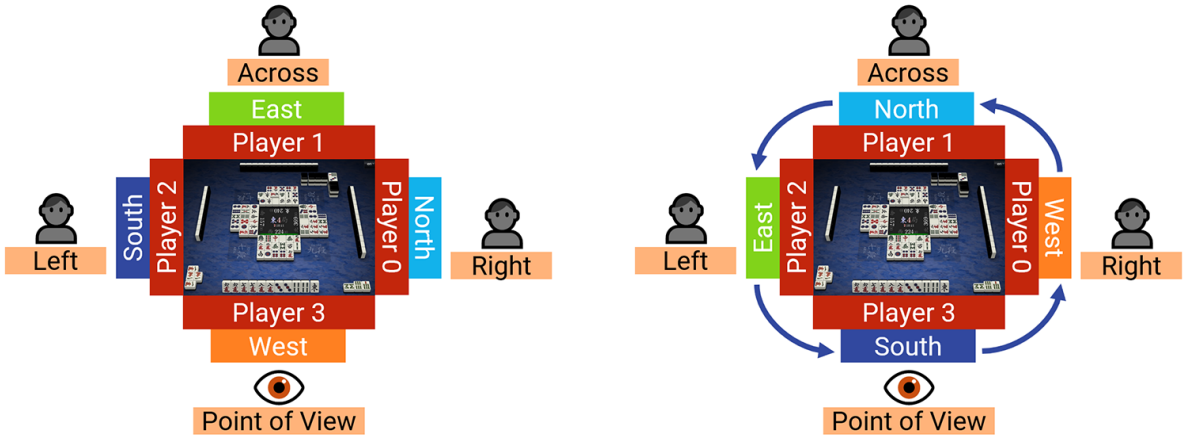
Wind Rounds

One important aspect of the game is the notion of *winds*. There are four different winds in the game: **East**, **South**, **West**, and **North**. Unlike the traditional cardinal directions, the position for East and West are swapped in Mahjong.

A game of Japanese Mahjong consists of two rounds, where each round consists of four seat rotations. Each round of the game is associated with a wind, called the *round wind*. The first round is always associated with **East**, and the second with **South**. A game can terminate before the **South**-round is played if a player runs out of points before that.

The game begins by assigning each player an initial *seat wind*. Throughout the game the seat winds will rotate anti-clockwise, as seen in Figure 2.4. Note that the order of seat winds remains the same.

At any point during the game, a round wind is always associated with the current round, and each player is always associated with a seat wind.



(a) First sub-figure

(b) Second sub-figure

Figure 2.4: Illustrates how seat winds rotates while the player positioning remain unchanged. The rotation happens anti-clockwise.

2.2 Inversions

Given a sequence of numbers, an inversion is a pair of positions that, if swapped, proceeds said sequence one step closer towards the sorted version of the sequence. Although the general definition of inversions allows for non-adjacent positions, we will restrict the definition to adjacent positions only.

Another way to understand this difference, is to associate it with number of swaps required using bubble sort versus selection sort. Counting number of inversions without any restriction of position adjacency, is the same as counting the number of necessary swaps when using selection sort. In Chapter 5 we count inversions in similar veins as counting number of necessary swaps when using bubble sort. The reason for this specification is because we want to calculate the distance away the array is from being sorted. By restricting it to adjacent positions, this distance become greater if values are further away from their target positions. Using the original definition will not increase the distance in such cases.

2.3 Machine Learning Concepts

2.3.1 Artificial Neural Network (ANN)

Simplified, an *Artificial Neural Network (ANN)* [15] is a configurable function that, if given some numerical values as input, processes the values and outputs processed numerical values. The main selling point of ANNs is that, if configured well, we can use them to solve complex problems if the problem can be stated as numerical values.

In general, ANNs consists of layers of *neurons*, as seen in Figure 2.5. The first layer is the *input layer*, whereas the last layer the *output layer*. The layers between the input layer and the output layer are called *hidden layers*.

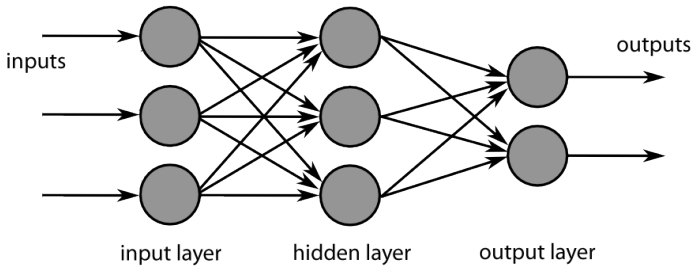


Figure 2.5: A Neural network with two layers. By Chrislb, CC BY-SA 3.0, via Wikimedia Commons.

A single neuron can be thought of as a variable that can hold a numerical value. Neurons are connected to the neurons in the subsequent layer through *weights*: numerical variables that we can configure. Furthermore, an additional neuron with value 1.0 called the *bias* resides in each layer, and is used to shift the output. A layer is *dense* if each of its neurons is connected with all neurons in the previous layer.

The input values given to the network are assigned to the neurons in the input layer. When a neuron i passes its value x_i to another neuron, the value x_i is multiplied by the corresponding weight w_i before the neuron in the subsequent layer receives the output. Thus, this becomes a weighed input $x_i w_i$ for the receiving neuron. Consequently, a receiving neuron's final value y is the sum of the set of weighted inputs and the bias weight w_0

$$y = w_0 + X^T \times W \tag{2.1}$$

where

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}, \quad W = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix}.$$

This value is then passed on as the previous layer. This process of passing the value forward is repeated until the values have passed all layers. The final layer, the output layer, outputs the final output of the network.

Changing the weights' value will directly affect the value passed through the network. To train an ANN is to optimise the weights such that the overall network outputs a more satisfactory output.

2.3.2 Convolution Neural Network (CNN)

A Convolutional Neural Network (CNN)[15, 16] is a specialised ANN that facilitates 2D input data using convolutions in lieu of matrix multiplication. Convolution is a specialised linear operation that translates the input data into a feature map by applying filters. Feature maps are 2D data structures that capture features found in the input data. By chaining up convolutional layers, the CNN can learn to capture more high-level features. In a face recognition task, the first convolutional layer may capture facial features such as eyes, mouth, etc., and generate feature maps from them. The next convolutional layer can use said feature maps to capture higher-level features such as face and facial structures.

These feature maps are then passed through pooling layers, which downsamples the feature maps into smaller sizes, making them more manageable for future steps. Then finally, the downsampled data is flattened by subsequent layers, transforming the 2D data into 1D vectors.

From there on, any traditional ANNs can be used to process the final vector. In simple terms, CNNs transform a 2D problem into a 1D problem. Thus, CNN remains a common architecture for problems such as image recognition and video recognition, where the input data are images.

2.4 Activation Functions

A neural network consists of neurons. A neuron's output is the sum of a set of inputs multiplied by said neuron's corresponding weights. The node's output is not bounded, meaning that it can be a value between −inf and inf. In machine learning, activation functions are used to squash a neuron's output within a fixed interval between -1 and 1.

2.4.1 Rectified Linear Unit (ReLU)

One commonly used activation function is the *ReLU (Rectified Linear Unit) activation function* [17], which is defined as the positive part of its argument

$$f(x) = \max(0, x) \tag{2.2}$$

2.4.2 Leaky ReLU

A known caveat with ReLU is that it can lead to the *Dying ReLU problem*, where neuron can be pushed into perpetually inactive state (0), becoming a inactive blockade for future inputs, effectively decreasing the model capacity.

Leaky ReLU [18] is an activation function that extends on the original ReLU, designed to combat the Dying ReLU problem. The difference being the assignment of a small positive slope for $x < 0$.

$$f(x) = \begin{cases} x & \text{if } x > 0, \\ 0.01x & \text{otherwise.} \end{cases} \tag{2.3}$$

The steepness of the slope can be configured, however, this thesis will use $0.01x$.

2.5 Softmax function

The softmax function σ [15] takes as input a vector z of K real numbers and normalises it into a probability distribution consisting of K probabilities that sum to 1,

$$\sigma(\overrightarrow{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2.4}$$

where $\sigma(\overrightarrow{z})_i$ is the i^{th} probability of normalised vector $\sigma(\overrightarrow{z})$.

2.5.1 Attention Mechanism

Attention [15] in machine learning is about mimicking how cognitive attention works: Focus on certain parts of the input data while ignoring less important details. Which part of the data is deemed necessary depends on the context. An *attention mechanism* is a component of the model that handles this calculation.

Multi-head attention layer

In 2017 Vaswani [19] proposed the *transformer* model architecture, an encoder-decoder structure relying entirely on *self-attention*. In short, self-attention is an approach where some data within the given input data is associated with another data input, forming the attention for said data.

One of the main components of the transformer is the multi-head self-attention mechanism. In short, this mechanism takes in input data and returns a matrix called the attention filter, a 2D matrix that accrues values based on their importance in the learned context. We use this filter on the original input data to determine which feature the model should prioritise or ignore.

2.6 State of the Art

An early implementation of a Mahjong AI is Mizukami's [12], which used Monte Carlo simulation to predict an opponent's movement using a probability distribution. This implementation used logistic regression to predict the opponent's wait and its potential value. One of the strongest aspects of this model is its defensive playstyle: If the current hand of tiles is far away from the tenpai, or an opponent declares riichi, the model shifts to a respectable defensive playstyle. However, a shortcoming with this model was that the model did not consider the current player score when deciding the playstyle to play. Nonetheless, the model managed to reach a prediction accuracy of 62.1% for choosing the correct tile to discard, given the board state.

In 2017, Tsuyoshi [20] proposed a model that used a CNN architecture to predict which tile to discard in Japanese Mahjong, which managed to reach a prediction accuracy of 53.98%. They summarised that the subpar accuracy was due to the limited data structure and that enlargement of the data structure would focus on future work.

Gao [8] proposed in 2018 a Mahjong AI using a CNN architecture as well. Unlike Tsuyoshi's approach, Gao's approach went with a more fleshed out data structure consisting of multiple planes, a plane being a 34 x 4 matrix. Although the data structure managed to include more of the board information, they left out some information such as the player score. All in all, their proposed model managed to reach a prediction accuracy of 68.8%. In 2019 [9], they improved their data structure to include more information of past actions, increasing the model's accuracy to 70.44%.

Honghai [11] went with a different approach than Gao's. Unlike Gao's solution, Honghai's discard model did not use a CNN architecture but instead a traditional feed-forward architecture consisting of dense layers. Alongside their discard model, they combined the usage of an additional network, namely the yaku prediction network, with their discard model. Their yaku prediction network learned to predict which yaku was reachable from the hand. Their discard model achieved a discard prediction accuracy of 64.88% without the yaku prediction assistance, whereas 66.81% when combined with the yaku network.

In 2020, Honghai [10] proposed a discard model using the CNN architecture of 30 residual blocks. Alongside their model, they presented an extensive data structure containing more elaborated information on yaku than previous solutions. Their model reached a final prediction accuracy of 72.3%.

In 2020, Microsoft revealed Suphx [7], the Mahjong AI that managed to topple the top human players on the Tenhou.net platform. It is regarded as the strongest Mahjong AI at the time of writing [6]. Suphx consists of five different models, each corresponding to a playable action within Japanese Mahjong. Their discard model is a deep CNN model with 50 residual blocks. Their discard model manages to reach a discard prediction accuracy of 76.7% through

both supervised and unsupervised learning.

Summary of State-of-the-art Models

Table 2.4 lists the state-of-the-art models and their results.

Published	Model	Source	Training Set	Validation Set	Testing Set	Accuracy
2020	Suphx	[7]	15,000,000	10,000	50,000	76.7%
2020	Honghai	[10]	18,000,000	26,000	-	72.3%
2019	Gao	[9]	1,921,798	213,533	50,000	70.4%
2019	Honghai (w/ yaku)	[11]	1,260,000	140,000	-	66.8%
2019	Honghai (w/o yaku)	[11]	1,260,000	140,000	-	64.9%
2018	Gao	[8]	540,000	60,000	30,000	68.8%
2017	Tsuyoshi	[20]	-	-	-	54.0%
2015	Mizukami	[12]	-	-	-	62.1%

Table 2.4: Summary of contemporary discard models and their corresponding results, sorted by achieved accuracy (rounded to the nearest tenths). A dash (-) indicates unpublished information. Note that it would be unwise to directly compare the results due to vastly different training and testing scheme.

For some of the aforementioned models the top-K accuracy achieved are disclosed. These results are listed in Table 2.5.

			Top-K Accuracy		
Published	Model	Source	Top 1	Top 2	Top 3
2018	Gao (2018)	[8]	68.8%	93.6%	96.5%
2015	Mizukami	[12]	62.1%	82.9%	90.9%

Table 2.5: Top-K accuracy from some of the models.

Chapter 3

Methodology

3.1 Research Overview

We have chosen to adopt a quantitative approach in attempt to answer the questions and hypotheses presented in Chapter 1, primarily by comparing the numbers gathered through experiments, and by comparing them with those of other related works.

Hypothesis 1 questions whether adding an attention layer to the architecture leads to a better performance than if omitted. To examine this hypothesis, we propose two model architectures: one non-attention-based and one attention-based, where the latter is the same as the former, but with an added attention mechanism. Performing the same experiments on both architectures should yield different results, which will provide the basis of our conclusion for Hypothesis 1.

Hypothesis 2 explores if there is a positive correlation between the size of the training set and the final prediction accuracy. To answer this, we will train the same model architecture against two training sets of different sizes, effectively producing two different models: MHA-B and MHA-S. Comparing the results from testing these models against the same testing set should answer Hypothesis 2, provided there are no other variables involved.

Hypothesis 3 states a correlation between our chosen models' prediction accuracy and the progression of a given round. We choose to define a round's progress by the number of discarded tiles visible on the board - more tiles revealed implies that a round is closer to completion. We define three stages of round progression: **Early**, **Mid** and **Late**. We will refer to these stages hereafter as **game phases**. To answer Hypothesis 3, we design three different testing sets, one for each of the three game phases, such that each set only contains game states for its corresponding game phase. We coin these test sets as *Phase Sets*, of which we will go into greater detail in section Section 4.4.3.

We note that there is an overlap between the proposed experiments for Hypothesis 1 and Hypothesis 4, the main difference being the specific data analysed. Instead of concentrating on the prediction accuracy, for Hypothesis 4 we will analyse the number of invalid discards predicted during the experiments.

3.2 Method Overview

We approach the tile discarding task as a 34-class classification problem, where each class correspond to each of the possible tile classes within Japanese Mahjong. The goal of the task is to predict which tile the observed player will discard given a board state.

Calling it a classification problem implies that there is a single correct answer. In the actual game, however, selecting which tile to discard is closer to a multi-label classification problem

since multiple tiles can be regarded as good discards. Nonetheless, we have chosen to define it as a multiclass classification problem as it is easier to evaluate.

3.2.1 Model Operation Overview

This section describes the procedure on how the models will be used with the given data. To keep the explanation in this section brief, we omit the following two concepts:

1. **The entire model architecture:** For this section, the inner workings of the model are generalised. We defer the full description of the proposed model architectures till Section 3.4.
2. **The full specification of the proposed data structure:** How a board state is encoded, and how the encoded data structure looks like, will be omitted for this section. The data structure will be revisited in Section 4.1.

We get board states by extracting it from game logs from Tenhou.net. Note that a single game log contains multiple board states. Assume that we have a method to encode a board state into a one-dimensional (1D) array of 374 integers. This process is illustrated in Figure 3.1.

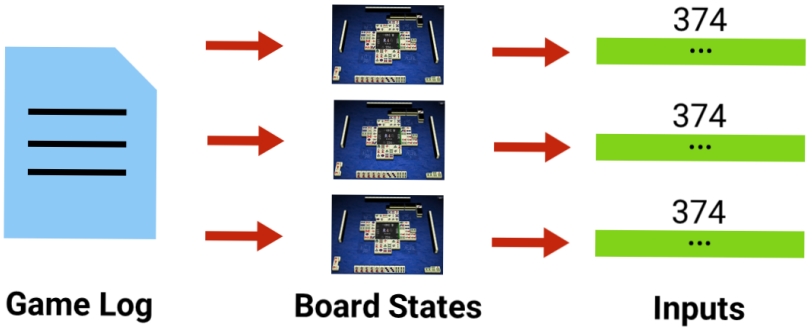


Figure 3.1: Illustrates the process from game logs to input data.

We use the encoded data as an input for the proposed model, which will output a new 1D array of 34 floats called *logits*. This interaction is depicted in Figure 3.2.

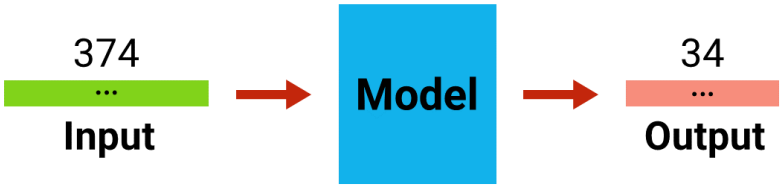


Figure 3.2: Illustrates the input size versus the output size.

The outputted logits can be mapped through a softmax function, which effectively translates the array of logits into a probability distribution. Figure 3.3 depicts this procedure.

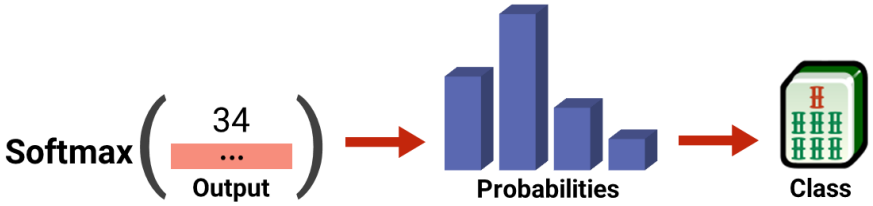


Figure 3.3: Outputted logits are mapped to probabilities which in turn becomes the model's predictions.

The final probabilities are ordered, where each probability correspond to one of the 34 tile classes. For instance, the first probability will always correspond to **1 man** (🀇), the second to **2 man** (🀈), and so on. Figure 3.4 shows the complete order of class tiles.

The value of each probability represents the amount of "confidence" the model allocates to the corresponding tile class. This particular interpretation will be used to determine which tile the model suggests to discard. The class with the highest probability will be the model's 1st priority discard, and the class with the second-highest probability will become the model's 2nd priority discard, and so on.

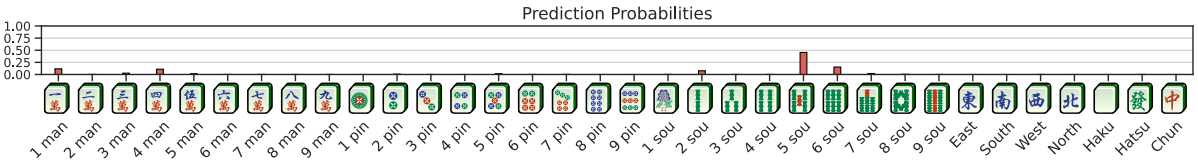


Figure 3.4: An example of a predicted probability distribution.

3.2.2 Advantages with 34 Classes

One might argue that the idea of having the output as 34 classes and not less. A reasonable suggestion is to reduce this problem to a 14-class classification problem instead, to match the maximum number of tiles a hand can hold. Doing this, each class corresponds to each tile in the current hand. We keep with 34 classes for mainly two reasons:

1. Previous research [8, 9] on this topic has come to the conclusion that discard models performed better when the problem was treated as a 34-classification problem rather than a 14-class problem.
2. We prioritise coherence and uniformity. The problem becomes more coherent when the uniformity of the output remains the same *regardless* of the input data: The position of elements always correspond to the same tile classes (i.e., the first classes being **1 man** (🀇), **2 man** (🀈), and so on), irrespective of the current hand. If the problem is converted into a 14-class classification problem, this is no longer the case.

3.3 Metrics

3.3.1 Prediction Accuracy

A model's prediction accuracy is how often it correctly predicts the tile the observed player discards in a given board state. We can calculate the model's prediction accuracy by calculating how often the model manages to predict the correct class or, conversely, how often the target class' index has the highest probability for all board states in the dataset.

3.3.2 Top-K Accuracy

When our models try to predict which tile to discard, an incorrect prediction does not necessarily mean it is a bad move. In practice, the wrongly predicted tile class may be the optimal choice given the current board state. Nonetheless, such predictions are regarded as *incorrect* since we define the problem as a *multiclass classification* problem.

We introduce *Top-K accuracy* which allows for some degree of mistake, as a way to alleviate this strictness. While the above prediction accuracy only concerns the class with the highest allocated probability, top-K accuracy considers the top K probability. It counts as an accurate prediction if the target tile class is predicted within the model's top K probabilities. In other words, the definition for prediction accuracy is equivalent to top-1 accuracy.

We will use top-K accuracy as an additional metric when evaluating the models' performance. The inclusion of top-K accuracy is directly inspired by [8].

3.3.3 Invalid Discards and Top-K Invalid Discards

Recall that a hand can only hold a maximum of 14 tiles at a time. Therefore, the highest number of *unique* tile classes in hand is also 14. Since the model is permitted to predict any of the 34 available tile classes, the set of tile classes a hand can possess is always a subset of the prediction space. This fact implies that the model can suggest discarding tiles not present in the current hand. We label such discards as *invalid* because they are impossible to perform in an actual Mahjong game and would hypothetically lead to an illegal board state.

During model evaluation, we will assess how often a model predicts a tile that would lead to an invalid discard. Note that this is another metric for accuracy: We observe how frequently the model is able to avoid invalid discards. This way, the definition for top-K accuracy will be applicable for invalid discards as well. To make the distinction clear on which accuracy we are talking about, we will hereafter use *top-K accuracy* for *prediction accuracy*, and *top-K invalid discards* for *invalid discards*.

3.4 Proposed Model Architectures

In Section 2.6 we briefly introduced state-of-the-art Mahjong AI models, where many of them incorporate a CNN architecture. As declared in Chapter 1, their architectures are big, something we believe is a direct consequence from using a CNN architecture in the context of Mahjong.

Thus, in order to avoid reproducing an oversized model architecture, we refrain from using a CNN architecture altogether. The following sections explore the two model architectures we propose. Later, we go into details about how we use these architectures to generate three different models[1].

3.4.1 Non-Attention-Based

Our non-attention-based architecture is a traditional multilayer perceptron with 7 hidden layers of descending size. A Leaky ReLU activation function is implemented between each hidden layer. As a result, the final architecture ends up with 12,718,242 trainable parameters. Figure 3.5 illustrates this architecture.

[1]Note that we separate the definition between *model architecture* and *model*. A model implements a specific model architecture and can therefore be used on data, whereas a model architecture is just a theoretical blueprint for a potential model. This discrepancy is significant as we will later introduce two different models that use the same architecture.

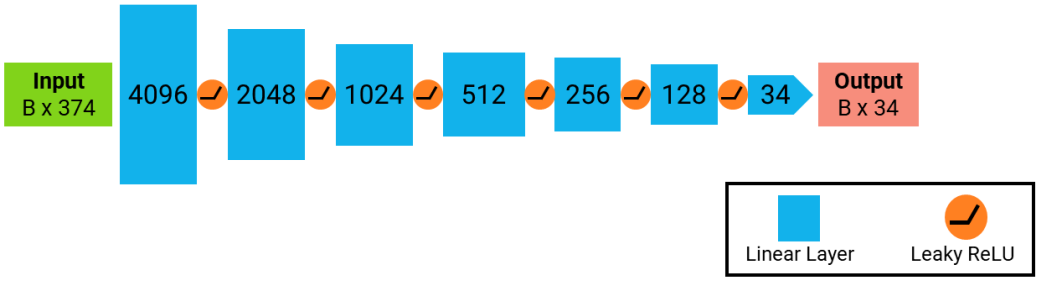


Figure 3.5: The non-attention-based model architecture. Dimension B is the batch size.

3.4.2 Attention-Based

Our attention-based architecture is the same as the non-attention-based one but with a multi-head attention layer right after the input layer, as illustrated in Figure 3.6. Despite its name, our multi-head attention layer is equipped with a single head rather than multiple[2]. Thus, the architecture has 13,279,242 trainable parameters, which is 561,000 more than its non-attention-based counterpart.

The multi-head attention layer is the same attention mechanism found in transformers, initially proposed by [19]. We choose to adopt this specific attention mechanism because the inception of the transformer is still recent, and yet it has already seen great success in multiple areas including Natural language processing [19, 21], image classification [22, 23], video classification [24] and image processing [25, 26]. As state-of-the-art Mahjong AIs seem to perform exceptionally well using CNNs, we seek to determine if transformers also can achieve success in Mahjong, similar to how transformers reach a similar performance as CNNs in some of the mentioned areas. With that said, our attention-based architecture is not a transformer; we are only adopting one of its main components.

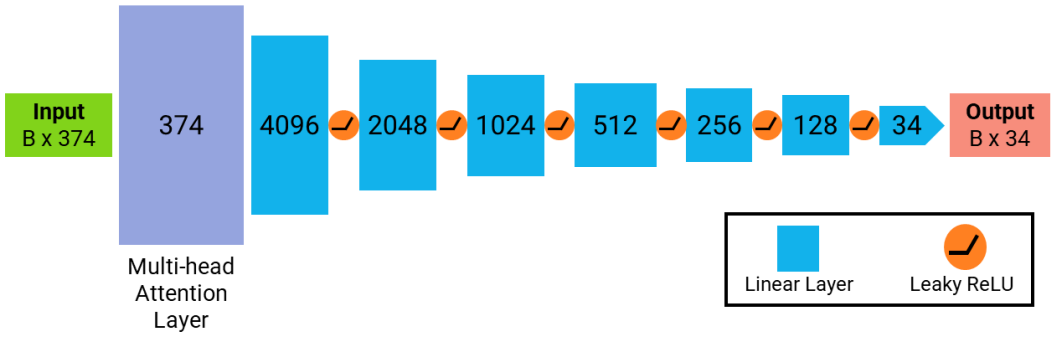


Figure 3.6: The attention-based model architecture. Dimension B is the batch size.

[2]We did experiment with 1, 11, and 34 heads during writing. All three experiments ended with the exact same results and trainable model weights after training and validation. We concluded this to be due to how our data is not composed of sequential data. We went with 1 head in the end due to it being the most simple architecture.

3.5 Proposed Models

3.5.1 The Feed-forward-only Model (FFO-B)

One of the proposed models is the feed-forward-only model (FFO-B), which implements the non-attention-based architecture from Section 3.4.1. Due to its simplicity, we do not expect any competitive results from it that will outperform state-of-the-art models'. We will mainly use the FFO-B to benchmark against other proposed models in the hope of finding meaningful support for Hypotheses 1 and 4.

3.5.2 The Multi-Head Attention Incorporated Models (MHA-B and MHA-S)

The remaining two proposed models, MHA-B and MHA-S, use the attention-based model architecture explained in Section 3.4.2. The difference between these models is that MHA-S is trained on a training set half the size of the one used for MHA-B. Both FFO and MHA-B train on the same training set. We will explore the actual specifications of these datasets in Chapter 4. The decision to introduce *two* attention-based models was made so their results could be compared, in order to either confirm or reject Hypothesis 2.

3.5.3 Optimiser and Loss function

Stochastic Gradient Descent (SGD) Optimiser

We use a SGD optimiser as our optimiser during model fitting. The SGD optimiser is configured with a fixed learning rate of 0.001 without any decay rate[3]. This differs from Honghai's [10] experiments, where they instead used a learning rate of 0.01 with a decay rate of 0.5.

Cross-Entropy Loss

As we are dealing with a multiclass classification problem, we use the cross-entropy loss function as our loss function, similar to [10].

[3]When we used a learning rate of 0.01 or higher the training would return `NaN` values, which in PyTorch indicates that the gradients are overflowing. Lowering the learning rate fixed this issue.

Chapter 4

Japanese Mahjong Dataset

4.1 Data Structure

Up to this point, we have focused more on model architecture rather than the data structure to be used alongside it. When we say data structure, we mean the encoded board state that the models can interpret and parse. Designing a data structure encompasses specifying feature selection and representation: Which features of a Mahjong board we want to include, and how we choose to encode said features.

Finding an efficient way to represent board states of Mahjong is important. Optimally, the data structure should be small but at the same time be able to represent all details without loss of information. Furthermore, the data should be uniform in size, a measure to reduce confusion during model training. Thus, the challenge here is to find a balance between how much data we want to represent without taking away too much of the information.

Our proposed data structure is an 1D array of 374 integer values, as seen in Figure 4.1. Each field in the data structure represent some information, something we go into details in Section 4.1.2.

	Round wind	Dealer	POV player	# honba sticks	# riichi sticks	# wall tiles	P0 score	P1 score	P2 score	P3 score	P0 riichi status	P1 riichi status	P2 riichi status	P3 riichi status	Padding	Padding	Padding	Padding	Padding	Padding	Padding	Padding	Padding	Padding	Padding	Padding	Padding	Padding	Padding	Padding	Padding	Padding	Padding	Padding
Metadata	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
Dora indicators	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67
POV hand	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101
P0 melds	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135
P1 melds	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169
P2 melds	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203
P3 melds	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237
P0 pool	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271
P1 pool	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305
P2 pool	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339
P3 pool	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373
	1 man	2 man	3 man	4 man	5 man	6 man	7 man	8 man	9 man	1 pin	2 pin	3 pin	4 pin	5 pin	6 pin	7 pin	8 pin	9 pin	1 sou	2 sou	3 sou	4 sou	5 sou	6 sou	7 sou	8 sou	9 sou	East	South	West	North	Haku	Hatsu	Chun

Figure 4.1: Note that the data is actually 1D array (1×374) but has been projected into a 2D format (11×374) for clearer demonstration.

4.1.1 Data Structure Characteristics

Our data structure has some characteristics that should be elaborated.

Player Identifier Number

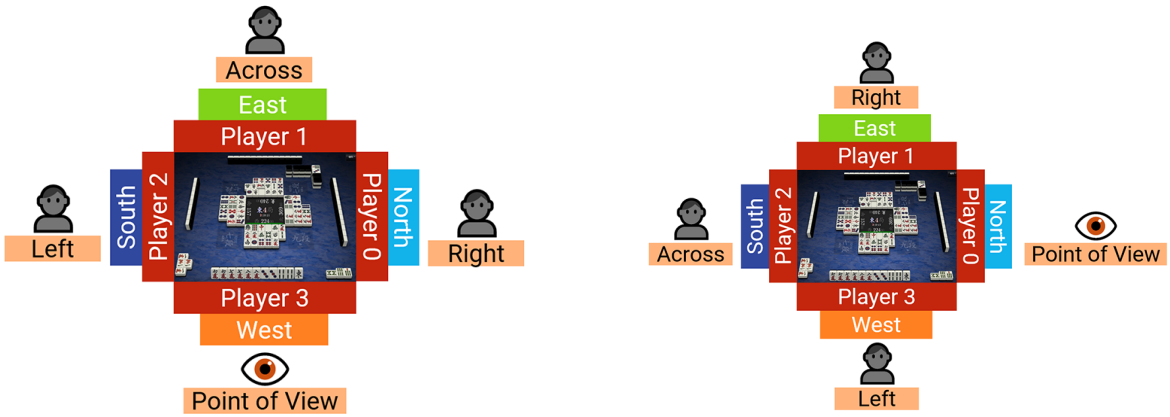
At the start of a game, each of the four players is assigned a unique identifier ranging from `0` to `3`. The player that begins in the East seat is always player `0`. The next player to their right is correspondingly assigned as player `1`, and so on in an anti-clockwise manner. In our data structure, the fields `Dealer` and `POV player` use this identifier value.

Relative Positioning

In our case, a board state is perceived from one of the four players' point-of-view (POV), namely, the *POV player*. The board state being seen from one specific player's POV has two significant implications: First, we can see the POV player's hand, whereas the other players' hands remain hidden. Second, as depicted in Figure 4.2, we can view the players' seat positions relative to the POV player's position:

- `P0` = POV player - yourself
- `P1` = Right player - the player to your right
- `P2` = Across player - the player across
- `P3` = Left player - player to your left

Note that the `P` stands for *Position*, and not *Player*. The former is about relative positioning, whereas the latter can be mistaken with player identifier number as elaborated in Section 4.1.1. In terms of our data structure, the encoded information is strategically arranged such that they are relative to the current POV player. Doing so ensures *uniformity*: the data arrangement is consistent regardless of the given board state. For example, the fields for `P3 pool` will always mean the discard pool of the player to our left, regardless of the given board state.



(a) Example 1 **(b)** Example 2

Figure 4.2: Player positions are always relative to POV player's position.

Wind Rotation

As there are four winds in the game, we assign the winds the following numerical values:

- `0` = East
- `1` = South
- `2` = West
- `3` = North

We use these values in the `Round wind` field in our data structure. A point worth noting is that both the seat wind and the round wind rotates anti-clockwise during play. If a player's current seat wind is `3` = North, the next seat rotation will make said player's seat wind come round again back to `0` = East.

4.1.2 Metadata and Tile data

The data structure is composed by two logical groupings: *Metadata* and *Tile data*, depicted in Figure 4.3.

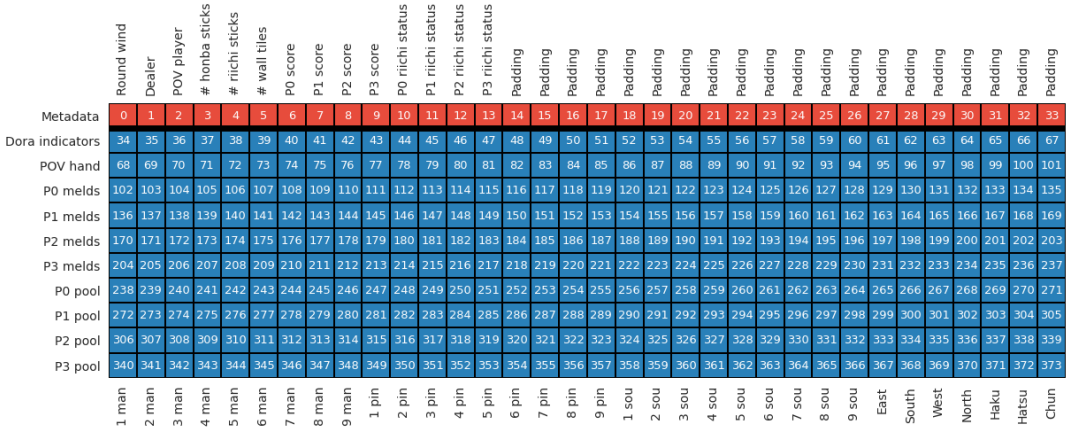


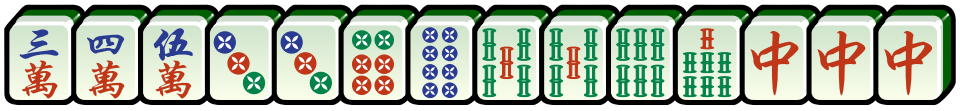
Figure 4.3: The two logical grouping within our data structure. Metadata is in red whereas tile data is in blue.

Metadata

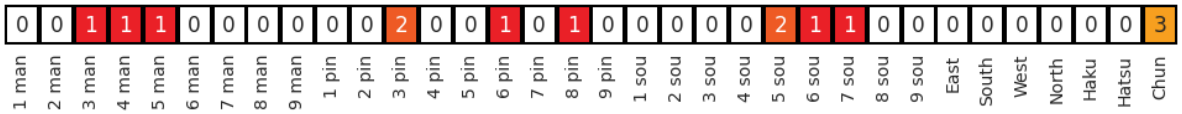
The first 34 values of the proposed data structure make up the metadata part. These values represent data *not* associated with any specific tile class. The explanation for the data is summarised in Table 4.1.

Tile Data

Each value in the Tile Data group is associated with one of the 34 tile classes, and its value represents the number of tiles of its corresponding tile class. Naturally, as there are four copies of each tile class in the game, the possible values ranges from `0` to `4`. An example is shown in Figure 4.4. Table 4.2 summarises the semantics behind the data values.



(a) Example hand



(b) Array notation

Figure 4.4: How tiles are transcribed into an array of 34 elements in the proposed data structure.

Name	Indices	Possible values	Description
Round Wind	0	$0, 1, 2, 3$	Represents the current round wind, which is either 0 (East), 1 (South), 2 (West), and 3 (North). As Mahjong games rarely go beyond the South-round, West- and North-rounds are seldom played.
Dealer	1	$0, 1, 2, 3$	Represents the current East player. Dealer is either 0, 1, 2, or 3, which represents either Player 0, Player 1, Player 2, or Player 3, respectively.
POV player	2	$0, 1, 2, 3$	The player the board state is observed from. This value is value is important as tile data is relative to the POV player value. Follows the same convention as Dealer.
# honba sticks	3	$[0, 127]$	Represents the number of Honba sticks on the board.
# riichi sticks	4	$[0, 127]$	Represents the number of Riichi sticks on the board.
# wall tiles	5	$[0, 69]$	Represents the number of tiles left in the live wall.
P0 - P3 score	6 - 9	$[-128, 127]$	Represent the player scores given by relative position. To make the player score fit within signed 8-bit integers, the player score are divided by 1000 and then rounded using banker's rounding.
P0 - P3 riichi status	10 - 13	$0, 1$	Represent if player at position 0-3 have declared riichi this round. As this is a boolean status, the value is either 0 or 1. An important detail here is that P0-riichi status is always 0 as the dataset does not include board states where POV player is in riichi.
Padding	14 - 33	-128	Unused values and is always set as -128, the lowest possible value for a signed 8-bit integer. The inclusion of padding is to make the data (1×374), which is directly compatible with (11×34) shape if needed.

Table 4.1: Metadata values and their description.

Name	Indices	Possible values	Description
Dora indicators	34-67	$0, 1, 2, 3, 4$	Represents the current visible dora indicators.
POV hand	68-101	$0, 1, 2, 3, 4$	Represents the tiles in the POV player's hand.
P0 - P3 melds	102-237	$0, 1, 2, 3, 4$	Represents the players' current open melds relative to POV player.
P0 - P3 pool	238-373	$0, 1, 2, 3, 4$	Represents the players' pool relative to POV player.

Table 4.2: Tile data values and their description.

4.1.3 Data Structure Example

Figure 4.5 shows a board state from an actual game. This board state can be transcribed into the proposed data structure as shown in Figure 4.6.

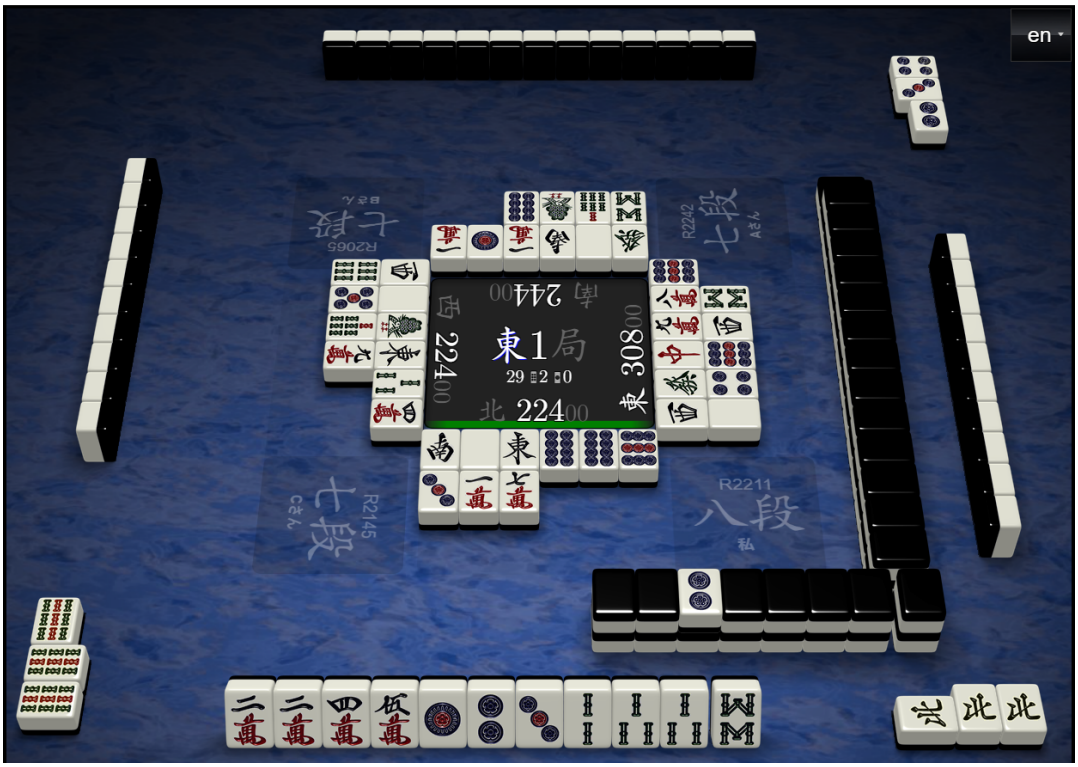


Figure 4.5: How a game of Mahjong is displayed by the replay system[1] in Tenhou.net. This specific board state is from the game `2018052710gm-00a9-0000-863640fd`.

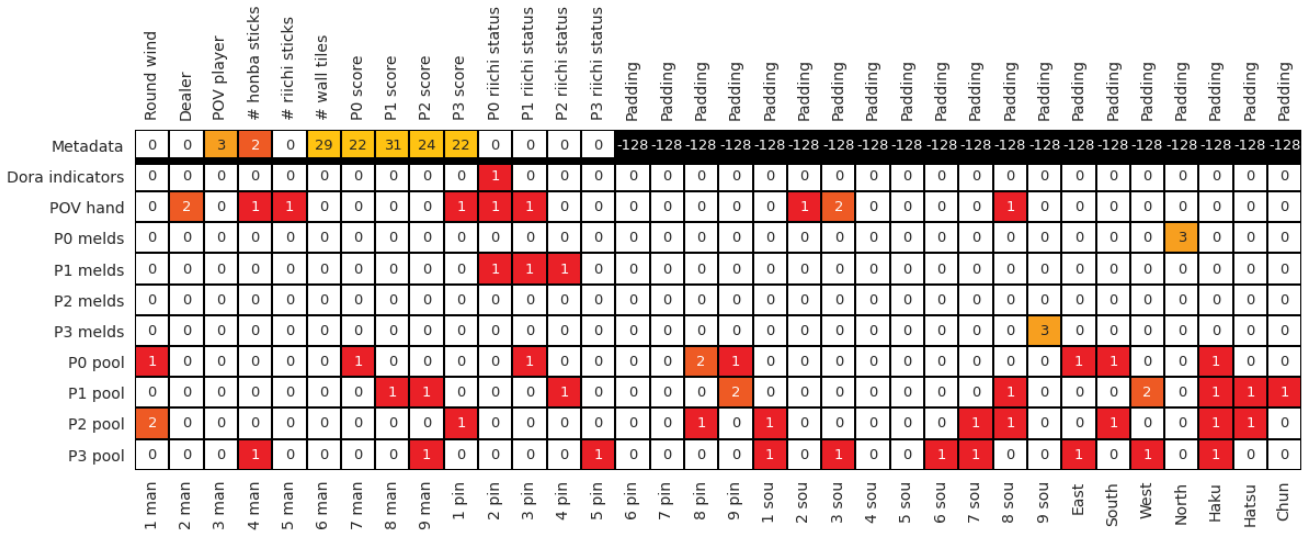


Figure 4.6: The array representation of Figure 4.5.

4.1.4 Data Structures in other works

In Chapter 1, we highlight that a recurring problem found in other works is the extensive size of their proposed data structures. This section summarises some of them briefly.

As convolutional layers are designed for two-dimensional input, Mahjong AIs that use a CNN architecture opt for a two-dimensional data structure. Their data structures consist entirely of *channels*: 1D arrays of size 34, where the size 34 corresponds to the number of tile classes in Japanese Mahjong. Despite using similar compositions, the usage of the channels varies a lot between the various papers. Some examples of variations are as follows:

- The paper behind Suphx [7] proposes a 3D data structure of 838 channels to encode a single board state. Features such as the observed player's hand are encoded using four channels. Categorical features, such as dealer state, are encoded as multiple channels where each channel being either all 0's or all 1's. Integer features, such as the number of tiles left in the live wall, are encoded in a similar fashion as categorical features. Additionally, they also encode look-ahead features such as potential scores of the current hand. Their data structure consists of $34 \times 1 \times 838 = 28,492$ values.
- Gao [8] proposes a smaller but similar data structure as Suphx's. Their data structure consists of multiple *planes*, where a plane is the same as four channels, or a 4x34 matrix. Their data structure consists of 55 planes, totalling $34 \times 4 \times 55 = 7,480$ values. Gao's data structure contains information from the four preceding steps, including the player's hand before the current state and recently discarded tiles. Exactly 40 out of 55 planes are used for the encoding of said past information.
- Gao [9] extends the one used in [8] with 31 more planes, totalling $34 \times 4 \times 86 = 11,696$ values. Some of the new additions include extending the past information to six steps back instead of four. They also include information on red five tiles, called *Akadora* tiles.
- Honghai [11] proposes the most extensive data structure of the presented papers, which contains 273 planes, or a total of $34 \times 4 \times 273 = 37,128$ values. The additional planes are used to encode the entire history of discarded tiles. Furthermore, 89 planes are used to encode information about present melds found in hand (i.e., sequences, triplets and quads).

[1]Tenhou.net's replay system can be accessed at `https://tenhou.net/4/?log=` together with a valid Tenhou.net game id, e.g. `https://tenhou.net/4/?log=2018052710gm-00a9-0000-863640fd` for the game shown in Figure 4.5.

Finally, 65 planes are used to encode information on the player hand's compatibility with existing yakus - possible winning hand forms.

- Honghai [10] proposes two data structures as they have two different discard models. The data structures are 1D arrays of sizes 472 and 481. Out of the data structures presented, these are the most similar to our own. The main difference between the two is that the latter encodes data on yaku as well. Their data structure differs from ours in that they focus more on melds and yaku compatibility. A total of 256 data values are used to encode information about melds for each player.

Table 4.3 summarises the sizes of various data structures.

Year	Model	Source	dim	Total number of data values
2020	Honghai	[10]	3D	$34 \times 4 \times 273 = 37{,}128$
2020	Suphx	[7]	3D	$34 \times 1 \times 838 = 28{,}492$
2019	Gao	[9]	3D	$34 \times 4 \times 86 = 11{,}696$
2018	Gao	[8]	3D	$34 \times 4 \times 55 = 7{,}480$
2019	Honghai (w/ Yaku)	[11]	1D	481
2019	Honghai (w/o Yaku)	[11]	1D	472
2021	Ours	Ours	1D	374

Table 4.3: Comparisons of different data structure sizes from other works and ours. Sorted by total number of data values.

4.1.5 Advantages and disadvantages with the proposed data structure

Advantages

Our data structure is the smallest of the presented ones regarding the total number of data values. Twenty of the fields are used as padding and do not represent anything significant. This makes our proposed data structure effectively a 1D array of 354 values rather than 374 values. A smaller data structure implies fewer data values for the models to learn, resulting in faster convergence.

Another advantage is that the design of our data structure is flexible in two ways. First, one can replace the padding fields with meaningful values without sacrificing existing data. Second, one can transpose it into an 11×34 matrix without adjusting the previous composition. We hope that this makes it more appealing for future work to extend our data structure for CNNs.

Disadvantages

One major disadvantage with our data structure is that the order of tiles is not present. The most notable areas affected are the order of discarded tiles and the order of declared melds. Furthermore, this makes it impossible to discern which tile the POV player just drew. Abandoning tile order is a compromise to keep the data structure as minimal as possible and is one of the main driving force to how we managed to make it notably small.

Although the data structure identifies which players have declared riichi, we do not disclose when said players made the declaration. This is yet another significant compromise to shorten down the data structure. A potential modification to accommodate this is replacing the existing riichi status fields with four new data fields. When the players declare riichi, we can encode the number of tiles left in the live wall in the corresponding data field. If a player has not declared riichi yet, the related field can be set to some default value, e.g. -1.

Although we encode tiles found in open melds, we do not signify which sort of melds they are, like [10]. How omitting such information affects model performance is left as future work.

Unlike [9], our data structure does not include information on red fives. Red fives are valuable tiles as they yield bonus points in Mahjong. It is conceivable that possessing a red five in hand will affect the selection of the tile to discard for most players.

4.2 Data Retrieval and Processing

Tenhou.net periodically archives Japanese Mahjong game logs from games played on their platform, which one can retrieve from `http://tenhou.net/sc/raw/`. Their game log archive contains logs from games played from 2009 to the present time. The following sections go into details on how we filter and process the data into our usage. The final version of the game logs can be retrieved here: `https://www.kaggle.com/trongdt/japanese-mahjong-board-states`. That said, we only use a subset of the processed data in our experiments, of which we go into further details in Section 4.4.

4.2.1 Tenhou.net Game Log

Each game log from Tenhou.net corresponds to a single game played on their platform. Each game is assigned a unique identifier. A game log contains a history of all actions performed by all players throughout a single game in a serial manner.

4.2.2 Game Log Filtering

To assure consistency in our experiments, we define very specific inclusion rules for game logs.

Ruleset Inclusion Rules

Tenhou.net supports multiple variations of Japanese Mahjong, e.g. playing the 3-player format instead of the 4-player format. We only include game logs from games played with the following rules:

- Uses the 4-player format
- Include *akadora* tiles, or red fives
- Allows for open tanyao
- Fast mode is disabled
- Full game is enabled, i.e. both East and South rounds are played

We use the above collection of rules as it comprises the most commonly played ruleset, as summarised in Table 4.4.

Include akadora	Open tanyao allowed	Fast mode	East+South	3-player mode	% of games
True	True	False	True	False	55.635
True	True	True	False	False	22.320
True	True	False	True	True	21.557
True	True	False	False	True	0.196
False	True	False	True	False	0.086
True	True	True	True	False	0.075
True	True	False	False	False	0.064
True	True	True	True	True	0.037
True	True	True	False	True	0.018
False	False	False	False	False	0.009
False	True	False	False	False	0.002
False	False	False	True	False	0.001

Table 4.4: The occurrence of different rulesets in the dataset of 2,506,325 unique games. Each row represents an available ruleset players can play with in Tenhou.net. Sorted by percentage of games played with corresponding ruleset.

Player Rating Inclusion Rules

Tenhou.net has multiple "lobby rooms" where players can play in if they meet the player rank and rating requirements for the room. Tenhou.net's highest level room is the *Phoenix Room*, which can only be accessed by players that meet the following requirements:

- has a minimum rank of 7-Dan
- has a minimum R-rating of 2000.0

To assure that we only use game logs from high-level games, we select game logs exclusively from games played in the Phoenix Room.

Avoid POV player in riichi

Once a player has declared riichi, the player locks their current hand. The player is forced to discard the most recent drawn tile. In other words, the player loses the right to decide which tile to discard. As we want our models to learn to predict the next discard, including board states where the POV player is in riichi is therefore harmful. This is why we exclude game states where the POV player is in riichi. This exclusion rule is directly inspired by [9].

4.3 Dataset Generation

How we collect board states for a dataset differs depending on if the set is for training or evaluation: For training sets, the board states can stem from the same game. For validation and testing sets, however, no two board states are ever from the same game. We hope that restricting the possibility for game states to originate from the same game eliminates the potential correlation that may occur if game states share an origin. Removing such correlation is essential as it may affect the results, something we want to avoid during evaluation. This method is directly inspired by [9]. Conversely, we do not apply the same restriction for the training sets, as such correlations are not necessarily harmful when fitting the models.

4.4 Proposed Datasets

For this thesis, we propose multiple datasets. All of our datasets have in common the following properties:

- They are balanced, meaning each dataset contains the same number of game states for 34 tile classes.
- The dataset sizes are all multiples of 34.
- They only contain game states from a particular year. We do this for two reasons: Firstly, to assure that no game states are shared across their intended usage, i.e., for training, validation and testing. Secondly, to make it easier for future work to generate similar datasets if need be.

Table 4.5 summarises all the datasets that will be used in the experiments.

				Number of cases per game phase		
Name	**Set**	**Year**	**Total Size**	**Early**	**Mid**	**Late**
Big	Training	2016	34,000,000	17,239,627	12,795,128	3,965,245
Small	Training	2016	17,000,000	8,620,136	6,395,584	1,984,280
Validation	Validation	2018	34,000	17,462	12,751	3,787
Default	Testing	2019	68,000	34,935	25,499	7,566
Phase 0	Testing	2015	34,000	34,000	0	0
Phase 1	Testing	2015	34,000	0	34,000	0
Phase 2	Testing	2015	34,000	0	0	34,000

Table 4.5: The datasets we will use in our experiments.

4.4.1 Training Set

We have two training sets:

- **Big:** a dataset of 34,000,000 game states from games played in 2016.
- **Small:** Same as Big, but half the size (i.e., a total of 17,000,000 game states)

For our experiments, FFO-B and MHA-B are trained on **Big**, whereas MHA-S on **Small**.

4.4.2 Validation Set

We have a single validation set that will be used for all model validation. This validation set contains 34,000 game states from games played in 2018.

4.4.3 Testing Sets

We have two subcategories of testing set: *Default* and *Phase Sets* — the former consists of a single testing set containing 68,000 game states from games played in 2019. The latter subcategory contains three different testing sets: **Early**, **Mid** and **Late**. Each phase set contains 34,000 game state from games played in 2015, but differ in that they exclusively contains only certain game states, defined as follows:

- **Early:** A testing set that contains only game states with a total of 24 or fewer tiles in discard pools.
- **Mid:** A testing set that contains only game states with a total of 25 to 48 tiles in discard pools.
- **Late:** A testing set that contains only game states with a total of 49 or more tiles in discard pools.

Note that the definition of phases is not an official term within Mahjong. They are defined in this thesis for the sake of explanation. The total tiles for each phase correspond with the number of rows in each discard pool, given that no melds have been created (an entire discard pool row is six tiles long).

4.5 Valid Class Distribution

Recall that a player's hand can hold a maximum number of 14 unique tile classes. Conversely, the minimum number is 1 (e.g. a hand that have dwindled to a single pair of the same tile class). For our experiments, it is vital to know the number of unique tiles in the current hand, as these make up the tile classes that the model can discard. Hence, we call these tiles *valid classes*. In contrast, we have *invalid classes*: Tile classes that are not present in the current hand. Table 4.6 summarises the total number of valid classes for each of the datasets.

Number of valid class	Big	Small	Validation	Default Test	Early Phase	Mid Phase	Late Phase
1	47	28	0	0	0	0	1
2	7,845	3,911	10	18	6	11	34
3	114,933	56,912	127	219	32	167	270
4	318,664	159,024	320	621	93	479	770
5	463,980	231,045	472	935	194	654	959
6	970,986	485,642	952	1,830	497	1,361	1,846
7	1,580,104	790,376	1,483	2,943	956	2,091	2,959
8	2,173,605	1,086,816	2,061	4,112	1,550	2,713	3,671
9	3,855,827	1,927,665	3,782	7,512	3,258	4,545	5,337
10	5,590,652	2,795,036	5,450	11,247	5,336	5,799	6,361
11	7,211,039	3,604,982	7,396	14,609	7,940	6,829	5,455
12	7,192,940	3,595,987	7,342	14,631	8,450	6,162	4,263
13	3,808,192	1,904,903	3,875	7,887	4,711	2,849	1,873
14	711,186	357,673	730	1,436	977	340	201
Total	34,000,000	17,000,000	34,000	68,000	34,000	34,000	34,000

Table 4.6: Number of cases per number of valid class for each dataset.

Chapter 5

Results and Discussion

5.1 Example Board State Prediction

This section will explore how our proposed models perform on an arbitrary board state.

Figure 5.1: An example board state. Watch the game here: Tenhou.net Replay Link.

Situation

Figure 5.1 shows a board state where the POV player is in a perilous situation. The left and right player are both in riichi state, meaning they are both one tile away from winning the round. This is marked by the side-way placed tile in the pool. Furthermore, no players have formed open melds, making the current board difficult to read. In this particular game, the POV player

decides to discard the **7 sou** (🀖), which passes without any problem. This action is followed by the right player drawing the **7 man** (🀖), which coincidentally is the single tile the player was waiting for. The right player declares victory and wins the round.

Analysis

Discarding the **7 sou** (🀖) is a well-founded move in this particular example as it is safe from all players due to the rules of *Furiten*:

- It is illegal for a player to call victory on a tile if said player has it in its discard pool. In other words, the Across player cannot claim victory on other players' **7 sou** (🀖) if they discard it as he/she has already discarded it beforehand.
- If a player is in riichi state and does not claim the winning tile when another player discards it, the player in riichi is no longer allowed to claim said tile if it appears later. This rule is applicable here as both the left player and the right player let the **7 sou** (🀖) pass when the across player discarded it, and thus can no longer claim victory on it if the POV player discards it.

Model Predictions

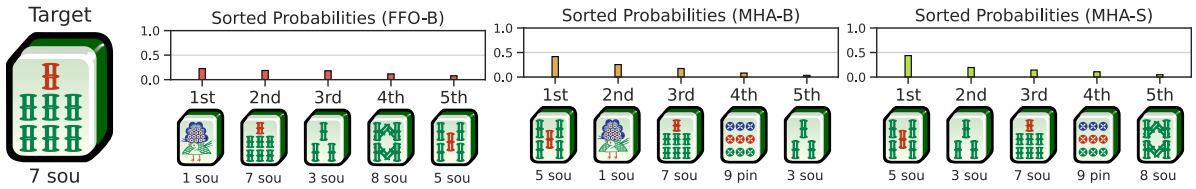
In this example, all proposed models fail to predict **7 sou** (🀖) as their first priority, as seen in Figures 5.3 and 5.4. In terms of predicting the actual target, FFO-B trumps both MHA-B and MHA-S by a small margin when comparing the allocated probability values.

Recall that the order of the tiles is not recorded in the proposed data structure. A consequence of this is that one cannot be certain that the **7 sou** (🀖) is safe to discard in this particular example, as the data does not show that it was discarded AFTER both the left and the right player declared riichi. This can be seen in Figure 5.2.

	Round wind	Dealer	POV player	# honba sticks	# riichi sticks	# wall tiles	P0 score	P1 score	P2 score	P3 score	P0 riichi status	P1 riichi status	P2 riichi status	P3 riichi status	Padding	Padding	Padding	Padding	Padding	Padding	Padding	Padding	Padding	Padding	Padding	Padding	Padding	Padding	Padding	Padding	Padding	Padding	Padding	Padding	Padding
Metadata	0	3	0	0	0	24	25	35	16	24	0	1	0	1	-128	-128	-128	-128	-128	-128	-128	-128	-128	-128	-128	-128	-128	-128	-128	-128	-128	-128	-128	-128	-128
Dora indicators	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
POV hand	0	0	2	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	2	0	1	0	2	0	1	2	0	0	0	0	0	0	0	0	0
P0 melds	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P1 melds	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P2 melds	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P3 melds	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P0 pool	2	1	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	2	0	0	0	2	0	0	0	
P1 pool	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	2	0	0	1	1	0	0	0	1	1	0	
P2 pool	0	0	0	0	0	0	0	0	2	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	1	2	0	1	1	0	
P3 pool	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	2	2	0	1	1	0	0	
	1 man	2 man	3 man	4 man	5 man	6 man	7 man	8 man	9 man	1 pin	2 pin	3 pin	4 pin	5 pin	6 pin	7 pin	8 pin	9 pin	1 sou	2 sou	3 sou	4 sou	5 sou	6 sou	7 sou	8 sou	9 sou	East	South	West	North	Haku	Hatsu	Chun	

Figure 5.2: The board state in Figure 5.1 when converted into the proposed data structure.

Interestingly, both attention-based models predict the **5 sou** (🀔) as their top priority. The "fives", that is, the **5 man** (🀋), **5 pin** (🀝), and **5 sou** (🀔), are regarded as valuable tiles in Japanese Mahjong as they are the most efficient tiles to form sequences [13]. In turn, this makes the "fives" dangerous to discard as they are more likely to befit other players' hand. Furthermore, discarding the **5 sou** (🀔) breaks the current pair the POV player has in the hand, which can be detrimental for the POV player.



(a) Target **(b)** FFO-B **(c)** MHA-B **(d)** MHA-S

Figure 5.3: Target tile and the corresponding predictions per model for the board state seen in Figure 5.1.

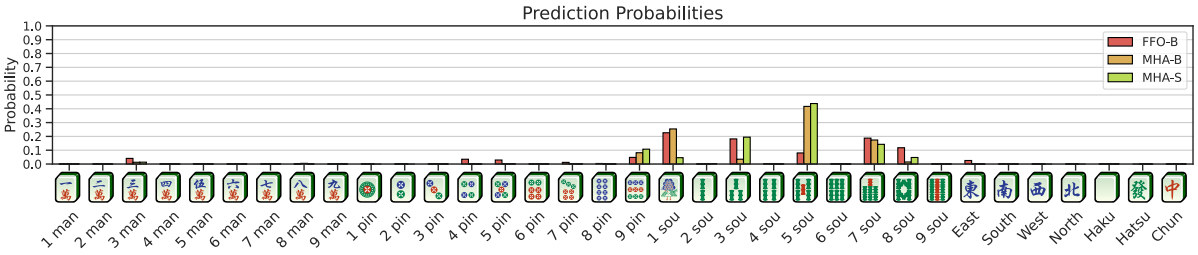


Figure 5.4: The prediction probabilities allocated by proposed models for all tiles.

The attention matrices created by MHA-B and MHA-S are shown in Figure 5.5 and Figure 5.6, respectively. Although the generated attention matrix changes in correspondence with the given input data, the general distribution of attention values remains unchanged.

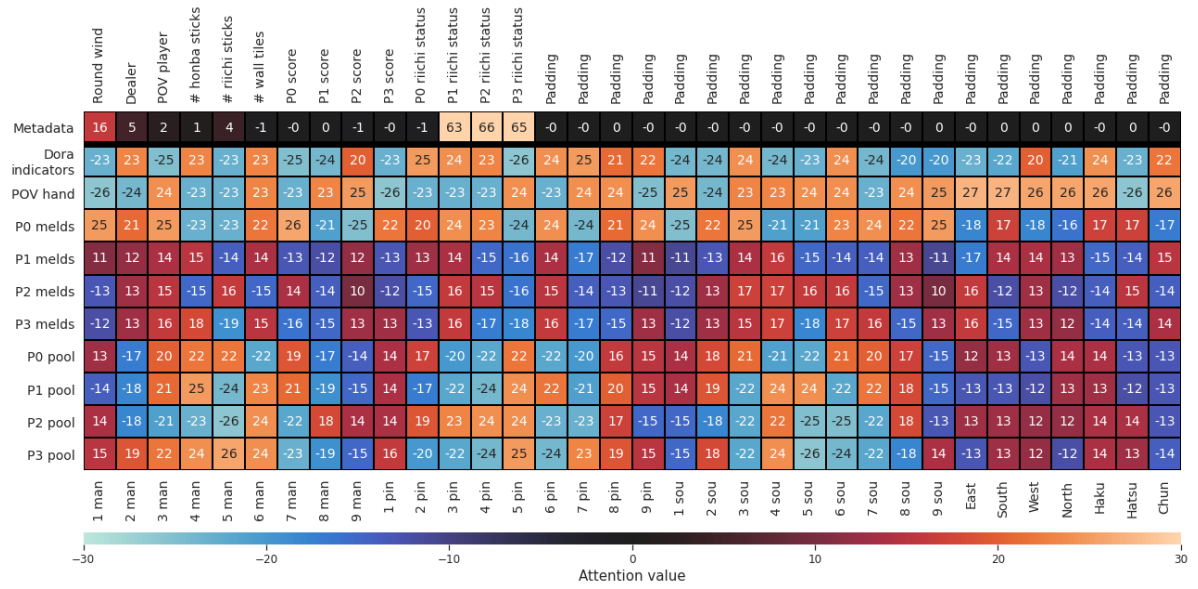


Figure 5.5: The attention matrix generated by MHA-B when fed the data seen in Figure 5.2.

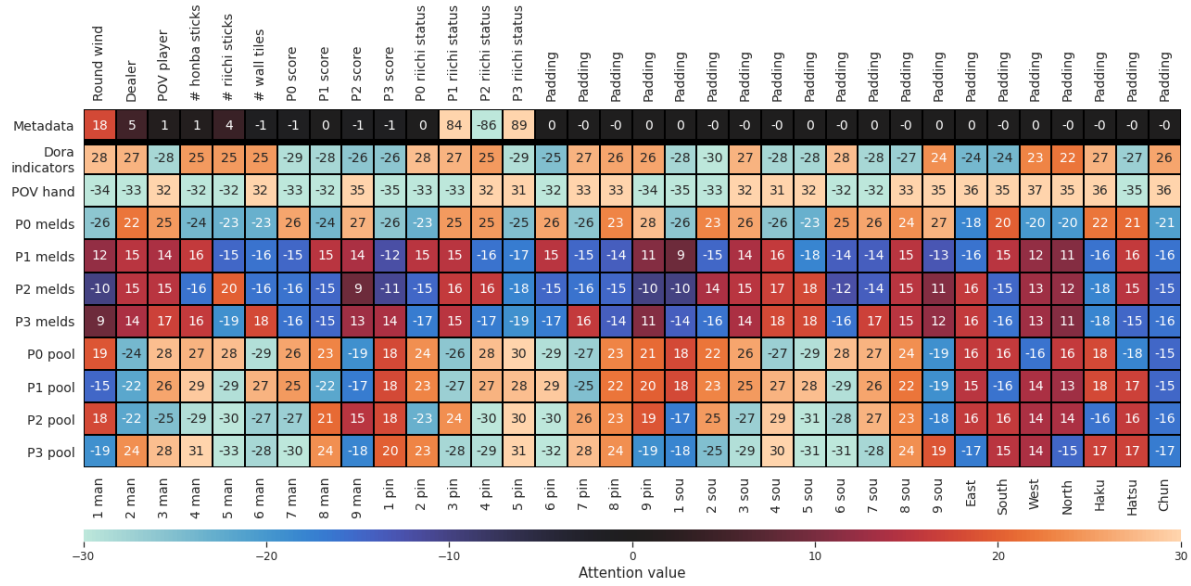


Figure 5.6: The attention matrix generated by MHA-S when fed the data seen in Figure 5.2.

5.2 Model Training and Validation Results

This section summarises the accuracy and loss achieved during training and validation for each of the proposed models.

5.2.1 Training

The proposed models are trained for 25 epochs each. The results are shown in Figures 5.7 and 5.8.

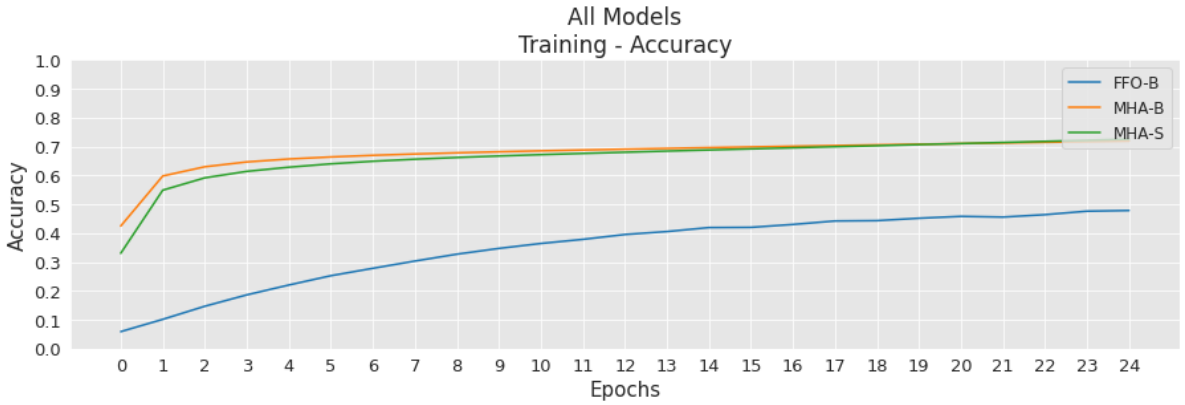


Figure 5.7: The accuracy achieved by proposed models during training.

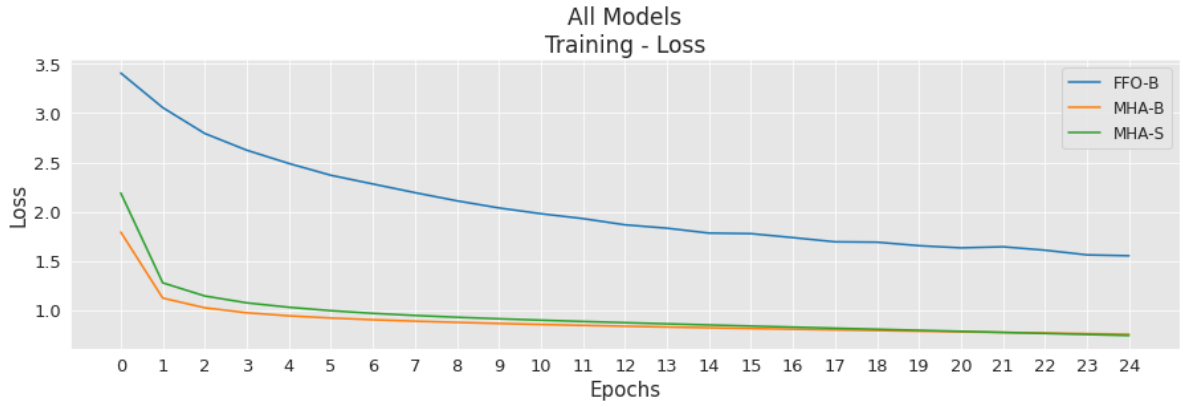


Figure 5.8: The loss values by proposed models during training.

5.2.2 Validation

The proposed models are validated after each training epoch using the validation set. The results from each validation epoch are shown in Figures 5.9 and 5.10.

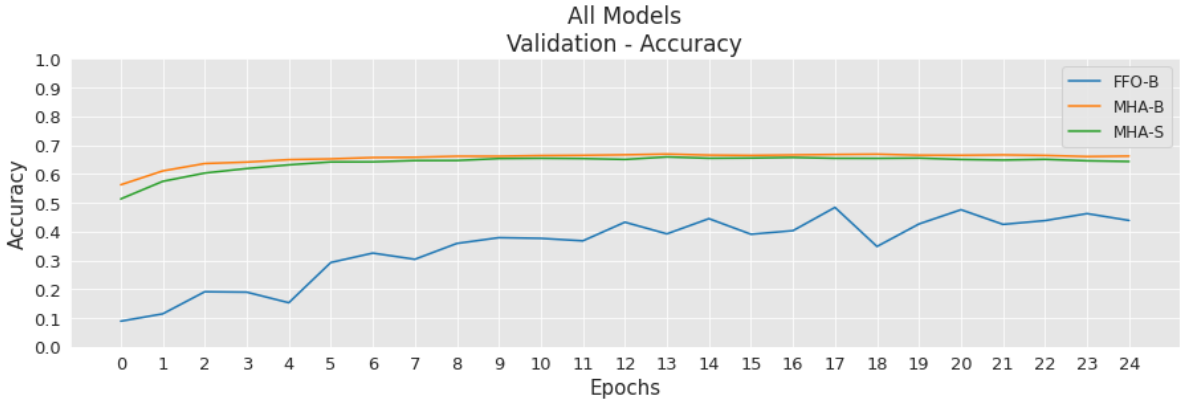


Figure 5.9: The accuracy achieved by proposed models during validation.

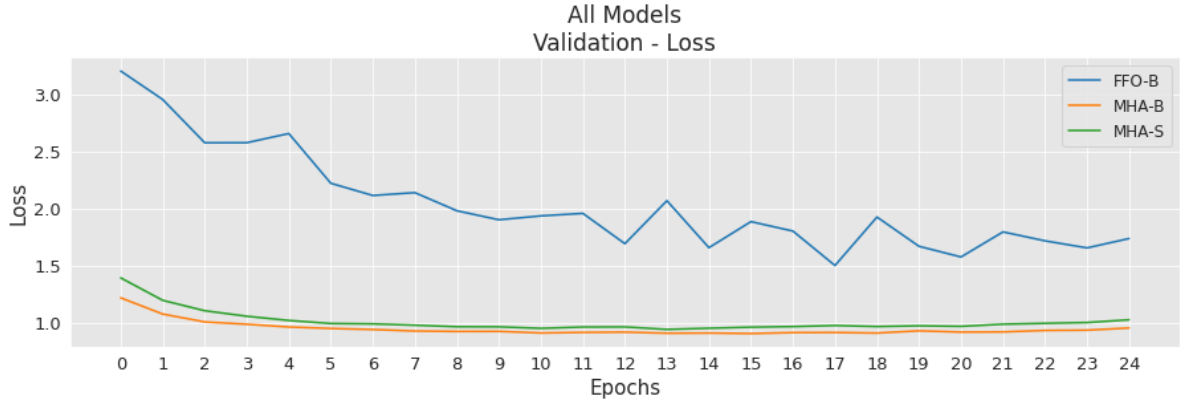


Figure 5.10: The loss values by proposed models during validation.

Summary of Training and Validation

Table 5.1 summarises the best epoch for each of the proposed models. The attention-based models perform considerably better than the non-attention-based FFO-B. Furthermore, MHA-S reaches its best performance at epoch 13, whereas MHA-B reaches its best two epochs later. Of the proposed models, the MHA-B performs the best with a validation accuracy of 66.4%, with MHA-S close behind with 65.9%. Surprisingly, MHA-S reaches a performance close to MHA-B's despite using a halved training set compared to MHA-B. On average, MHA-S uses half the time per epoch than what MHA-B uses.

		Training		Validation		
Model	Epoch	Loss	Accuracy	Loss	Accuracy	Mean time/epoch (hh:mm:ss)
FFO-B	17	1.693	0.442	1.501	0.484	00:52:16
MHA-B	15	0.814	0.699	0.905	0.664	01:11:06
MHA-S	13	0.861	0.684	0.940	0.659	00:35:22

Table 5.1: The best epochs during training/validation for each models. The best epoch for each models is the epoch where the model achieved the lowest validation loss.

5.3 Model Testing Results

Here, we will explore the results the proposed models achieve on the proposed testing sets: **Default** and **Phase sets**.

5.3.1 Default Testing Set

Top-K Accuracy

To calculate the top-K accuracy as a percentage, we take the number of test cases the target class falls within the model's top K predictions, divided by the total number of test cases.

MHA-B achieves a top-1 accuracy of 66.71% while MHA-S is close behind with 65.18%. FFO-B concludes with a top-1 accuracy of 48.22%, a significantly worse result compared to its attention-based counterparts. Furthermore, MHA-B outperforms MHA-S when comparing the top-K accuracy for $K = 1, 2, 3, 4, 5$. Based on these results, we can conclude that MHA-B > MHA-S > FFO-B in terms of prediction accuracy.

In conclusion, the results supports both Hypotheses 1 and 2. For the former, the attention-based models achieve a higher prediction accuracy than non-attention-based when presented the same task. For the latter, MHA-B achieves a better prediction accuracy than MHA-S.

	Top-K Accuracy (%)				
Model	**Top 1**	**Top 2**	**Top 3**	**Top 4**	**Top 5**
FFO-B	48.22	68.54	79.17	85.83	90.08
MHA-B	66.71	85.75	93.07	96.31	97.99
MHA-S	65.18	84.56	92.21	95.76	97.55

Table 5.2: Proposed models' top-k accuracies.

Top-K Invalid Discards

Top-K invalid discards is the number of test cases a model predicts at least one invalid class within its top K predictions. Naturally, it is desirable to have a lower total of invalid discards.

The top-k invalid discards for each of the proposed models are shown in Tables 5.3 and 5.4. A similar pattern from the previous sections is seen here: FFO-B performs subpar when compared to MHA-B and MHA-S. For $K = 1$, FFO-B predicts an invalid class in 0.150% of test cases. Although somewhat impressive in its own right, this result is overshadowed by 0.001% and 0.004% achieved by MHA-B and MHA-S, respectively.

Both our attention-based models achieve better results than Honghai's proposed model [11], which discarded an invalid class in approximately 0.04% of its test cases. That said, they used a smaller training set of 1,260,000 board states, and a more extensive testing set of 140,000 board states.

Just like for prediction accuracy, the attention-based models outperform the non-attention-based counterpart when we measure the top-k invalid discards. These results support Hypothesis 4.

	Top-K Total Invalid Discards				
Model	**Top 1**	**Top 2**	**Top 3**	**Top 4**	**Top 5**
FFO-B	102	966	4572	12155	21912
MHA-B	1	44	261	1247	3733
MHA-S	3	51	312	1324	3811
Minimum	0	0	18	219	621

Table 5.3: The top-K total invalid discards for each model on the default testing set. **Minimum** is the lowest possible number of invalid discards for the corresponding Top-K.

	Top-K Total Invalid Discards (%)				
Model	**Top 1**	**Top 2**	**Top 3**	**Top 4**	**Top 5**
FFO-B	0.150	1.421	6.724	17.875	32.224
MHA-B	0.001	0.065	0.384	1.834	5.490
MHA-S	0.004	0.075	0.459	1.947	5.604
Minimum	0.000	0.000	0.026	0.322	0.913

Table 5.4: Same as Table 5.3, but as percentages of total testing set.

5.3.2 Phase Sets

Alongside testing on the default testing set, the models are also tested on the game phase testing sets.

Top-K Accuracy

Table 5.5 summarises the results from testing on the phase datasets for each models.

Again, attention-based models outperform the non-attention-based model, where the MHA-B achieves a slightly better prediction accuracy than MHA-S. An observation is that all proposed models perform better on **Early** > **Mid** > **Late**. The only outliers are MHA-B and MHA-S' top-1

accuracy on **Early** and **Mid**, where both models performed better on the **Mid** set than the **Early** set.

		Top-K Accuracy (%)				
Model	**Phase**	**Top 1**	**Top 2**	**Top 3**	**Top 4**	**Top 5**
FFO-B	Early	49.41	71.14	82.50	88.79	92.53
	Mid	48.98	67.24	77.30	83.77	88.34
	Late	43.39	60.11	70.17	77.49	82.94
MHA-B	Early	66.97	86.97	93.99	96.93	98.36
	Mid	68.15	85.78	92.41	95.76	97.56
	Late	62.15	80.16	88.80	93.62	96.35
MHA-S	Early	65.93	85.99	93.44	96.54	98.10
	Mid	66.62	84.24	91.39	94.96	97.17
	Late	61.02	79.36	87.96	92.82	95.87

Table 5.5: Summary of proposed models' top-k accuracy on phase sets.

Top-K Invalid Discards

Tables 5.6 and 5.7 summarises the top-K invalid discards achieved by each model on the phase sets.

Again, a similar pattern is observed: Attention-based models predict less invalid discards than the non-attention-based mode. MHA-B achieves a slightly better score than MHA-S for all K on **Mid** and **Late** sets. However, on the **Early** set, MHA-S manages to topple MHA-B for $K > 1$.

		Top-K Invalid Discards				
Phase	**Model**	**Top 1**	**Top 2**	**Top 3**	**Top 4**	**Top 5**
Early	FFO-B	36	450	2673	7347	13312
	MHA-B	2	21	122	486	1547
	MHA-S	2	20	112	481	1444
	Minimum	0	0	6	32	93
Mid	FFO-B	59	470	1752	4444	8515
	MHA-B	1	36	206	881	2470
	MHA-S	2	26	210	898	2542
	Minimum	0	0	11	167	479
Late	FFO-B	57	483	1908	4238	7489
	MHA-B	4	31	160	781	2308
	MHA-S	7	42	216	869	2481
	Minimum	0	1	34	270	770

Table 5.6: Top-K invalid discards on each phase dataset by the proposed models. **Minimum** is the lowest possible number of invalid discards for the corresponding top-K.

Phase	Model	Top-K Invalid Discards (%)				
		Top 1	**Top 2**	**Top 3**	**Top 4**	**Top 5**
Early	FFO-B	0.106	1.324	7.862	21.609	39.153
	MHA-B	0.006	0.062	0.359	1.429	4.550
	MHA-S	0.006	0.059	0.329	1.415	4.247
	Minimum	0.000	0.000	0.018	0.094	0.274
Mid	FFO-B	0.174	1.382	5.153	13.071	25.044
	MHA-B	0.003	0.106	0.606	2.591	7.265
	MHA-S	0.006	0.076	0.618	2.641	7.476
	Minimum	0.000	0.000	0.032	0.491	1.409
Late	FFO-B	0.168	1.421	5.612	12.465	22.026
	MHA-B	0.012	0.091	0.471	2.297	6.788
	MHA-S	0.021	0.124	0.635	2.556	7.297
	Minimum	0.000	0.003	0.100	0.794	2.265

Table 5.7: Same as Table 5.6 but in percentages.

Discussion

When comparing both prediction accuracy and the number of invalid discards predicted, it is clear that the **Late** phase set is the most demanding one of the three phase sets. This contradicts Hypothesis 3, which states that the models would perform better on board states of later phases than earlier. Although more information is revealed on the board as a round progresses, the models produce less satisfactory results on later stages of the board. A reason for this drop in performance might be due to how later phases highlight the following dilemma for each player: To pursue victory for the current round or fold to cut losses?

Recall that players race to build their hand into a ready state in each round, which hand compositions that are considered as ready do not change across different games as they are part of the game rules. Due to the fixed nature of hand readiness, player actions tend to be more streamlined early into the game, when players are more likely still building their hand towards the ready state.

As players' hands near completion as the round progresses, the likelihood to play into your opponents' hands increases correspondingly — naturally, the motivation to fold increases as well. Players fold by discarding tiles that are less likely to be claimed by other players, colloquially referred to as *safe tiles*. Whether a tile is considered safe to discard depends on the current board state. Therefore, the difference between selecting which tile to discard when pursuing a ready state versus folding is significant: When building towards a ready state, players rely on fixed game rules. For folding, a correct reading of the current board state is essential to determine which tile is considered safe to discard.

For folding, a correct reading of the current board state is essential to determine "safe tiles", whereas pursuing tenpai state relies more on fixed game rules. The more dynamic nature of folding might explain why the models perform worse on the late set than on the **Early** and **Mid** set.

5.4 Comparison with State-of-the-Art Models

The research question inquires how our proposed models fare compared to Suphx. In Section 2.6 the results from multiple state-of-the-art models were explored and compared against each other. As seen in Table 5.8, our own proposed models have integrated into the same table to see how their results measure up to the others, including Suphx.

Published	Model	Source	Training Set	Validation Set	Testing Set	Accuracy
2020	Suphx	[7]	15,000,000	10,000	50,000	76.7%
2020	Honghai	[10]	18,000,000	26,000	-	72.3%
2019	Gao	[9]	1,921,798	213,533	50,000	70.4%
2018	Gao	[8]	540,000	60,000	30,000	68.8%
2019	Honghai (w/ yaku)	[11]	1,260,000	140,000	-	66.8%
2021	**MHA-B**	**Ours**	**34,000,000**	**34,000**	**68,000**	**66.7%**
2021	**MHA-S**	**Ours**	**17,000,000**	**34,000**	**68,000**	**65.2%**
2019	Honghai (w/o yaku)	[11]	1,260,000	140,000	-	64.9%
2015	Mizukami	[12]	-	-	-	62.1%
2017	Tsuyoshi	[20]	-	-	-	54.0%

Table 5.8: Summary of contemporary discard models and their corresponding results, sorted by publishing date. A dash (-) indicates unpublished information. Note that it is not recommended to directly compare the results due to vastly different training and testing scheme.

The difference between the Suphx and our two attention-based models MHA-B and MHA-S, is about 10.0% and 11.5%, respectively. It is safe to assume that the gap is considerable and that our models still have a long way to go. Similarly, Honghai's (2020) model surpasses MHA-B by a margin of 5.6% and MHA-S by 7.1%.

Both Gao (2018) and Gao (2019) achieve a better prediction accuracy than our models. Although the gap in accuracy is small, the sizes of used training sets vary greatly. Although just speculations, it is reasonable to assume that Gao (2018) and Gao (2019) would both achieve a better accuracy if trained on training sets similarly sized as ours as more data is often better.

Both our attention-based models achieve very similar results compared to Honghai's (2019) two models. Although just barely, our models surpass Honghai's discard model without yaku prediction assistance. With yaku prediction assistance, Honghai's discard model manages to outperform both our models by a small margin. That said, we train our models with a much larger training set than the one used for theirs, which might skew the results.

Evidence suggests that both our attention-based models manage to outdo both Tsuyoshi's and Mizukami's models. Nonetheless, as they have not disclosed the size of their datasets, this comparison may be inconclusive.

5.4.1 Top-K accuracy

Table 5.9 compares the top-K accuracy achieved by our models and Gao's (2018) and Mizukami's. Evidently, there is a gap between our models and Gao's.

			Top-K Accuracy		
Published	**Model**	**Source**	**1st**	**2nd**	**3rd**
2018	Gao (2018)	[8]	68.8%	93.6%	96.5%
2021	**MHA-B**	**Ours**	**66.6%**	**85.7%**	**92.9%**
2021	**MHA-S**	**Ours**	**65.0%**	**84.3%**	**92.0%**
2015	Mizukami	[12]	62.1%	82.9%	90.9%

Table 5.9: Summary of top-K accuracy achieved by various models.

5.5 Sorted Probability Distribution

Figure 5.11 shows the average distribution of probabilities sorted by prediction priority for the proposed models. Note that prediction priority does *not* take into consideration if the prediction is correct or not. Instead, we use the average probability distribution as a metric for how *confident* the models are with their predictions.

On average, attention-based models allocate more probability on the 1st prioritised prediction than the non-attention-based model does. For each prediction after the first, the roles are flipped, and the non-attention-based model begins to assign more probability than its attention-based counterparts. Interestingly, MHA-B and MHA-S have close to identical distributions despite their different training schemes.

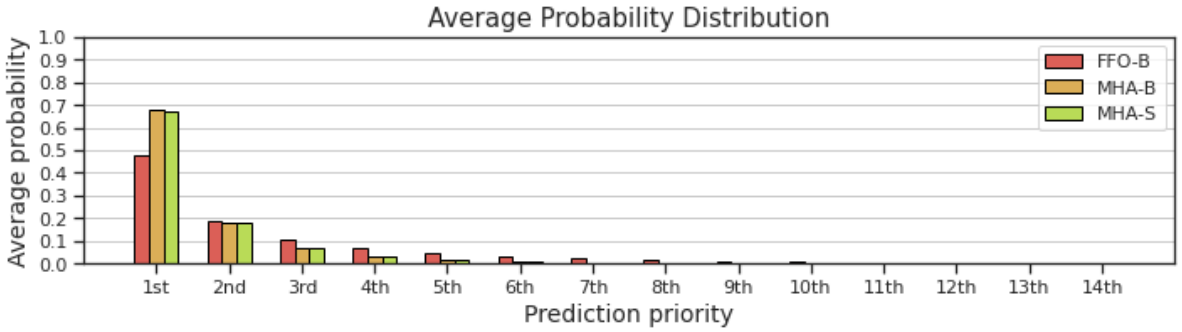


Figure 5.11: The average probability distribution, sorted on priority from highest to lowest, for each model on the default testing set. Only the top 14 out of 34 prediction priorities are shown for the sake of brevity.

5.6 Attention Matrix Generated

Something worth exploring when using attention-based models is how their attention mechanisms operate. A way to do this is to analyse their attention layer's output, the *attention matrix*. Although it is named attention matrix, it is still a 1D array of 374 values since it follows the exact dimensions as the given input.

When given the input seen in Figure 5.12, MHA-B's attention layer outputs the attention matrix seen in Figure 5.13, whereas MHA-S's the attention matrix seen in Figure 5.14. Although the values of the outputted attention matrix change depending on the given input data, the changes are slight and therefore negligible.

One should not reason with the attention matrix values as-is but instead analyse how far away the values are from 0. This is because the input data is multiplied by the attention matrix,

applying the computed attention. Therefore, zeros in the attention matrix will result in a zero in the corresponding field in the input data after matrix multiplication. These fields, if zero, will propagate the zero further into the dense net as weights will not change it. As a result, zeros in the attention matrix highlight fields in the input data the model ignores.

From both attention matrices, we can see that the models learn to disregard the padding fields by allocating them the value 0. Another observation is that the models learn to discern the different logical groupings of fields on their own. For example, P1 - P3 melds share similar values, as we demonstrate using colours. Interestingly, the values in the P0 melds fields are different from P1 - P3 melds, despite also representing melds. This distinction implies that the model learns to treat P0 melds differently from P1 - P3 melds. A similar observation is seen with the P0 pool and P1 - P3 pools as well. A possible reason behind this difference is that the P1 - P3's fields, i.e. the opponents' fields, require greater attention than other fields as they belong to the opponents. How much this difference affects the final predictions is challenging to estimate without examining the neurons inside the dense layers after the attention layer.

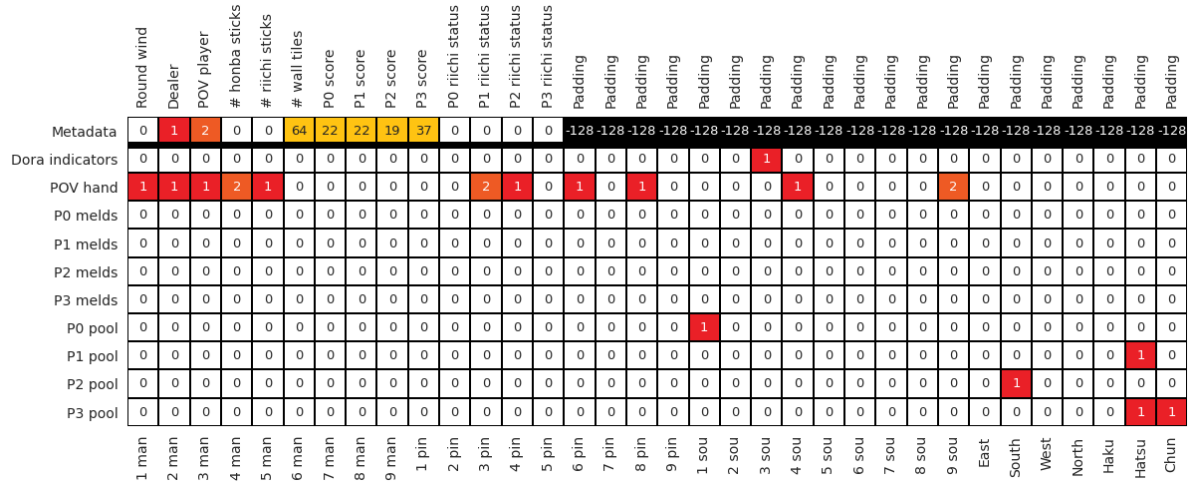


Figure 5.12: An example of data structure.

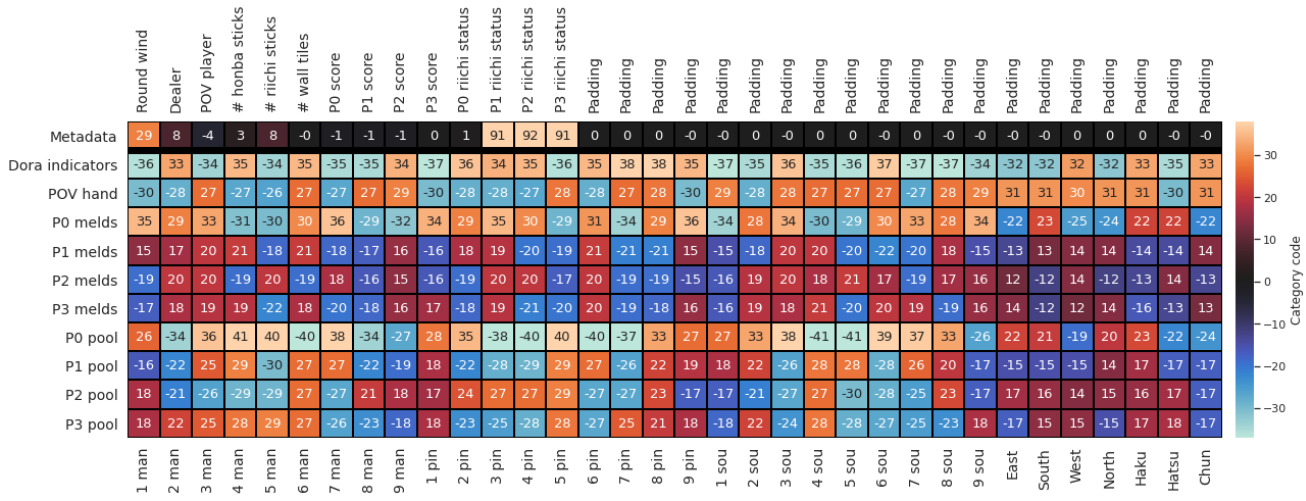


Figure 5.13: MHA-B's generated attention matrix when fed the data input shown in Figure 5.12.

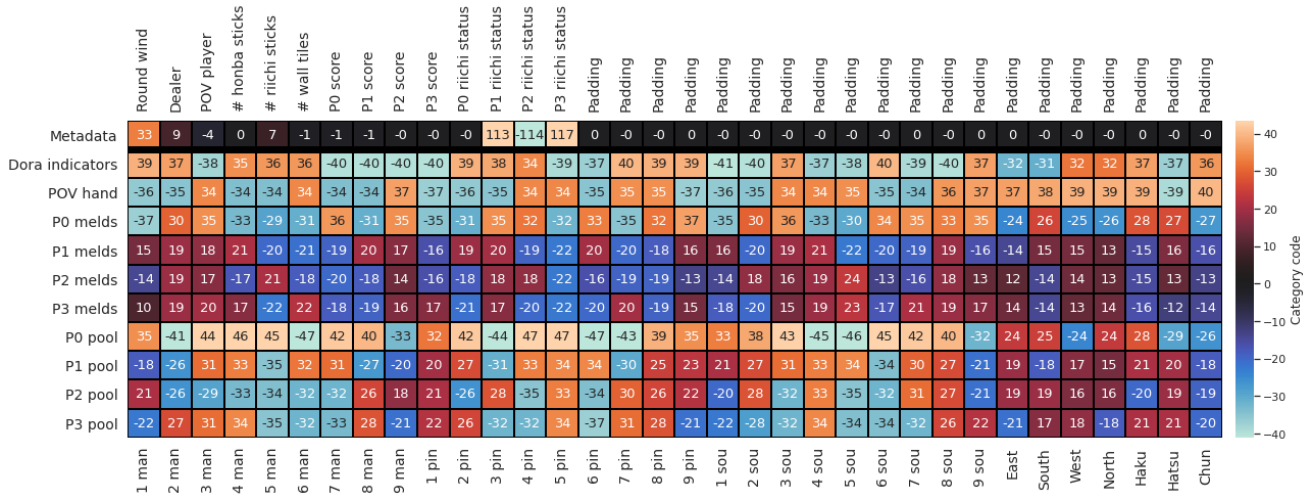


Figure 5.14: MHA-S's generated attention matrix when fed the data input shown in Figure 5.12.

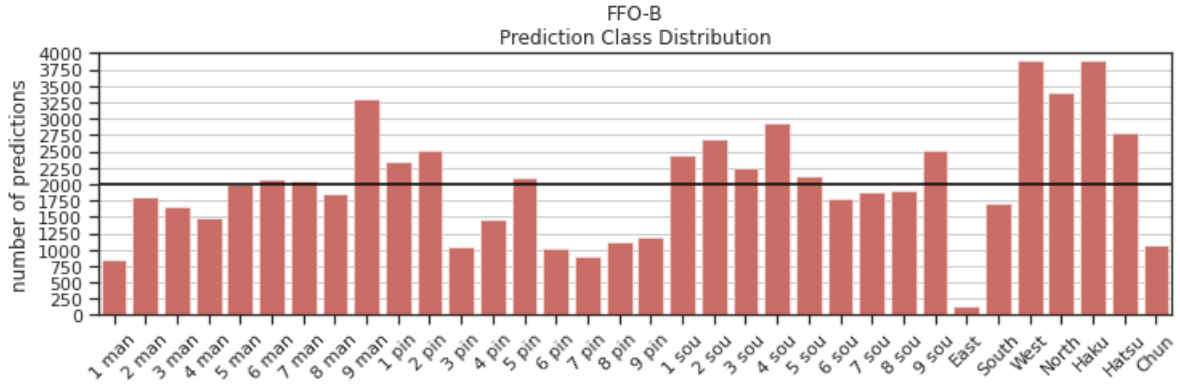
5.7 Prediction Class Distribution

The prediction class distributions for each model can be seen in Figure 5.15. The associated confusion matrices are found in Appendix B and the precision scores in Appendix A. An interesting observation is that the prediction class distribution of FFO-B is very imbalanced compared to its attention-based counterparts'. The East and Chun tile classes are most notable, which are predicted in less than 250 and 1250 test cases, a severe underrepresentantion. It seems like the FFO-B prioritise other honour tiles, most strikingly the **West** (🀂), **Haku** (🀆) and **North** (🀃).

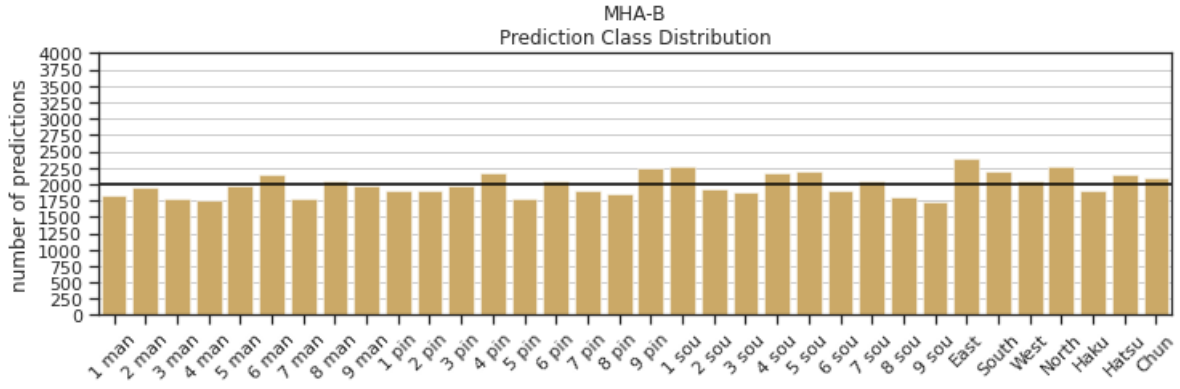
The question remains: did the FFO-B learn the innate value of **East** (🀀), and therefore lean towards deprioritising discarding it? In Japanese Mahjong, the East tile is regarded as a rather valuable tile. Recall that the round wind begins in East, which facilitates additional winning condition, i.e. yaku, if you collect three or more of the East tiles as it matches the round wind. This winning condition is regarded as fast and easy to reach, making it attractive for all players. Furthermore, the East tile is especially valuable for the current dealer because their seat wind matches the tile, increasing its potential value due to another yaku. Dealers are incentivised to retain the dealer position due to its benefits, and will therefore in many cases seek the East tile specifically.

Recall that the training set is balanced, which should in theory counterbalance such biases. It can also be due to some subtle correlation found in the board states that singles out the East tile through sheer coincidence.

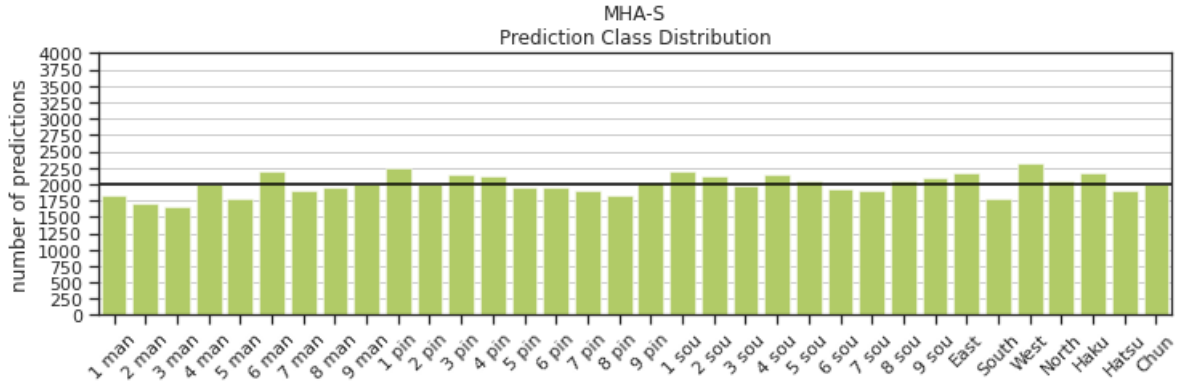
Whatever the reason that setbacks the FFO-B did not affect the attention-based models in the same way. Both MHA-B and MHA-S discarded the East tile above the target number. Other than that, both MHA-B and MHA-S manages to predict a rather balanced prediction distribution. The elaborated results does support Hypothesis 1 from an alternative direction than the experiments on top-k prediction accuracy did.



(a) FFO-B



(b) MHA-B



(c) MHA-S

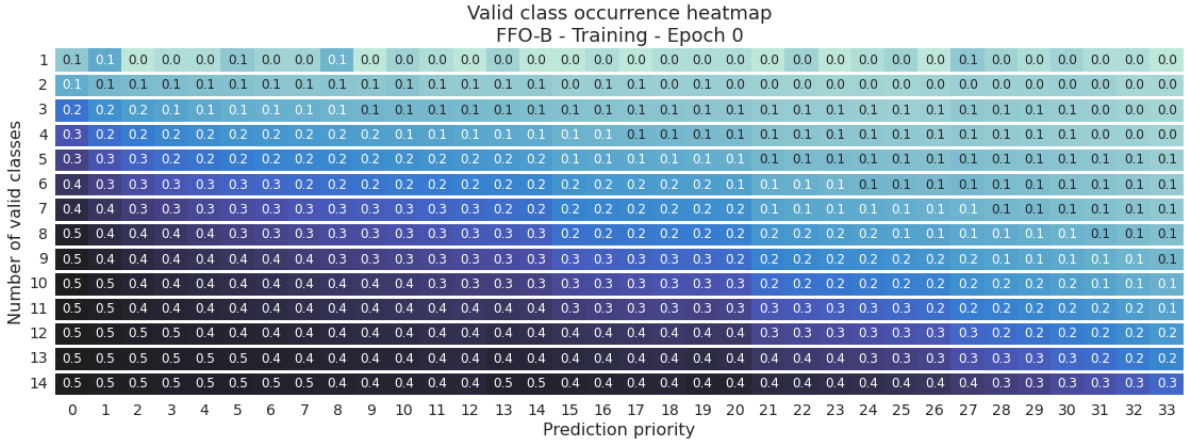
Figure 5.15: The distribution of predicted class for each model on the default testing set. The black line marks the actual cases, i.e. 2000 cases per tile class.

5.8 Valid class occurrence

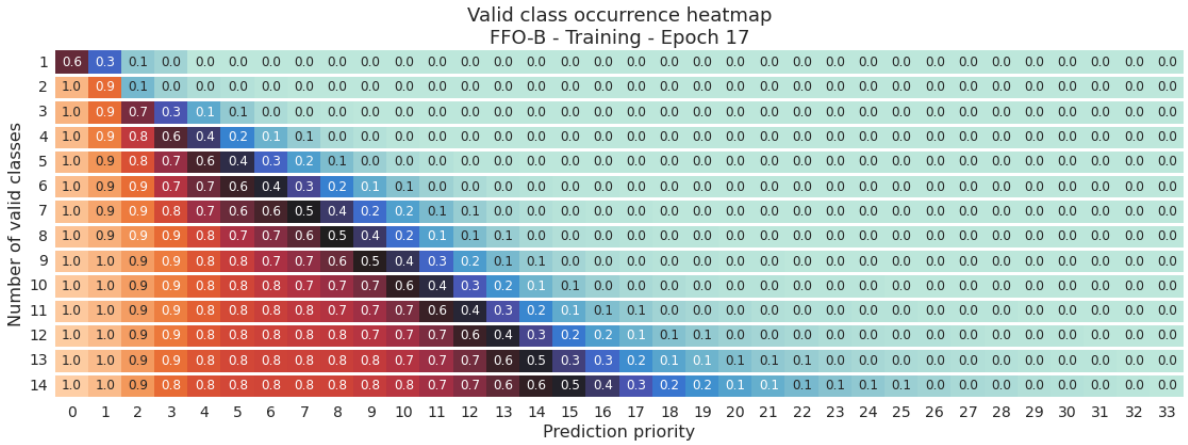
In Sections 5.3.1 and 5.3.2 we measured the top-K invalid discard for each model as a way to determine the models' ability to discern valid class from invalid classes. An alternative approach is to count how often valid classes occur sorted by prediction order. We call this metric the *valid class occurrence*. As some board states have fewer valid classes than other board states, e.g. the POV player's hand has dwindled or contains duplicates, we separate cases by the number of valid classes. This section demonstrates the results using this method.

5.8.1 FFO-B

FFO-B's ability to separate valid classes from invalid ones improves from the first epoch to epoch 17, as seen in Figure 5.16. In epoch 0, the model has not fully learned the separation as the values are more on the cold side - closer to 0 than 1. In epoch 17, the values become warmer towards the left, resembling a downward "staircase". This staircase formation is optimal.



(a) Valid Class Occurrence Heatmap after first epoch

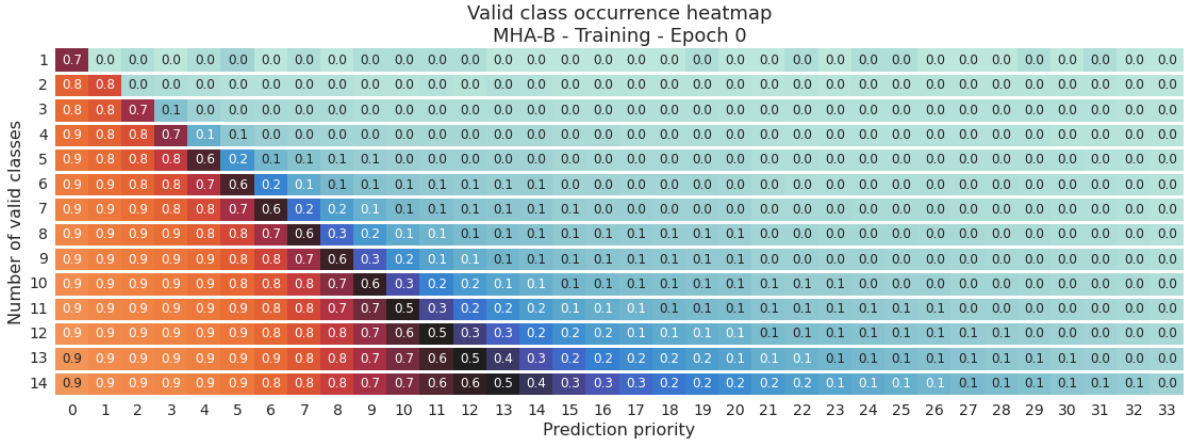


(b) Valid Class Occurrence Heatmap after best epoch

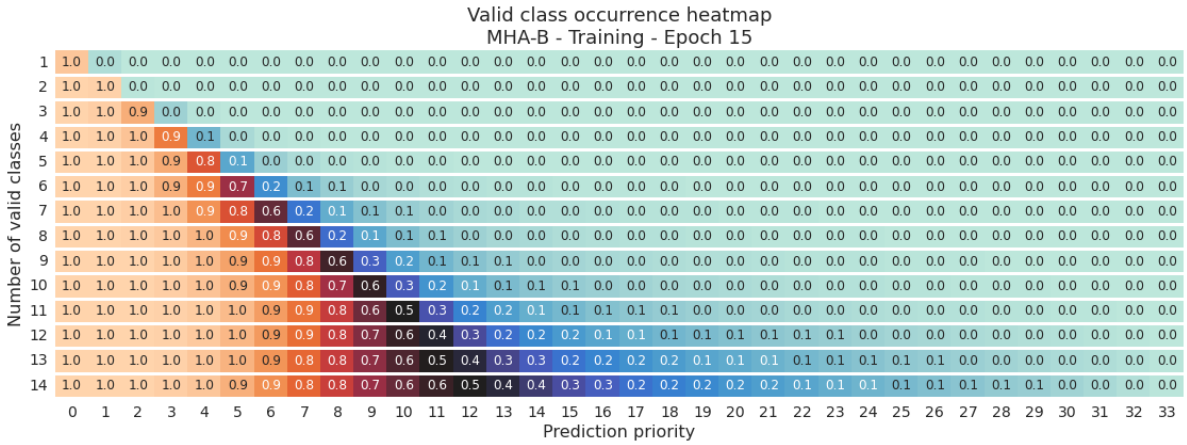
Figure 5.16: The valid class occurrence heatmap of FFO-B during training.

5.8.2 MHA-B

Figure 5.17 shows the valid class occurrence for MHA-B. In epoch 0, MHA-B has already grasped the importance of valid classes as the hot staircase formation has already taken form, as seen in Figure 5.17. In epoch 15, this staircase has become more pronounced.



(a) Valid Class Occurrence Heatmap after first epoch

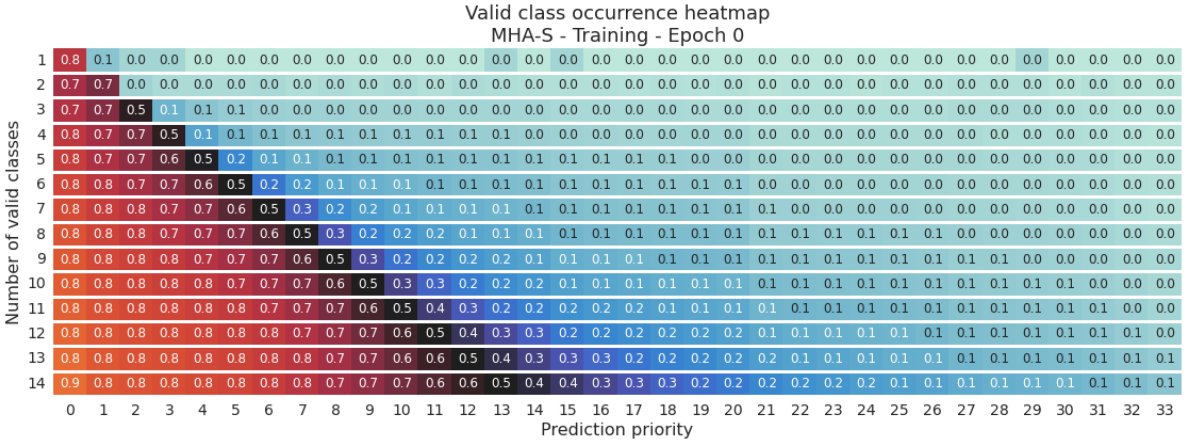


(b) Valid Class Occurrence Heatmap after best epoch

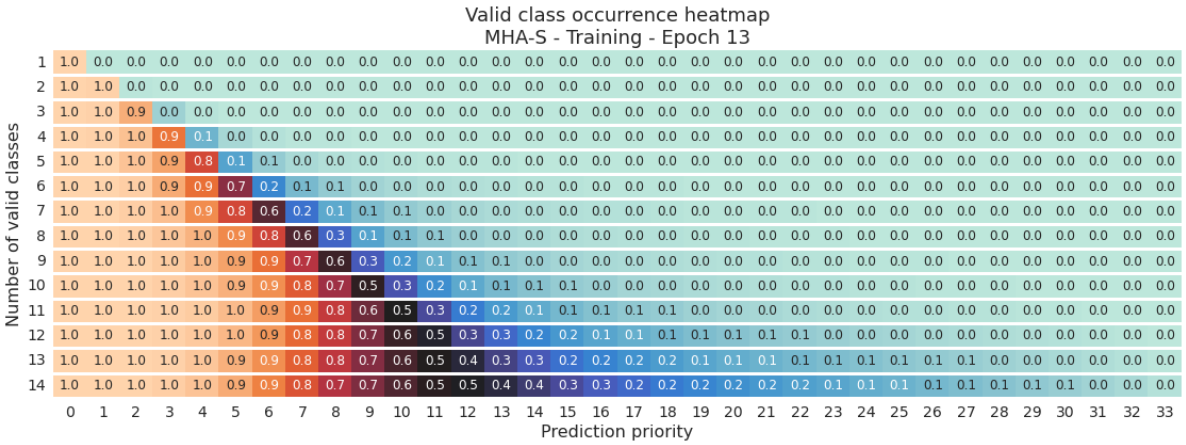
Figure 5.17: The valid class occurrence heatmap of MHA-B during training.

5.8.3 MHA-S

MHA-S follows a similar trail as MHA-B, as seen in Figure 5.18.



(a) Valid Class Occurrence Heatmap after first epoch



(b) Valid Class Occurrence Heatmap after best epoch

Figure 5.18: The valid class occurrence heatmap of MHA-S during training.

5.8.4 Comparison of valid class occurrences from each models

Attention-based models learn to discern valid classes from the invalid classes at a higher capability than the non-attention-based model. Similar to previous experiments, MHA-B has the upper hand compared to MHA-S, as the staircase formation is more pronounced in MHA-B's results. All in all, this is yet another support for Hypothesis 4.

5.9 Inversions

An optimal prediction would prioritise all valid classes above all invalid classes. The ordeal of separating valid classes from invalid ones can be thought of as a sorting problem of 34 binary numbers, where 0 represents an invalid class whereas 1 a valid class. The goal in our case would be to sort the array of 34 binary numbers in a descending order: all 1's are before all 0's. A way to determine how far away the array is from being sorted is by using the number of *inversions* as a metric. Lowering the number of inversions is optimal.

5.9.1 FFO-B

Table 5.10 summarises the inversions statistics of FFO-B. When comparing the mean number of inversions between epoch 0 and 17, we can see that the model has learned to decrease the number of inversions. In other words, the model has learned to better prioritise valid class from invalid class.

		Max		Mean		Median	
# Valid classes	# Cases	Epoch 0	Epoch 17	Epoch 0	Epoch 17	Epoch 0	Epoch 17
1	47	32	3	13.43	0.51	10	0
2	7845	64	33	25.93	0.46	26	0
3	114933	93	40	33.92	1.23	32	0
4	318664	119	52	40.72	2.27	39	1
5	463980	145	55	50.13	3.58	48	2
6	970986	166	66	57.37	5.33	55	4
7	1580104	187	71	63.52	6.76	61	5
8	2173605	203	100	71.35	7.26	68	6
9	3855827	223	90	78.83	8.55	75	7
10	5590652	238	113	86.54	10.52	82	8
11	7211039	253	135	95.58	13.20	91	11
12	7192940	264	135	103.42	16.66	99	14
13	3808192	271	147	111.41	20.53	109	18
14	711186	274	142	122.02	25.66	122	23

Table 5.10: Inversion statistics of FFO-B during training.

5.9.2 MHA-B

Table 5.11 summarises the inversions statistics of MHA-B. The mean number of inversions are below 1 in for when the number of valid classes is ≤ 5.

		Max		Mean		Median	
# Valid classes	# Cases	Epoch 0	Epoch 15	Epoch 0	Epoch 15	Epoch 0	Epoch 15
1	47	31	1	4.57	0.02	0	0
2	7845	64	14	5.23	0.01	0	0
3	114933	93	27	7.75	0.12	0	0
4	318664	120	38	9.22	0.34	0	0
5	463980	141	62	11.91	0.79	0	0
6	970986	164	74	13.53	2.01	1	0
7	1580104	183	86	15.11	3.41	2	1
8	2173605	200	105	17.29	3.84	2	1
9	3855827	221	124	19.05	5.27	3	2
10	5590652	237	150	21.22	6.63	4	3
11	7211039	253	140	24.05	9.04	6	5
12	7192940	263	170	27.15	14.16	8	10
13	3808192	269	158	30.88	19.86	12	16
14	711186	277	157	36.47	24.77	18	21

Table 5.11: Inversion statistics of MHA-B during training.

5.9.3 MHA-S

Table 5.12 summarises the inversions statistics of MHA-S.

		Max		Mean		Median	
# Valid classes	# Cases	Epoch 0	Epoch 13	Epoch 0	Epoch 13	Epoch 0	Epoch 13
1	28	29	0	2.29	0.00	0	0
2	3911	63	3	8.66	0.01	0	0
3	56912	93	23	13.21	0.11	1	0
4	159024	119	28	15.56	0.37	1	0
5	231045	144	56	19.88	0.98	4	0
6	485642	168	62	22.13	2.16	4	0
7	790376	184	73	24.50	3.56	5	1
8	1086816	208	117	28.08	3.81	7	1
9	1927665	221	109	30.73	5.13	7	2
10	2795036	240	120	33.78	6.37	9	3
11	3604982	253	135	37.23	8.63	10	5
12	3595987	264	147	40.38	14.00	12	10
13	1904903	269	161	43.92	20.71	15	16
14	357673	271	162	49.28	28.30	20	24

Table 5.12: Inversion statistics of MHA-S during training.

Discussion on inversion statistics

The difference between the MHA-B and FFO-B at their best epochs is not as significant as to other metrics we have elaborated earlier. The most evident difference is their results on epoch 0, where FFO-B has not learned to separate valid classes from invalid ones as well as MHA-B does.

On epoch 15, MHA-B manages to lower the median number of inversions to 0 for cases when the number of valid classes is ≤ 6. FFO-B manages to achieve the same for cases where the number of valid classes is ≤ 3.

For both MHA-B and MHA-S, the mean number of inversions are below 1 for cases when the number of valid classes is ≤ 5. It is worth noting that we are comparing the models' performance during their training, meaning that the comparison between FFO-B and MHA-B is more fair as both use the identical training set, compared to MHA-S'.

In conclusion, it seems like the all three models learn to separate the classes at a similar level at their each best epoch. The difference is how fast they reach that level, where the non-attention-based FFO at epoch 0 begins with a significant worse results. Although this somewhat supports Hypothesis 4, it is not to be reliable as the results found in Section 5.3.1, where we compared the results from model testing rather than model training.

Chapter 6

Conclusion

This thesis investigates to what extent attention-based models can learn to predict the next tile to be discarded in Japanese Mahjong at a level comparable to Suphx, the current state-of-the-art Mahjong AI.

We propose four hypotheses as a basis for our experiments: First, that an attention-based model will yield a higher prediction accuracy the non-attention-based version of the same model when presented with the same tile discarding task. Second, that the attention-based models will yield a higher prediction accuracy when trained on a larger training set than a smaller one. Third, that the proposed models will yield higher accuracy on board states that is nearing its end than board states that are closer to the onset of the round. Fourth, that the proposed attention-based models will predict less invalid classes than non-attention based model when presented with the same task.

Due to its high requirements, a deliberate choice to avoid using a CNN architecture is made to improve the reproducibility of our experiments. Instead, we produced a smaller and simpler model. Our model architecture incorporates a multi-head attention layer. Furthermore, we propose a data structure that is the smallest of its kind compared to related works.

To determine whether the addition of an attention mechanism leads to improved performance, we introduce a simpler non-attention-based model, FFO-B, to serve as a benchmark. Our attention-based models MHA-B and MHA-S achieve a prediction accuracy of 66.7% and 65.2%, respectively. Our attention-based models outperform their non-attention-based counterparts when it comes to prediction accuracy, which supports our first hypothesis.

The second hypothesis questions whether there is a correlation between the models' final performance and the number of board states used for their training. To determine how much the size of the training set affects the models' performance, MHA-S is trained on a training set half of the size of the one MHA-B trains on. Although MHA-B scores higher than MHA-S in all of our experiments, the difference in results is relatively insignificant.

We design unique testing sets called phase sets to see how different stages of a round affect a model's prediction accuracy. The results show that our models perform better on board states on rounds in Early to Mid phases as opposed to rounds in the Late phases. This invalidates our third hypothesis.

Due to the nature of the presented prediction space, our models can predict invalid tiles classes. MHA-B and MHA-S incorrectly predict invalid classes in 0.001% and 0.004% of the given board states, respectively. Likewise, with prediction accuracy, both MHA-B and MHA-S surpass FFO-B by a significant margin, and therefore supports our fourth hypothesis.

Our attention-based models outperform older models such as Mizukami's and Tsuyoshi's, and they come close to some of the contemporary models like Honghai's. However, in comparison to the current leading model Suphx, there is still a significant gap.

Nonetheless, the proposed models show promising results despite not using a CNN architecture, being smaller than contemporary models, and being trained on a compact data structure. All in all, this forecasts a potential future for attention-based AI in Japanese Mahjong.

6.1 Future Work

Although we filter the used data meticulously, certain aspects were left untouched. We propose two potential improvements: First, is to filter out games where a player has disconnected, as any absent player will automatically discard any tile they draw. We believe the inclusion of these games will prove to be detrimental to model training. Second, is to limit the experiments to game states where the POV player is always the victorious player. We include game states from all players' POV, which may not be the optimal method.

Honghai's [11] solution is similar to ours when it comes to the model architecture and the scale of the data structure used. They found success in incorporating information of certain yakus in their data structure, which we believe is something worth exploring for the same purposes.

Another suggestion for future work is to examine if including information about past actions in our data structure, similar to how [8] and [9] does it, will improve models' accuracy.

Bibliography

[1] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel and et al., 'A general reinforcement learning algorithm that masters chess, shogi, and go through self-play,' *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018, ISSN: 0036-8075. DOI: 10.1126/science.aar6404.

[2] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel and et al., 'Mastering atari, go, chess and shogi by planning with a learned model,' *Nature*, vol. 588, no. 78397839, pp. 604–609, Dec. 2020, ISSN: 1476-4687. DOI: 10.1038/s41586-020-03051-4.

[3] N. Brown and T. Sandholm, 'Superhuman ai for multiplayer poker,' *Science*, vol. 365, no. 6456, pp. 885–890, 2019. DOI: 10.1126/science.aay2400.

[4] OpenAI, C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. d. O. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski and S. Zhang, *Dota 2 with large scale deep reinforcement learning*, 2019. DOI: https://doi.org/10.1038/s41586-019-1724-z. arXiv: 1912.06680 [cs.LG].

[5] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, 'Grandmaster level in starcraft ii using multi-agent reinforcement learning,' *Nature*, vol. 575, no. 7782, pp. 350–354, 2019. DOI: https://doi.org/10.1038/s41586-019-1724-z.

[6] Y. Zheng and S. Li, 'A review of mahjong ai research,' in *Proceedings of the 2020 2nd International Conference on Robotics, Intelligent Control and Artificial Intelligence*, ACM, Oct. 2020, pp. 345–349, ISBN: 978-1-4503-8830-6. DOI: 10.1145/3438872.3439104. [Online]. Available: https://dl.acm.org/doi/10.1145/3438872.3439104.

[7] J. Li, S. Koyamada, Q. Ye, G. Liu, C. Wang, R. Yang, L. Zhao, T. Qin, T.-Y. Liu and H.-W. Hon, 'Suphx: Mastering mahjong with deep reinforcement learning,' *arXiv:2003.13590 [cs]*, Mar. 2020. [Online]. Available: http://arxiv.org/abs/2003.13590.

[8] G. Shiqi, O. Fuminori, K. Yoshihiro and T. Yoshimasa, 'Supervised learning of imperfect information data in the game of mahjong via deep convolutional neural networks,' vol. 2018, Nov. 2018, pp. 43–50. [Online]. Available: http://id.nii.ac.jp/1001/00191963/.

[9] S. Gao, F. Okuya, Y. Kawahara and Y. Tsuruoka, 'Building a computer mahjong player via deep convolutional neural networks,' *CoRR*, vol. abs/1906.02146, 2019. arXiv: 1906.02146. [Online]. Available: http://arxiv.org/abs/1906.02146.

[10] H. Long and K. Tomoyuki, 'Training japanese mahjong agent with two dimension feature representation,' vol. 2020, Nov. 2020, pp. 125–130. [Online]. Available: http://id.nii.ac.jp/1001/00207563/.

[11] L. Honghai and T. Kaneko, 'Improving mahjong agent by predicting types of yaku,' vol. 2019, pp. 206–212, Nov. 2019.

[12] N. Mizukami and Y. Tsuruoka, 'Building computer mahjong players based on expected final ranks,' in *The 20th Game Programming Workshop*, 2015, pp. 179–186. [Online]. Available: http://id.nii.ac.jp/1001/00145771/.

[13] D. Chiba, *Riichi Book I: A Mahjong Strategy Primer for European Players*. 2016. [Online]. Available: https://dainachiba.github.io/RiichiBooks/.

[14] O. Morgenstern and J. Von Neumann, *Theory of games and economic behavior*. Princeton university press, 1953.

[15] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org.

[16] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard and L. D. Jackel, 'Backpropagation Applied to Handwritten Zip Code Recognition,' *Neural Computation*, vol. 1, no. 4, pp. 541–551, Dec. 1989, ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541. eprint: https://direct.mit.edu/neco/article-pdf/1/4/541/811941/neco.1989.1.4.541.pdf. [Online]. Available: https://doi.org/10.1162/neco.1989.1.4.541.

[17] V. Nair and G. E. Hinton, 'Rectified linear units improve restricted boltzmann machines,' in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML'10, Haifa, Israel: Omnipress, 2010, pp. 807–814, ISBN: 9781605589077.

[18] A. L. Maas, A. Y. Hannun and A. Y. Ng, 'Rectifier nonlinearities improve neural network acoustic models,' in *Proc. icml*, vol. 30, 2013, p. 3.

[19] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, *Attention is all you need*, 2017. arXiv: 1706.03762 [cs.CL].

[20] T. Tsuyoshi and S. Kazutomo, 'Cnn mahjong - an effectivity of cnn structure for mahjong,' vol. 2017, Nov. 2017, pp. 163–170. [Online]. Available: http://id.nii.ac.jp/1001/00183764/.

[21] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest and A. Rush, 'Transformers: State-of-the-art natural language processing,' in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. DOI: 10.18653/v1/2020.emnlp-demos.6. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[22] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly and et al., 'An image is worth 16x16 words: Transformers for image recognition at scale,' *arXiv:2010.11929 [cs]*, Oct. 2020. [Online]. Available: http://arxiv.org/abs/2010.11929.

[23] C.-F. Chen, Q. Fan and R. Panda, *Crossvit: Cross-attention multi-scale vision transformer for image classification*, 2021. arXiv: 2103.14899 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2103.14899.

[24] A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lučić and C. Schmid, *Vivit: A video vision transformer*, 2021. arXiv: 2103.15691 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2103.15691.

[25] H. Chen, Y. Wang, T. Guo, C. Xu, Y. Deng, Z. Liu, S. Ma, C. Xu, C. Xu and W. Gao, *Pre-trained image processing transformer*, 2020. arXiv: 2012.00364 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2012.00364.

[26] J. Chen, Y. Lu, Q. Yu, X. Luo, E. Adeli, Y. Wang, L. Lu, A. L. Yuille and Y. Zhou, *Transunet: Transformers make strong encoders for medical image segmentation*, 2021. arXiv: 2102.04306 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2102.04306.

Appendix A

Precision, Recall and F1-Score

Class	Precision	Recall	F1-score	Support
1 man	0.556	0.233	0.328	2000
2 man	0.498	0.448	0.472	2000
3 man	0.568	0.470	0.514	2000
4 man	0.596	0.442	0.507	2000
5 man	0.512	0.513	0.513	2000
6 man	0.506	0.523	0.514	2000
7 man	0.544	0.554	0.549	2000
8 man	0.540	0.500	0.520	2000
9 man	0.440	0.726	0.548	2000
1 pin	0.510	0.597	0.550	2000
2 pin	0.460	0.579	0.512	2000
3 pin	0.651	0.336	0.443	2000
4 pin	0.595	0.432	0.501	2000
5 pin	0.402	0.423	0.413	2000
6 pin	0.647	0.328	0.435	2000
7 pin	0.663	0.296	0.409	2000
8 pin	0.498	0.277	0.356	2000
9 pin	0.538	0.319	0.401	2000
1 sou	0.501	0.614	0.552	2000
2 sou	0.447	0.600	0.512	2000
3 sou	0.506	0.568	0.535	2000
4 sou	0.425	0.623	0.505	2000
5 sou	0.512	0.543	0.527	2000
6 sou	0.549	0.485	0.515	2000
7 sou	0.548	0.517	0.533	2000
8 sou	0.531	0.502	0.516	2000
9 sou	0.508	0.639	0.566	2000
East	0.484	0.031	0.058	2000
South	0.450	0.382	0.413	2000
West	0.395	0.767	0.522	2000
North	0.417	0.707	0.524	2000
Haku	0.346	0.672	0.457	2000
Hatsu	0.371	0.516	0.431	2000
Chun	0.438	0.232	0.303	2000
accuracy	0.482	0.482	0.482	2000

Table A.1: Various classification metrics on FFO-B on the default testing set.

Class	Precision	Recall	F1-score	Support
1 man	0.702	0.637	0.668	2000
2 man	0.667	0.649	0.658	2000
3 man	0.740	0.653	0.694	2000
4 man	0.719	0.631	0.672	2000
5 man	0.653	0.644	0.649	2000
6 man	0.652	0.697	0.674	2000
7 man	0.711	0.636	0.672	2000
8 man	0.654	0.666	0.660	2000
9 man	0.666	0.658	0.662	2000
1 pin	0.669	0.639	0.654	2000
2 pin	0.679	0.644	0.661	2000
3 pin	0.673	0.667	0.670	2000
4 pin	0.657	0.709	0.682	2000
5 pin	0.699	0.624	0.659	2000
6 pin	0.678	0.697	0.687	2000
7 pin	0.704	0.671	0.687	2000
8 pin	0.701	0.645	0.672	2000
9 pin	0.641	0.721	0.678	2000
1 sou	0.623	0.710	0.664	2000
2 sou	0.689	0.663	0.676	2000
3 sou	0.685	0.646	0.665	2000
4 sou	0.656	0.712	0.683	2000
5 sou	0.626	0.690	0.657	2000
6 sou	0.686	0.653	0.669	2000
7 sou	0.670	0.689	0.679	2000
8 sou	0.695	0.626	0.659	2000
9 sou	0.729	0.627	0.674	2000
East	0.609	0.730	0.664	2000
South	0.630	0.694	0.660	2000
West	0.669	0.688	0.678	2000
North	0.636	0.720	0.675	2000
Haku	0.645	0.610	0.627	2000
Hatsu	0.628	0.676	0.651	2000
Chun	0.628	0.660	0.643	2000
accuracy	0.667	0.667	0.667	2000

Table A.2: Various classification metrics on MHA-B on the default testing set.

Class	Precision	Recall	F1-score	Support
1 man	0.683	0.626	0.653	2000
2 man	0.681	0.584	0.629	2000
3 man	0.731	0.605	0.662	2000
4 man	0.673	0.675	0.674	2000
5 man	0.673	0.595	0.631	2000
6 man	0.632	0.695	0.662	2000
7 man	0.683	0.646	0.664	2000
8 man	0.654	0.635	0.644	2000
9 man	0.649	0.651	0.650	2000
1 pin	0.612	0.690	0.649	2000
2 pin	0.639	0.635	0.637	2000
3 pin	0.633	0.677	0.654	2000
4 pin	0.646	0.685	0.665	2000
5 pin	0.662	0.646	0.654	2000
6 pin	0.683	0.665	0.674	2000
7 pin	0.684	0.651	0.667	2000
8 pin	0.685	0.628	0.655	2000
9 pin	0.644	0.653	0.648	2000
1 sou	0.629	0.689	0.657	2000
2 sou	0.629	0.665	0.647	2000
3 sou	0.667	0.660	0.663	2000
4 sou	0.645	0.693	0.668	2000
5 sou	0.643	0.655	0.649	2000
6 sou	0.669	0.643	0.656	2000
7 sou	0.686	0.650	0.667	2000
8 sou	0.635	0.647	0.641	2000
9 sou	0.654	0.686	0.670	2000
East	0.626	0.679	0.652	2000
South	0.656	0.586	0.619	2000
West	0.623	0.719	0.667	2000
North	0.651	0.664	0.658	2000
Haku	0.585	0.637	0.610	2000
Hatsu	0.634	0.602	0.618	2000
Chun	0.640	0.638	0.639	2000
accuracy	0.652	0.652	0.652	2000

Table A.3: Various classification metrics on MHA-S on the default testing set.

Appendix B

Confusion Matrices

See next page.

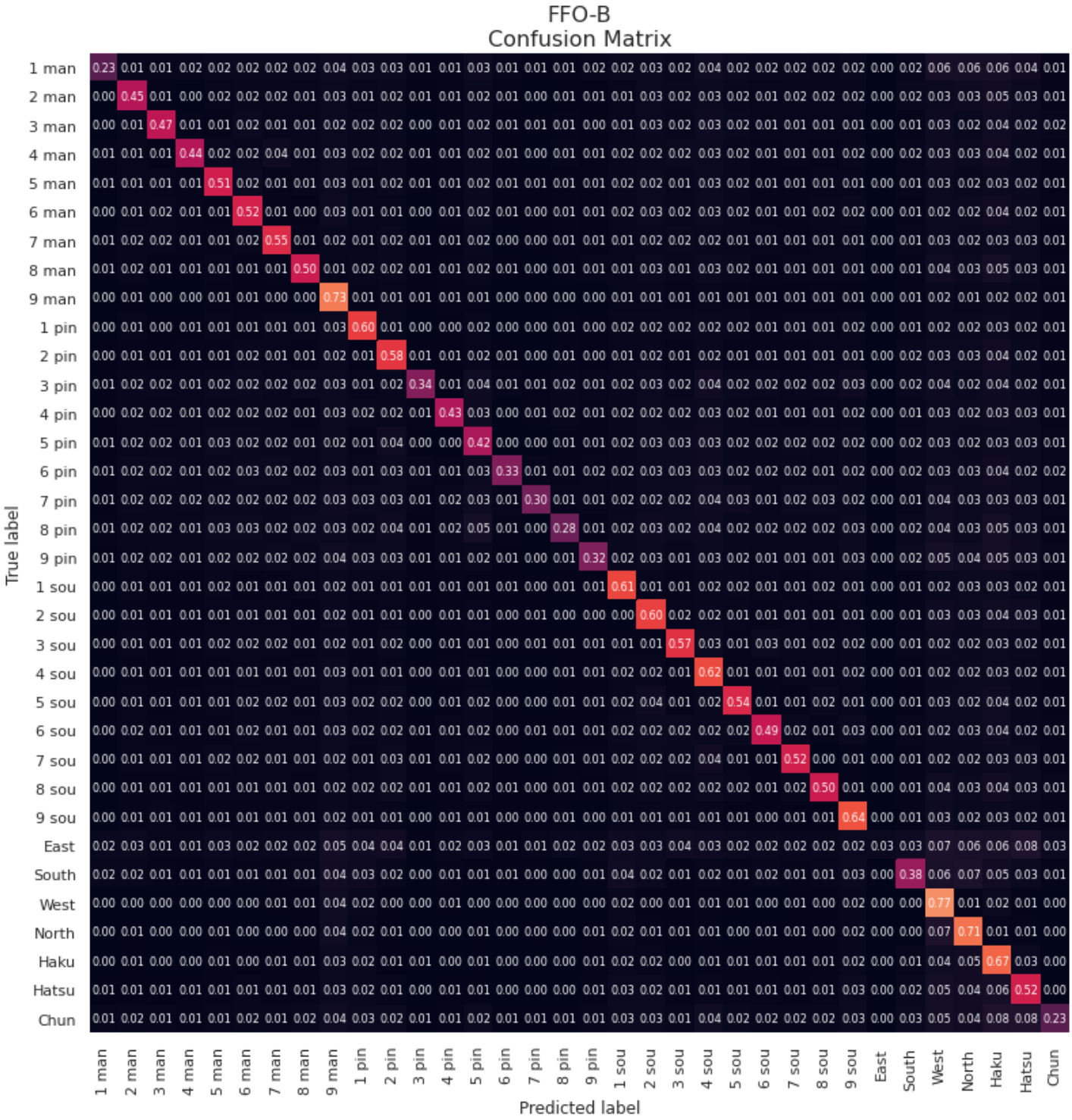


Figure B.1: FFO-B's confusion matrix on the default testing set.

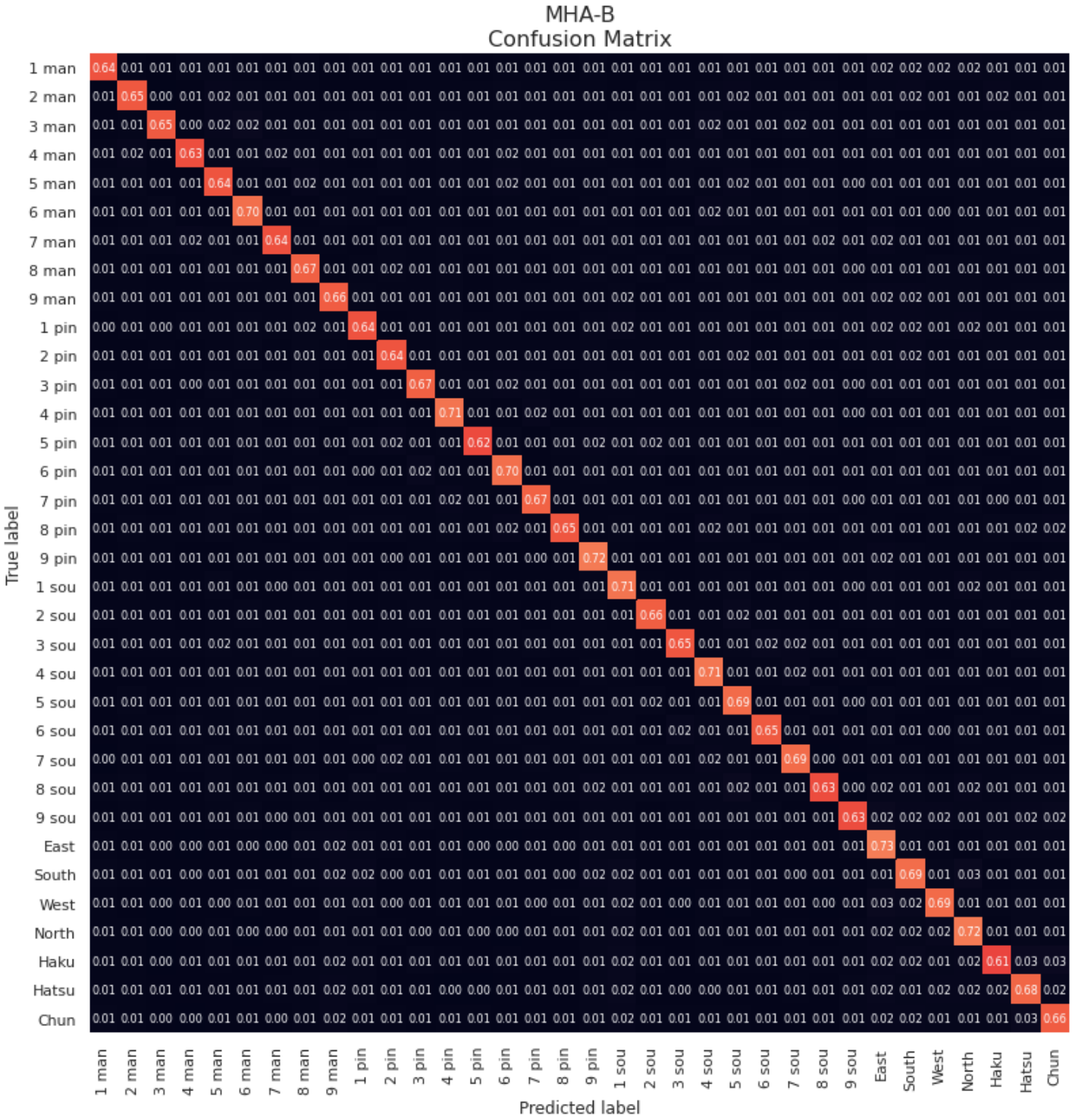


Figure B.2: MHA-B's confusion matrix on the default testing set.

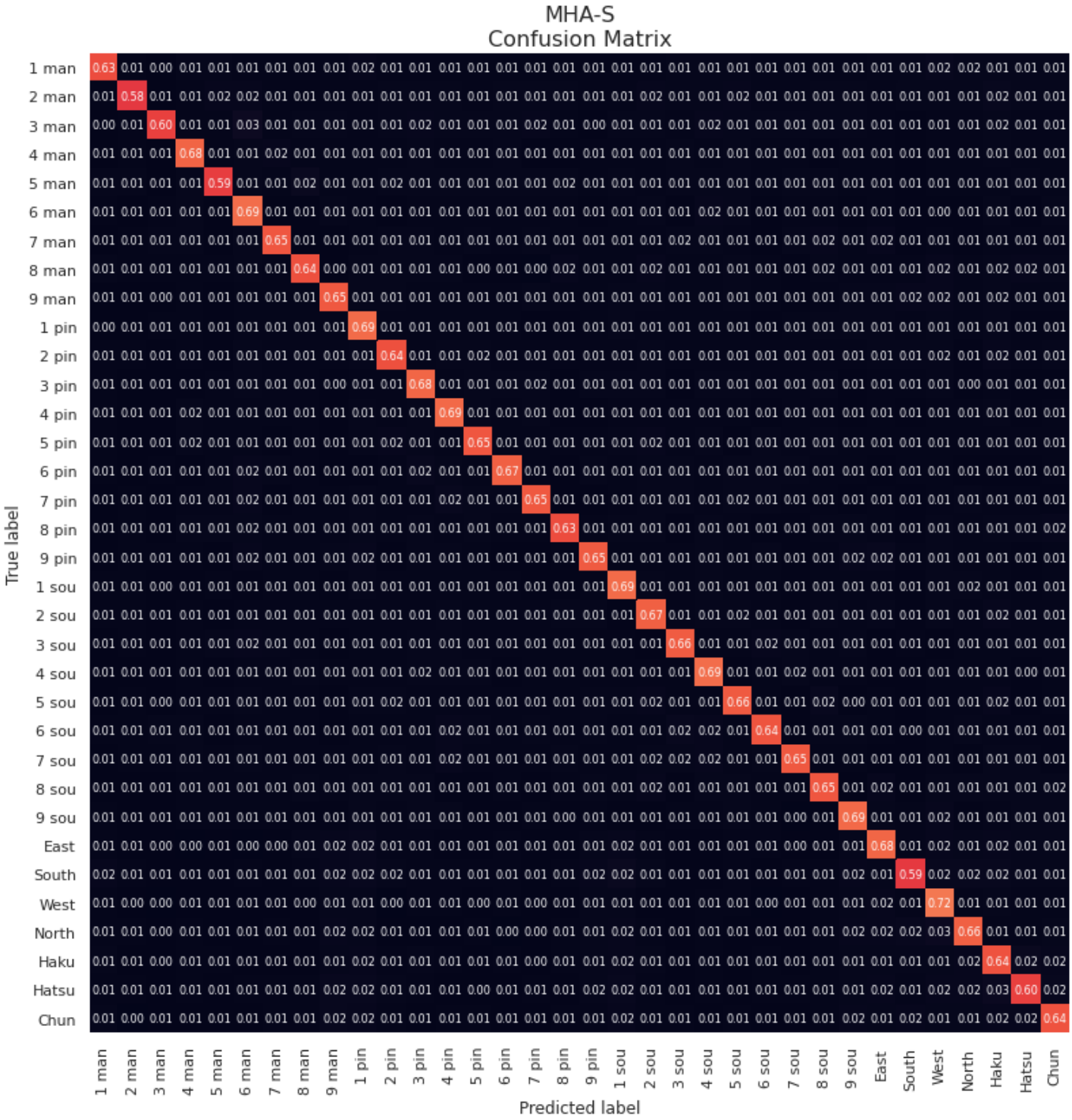


Figure B.3: MHA-S' confusion matrix on the default testing set.

Appendix C

Model Implementation

Code listing C.1: PyTorch implementation of the attention-based architecture

```
import torch

class AttentionNet(torch.nn.Module):
    """ Attention Layer into feed-forward net. """

    def __init__(self):
        super(AttentionNet, self).__init__()

        self.mha1 = torch.nn.MultiheadAttention(embed_dim=374,
                                                num_heads=1,
                                                dropout=0.0,
                                                add_zero_attn=False,
                                               )

        self.fc1 = torch.nn.Linear(11 * 34, 4096)
        self.fc2 = torch.nn.Linear(4096, 2048)
        self.fc3 = torch.nn.Linear(2048, 1024)
        self.fc4 = torch.nn.Linear(1024, 512)
        self.fc5 = torch.nn.Linear(512, 256)
        self.fc6 = torch.nn.Linear(256, 128)
        self.fc7 = torch.nn.Linear(128, 34)

        self.relu_1 = torch.nn.LeakyReLU()
        self.relu_2 = torch.nn.LeakyReLU()
        self.relu_3 = torch.nn.LeakyReLU()
        self.relu_4 = torch.nn.LeakyReLU()
        self.relu_5 = torch.nn.LeakyReLU()
        self.relu_6 = torch.nn.LeakyReLU()


    def forward(self, x):

        batch_size = x.shape[0]
        x = x.reshape(1, batch_size, 374)  #  => x.shape[0] = Batch Size
        attn_output, attn_output_weights = self.mha1(query=x, key=x, value=x, need_weights=False)
        x = (x * attn_output).reshape(batch_size, 374)

        x = self.fc1(x)
        x = self.relu_1(x)

        x = self.fc2(x)
        x = self.relu_2(x)

        x = self.fc3(x)
        x = self.relu_3(x)

        x = self.fc4(x)
```

```
        x = self.relu_4(x)

        x = self.fc5(x)
        x = self.relu_5(x)

        x = self.fc6(x)
        x = self.relu_6(x)

        x = self.fc7(x)

        return x


# Initialization
model = AttentionNet()
criterion = torch.nn.CrossEntropyLoss()  # Loss function
optimizer = torch.optim.SGD(model.parameters(), lr=0.001)
```