Department of Business and Management

Bachelor's Degree in Management and Computer Science

Convolutional Neural Networks and Tree Search Algorithms for Mastering the 2048 Game

Supervisor:

Prof. Alessio Martino

Candidate:

Davide Pisano

Role No. 286881

Academic Year 2024/2025

Abstract

The game 2048, despite its basic simplicity, poses significant hurdles for artificial intelligence, mostly because of its stochastic characteristics, vast state space, and complex demands for strategic planning. This thesis investigates the application of a deep Convolutional Neural Network (CNN) with extensive convolutional layers (1024 filters) as an advanced state evaluation function. The CNN evaluator was trained using a self-play methodology involving Deep Q-Learning combined with Experience Replay. To facilitate effective learning, reward shaping techniques were employed. The CNN evaluator was used along with search algorithms, such as: Beam Search and Expectimax. Moreover, a Hybrid Beam Search agent was developed, combining the CNN's learned evaluation capabilities with explicit heuristic strategies emphasizing empty cells, board monotonicity, smoothness, and cornering high-value tiles. Agent performance was evaluated based on metrics such as maximum tile achieved, final game score, and game duration. This research contributes meaningful insights into the potential of hybrid AI approaches for effectively navigating complex stochastic combinational challenges like 2048.

Contents

I Introduction

Artificial intelligence (AI) refers to the development of computer systems that can perform tasks typically requiring human intelligence, such as learning, problem-solving, and decision-making. One of the most compelling and accessible domains for exploring AI is through games, which provide structured environments with clear rules, objectives, and feedback —ideal conditions for training and testing intelligent behavior. Teaching AI to play games involves programming it to make decisions based on the game's state, often using techniques from machine learning and reinforcement learning. In this process, the AI agent learns strategies through trial and error, receiving rewards or penalties based on its actions. Over time, it improves by identifying patterns and optimizing its behavior to achieve better outcomes. Games like chess, Go, and video games have become benchmarks for AI research, showcasing how machines can master complex environments and even surpass human experts.

One of the earliest milestones was in 1997 when IBM's Deep Blue defeated world chess champion Garry Kasparov, marking the first time a reigning champion lost to a computer under standard tournament conditions. This victory demonstrated the potential of brute-force search algorithms combined with domain-specific knowledge and laid the groundwork for more sophisticated AI systems

Another significant milestone in AI game playing came in 2011 when IBM's Watson triumphed on the quiz show *Jeopardy!* against two of the game's greatest champions, Ken Jennings and Brad Rutter. Unlike games such as chess or Go, *Jeopardy!* posed a different kind of challenge: understanding and responding to complex natural language clues across a wide range of topics, often involving puns, riddles, and cultural references. Watson's success demonstrated major advances in natural language processing, information retrieval, and confidence-based decision-making, representing a leap forward in AI's ability to handle unstructured human knowledge and language.

This victory preceded another landmark moment, dated 2016, when DeepMind's AlphaGo defeated Go world champion Lee Sedol, a feat previously thought to be decades away due to the game's immense complexity and intuitive play. As will thoroughly discussed in

later chapters, AlphaGo used a mixture of deep neural networks and reinforcement learning. More recently, AI has conquered multiplayer and real-time games such as Dota 2 and StarCraft II, where OpenAI Five and DeepMind's AlphaStar showcased the ability of AI to operate in dynamic, partially observable environments.

While Deep Blue relied heavily on brute-force search, and AlphaGo used deep reinforcement learning, Watson highlighted how AI could parse and act on messy, real-world data. Together, these milestones —Deep Blue in chess, Watson in *Jeopardy!*, and AlphaGo in Go— trace a path of growing sophistication in AI systems, from rigid, rule-based strategies to adaptive, learning-based intelligence capable of navigating complex, uncertain environments.

This thesis is framed in the research area of AI playing games and the core objective is to use a mixture of Convolutional Neural Networks (CNN) and Deep Q-Learning to let an AI to learn how to play the game 2048. In the remaining part of this introductory chapter, we provide the reader an overview of the rules of the game 2048, alongside with a brief description of the major methodological aspects and the roadmap of the thesis.

I.1 The Game 2048

Gabriele Cirulli developed the game 2048 in 2014, and it is played on a 4×4 grid. Two tiles are randomly placed on the grid to begin the game. The tiles can be moved either vertically or horizontally by the players. When two identical tiles slide together, they form a single tile that shows the sum of their values. Each move results in the appearance of a new tile in an empty cell, tile 2 with 90% probability or tile 4 with 10% probability. Sliding tiles and combining them to reach 2048, is the game's goal. When there are no more movements or tile combinations available, the game is over (see Figure I for an example).

I.2 Overview of the Approach

The core challenge in developing a proficient 2048 agent lies in its ability to accurately assess the long-term potential of board configurations and to navigate the game's inherent stochas-

Figure I: An example of 2048 ended game state.

ticity. Our methodology addresses this by first constructing a sophisticated state evaluation function using deep learning, and then leveraging this learned function within established tree search paradigms to facilitate strategic lookahead and decision-making.

The foundation of our evaluative capability is a CNN, specifically architected to interpret the 4×4 grid of the 2048 game. This deep network, featuring a cascade of convolutional layers with a substantial initial filter capacity (1024 filters, progressively decreasing to 128), Batch Normalization, Layer Normalization, Pooling, and Fully Connected layers, was trained through extensive self-play. The training regimen employed principles from Deep Q-Learning, utilizing a large experience replay buffer to store and sample game transitions, and an ϵ-greedy exploration strategy to ensure a balance between exploiting known good states and discovering new ones. Crucially, to overcome the limitations of sparse game rewards, a significant reward shaping scheme was integrated into the training process, providing denser feedback signals to guide the network towards learning strategically sound patterns beyond immediate tile merges. The outcome of this training is a function, $V_\theta(s)$, which provides a learned estimate of the strategic value for any given board state s.

With this potent learned evaluator in hand, that is, $V_\theta(s)$, we then explored its application within three distinct search-based frameworks to plan moves:

Firstly, a CNN-guided Expectimax agent was developed. This approach directly confronts

the probabilistic nature of 2048's random tile spawns (90% chance for a '2' tile, 10% for a '4'). By recursively calculating expected values over these chance events and maximizing player choices up to a search depth of 5, this agent aims for robust decisions, using $V_\theta(s)$ to evaluate the terminal positions of its search.

Secondly, we implemented a CNN-guided Beam Search agent. Configured with a search depth of 5 and a beam width of 10, this method employs the CNN's evaluation, $V_\theta(s)$, to heuristically prune the vast search tree, retaining only a fixed number of the most promising board states at each level of lookahead.

Finally, a Hybrid Beam Search agent was constructed. This model builds upon the Beam Search by augmenting its evaluation mechanism during the search. It integrates the CNN's learned value, $V_\theta(s)$, with a weighted combination of handcrafted heuristics that explicitly reward desirable board characteristics such as ample empty cells, monotonic tile arrangements, board smoothness, and corner placement of the highest-value tile.

This structured progression,from learning a deep state evaluator to deploying it within different search algorithms, including a hybrid model, allows for a comprehensive investigation into the synergies between deep learning and classical search techniques for mastering the complex, stochastic environment of 2048.

I.3 Thesis Outline

This thesis is presented in the following order: the discussion of methods is first, with the findings and broader implications discussed last. Chapter II, "Related Work and Background," situates this thesis in context by summarizing previous 2048 research, including building on the theoretical work of Slizkov (2023), and presents the AlphaZero algorithm as a key benchmark to which we can compare our methods.

Chapter III, "Methods," describes our core algorithmic components in detail. These components include the architecture of and Deep Q-Learning based training of our Convolutional Neural Network (CNN) evaluator that incorporates Experience Replay and carefully engineered Reward Shaping. Then, we describe the principles and our implementations of the

CNN-guided Expectimax algorithm, the CNN-guided Beam Search algorithm, and the hybrid method of CNN-guided Beam Search.

Chapter IV, "Experimental Results and Analysis," presents and analyzes the empirical performance of our developed agents. This chapter describes the experimental setting and evaluation metrics and compares agents on the 2048 performance metrics of game scores, maximum tile achieved, game progression behaviors, and computational run-time efficiency. Each section includes summary statistics and visualization techniques that provide evidence to identify effective architectures and methods or strategies.

Ultimately, Chapter V, "Conclusions and Future Work," wraps together our key contributions and findings. We summarize the comparative performance of agents, discuss and reflect on implications of our research for the field of AI in games and hybrid model design, discuss limitations of study, and provide suggestions for future research, including future work to extend the training of the CNN evaluator, as well as other methodologies

II Related Works and Background

The game of 2048, since its release, has attracted considerable attention from both players and researchers in artificial intelligence. Its deceptive simplicity, blending deterministic player moves with stochastic tile spawns, presents a compelling testbed for various algorithmic strategies. Early efforts often focused on heuristic-based approaches, attempting to encode human intuition about good board configurations, such as maintaining monotonicity, smoothness, and maximizing empty cells. These heuristics were frequently coupled with search algorithms like Minimax or Expectimax to look several moves ahead.

More recently, research has also explored the theoretical limits and computational aspects of 2048. For instance, Slizkov (2023) (5) in "Computational bounds for the 2048 game" rigorously investigates the game's solvability and win probabilities. This work delves into the vast state space of 2048, which, even for a goal tile like 256 on a 4×4 grid, involves an enormous number of positions, that is, $2^{8 \cdot 16}$. Slizkov's approach involves traversing a layered game graph, where layers are defined by the sum of tile values on the board. By analyzing these layers and employing significant computational resources, the paper proves a lower bound for the maximum achievable tile (256 is guaranteed with probability 1) and establishes a high winning probability (at least 0.99969) for reaching the 2048 tile under an optimal strategy. This research highlights the sheer scale of the 2048 problem and the computational intensity required for formal proofs of reachability and optimality, even with state-space reductions and advanced indexing techniques for managing the game graph layers. Such work underscores the need for efficient approximation methods, when perfect solutions are computationally intractable for real-time play or broader strategic learning. Beyond theoretical bounds, machine learning, particularly reinforcement learning (RL), has been a popular avenue for developing high-performing 2048 agents. Many of these approaches aim to learn an evaluation function or a direct policy for playing the game. The challenges in applying RL to 2048 mirror those in more complex games like Chess, Go, and Shogi: accurately evaluating board states, dealing with sparse rewards (especially for reaching high tiles), and efficiently exploring the state

space.

Solving complex games like Chess, Go and Shogi, has historically been a great challenge for artificial intelligence. The main issues include developing accurate evaluation functions to guide the search effectively, overcoming the problem of inherent long games with delayed rewards, balancing exploration and exploitation, managing high branching factors arising from either player moves or stochastic game elements. Algorithms must explore just a fraction of the potential game tree, often relying on sophisticated evaluation mechanism and search strategies.

One of the most important and new approaches has been the AlphaZero (3) algorithm, the successor to AlphaGo Zero, the first one being a generic version of AlphaGo. It replaced domain-specific augmentations and hand crafted knowledge of the game used in traditional game-playing programs, with a tabula-rasa approach. This new approach consists of providing the neural network with just the rules of the game, and it aims at demonstrating that a general-purpose reinforcement-learning algorithm can achieve superhuman performances.

AlphaZero uses a deep neural network

$$(p, v) = f_\theta(s) \tag{1}$$

with parameters θ, takes as input the board positions s and outputs a vector of move probabilities p with components $p_a = \Pr(a|s)$ for each action a, and a scalar value v estimating the expected outcome z from position s: $v \approx \mathbb{E}[z|s]$. These move probabilities are learned entirely from self-play and are then used to guide the search.

The search is performed using a Monte Carlo tree search algorithm (MCTS). Each step consists of a series of simulated games based on the board position, that then traverse a tree from root s to a leaf (the move to do). Each simulation proceeds by selecting in each state s a move a with low visit count, high move probability, and high value. The search then returns

a vector π representing a probability distribution over moves.

The parameter θ is trained on self-play reinforcement learning, starting from a randomly selected parameter. The games are played by selecting the moves from the MCTS for both players, the terminal position s_t is scored according to the rules of the game and in case of a win $z = +1$, in case of a tie $z = 0$, in case of a loss $z = -1$.

The parameters are adjusted by gradient descent on a loss function L that sums over the mean-squared error and cross-entropy losses, respectively. The search is focused on promising variations both by extending the search depth of promising variations and by reducing the search depth of unpromising variations based on heuristics like history and moving piece type.

AlphaZero evaluates positions using non-linear function approximation based on a deep neural network, rather than linear function approximation used in other chess/go/shogi programs. This provides a more powerful representation, and introduced the need of averaging across these errors that were then canceled out during the search. Using MCTS allowed AlphaZero to effectively combine its neural network representations with a powerful domain-independent search.

While 2048 has a smaller board and simpler move mechanics compared to Chess, Go and Shogi, it introduces on its own, significant challenges that reflect those in classic games. The random appearance of new tiles, either 2 or 4, adds a layer of stochasticity not present in other deterministic games, requiring agents to plan moves ahead. Achieving higher-values tiles requires a precise sequence of merges, making the later stages far more difficult. Finally, the ultimate goal of achieving the 2048 tile, represents a very sparse reward, making it difficult for standard reinforcement learning to learn necessary long-term strategies without additional guidance.

The approach we followed shares fundamental principles with the AlphaZero framework, while still having key differences tailored specifically for such game. Both approaches leverage the core idea of combining a powerful learned neural network for state evaluation, with a search algorithm. We both employ self-play reinforcement learning, enabling the agent to learn complex strategies from experience without the need to rely on human expert data. We both employ non-linear function approximators for evaluating game states, that then guide the search process, focusing on the computation of specific lines of play.
Despite the existence of some similarities, some important differences occur. Firstly, regarding the neural network output, AlphaZero's framework predicts both the move probabilities p and a state value v, whereas our Convolutional Neural Network is designed and trained as a state value function

$$V_\theta(s) \approx \mathbb{E}_{\pi,P}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right] \tag{2}$$

which estimates the expected total discounted future reward from state s. The network parameters θ are updated by minimizing the Mean Squared Error (MSE) between the predicted value V_θ and the target value y over batches sampled from the replay buffer:

$$L(\theta) = \mathbb{E}_{(s,r,s',d)\sim\mathcal{B}}\left[(y - V_\theta(s))^2\right] \tag{3}$$

the target value calculation for a sampled transition (s, r, s', d), $y = r + \gamma(1-d)V_{\theta'}(s')$, where $\gamma = 0.95$. Move selection in our system is done via a comparison carried out by the search algorithm that compares the CNN-evaluated values of subsequent states. Secondly, the search algorithms: AlphaZero employs Monte Carlo Tree Search (MCTS), while we utilize: Expectimax and Beam Search.

Another significant difference is the incorporation of the domain knowledge, as said before, AlphaZero adopts a "tabula-rasa" approach. We utilize domain insights via two mechanisms: extensive reward shaping during the CNN's training (milestones, corner bonuses , etc.) to accelerate the learning and address sparse rewards, and the explicit combination of the CNN

evaluation with weighted, handcrafted heuristics during search in the Hybrid Beam Search agent. Additionally, the reward signal used for training differs significantly. While AlphaZero relies on a simple terminal reward z based on the final game outcome (win/tie/loss) : $\in \{+1, 0, -1\}$, our CNN training employed a more dense, intermediate reward shaping strategy. The reward r_t used to compute the target value y involves multiple components beyond basic tile merge score, such as: achieving milestone tiles, a bonus for positioning highest tile in a corner, a progress-based reward, that aimes at reducing the time needed for the NN at learning specific patterns and strategies, by providing a more frequent feedback to the agent, reducing the challenge posed by the sparse ultimate reward initially faced.

Lastly, the training objectives differ: AlphaZero minimizes a combined policy and value loss based on MCTS outcomes, while our CNN is trained using a Deep Q-Learning-style with Experience Replay, minimizing temporal difference errors.

III Methods

To address the problem of mastering the 2048 game, we adopted a multipart approach, centered around the development of deep-learning state evaluation function then leveraged by tree search algorithms. This section details the core components of our methodology, the architecture and training blueprint of the Convolutional Neural Network (CNN) designed to learn the value of game states, then the implementations of the Expectimax and Beam Search algorithms that uses this learned knowledge for strategic move planning. Finally, we tried a hybrid approach, giving more weight to handcrafted heuristics function withing the Beam Search framework. The goal was to create agent that are capable of deep lookahead and robust decision-making.

III.1 Convolutional Neural Network

At the heart of our approach lies a Convolutional Neural Network (CNN) (4) , a class of deep neural networks particularly adept at processing grid-like data, making them a natural choice for interpreting the 4×4 board of the 2048 game. The fundamental idea was to train this CNN to serve as a sophisticated evaluation function, capable of looking at a given board configuration and outputting a scalar value that represents its strategic worth or potential for leading to a high score.

The architecture of our CNN was carefully designed to capture the spatial hierarchies inherent in 2048 board states. Recognizing that the value of a tile is not just its magnitude but also its position relative to other tiles and empty spaces, we chose a deep architecture. The input to the network is not simply the raw tile values. Instead, each of the 16 cells on the 4×4 board is transformed into a 16-channel one-hot encoded vector. Each channel corresponds to a power of 2, from 2^0 (representing an empty cell), up to 2^{15}. If a cell contains the tile '8' 2^3, the third channel for that cell's vector would be active (1), and all other zero. This representation provides a distinct and unambiguous input for each possible tile value up to

32768, allowing the network to differentiate clearly between tile magnitudes without being biased by the numerical scale.

The network itself comprises a sequence of convolutional blocks. Early blocks feature 3×3 convolutional layers with a large number of filters, designed to learn a rich set of low-level spatial patterns from the input. These are followed by Batch Normalization layers, to stabilize activations and accelerate training. Max Pooling layers are then used to reduce the spatial dimensions of the feature maps, which helps the network build more abstract, higher-level representations and achieve a degree of translation invariance for the learned features. As the data flows deeper into the network, the number of filters in convolutional layers progressively decreases (to 512, then 256, then 128), while the features become more complex. A later stage of the network incorporates 1×1 convolutional layers, which act as channel-wise transformations, followed by Layer Normalization, offering an alternative normalization strategy well-suited for this stage.

Finally, after the convolutional and pooling layers have processed the spatial information down to a compact representation, where the featureas are flattened and passed through several Fully Connected Layers. These layers perform the final mapping from the high-level features to a single scalar output. This output $V_\theta(s)$ represents the network's learned estimation of the value of the input board state s, given the network parameter θ. Dropout layers are interspersed throughout the network to act as a regularizer, preventing overfitting to the training data.

CNN was trained using a self-play methodology, guided by principles adapted from Deep Q-Learning (DQN). The network learns the state-value function through interaction with the game environment. Key to this process is Experience Replay. As the agent plays games, starting from the initial state s_t, the shaped reward r_t received, the resulting next state s_{t+1} and flag a d_t indicating if s_{t+1} is actually a terminal state. For training, mini batches of these experiences are randomly selected. This sampling, break the temporal correlation leading

to a more stable and robust learning. A target network θ', is used to generate stable target values for the learning updates. The parameters θ' are updated by copying the parameters θ from the main network that is actively trained. The target value y_t for a sampled transition is calculated using the Bellman Equation:

$$y_t = r_t + \gamma(1 - d_t)V_\theta(s_{t+1}) \tag{4}$$

where γ is a discount factor set to 0.95 that prioritizes more imeediate rewards. The network's parameter θ are updated by minimizing the Mean Squared Error (MSE) loss between its prediction $V_\theta(s_t)$ and the calculate target

$$L(\theta) = \mathbb{E}_{(s_t, r_t, s_{t+1}, d_t) \sim \mathcal{B}} \left[(y_t - V_\theta(s_t))^2 \right], \text{ where } y_t = r_t + \gamma(1 - d_t)V_\theta(s_{t+1}) \tag{5}$$

The Adam optimizer was employed for this minimization. During the self-play training phase, an ϵ-greedy exploration strategy was implemented. This strategy dictates that the agent chooses a random valid action with a small probability ϵ, and otherwise selects the action deemed best according to its current learned value function (by evaluating potential next states). The value of ϵ was decayed over the course of training, gradually shifting the agent from an explorative mode to a more exploitative one as its value function improved.

A crucial element of our training methodology was Reward Shaping. The natural reward in 2048 (the sum of merged tiles in a move) is relatively frequent but may not provide sufficient signal for learning the long-term strategies needed to achieve high tiles. The ultimate goal of reaching the 2048 tile, for example, is a very sparse event. To provide denser and more informative feedback, the reward r_t given to the agent was augmented. Beyond the immediate move score, this shaped reward included: significant bonuses for achieving milestone tiles (512, 1024), a progress-based reward for forming a new highest tile on the board, a small continuous bonus for maintaining high-value tiles, and an incentive for positioning the highest tile in one of the board's corners. This multifaceted reward signal was designed to

guide the network towards learning configurations and move sequences that are strategically valuable in the long run.

III.2 Expectimax

Having established a method for the Convolutional Neural Network to assign a quantitative measure of value to any given board state, we then sought to leverage this learned evaluation within a structured search framework. For a game like 2048, characterized by player decisions interspersed with probabilistic environmental events (the random tile spawns), the Expectimax (2) algorithm presents a principled approach to decision-making. Expectimax can be understood as an adaptation of the Minimax algorithm, specifically tailored for scenarios involving chance. The central tenet of Expectimax is to select actions that maximize the expected outcome, meticulously accounting for the probabilities associated with all potential random events.

The operational mechanism of the Expectimax algorithm involves the construction and traversal of a game tree originating from the current board configuration. This tree inherently alternates between two distinct types of decision points, or nodes. The first type, often termed Max nodes, corresponds to junctures where the agent (our player) must select a move. At these points, the algorithm explores all valid moves available to the agent. For each potential move, it anticipates the subsequent probabilistic response from the game environment and calculates an expected value for the state that would result. The algorithm's choice at a Max node is then to select the move that yields the highest of these calculated expected values.

Following a player's move, the game transitions to what is effectively the environment's turn, represented in the Expectimax tree by Chance nodes. It is at these nodes that the stochastic nature of 2048 comes into play. After a player's swipe, the game introduces a new tile—either a '2' (with a 90% probability) or a '4' (with a 10% probability)—into one of the randomly chosen empty cells on the board. The value attributed to a chance node is not a maximum,

but a weighted average. This average is computed over all possible outcomes stemming from the random tile spawn, where each outcome's value is weighted by its specific probability of occurrence. This involves considering every empty cell as a potential spawn location and both possible new tile values ('2' or '4').

This recursive process of evaluating Max nodes and Chance nodes continues down the game tree until a predefined search depth is reached. In our implementation, this depth was set to 5. When a leaf node of this search is reached -it means that we are in two possible scenarios, a maximum depth has been attained, or the terminal state, game over, has been encountered- the trained CNN, $V_\theta(s)$ is called. The CNN provides a static evaluation, a numerical score, for that leaf board state. This evaluated score is then propagated back up the tree, contributing to the expected value calculations at parent Chance nodes and the maximization choices at parent Max nodes.

Ultimately, the action selected by the Expectimax agent for the current turn is the one at the very root of the search tree that initiated the path leading to the highest overall expected value. The decision to employ Expectimax was primarily driven by its inherent ability to formally and directly address the stochastic tile spawns that are a defining characteristic of 2048. By systematically averaging over all probabilistic tile placements and values, Expectimax endeavors to identify moves that are robustly advantageous, rather than those that might appear optimal under a fortunate sequence of random events but perform poorly otherwise. To enhance the practical efficiency of our Expectimax implementation, particularly given the computational expense of repeated CNN evaluations, a transposition table was utilized. This mechanism caches the evaluated values of previously encountered board states, thereby preventing redundant computations when identical sub-problems arise at different points in the search tree. Furthermore, techniques for batching state evaluations were implemented to better leverage GPU parallelism when invoking the CNN for multiple leaf node evaluations.

III.3 Beam Search

While Expectimax provides a thorough method for navigating stochastic game trees, its computational demands can escalate rapidly with increasing search depth, primarily due to the expansive branching factor introduced at each Chance node. As an alternative strategy, aimed at achieving deeper lookahead within practical time constraints, we implemented a Beam Search algorithm. Beam Search is a heuristic search technique that navigates the game tree by selectively exploring only a limited, fixed number of the most promising candidate states at each level of depth. This approach prunes the search space more aggressively than Expectimax, trading completeness for speed.

The Beam Search process commences (1) with the current game state forming the initial "beam." At each subsequent step, or depth level, of the search, the algorithm expands from the states currently held within the beam. For every state in this beam, all possible successor states are generated. In the context of 2048, a successor state is the configuration resulting from the player executing one valid move, immediately followed by the game engine introducing a new random tile onto the board. Each of these newly generated successor states must then be evaluated to determine its promise.

In our standard CNN-Beam Search implementation, the score assigned to a successor state s'' is a composite of the accumulated reward (the path score) obtained in reaching s'' from the initial state, augmented by the value $V_\theta(s'')$ assigned to s'' by our trained Convolutional Neural Network.

Once all potential successors at the current depth have been generated and scored, a selection process occurs. From this entire pool of candidates, only the top k states—where k is a predefined parameter known as the `beam_width`—with the highest evaluation scores are chosen to constitute the beam for the subsequent depth level. All other candidate states generated at this step are discarded and not explored further. This iterative process of expansion, evaluation, and selection is repeated until a specified `search_depth` is attained. The action ultimately chosen by the Beam Search agent for the current turn is the very first move in the sequence of actions that led to the state possessing the highest evaluation score within

the beam at the final depth. For our experimental setup, the Beam Search was configured with a search depth of 5 and a beam width of 10.

The rationale for selecting Beam Search lies in its pragmatic approach to managing computational resources. By strictly limiting the number of paths explored at each depth, it can often probe deeper into the game tree than a full Expectimax search might within the same allotted computation time. However, the efficacy of Beam Search is intrinsically linked to the quality of its evaluation function—in our case, the CNN. The search relies heavily on the CNN's ability to accurately discern and prioritize genuinely promising board configurations, ensuring that high-potential paths are likely to be retained within the limited width of the beam.

Recognizing that even a proficiently trained CNN might not encapsulate every nuance of optimal 2048 strategy, or that its learned insights could be fruitfully complemented by established game-playing principles, we also developed a Hybrid CNN-Beam Search agent. This variant introduces a modification to the evaluation function used to score and select states within the beam search procedure. Instead of relying exclusively on the combination of the path score and the direct output of the CNN, $V_\theta(s)$, the Hybrid agent employs a more composite score. This score is a weighted linear combination of the CNN's predicted value and the outputs of several handcrafted heuristic functions.

These heuristic functions were specifically designed to quantify strategically desirable board characteristics, such as: a bonus for a greater number of empty cells (as more empty cells generally afford greater flexibility), a bonus for board monotonicity (rewarding rows and columns where tile values are arranged in a consistently increasing or decreasing order, which facilitates merging), a bonus for board smoothness (penalizing large, abrupt differences in value between adjacent tiles, as these can hinder merges), and a significant bonus for positioning the highest-value tile in one of the board's corners (a common and effective human strategy). The relative influence of the CNN's evaluation versus these explicit heuristics in the composite score was governed by adjustable weights. The underlying motivation for this hybrid approach was to investigate whether the explicit injection of these established strategic principles could further refine and enhance the performance achieved by the learned

CNN evaluator operating within the Beam Search framework, potentially leading to more consistently robust or higher-scoring gameplay by ensuring that fundamental good board properties were always actively considered in conjunction with the network's more nuanced, learned understanding of the game.

IV Experimental Results and Analysis

The following section will illustrate the performance of our agents: Beam Search (Depth 4, Width 10), Beam Search (Depth 5, Width 10), Expectimax (Depth 5), Hybrid Beam Search (Depth 5, Width 10), they were all evaluated over a series of 2048 game episodes. The metrics used are final game score, maximum tile achieved, number of steps per game, and computation time. A summary of the aggregate performance statistics is presented in Table I.

Table I: Comparison of Different Algorithms for 2048 Game

Algorithm	**Mean Score**	**Max Score**	**Mean Steps**	**Success Rate**	**2048 Tiles**	**1024 Tiles**	**Avg. Time (s)**
Beam Search ($d = 4$)	11,769.4 $\pm$ 4,065.2	16,432	697.1 $\pm$ 237.6	0%	0	20	117.4
Beam Search ($d = 5$)	15,880.7 $\pm$ 6,365.0	35,820	906.7 $\pm$ 293.1	16%	16	84	190.9
Expectimax ($d = 5$)	11,730.0 $\pm$ 5,456.5	26,032	704.2 $\pm$ 242.0	8%	8	72	523.4
Hybrid Beam Search	6,972.7 $\pm$ 3,250.3	15,556	462.1 $\pm$ 167.8	0%	0	28	342.4

Note: Success Rate represents the percentage of games where a 2048 tile was achieved. Paramter $d = \{4, 5\}$ refers to the search depth. Scores and steps are shown as average $\pm$ standard deviation.
Training setup: The CNN has been trained on a virtual machine

All training for the CNN and the subsequent search-based agent were conducted on a consistent high-performance computing environment. Specifically, these experiments utilized a virtual machine equipped with an NVIDIA H100 SXM GPU featuring 80GB of GPU RAM, complemented by an Intel Xeon Platinum 8480C series CPU. The software environment was standardized with Python version 3.10.10 and PyTorch version 2.0.0.

The final game scores (Figure II) serve a primary indicator of an agent's ability to make consistently effective moves and build high value board configurations. An initial examination of the mean scores, reveals that the CNN-Beam Search (Depth 5) agent stands out, achieving an average score of approximately 15,803. This substantially surpasses the standard Beam Search (Depth 4) which approximately achieves a score of 12,713 and the Expectimax with a score of 11,602, suggesting that the combination of deeper search along with beam heuristic provided a significant strategic advantage. The Hybrid Beam Search registered the lowest mean score of approximately 6,973: a point we will discuss later.

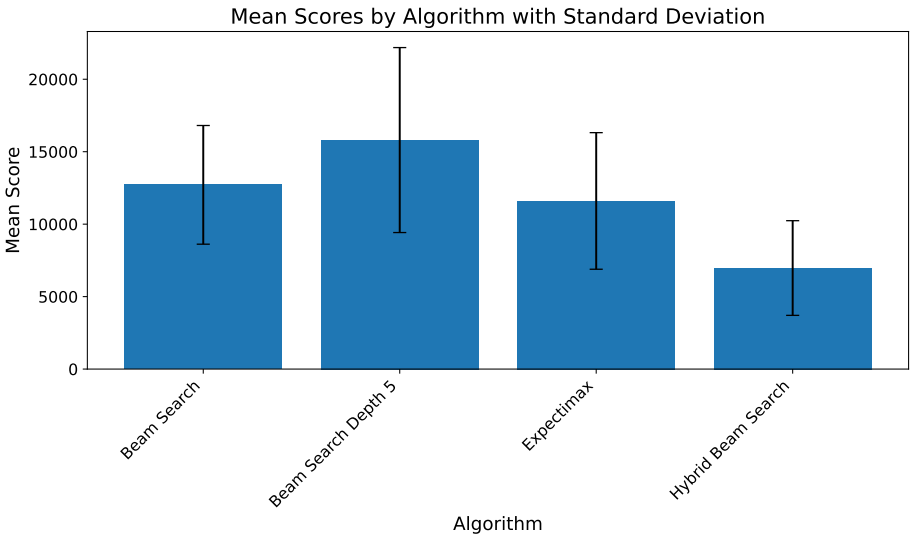

Figure II: Comparison of mean scores.

The most direct measure of success to address the ability of the ability to create high-value tiles is the percentage of games reaching the 2048 tiles over the sample set, Figure III clearly distinguishes the Beam-Search (Depth 5) as the most proficient. It synthesized the 2048 tile in approximately 18.2% of it evaluated episodes. This success rate, while not only approaching certainty, is a significant achievement given the game's complexity and stochasticity. Expectimax agent also proved capable of reaching this milestone, doing so in approximately 5.1% of its games. This demonstrates its probabilistic lookahead, coupled with the CNN evaluator, it can navigate the board to the 2048 tile. Crucially neither the standard Beam-Search (Depth 4) nor the Hybrid Beam-Search achieved the 2048 tile in any of the recorded episodes. This starkly indicates that either the shallower reduced search depth or the specific heuristic combination presented a fundamental barrier to reaching the game's primary objective within this set of trials.

When looking at the overall tile progression (Figure IV), it is interesting to see that the CNN-Beam Search (Depth 5) agent not only produced the 2048 tile, but also reliably produced the 1024 tile in a significant portion of its games (approximately 84.8%). This implies there was

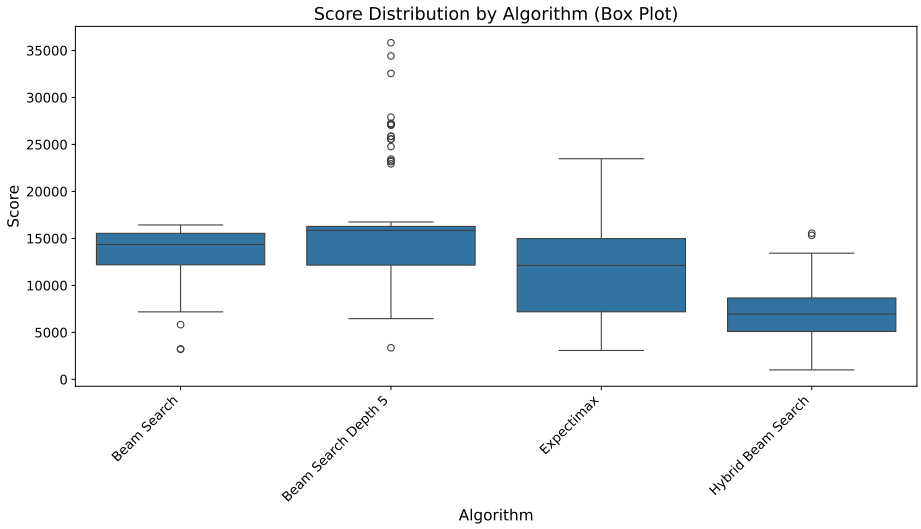

Figure III: Distribution of scores.

a consistent ability to progress through the mid-to-late stages of the game. The Expectimax agent, while it produced the 2048 tile getting there less frequently, produced 1024 in most cases (approximately 62.8% of games), indicating the search is generally effective in creating valuable tile states, if not always a provable optimal path to 2048. The standard Beam Search (Depth 4) agent generally played to the 1024 tile (80% of its games), which indicates that the rooted depth of the beam search was the most significant limitation to obtaining larger tile values, but of course may also be due to competence as a relative player. The Hybrid Beam Search agent often produced 512 tile (73% of games) and 1024 along less often (19% of games), meaning this agent likely performed marginally worse and less reliably than the standard. The finding could also imply that the intended mix of heuristics may push players towards certain controlling tile configuration strategies, which are locally beneficial to the heuristics, but more broadly do not enable progression to towards the high-value tiles critical for reaching 2048. For instance, it is plausible that the heuristics provide higher priority to honest, consistent board state which are lower order based on tile value for tiles such as 2048, while creating less useful long-term merge flurry or extreme value tiles; mere innovations to create tiles without higher order significance.

To gain deeper insights into the performance dynamics of the agents, we examined the

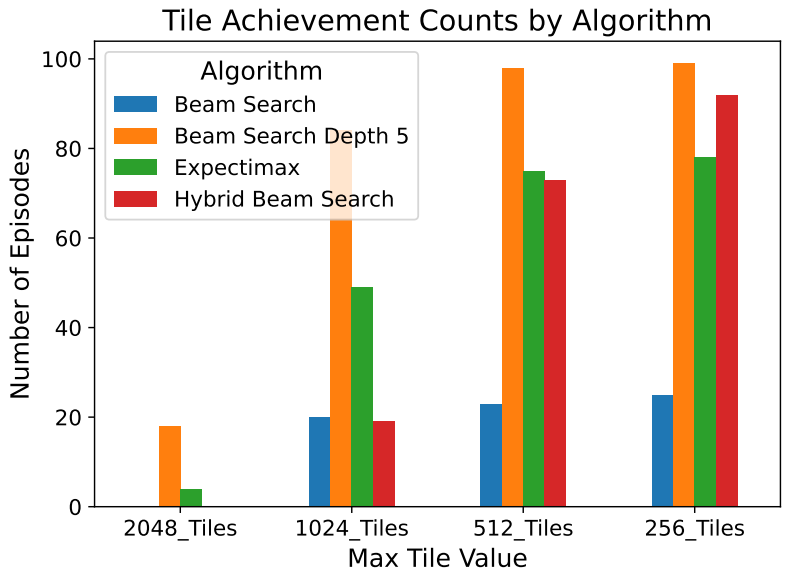

Figure IV: Tile counts.

relationship between their final scores and two critical indicators of game progression: the highest tile achieved during the game and the total number of moves made. Figure V provides a detailed breakdown of final score distributions conditioned on the maximum tile an agent managed to create. A consistent trend observed across all algorithms is a strong positive correlation: games that result in higher maximum tiles generally produce correspondingly higher final scores. For example, when the Beam Search (Depth 5) or Expectimax agents reached the 2048 tile, their scores predominantly fell within a significantly higher range, typically above 22,000, whereas games capped at the 1024 tile usually resulted in scores between 10,000 and 17,000. This correlation appears logical, as achieving higher tiles necessitates greater merges, which directly boost the overall score.

Despite this general pattern, the results also reveal subtle variations in scoring efficiency for a given maximum tile. Taking the 1024 tile as an example, the Beam Search (Depth 5) not only reached this milestone more consistently but also demonstrated a tighter and higher median score distribution when compared to Beam Search (Depth 4) or Expectimax. This suggests that the deeper Beam Search strategy was not just more proficient at achieving this tile but also tended to do so on boards that accumulated higher scores through efficient

intermediate merges. On the other hand, while Expectimax was effective at reaching the 1024 tile and earned solid scores, its score distribution for this tile exhibited a wider spread, indicating greater variability in the quality of the board configurations that led to forming the tile.

The Hybrid Beam Search, however, displayed a different scoring pattern. When it reached the 1024 tile, its scores tended to fall at the lower end of the spectrum compared to standard Beam Search agents, potentially reflecting less optimal strategies that prioritized heuristic objectives over total score accumulation, potentially at the cost of missing intermediate merges. For lower tiles such as 256 or 512, score distributions across all agents were more compressed, as might be expected. Nevertheless, even within these ranges, the Hybrid agent generally occupied lower positions within the score spectrum, pointing to less efficient paths or strategies across various gameplay scenarios.

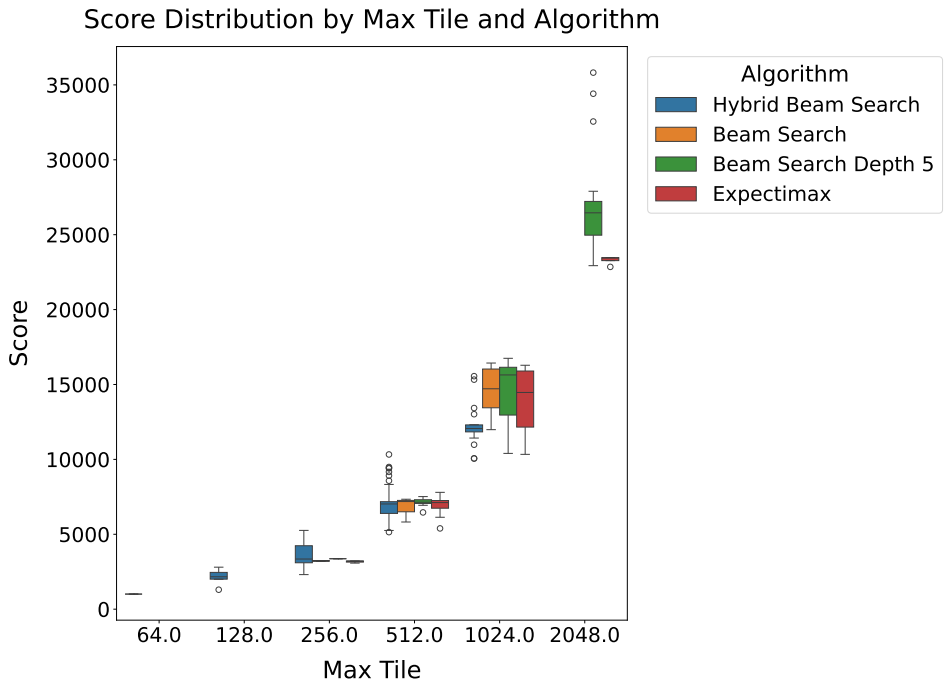

Figure V: Max Tile by Score.

The connection between the number of steps taken in a game and the ensuing final score,

as seen in Figure VI demonstrates important features of both durability and scoring, i.e. efficiency per step. Across all algorithms, a consistently positive linear correlation can be seen, indicating that a greater number of steps generally means a better score. This inherently stems from an increased number of chances to move strategically and align tiles in a longer game.

Nevertheless, the intensity and slope of these relationships differ between agents. For example, the Beam Search (Depth 5) has the furthest reach along both axes, especially evident for its highest scores and successful 2048 tile plays. This suggests that this agent not only has greater durability, yet also possesses a commendable score efficiency. Similar to the Expectimax agent has a modest slope for its successful 2048 runs. In contrast, the standard Beam Search (Depth 4) is ever so slightly more peaked, as indicated by its earlier game terminations and how its points are clustered in a more moderate step and score range.

The Hybrid Beam Search has the lowest average game time when looking at its position in the graph, and has its points clustered in the lower left, indicating that most of its games had earlier endings with lower scores. This section of the graph implies that either the Hybrid agent reached or generated unrecoverable board states earlier, or, was using a heuristic-driven approach focused on locally optimal moves. While this way of playing was successful in the moment, it may not have prioritized game's sustainability, and in doing so, resulted in many premature game endings.

With that said, each agent is characterized by tightly clustered points along its respective trendline, indicating a relatively similar score-per-step rate across their main gameplay duration.

In Figure VII, we see a clear separation in computational overhead between agents. The Expectimax agent was easily the most time-consuming with an average per episode rate of about 521 seconds. The large time duration is a function of its algorithmic nature - at each chance node, Expectimax evaluates the expected value by considering every possible random tile placement in every empty cell, and for both possible new tile values ('2' and '4'). Expectimax was therefore evaluating a significant amount of the probabilistic outcomes possible, particularly when considering a depth of 5, leading to a significant number of states explored

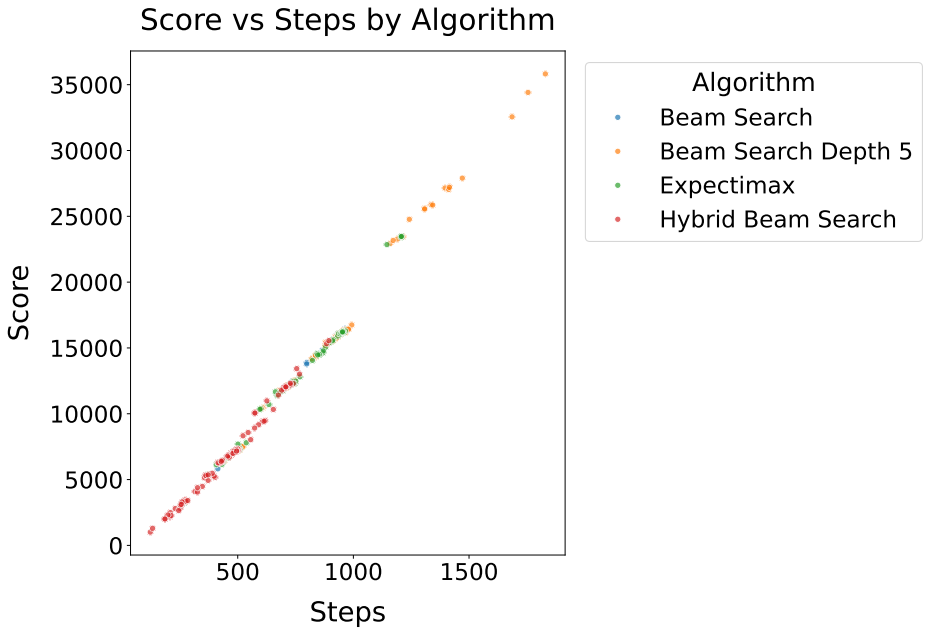

Figure VI: Score by Step.

and evaluated through the CNN, and ultimately taking a long time.

The Hybrid CNN-Beam Search was also time-consuming with an average of approximately 342 seconds per episode, although this is longer than the non-hybrid, Beam Search (Depth 5) (190 seconds). The additional time for the Hybrid agent can be explained by the additional computation of the set of hand-crafted heuristics (empty cells, monotonicity, smoothness, corner bonus) for every state at each step of the search, considered within the beam. The heuristics were meant to provide better direction, but qualifying them does add a computational burden.

The non-hybrid CNN-Beam Search agents were significantly faster than the hybrid agents. CNN-Beam Search (Depth 5) had an average of approximately 190 seconds per episode and the shallower CNN-Beam Search (Depth 4) was the fastest of all agents explored at about 113 seconds. The difference between a depth 5 and a depth 4 Beam Search is reflected in the depth of the search: the more states to evaluate, the more time required. Regardless, both

Beam Search agents were also significantly faster than the Expectimax and the Hybrid Variation. The two Beam Search agents displayed the efficiency improvements from the heuristic pruning produced by the beam search process the majority of the time in the rarest case, in which the agent was forced to explore an unreasonable part of the tree.

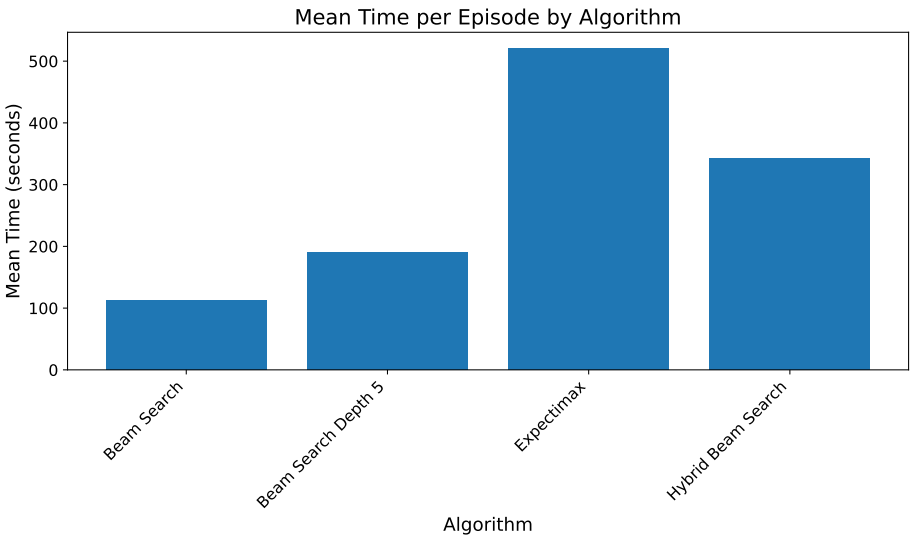

Figure VII: Mean Time in seconds.

To summarize, the evaluation of the four agent architecture : Beam Search (Depth 4), Beam Search (Depth 5), Expectimax (Depth 5), and Hybrid Beam Search (Depth 5), have yielded a varied understanding of their respective capabilities and efficiencies in mastering the 2048 game. Our analysis, grounded in metrics of game score, maximum tile achievement, mean time, and score by step, reveals a very clear performance hierarchy between learned evaluation functions, search depth, strategy and explicit integration of heuristic knowledge. The examination of the overall score (Figure II) and score distribution (Figure III), unequivocally positions the Beam Search (Depth 5) as the most consistently high performing agent. Its superior mean and high scores, tied with a score distribution whose interquartile range is situated at a higher level that its counterparts, indicate a capacity for achieving high scores with a high level of consistency, doing so, it also yielded better results in terms of computational time required, compared to Expectimax (Depth 5). Beam Search (Depth 4) performed respectably but was evidently constrained by its shallower lookahead. Hybrid Beam Search

registered the lowest scores, indicating that the particular mixture of learned evaluation, did not provide useful incentives to achieve the results we hoped for; this may be due to an over exageration of the heuristics (Snake Pattern, Corner Bonus). Regarding the critical objective of creating high-value tiles, Beam Search (Depth 5) again demonstrated its superiority Figure IV. It's success rate of approximately 18,2% was unmatched by any other agent. This proficiency also extends to the consistency it showed in achieving the 1024 tile (84,9%), underscoring a robust capability to navigate mid-to-late game scenarios. Expectimax agent also proved to be capable of achieving the 2048 tile (5,1%) and frequently produced 1024, confirming the viability of its probabilistic search approach for significant tile progression.

The analysis of the relationships regarding game progression, including maximum tile achieved and final score (Figure V), and score versus steps Figure VI, yielded further observations. In fact, all agents exhibited a strongly positive correlation supported by the evidence: max tiles are associated with higher final scores, and greater steps (longer game) are tied to higher final scores. However, these correlations clearly show distinctions. The CNN-Beam Search (Depth 5) not only achieved the highest tiles more often, but also scored higher whenever it achieved a high tile, offering evidence of generating the highest tiles more effectively in terms of accumulation via a score (Figure V). Further, it resulted in higher steps (Figure VI), demonstrating the ability to make efficacious moves, and extended gameplay focused on scoring higher. Comparatively, the Hybrid Beam Search exhibited shorter gameplay duration and lower scores for the same maximum tile achieved, reinforcing the idea that the hybrid nature of the strategy led to premature endings to the game or a path to less optimal means of accumulating score points resulting in terminus game endings. Although overall trajectories were different, the strong clustering of scores as a function of linear trajectory regarding steps taken (Figure VI) for each agent indicates the scores stabilized after sustaining key strategies at least regarding mean score per move. In conclusion, the evaluation of computational efficiency (Figure VII) illustrated the tangible considerations at play. The Expectimax agent, as is expected for such an exhaustive probabilistic evaluation approach, was the most computationally taxing, and took many more seconds to process each game episode than any other agent, and it elucidated the extent of the trade-off derived from what was initially thought

to be more feasible approach toward examining stochasticity. The Hybrid Beam Search, as expected, indicated some reasonable penalty in time on comparing to the non-hybrid Beam Search agents; this was largely based on the expense of processing multiple heuristics for every state along which the beam search conceptually worked. Indeed, the non-hybrid Beam Search agents were especially efficient, and the Depth 4 was the most efficient, while the Depth 5 agent was an excellent middle ground of sorts, achieving the best overall score and 2048 modal tile value, while also having a mean computation time much less than both Expectimax and the Hybrid agent. All of this speaks to the efficiency gains realized through the pruning of Beam Search heuristics, along with limits placed on the agent's exploration of the built out excessively large game tree.

V Conclusions and Future Directions

This thesis presented a study of combining Convolutional Neural Networks with a tree search to engage with the stochastic and strategically complex game of 2048, with the goal of developing a robust deep learning-based state evaluator and testing it in different search contexts. Our experiments illustrate a distinct performance ordering among the various agents, with Beam Search (Depth 5) as the best performing agent consistently achieving the highest game scores and the highest success rate of creating the 2048 tile. This illustrates the strategic advantage of deeper search guided by a competent learned evaluator. Although Expectimax (Depth 5) was capable of reaching 2048, it was often more volatile in an overall performance sense and had a much greater computational expense. The shallower Beam Search (Depth 4) had an efficient computational cost but was limited by it's depth of look ahead, and lastly the Hybrid Beam Search did not provide a performance improvement. This study contributes to the literature of deep learning showing that deep CNNs are robust state evaluators in stochastic games, and showed the efficiency of a number of related search structure with the state evaluator, while the Hybrid agent performance also serves as an important case study on the complexities and challenges of how to best reasonably combine learned knowledge and symbolic knowledge. Although this work was comprehensive in objective, there were some limitations in this study, such as compute power available to better optimize hyper-parameters and CNN training time (400,000 episodes), as well as the final configuration of heuristics for the Hybrid agent. These limitations will allow for a number of suggestions and future research opportunities. First, we will acknowledge that if our current CNN was trained for a longer duration (billions of episodes), the evaluative capabilities of the game policy would improve, and perhaps create the opportunity for the CNN to better evaluate and internalize complex strategies in the game scenario. Secondly, potential exploration of the CNN architect could explore further features in the models, for example attention mechanisms. As there are considerable computational costs in reinforcement learning experiments, it is vested interest to develop lower computational cost design, such as knowledge distillation of the original model into smaller, faster models, searching for reinforcement learning algorithms

that are more sample efficient, or developing adaptive search algorithms that have parameters that evolve according to state . The hybrid framework can certainly be further refined, such as systematically engaging with the full range of heuristics available, and perhaps using an automated learning of weights on the heuristics in the context of its use with the CNN to develop more dynamic and synergistic integrated forms. Finally, the next steps could also explore different search models, particularly simulating a Monte Carlo Tree Search with our trained CNN, to see how our method, with its emphasis on efficiency in calculating trade-offs of exploration including designed bias towards the 2048 tile.

References

Liang Huang, Kai Zhao, and Mingbo Ma. When to Finish? Optimal Beam Search for Neural Text Generation (modulo beam size), August 2018. arXiv:1809.00069 [cs].

Alexis Roche. EM algorithm and variants: an informal tutorial, September 2012. arXiv:1105.1476 [stat].

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm, December 2017. arXiv:1712.01815 [cs].

Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition, April 2015. arXiv:1409.1556 [cs].

Alexey Slizkov. Computational bounds for the 2048 game, March 2023. arXiv:2303.07266 [cs].