CNN-Based game agent for Chrome Dino Game Revisited

Ashok Gaire[1], Yangyang Tao[1], and Junxiu Zhou[1]

[1]School of Computing and Analytics, Northern Kentucky University, Highland Heights, KY, USA
gairea1@mymail.nku.edu, taoy1@nku.edu, and zhouj2@nku.edu

***Abstract*—The automation of game-playing agents is a key domain for advancing AI algorithms. This research applies Convolutional Neural Networks (CNNs) to create an agent for the Chrome Dino game, overcoming the limitations of traditional rule-based systems. We address the core challenge of processing real-time visual data to execute precise jump and duck actions in a dynamic environment. Our solution is an end-to-end CNN model that learns control policies directly from pixels, eliminating the need for hand-crafted features or internal game access. Experimental results confirm the model's effectiveness, with performance metrics demonstrating proficient real-time decision-making and strong potential for visual-based automation in rapidly changing environments.**

***Index Terms*—Chrome Dino Game, CNN, AI Agent for Game**

I. Introduction

The Chrome Dino game (Figure 1), accessible at chrome://dino, is a popular offline game built directly into Google Chrome. The game challenges players to navigate a pixelated dinosaur through an endless desert while avoiding obstacles such as cacti and flying pterodactyls. The game increases in speed over time, requiring quick reflexes and precise timing to achieve a high score. Traditionally, human players rely on visual cues and reaction time to control the dinosaur's jumping and ducking actions. In this research project, we apply CNN to train an AI agent to play the game autonomously. This study explores the performance of the CNN-based model in various aspects, including accuracy, adaptability, and decision-making efficiency, demonstrating the potential of deep learning in real-time game automation and other fields like robotics and automobiles.

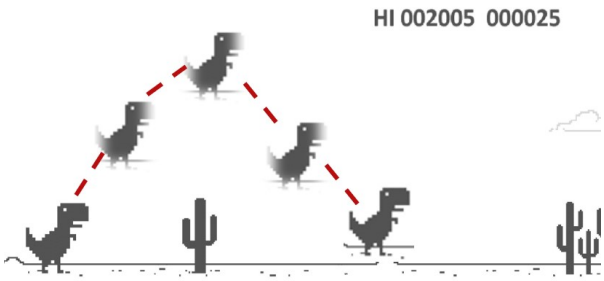

Fig. 1: Chrome Dino Game.

The application of artificial intelligence to gaming agents often stems from the desire to overcome the fundamental limitations of traditional, non-generalizable, rule-based systems. While rule-based agents are predictable and computationally efficient, their behavior is inherently limited, brittle, and lacks adaptability. This prevents them from generating novel tactics or responding intelligently to unforeseen player strategies, resulting in repetitive and ultimately predictable gameplay that undermines long-term engagement [1]. Consequently, the field has turned to machine learning to create more dynamic and robust agents. However, this shift presents a new set of challenges; the adaptive and "black-box" nature of learning models makes them difficult to control, often leading to unpredictable or superhuman behavior that, like the rigid behaviors they are designed to replace, can be detrimental to the player experience [2]. The core issue, therefore, evolves from simple capability engineering to balancing the flexibility of AI with the design imperative of reliable, transparent, and engaging interactions.

In this paper, we would like to design an

game agent based on CNN. The foundation of this research involved the systematic collection and processing of game key frames from the Chrome Dino game. Screenshots of a defined size (250, 770) were selected based on preliminary experiments to balance computational efficiency with sufficient visual detail. The data collection pipeline consisted of three key steps: first, a program-based AI was developed to capture and store screenshots as data frames in a CSV format; second, these data frames were shrunk for manageability; and third, the data was loaded and transformed into a format suitable for training. These processed frames were used to train a CNN. The model's architecture was designed for image classification, featuring three convolutional layers for feature extraction, followed by max-pooling layers to reduce overfitting. A flattening layer then converted the features into a one-dimensional vector, which was passed to two dense layers, culminating in a softmax activation for multi-class classification. By learning to recognize visual patterns like obstacles directly from the pixel data, this CNN-based agent could make real-time jump or duck decisions, effectively playing the game through learned perception rather than manually coded rules. The benefits of the proposed solution are below:

The proposed method offers several key benefits. Firstly, the Convolutional Neural Network (CNN) excels at automatically learning hierarchical visual features directly from raw pixel data, enabling it to discern not just simple obstacles but also complex patterns and contextual cues within the game frame that would be difficult to hand-engineer. Secondly, we established a fully automated data collection pipeline that systematically harvests game frames from the Chrome Dino game, ensuring a large, consistent, and tailored dataset for training. Finally, rigorous evaluation confirms that the trained agent achieves a high level of performance, demonstrating robust and reliable gameplay with statistically significant accuracy.

The remainder of this paper is structured as follows: Section II provides an overview of related work, while Section III introduces the System model and proposed design. Experiment and Result are presented in Section IV, followed by the conclusion in Section V.

II. Related Works

A. AI Agent in Games

Recent years have witnessed significant progress in game AI, largely driven by implementations of Deep Learning. The seminal work of [3] demonstrated that a deep neural network could be trained with Q-learning to solve complex tasks directly from visual inputs, paving the way for numerous AI agents in games like Atari and Mario. A central advantage of such model-free RL algorithms is their capacity to learn optimal actions without relying on a pre-defined transition model, making them exceptionally suited for dynamic and complex environments. Building upon this body of research, our study explores the application of various model-free RL algorithms to the Chrome Dino Run game, with the objective of benchmarking their relative performance in this specific domain.

B. Convolutional Neural Network

CNNs have emerged as a foundational methodology in the domain of visual perception for intelligent systems, establishing a powerful paradigm for processing spatial data. Pioneering architectures such as LeNet [4] and AlexNet [5] demonstrated the profound capability of CNNs to automatically and hierarchically learn feature representations directly from raw pixel inputs, a significant advancement over hand-crafted feature engineering. This core principle—utilizing convolutional layers to detect spatial hierarchies of patterns, from simple edges to complex objects—has made CNNs the standard for image classification, object detection, and scene understanding. Their success has naturally extended to game environments, where they have been widely adopted to transform visual state information into actionable policies. For instance, researchers have leveraged CNNs as the visual backbone for Deep Q-Networks (DQN) to achieve human-level performance in complex Atari games [3], mapping pixels directly to game commands. This established precedent validates the use of CNNs for parsing the visual state of the Chrome Dino game, providing a robust and well-understood

framework upon which to build a perceptive game agent. Our work builds upon this rich tradition, applying a customized CNN architecture to learn the specific visual features required for obstacle recognition and navigation in a side-scrolling environment.

C. Existing Solutions

The development of AI agents for the Chrome Dino game has yielded several technical approaches, each with distinct advantages. Deep Learning represents a powerful, end-to-end solution that learns directly from raw screen pixels. This method's key strength lies in its ability to learn optimal feature representations automatically, thereby circumventing the limitations and potential inaccuracies of manual feature engineering; studies have demonstrated its superiority over traditional feature-based algorithms [6]. In contrast, another effective approach utilizes Multi-Layer Perceptrons (MLPs) that are trained on precise, low-dimensional game-state data, including obstacle distance and game speed [7]. Although this method can yield high performance, its dependency on internal game parameters makes it less generalizable than vision-based models that operate solely on pixel input. Our approach will use a more specific model CNN to create the agent.

III. System Model and Proposed Design

A. System Model

Our methodological design centers on an automated pipeline for collecting a labeled dataset of game frames from the Chrome Dino game. The implementation leverages a suite of Python libraries to orchestrate this process: pyautogui for screen capture, keyboard for input detection, os for file management, and threading to run data collection concurrently with gameplay, ensuring synchronization. NumPy and pandas are utilized for efficient image data handling and structuring the final dataset. The system's architecture comprises five integrated components: (1) Data Storage Setup, which initializes the directory structure; (2) Helper Functions for core utilities; (3) a CSV Storage module for logging metadata; (4) a Screenshot Capture engine; and (5) a Key Press Monitoring module. In practice, as a user plays the game, the script runs concurrently, capturing screenshots precisely upon a keypress event (e.g., spacebar). Each captured frame is automatically labeled with the corresponding action and saved as an image file, while its file path and action label are concurrently logged in a structured CSV file. This workflow results in a robust, time-synchronized dataset ideal for training a vision-based game agent.

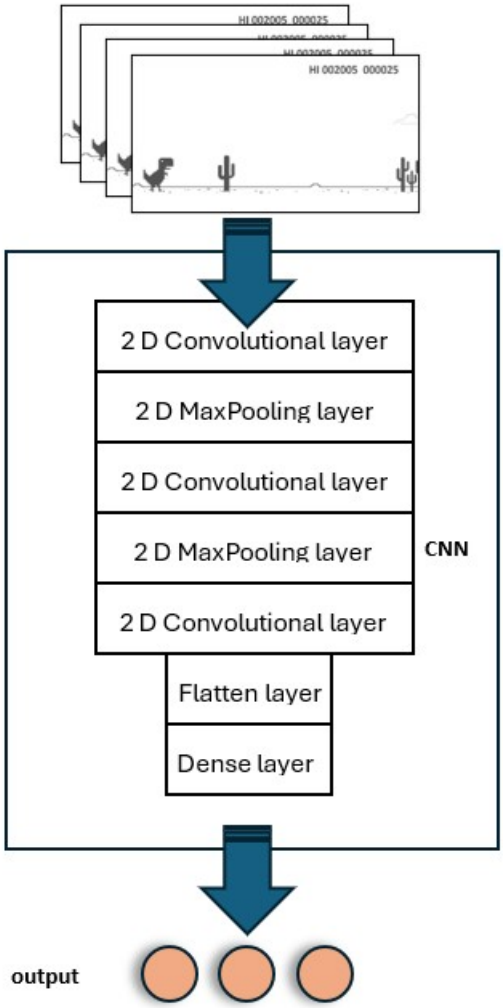

Fig. 2: CNN model.

B. Proposed CNN Design

The core of our agent is a CNN (Figure 2) designed to process game frames and output action commands. The architecture is engineered for the specific task of playing the Chrome Dino

game, balancing feature extraction capability with computational efficiency. The detailed design of each component is as follows: (1) Input Layer: The model accepts pre-processed grayscale game screenshots with a normalized resolution. (2) Feature Extraction Block (Convolutional and Pooling Layers): This block is responsible for hierarchically learning the spatial features essential for identifying obstacles, the horizon, and other game elements. (3) Convolutional Layer (Conv2D): The model employs the Conv2D layers, act as trainable feature detectors. (3) MaxPooling2D Layer following each Conv2D layer: This layer performs a down-sampling operation that reduces the spatial dimensions. (4) Classification Block (Flatten Layers): the high-dimensional, multi-channel feature maps are transformed into a one-dimensional vector by this layer. Dense Layers (Fully Connected): The flattened vector is then passed through two Dense layers. The first Dense layer performs high-level reasoning on the aggregated features. The final Dense layer acts as the output classifier. Its number of neurons is equal to the number of possible actions (3 for Jump, Duck, No-op). It uses a Softmax activation function to output a probability distribution over these actions, where the neuron with the highest probability is selected as the model's decision.

IV. Experiment and Result

The first step (Algorithm 1) is data collection. The core process involves capturing game frames synchronized with their corresponding action labels in real-time. The script utilizes a continuous loop where it performs four critical steps per iteration. First, it captures a screenshot of a predefined game region (770x250 pixels), isolating the core gameplay area. Second, the script emulates an AI player by analyzing specific pixel columns within this screenshot to detect obstacles. It calculates the background color and then scans a vertical search zone ahead of the dinosaur. If pixels in the lower zone deviate from the background, it identifies a ground obstacle (cactus) and triggers a jump. If pixels in an upper zone indicate a gap, it triggers a duck action. If no obstacles are detected, the action is labeled as walk/run. Third, the captured screenshot is converted to grayscale and then flattened into a single-row DataFrame. The determined action label is appended as a new column to this row. Finally, each processed DataFrame is stored in a list.

The second step the training the CNN model (Algorithm 2). It contains 2 steps, one is the dataset reconstruction and the other is the model training. The dataset reconstruction procedure is executed as follows: It begins by loading the original dataset where each image contains a flattened array of 192,500 pixels (250x770 grayscale image) and a corresponding action label. Then each image is first reshaped back into its original 2D format. The core resizing operation is performed using the Python Imaging Library (PIL). Each reconstructed image is downsampled to a significantly smaller resolution of 25x77 pixels. This 100x reduction in the number of pixels is crucial for reducing the computational cost and memory footprint of the subsequent model training without completely sacrificing the spatial structure needed to identify obstacles. The last step is data reformation and storage. The resized image is then converted back into a flattened 1D array of 1,925 elements. This array, along with its original action label, is stored in a dataframe for training.

Algorithm 1 Automated Dino Game Data Collection

```
procedure DataCollection
    Initialize capture region and detection zones
    Delay(3)                    ▷ Wait for Chrome focus
    while True do
        if KeyPressed('q') then break
        end if
        Capture frame and sample background
        for i ← end downto start do
            if Ground obstacle then
                Jump, label=1, break
            else if Flying bird then
                Duck, label=2, break
            end if
        end for
        Store labeled frame
        Adjust for game speed
    end while
end procedure
```

Algorithm 2 Image Preprocessing and CNN Training

procedure PREPROCESSANDTRAIN
  for $i \leftarrow 0$ **to** $n-1$ **do**
    Resize frame $250 \times 770 \rightarrow 25 \times 77$
    Flatten and label data
  end for
  Save shrunken dataset
  $(X_{train}, y_{train}) \leftarrow$ LOADDATA
  Normalize and reshape X_{train}
  model $\leftarrow$ BUILDCNN
  Train and evaluate model
end procedure
function BUILDCNN
  return Conv2D(32,3)-MaxPool $\rightarrow$ Conv2D(64,3)-MaxPool $\rightarrow$ Conv2D(128,3) $\rightarrow$ Flatten $\rightarrow$ Dense(128) $\rightarrow$ Dense(3, softmax)
end function

The above data is reshaped into a 4D tensor of dimensions (num of samples, 25, 77, 1), which corresponds to (number of images, height, width, channels). The pixel values are normalized from the integer range [0, 255] to the floating-point range [0, 1] as the input for CNN. The CNN model is constructed with a sequential architecture specifically designed for the image classification task of interpreting the resized game frames. The basic architecture is shown in Figure 2. The CNN is a sequential stack of 6 layers. 2 convolutional blocks each with a Conv2D Layer and a MaxPooling2D Layer. The only difference is the filer size of the Conv2D Layer, one is 32 and the other is 64. Then a third layer of Conv2D Layer with 128 filters. The rest 2 layers are Flatten Layer and Fully Connected (Dense) Layer with 3 outputs (corresponding to the actions: Walk, Jump, Duck). The training process using batches of 32 samples. After training, The model's performance is simultaneously validated on the test set. The model is then saved to disk for future deployment in the game.

The training performance is shown in Figure 3. The model reaches stable after 4 epochs. Then the model is deployed in the real environment to test the performance. The Dino game is running in the web browser at address chrome://dino. The model is running as a background script to capture the real time screenshots of the game window. Under the ideal scenario, with manual start of the game, the highest score so far achieved is 1500.

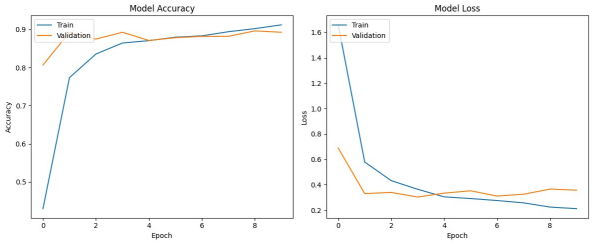

Fig. 3: CNN model training.

V. CONCLUSION

This project successfully developed a vision-based agent for the Chrome Dino game using a CNN. An automated pipeline collected and labeled game frames to train an end-to-end model that maps pixels directly to actions. The agent learned robust features from downsampled images, confirming deep learning as a viable alternative to manual, rule-based programming. While limitations exist regarding training data dependency and potential information loss from resizing, this work establishes a strong proof-of-concept. It paves the way for future enhancements using Deep Reinforcement Learning, ensemble methods, or recurrent architectures to better handle temporal dynamics.

REFERENCES

[1] G. N. Yannakakis and J. Togelius, *Artificial intelligence and games*. Springer, 2018, vol. 2.

[2] M. Mateas, "Expressive ai: A hybrid art and science practice," *Leonardo*, vol. 34, no. 2, pp. 147–153, 2001.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[4] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 2002.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[6] D. Marwah, S. Srivastava, A. Gupta, and S. Verma, "Chrome dino run using reinforcement learning," *arXiv preprint arXiv:2008.06799*, 2020.

[7] J. Ke, Y. Zhao, and H. Wei, "Ai for chrome offline dinosaur game," 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:27833607