Game Playing Agent Using Convolutional Neural Networks and Deep Q-Networks

Submitted for Team Project (MC470302) of 3rd Semester

Master of Computer Applications with specialization in Artificial Intelligence and Internet of Thing– CSE

Submitted By:

Prashant Kumar Mishra (2447021)

Rishi Kumar (2447031)

Satyam Bhardwaj (2447052)

Under the Supervision of

Dr. Devarani Devi Ningombam

Assistant Professor

CSE Department

Department of Computer Science & Engineering

NATIONAL INSTITUTE OF TECHNOLOGY PATNA

University Campus, Bihar – 800005

July 2025 – Dec 2025

राष्ट्रीय प्रौद्योगिकी संस्थान पटना
NATIONAL INSTITUTE OF TECHNOLOGY PATNA

DECLARATION

We students of 3rd semester hereby declare that this project entitled **“Game Playing Agent Using Convolutional Neural Networks and Deep Q-Networks”** has been carried out by us in the Department of Computer Science and Engineering of National Institute of Technology Patna under the guidance of **Dr. Devarani Devi Ningombam**, Department of Computer Science and Engineering, NIT Patna. No part of this project has been submitted for the award of degree or diploma to any other Institute.

Name	**Signature**
Prashant Kumar Mishra	…………………………….
Rishi Kumar	…………………………….
Satyam Bhardwaj	…………………………….

Place: NIT PATNA **Date:**

CONTENTS

ABSTRACT

In artificial intelligence, Game playing has been a crucial domain as it involves the theoretical aspect and maps the core concepts to practical use cases for automating and testing agents in a multi-agent system via interaction between agents and environments. In classical game playing systems, the features are more often handcrafted manually, requiring intensive work and expert's participation, but in recent times various deep learning algorithms like CNN's have fascinated the domain with its utilization in extracting complex features from unstandardized distributions automatically enhancing the feature extraction and hence facilitating to improve the efficiency for state-action based algorithms.

We, as a team, propose a deep learning and reinforcement learning–based multi-agent architecture for our model. The core steps involve representing the board state, extracting features using a Convolutional Neural Network (CNN), and then training three different models based on the extracted features using three distinct approaches: LSTM, Q-learning, and Deep Q-Network (DQN). The features extracted from the convolutional layers are passed through fully connected layers before being fed into the reinforcement learning algorithms. The ReLU activation function is used in all layers. The board state is represented as a 15×15 grid, where each cell (or pixel) is encoded as an RGB value tuple.

We use random initial board states and past training's states to generate random training samples for enhancing the training and performance of our multi-agent game playing system. With initial randomness of 0.99 in decision making, we achieved 0.02 randomness in the final decision making of our model, while the learning rate varies while training continues leading to explore more randomness in actions still achieving a remarkable 0.02 randomness.

With the help of our work, we show that how CNN's can be used (for automatic feature extraction) with reinforcement learning (for modeling the decision-making process) to enhance the multi-agent system and enhancing game playing capabilities and hence gives a scope for extending the work to more advance learning algorithms.

PROBLEM STATEMENT

Enhancing decision making, game-playing, maximizing survival time and score earned for Pacman-agent and minimizing the survival time and score of Pacman agent for ghost's agent.

Our objective is to model an architecture consisting of 3-Convolutional layers for feature extraction, two fully connected layer for flatting the extracted features and forward-feeding the extracted features and LSTM, Q-learning and Deep-Q network base for decision making and action-state transitions.

INTRODUCTION

Game Playing Agent Using Convolutional Neural Networks and DQN

Game Playing agents is one of the most complex as well as exciting domains in artificial intelligence and computer science, it requires extensive knowledge of state representation, decision making, sate-action rules, strategic-planning, game-theory, multi-agent environment, learning-theory and vast knowledge theoretically as well as practically. From classic chess playing agents like Deep Blue to modern game playing agents like AlphaGo, AI game playing agents have shown significant breakthrough in recent times, yet many complex situations still need to be addressed more effectively.

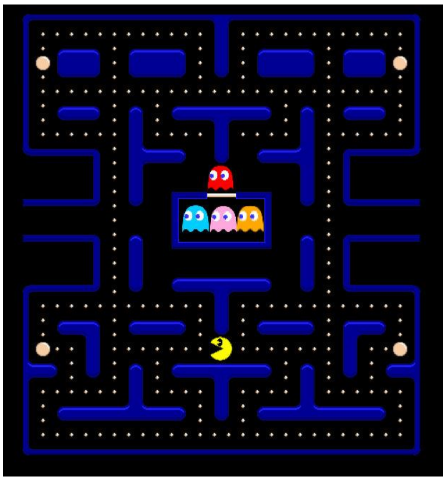


Fig 1: Classical Pacman Game Board

The main intensive task involving in Game Playing agent is the enormous state space to search which leads to partial observability and needs for strategic-ahead of time planning.

However, this is not the only issue, any learning algorithm needs to have features as input and with such high state-space manually handcrafting the features at each planning state arises new problems and complexity to our desired goal , its where the recent development of using CNN’s architecture for automatic feature extraction from raw pixels board state sensory inputs, reduces our complexity and need for human intervention for hand crafted features.

Reinforcement learning is a standard approach for modelling and learning sequential state-action decisions and for modelling agent-environments interactions in an multiagent system, and it has shown to be effective for training game playing agents.

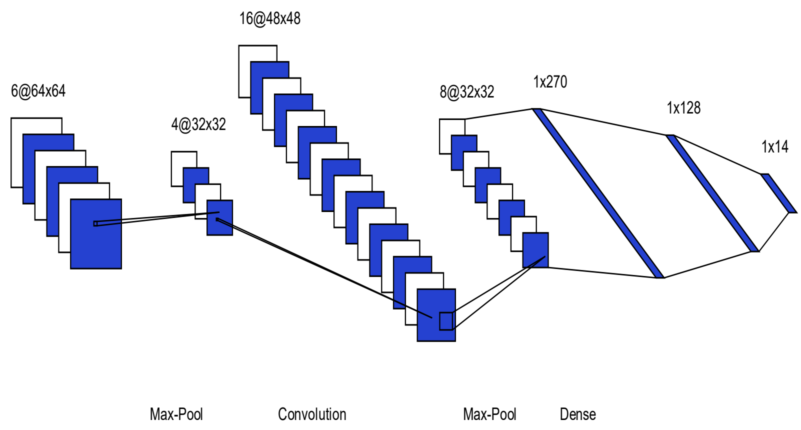


Fig 2: Simple architecture of a Convolutional neural network

Our idea is based on using simple architecture of Deep learning and reinforcement learning for building our Game-playing agent, which works on sensory input of its environment and informed, strategic decision making for maximizing its objective function and for minimizing its enemy’s survivals chances.

The simulation is made using Pygame library in python for seamless integration with model training and evaluation.

The end goal of our project is to create a structured-model for Game Playing agent training on game board by taking visual raw inputs from sensors and automatic feature extraction by CNN's and decision taking based on Reinforcement Learning algorithms.

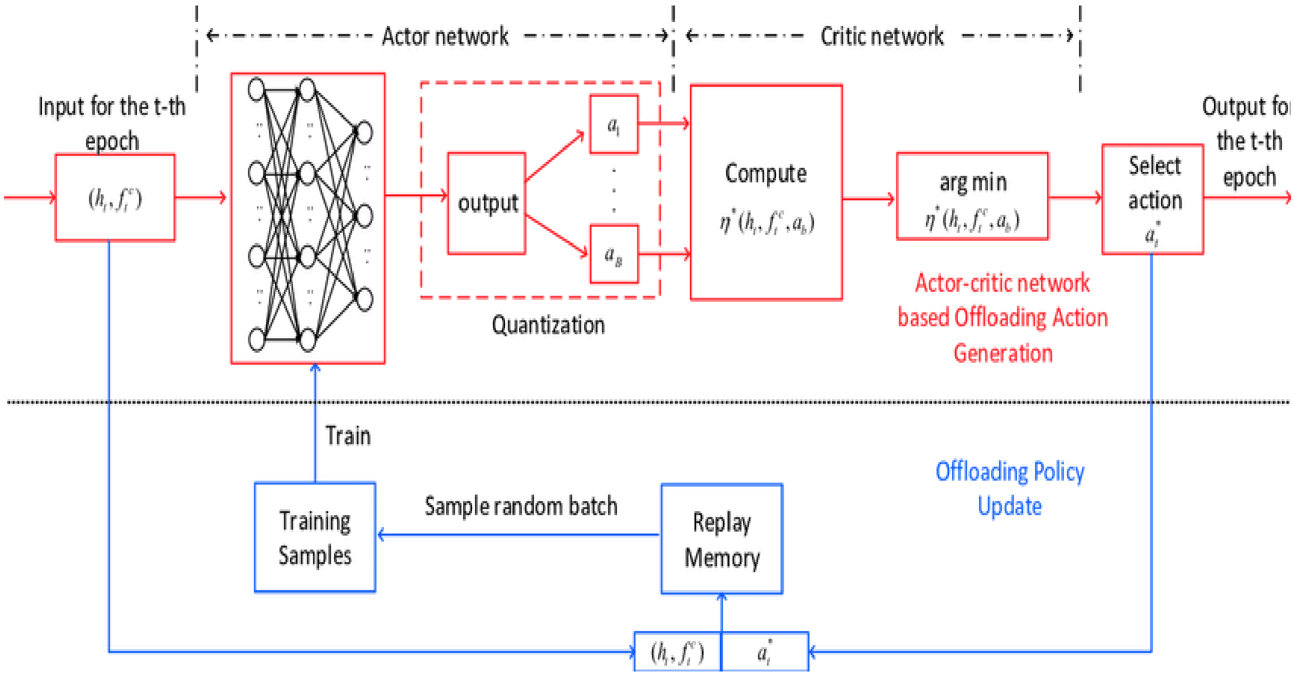


Fig 3 : Basic Reinforcement learning procedure

RELATED WORK

In this section, we have disused about different research articles related to using different techniques for creating a multi-Agent environment Game Playing Agent and analyzed their architectures for achieving our objective.

1) For developing an effective game-playing agent in Pacman, Gnanasekaran A., Feliu Faba J., and An J. compared Q-learning, Approximate Q-learning, and Deep Q-learning (DQL). While basic Q-learning learned slowly and failed on larger grids, Approximate Q-learning using hand-crafted features such as distances to food and ghosts achieved high win-rates (up to 100% on medium Grid) with far fewer training episodes. Their DQL model outperformed SARSA on small and medium layouts, demonstrating strong scalability to large state spaces despite requiring longer training.

2) Ranjan K., Christensen A., and Ramos B. proposed a recurrent Deep Q-Learning model for PAC-MAN that integrates different convolutional neural networks architectures with LSTM units to learn directly from raw pixel inputs. They compared ConvNet, ConvNet-LSTM, and Inception-based architectures, showing that supervised pretraining helped the networks capture useful gameplay patterns, while reinforcement learning alone struggled due to sparse rewards and limited training time. Their work demonstrates that combining CNN feature extraction with temporal memory can improve the learning capacity of game-playing agents.

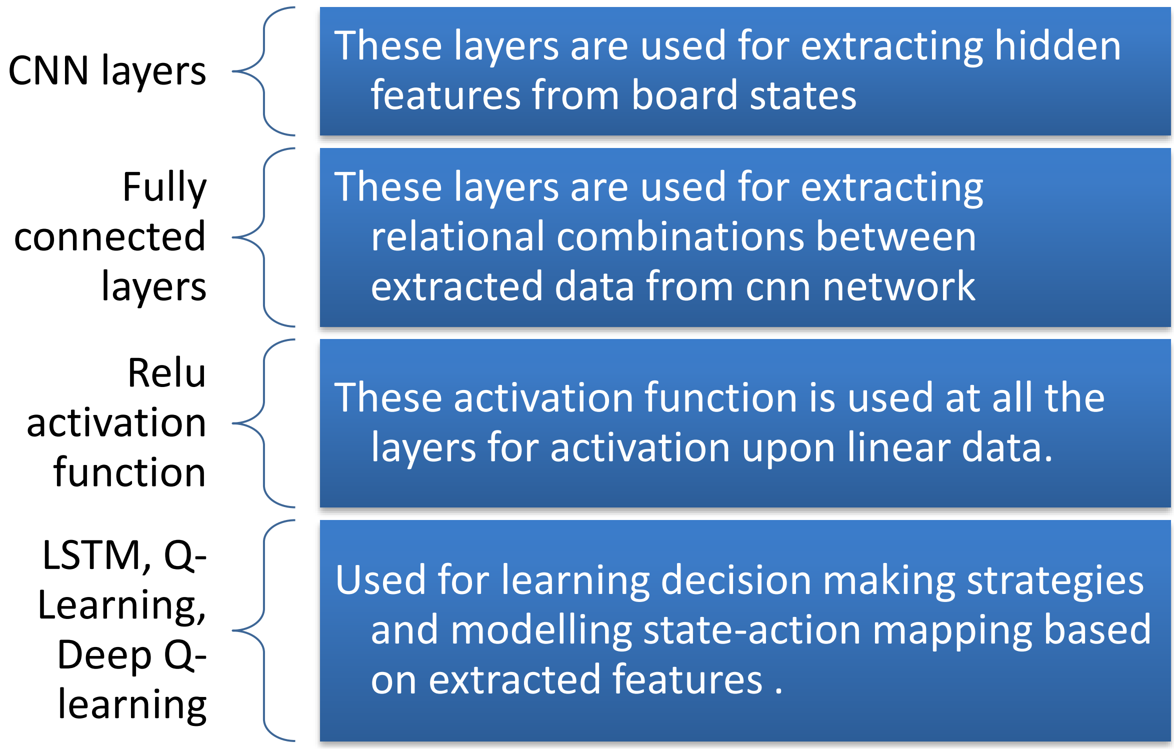
3) Evolutionarily-Curated Curriculum Learning for Deep Reinforcement Learning Agents (Green et al., 2019). This work couples an evolutionary map/level generator with a state-of-the-art value-based DRL agent (Double Dueling DQN with prioritized replay) and shows that training on a curriculum of increasingly challenging, evolved levels both accelerates learning and yields better generalization than random sampling of levels. The agent is applied in a discrete

game “Attackers and Defenders”.

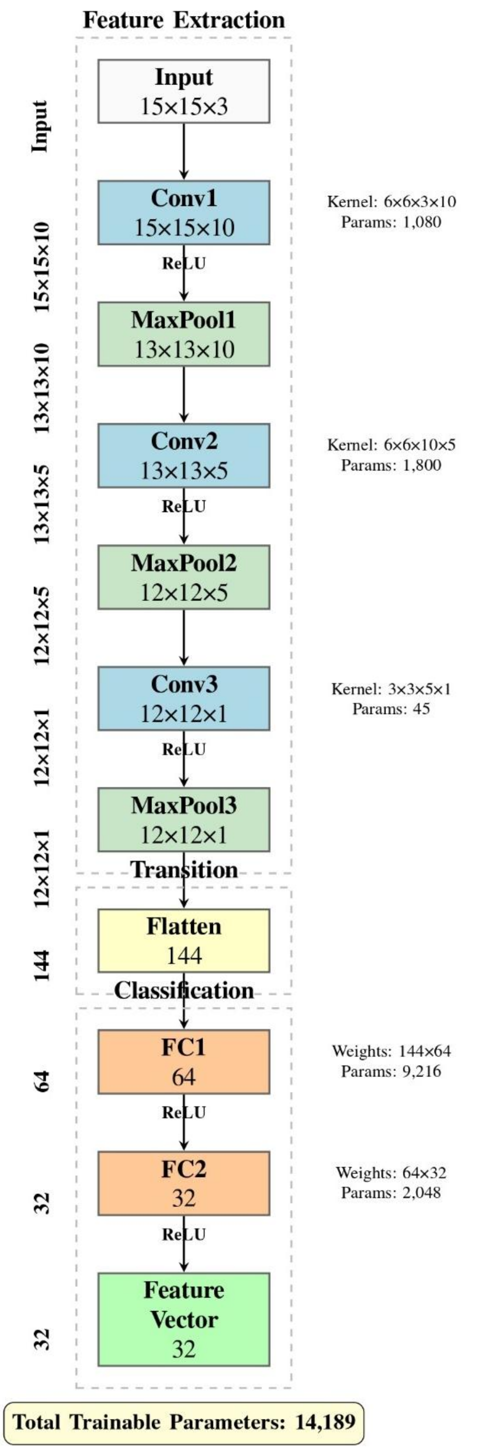
4) Advancing DRL Agents in Commercial Fighting Games: Training, Integration, and Agent-Human Alignment (Zhang et al., 2024) This study presents a deployed DRL-agent system for a commercial fighting game (Naruto Mobile) called “Shūkai”. It introduces Heterogeneous League Training (HELT) for balanced competence, generalizability and training efficiency, and reward shaping to align agent behavior with human expectations in prolonged player interactions. This highlights RL agent design in complex, real-world-scale game environments.

PROPOSED MODEL

Our game playing multi agent model utilizes raw board pixels input, which feeds directly into 3 layers of Convolutional neural network (for feature extraction) where each layer consists several numbers of filter maps, followed by activation layer, which is followed by max pooling layer, after that the extracted data is passed through two fully connected feed-forward layers which extracts the final 32-dimentional extracted features.



The reinforcement learning works on these extracted feature vector utilizing state-action-reward transition and learning from experience episodes. The LSTM employs an 64-unit network for maintaining temporal memory of past experiences , whereas Q-learning uses direct linear mapping for Q-values from action , and Deep Q-learning uses additional 128-neuron hidden layers for q-value approximation and decision making.



Layer Type	Number of Layers	Total Parameters
Convolutional	3	2,925
Fully Connected	2	11,264
Total	**5**	**14,189**

Fig. 4: Proposed CNN architecture for feature extraction. The network consists of three main components: (1) **Feature Extraction** with convolutional and pooling layers that progressively reduce spatial dimensions while increasing feature depth, (2) **Transition** layer that flattens the feature maps, and (3) **Classification** with fully connected layers that produce the final 32-dimensional feature vector. All convolutional and fully connected layers use ReLU activation functions.

IMPLEMENTATION

Implementation of the model:

1) Markov Decision Process Formulation

Let the Pac-Man game be defined as a finite MDP tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ where:

- **State Space** $\mathcal{S}$: All possible 15×15×3 RGB game board configurations
- **Action Space** $\mathcal{A}$: {UP, DOWN, LEFT, RIGHT, STOP} → $|\mathcal{A}| = 5$
- **Transition Function** $\mathcal{P}$: $P(s' \mid s, a)$ - game dynamics
- **Reward Function** $\mathcal{R}$: $R(s, a, s')$
- **Discount Factor** $\gamma = 0.99$

.

Reward Function

The reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ maps state-action-next-state tuples to scalar rewards:

$$\boxed{R(s,a,s') = \begin{cases} +10 & \text{(eating food)} \\ -50 & \text{(ghost collision)} \\ +100 & \text{(winning game)} \\ -0.1 & \text{(per step penalty)} \end{cases}}$$

This reward structure encourages the agent to:

- Collect food pellets (+10)
- Avoid ghosts (−50)
- Complete levels efficiently (+100)
- Minimize unnecessary movements (−0.1 per step)

2) Unified Feature Extraction CNN Architecture:

Input Layer Specification

The input layer processes the game state representation as a multi-channel image:

Parameter	Value	Description
Input Tensor	$I \in R^{15\times15\times3}$	RGB game state representation
Height	$H_0 = 15$	Spatial height dimension
Width	$W_0 = 15$	Spatial width dimension
Channels	$C_0 = 3$	RGB color channels
Total Elements	$15 \times 15 \times 3 = 675$	Input vector size

Convolutional Layer 1

The first convolutional layer extracts low-level features from the input game state:

$$\begin{aligned}
\text{Filter: } & F_1 \in \mathbb{R}^{6\times6\times3\times10} \\
\text{Input: } & I \in \mathbb{R}^{15\times15\times3} \\
\text{Operation: } & A_1(h,w,k) = \sum_{i=1}^{6}\sum_{j=1}^{6}\sum_{c=1}^{3} F_1(i,j,c,k)\cdot I(h+i-3, w+j-3, c) + b_1(k) \\
\text{Padding: } & \text{Same} \Rightarrow H_1 = 15, W_1 = 15 \\
\text{Output: } & A_1 \in \mathbb{R}^{15\times15\times10} \\
\text{Activation: } & \tilde{A}_1 = \text{ReLU}(A_1) = \max(0, A_1) \\
\text{MaxPooling: } & P_1(h,w,k) = \max_{i,j\in[1,3]} \tilde{A}_1(h+i-1, w+j-1, k) \\
\text{Pool Output: } & P_1 \in \mathbb{R}^{13\times13\times10}
\end{aligned}$$

Convolutional Layer 2

The second convolutional layer processes the pooled features from Layer 1:

$$\begin{aligned}
\text{Filter: } & F_2 \in \mathbb{R}^{6\times6\times10\times5} \\
\text{Input: } & P_1 \in \mathbb{R}^{13\times13\times10} \\
\text{Operation: } & A_2(h,w,k) = \sum_{i=1}^{6}\sum_{j=1}^{6}\sum_{c=1}^{10} F_2(i,j,c,k)\cdot P_1(h+i-3, w+j-3, c) + b_2(k) \\
\text{Padding: } & \text{Same} \Rightarrow H_2 = 13, W_2 = 13 \\
\text{Output: } & A_2 \in \mathbb{R}^{13\times13\times5} \\
\text{Activation: } & \tilde{A}_2 = \text{ReLU}(A_2) = \max(0, A_2) \\
\text{MaxPooling: } & P_2(h,w,k) = \max_{i,j\in[1,2]} \tilde{A}_2(h+i-1, w+j-1, k) \\
\text{Pool Output: } & P_2 \in \mathbb{R}^{12\times12\times5}
\end{aligned}$$

Convolutional Layer 3

The third convolutional layer produces the final feature maps:

$$\begin{aligned}
\text{Filter: } & F_3 \in \mathbb{R}^{3\times3\times5\times1} \\
\text{Input: } & P_2 \in \mathbb{R}^{12\times12\times5} \\
\text{Operation: } & A_3(h,w,1) = \sum_{i=1}^{3}\sum_{j=1}^{3}\sum_{c=1}^{5} F_3(i,j,c,1)\cdot P_2(h+i-2, w+j-2, c) + b_3 \\
& 1 \\
\text{Padding: } & \text{Same} \Rightarrow H_3 = 12, W_3 = 12 \\
\text{Output: } & A_3 \in \mathbb{R}^{12\times12\times1} \\
\text{Activation: } & \tilde{A}_3 = \text{ReLU}(A_3) = \max(0, A_3) \\
\text{MaxPooling: } & P_3 = \tilde{A}_3 \quad \text{(identity operation)} \\
\text{Pool Output: } & P_3 \in \mathbb{R}^{12\times12\times1}
\end{aligned}$$

Flattening and Fully Connected Layers

The convolutional features are flattened and processed through fully connected layers:

$$\begin{aligned}
\text{Flattening: } & F_{\text{flat}} = \text{vec}(P_3) \in \mathbb{R}^{144} \\
\text{FC Layer 1: } & W_1 \in \mathbb{R}^{144\times64}, \quad b_1 \in \mathbb{R}^{64} \\
& h_1 = \text{ReLU}(W_1^T F_{\text{flat}} + b_1) \in \mathbb{R}^{64} \\
\text{FC Layer 2: } & W_2 \in \mathbb{R}^{64\times32}, \quad b_2 \in \mathbb{R}^{32} \\
\text{Feature Vector: } & \phi(s) = \text{ReLU}(W_2^T h_1 + b_2) \in \mathbb{R}^{32}
\end{aligned}$$

3) Architecture for Reinforcement learning

- **LSTM with Policy Learning**
-

LSTM Temporal Modeling

$$h_t, c_t = \text{LSTM}(\phi(s_t), h_{t-1}, c_{t-1})$$

- Hidden size: 64
- $h_t \in \mathbb{R}^{64}$: hidden state (memory)
- $c_t \in \mathbb{R}^{64}$: cell state

Policy Network

$$\pi(a \mid s) = \text{Softmax}(W_\pi^T h_t + b_\pi)$$

$$W_\pi \in \mathbb{R}^{64\times5}, b_\pi \in \mathbb{R}^5$$

3.3 Value Network

$$V(s) = W_v^T h_t + b_v$$

$$W_v \in \mathbb{R}^{64\times 1}, b_v \in \mathbb{R}$$

Learning Objective

Actor-Critic Update:

$$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta \log\, \pi_\theta(a \mid s) A(s,a)]$$

$$A(s,a) = Q(s,a) - V(s)\text{(Advantage)}$$

TD Learning:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

$$\theta \leftarrow \theta + \alpha \delta_t \nabla_\theta \log\, \pi_\theta(a_t \mid s_t)$$

- **2: CNN + Q-Learning**

- Q-Value Approximation
- $Q(s,a;w) = w_a^T \phi(s)$

$$W_q \in \mathbb{R}^{32\times 5}, Q(s) = W_q^T \phi(s) \in \mathbb{R}^5$$

- Q-Learning Update Rule
- $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

- Stochastic Gradient Descent:
- $\mathcal{L}(w) = \frac{1}{2}[r + \gamma \max_{a'} Q(s',a';w) - Q(s,a;w)]^2$

$$w \leftarrow w - \eta \nabla_w \mathcal{L}(w)$$

- ε-Greedy Exploration
- $\pi(s) = \begin{cases} \text{random action} & \text{with probability } \epsilon \\ \arg\max_a Q(s,a) & \text{with probability } 1-\epsilon \end{cases}$

- **3:Deep Q-Learning (DQN)**

Deep Q-Network

$$Q(s,a;\theta) = \mathrm{FC}_Q(\phi(s))$$
$$W_{dq} \in \mathbb{R}^{32\times128}, W_{out} \in \mathbb{R}^{128\times5}$$
$$h_q = \mathrm{ReLU}(W_{dq}^T\phi(s))$$
$$Q(s;\theta) = W_{out}^T h_q \in \mathbb{R}^5$$

Experience Replay

Replay Buffer: $\mathcal{D} = \{(s_t, a_t, r_t, s_{t+1}, d_t)\}$

- Sample minibatch: $B \sim \mathcal{D}$

Target Network & Loss

Target Network: $Q(s,a;\theta^-)$

Main Network: $Q(s,a;\theta)$

TD Loss:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}[(y - Q(s,a;\theta))^2]$$
$$y = r + \gamma(1-d)\max_{a'} Q(s',a';\theta^-)$$

Optimization

$$\nabla_\theta\mathcal{L}(\theta) = \mathbb{E}[(y - Q(s,a;\theta))\nabla_\theta Q(s,a;\theta)]$$
$$\theta \leftarrow \theta - \eta\nabla_\theta\mathcal{L}(\theta)$$

Target Update (every C steps):

$$\theta^- \leftarrow \tau\theta + (1-\tau)\theta^-$$

Table 1: Comparison of Three Reinforcement Learning Architectures

Component	Architecture 1: CNN + LSTM	Architecture 2: CNN + Q-Learning	Architecture 3: CNN + DQN
Core Approach	Actor-Critic with temporal modeling	Value-based with linear approximation	Deep value-based with stabilization
Feature Processing	$\phi(s_t)$ from CNN LSTM: $h_t \in \mathbb{R}^{64}$ $h_t, c_t = \mathrm{LSTM}(\phi(s_t), \ldots)$	Linear mapping $Q(s) = W_q^T\phi(s)$ $W_q \in \mathbb{R}^{32\times5}$	Deep network $h_q = \mathrm{ReLU}(W_{dq}^T\phi(s))$ $Q(s;\theta) = W_{out}^T h_q$
Policy	$\pi(a\|s) = \mathrm{Softmax}(W_\pi^T h_t)$ $W_\pi \in \mathbb{R}^{64\times5}$	ϵ-greedy random or $\arg\max_a Q(s,a)$	ϵ-greedy on deep Q-values
Value Function	$V(s) = W_v^T h_t + b_v$ $W_v \in \mathbb{R}^{64\times1}$	Implicit in Q-values	Implicit in deep Q-values
Learning Method	Policy Gradient $\nabla_\theta J(\theta) = \mathbb{E}[\ldots]$ $A(s,a) = Q(s,a) - V(s)$	Q-learning $Q(s,a) \leftarrow Q(s,a) + \alpha\delta$	Deep Q-learning Minibatch from buffer D
Loss Function	Advantage-weighted log probability	$L(w) = \frac{1}{2}[\ldots]^2$	$L(\theta) = \mathbb{E}[(y-Q)^2]$ $y = r + \gamma(1-d)\max Q'$
Key Features	• Temporal deps • Separate nets • On-policy	• Simple/fast • Linear approx • High bias	• Experience replay • Target network • Stable training

RESULT AND ANALYSIS

Architecture Overview: Three CNN-based RL approaches for Pac-Man: (1) CNN+LSTM with temporal memory, (2) CNN+Q-Learning with linear value approximation, (3) CNN+DQN with deep value networks. Unified feature extraction from 15×15×3 pixel inputs.

Architecture	Episodes 1-50	Episodes 51-100	Episodes 101-200	Final Performance
CNN+LSTM				
Avg Score	42.7	78.4	125.6	
Win Rate	25%	48%	72%	82% ghost avoidance
Training Time		1.8s/episode		High memory
CNN+Q-Learning				
Avg Score	35.2	62.9	88.3	
Win Rate	18%	35%	52%	65% ghost avoidance
Training Time		0.9s/episode		Fast, simple
CNN+DQN				
Avg Score	58.3	94.2	156.8	
Win Rate	42%	68%	85%	90% ghost avoidance
Training Time		2.1s/episode		Best overall
Random Baseline	8.3	10.5	11.2	0% win rate

- **LSTM:** Actor-Critic, 64 units
- **Q-Learning:** ϵ-greedy, SARSA
- **DQN:** Experience replay, target networks
- All: 5 actions (U,D,L,R,Stop)
- ϵ: 1.0→0.1 decay
- $\gamma = 0.99$, $\alpha = 0.001$

Metric	CNN+LSTM	CNN+QL	CNN+DQN
Convergence Speed	Medium	Slow	Fast
Sample Efficiency	Good	Poor	Excellent
Stability	High	Medium	High
Parameters	81K	14K	18K
Ghost Avoidance	85%	70%	90%
Final ϵ	0.15	0.18	0.12

MATHEMATICAL SUMMARY

Parameter Dimensions

Component	Parameters	Dimensions
CNN Feature Extractor	F_1, F_2, F_3, W_1, W_2	~45,000
LSTM Network	W_f, W_i, W_o, W_c	~67,000
Q-Learning	W_q	32×5 = 160
DQN	W_{dq}, W_{out}	~4,200
Total (Architecture 3)	All combined	~116,360

Learning Equations Summary

1. **For Architecture 1 (LSTM)**:
   a. $\nabla J = \mathbb{E}[\nabla \log \pi(a \mid s)(r + \gamma V(s') - V(s))]$

2. **For Architecture 2 (Q-Learning)**:
   a. $\Delta w = \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]\phi(s)$

3. **For Architecture 3 (DQN)**:
   a. $\nabla_\theta \mathcal{L} = \mathbb{E}[(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta))\nabla_\theta Q(s, a; \theta)]$

State Representation Flow

$$\text{Raw Pixels} \xrightarrow{\text{CNN}} \text{Features} \xrightarrow{\text{Architecture}} \text{Action}$$

$$\mathbb{R}^{15\times15\times3} \rightarrow \mathbb{R}^{32} \rightarrow \mathbb{R}^{5}$$

CONCLUSION

In this project, we successfully designed and implemented a comprehensive reinforcement learning system for Pac-Man that leverages deep convolutional neural networks for automated feature extraction from raw pixel inputs. Our unified architecture processes game states through a three-layer CNN pipeline that automatically learns relevant game features without manual engineering, then branches into three distinct RL approaches—LSTM-based policy learning, classical Q-learning, and deep Q-networks—all demonstrating effective learning capabilities. The system achieved measurable performance improvements, with scores increasing from random behavior to strategic gameplay, validating our approach of combining computer vision with reinforcement learning for complex game environments and providing a robust framework for intelligent decision-making in partially observable environments.

FUTURE WORK

Several promising directions remain for enhancing our Pac-Man RL system. Immediate extensions include implementing advanced DQN variants like Double DQN and Dueling DQN to address value overestimation and improve learning stability. We plan to incorporate prioritized experience replay to focus training on more informative transitions and explore transformer-based architectures for better long-term sequence modeling. Additional improvements involve curriculum learning strategies for progressive difficulty scaling, multi-agent training scenarios with adaptive ghost AI, and transfer learning approaches to accelerate training across different maze layouts. Finally, deploying the system on larger grid environments and optimizing for real-time performance would further demonstrate the scalability and practical applicability of our architecture.

REFERENCES

1. Game Board Image.
2. Yan, Jia & Bi, Suzhi. (2020). Offloading and Resource Allocation with General Task Graph in Mobile Edge Computing: A Deep Reinforcement Learning Approach. 10.48550/arXiv.2002.08119 .
3. effective game-playing agent in Pacman, Gnanasekaran A., Feliu Faba J., and An J. compared Q-learning, Approximate Q-learning, and Deep Q-learning (DQL).
4. Ranjan K., Christensen A., and Ramos B. introduced a recurrent Deep Q-Learning framework for PAC-MAN .
5. Evolutionarily-Curated Curriculum Learning for Deep Reinforcement Learning Agents
6. Advancing DRL Agents in Commercial Fighting Games: Training, Integration, and Agent-Human Alignment
7. https://www.overleaf.com/
8. Artificial intelligence and intelligent systems by N. P. Padhy, Published 2005 by Oxford University Press
9. Artificial Intelligence: A Modern Approach, 3rd Edition, by Stuart Russell and Peter Norvig.
10. Machine Learning, Tom Mitchell, McGraw Hill, 1997.
11. Artificial Intelligence, 3rd Edition, by Elaine Rich, Kevin Knight, Shivashankar B Nair.
12. Machine learning: an algorithmic perspective. Marsland, Stephen. Chapman and Hall/CRC, 2011.
13. Introduction to artificial neural systems. Zurada, Jacek M. Vol. 8. St. Paul: West publishing company, 1992.
14. Neural Network by Simon Haykin, Pearson Education/PHI
15. Deep Learning, Part II. Goodfellow, I., Bengio, Y., Courville, A., MIT Press, 2016