# Pixel to Policy: Mastering Super Mario Bros with Deep Reinforcement Learning

Anonymous Authors

***Abstract*—This work explores the application of Deep Q-Networks (DQN) to Super Mario Bros [1], focusing on improving feature extraction through residual connections. By integrating a ResNet-based architecture, we improve learning efficiency and stability in high-dimensional visual environments. The study highlights the importance of architectural choices in reinforcement learning and suggests future directions for improvement.**

## I. Introduction

Reinforcement Learning (RL) has gained significant attention for its ability to learn optimal behaviors through interactions with an environment. In this work, we specifically focus on applying Deep Q-Networks (DQN) to complex visual environments, particularly the game of Super Mario Bros[1]. The objective is to explore how well an RL agent can navigate a dynamic, multi-stage environment with high-dimensional observations while facing varying levels of uncertainty and delayed rewards.

This aspect of RL is particularly relevant because many real-world applications, such as autonomous driving and robotic control, involve learning from high-dimensional state representations. Video games provide an excellent benchmark for these tasks due to their structured environments and well-defined objectives. However, applying RL to such domains presents several challenges, including the curse of dimensionality, sparse rewards, and the need for efficient exploration strategies.

Prior work in RL has demonstrated impressive results using deep learning for policy learning. DQN, introduced by Mnih et al. (2015) [2], marked a breakthrough by successfully learning to play Atari games from raw pixel inputs using convolutional neural networks (CNNs). However, DQN struggles with sample inefficiency and catastrophic forgetting, prompting research into improvements such as Double DQN, Prioritized Experience Replay, and distributional RL. These advancements have addressed some of DQN's limitations but leave room for further exploration.

In our approach, we implement a DQN-based agent with ResNet-inspired convolutional layers to enhance feature extraction from pixel-based observations. The agent is trained in a simulated environment of Super Mario Bros[1], where it learns to navigate levels and avoid obstacles through reward-driven optimization. Our evaluation focuses on the agent's learning efficiency, generalization to unseen levels, and robustness against varying game conditions.

Our experiments demonstrate that incorporating residual connections could improve learning stability and convergence speed compared to traditional CNN architectures. However, challenges remain, particularly in overcoming sparse rewards and achieving consistent performance across different levels. The study highlights the importance of balancing exploration and exploitation in visually complex RL environments.

Despite these promising results, several limitations exist. Training deep RL agents remains computationally expensive, and fine-tuning hyperparameters for stability requires extensive experimentation. Future work could explore alternative architectures, such as Transformer-based RL models, or integrate auxiliary learning objectives to enhance representation learning.

To facilitate reproducibility, our full implementation is available in a repository.

## II. Background

Reinforcement Learning (RL) is a branch of machine learning where an agent learns to make decisions by interacting with an environment. The problem is typically formulated as a Markov Decision Process (MDP), defined by the tuple $(S, A, P, R, \gamma)$. Here, $S$ represents the state space, $A$ the action space, $P(s'|s, a)$ the transition probability of reaching state $s'$ from state $s$ by taking action $a$, $R(s, a)$ the reward function, and $\gamma \in [0, 1]$ the discount factor that balances immediate and future rewards.

A fundamental concept in RL is the policy $\pi(a|s)$, which defines the probability of taking action $a$ in state $s$. The objective of the agent is to learn an optimal policy $\pi^*$ that maximizes the expected cumulative reward:

$$J(\pi) = \mathbb{E}_{\tau \sim p_\pi} \left[ \sum_{t=0}^{T} \gamma^t R(s_t, a_t) \right], \tag{1}$$

where $\tau = (s_0, a_0, s_1, a_1, ...)$ represents a trajectory sampled under policy $\pi$.

Deep Q-Networks (DQN) [2] revolutionized RL by enabling agents to learn policies directly from high-dimensional inputs, such as raw pixel images. DQN employs a deep neural network to approximate the action-value function $Q(s, a)$, which estimates the expected return of taking action $a$ in state $s$ and following the optimal policy thereafter. The Q-learning update rule is given by:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left( R_t + \gamma \max_{a'} Q_{\text{target}}(s_{t+1}, a') - Q(s_t, a_t) \right), \tag{2}$$

where $\alpha$ is the learning rate. The Q-network is trained using gradient descent with a loss function based on the Bellman equation. Typically, Mean Squared Error (MSE) or Smooth

L1 loss is used to measure the difference between predicted and target Q-values:

$$\mathcal{L}(\theta) = \mathbb{E}\left[(y_t - Q(s_t, a_t; \theta))^2\right] \tag{3}$$

where $y_t = R_t + \gamma \max_{a'} Q_{\text{target}}(s_{t+1}, a')$ is the target Q-value. The optimization is performed using gradient-based methods such as Adam.

However, DQN suffers from instability and divergence, leading to several improvements such as experience replay and target networks.

Residual networks (ResNets) [3] introduce skip connections to ease training in deep architectures by allowing gradients to propagate more effectively. This principle has been adopted in RL to improve feature extraction from complex visual inputs. By incorporating ResNet-inspired architectures into DQN, it is possible to enhance representation learning, leading to more stable training and improved performance in visually rich environments such as Super Mario Bros [1].

Applying RL to high-dimensional control tasks like Super Mario Bros[1] introduces challenges such as sparse rewards, high-dimensional state spaces, and the need for efficient exploration strategies. Techniques such as Double DQN [4], Prioritized Experience Replay [5], and distributional RL [6] have been proposed to address these limitations, providing better sample efficiency and stability.

In the following sections, we build on these foundations to investigate the effectiveness of a ResNet-enhanced DQN model for learning complex visual tasks in the Super Mario Bros environment[1]. Our approach leverages deep convolutional architectures to improve feature extraction, aiming to enhance training efficiency and policy robustness.

## III. Methodology and Implementation

This section details our approach to implementing a Deep Q-Network (DQN) agent for the Super Mario Bros environment, including architecture design, training procedure, and experimental setup.

### A. Environment

As stated in Section II, we utilized the Super Mario Bros environment provided by the `gym-super-mario-bros` [1] library, which offers a RL-friendly interface to the Nintendo Entertainment System (NES) game. This environment presents several challenges that make it an interesting testbed for reinforcement learning:

- High-dimensional visual input (240 × 256 × 3 RGB pixels)
- Complex dynamics requiring both reactive and planning behaviors
- Sparse rewards, primarily obtained from defeating enemies and progressing rightward
- Long-term dependencies between actions and outcomes

*gym-super-mario-bros* provides different resolution of environment, from the original game (*SuperMarioBros-v0*) to a really downgraded version (*SuperMarioBros-v3*). Even though it would be easier to train on the downgraded version, we choose to keep the original version, especially because we noticed that, in low quality, the enemy and the ground were exactly the same texture. It could have been a huge issue in the learning process. Also, to make the environment more suitable for learning, we implemented several preprocessing steps:

1) **Action space reduction**: We used the `JoypadSpace` wrapper to reduce the original action space from 256 possible button combinations to a simplified set of 5 meaningful actions (RIGHT, RIGHT+A, RIGHT+B, RIGHT+A+B, A).
2) **Frame skipping**: We applied frame skipping with a factor of 4, where the agent only observes every 4th frame but the same action is repeated for the skipped frames. This technique speeds up training while maintaining the temporal structure of the game.
3) **Grayscale conversion**: Color information was removed by converting RGB frames to grayscale, reducing the input dimensionality.
4) **Resizing**: Frames were downsampled to 84 × 84 pixels, significantly reducing the state space while preserving important visual features.
5) **Frame stacking**: We stacked 4 consecutive frames to provide the agent with temporal information, allowing it to infer velocities and directions.

The resulting observation space is a 4 × 84 × 84 tensor, representing 4 stacked grayscale frames.

### B. Network Architecture

We implemented and compared two neural network architectures for the Q-function approximation:

1) **Standard DQN**: A conventional architecture similar to the original DQN paper [2], consisting of three convolutional layers followed by two fully connected layers.
2) **ResNet-enhanced DQN**: An improved architecture incorporating residual connections inspired by ResNet [3]. This architecture includes:
   - An initial convolutional layer with 32 filters (8×8 kernel, stride 4)
   - Two residual blocks, each containing two convolutional layers (3×3 kernels) with batch normalization
   - Max pooling layers after each residual block
   - A fully-connected layer with 512 units
   - An output layer with units corresponding to the 5 possible actions

Figure 1 illustrates the ResNet-enhanced DQN architecture. The residual connections are implemented as:

$$H(x) = F(x) + x \tag{4}$$

where $F(x)$ represents the function learned by the stacked layers, and $H(x)$ is the desired mapping. This allows gradients to flow directly through the network during backpropagation, addressing the vanishing gradient problem in deep networks.

### C. Agent Implementation

Our DQN agent implementation includes several enhancements over the vanilla algorithm:

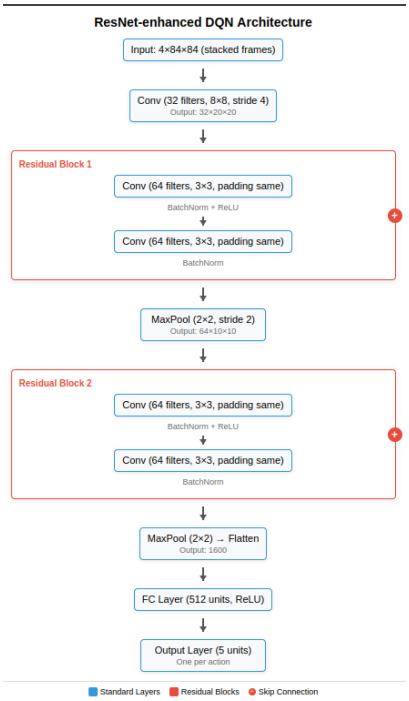


Fig. 1. Residual block structure in our ResNet-enhanced DQN architecture.

1) **Experience Replay**: We used a replay buffer with a capacity of 30,000 transitions. Experiences $(s_t, a_t, r_t, s_{t+1}, done_t)$ are stored and randomly sampled in batches of 64 during training.
2) **Target Network**: We employed a separate target network that is periodically updated every 2,500 steps to stabilize training.
3) **Epsilon-greedy Exploration**: The agent follows an epsilon-greedy policy with initial $\epsilon = 0.9$, decaying to $\epsilon = 0.02$ with a decay factor of 0.999 per episode.
4) **Smooth L1 Loss**: Instead of MSE, we used Smooth L1 Loss (Huber Loss) which is less sensitive to outliers:

$$L(y, \hat{y}) = \begin{cases} 0.5(y - \hat{y})^2, & \text{if } |y - \hat{y}| < 1 \\ |y - \hat{y}| - 0.5, & \text{otherwise} \end{cases} \quad (5)$$

5) **Learning Rate Scheduling**: We implemented a step learning rate scheduler that reduces the learning rate by a factor of 0.99 every 10 episodes.
6) **Multiple Experience Replay**: For each environment step, we performed 10 gradient updates using different minibatches from the replay buffer, improving sample efficiency.

The action selection process during training incorporates both exploration and exploitation:

### D. Training Procedure

We trained our agent on differents groups of levels, to assess the performance of our approach. The training procedure consists of the following steps:

1) Initialize the DQN agent with either the standard or ResNet-enhanced architecture.

**Algorithm 1** Action Selection in DQN Agent

**procedure** ACT($s_t$, evaluate, sample)
  **if** $random() < \epsilon$ **and** not evaluate **then**
    **return** random action
  **else**
    $Q_{values} \leftarrow$ network$(s_t)$
    **if** sample **then**
      **return** action sampled according to softmax($Q_{\text{values}}$)
    **else**
      **return** $\arg\max_a Q_{values}(s_t, a)$
    **end if**
  **end if**
**end procedure**

2) For each episode:
   - Reset the environment to get the initial state $s_0$.
   - For each timestep $t$ until episode termination:
     - Select action $a_t$ using epsilon-greedy policy.
     - Execute action $a_t$, observe reward $r_t$ and next state $s_{t+1}$.
     - Store transition $(s_t, a_t, r_t, s_{t+1}, \text{done}_t)$ in replay buffer.
     - Sample random minibatch from replay buffer.
     - Update Q-network parameters using gradient descent on the loss function.
     - Update target network every 2,500 steps.
   - Update epsilon value based on decay rate.
   - Perform evaluation on validation levels.

The reward structure provided by the environment includes:

- Small positive rewards for moving right (+1 for each step to the right)
- Larger rewards for destroying enemies (+100)
- Significant rewards for collecting coins (+100)
- Large rewards for completing a level (+1000)
- Small negative rewards for moving left (-1 for each step to the left)

### E. Experiment conducted

To evaluate the performance of our DQN implementations, we conducted the following experiments:

1) **Architecture Comparison**: We compared the standard DQN architecture against our ResNet-enhanced DQN on level 1-1.
2) **Impact of training levels**: After training on levels 1-1 to 1-4, we tested the agent's ability on seen levels (1-2, 1-3, and 1-4). We also performed training on level 1-1 to test on level 1-1

Each experiment was evaluated using the following metrics:

- **Average reward** per episode
- **Training stability** (variance in performance across episodes)

For reliable evaluation, we ran each configuration three times with different random seeds and reported the mean and standard deviation of the results.

### F. Implementation Details

Our implementation uses PyTorch for deep learning components and OpenAI Gym for the environment interface. The code structure consists of:

- `agent.py`: Defines the DQN agent with experience replay and target networks
- `models.py`: Contains network architectures (standard DQN and ResNet-enhanced DQN)
- `env.py`: Implements environment wrappers for preprocessing
- `main.py`: Orchestrates training and evaluation loops
- `record.py`: Utilities for recording agent gameplay for visualization

### G. Future Objectives

Building on the current implementation of our ResNet-enhanced DQN agent for Super Mario Bros, several promising research directions could further improve performance and understanding. This section outlines key areas for future investigation.

*1) Hyperparameter Optimization:* The current hyperparameter configuration, while effective, was chosen based on common practices and limited exploration. A more systematic approach to hyperparameter tuning could yield significant performance improvements:

- **Automated Hyperparameter Search**: Implementing Bayesian optimization or grid search to systematically explore the hyperparameter space, particularly focusing on:
  - Learning rate schedules (initial value, decay rate, and decay frequency)
  - Replay buffer capacity (10,000-100,000 transitions)
  - Batch sizes (32-256)
  - Discount factor $\gamma$ (0.9-0.99)
  - Exploration parameters (initial $\epsilon$, decay rate)
  - Number of replay for the student network
- **Sensitivity Analysis**: Conducting ablation studies to quantify the impact of each hyperparameter on performance, stability, and training speed to better understand their interactions.

*2) Network Architecture Exploration:* The ResNet-enhanced architecture shows promise, but several architectural variations warrant investigation.

*3) Policy gradient:* Policy gradient methods could be a valuable direction for future work as they offer several advantages over value-based methods like Deep Q-Networks (DQN). Instead of learning a value function to indirectly derive a policy, policy gradient methods directly optimize the policy itself, making them well-suited for high-dimensional and continuous action spaces.

Future work could involve implementing an actor-critic framework, such as Advantage Actor-Critic (A2C) or PPO, to combine the benefits of policy gradients with value-based methods. This could further enhance sample efficiency and enable the agent to adapt more effectively to different levels and gameplay scenarios.

*4) Enhanced Training Procedure:* Several modifications to the training process could improve agent performance:

- **Curriculum Learning**: Implementing progressive training stages:
  1) Begin with simple levels (1-1)
  2) Gradually introduce more complex levels with new mechanics
  3) Finally train on challenging levels requiring advanced techniques
- **Environment Diversity**: Training across a diverse set of levels to improve generalization:
  - Including underground levels with different visual features
  - Extended action set: Instead of limiting the agent to a basic subset, we could allow access to all available discrete actions in the Gym Super Mario Bros environment, including running and jumping variations.
- **Multi-objective Optimization**: Currently, the agent optimizes for a single reward signal. Future work could explore multi-objective reinforcement learning to balance:
  - Completion speed
  - Coin collection
  - Enemy elimination
  - Survival (minimizing deaths)
- **Reward Shaping**: Developing more sophisticated reward functions to guide exploration and learning:
  - Curiosity-driven exploration bonuses for visiting new areas
  - Intermediate rewards for mastering specific skills (jumping gaps, defeating certain enemies)
  - Penalties for repetitive or unproductive behavior

*5) Interpretability and Visualization:* Understanding what the agent learns would provide valuable insights:

- **Feature Visualization**: Implementing techniques to visualize what features the network attends to when making decisions.
- **Saliency Maps**: Generating attention heatmaps to show which parts of the game screen influence the agent's decisions most strongly.

These future directions will aim at improving the performance of our reinforcement learning agent on Super Mario Bros, but also contribute to our understanding of how deep reinforcement learning algorithms can be optimized for complex visual control tasks more generally.

## IV. Results and Discussion

To better visualize performance, we recorded both the training rewards and the evaluation rewards, where the latter represents the rewards obtained under the greedy policy on the training levels.

Due to computational limitations, we restricted the number of training episodes. To increase the likelihood of obtaining meaningful results, we also limited the scope of our experiments to the first four levels (World 1) of the Gym environment. The results, shown in Fig. 2, indicate extreme

instability in training and highlight the difficulty of tuning hyperparameters. One major issue contributing to this instability is likely the insufficient number of training episodes.

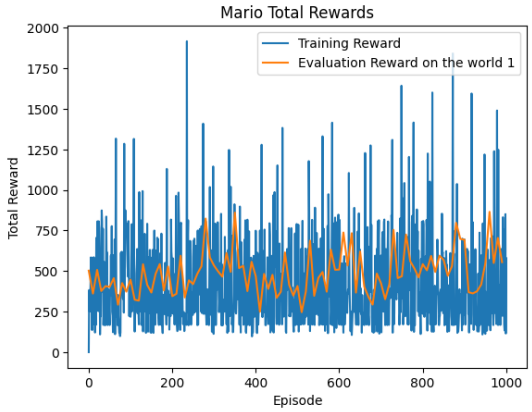


Fig. 2. Training and evaluation reward across 1000 training episodes on World 1 (ResNet-enhanced DQN)

To address this, we further constrained our experiments to a single level (the first level of the first world). This adjustment aimed to simplify the training dynamics and allow the agent to learn within a more controlled environment. However, even with this limitation, the results in Fig. 4 still exhibit high variance, suggesting that additional tuning and extended training periods are necessary for achieving stable performance.

We also analyze the impact of residual connections by comparing the performance of the two backbone architectures before (Fig. 3) and after (Fig. 4) their inclusion (note that the ResNet version is deeper). Due to the limited number of training episodes, we are unable to fully evaluate the effectiveness of residual connections at this stage. However, based on intuition and prior research on ResNets, we anticipate that their inclusion will lead to improved performance with extended training.

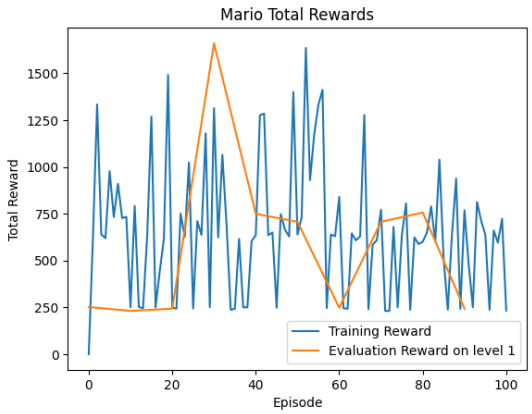


Fig. 3. Training and evaluation reward across 100 training episodes on level 1-1 (Standard DQN)

In an effort to improve hyperparameter tuning, we conducted a hyperparameter optimization using Optuna. However due to computational constraints we were only able to train

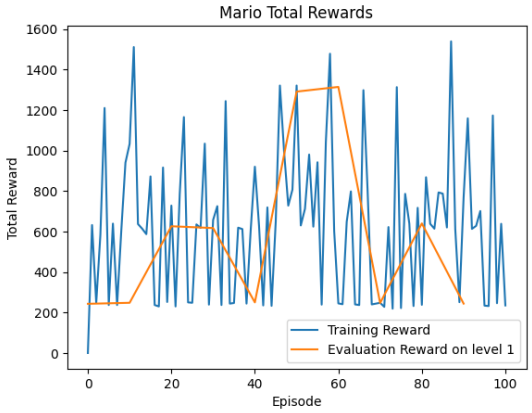


Fig. 4. Training and evaluation reward across 100 training episodes on level 1-1 (ResNet-enhanced DQN)

each configuration for five episodes and for 50 sets of HP, which was clearly insufficient to get meaningful insights. As a result the selected hyperparameters did not lead to significant improvements and the overall performance remained suboptimal.

This limitation highlights the importance of allocating sufficient training time for HPO as evaluating hyperparameter effectiveness over such a short duration is unlikely to yield reliable results. In future iterations, we aim to explore more efficient ways to optimize hyperparameters while managing computational cost.

## V. Code Availability

The code associated with this project is available on this GitHub repository .

## VI. Conclusions

This project explored deep reinforcement learning for training an agent to play Super Mario Bros using raw visual inputs. By implementing a Deep Q-Network (DQN) with residual connections, we examined the impact of architectural choices on learning stability and performance. The results highlighted the challenges of hyperparameter tuning and training stability, particularly due to computational constraints and limited episodes. While the agent demonstrated some learning capability, performance remained inconsistent.

## References

[1] C. Kauten, "Super Mario Bros for OpenAI Gym," GitHub, 2018. [Online]. Available: https://github.com/Kautenja/gym-super-mario-bros

[2] V. Mnih, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb 2015. [Online]. Available: https://doi.org/10.1038/nature14236

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1512.03385

[4] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," 2015. [Online]. Available: https://arxiv.org/abs/1509.06461

[5] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2016. [Online]. Available: https://arxiv.org/abs/1511.05952

[6] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," 2017. [Online]. Available: https://arxiv.org/abs/1707.06887