**Image Understanding Pipeline and Reinforcement Learning Design**

**Abhay Prabhakar**

**Team-8**

**Date: April 28, 2025**

**Table of Contents**

**(Generate this in Word after pasting content)**

**1. Project Overview & Key Decision**

This project encompassed two primary tasks focused on core concepts in modern Artificial Intelligence: Part A involved developing an image understanding pipeline using deep learning, while Part B focused on the conceptual design of a Reinforcement Learning (RL) strategy for game playing.

The initial plan for Part A required training a Generative Adversarial Network (GAN) on the MS COCO dataset to generate images, followed by object detection and image captioning on those outputs. However, **a key decision during the project was to adapt the Part A methodology.** Initial attempts to train the GAN model (detailed in Section 2.3.1) yielded images lacking sufficient realism for subsequent tasks within the project's constraints **(See Appendix A)**. Therefore, aligning with instructor guidance allowing alternatives, a **fallback strategy** was adopted: five diverse, pre-existing images were selected as inputs for a refined pipeline focused solely on object detection and image captioning using established pre-trained models.

Part B involved the theoretical design of an RL agent, without implementation, exploring the required components like states, actions, rewards, network architecture, and training procedures for the classic game Pong. This report details the methodologies, results, and design choices for both project parts.

**2. Part A: Image-Based Deep Learning Tasks**

**2.1. Task Overview**

The objective of Part A was to construct and evaluate a pipeline performing sequential computer vision tasks: image generation, object detection, and image captioning. This provided practical experience with common deep learning applications in image understanding.

**2.2. Image Dataset: MS COCO**

The designated dataset for the initial image generation training attempt was the Microsoft Common Objects in Context (COCO) 2017 Train Images set (train2017.zip), accessed via the official website [1]. COCO provides a large-scale benchmark with diverse images and annotations suitable for training vision models. The project plan involved using **a subset of images** (~10k were processed in the attempt) from this dataset for the GAN training attempt, and the necessary train2017.zip file (~18GB) was successfully downloaded into the Google Colab environment. However, as the GAN training did not yield usable results (see Section 2.3.1), this downloaded training data was not directly utilized for the final object detection and captioning results, which were generated using the 5 fallback images. The use of pre-trained models implicitly leverages knowledge gained from large datasets like COCO during their original training.

**2.3. Methodology and Models**

**2.3.1. Image Generation (GAN Attempt & Fallback Justification)**

The first stage aimed to generate 5 images using a GAN. A standard DCGAN architecture was implemented in PyTorch. The downloaded COCO train2017 images (a subset of ~10k) were prepared using a standard pipeline (unzipping, resizing to 64x64, transforms, DataLoader). The model was trained using BCE loss and Adam optimizers for 15 epochs on an A100 GPU via Google Colab. Sample images generated after each epoch showed a lack of convergence towards realism, appearing abstract and noisy **(See Appendix A for example generated images)**. This difficulty justified the decision to employ the **fallback strategy**, using 5 manually selected images for the subsequent pipeline stages.

**2.3.2. Object Detection (YOLOv8n)**

For detecting objects in the 5 fallback images, the pre-trained YOLOv8n (nano) model was used via the ultralytics library [2]. This model, already trained on the COCO dataset, was chosen for its speed, efficiency, and ease of use via the library, providing a strong baseline for identifying common object classes without further training. The implementation involved loading the model onto the GPU (cuda device), validating each input image using PIL, and running inference using model.predict(image\_path, device=device). The YOLOv8n model provides a confidence score for each detection; only detections exceeding a default internal threshold (typically around 0.25 or 25%) are usually returned and displayed. The ultralytics pipeline handles internal preprocessing. This process yielded images annotated with bounding boxes, class labels, and confidence scores, which were saved, and object counts were extracted.

**2.3.3. Image Captioning (BLIP-base)**

To generate descriptive captions, the pre-trained Salesforce/blip-image-captioning-base model was employed through the Hugging Face transformers library [3]. BLIP integrates vision understanding and language generation effectively, offering strong out-of-the-box captioning performance. The implementation explicitly loaded the BlipProcessor and BlipForConditionalGeneration model onto the GPU (cuda). For each validated image, the processor prepared the input, and model.generate() produced the output token sequence, decoded into the final English caption.

**2.4. Results and Discussion**

The refined pipeline (detection and captioning on fallback images) successfully processed all 5 selected input images. The following sections present the visual results and generated outputs for each image. Note that the detection confidence threshold was the default used by the YOLOv8n implementation (approx. 0.25); values below this indicate uncertain detections that might be filtered visually or reflect model limitations.

**Image 1: image 1 object detection.jpg**

The original input image containing stationary bikes is shown in **Figure 2.1**.

**Figure 2.1:** Original Image 1 (Stationary Bikes)

A group of exercise bikes in a gym

AI-generated content may be incorrect.

* **Detection Output:** The object detection results from YOLOv8n are presented in **Figure 2.2**.  
  **Figure 2.2:** YOLOv8n detections for Image 1 (bicycle: 30%, motorcycle: 25%)

A room with exercise bikes

AI-generated content may be incorrect.

Detected Objects: {'bicycle': 1, 'motorcycle': 2}

* **Generated Caption:** 'a row of stationary bikes in a gym'

**Image 2: image 2 object detection.jpg**

The second input image, featuring a girl in a carriage, is displayed in **Figure 2.3**.

**Figure 2.3:** Original Image 2 (Girl in Carriage)



* **Detection Output:** **Figure 2.4** shows the detections found by YOLOv8n for the second image.  
  **Figure 2.4:** YOLOv8n detections for Image 2 (person: 51%, tv: 27%)
  + A child in a carriage

    AI-generated content may be incorrect.

Detected Objects: {'person': 1, 'tv': 1}

* **Generated Caption:** 'a little girl sitting in a horse drawn carriage'

**Image 3: image 3 object detection.jpg**

The third input image, depicting a tall tower, is presented in **Figure 2.5**.

**Figure 2.5:** Original Image 3 (Tower)

* A tall tower with a black dome

  AI-generated content may be incorrect.

**Detection Output:** **Figure 2.6** illustrates the output from YOLOv8n. No objects were detected above the confidence threshold.  
**Figure 2.6:** YOLOv8n detections for Image 3 (No objects detected)

A tall tower with a black dome

AI-generated content may be incorrect.

Detected Objects: {}

* **Generated Caption:** 'a tall tower with a clock on top'

**Image 4: image 4 object detection.jpg**

The fourth input image, showing a bus on a road, is presented in **Figure 2.7**.

**Figure 2.7:** Original Image 4 (Bus)

* A white bus with buffalo images on it

  AI-generated content may be incorrect.

**Detection Output:** **Figure 2.8** displays the detections generated by YOLOv8n.  
**Figure 2.8:** YOLOv8n detections for Image 4 (truck: 71%, person: 50%, sheep: 82%, bus: <70% implied)

A bus with sheep on the side

AI-generated content may be incorrect.

Detected Objects: {'sheep': 1, 'truck': 1, 'bus': 1, 'person': 1}

* **Generated Caption:** 'a bus parked on the side of the road'

**Image 5: image 5 object detection.jpg**

The final input image, featuring a bird, is shown in **Figure 2.9**.

**Figure 2.9:** Original Image 5 (Bird)

A bird on a branch

AI-generated content may be incorrect.

**Detection Output:** The detection result from YOLOv8n is shown in **Figure 2.10**.  
**Figure 2.10:** YOLOv8n detections for Image 5 (bird: 74%)

A bird on a branch

AI-generated content may be incorrect.

Detected Objects: {'bird': 1}

* **Generated Caption:** 'a bird sitting on top of a tree branch'

**Discussion:**  
As shown in Figures 2.2 through 2.10, the YOLOv8n model successfully identified primary subjects in 4 out of 5 images, including 'person' (51% conf., Fig 2.4; 50% conf., Fig 2.8), 'truck' (71% conf., Fig 2.8), 'bus' (Fig 2.8), and 'bird' (74% conf., Fig 2.10). No objects were detected above threshold in the 'tower' image (Fig 2.6). Limitations were evident: the model misclassified stationary bikes (Fig 2.2, 'bicycle': 30%, 'motorcycle': 25%) and part of the carriage ('tv': 27%, Fig 2.4). The detection of the 'sheep' graphic (82% conf., Fig 2.8) highlights pattern recognition capabilities but also potential confusion on complex surfaces. These results demonstrate expected behavior for a lightweight pre-trained detector; high confidence on clear, common objects, and lower confidence or misclassifications on ambiguous or less common items.  
The BLIP-base captioning model generated relevant and coherent captions for all images, effectively summarizing the visual content (e.g., "a little girl sitting in a horse drawn carriage," "a tall tower with a clock on top").

**2.5. Limitations & Future Work (Part A)**

The primary limitation was the inability to generate high-quality GAN images within the project scope **(Appendix A)**. The detection results, while identifying main subjects, could be improved regarding confidence and avoiding misclassifications on ambiguous objects. **Future work** could involve using a larger model like YOLOv8s, increasing the confidence threshold (e.g., to 0.40) to reduce noise, or exploring models trained on broader datasets. For captioning, testing the BLIP-large variant could provide richer textual descriptions.

**3. Part B: Reinforcement Learning Strategy Design (Pong)** **A black and white screen with numbers

AI-generated content may be incorrect.**

**3.1. Game Description**

* **Game:** Classic Pong
* **Rationale for Choice:** Pong was selected due to its simple yet dynamic nature, offering a clear objective, a well-defined and low-dimensional state space, a small discrete action space, and relatively straightforward reward mechanisms, making it an ideal environment for illustrating core RL concepts without excessive complexity.
* **Objective:** Control a vertical paddle to intercept a moving ball, preventing it from passing the player's side of the screen while attempting to make the ball pass the opponent's paddle to score points. Typically played until one player reaches a target score.
* **Environment:** 2D rectangular screen. Two paddles move vertically along the left and right edges. A ball moves horizontally and vertically, bouncing realistically off the top and bottom walls, and changing vertical direction based on where it hits a paddle.

**3.2. States (S), Actions (A), and Rewards (R)**

* **States (S):** The state was represented by a vector containing:
  + Player Paddle Y Position: Vertical coordinate of the player's paddle center.
  + Ball X Position: Horizontal coordinate of the ball.
  + Ball Y Position: Vertical coordinate of the ball.
  + Ball X Velocity: Ball's horizontal speed and direction.
  + Ball Y Velocity: Ball's vertical speed and direction.
  + *(Optional)* Opponent Paddle Y Position: Vertical coordinate of opponent paddle.
* **Actions (A):** The agent's discrete actions were:
  + Move Up: Move the paddle upwards.
  + Move Down: Move the paddle downwards.
  + Stay Still: Keep the paddle stationary.
* **Rewards and Penalties (R & P):**
  + Reward (+1.0): When the player scores a point (opponent misses).
  + Penalty (-1.0): When the opponent scores a point (player misses).
  + Shaping Reward (+0.1): (Optional) Small positive reward for hitting the ball.
  + Other Events (Reward = 0.0): Neutral outcome for wall bounces.

**3.3. Modeling Hypothesis**

* **State Representation:** A feature vector containing the 5-6 normalized state variables.
* **Action Selection Rule:** A Deep Q-Network (DQN) would estimate Q-values for each action based on the state. An epsilon-greedy strategy during training would balance exploration (random action) and exploitation (action with highest Q-value).
* **State/Reward Distribution:** State transitions follow deterministic physics. Rewards are sparse, primarily occurring at the end of a rally.

**3.4. Neural Network Structure**

* **Model Type:** Deep Q-Network (DQN).
* **Rationale:** Suitable for continuous state spaces and discrete actions common in classic games like Pong.
* **Architecture:** Multi-Layer Perceptron (MLP).
  + **Input Layer:** ~6 units (normalized state features).
  + **Hidden Layers:** Two Dense layers (e.g., 64-128 neurons each) with ReLU activation.
  + **Output Layer:** Dense layer with 3 Linear output neurons (Q-values for Up, Down, Still).

**3.5. Reward Strategy**

* **Algorithm:** Deep Q-Learning (DQN) with Experience Replay and a Target Network.
* **Objective:** Maximize the discounted sum of future rewards (maximize score difference).
* **Discount Factor (gamma):** ≈ 0.99 (highly value future points).
* **Exploration:** Epsilon-greedy, annealing epsilon from 1.0 down to ≈0.05-0.1 over training.
* **Experience Replay:** Buffer size ≈ 50k-100k transitions, sampling mini-batches for updates.
* **Target Network:** Update target Q-network periodically (e.g., every 1k-10k steps) to stabilize learning.

**3.6. Analysis Framework**

* **Training Procedure Summary:** The agent learns via the standard DQN loop with experience replay (buffer size: ≈50k-100k), target network updates (e.g., every 1k-10k steps), and ε-decay (e.g., from 1.0 to 0.05) to balance exploration and exploitation. *(See Appendix B for potential detailed step-by-step pseudocode)*.
* **Evaluation Metrics:** Track performance via Average Score per Episode, Win Rate, and Average Rally Length.
* **Visualization:** Plot learning progress (e.g., score vs. episodes).

**4. Conclusion**

This project successfully addressed the core requirements for both image-based deep learning and reinforcement learning design. Part A demonstrated the construction of an image processing pipeline, including a documented attempt at GAN-based image generation **(Appendix A)** which justified adopting a fallback strategy. The subsequent pipeline using pre-trained YOLOv8n and BLIP models effectively performed object detection and image captioning **(Section 2.4)**, providing plausible results while highlighting model limitations. Part B presented a comprehensive theoretical design for an RL agent to play Pong, detailing the necessary states, actions, rewards, DQN architecture, and conceptual training framework. The project highlighted both the power of leveraging pre-trained models and the design principles of RL. ***Future work could involve implementing the Pong RL agent or exploring more advanced models and fine-tuning techniques for the Part A pipeline.***

**Appendix A: Sample GAN Generated Images**

**Figure A.1:** Sample Images Generated by DCGAN after 15 Epochs of Training on COCO Subset (Illustrative of convergence issues).

A screenshot of a computer screen

AI-generated content may be incorrect.

**Appendix B: Detailed RL Training Procedure (Pong)**

1. Initialize main Q-network and target Q-network with identical random weights.
2. Initialize experience replay buffer (e.g., size 50,000).
3. Loop for N episodes (e.g., N = 10,000):
4. Reset Pong environment, get initial state S.
5. Loop until episode terminates (e.g., score limit reached):
   * Select action A using epsilon-greedy policy based on Q\_main(S).
   * Execute action A, observe reward R and next state S'.
   * Store transition (S, A, R, S') in replay buffer.
   * Update S = S'.
   * Sample a mini-batch of M transitions from the replay buffer (e.g., M=64).
   * Calculate target Q-values for each transition (s, a, r, s') in the batch:
     + If s' is a terminal state, target = r.
     + Otherwise, target = r + gamma \* max\_a'(Q\_target(s', a')).
   * Calculate loss (e.g., Huber or MSE) between predicted Q\_main(s, a) and target for the batch.
   * Perform gradient descent step on main Q-network parameters using the calculated loss.
   * Periodically (e.g., every C steps), update target network weights: Target\_Net\_Weights = Main\_Net\_Weights.
   * Decay epsilon according to annealing schedule.
6. End of inner loop (episode finished).
7. End of outer loop (training finished).