**Image-Based Deep Learning & Reinforcement Learning Strategy Design**

Pamela Alvarado-Zarate

Mark Hite

Harshal Kamble

Vishnu Sankhyan

Venkatesh Subramony

# **Part A:** Image-Based Deep Learning

# **Task Overview**

## **Image Generation**

The objective of the image generation task is to train a Generative Adversarial Network (GAN) model on randomly selected images to produce fake images that capture and mimic the likeness of real data.

## **Object Detection**

## **Image Captioning**

The purpose of the image captioning task is to generate captions by using a pre-trained convolutional neural network encoder (e.g., ResNet) and a recurrent neural network decoder to represent the extracted visual features as a sequence of words (a.k.a. caption).

# **Image Dataset**

For the image generation task, the Common Objects in Context (COCO) dataset was used. COCO is a large-scale dataset containing images of people, animals, and everyday objects. We extracted data from the 2017 Train Images set, which consists of roughly 18GB of images. However, to reduce computation cost, we selected a random subset of 100,000 images to train the GAN model.

# **Model Design/Architecture**

## **Image Generation**

The GAN model consists of two main components: the Generator and the Discriminator. The two are trained in an adversarial manner, where the Generator attempts to produce realistic images while the Discriminator tries to distinguish between real and synthetic images. The goal is to train the Generator to the point where it can successfully generate images that are realistic enough to fool the Discriminator.

The Generator accepts a noise vector with shape (100, 1, 1) as input. It is passed through 5 convolutional layers, which upsample the input to progressively larger spatial dimensions until the desired image size (64x64) is reached. Each convolutional layer (besides the last) is followed by batch normalization, which helps stabilize training and reach convergence. The Generator uses ReLU activation functions in the first 4 layers to introduce nonlinearity and allow the model to capture complex patterns. In the last layer, a Tanh activation function is used to convert output values to be in the range [-1, 1], matching the normalized pixel range of the dataset.

The Discriminator accepts an input vector with shape (3, 64, 64), corresponding to a 64x64 image with RGB color. The input is passed through 5 convolutional layers to downsample the image and extract features. Because it is important for the Discriminator to extract features, it relies on Leaky ReLU activation, which, unlike ReLU, allows some representation for negative inputs. The final layer uses a Sigmoid activation function to convert the output to a single value in the range [0, 1] predicting whether the image is real (close to 1) or fake (close to 0).

Since the training images can be different sizes, we transformed our random subset before passing them to the model. Images were resized to 64x64 resolution, center cropped, converted to a PyTorch tensor, and the tensor values were normalized to be in the range [-1, 1].

For training, the model uses a learning rate of 0.0002, which is a safe choice that balances training speed and stability. The Adam optimizer is used with binary cross-entropy (BCE) loss. BCE is used since we are essentially training a binary classifier. We tested with various combinations of training images, image sizes and number of epochs, but the combination yielding the best result was 100,000 64x64 images trained over 20 epochs.

## **Object Detection**

The object detection part of our project utilized the YOLOv5s model, a lightweight one-stage detector designed for fast and accurate object recognition. The model was pre-trained on the COCO dataset, which contains 80 object classes including common categories like animals, vehicles, and everyday objects.

Our object detection pipeline began by uploading five different images into Google Colab, covering diverse scenes such as animals, cars, and urban settings. No additional training or fine-tuning was performed; instead, inference was directly run using the pre-trained YOLOv5s model. This allowed us to evaluate the model's out-of-the-box generalization to unseen data.

Each input image was automatically resized by the YOLOv5 pipeline without manual intervention. The model processed images through its standard architecture:

* A **CSPNet**-based backbone extracted hierarchical feature representations.
* A **PANet neck** aggregated features at multiple scales.
* A **detection head** predicted bounding boxes, class labels, and confidence scores.

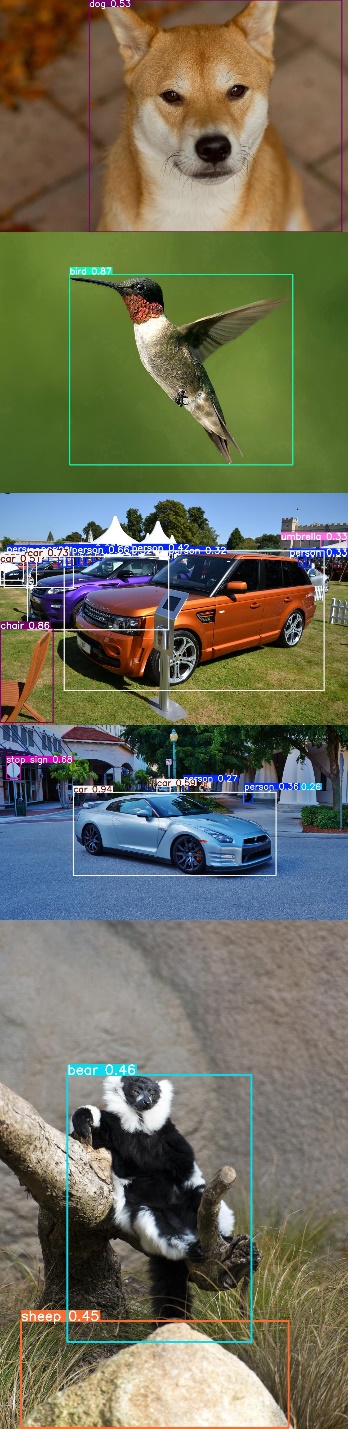
The detection threshold was set to 0.25, meaning only predictions with at least 25% confidence were considered valid. Non-maximum suppression (NMS) was applied to remove redundant overlapping detections, ensuring clean outputs with minimal duplication.

The object detection results were as follows:

* In the **dog image**, the model correctly identified the dog with a confidence score of 0.53.
* In the **bird image**, a hummingbird was detected accurately with a high confidence score of 0.87.
* In the **car exhibition image**, multiple cars, people, a chair, and even an umbrella were detected, showcasing YOLOv5's multi-object capability.
* In the **urban street image**, the car was detected with a very high confidence of 0.94, along with a nearby stop sign and several pedestrians.
* In the **lemur image**, some misclassification occurred: the model labeled the lemur as a "bear" and a foreground rock as "sheep," reflecting known limitations of pre-trained models on non-COCO object categories.

The full detection flow can be summarized:

* **Input:** Uploaded 5 images of varying resolutions.
* **Preprocessing:** Automatic resizing inside YOLOv5 inference.
* **Model:** Pre-trained YOLOv5s with COCO weights.
* **Output:** Annotated images with bounding boxes and class labels based on detection confidence.



Overall, the YOLOv5s model performed well for standard objects such as cars, animals, and people, but struggled with unusual or non-standard objects outside its original training distribution. The fast inference time, simplicity of setup, and reasonable accuracy made YOLOv5s an ideal choice for this project phase.

## **Image Captioning**

The image‐captioning model follows an encoder–decoder paradigm with a convolutional backbone for visual feature extraction and a recurrent architecture for language generation. Hyperparameters via a Keras tokenizer were implemented to restrict the number of generated tokens to 34 tokens (MAX\_LENGTH) and a vocabulary to the top 5,000 common words (VOCAB\_SIZE). Any rare or unfamiliar words are assigned as a <unk> token. Special tokens of <start> and <end> were included as part of the sequence length and markers.

The convolutional backbone network consisted of a ResNet-50 model pretrained on ImageNet for visual feature extraction. Characteristics of the network include the removal of the final classification layers and resizing each image to 224x224 pixels. The model outputs a single 2048-dimensional feature vector to be representative of the image by using the global average pooling operation.

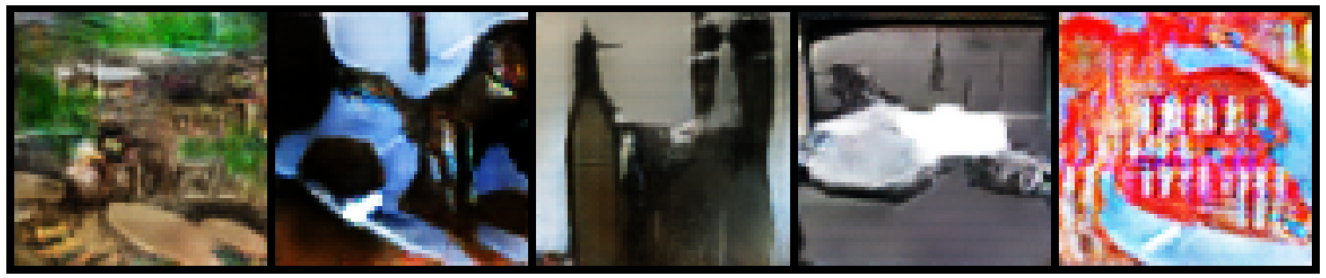
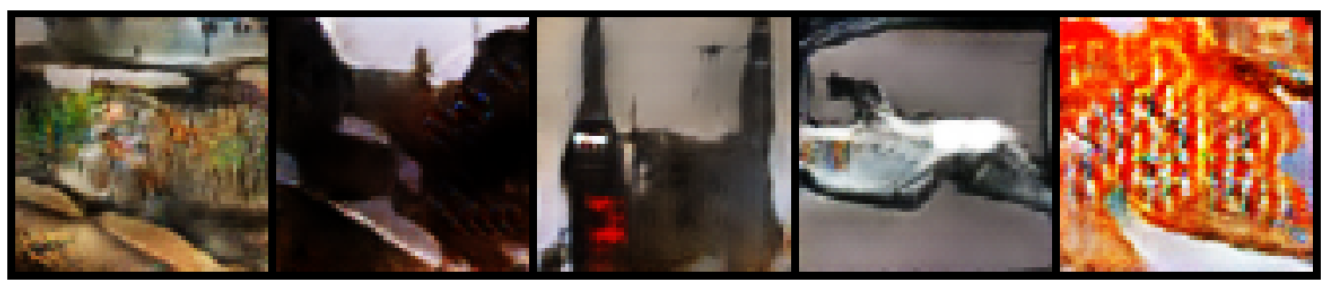
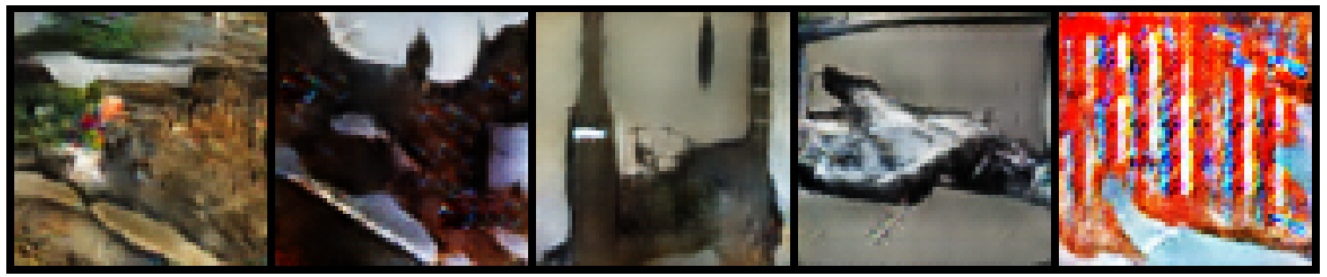
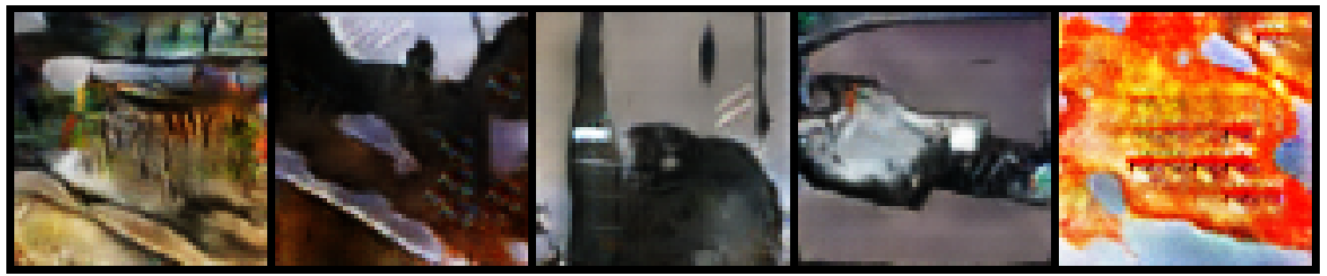
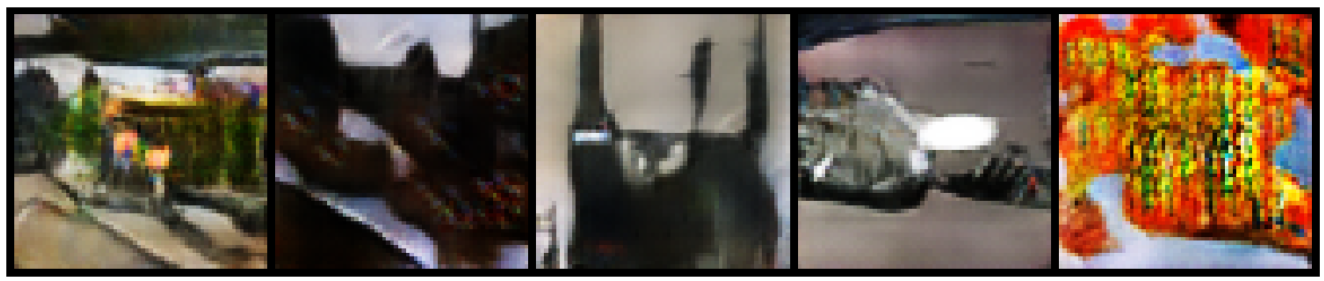
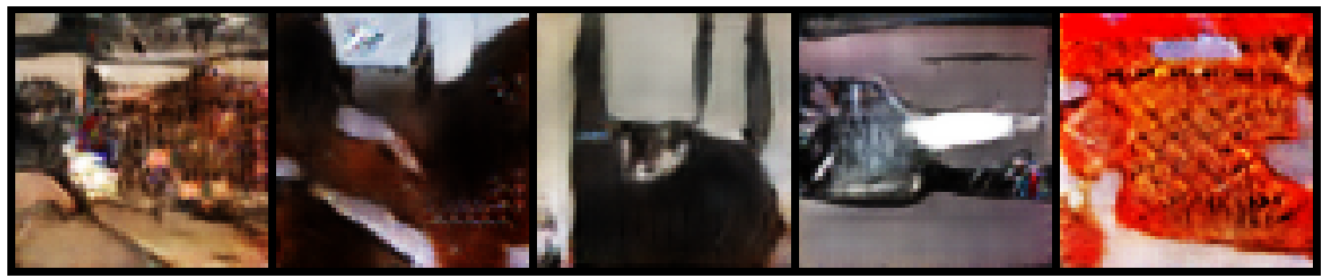
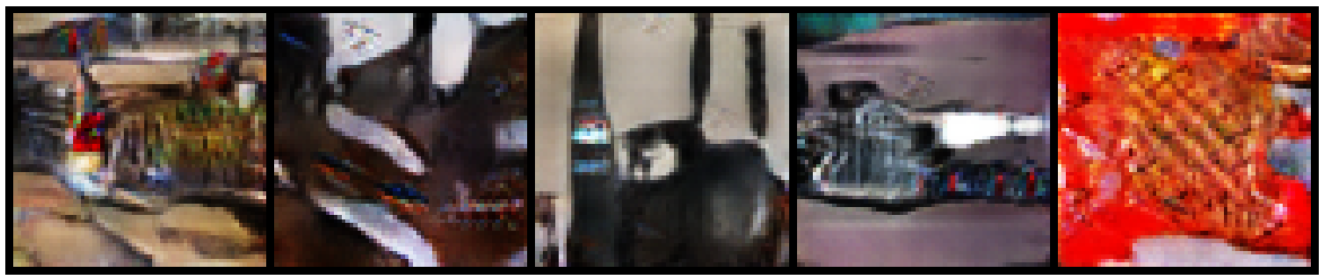
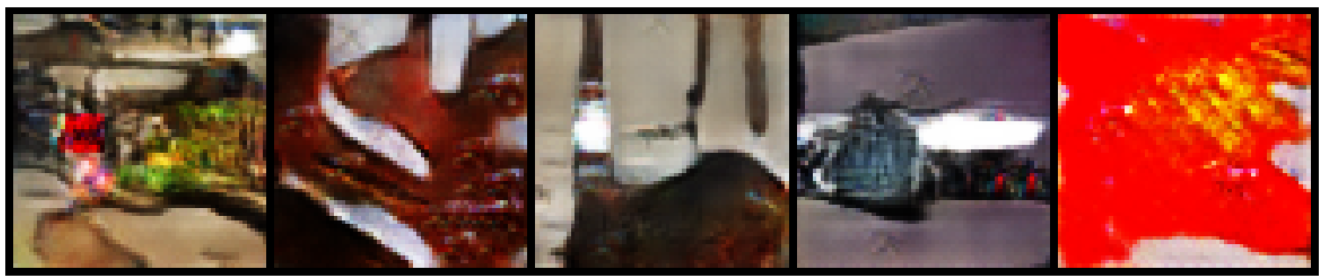
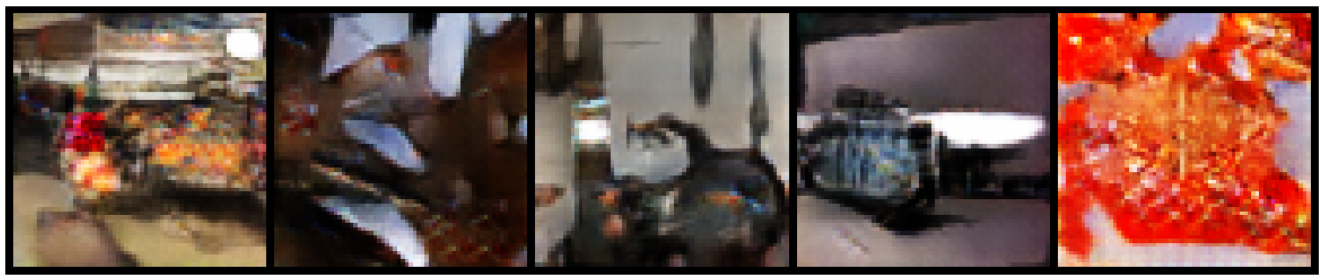
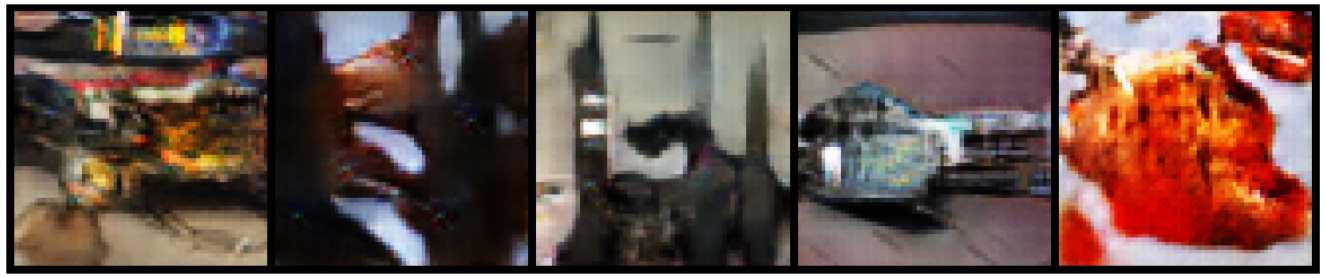
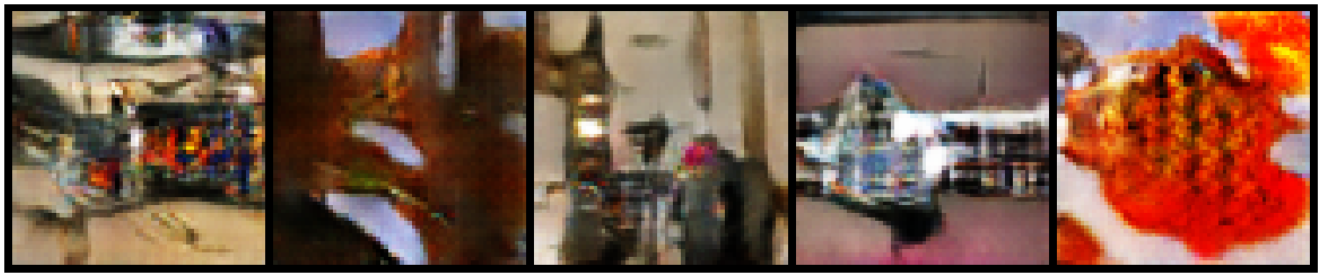
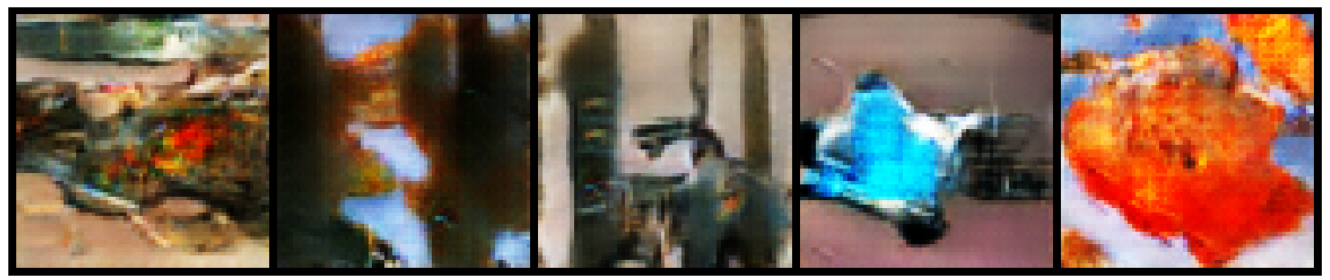
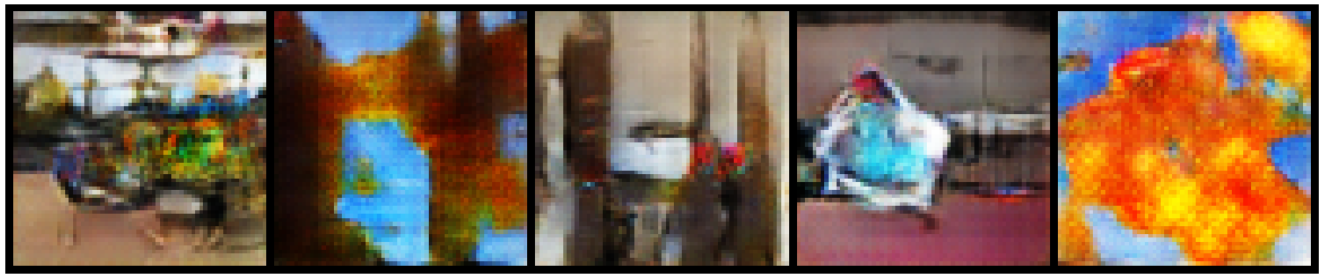
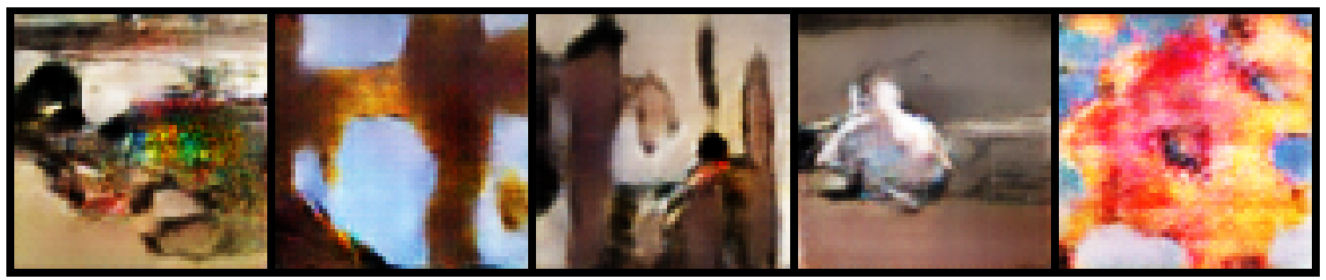
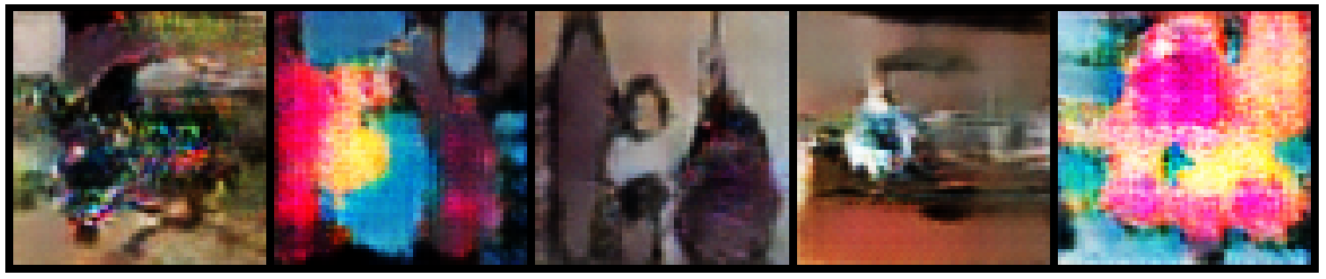
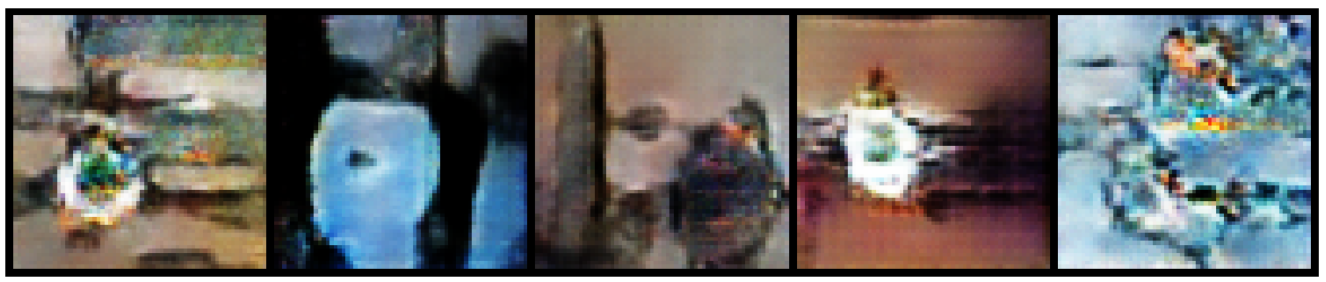
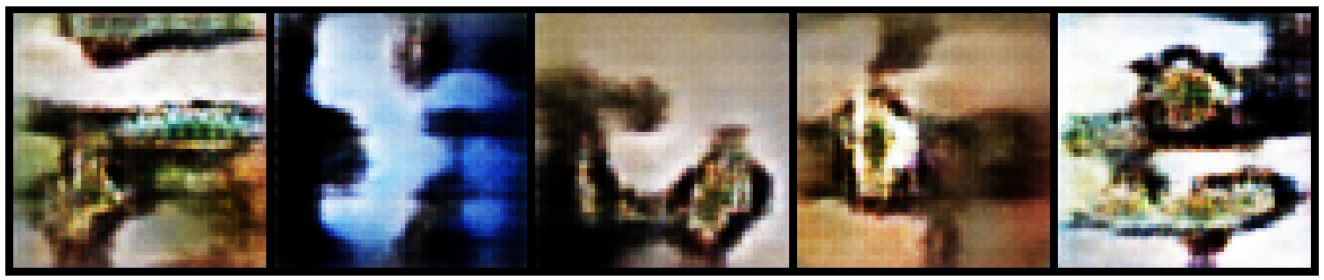
The recurrent architecture consists of a custom recurrent neural network and a Long Short-Term Memory (LSTM) layer with 512 hidden units. The RNN decoder receives and expands the 2048-dimensional feature vector along the time axis and replicates it to match the current caption length the model to visualize the image for each predicted token. Tokens embedded into a 256-dimensional vector are received as well. Within each process, the LSTM model outputs a sequence of hidden states, which are passed through two dense layers. The first dense layer maintains linearity, whereas the second layer produces raw logits over the 5,000-word vocabulary.

Teacher forcing is utilized for training, where each ground-truth token sequence guides the decoder to predict the next token. Sparse categorical cross-entropy as loss is selected. Parameter updates are carried out by using the Adam optimizer. Sixty-four (64) mini-batches over 30 epochs are sampled randomly with the average batch loss for each epoch logged to monitor convergence. The decoder’s learned weights and fitted tokenizer are saved afterward.

For inference, caption generation runs in a step-by-step loop. A feature vector is extracted from each image, where the caption begins with the <start> token. The decoder tokenizes and pads the current sequence, outputs logits for the next token, and selects the highest‐probability word. The process continues until either the <end> token is generated or the maximum sequence length is reached. The final caption is returned without the special tokens.

# **Results**

## **Image Generation**



## **Image Captioning**

**Image 1**

* **Generated Caption:** a close dog dog dog dog dog dog dog dog dog dog dog dog dog dog dog dog dog dog dog dog dog dog dog dog dog dog dog dog dog dog dog dog
* **Ground-Truth**

1. <start> a close-up portrait of a tan and white dog with perked ears looking at the camera <end>
2. <start> a fox-like dog sitting on a stone patio gazing forward <end>
3. <start> a golden canine with erect ears and a black nose on paving stones <end>
4. <start> a shiba inu style dog with warm russet fur staring at the lens <end>
5. <start> a medium-sized dog with white chest and alert expression on tiled ground <end>

**Image 2**

* **Generated Caption:** a male wings hummingbird hovering hummingbird hovering hummingbird hovering hummingbird hovering hummingbird hovering hummingbird hovering hummingbird hovering hummingbird hovering hummingbird hovering hummingbird hovering hummingbird hovering hummingbird hovering hummingbird hovering hummingbird hovering hummingbird hovering hummingbird

1. <start> a ruby-throated hummingbird hovering in front of a green background <end>
2. <start> a small hummingbird with iridescent red throat and outstretched wings mid-flight <end>
3. <start> a tiny bird with long beak flapping its wings in midair <end>
4. <start> a close-up of a hovering hummingbird showing blurred wings <end>
5. <start> a male hummingbird suspended in flight against a soft green backdrop <end>

**Image 3**

* **Generated Caption:** a manicured orange orange orange orange orange orange orange orange orange orange orange orange orange orange orange orange orange orange orange orange orange orange orange orange orange orange orange orange orange orange orange orange
* **Ground-Truth**

1. <start> an orange Range Rover Evoque parked on grass at a car show <end>
2. <start> a vivid orange SUV displayed beside a purple crossover under white tents <end>
3. <start> two sporty SUVs, one orange and one purple, lined up on a manicured lawn <end>
4. <start> a bright orange luxury crossover with polished wheels on the grass <end>
5. <start> an orange Range Rover model exhibited outdoors with onlookers <end>

**Image 4**

* **Generated Caption:** a city silver r silver silver silver silver silver silver silver silver silver silver silver silver silver silver silver silver silver silver silver silver silver silver silver silver silver silver silver silver silver silver
* **Ground Truth Captions**
  1. <start> a sleek silver Nissan GT-R parked on a city street in front of columns <end>
  2. <start> a metallic silver sports car with black wheels parked on asphalt <end>
  3. <start> a high-performance coupe under leafy trees on a downtown road <end>
  4. <start> a low-slung silver GT-R near white arched architecture <end>
  5. <start> a luxury performance vehicle parked by a plaza with pillars <end>

**Image 5**

* **Generated Caption:** a fluffy lemur end end end end end end end end end end end end end end end end end end end end end end end end end end end end end end end
* **Ground Truth Captions**
  1. <start> a black-and-white ruffed lemur sitting on a tree branch with a fluffy collar <end>
  2. <start> a lemur with tufted ears lounging against a log in its enclosure <end>
  3. <start> a primate with monochrome fur perched on a branch looking away <end>
  4. <start> a relaxed black-and-white lemur draped over a wooden limb <end>
  5. <start> a fluffy-collared lemur casually resting on a tree branch <end>

# **Part B:** Reinforcement Learning Strategy Design

**Video Game:** 2D Street Racer (Top-Down Endless Racing Game)

### **1. Game Description**

* **States (S):**
  + Player’s lane position (discrete lanes: left, center, right)
  + Positions of obstacles (relative distance and lane of other cars)
  + Positions of coins (location and lane)
  + Player’s speed (discrete or continuous value)
  + Distance traveled (cumulative progress in the game)
  + Fuel level or energy (optional extension for more complex behavior)
* **Actions (A):**
  + Move Left (shift to adjacent left lane if possible)
  + Move Right (shift to adjacent right lane if possible)
  + Accelerate (increase speed within limits)
  + Decelerate (decrease speed to avoid obstacles)
  + Do Nothing (maintain current lane and speed)
* **Rewards (R):**
  + +1 for collecting a coin (encourages risk-taking and exploration)
  + +5 for every 100 meters traveled (encourages survival and long-term planning)
  + -10 for colliding with another car (severe penalty for unsafe driving)
  + -1 for moving out of the road boundary (penalty for losing control)
* **Penalties:**
  + Game ends immediately upon collision or moving completely off-road.

### **2. Modeling Hypothesis**

* **Action Rules:**
  + The agent will choose actions based on the current state using a probability distribution π(a|s) derived from estimated Q-values.
  + Early training favors exploration to discover effective strategies, while later training focuses on exploiting learned knowledge.
* **State/Reward Distribution:**
  + Movement of other vehicles and appearance of coins are stochastic and unpredictable, introducing randomness into the environment.
* **Neural Network Structure:**
  + **Input Layer:** Vectorized environment features (positions, speeds, coin locations) or a processed 2D visual snapshot of the road.
  + **Hidden Layers:**
    - Dense layer (64 neurons, ReLU activation)
    - Dense layer (64 neurons, ReLU activation)
  + **Output Layer:**
    - Linear layer with one neuron per action, outputting Q-values for each possible action.
* **Alternative (Image-based Input):**
  + Convolutional layers to extract spatial relationships from top-down images:
    - Conv2D(32 filters, 3x3 kernel, ReLU)
    - Conv2D(64 filters, 3x3 kernel, ReLU)
    - Flatten
    - Dense(128 neurons, ReLU)
    - Output layer (Linear activation)
* **Why We Chose This Model:**
  + A Deep Q-Network (DQN) effectively handles high-dimensional input spaces where simple tabular Q-learning would fail.
  + Fully connected layers are sufficient for structured state inputs, while CNN layers allow processing raw pixel data efficiently.
  + ReLU activations are chosen for faster convergence and to avoid vanishing gradients.
  + A linear output layer aligns with the Q-learning objective of estimating real-valued action-value functions.

### **3. Reward Strategy**

* **Primary Reward Focus:**
  + Maximize total rewards by surviving longer and collecting coins efficiently.
  + Introduce a strong deterrent against reckless driving (heavy collision penalty).
* **Cumulative Rewards:**
  + The agent should not just chase immediate rewards (coins) but plan for longer survival.
  + Future rewards are discounted by a factor γ = 0.95 to prioritize near-future safety but still account for long-term gains.
* **Why Use Discounted Rewards:**
  + Immediate small gains (coins) should not outweigh the value of survival.
  + Discounting future rewards encourages the agent to think strategically rather than reactively.

### **4. Analysis Framework**

**Step-by-Step Procedure:**

1. **Initialization:**
   1. Start from an initial safe state: center lane, moderate speed, clear surroundings.
   2. Initialize Q-network with random weights.
2. **Action Selection:**
   1. At each time step, select an action based on an ε-greedy policy:
      1. High ε (e.g., 1.0) initially to explore various strategies.
      2. Gradually decay ε to 0.1 as the agent becomes better.
3. **Environment Interaction:**
   1. Execute the selected action.
   2. Observe the new state (next position, updated environment) and received reward.
4. **Experience Storage:**
   1. Store each (state, action, reward, next state) tuple into a replay buffer for training.
   2. The replay buffer helps to break correlations between sequential experiences.
5. **Learning Phase:**
   1. Randomly sample mini-batches from the replay buffer.
   2. Update the Q-network by minimizing the loss:
   3. Use gradient descent to adjust network weights.
6. **Policy Improvement:**
   1. Regularly evaluate performance and retrain if agent behavior stagnates.
   2. Optionally, use a target network updated at fixed intervals to stabilize learning.
7. **Evaluation Metrics:**
   1. Average Distance Traveled per Episode
   2. Total Coins Collected per Episode
   3. Number of Collisions (lower is better)
   4. Average Cumulative Reward
   5. Stability of Q-values over training epochs
8. **Termination Criteria:**
   1. Training can stop once the agent consistently achieves a high average distance without frequent collisions.

### **Summary**

In this strategy, the AI learns to prioritize survival and strategic coin collection while adapting to increasingly unpredictable traffic patterns. The agent balances short-term actions with long-term rewards using discounted future rewards and improves steadily through experience replay and Q-network optimization.

By choosing a simple but powerful Deep Q-Network architecture, we ensure the model is capable of generalizing across different states and handling both structured feature vectors and raw visual input.