Scene Manipulation Using Deep Learning: From Object Detection to Intelligent Image Editing

Student Name

June 14, 2025

1 Introduction and Initial Approach

At the start of this project, I felt somewhat confident since I had used YOLO before through Ultralytics. I had seen how it detected objects using bounding boxes and assumed that scene manipulation would be a matter of detecting and moving things around.

That assumption didn't last long.

I began by writing basic YOLO detection code and tried editing images directly using those bounding boxes. The results were extremely poor — objects would move, but the edits looked unrealistic and broken. It became clear that simply detecting where things are isn't enough. Scene manipulation required understanding object boundaries, context, and how to reconstruct backgrounds — in other words, it needed a full pipeline.

2 Object Detection

Even though I'd used YOLO before, integrating YOLOv8 into a custom setup taught me new things. The detection results weren't always reliable, especially when using fixed thresholds. At first, I hardcoded object classes like "person" and "car" one by one, but later made the system dynamic to handle any class.

Confidence thresholds also needed tuning. Some low-confidence detections were actually accurate, while some high-confidence ones were misleading. It took a lot of trial and error to find the right balance depending on object type.

3 Segmentation with SAM

The biggest turning point was discovering the Segment Anything Model (SAM). Before this, I didn't fully understand how segmentation worked. I thought bounding boxes were enough — but SAM showed that precise masks were necessary for realistic edits.

Integrating SAM with YOLO wasn't easy. I had to convert YOLO's bounding box outputs into SAM input prompts and fix issues with coordinate mismatches. Eventually, I got it to produce pixel-perfect masks for detected objects. This was a major moment for me — seeing clean segmentation results coming from raw detection boxes made the entire pipeline feel much more powerful.

4 Natural Language Understanding

For user instructions, I started off using basic keyword matching. It worked for simple commands like "remove car," but failed with anything more complex. I moved to using spaCy to better parse natural language. This involved learning how dependency parsing works, identifying object-action pairs, and supporting more natural phrasing like “move the car slightly to the left.”

This step turned out to be important for making the system usable with real-world instructions, not just strict command syntax.

5 Inpainting and Background Reconstruction

I had previously heard about Stable Diffusion for image generation, but didn’t know it could do inpainting. Once I tried it for object removal, the results were far better than simple masking techniques. Instead of obvious holes, the backgrounds looked natural and coherent.

Setting up the inpainting pipeline involved learning how to generate proper binary masks and ensuring the masked regions matched object shapes. I also learned the role of prompt tuning in guiding the diffusion model to generate more contextually accurate fills.

6 Compositing and Visual Quality

Early attempts to relocate objects looked very unrealistic — pasted objects had hard edges, lighting didn’t match, and the overall composition looked fake. I explored blending techniques like alpha compositing with feathered edges, and experimented with basic lighting adjustments to match object and background tones.

I also realized that even technically correct object placement can look bad if context is ignored — for example, a car floating off the road. I started considering context and rough ground-plane constraints to make object placements more plausible.

7 Technical Challenges

7.1 Memory Management

Running YOLO, SAM, and Stable Diffusion together was GPU-intensive. I ran into out-of-memory errors frequently. I handled this by loading models on demand, moving some processing to CPU, and optimizing memory usage.

7.2 Pipeline Design

Initially, the project was just a collection of loosely connected scripts. Failures in one part would crash everything. Eventually, I built a unified `SceneManipulator` class that handled model loading, image transformations, error checking, and metadata flow. This made the entire pipeline more robust and reusable.

7.3 Evaluation

At first, I evaluated results by visual inspection. Later, I added CLIP-based similarity scoring to compare the original instruction with the edited image. This provided a way to quantify how well the manipulation aligned with the command. Scores between 0.75–0.95 indicated strong alignment.

8 Final System Capabilities

The pipeline I built is capable of:

- **Object Detection:** Using YOLOv8 to locate objects in images.
- **Segmentation:** Using SAM to generate precise object masks.
- **Instruction Parsing:** Using spaCy to extract action-object pairs from natural language.
- **Inpainting:** Using diffusion models to reconstruct masked backgrounds.
- **Compositing:** Relocating objects with smooth blending and rough context-awareness.
- **Evaluation:** Using CLIP similarity scores to assess success.

9 Conclusion and Future Work

This project helped me understand that real scene manipulation involves combining multiple specialized models into a cohesive system. I moved from thinking of YOLO as the full solution to realizing that detection, segmentation, NLP, and generative tools must all work together.

In future versions, I would like to explore:

- Adding video support and temporal consistency
- Incorporating artistic style transfer
- Simulating lighting and shadows for better realism

Overall, this project gave me a deeper understanding of applied AI systems and the practical challenges in making them work together in real-world tasks.