AI Internship – Object Detection Assignment

Experience Report by Saurav Ghoshal

Completing this assignment was a mix of challenge, growth, frustration, and satisfaction. When I first started, I thought using a YOLO-style detection head with a MobileNet backbone wouldn't be *that* hard. I was wrong. It wasn't a plug-and-play task – a lot of details had to be handled carefully: target formatting, resizing, detection head integration, loss function, and debugging inference issues.

1. How the System Works (An Overview)

In this project, I designed a YOLO-style object detection model using a **MobileNet** backbone trained on the **Pascal VOC 2007** dataset. The overall system was structured in three main stages:

1. **Feature Extraction with MobileNet Backbone:**

   I used a lightweight, pre-trained **MobileNetV2** model and removed its classification layers to extract spatial features. This helped reduce model size and training time while retaining decent performance on VOC images.

2. **Custom YOLO-style Detection Head:**

   A custom convolutional detection head was added to predict bounding box coordinates (center x, center y, width, height), objectness scores, and class probabilities. Each grid cell in the final feature map was responsible for detecting objects in its region.

3. **Training and Evaluation:**

   The model was trained using Pascal VOC annotations (converted to a YOLO-style format), and evaluated with metrics including **mAP**, **precision**, **recall**, and **IoU**. I implemented utility functions for both **target encoding** and **metric computation**, and used a **custom collate function** to handle variable-sized annotations.

---

2. Challenges Faced During Implementation

This assignment threw more challenges at me than I initially expected, especially in areas I previously took for granted when using pre-built models.

1. **Integrating Detection Head with MobileNet**

   MobileNet is optimized for classification, so adapting it for detection required modifying internal layers without breaking the architecture. Extracting feature maps and ensuring the correct spatial resolution for the YOLO-style head involved several failed attempts and reshaping mismatches.

2. **Target Encoding for YOLO-style Output**

   Converting Pascal VOC bounding boxes (absolute coordinates) into normalized YOLO format (center x, center y, w, h) was trickier than expected. Errors in normalization caused the model to predict wildly incorrect boxes until I added debugging visualizations to catch and fix the logic.

3. **Handling Inconsistent VOC Annotations**

   The XML format varied between images — some had a single object, others multiple. This inconsistency led to RuntimeError: each element in list of batch should be of equal size, which I resolved by writing a custom collate function that padded or adjusted annotations on the fly.

4. **Designing a Functional Loss Function**

   YOLO-style loss functions are complex, combining classification, box regression, and objectness. I initially used a simple combination of MSE and BCE losses, but performance only improved after refining it and making sure all outputs were properly activated (sigmoid for objectness).

5. **Low Confidence Predictions During Inference**

   At first, my inference script returned no boxes — everything was getting filtered out. I traced the issue to missing sigmoid activations on the confidence scores and overly aggressive confidence thresholds. After applying the correct activation and tuning the threshold, detections began working correctly.

6. **Metric Calculation and mAP**

Implementing custom **IoU**, **precision**, **recall**, and **mean Average Precision (mAP)** was educational but error-prone. Mistakes in batch dimension handling led to metrics >1, which made me double-check each step with manual box comparisons.

---

3. How I Used AI Tools to Help With Coding

AI tools like **ChatGPT** played a crucial supporting role throughout this project. They helped me:

- Refactor and freeze/unfreeze parts of the MobileNet backbone.
- Implement the YOLO-style detection head with confidence and class predictions.
- Build the custom collate_fn for VOC's variable XML annotations.
- Visualize bounding boxes correctly for debugging inference.
- Clarify shape mismatches and PyTorch broadcasting issues in the loss function.
- Design a working evaluation pipeline (IoU, mAP, precision, recall).

Importantly, I didn't treat AI as a black box. I made sure I understood the logic behind each code snippet before integrating it into my workflow. This helped me retain control over the development process while speeding up troubleshooting and learning.

---

4. What I Learned From the Project

- **CNNs can be repurposed** — even a classification backbone like MobileNet can be adapted to detection with the right head.
- **YOLO's simplicity is deceptive** — its grid-based logic, loss function design, and multi-task outputs are non-trivial to implement from scratch.
- **Data preprocessing matters** as much as model performance. Encoding targets, normalizing values, and debugging annotation formats is half the job.
- **Visualization is critical** — it's often the only way to understand what's really happening in detection tasks.
- **mAP and IoU computation is not plug-and-play** — understanding how they're calculated deepens my grasp of what "accuracy" really means in object detection.

5. What Surprised Me About the Process

- How **messy real-world datasets** like Pascal VOC are — inconsistent annotations, missing labels, non-uniform image sizes.
- How **fragile object detection pipelines** are — a small mistake in normalization or output shaping can silently tank performance.
- How even a minimal, custom model can reach a decent mAP (~0.55) if the pipeline is well-structured.
- How much **time debugging takes** in detection — inference didn't work for a long time because of a missing sigmoid. It's the kind of small issue that costs hours if you're not visualizing outputs.

6. Balance Between Writing Code Myself vs. Using AI Assistance

This assignment showed me the value of using AI tools *strategically*. I didn't try to build the entire thing from scratch by hand — not out of laziness, but out of respect for time and complexity. That said, I:

- Wrote the model architecture and training logic myself.
- Used AI tools to fix bugs, refactor code, and accelerate repetitive tasks.
- Never used an AI-generated code block unless I fully understood what it did.

The ideal workflow for me was:

- **Think the problem through myself first**
- **Use AI to validate or correct my plan**
- **Implement, test, and debug with full visibility**

7. Suggestions for Improving the Assignment

- **Fix the backbone + head combo** (e.g., MobileNet + YOLO) so we focus on logic, not architecture decisions.
- Provide **a reference collate function** or data loader for VOC — it's a waste of time parsing XML trees when the goal is to learn detection logic.
- Include a **starter loss function** that works for basic training. Custom losses can come as an advanced exercise.
- Ask students to **visualize detections** and submit result images — it makes the pipeline clearer and helps catch bugs early.

- Consider sharing a **toy image or test set** to verify output format and inference flow.

Final Thoughts

This was a challenging but rewarding assignment. It forced me to dive into every layer of a detection system — from model building to data preprocessing to evaluation. I now understand how YOLO-style detection really works under the hood, and how even lightweight models like MobileNet can be extended to perform real-world vision tasks.

Equally important, I learned how to work with AI as a development partner. Instead of doing the work *for* me, it helped me do it *better and faster*. I feel much more confident in both object detection concepts and in leveraging AI to build robust machine learning pipelines.