Training Object Detectors from Scratch

Task 1: Project Overview

This assignment required building an object detection model from scratch. The main constraint was avoiding pre-trained weights, meaning I couldn't use ImageNet features. This required teaching the network fundamental visual features like edges and textures before it could identify specific objects.

This constraint changed my entire strategy. I couldn't simply fine-tune a massive ResNet, so I had to look for architectures that are parameter-efficient and handle gradient flow well to avoid vanishing gradients during early training epochs. Before writing the code, I researched architecture theory, specifically how modern backbones like CSPNet handle feature propagation.

1. Approaches

The prompt suggested trying a Custom CNN or Faster R-CNN. I implemented both to meet the requirements, but I also explored other options when those models hit performance ceilings.

Attempt 1: Custom Single-Stage CNN

I started by building a custom single-stage detector from scratch using PyTorch. I wanted to see if a lightweight 6-layer CNN could learn basic detection.

- **The Setup:** Simple Conv2d blocks followed by BatchNorm and LeakyReLU. I implemented a custom loss function combining Coordinate MSE and Objectness Entropy.
- **Performance Issues (mAP ~45%):**
  - **Grid Limitations:** The simple grid system I designed struggled when multiple objects appeared in the same cell.
  - **Lack of Feature Pyramid:** Without a Feature Pyramid Network, the model lost track of small objects. By the time the image was downsampled to the final layer, small objects like birds were often less than a pixel wide.
  - **Conclusion:** It was a good exercise for understanding the mathematics of detection but not suitable for production.

Attempt 2: Faster R-CNN with MobileNetV3

I implemented a Faster R-CNN with a MobileNetV3 backbone as this is a standard architecture known for high accuracy.

- **Training Difficulties:** Training this from scratch proved very difficult. A two-stage detector relies on a Region Proposal Network (RPN) to suggest bounding boxes.
  - **The RPN Issue:** The RPN needs good features to propose boxes, but the backbone needs good box gradients to learn those features. Since I started with random weights, the RPN proposed poor regions initially, making it hard for the system to bootstrap itself.

- **Result:** After 100 epochs, the model plateaued at **~21% mAP**. It was learning, but the convergence was extremely slow.

Attempt 3: CCTT Transformer Architecture

I researched recent literature and found a paper by Hong et al. (2024) which proposed mixing Convolutions and Transformers for training from scratch.

- **Implementation Outcome:** While the concept was interesting, Transformers lack inductive bias, which is the innate understanding of spatial locality. They typically require 300 or more epochs to learn what a CNN knows by default. I implemented the architecture, but given the time constraints, it could not converge effectively (mAP < 2%).

Attempt 4: YOLOv8 Nano

Finally, I used the YOLOv8 Nano architecture. This ended up being the best solution because it addresses the specific problems encountered in the previous attempts.

2. Technical Analysis of YOLOv8 Results

The YOLOv8n model succeeded because of specific architectural decisions that facilitate rapid convergence:

1. **Gradient Richness (C2f Modules):** Unlike my Custom CNN, YOLOv8 uses Cross-Stage Partial connections. This splits the gradient flow and ensures that even deep layers get strong updates. This is critical when you don't have pre-trained weights to guide the training.
2. **Anchor-Free Head:** In Faster R-CNN, I struggled to tune anchor box sizes for my custom dataset. YOLOv8 predicts object centers directly. This removed a complex hyperparameter and sped up convergence.
3. **Decoupled Head:** It separates the classification task from the regression task. My experiments showed this helped the model learn specific features for identifying the object versus locating the object without interference.

3. Data Augmentation Strategy

Since I was working with a small subset of PASCAL data and no pre-trained weights, avoiding overfitting was critical. I used the Albumentations library to create a robust pipeline.

- **Mosaic Augmentation:** This was the most effective strategy. By stitching 4 images together, I forced the model to learn to detect objects at different scales and locations independent of context.
- **HSV Jitter:** I randomly shifted Hue, Saturation, and Value. This ensured the model learned the shape of the object rather than relying on specific colors.
- **Disabling Augmentations:** I implemented a strategy where I disabled Mosaic for the last 10 epochs. This allowed the model to fine-tune on natural images, which gave me a noticeable boost in mAP at the end of training.

4. Final Results & Trade-offs

Model	mAP@50	Speed (FPS)	Analysis
YOLOv8 Nano	**61.5%**	**~100**	**Best balance of speed and accuracy.**
Custom CNN	45.0%	>120	Fast but struggled with overlapping objects.
Faster R-CNN	21.0%	~25	Too complex to initialize effectively from scratch.
Hybrid Transformer	1.8%	~15	Required significantly longer training time.

5. Retrospective and Future Work

I had to prioritize finishing the task within the timeframe, but there are other approaches I would have liked to explore if I had more resources:

1. **Semi-Supervised Learning (SSL):** I could have used a technique like SimCLR or MoCo to pre-train my backbone on the dataset without labels first. This would have provided a better starting point than random initialization and might have fixed the convergence issues with Faster R-CNN.
2. **Longer Training Schedules:** Research suggests that models trained from scratch often need 3 to 5 times more epochs than fine-tuning. I stopped at 50-100 epochs due to time constraints, but pushing to 300 might have made the Transformer approach viable.
3. **Detection Transformers (DETR):** I considered using DETR, but they are known for slow convergence. Given the constraints of this assignment, relying on DETR would have been a high risk.

6. Conclusion

This project demonstrated the difficulty of training deep learning models without pre-trained weights. The failure of the Faster R-CNN highlighted the importance of initialization, while the success of YOLOv8n showed the effectiveness of modern gradient flow architectures.

Final Deliverable: A robust, real-time detector (YOLOv8n) achieving **61.5% mAP**, backed by a comprehensive study of why it works better than the alternatives for this specific constraint.