Project Report: Custom Object Detection Pipeline (Faster R-CNN)

1. Architecture Design Choices

For this task, a **Faster R-CNN** (Region-based Convolutional Neural Network) architecture was implemented from scratch to solve the generic object detection problem on the PASCAL VOC 2012 dataset.

A. Backbone Network: VGG16

- **Choice:** The VGG16 architecture was selected as the feature extractor.
- **Implementation:** torchvision.models.vgg16(pretrained=False).
- **Reasoning:** VGG16 is a classic, structurally simple CNN that provides a robust baseline for feature extraction. Crucially, pretrained=False was used to meet the "train from scratch" requirement, ensuring the model learned features solely from the provided custom dataset rather than relying on ImageNet weights.
- **Modification:** The final max-pooling layer and fully connected layers were removed to preserve spatial dimensions for the RPN.

B. Region Proposal Network (RPN)

- **Function:** Slides a small network over the feature map to propose candidate object regions.
- **Anchors:** 9 anchors per location were used, derived from 3 scales ([128, 256, 512]) and 3 aspect ratios ([0.5, 1, 2]).
- **Loss:** Combination of Binary Cross Entropy (objectness) and Smooth L1 Loss (box regression).

C. ROI Head (Region of Interest)

- **Pooling:** RoIPool (7x7) was used to convert variable-sized proposals into fixed-size feature vectors.
- **Classification:** Two fully connected layers (fc_inner_dim=1024) lead to the final classification layer (21 classes: 20 objects + 1 background).
- **Inference:** Non-Maximum Suppression (NMS) with a threshold of 0.3 is applied to filter overlapping boxes during testing.

---

2. Data Augmentation Strategies

Data augmentation was critical to prevent overfitting, especially since the model was initialized with random weights.

A. Implemented Strategies

- **Random Horizontal Flip:** Implemented in dataset/voc.py. During training, images have a 50% probability of being flipped horizontally. The ground truth bounding boxes are mathematically transformed (x1 = width - x1 - w) to match the flip.

B. Identified Limitations & Future Work

- **Issue Observed:** During video testing, the model exhibited "hallucinations" (high-confidence false positives) due to domain shift (video lighting vs. training set lighting).
- **Proposed Fix:** The current pipeline lacks **Color Jitter** (brightness/contrast) and **Multi-Scale Training**. Adding these would force the model to learn shape-based features rather than relying on specific texture/color patterns found in PASCAL VOC.

---

3. Training Methodology

The training was conducted on a **GPU P100** environment to handle the computational load of a two-stage detector.

A. Dataset

- **Source:** PASCAL VOC 2012 (Train/Val set).
- **Classes:** 20 Object Classes (Person, Car, Bird, etc.) + Background.
- **Input Resolution:** Images are resized such that the shorter side is at least 600px and the longer side does not exceed 1000px.

B. Hyperparameters

- **Optimizer:** Stochastic Gradient Descent (SGD) with Momentum (0.9) and Weight Decay (5 *10^{-4}).
- **Learning Rate:** Initial LR of 0.0001 (conservative start for scratch training).
- **Scheduler:** MultiStepLR with decay steps at certain epoch intervals. This "refinement phase" allows the model to settle into local minima and distinguish between similar classes.
- **Epochs: 50 Total Epochs** (Required for convergence from random initialization).

C. Stability Technique: Gradient Accumulation

- **Problem:** The input images vary in size, forcing a batch_size=1 in the DataLoader. This leads to noisy gradients.
- **Solution:** Implemented acc_steps=4. Gradients are accumulated over 4 forward passes before a optimizer.step() is called, effectively simulating a batch size of 4 for stable convergence.

---

4. Results Comparison

The model was evaluated using the tools/infer.py script on the PASCAL VOC validation set.

Metric	Result	Notes

mAP (Mean Average Precision)	**0.17**	*Low because training from random initialization*
Inference Speed	**16.54 FPS**	*Measured on GPU. Expect ~15-20 FPS.*
Model Size	**167.64 MB**	*VGG16 is heavy.*

Experimental Observation: Initial training (20 epochs) resulted in a strong bias towards the "Person" class due to class imbalance. Extending training to 50 epochs and correcting the class mapping logic resolved this, allowing for multi-class detection.

---

5. Discussion: Trade-offs between Accuracy and Speed

A. Accuracy vs. Speed

- **Design Choice:** Faster R-CNN is a **two-stage detector**. It generates proposals first (RPN) and then classifies them (ROI Head).
- **Trade-off:** This design yields **higher accuracy** compared to single-stage detectors (like YOLO) because it refines the boxes twice. However, it is significantly **slower**. The RPN bottleneck prevents the model from reaching high real-time FPS (30+) on standard hardware.

B. Backbone Heavy-Lifting

- **VGG16 Impact:** VGG16 has approx. 138 million parameters. While it provides excellent feature depth for accuracy, it is computationally expensive.
- **Trade-off:** A lighter backbone like ResNet-50 or MobileNet would have offered 2x-3x faster inference speed with a minor drop in accuracy, but VGG16 was chosen for its architectural simplicity and proven stability for training from scratch.

C. From-Scratch Constraints

- **Training Time:** Training from scratch required 50 epochs to reach decent mAP. Using pre-trained ImageNet weights would have achieved similar accuracy in just 5-10 epochs. The trade-off here was **training compute time** vs. **model independence**.

Appendix: Real-Time Detection

(Attach your output_detection.gif here)

The video demonstrates the model running on a sample traffic scene. Note the detection of cars and persons. Occasional flickering is observed due to frame-by-frame processing without temporal smoothing.