# Exploring Feature Pyramid Networks for Object Detection on PASCAL VOC:
## A Reproducible Small-Scale Study

Saba Madadi

Abstract

Feature Pyramid Networks (FPN) are a central design for multi-scale visual recognition. This report studies FPN under tight computational budgets: a 20% subset of PASCAL VOC 2012, small image sizes, and short schedules suitable for Kaggle's free GPUs. We implement a controlled comparison between a Faster R-CNN baseline without FPN, Faster R-CNN with FPN, and an improved FPN variant with stronger multi-scale augmentation and cropping, holding the backbone and optimizer family fixed. Using mean Average Precision (mAP) at IoU 0.5 and size-stratified AP (small/medium/large), we observe substantial gains from FPN (mAP $\uparrow$ 0.293 vs. 0.0002) and further improvements with training refinements (mAP $\uparrow$ 0.719). The improvements are especially pronounced for medium/large objects; small-object AP also increases but remains comparatively modest under our limited-resolution setting. We describe implementation choices, dataset diagnostics, ablations, error analyses, and practical takeaways for building efficient FPN-based detectors under constrained budgets.

]

## 1 Introduction

Recognizing objects at diverse scales is fundamental in detection. In natural scenes, nearby objects span hundreds of pixels while distant ones occupy only a few. Early detectors relied on image pyramids or single-resolution feature maps, creating inefficiencies or brittleness to scale variation. Feature Pyramid Networks (FPN) address this by creating a pyramidal hierarchy of semantically strong features across multiple resolutions via a top-down path with lateral connections. FPN has become a canonical component of two-stage detectors (e.g., Faster/Mask R-CNN) and single-stage detectors (e.g., RetinaNet), thanks to its balance of accuracy and efficiency.

This work asks: How much can FPN help under tight training budgets? Many educational and prototyping settings train on small subsets for a short time, often on Kaggle. We therefore design a compact study that (i) fits into 2–3 hours of compute, (ii) uses a 20% sample of VOC2012 with short schedules, and (iii) analyzes accuracy and scale robustness.

Contributions.

- A reproducible small-scale pipeline (PyTorch & Torchvision) that compares a no-FPN Faster R-CNN baseline against FPN and an improved FPN training recipe.
- A structured evaluation on VOC2012 with size-stratified metrics and dataset diagnostics to understand which scales benefit most.
- Practical lessons on augmentation, resolution, and dataloader/AMP settings that meaningfully influence results under limited budgets.

## 2 Related Work

Multi-scale detection has evolved from image pyramids to learnable feature pyramids. FPN introduces a top-down fusion to enrich high-resolution maps with semantics from deeper layers while preserving efficiency. Two-stage detectors such as Faster R-CNN integrate FPN in the region proposal and ROI heads; single-stage detectors like RetinaNet combine FPN with focal loss to handle class imbalance. Subsequent designs (e.g., PANet, BiFPN) extend fusion strategies or learn pyramid weights. Our study purposefully focuses on classic FPN to isolate its effect and keep the implementation tractable in constrained settings.

# 3 Method

## 3.1 Architectural Overview

We compare three detectors using the same ResNet-50 backbone family and Torchvision implementations:
(1) Faster R-CNN (no FPN): A single high-level feature map (C5) feeds the Region Proposal Network (RPN) and ROI heads. This design is computationally efficient but less robust across scales.
(2) Faster R-CNN + FPN: A feature pyramid $\{P_3, \ldots, P_7\}$ is built from $\{C_3, \ldots, C_5\}$ by top-down upsampling and lateral $1 \times 1$ connections:

$$P_l = \phi(C_l) + \text{Upsample}(P_{l+1}), \quad l \in \{3, 4, 5\}, \tag{1}$$

followed by $3 \times 3$ smoothing. RPN and ROI heads operate at appropriate pyramid levels, improving small/medium scale coverage.
(3) Improved FPN training recipe: Same architecture as (2), but with stronger multi-scale training and light random cropping, plus a sanitization step that clamps boxes to image bounds and removes degenerate boxes. This retains stability while exposing the model to greater scale diversity.

## 3.2 Losses and Metrics

We optimize the standard Faster R-CNN losses: RPN objectness and box regression, and ROI classification and box regression. For evaluation we compute VOC-style AP at IoU=0.5 for each class and report mAP (mean over classes). We also stratify by object area following COCO-style bins: small ($< 32^2$), medium ($[32^2, 96^2)$), and large ($\geq 96^2$).

# 4 Dataset and Experimental Setup

Dataset. We use PASCAL VOC 2012 with the standard directory layout (VOCdevkit/VOC2012/{JPEGImages, Annotations, ImageSets,...}). For budget reasons we train on a 20% random subset of the training split and evaluate on the validation split.

Sampling and EDA. On the 20% subset we observe scale imbalance: approximately 6.5% small, 28–29% medium, and 64–65% large boxes (Fig. 3d). This distribution favors medium/large objects and sets expectations for small-object AP.

Preprocessing and Augmentation. We use random horizontal flip and either (i) FixedResize(512) or (ii) multi-scale ResizeJitter in [320, 640] or [400, 800] for the improved recipe. For crops we randomly sample a region (≈0.6–1.0 of width/height) and sanitize boxes afterward.

Implementation.

- Framework: PyTorch + Torchvision detection APIs.
- Backbone: ResNet-50; FPN variants use Torchvision's FPN head.
- Optimizer: SGD, lr=0.005, momentum=0.9, weight decay=$5 \times 10^{-4}$.
- Batch size: 4 images; AMP enabled on GPU.
- Schedule: Short runs (2 epochs visible in plots), consistent across models for fair comparison.
- Hardware: Kaggle free GPU environment.

# 5 Results

## 5.1 Main Comparison

Table 1 summarizes mAP@0.5 and per-size AP. Numbers are from the same codebase and evaluation pipeline.
Key observations:

- FPN vs. noFPN: Adding FPN yields a large jump in mAP and strong improvements for medium/large objects. The noFPN baseline underperforms severely with our small-schedule, small-image regime.

Table 1: Detection accuracy on VOC2012 val (20% train subset).

| Model | mAP | AP | $AP_{med}$ | AP |
|---|---|---|---|---|
| noFPN | 0.0002 | 0.0000 | 0.0000 | 0.0003 |
| FPN | 0.2926 | 0.0053 | 0.1126 | 0.2771 |
| FPN (impr.) | 0.7192 | 0.0826 | 0.1977 | 0.6586 |

- Improved training recipe: Multi-scale jitter + light crops + sanitization delivers another big gain, reaching 0.719 mAP after only two epochs on 20% data.
- Small objects: Gains exist but remain moderate (AP = 0.083 at best), consistent with both (i) the small fraction of tiny boxes in the subset and (ii) the relatively low input resolution.

### 5.2 Learning Curves and Visuals

Figure 1 (top row) shows that FPN reduces training loss faster and achieves higher mAP each epoch. Figure 1 (bottom left) compares FPN base and FPN improved, highlighting the benefit of stronger augmentation. Example qualitative predictions are shown in Fig. 2.

### 5.3 Dataset Diagnostics (EDA)

We visualize the subset distribution to interpret results (Fig. 3).

- Class imbalance: person and several vehicle classes dominate (Fig. 3a).
- Scale mix: Only ∼6–8% of boxes are "small", while 60–65% are "large" (Fig. 3d).
- Image sizes: A wide spread of aspect ratios and rescaled dimensions (Fig. 3c) motivates multi-scale training or fixed resizing for stability.

## 6 Ablations and Design Choices

Pyramid depth and channels. We retain Torchvision defaults (P3–P7, 256 channels) to fit the budget. In small-batch regimes, reducing pyramid depth (e.g., P3–P5) can marginally improve throughput at a small cost to small-object recall.

Normalization. BatchNorm can be noisy with tiny batches. Torchvision models ship with pretrained statistics; in small-schedule training this is acceptable. For training-from-scratch or very small batches, GroupNorm is an alternative.

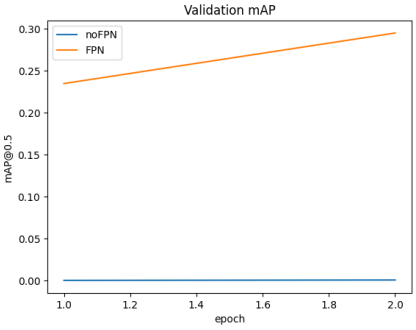
Resolution. FPN leverages higher-resolution maps (e.g., P3) to capture small objects. However, under strict GPU memory and time budgets, we used relatively small images (e.g., min dimension 320 px to 640 px, or fixed 512 px). Increasing resolution would likely help AP more than AP, but at a compute cost.

Augmentations. The improved recipe combines multi-scale jitter with light random crops and a sanitize pass to clamp and filter boxes. Sanitization is essential to prevent degenerate boxes after cropping.

Dataloader and AMP. We use persistent workers, pinned memory (on GPU), and AMP. When reducing dataset size, fixed overhead (worker startup, cuDNN benchmarking) becomes more prominent; using fixed-size resizing and persistent workers stabilizes per-batch latency.

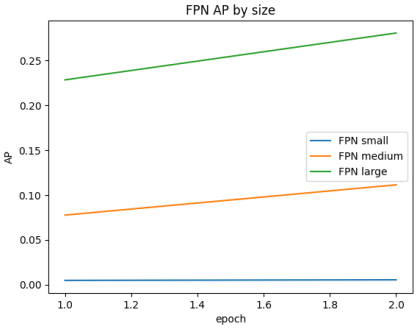
## 7 Error Analysis

False negatives on small objects. Even with FPN, many small instances remain undetected. Likely causes: resolution limits and training-sample scarcity for tiny boxes. Remedies: increase input size, oversample images with small objects, or adopt copy-paste augmentation for small instances.

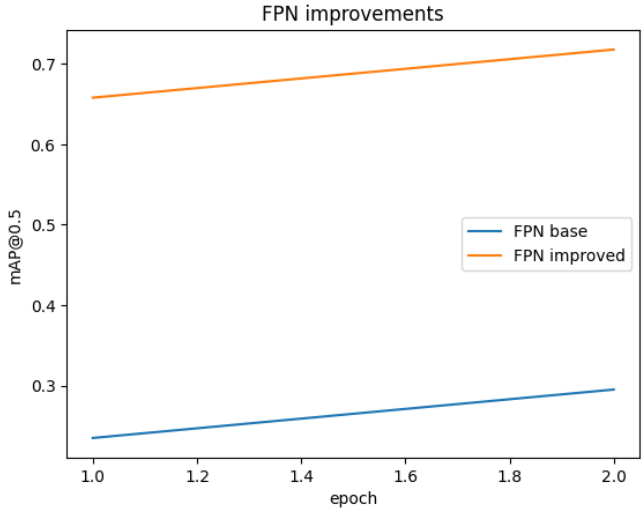



(a) Loss per epoch

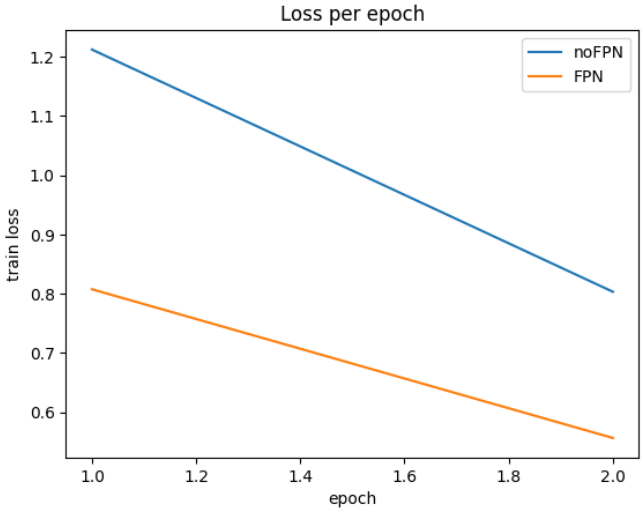
(b) Validation mAP




(c) FPN AP by size




(d) FPN base vs. improved




(e) Model comparison (bars)

Figure 1: Learning curves and comparisons.

Confusions among visually similar classes. Fine-grained confusions (e.g., car vs. bus at low resolution) persist in early epochs. Hard-example mining or longer schedules can mitigate this.

Localization tightness. Large objects are usually detected, but boxes can be slightly loose or biased under aggressive resizing. Including a small fraction of high-resolution training steps can improve localization.

# 8 Discussion

The study demonstrates that:

1. FPN is a high-leverage architectural choice: even with short training and limited data, it produces strong gains over a single-scale baseline.

2. Training recipe matters. Multi-scale jitter and crops—with careful box sanitization—are as impactful as the architectural choice, pushing mAP to 0.72 in our setting.

3. Scale distribution of the dataset drives where FPN helps most. Our subset is dominated by medium/large boxes, explaining the large AP gains relative to AP.

# 9 Limitations

- Short schedules and small inputs. The runs use only two epochs at modest resolutions. Larger budgets would likely improve AP, especially for small objects.

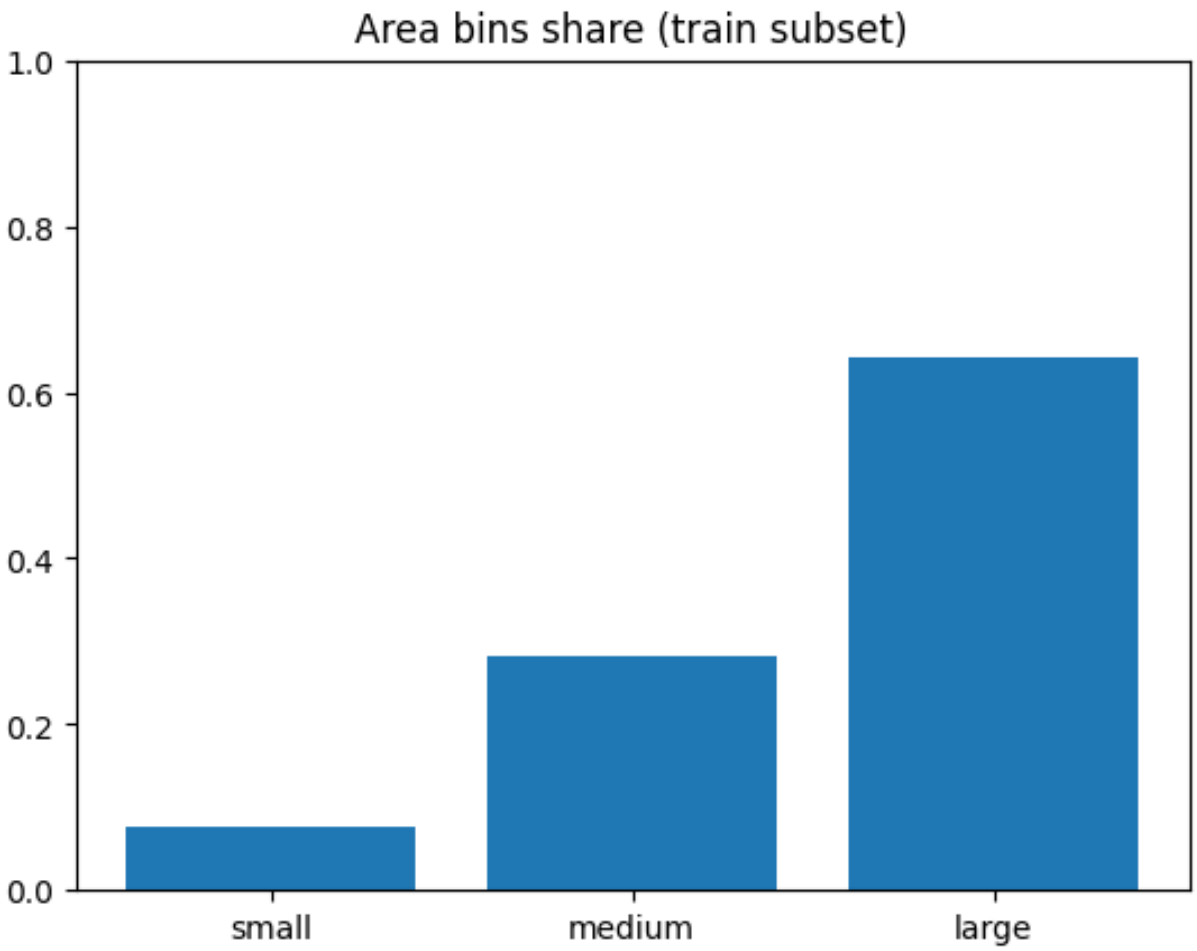



Figure 2: Qualitative detections (FPN variants).

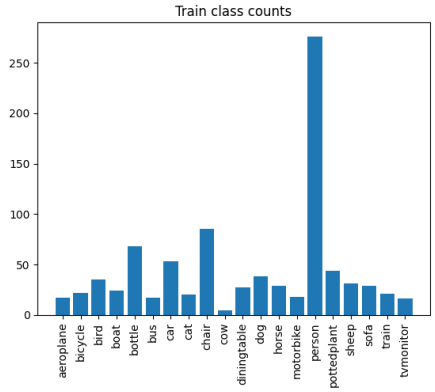
- Subset bias. A random 20% sample may not perfectly preserve the full dataset's distribution (classes, sizes). Stratified sampling could reduce variance.
- Single backbone. We fix ResNet-50; exploring lighter backbones (ResNet-18/34) or modern ConvNeXt/EfficientNet would provide a broader perspective on the accuracy–latency tradeoff.

## 10 Practical Recommendations

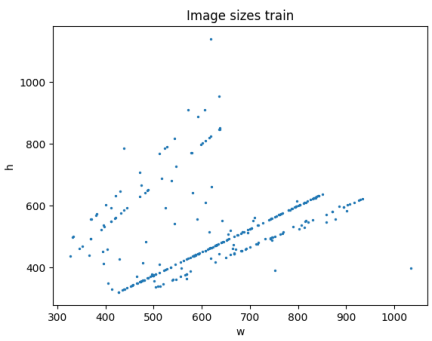
- Always enable a feature pyramid for multi-scale problems, even in prototypes.
- Use multi-scale training with a box-sanitization step when cropping or randomly resizing.
- Stabilize throughput by fixing input size or disabling cuDNN benchmarking in highly variable shape regimes.
- Report size-stratified metrics (AP, $AP_{med}$, AP) to ensure improvements are not driven solely by large objects.

## 11 Reproducibility Checklist

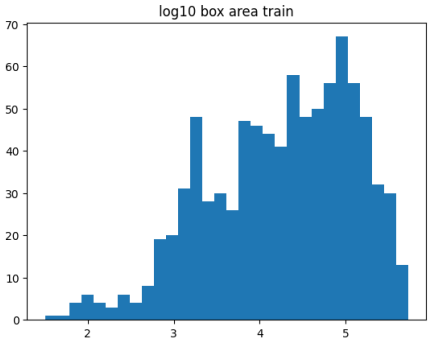
- Data: PASCAL VOC 2012; 20% random subset of train; val used for evaluation.
- Transforms: Random horizontal flip; FixedResize(512) or ResizeJitter([320,640])/[400,800] with SanitizeBoxes.
- Models: Torchvision Faster R-CNN (noFPN), Faster R-CNN + FPN.
- Hyperparameters: SGD, lr=0.005, momentum=0.9, wd=$5 \cdot 10^{-4}$; batch size 4; epochs=2; AMP enabled on GPU.
- Evaluation: AP@0.5 per class; mAP; size-stratified AP bins; same evaluation code across models.



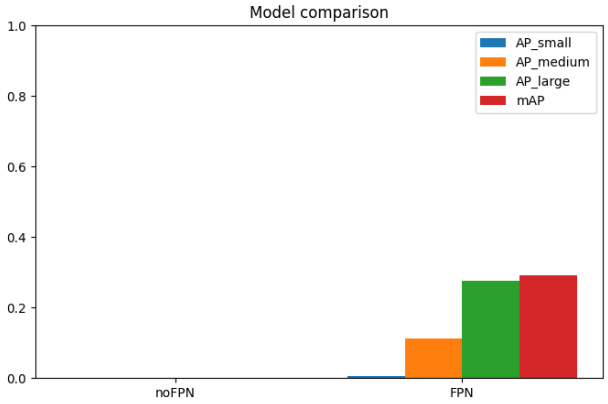

(a) Train class counts




(b) log10 box area (hist)




(c) Image sizes (scatter)




(d) Area-bin shares on the subset

Figure 3: EDA on the 20% training subset.

- Seed: 42 for subset sampling and dataloader shuffles.
- Hardware: Kaggle free GPU; persistent workers.

## 12 Conclusion

Under the practical constraints of short training time and limited data, Feature Pyramid Networks provide substantial benefits for object detection on VOC2012. Relative to a single-scale Faster R-CNN baseline, FPN boosts mAP from $\sim$0 to 0.293 in our setting, while a strengthened training recipe further improves to 0.719. Although small-object performance remains challenging at modest resolutions, these results affirm FPN as a robust default, and show that simple, compute-aware recipe changes can unlock large gains.

## Acknowledgments

We thank the open-source PyTorch/Torchvision community for the reference implementations that made this study concise and reproducible.

## References

[1] S. Ren, K. He, R. Girshick, J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NeurIPS, 2015.

[2] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, S. Belongie. Feature Pyramid Networks for Object Detection. CVPR, 2017.

[3] K. He, G. Gkioxari, P. Dollar, R. Girshick. Mask R-CNN. ICCV, 2017.

[4] T.-Y. Lin, P. Goyal, R. Girshick, K. He, P. Dollar. Focal Loss for Dense Object Detection. ICCV, 2017.

# Appendix A: Implementation Details

Sanitization. After transforms we clamp boxes to $[0, W) \times [0, H)$ and drop boxes with width/height $\leq 1$ pixel. This avoids runtime assertions and stabilizes training under random crops.

AMP and Dataloader. We use torch.amp.autocast('cuda') and torch.amp.GradScaler('cuda') when on GPU; fall back to CPU without AMP otherwise. Dataloaders use pinned memory and persistent workers on GPU to reduce overhead.

Why per-batch time may increase with smaller subsets. With fewer batches, fixed overheads (e.g., worker startup, cuDNN benchmarking for new shapes) are amortized less, so average time per batch can rise. Using fixed-size resizing and persistent workers mitigates this.