iSAID Instance Segmentation with Custom Mask R-CNN

Project Report

Authors: [Michał Pokładowski 160278, Marek Olejniczak 160303]
GitHub Repository: https://github.com/michaelo-ponteski/isaid-instance-segmentation

---

Table of Contents

---

1. Introduction

This project implements instance segmentation on aerial/satellite imagery using a custom Mask R-CNN architecture. The goal is to detect and segment objects in the iSAID (Instance Segmentation in Aerial Images Dataset), which contains high-resolution satellite images with dense object annotations.

Instance segmentation combines object detection (bounding boxes + class labels) with semantic segmentation (pixel-level masks), making it one of the most challenging computer vision tasks.

1.1 Project Goals

- Implement a custom Mask R-CNN with >50% custom layers
- Integrate attention mechanisms (CBAM) for improved feature extraction
- Optimize anchor configurations for aerial imagery characteristics
- Achieve competitive mAP scores on the iSAID validation set
- Provide comprehensive experiment tracking with Weights & Biases

2. Dataset Description

2.1 iSAID Dataset Overview

The **iSAID (Instance Segmentation in Aerial Images Dataset)** is a large-scale benchmark dataset for instance segmentation in aerial imagery. It is derived from the DOTA dataset and provides pixel-level annotations for instance segmentation.

Property	Value
Total Images	2,806 high-resolution images
Total Instances	655,451 annotated instances
Image Resolution	Varies (from 800x800 to 20,000x20,000 pixels)
Patch Size Used	800×800 pixels
Instances after patches applied	940,215 annotated instances
Number of Classes	15 object classes + background
Annotation Format	COCO-style JSON

Split	Images	Avg Boxes/Image	Empty Images
Train	~28k	~25	~33%
Val	~9.5k	~24	~36%

2.2 Object Classes

The dataset contains 15 object categories commonly found in aerial imagery:

Class ID	Class Name	Description
0	Background	Non-object regions
1	Ship	Maritime vessels
2	Storage Tank	Industrial storage containers
3	Baseball Diamond	Sports facilities
4	Tennis Court	Sports facilities
5	Basketball Court	Sports facilities
6	Ground Track Field	Athletic tracks
7	Bridge	Infrastructure
8	Large Vehicle	Trucks, buses
9	Small Vehicle	Cars, motorcycles
10	Helicopter	Aircraft
11	Swimming Pool	Recreational facilities
12	Roundabout	Road infrastructure
13	Soccer Ball Field	Sports facilities
14	Plane	Aircraft
15	Harbor	Maritime infrastructure

2.3 Dataset Statistics

Class Distribution:

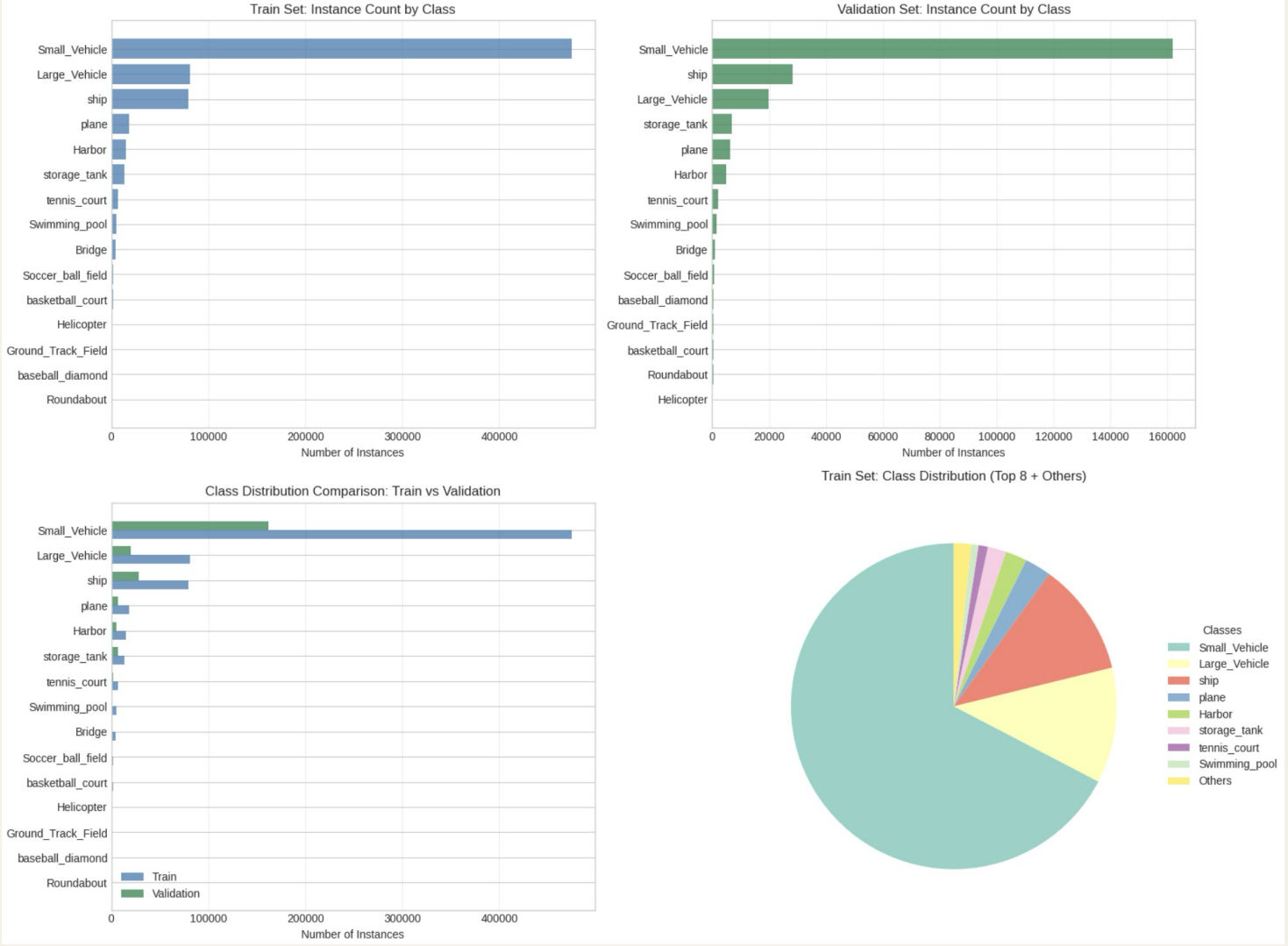


Figure 2.1: Distribution of instances across classes in the training and validation sets

Objects per Image Distribution:

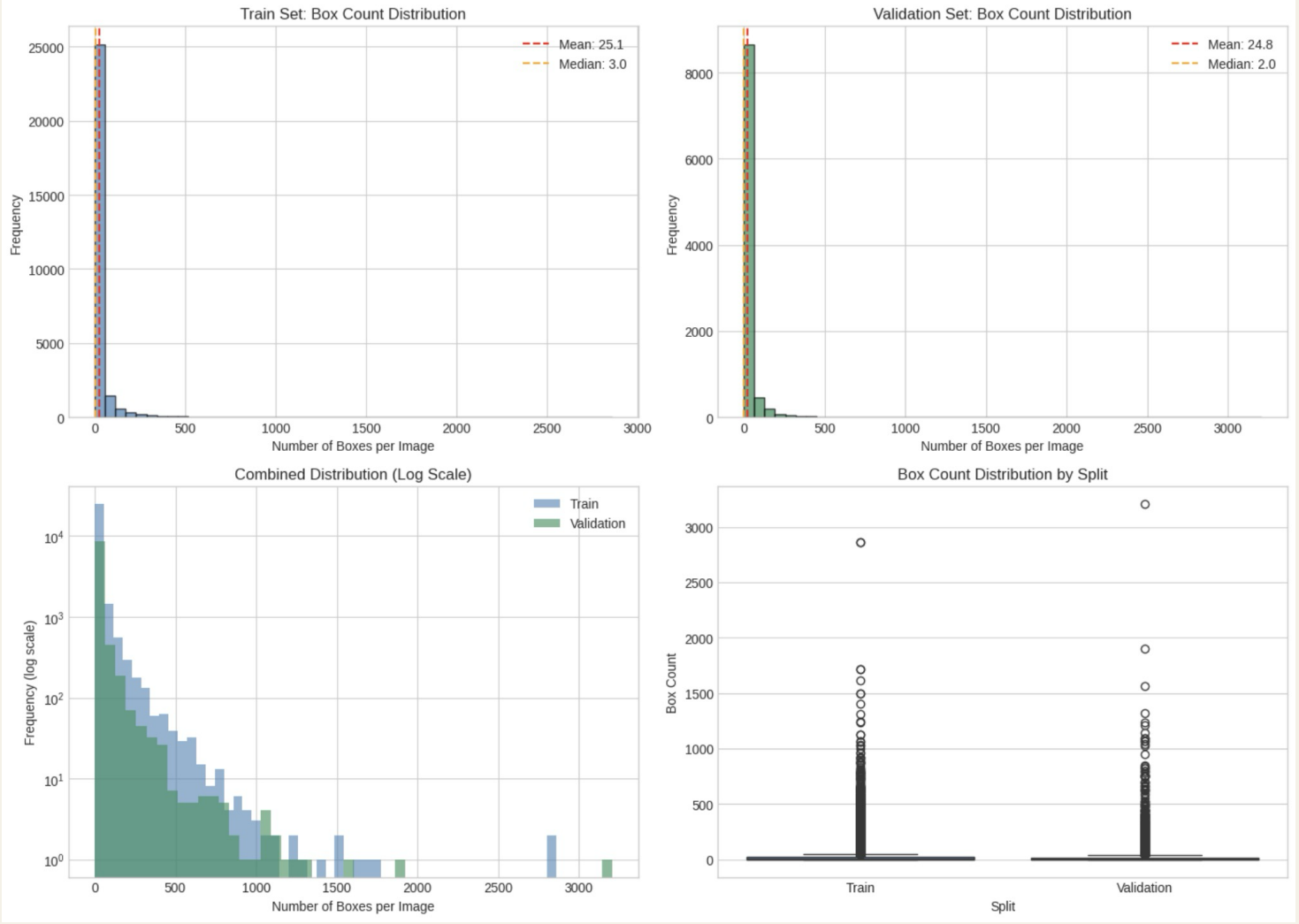
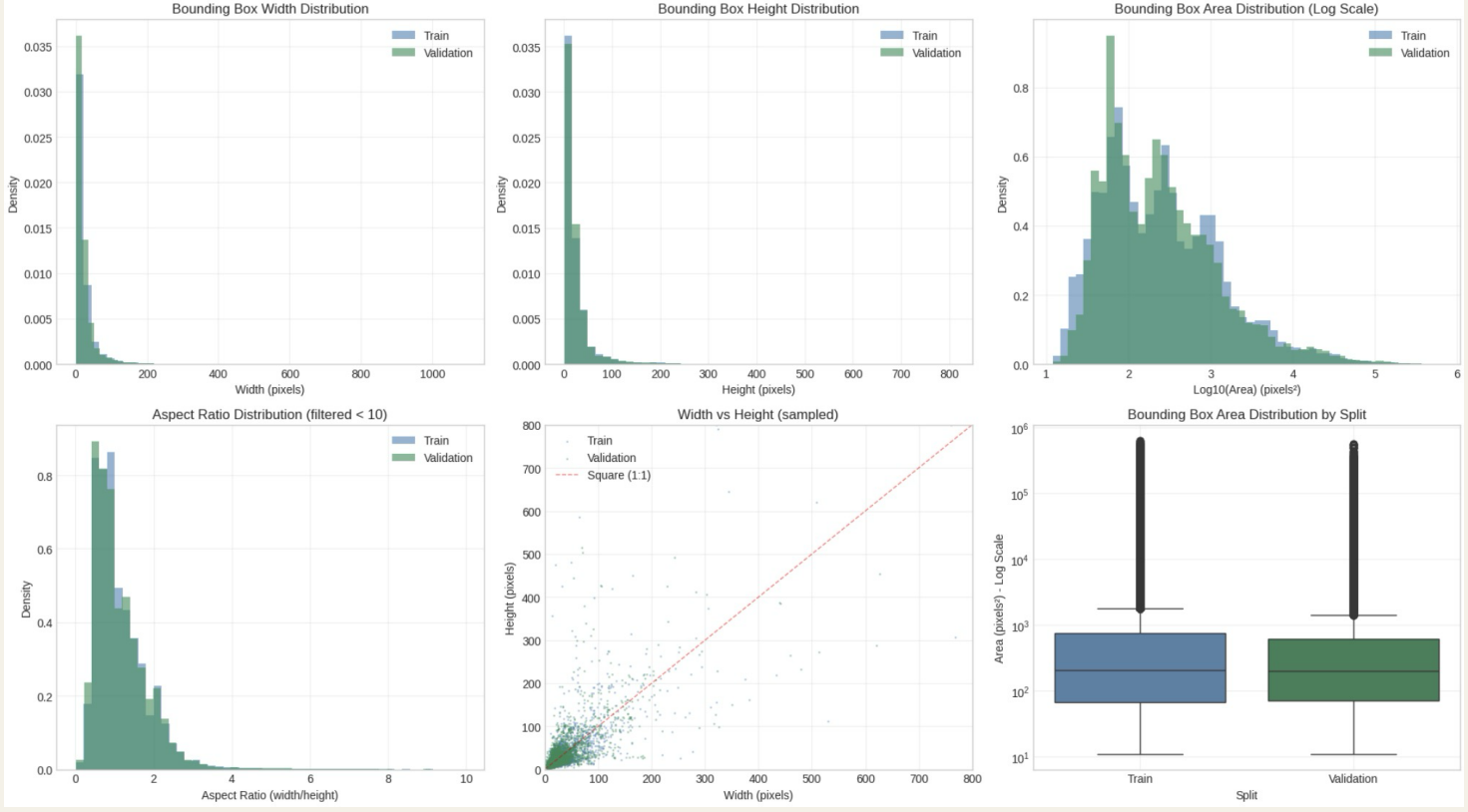


Figure 2.2: Histogram of number of objects per image + box plot with detected outliers

Bounding Box Size Distribution:


Bounding Box Width Distribution
Train
Validation
Density
Width (pixels)
Bounding Box Height Distribution
Train
Validation
Density
Height (pixels)
Bounding Box Area Distribution (Log Scale)
Train
Validation
Density
Log10(Area) (pixels²)
Aspect Ratio Distribution (filtered < 10)
Train
Validation
Density
Aspect Ratio (width/height)
Width vs Height (sampled)
Train
Validation
Square (1:1)
Height (pixels)
Width (pixels)
Bounding Box Area Distribution by Split
Area (pixels²) - Log Scale
Train
Validation
Split

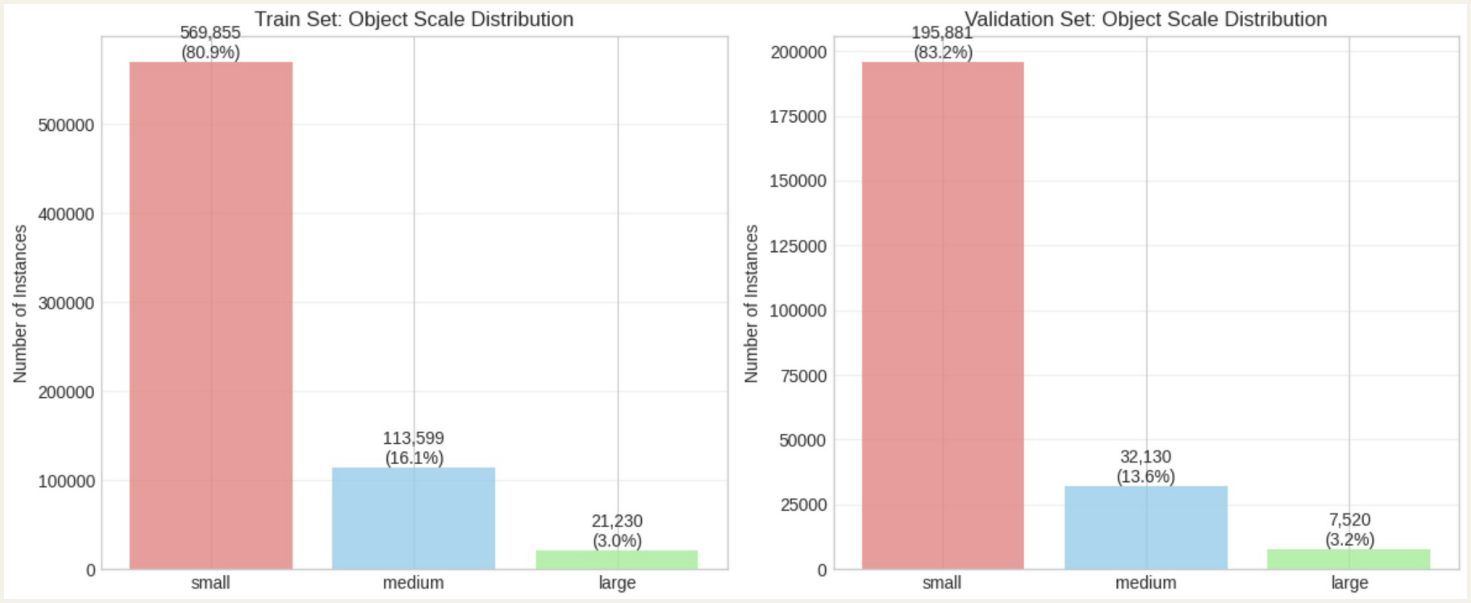

Train Set: Object Scale Distribution
569,855
(80.9%)
113,599
(16.1%)
21,230
(3.0%)
Number of Instances
small
medium
large
Validation Set: Object Scale Distribution
195,881
(83.2%)
32,130
(13.6%)
7,520
(3.2%)
Number of Instances
small
medium
large

Figure 2.3: Distribution of bounding box sizes (small objects dominate)

2.4 Sample Images

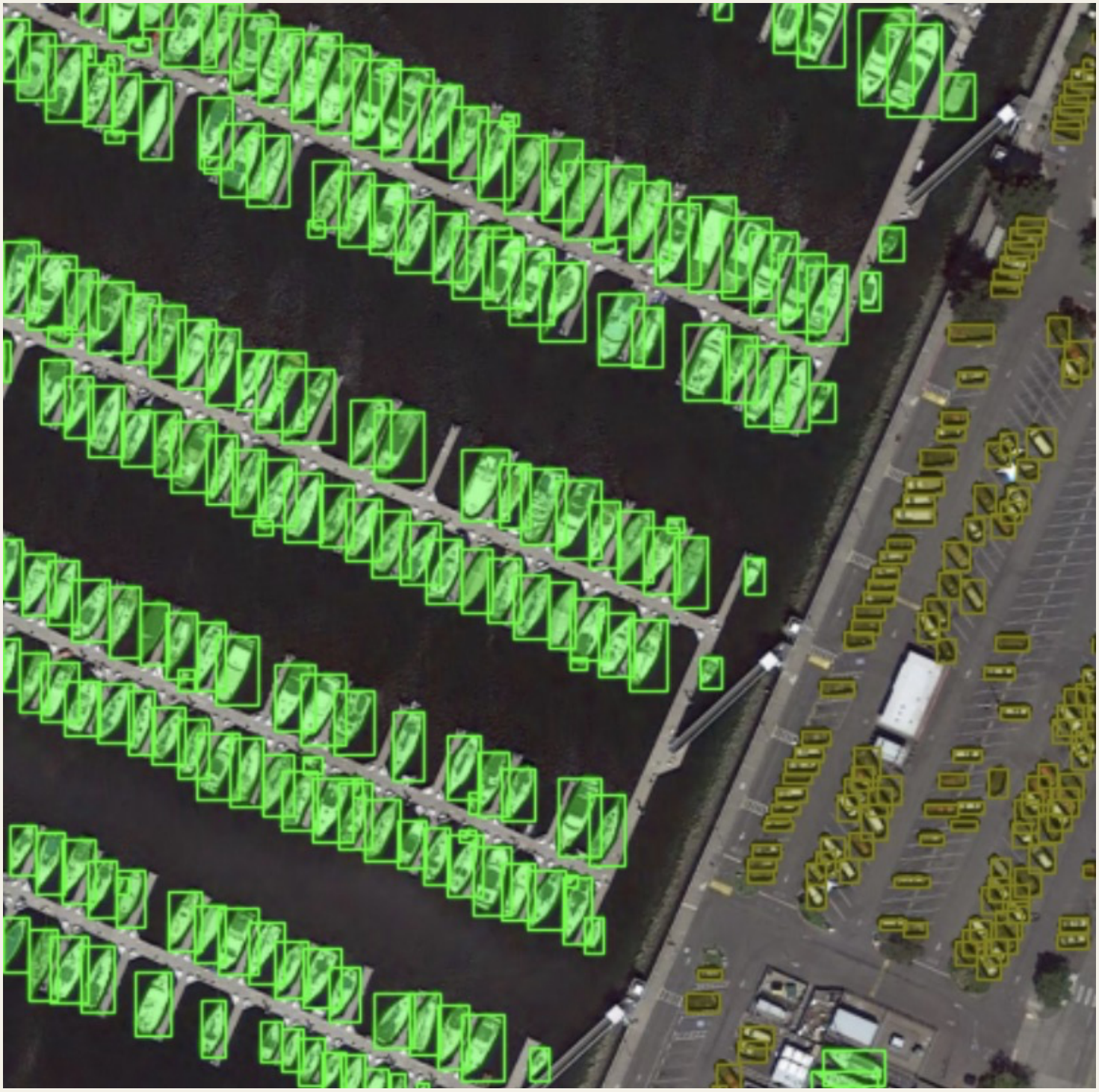


Figure 2.4: Sample image with ground truth annotations showing ships and vehicles

Figure 2.5: Sample image showing storage tanks and large vehicles

Figure 2.6: Sample image with planes at an airport

2.5 Dataset Preprocessing

The original high-resolution images are preprocessed into patches:

1. **Patching**: Original images are split into 800×800 patches with overlap
2. **Filtering**:
   - Images with >400 bounding boxes are excluded (memory optimization)
   - Empty images are limited to 30% of the dataset (training stability)
3. **Train/Val Split**: Standard DOTA-based split preserved

```
# Dataset filtering configuration
max_boxes_per_image: 400  # Filter outliers with extreme box counts
max_empty_fraction: 0.3   # Control empty image ratio
```

Final preprocessed dataset can be found here

2.5.1 Input Normalization

All input images are normalized using ImageNet mean and standard deviation values.
This step is required to ensure compatibility with the pretrained EfficientNet backbone, which was trained on ImageNet-scale data.

Operation	Parameters	Purpose
Normalize	mean = [0.485, 0.456, 0.406]std = [0.229, 0.224, 0.225]	Match ImageNet input distribution for stable training and faster convergence

No geometric or stochastic augmentations are applied at this stage.

---

3. Problem Description

3.1 Task Definition

Instance Segmentation is the task of detecting individual objects in an image and generating a pixel-level mask for each detected instance. Unlike semantic segmentation, which assigns a class to each pixel, instance segmentation distinguishes between different instances of the same class.

For each detected object, the model outputs:

- **Bounding Box**: `(x1, y1, x2, y2)` coordinates
- **Class Label**: One of 16 classes (including background)
- **Confidence Score**: Probability that the detection is correct
- **Instance Mask**: Binary mask indicating which pixels belong to the object

3.2 Challenges in Aerial Imagery

Aerial/satellite instance segmentation presents unique challenges:

Challenge	Description	Our Solution
Small Objects	Many objects (vehicles, ships) appear very small	Custom anchor sizes optimized for small objects
Dense Scenes	Images can contain hundreds of objects	Dataset filtering, efficient batch processing
Scale Variation	Objects vary from tiny cars to large fields	Multi-scale FPN with 4 levels
Class Imbalance	Vehicle classes dominate	Focal loss consideration (future work)
Similar Appearances	Different classes can look similar from above	CBAM attention for discriminative features
Partial Occlusion	Objects can be partially visible	RoI-based approach handles partial views

3.3 Evaluation Criteria

The primary evaluation metric is **mAP@0.5** (mean Average Precision at IoU threshold 0.5), following COCO evaluation protocols.

4. Model Architecture

4.1 Proposed Architecture: Attention-Enhanced EfficientNet Mask R-CNN

To address the specific challenges of high-precision instance segmentation in aerial imagery, we designed a custom modular architecture that integrates attention mechanisms and enhanced regularization into the standard Mask R-CNN framework.

4.1.1 Backbone: EfficientNet-B0 with Dual-Stage Attention

We replaced the standard ResNet backbone with EfficientNet-B0, initialized with ImageNet-1K weights, to optimize the balance between parameter efficiency and feature extraction capability.

- Embedded Attention (CBAM): To refine feature discriminability, we embedded Convolutional Block Attention Modules (CBAM) directly into the backbone at feature stages C_2 through C_5. These modules sequentially apply channel and spatial attention maps to emphasize relevant features while suppressing background noise.
- Attention-FPN: We developed a custom Feature Pyramid Network (AttentionFPN). Unlike a standard FPN, this module applies an additional CBAM block to every output level of the pyramid (P_2–P_5). This ensures that multi-scale features are denoised before reaching the detection heads.

4.1.2 Region Proposal Network (RPN)

The RPN was fine-tuned for small object detection. We utilized anchor sizes and aspect ratios specifically optimized for the dataset statistics (see anchor optimization notebook). Crucially, we lowered the foreground IoU threshold to 0.5 (from the default 0.7). This adjustment increases the recall for difficult, small objects by treating lower-overlap proposals as positive training samples.

4.1.3 Custom RoI Box Head (Regularized)

We replaced the standard Fast R-CNN predictor with a deeper, regularized architecture to prevent overfitting:

- Structure: Aligned RoI features (7×7) are processed by a custom feature extractor consisting of two fully connected (FC) layers.
- Dropout Regularization: To improve generalization, we injected Dropout layers ($p = 0.3$) between the FC layers and before the final prediction heads.
- Parallel Outputs: The head splits into two parallel branches for class classification scores and bounding box regression deltas.

4.1.4 Custom RoI Mask Head (Residual)

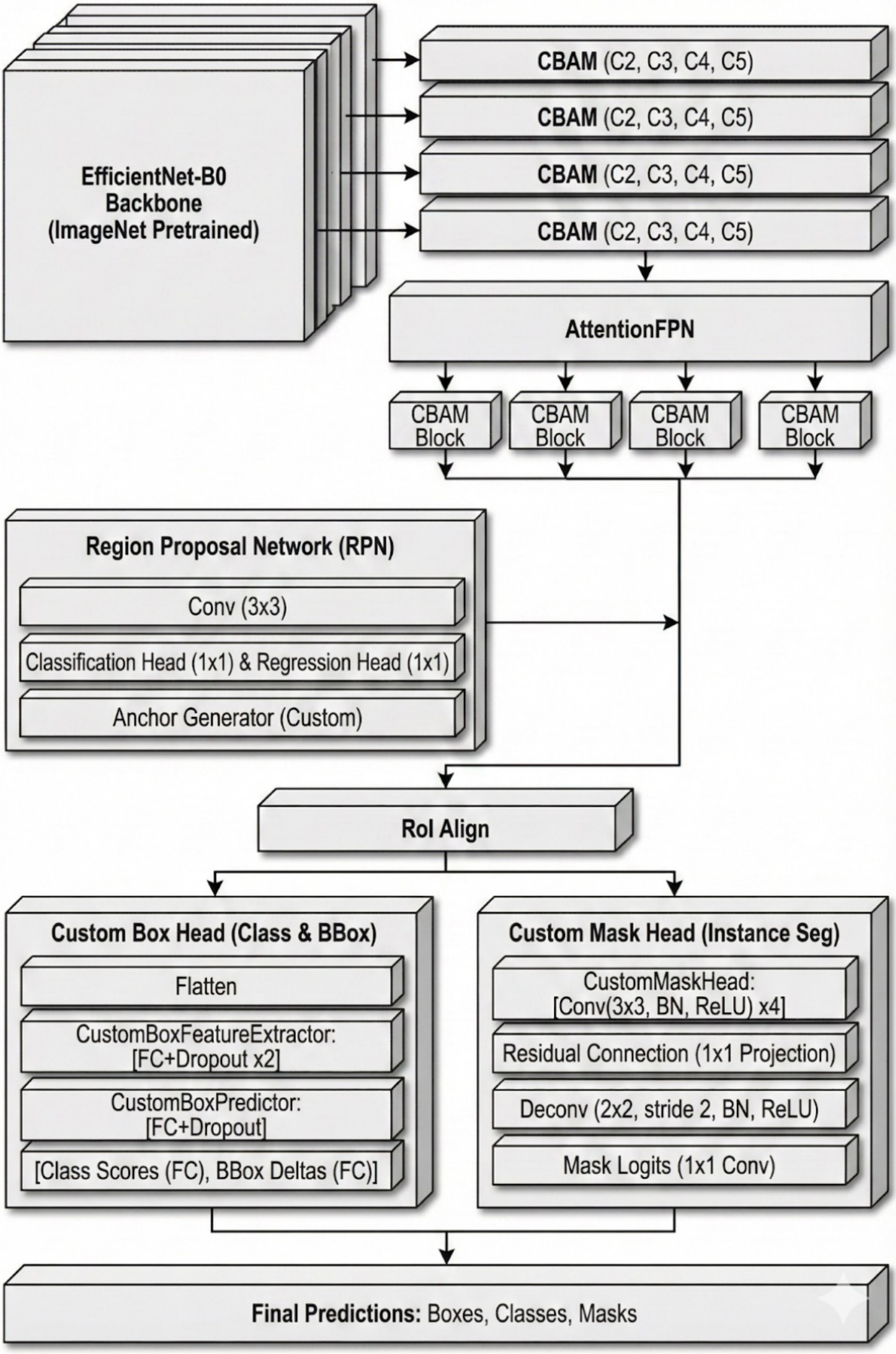
To capture finer spatial details, we implemented a robust "CustomMaskHead" that improves gradient flow and feature retention:

- Convolutional Stack: The 14×14 RoI features pass through four sequential 3×3 convolutional layers (with BatchNorm and ReLU).
- Residual Connection: We introduced a residual skip connection ($x = x + identity$) around the convolutional stack, facilitated by a 1×1 projection layer. This residual learning aids in training deeper mask branches.
- Upsampling: Features are upsampled via a Deconvolution (ConvTranspose2d) layer before the final 1×1 convolution generates pixel-wise mask logits.

4.1.5 Initialization:

All custom components (Attention modules, FPN, and Heads) were initialized using Kaiming Normal (He) initialization to ensure stable training dynamics.

The EfficientNet-B0 has initial weights loaded to speed up the training.


EfficientNet-B0
Backbone
(ImageNet Pretrained)
CBAM (C2, C3, C4, C5)
CBAM (C2, C3, C4, C5)
CBAM (C2, C3, C4, C5)
CBAM (C2, C3, C4, C5)
AttentionFPN
CBAM Block
CBAM Block
CBAM Block
CBAM Block
Region Proposal Network (RPN)
Conv (3x3)
Classification Head (1x1) & Regression Head (1x1)
Anchor Generator (Custom)
RoI Align
Custom Box Head (Class & BBox)
Flatten
CustomBoxFeatureExtractor:
[FC+Dropout x2]
CustomBoxPredictor:
[FC+Dropout]
[Class Scores (FC), BBox Deltas (FC)]
Custom Mask Head (Instance Seg)
CustomMaskHead:
[Conv(3x3, BN, ReLU) x4]
Residual Connection (1x1 Projection)
Deconv (2x2, stride 2, BN, ReLU)
Mask Logits (1x1 Conv)
Final Predictions: Boxes, Classes, Masks

4.2 Baseline Architectures: Standard ResNet Mask R-CNN

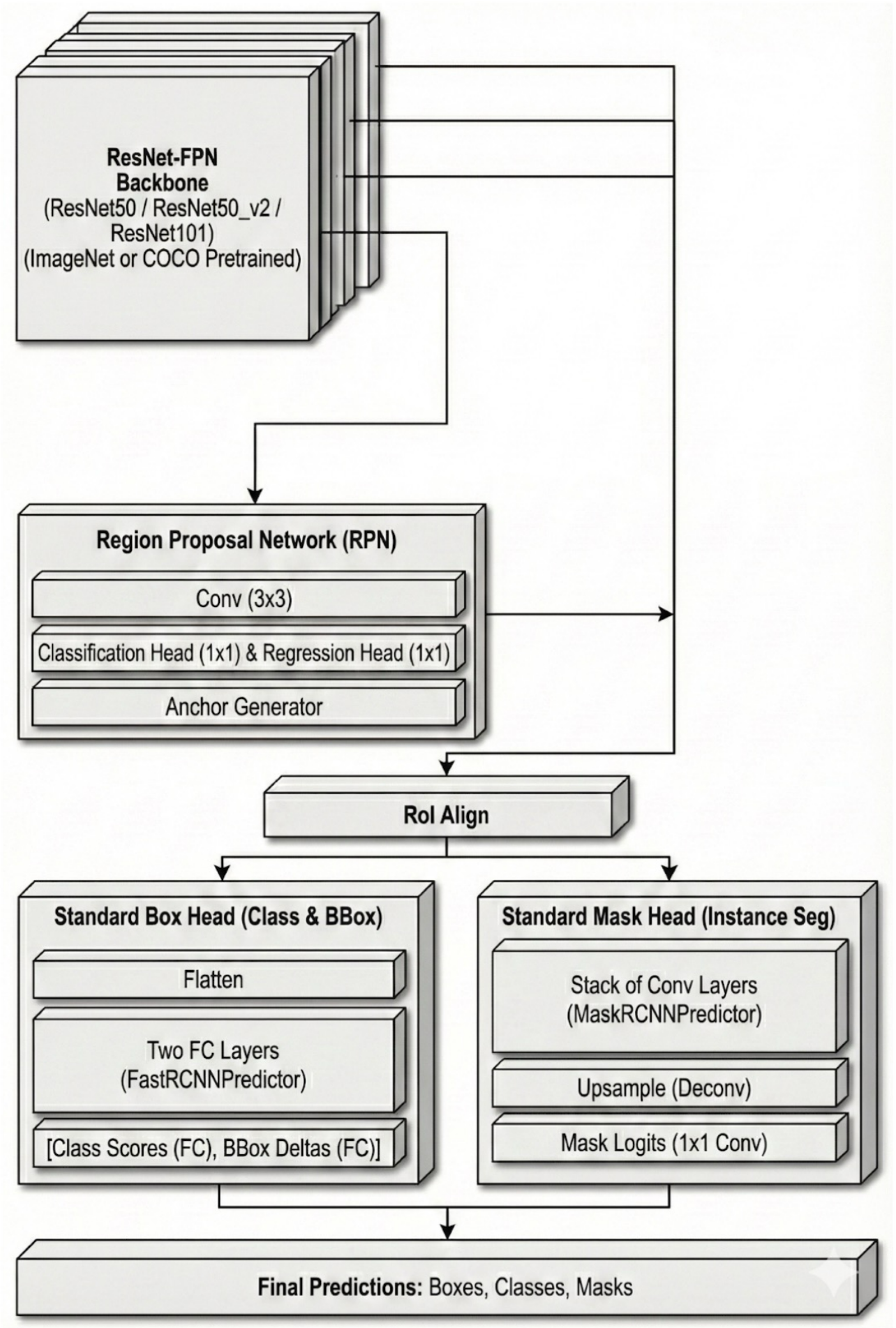
For comparative analysis, we evaluated two standard configurations of the Mask R-CNN architecture. These models serve as baselines to benchmark the efficiency and accuracy of our custom attention-based model.

4.2.1 ResNet-50-FPN (COCO Transfer Learning)

- Backbone & RPN: A standard ResNet-50 (V1) with a Feature Pyramid Network (FPN). When enabled, this variant is initialized with COCO V1 pretrained weights. This means the backbone, FPN, and Region Proposal Network (RPN) start with learned features for general object detection.
- Heads: The final Box and Mask predictors were replaced to match our dataset's 16 classes. Consequently, the prediction heads are initialized randomly, while the feature extraction layers benefit from transfer learning.
- Alternative: When COCO pretraining is disabled, the backbone is initialized with ImageNet-1K weights, and the rest of the network is initialized randomly.

4.2.2 ResNet-101-FPN (ImageNet Initialization)

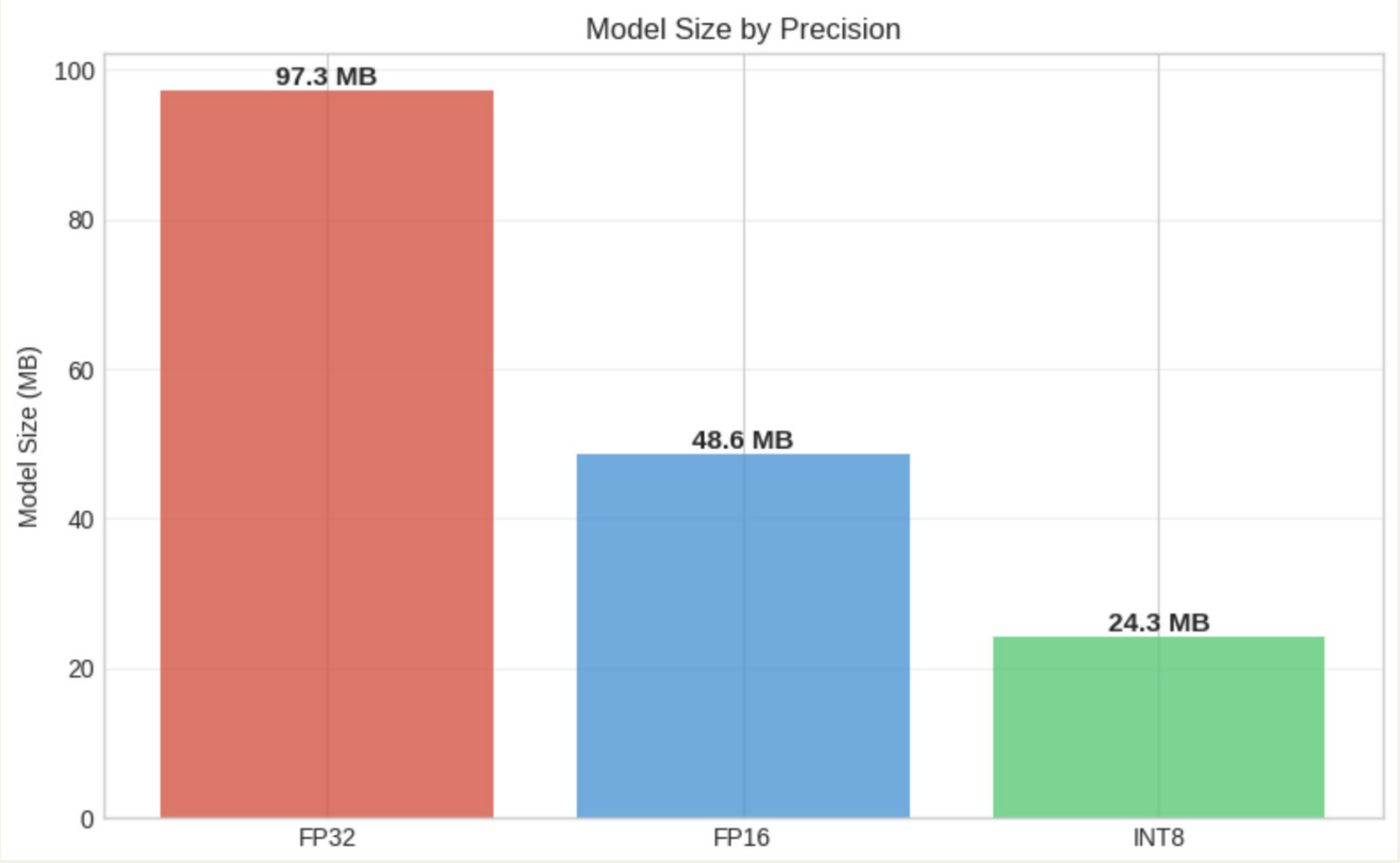
- Backbone: A deeper ResNet-101 backbone constructed manually. Unlike the ResNet-50 baseline, this model does not leverage COCO detection weights. Instead, the backbone is initialized with ImageNet-1K classification weights.
- Full Training: The FPN, RPN, and all detection heads are initialized from scratch (random initialization).
- Depth: All 5 layers of the backbone were set to be trainable (trainable_layers=5), allowing the deep residual network to fully adapt its feature hierarchy from classification tasks to the aerial segmentation domain.



5. Model Analysis

5.1 Model Size

Metric	Value
Total Parameters	~24.32 M
Trainable Parameters	~24.32 M
Model Size (FP32)	~97.26 MB
Model Size (FP16)	~48.63 MB
Model Size (INT8)	~24.32 MB



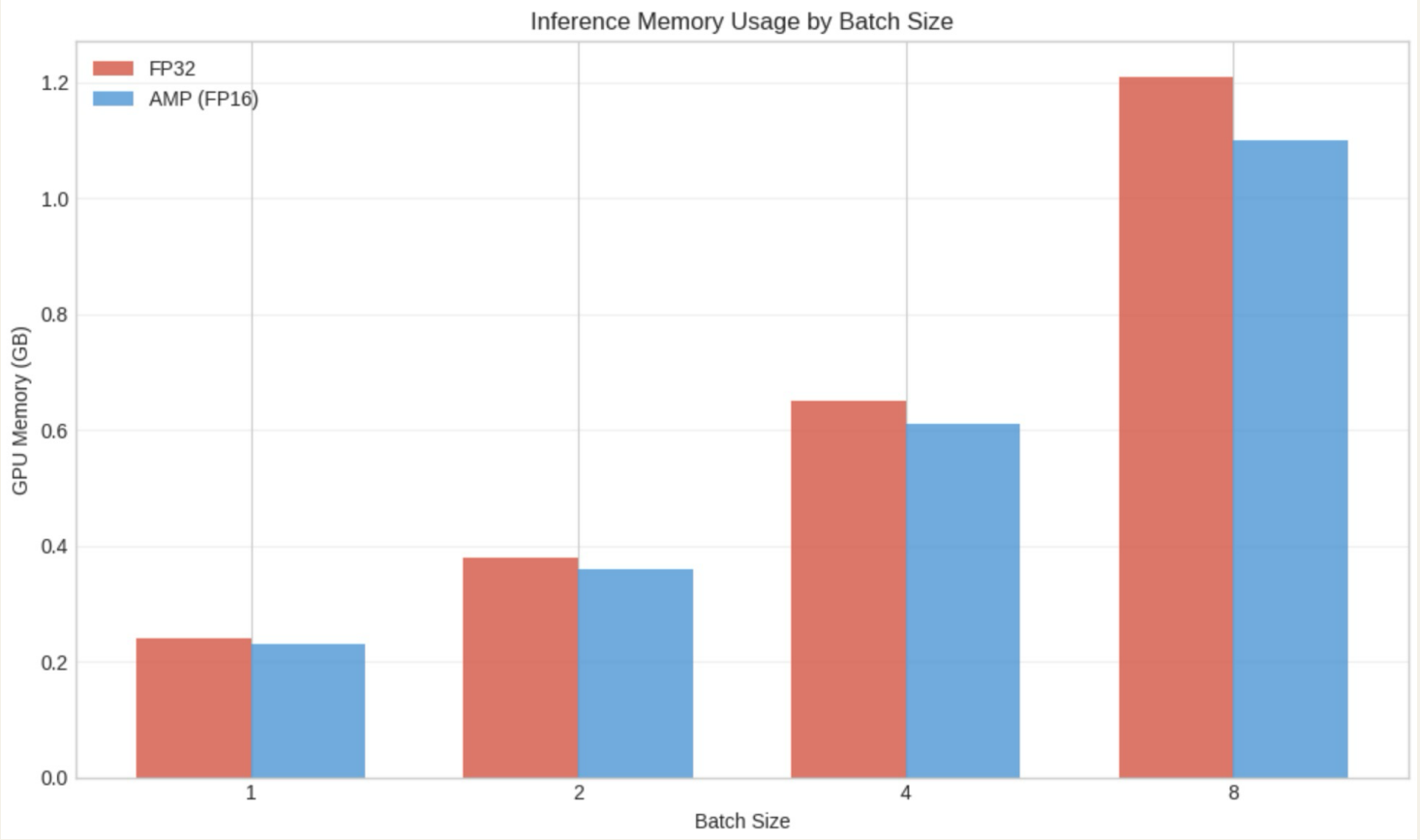
5.2 Parameter Breakdown by Component

Component	Parameters	Percentage
Backbone (EfficientNet-B0)	~6.56 M	26.98%
--- Backbone (base)	~6.33 M	26.05%
------ FPN	~2.54 M	10.44%
--- Backbone (CBAM Modules)	~227.90 K	0.94%
RPN	~597.79 K	2.46%
Box Head	~13.9 M	57.15%
Box Predictor	~565.84 K	2.33%
Mask Head	~2.70 M	11.08%
Total	~24.32 M	100%

5.3 Memory Requirements

Configuration	GPU Memory
Training (batch_size=8, AMP)	~30 GB
Training (batch_size=4, AMP)	~20 GB
Inference (single image)	~2 GB

5.4 Memory Usage (Inference)



Note that the training memory usage might be couple times larger, because of gradients and optimizer states

6. Training Description

6.1 Training Commands

Option 1: Using the Training Script (Recommended)

```
# Basic training with default config
python train.py --config config.yaml --data-root /path/to/iSAID_patches

# Training with custom parameters
python train.py \
  --config config.yaml \
  --data-root /path/to/iSAID_patches \
  --output-dir ./checkpoints \
  --epochs 25 \
  --batch-size 8 \
  --lr 0.0003

# Training without W&B logging
python train.py --config config.yaml --data-root /path/to/data --no-wandb

# Training with LR finder
python train.py --config config.yaml --data-root /path/to/data --find-lr

# Resume from checkpoint
python train.py --config config.yaml --resume checkpoints/last.pth
```

Option 2: Run one of the provided notebooks (GitHub)

6.2 Configuration File

The training is configured via `config.yaml` :

```yaml
# Dataset
data:
  root_dir: "iSAID_patches"
  num_classes: 16
  image_size: 800
  max_boxes_per_image: 400
  max_empty_fraction: 0.3

# Model
model:
  backbone: "efficientnet_b0"
  pretrained_backbone: true

# Training
training:
  epochs: 25
  batch_size: 8
  learning_rate: 0.0003
  weight_decay: 0.01
  amp: true
```

6.3 Training Pipeline

1. **Data Loading**: Images and annotations loaded with augmentations
2. **Forward Pass**: Image → Backbone → FPN → RPN → RoI Heads
3. **Loss Computation**: Multi-task loss (classification + regression + mask)
4. **Backward Pass**: Gradient computation with gradient clipping (max_norm=1.0)
5. **Optimization**: AdamW with differential learning rates

```
param_groups = [
        {"params": base_params, "lr": lr},
        {"params": roi_params, "lr": lr * roi_lr_alpha,
                    "name": "roi_heads"},]
optimizer = AdamW(param_groups)
```

Lower learning rate for RoI heads was used to prevent overfitting and stabilize training in later stages

6. **Validation**: Periodic mAP computation on validation set
7. **Checkpointing**: Save best model by validation loss and mAP

6.4 Data Augmentation

Training augmentations (applied with Albumentations):

Augmentation	Parameters	Purpose
Horizontal Flip	p=0.5	Increase viewpoint invariance
Color Jitter	brightness=0.2, contrast=0.2	Handle lighting variations

7. Metrics and Evaluation

7.1 Loss Functions

The total loss is a combination of five components:

$$\mathcal{L}_{total} = \mathcal{L}_{rpn\_cls} + \mathcal{L}_{rpn\_reg} + \mathcal{L}_{roi\_cls} + \mathcal{L}_{roi\_reg} + \mathcal{L}_{mask}$$

Loss Component	Type	Description
RPN Classification	Binary Cross-Entropy	Object vs. background
RPN Regression	Smooth L1	Anchor box refinement
RoI Classification	Cross-Entropy	Multi-class classification
RoI Regression	Smooth L1	Bounding box refinement
Mask Loss	Binary Cross-Entropy	Per-pixel mask prediction

7.2 Evaluation Metrics

7.2.1 mAP (mean Average Precision)

The primary metric, computed as:

1. AP is computed following the COCO evaluation protocol at IoU threshold 0.5.
2. Average across all classes

$$mAP = \frac{1}{N_{classes}} \sum_{c=1}^{N_{classes}} AP_c$$

7.2.2 IoU (Intersection over Union)

$$IoU = \frac{|A \cap B|}{|A \cup B|}$$

Where A is the predicted box/mask and B is the ground truth.

7.2.3 Mean IoU

Average IoU across all matched predictions and ground truths.

7.3 Evaluation Protocol

- **IoU Threshold**: 0.5 (following COCO protocol)
- **Score Threshold**: 0.5 for positive detections
- **Max Detections**: 100 per image
- **Evaluation Samples**: 200 images (for computational efficiency)

8. Hyperparameters

8.1 Hyperparameter Table

Hyperparameter	Value	Rationale
Learning Rate	3e-4	Estimated based on batch size and observations; standard for AdamW
Batch Size	8	Maximum fitting in GPU memory with AMP
Epochs	25	Sufficient for convergence on this dataset
Weight Decay	0.001	Standard regularization for AdamW
Image Size	800	Due to inconsistencies in patch generation, all images were explicitly resized to 800×800
RoI LR Multiplier	0.25	Lower LR for RoI heads prevents overfitting
Gradient Clip Norm	1.0	Stabilizes training, prevents exploding gradients
AMP (Mixed Precision)	True	2× memory savings, faster training
Dropout	0.3	Regularization in box/mask heads
CBAM Reduction	16	Balance between capacity and computation

8.2 Anchor Hyperparameters

FPN level	stride	Anchot Sizes (px)	Aspect Ratios	Target Effective Receptive Field
P2	4	8, 16	1:2, 1:1, 2:1	Small objects (Small Vehicles, Ships)
P3	8	16, 32	1:2, 1:1, 2:1	Small-medium objects (Large Vehicles, Planes)
P4	16	32, 64	1:2, 1:1, 2:1	Medium objects (Storage Tanks, Harbor)
P5	32	64, 128	1:2, 1:1, 2:1	Large objects (Fields, Courts)

8.3 Scheduler Configuration

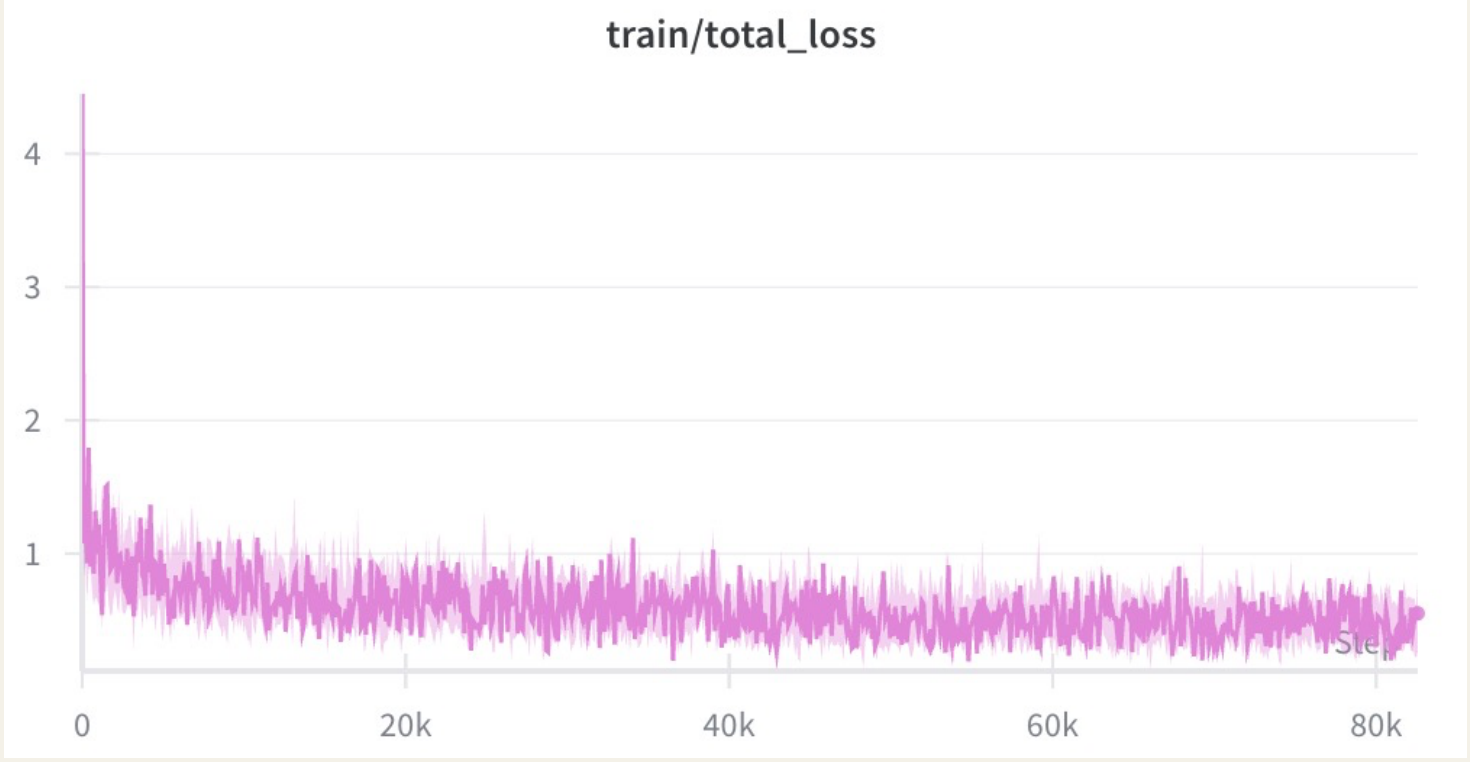
ReduceLROnPlateau:

Parameter	Value	Rationale
Mode	max	Monitoring mAP (higher is better) - our primary metric
Factor	0.5	Halve LR on plateau
Patience	3	Wait 3 epochs before reducing
Min LR	1e-6	Lower bound for learning rate

9. Results and Plots

9.1 Training Curves

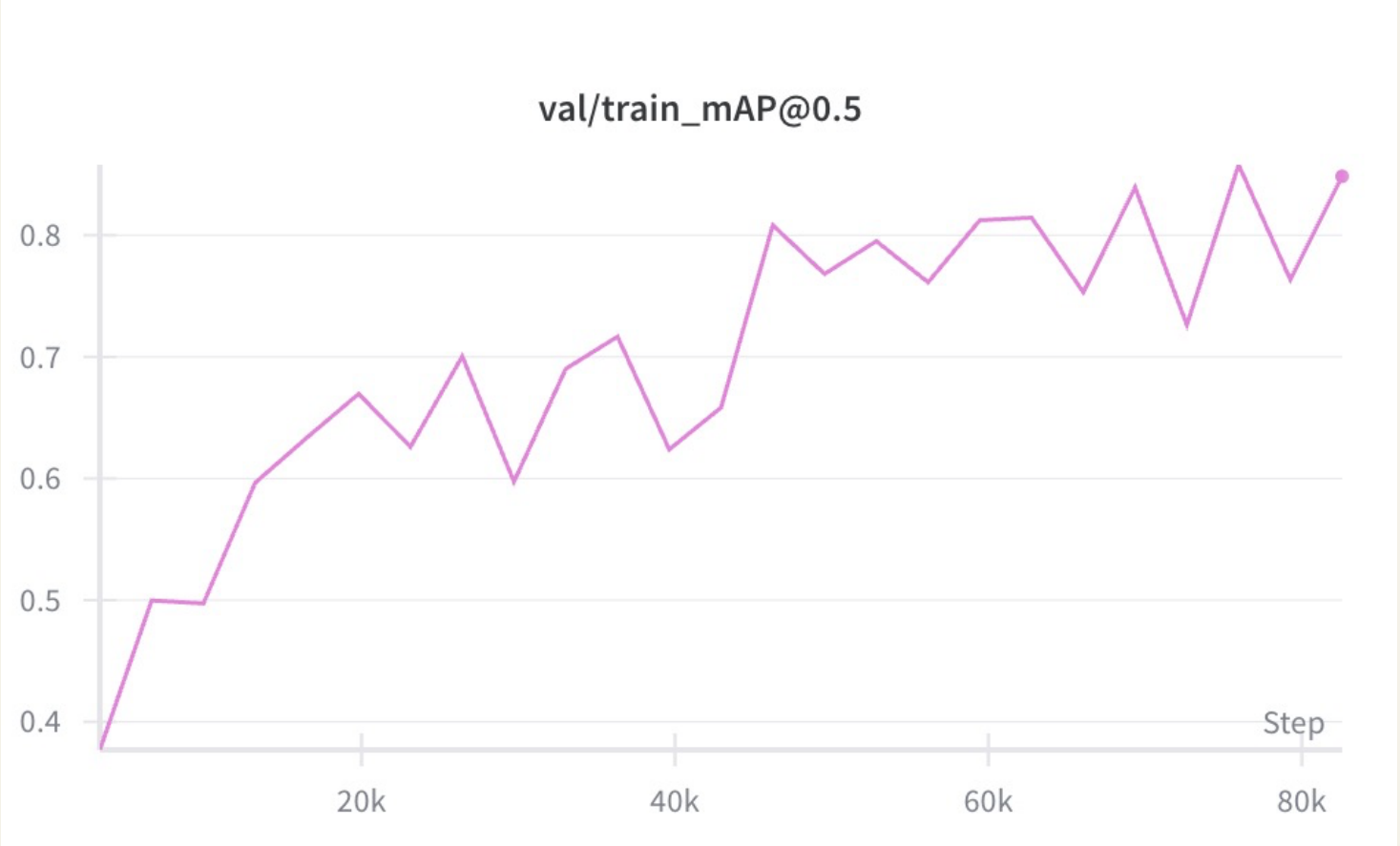
9.1.1 Loss Curves


train/total_loss
4
3
2
1
0
20k
40k
60k
80k


val/loss
1
0.95
0.9
0.85
0.8
Step
20k
40k
60k
80k

Figure 9.1: Training and validation loss over epochs

9.1.2 mAP Curves



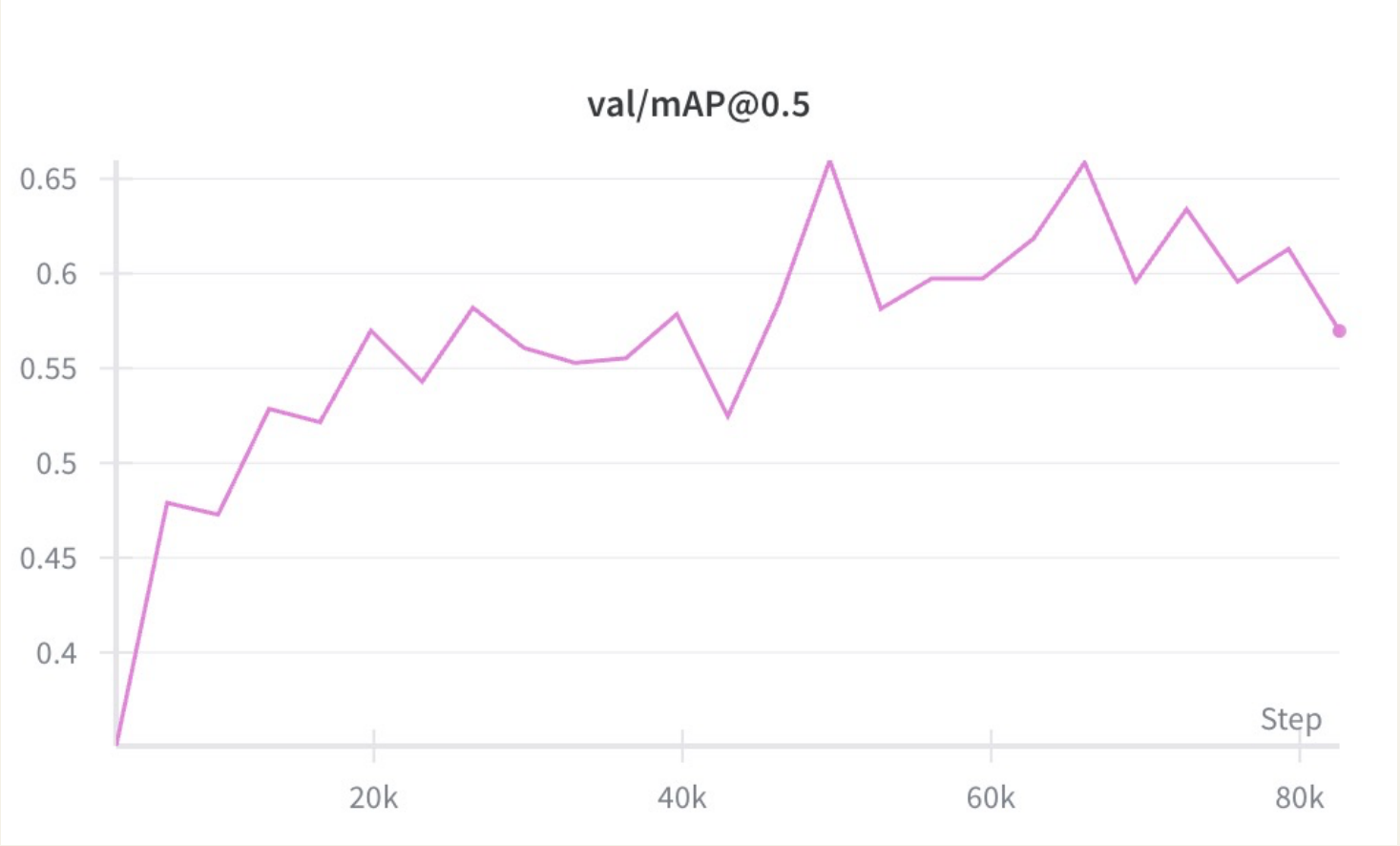


Figure 9.2: Training and validation mAP@0.5 over epochs

9.1.3 Learning Rate Schedule

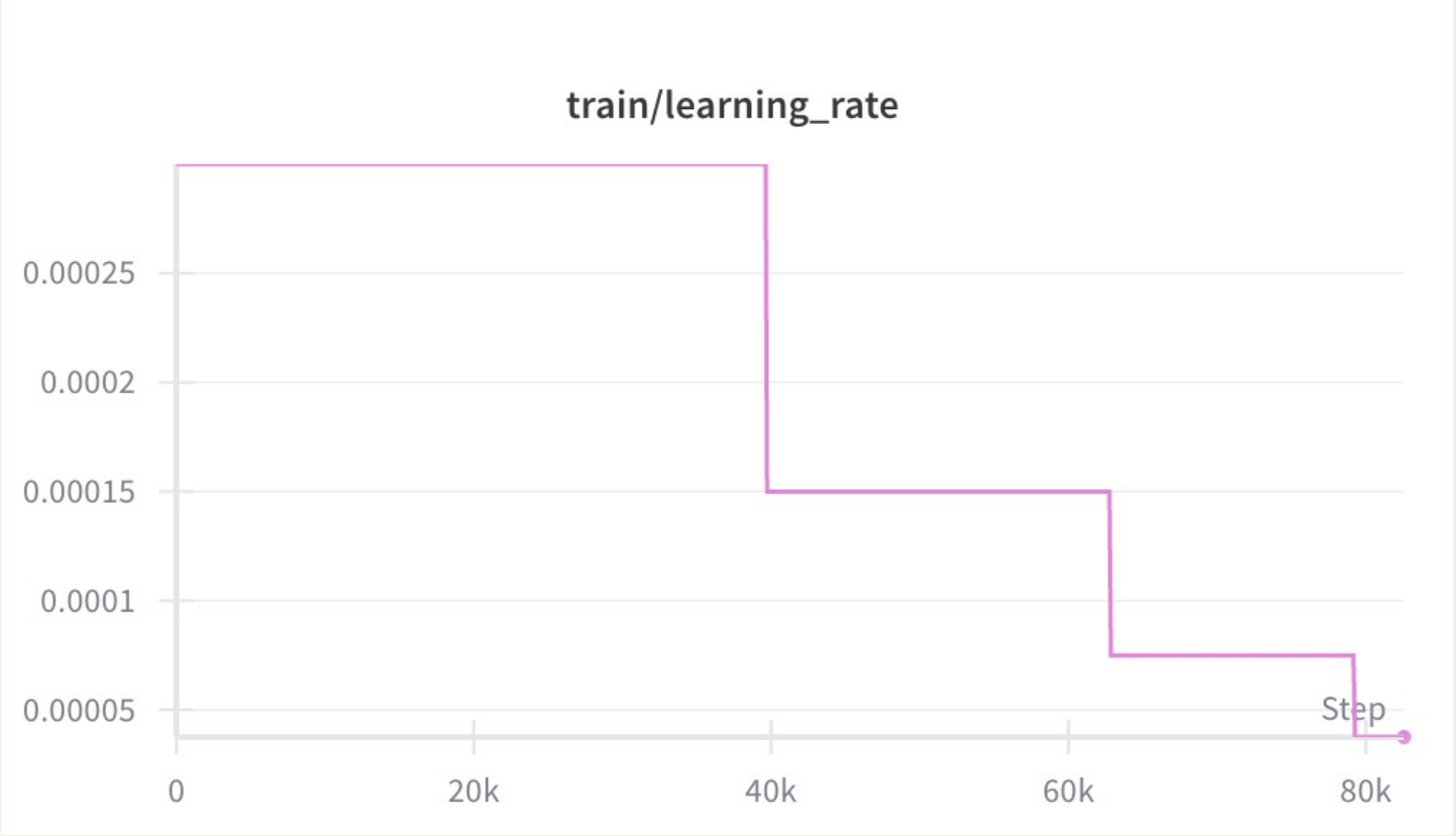


Figure 9.3: Learning rate schedule with ReduceLROnPlateau

9.1.4 Gradient Norms

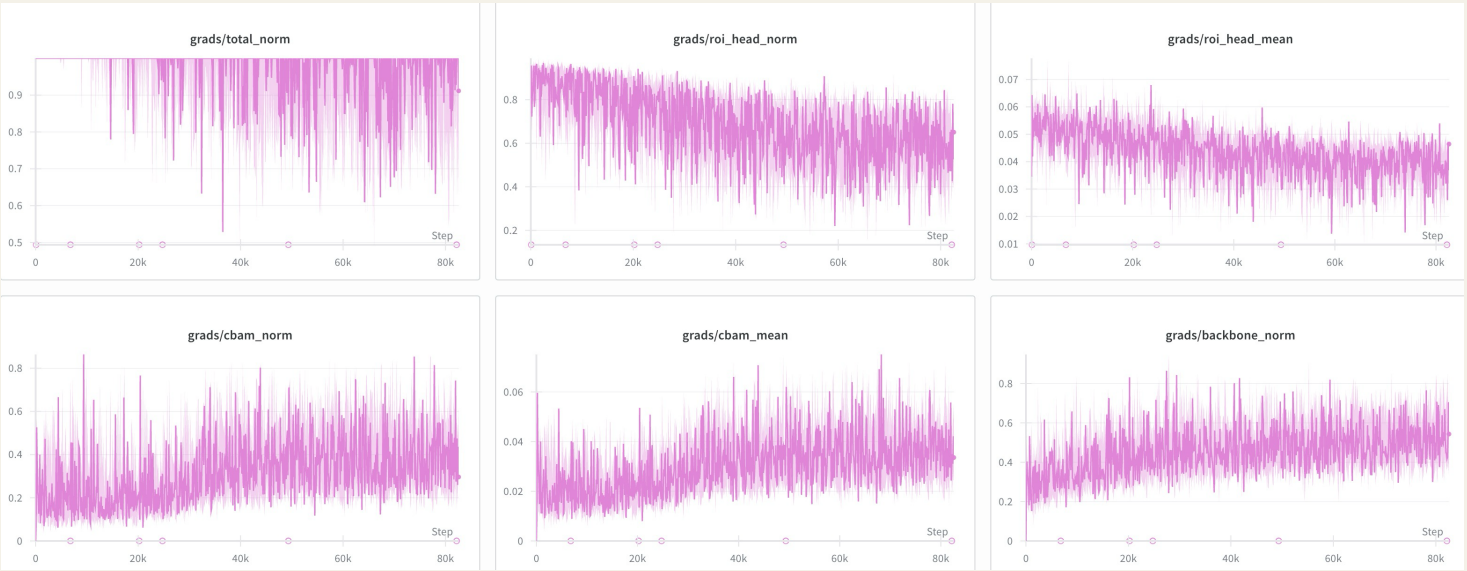


Figure 9.4: Gradient norm during training (stability indicator)

9.2 Final Results

Metric	Train	Validation
Total Loss	0.4874	0.8425
mAP@0.5	0.8579	0.5958
Mean IoU	-	0.5804

9.3 Qualitative Results

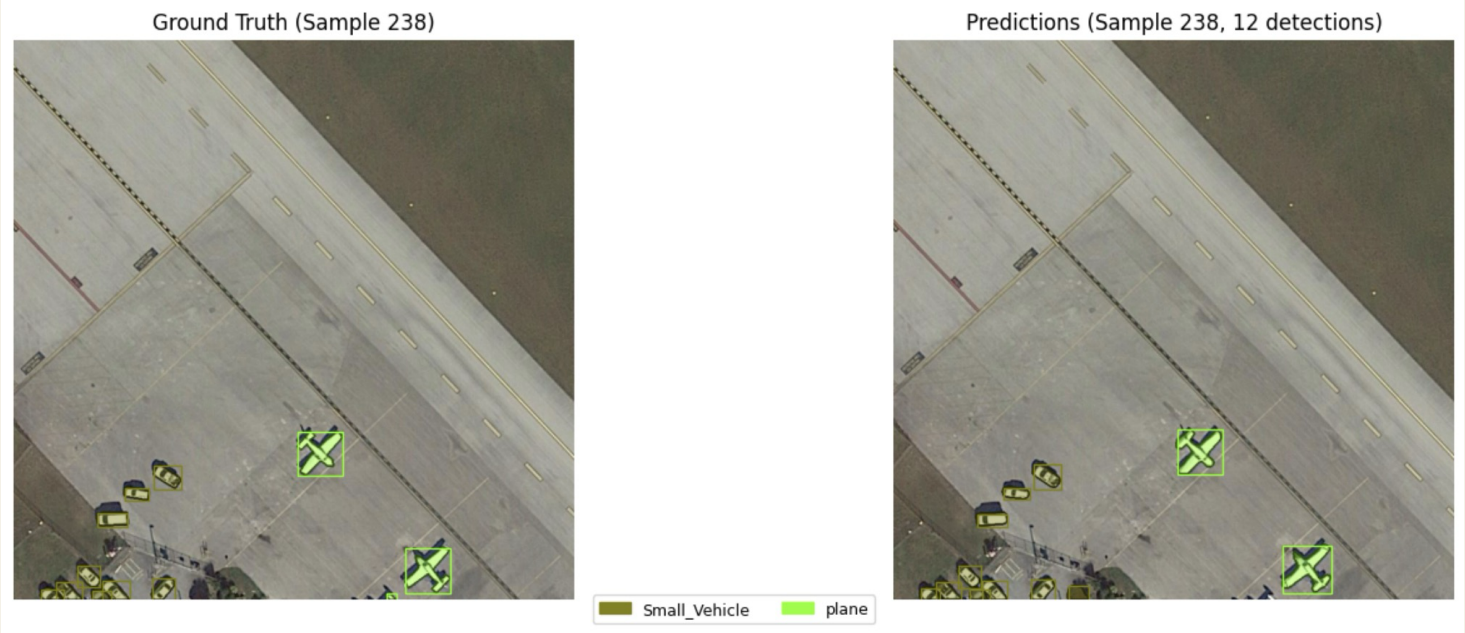


Figure 9.5: Model predictions on validation sample 1

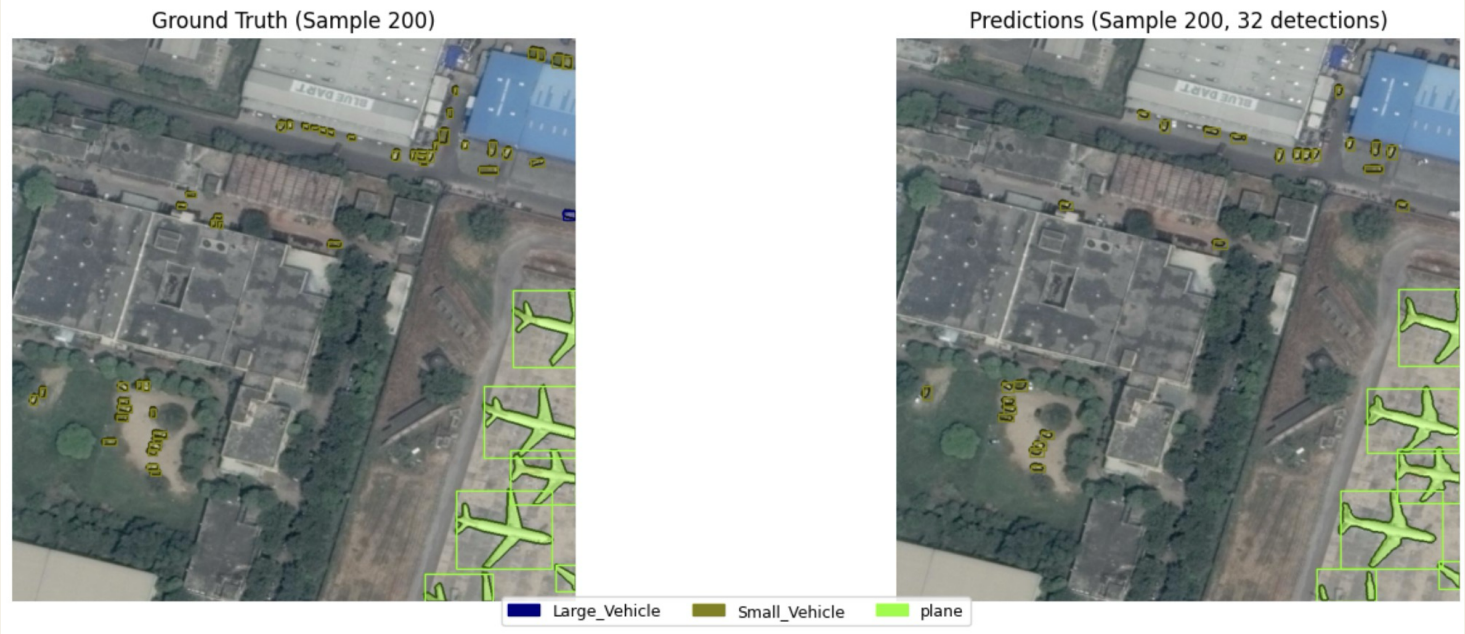


Figure 9.6: Model predictions on validation sample 2

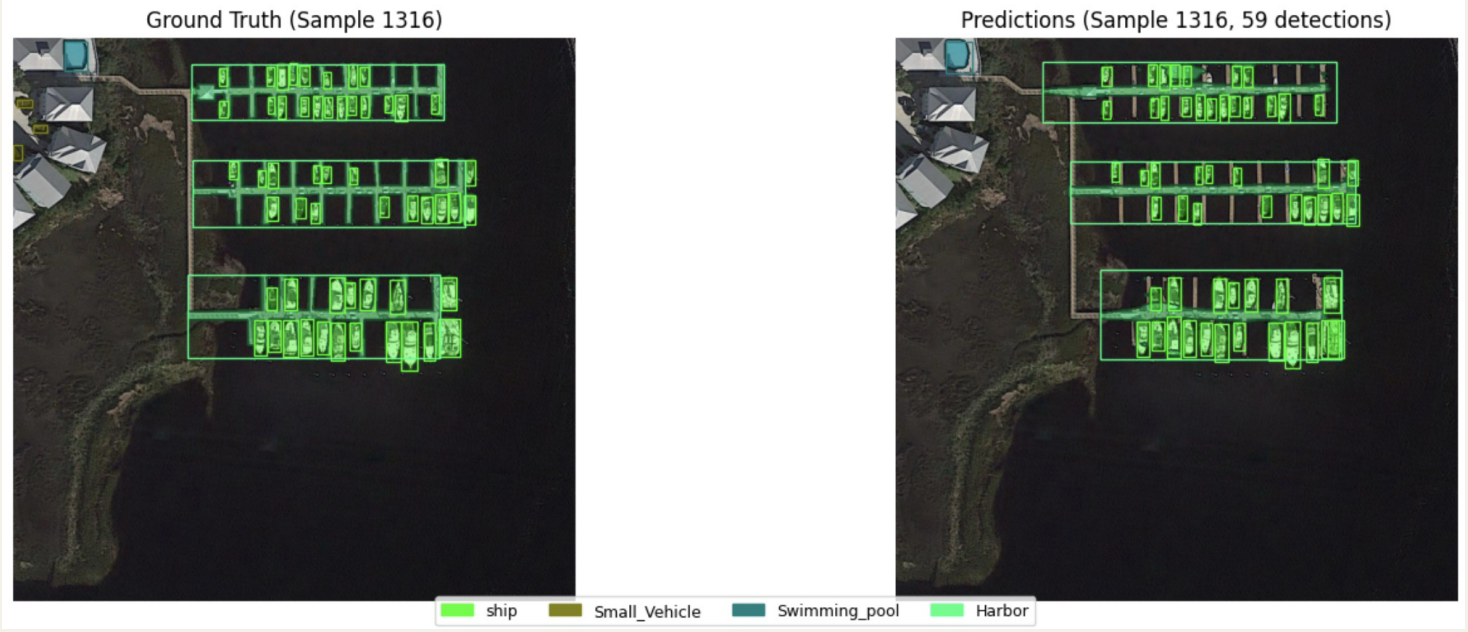


Figure 9.7: Model predictions on validation sample 3

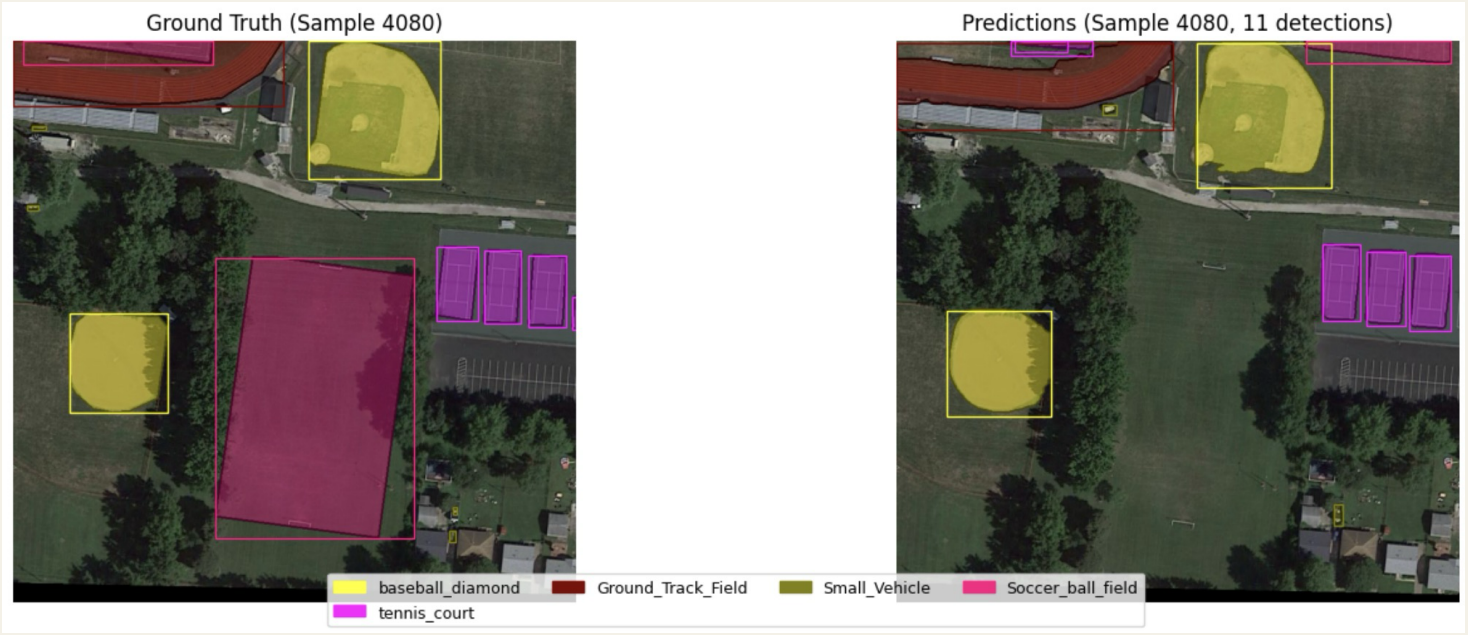


Figure 9.8: Model predictions on validation sample 4

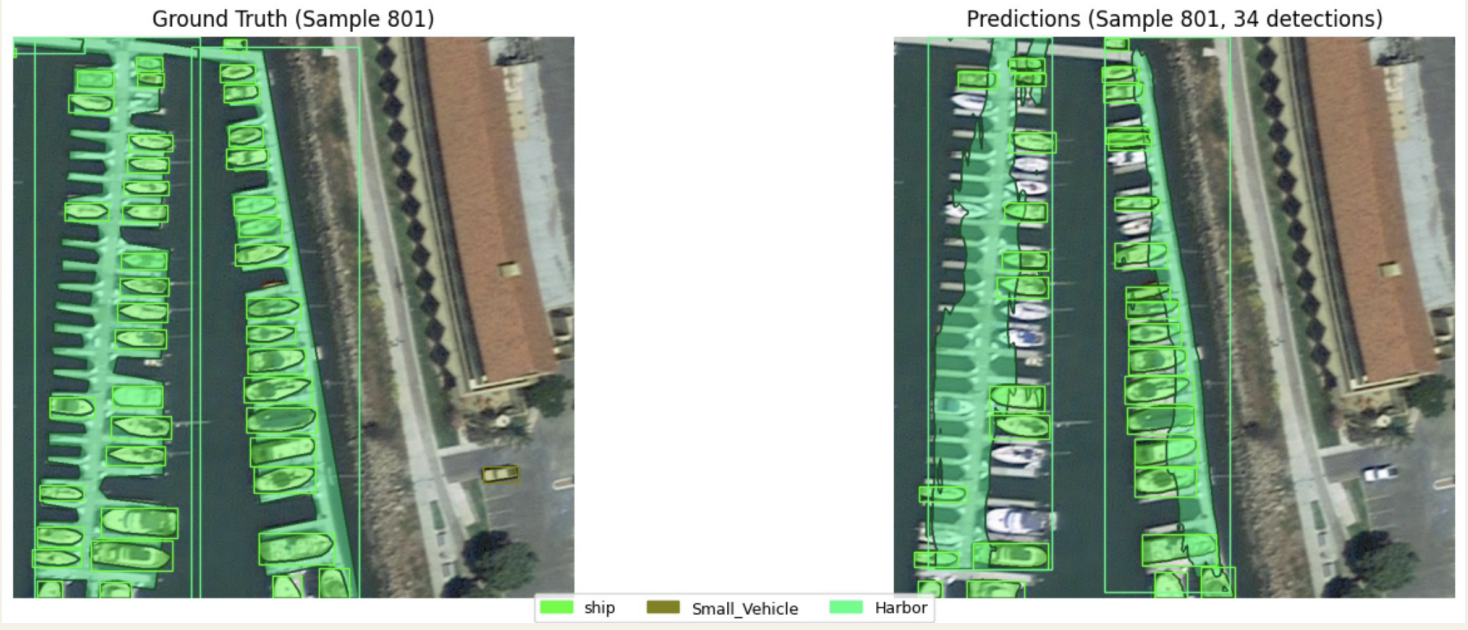


Figure 9.9: Model predictions on validation sample 5 (challenging - blurred dense image)

10. Model Comparison

10.1 Backbone Comparison

Training settings for all:

- "batch_size": 8,
- "val_batch_size": 16,
- "num_epochs": 25,
- NVIDIA A100-SXM4-40GB

Metric	Custom (EfficientNet+CBAM)	ResNet-50 (Baseline)	ResNet-101 (Heavy)
Parameters	24.3 M (Lowest)	44.0 M	62.9 M
Model Size (Weights)	276 MB	502 MB	756 MB
Training Time (Total)	11h (25 epochs)	6h (13 epochs)*	16h (25 epochs)
Epoch Time (Avg)	~22 min	~27 min	~41 min
Best mAP@0.5	**0.6597** (Highest)	0.6239	0.6180
Best Val Loss	0.7929	0.6595	**0.6610**

**ResNet-50 training was stopped early at 13 epochs.*

Analysis of Results

1. Custom Model (EfficientNet-B0 + CBAM): This architecture achieved the highest accuracy (0.66 mAP) while being the most parameter-efficient (63% smaller than ResNet-101). The success is attributed to the optimized anchor configuration and CBAM attention, which allowed the model to detect tiny objects that the baselines missed. The higher validation loss (0.79) alongside the best mAP confirms the RPN was successfully mining "hard" examples (small vehicles) that were penalty-heavy but increased Recall.
2. ResNet-50 (Baseline): Served as a robust baseline (0.62 mAP). Its performance benefitted significantly from COCO transfer learning, allowing it to converge rapidly (6 hours) and outperform the deeper ResNet-101. This highlights the value of initialization over pure depth.
3. ResNet-101: Despite being the heaviest computation (16h training) and deepest model, it yielded the lowest performance (0.61 mAP). Without COCO detection weights (ImageNet only), the massive parameter count led to overfitting and optimization difficulties, proving that increased model capacity is detrimental on the iSAID dataset without strong regularization or domain-specific adaptations.

10.2 Anchor Optimization Results

We evaluated many distinct anchor strategies to determine the optimal balance between coverage (Recall@0.5) and localization precision (Recall@0.7). The "Stride Adjusted" configuration was selected as it provides the best trade-off, maintaining high recall for small objects while ensuring anchors are physically alignable with the feature map strides.

Configuration	Anchor Sizes (P2 / P3 / P4 / P5)	Recall@0.5	Recall@0.7	Recall@0.75
Stride Adjusted (Selected)	(8, 16) / (16, 32) / (32, 64) / (64, 128)	0.8397	0.2956	0.1578
Data-Suggested - from BBox analysis	(8, 12) / (16, 24) / (32, 48) / (64, 96)	0.8478	0.2417	0.1127
Initial Guess	(16, 24) / (32, 48) / (64, 96) / (128, 192)	0.5303	0.3170	0.1767
Opt1 - optuna trials winner	(12, 24) / (64, 96) / (48, 144) / (64, 480)	0.7154	0.2005	0.1151

11. Challenges and Solutions

11.1 Memory Management

Challenge: Training on high-resolution images with dense annotations causes GPU memory overflow.

Solutions:

- Mixed precision training (AMP) reduces memory by ~50%
- Dataset filtering removes outlier images with >400 boxes

11.2 Small Object Detection

Challenge: Many objects in aerial images are very small (<32 pixels).

Solutions:

- Custom anchor sizes starting from 8 pixels
- Lower RPN foreground IoU threshold (0.5 instead of 0.7)
- Multi-scale FPN with P2 level for fine details
- CBAM attention for better feature refinement

11.3 Class Imbalance

Challenge: Vehicle classes have many more instances than rare classes.

Solutions:

- Stratified sampling in RoI heads (positive_fraction=0.25)
- Balanced batch composition

11.4 Training Stability

Challenge: Loss spikes and NaN values during training.

Solutions:

- Gradient clipping (max_norm=1.0)
- NaN/Inf detection and batch skipping
- Differential learning rates (lower for RoI heads)
- AdamW optimizer with weight decay

11.5 Overfitting

Challenge: Gap between training and validation mAP.

Solutions:

- Dropout (0.3) in box and mask heads
- Data augmentation (flip, color jitter)
- Lower learning rate for RoI heads
- Early stopping based on mAP gap (optional)

12. Runtime Environment

12.1 Hardware

Component	Specification
GPU	NVIDIA A100
GPU Memory	40 GB
RAM	80 GB

12.2 Software

Component	Version
Python	3.10+
PyTorch	2.0.0+
CUDA	11.8
cuDNN	8.x

---

13. Training and Inference Time

13.1 Training Time

Configuration	Time per Epoch	Total (25 epochs)
Batch Size 8, AMP	~25 min	~11 hrs

13.2 Inference Time

Configuration	Time per Image	FPS
800×800, GPU (P100)	~38.59 ms	~25.91
800×800, CPU	~1411.26 ms	~0.71

13.3 mAP Computation Time

Samples	Time
200	~14.19 s
Full Validation	~8.57 min

---

14. Libraries and Tools

14.1 Core Dependencies

See `requirements.txt` for complete list:

```
# PyTorch (CUDA 11.8)
torch>=2.0.0
torchvision>=0.15.0

# Core dependencies
numpy>=1.24.0
pillow>=9.0.0
opencv-python>=4.7.0
pyyaml>=6.0

# Image augmentation
albumentations>=1.3.0

# Visualization
matplotlib>=3.7.0
scipy>=1.10.0

# Deep learning utilities
timm>=0.9.0
tqdm>=4.65.0

# Hyperparameter optimization
optuna>=3.0.0

# Experiment tracking
wandb>=0.24.0
```

14.2 Tools

Tool	Purpose
Weights & Biases	Experiment tracking, visualization
Optuna	Hyperparameter optimization (anchors)
Albumentations	Data augmentation
Git	Version control

---

15. Bibliography

15.1 Dataset

1. Waqas Zamir, S., Arora, A., Gupta, A., Khan, S., Sun, G., Shahbaz Khan, F., ... & Shao, L. (2019). **iSAID: A Large-scale Dataset for Instance Segmentation in Aerial Images**. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (pp. 28-37). Paper Link
2. Xia, G. S., Bai, X., Ding, J., Zhu, Z., Belongie, S., Luo, J., ... & Zhang, L. (2018). **DOTA: A Large-scale Dataset for Object Detection in Aerial Images**. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 3974-3983). Paper Link
3. Kaggle iSAID link

4. Kaggle DOTA link

15.2 Model

1. He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). **Mask R-CNN**. In Proceedings of the IEEE International Conference on Computer Vision. Paper Link
2. Tan, M., & Le, Q. (2019). **EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks**. In International Conference on Machine Learning. Paper Link
3. Woo, S., Park, J., Lee, J. Y., & Kweon, I. S. (2018). **CBAM: Convolutional Block Attention Module**. In Proceedings of the European Conference on Computer Vision. Paper Link
4. Lin, T. Y., Dollár, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017). **Feature Pyramid Networks for Object Detection**. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Paper Link

15.3 Online Resources

9. PyTorch Vision Detection Documentation: https://pytorch.org/vision/stable/models.html#object-detection
10. Weights & Biases Documentation: https://docs.wandb.ai/
11. Albumentations Documentation: https://albumentations.ai/docs/

16. Evaluation Checklist

16.1.1 Problem

#	Requirement	Points
1	Instance segmentation	3
Total		3

16.1.2 Model

#	Requirement	Points
1	Own architecture (over 50% of own layers)	2
2	Non-trivial solution - Attention	1
3	Each subsequent model with a different architecture	1
Total		4

16.1.3 Training

#	Requirement	Points
1	Hyperparameter estimation - anchors	1
2	Adaptive hyperparametrs - learning rate scheduler based on metrics	1
3	Overfitting some examples from the training set	1
4	Data augmentation	1
Total		4

16.1.4 Tools

#	Requirement	Points
1	Weights & Biases	1
2	Explanation of 3 predictions	2
Total		3

16.2 Total

#	Category	Points
1	Problem	3
2	Model	4
3	Training	4
4	Tools	3
Total		14

Appendix A: Project Structure

```
isaid-instance-segmentation/
├── config.yaml              # Main configuration file
├── train.py                 # Training script (CLI)
├── requirements.txt         # Python dependencies
├── README.md                # Quick start guide
│
├── datasets/
│   ├── __init__.py
│   └── isaid_dataset.py     # Dataset loader with filtering
│
├── models/
│   ├── __init__.py
│   ├── backbone.py          # EfficientNet + CBAM backbone
│   ├── maskrcnn_model.py    # Custom Mask R-CNN
│   └── roi_heads.py         # Custom RoI heads
│
├── training/
│   ├── __init__.py
│   ├── trainer.py           # Training loop with W&B
│   ├── transforms.py        # Data augmentation
│   ├── anchor_optimizer.py  # Optuna-based anchor optimization
│   └── wandb_logger.py      # W&B logging utilities
│
├── visualization/
│   ├── __init__.py
│   └── gradcam_pipeline.py  # GradCAM visualization
│
├── utils/
│   ├── __init__.py
│   └── overfit_test.py      # Debugging utilities
│
└── notebooks/
    ├── 00_setup.ipynb       # Environment setup
    ├── 01_training.ipynb    # Basic training notebook
    ├── 02_anchoropt.ipynb   # Anchor optimization
    ├── 03_training_wandb.ipynb # Training with W&B
    ├── 04_training_kaggle_backbone_choice.ipynb
    ├── 05_dataset_analysis.ipynb
    └── 06_gradcam_visualization.ipynb
```

Appendix B: W&B Dashboard

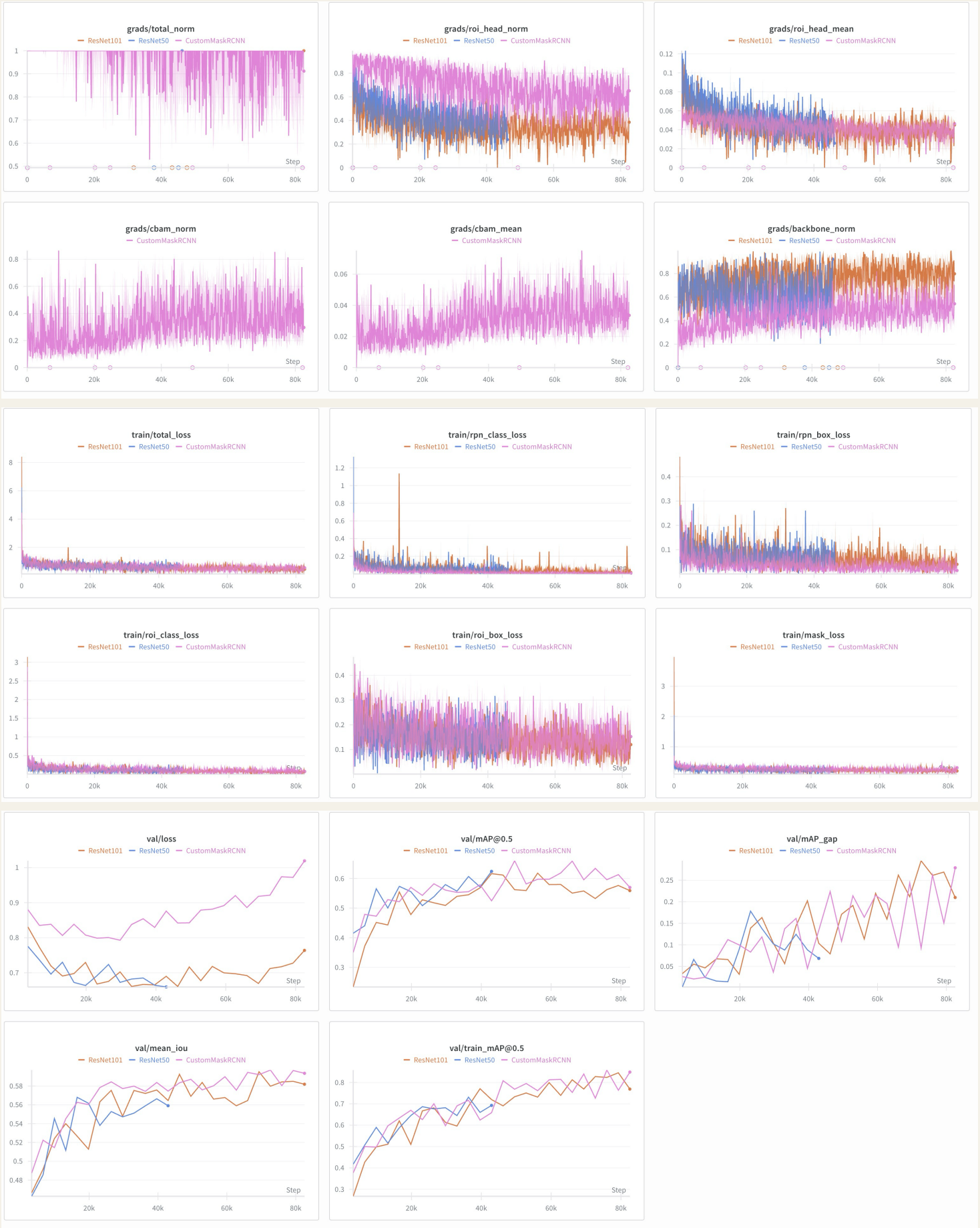
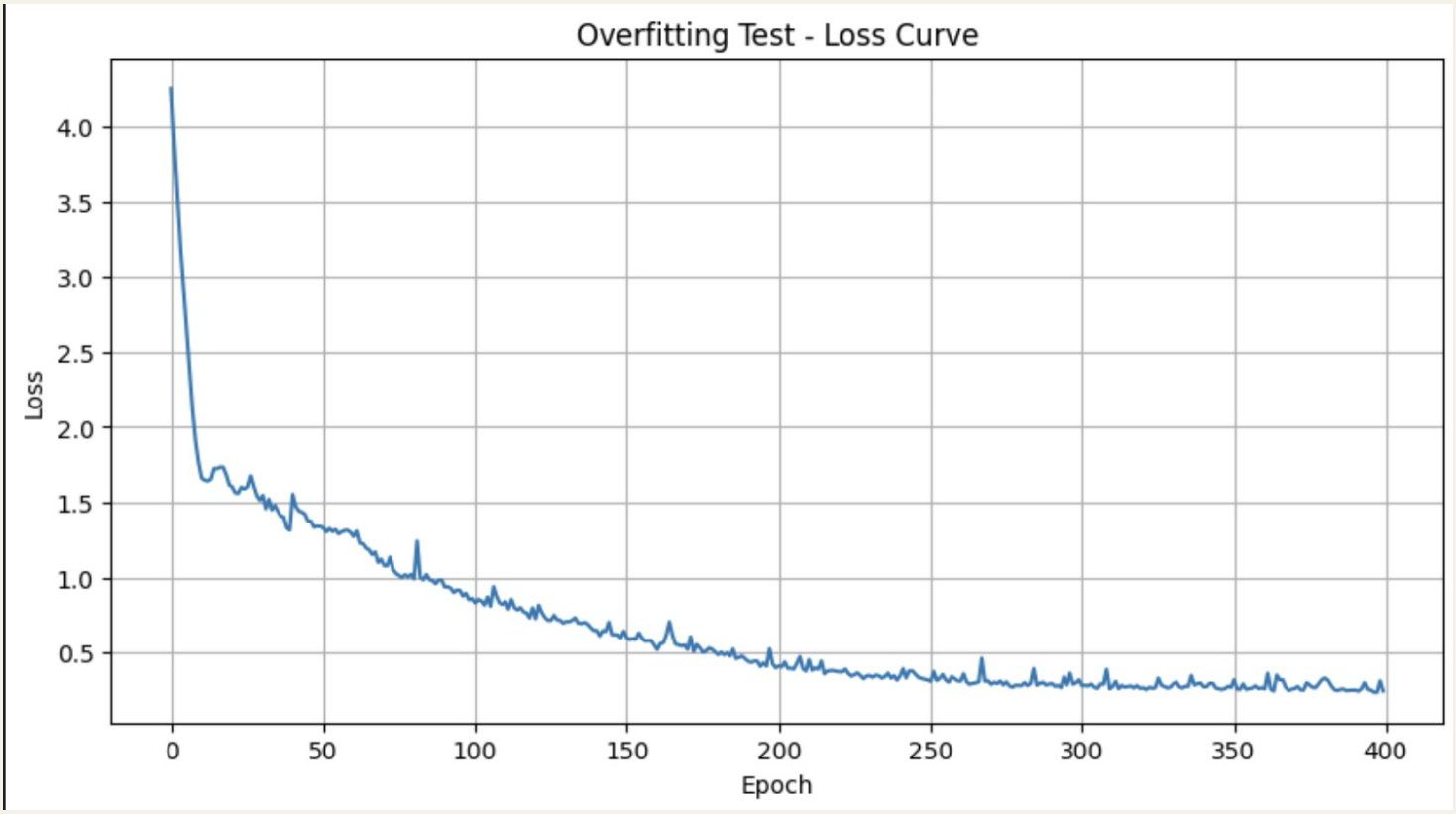
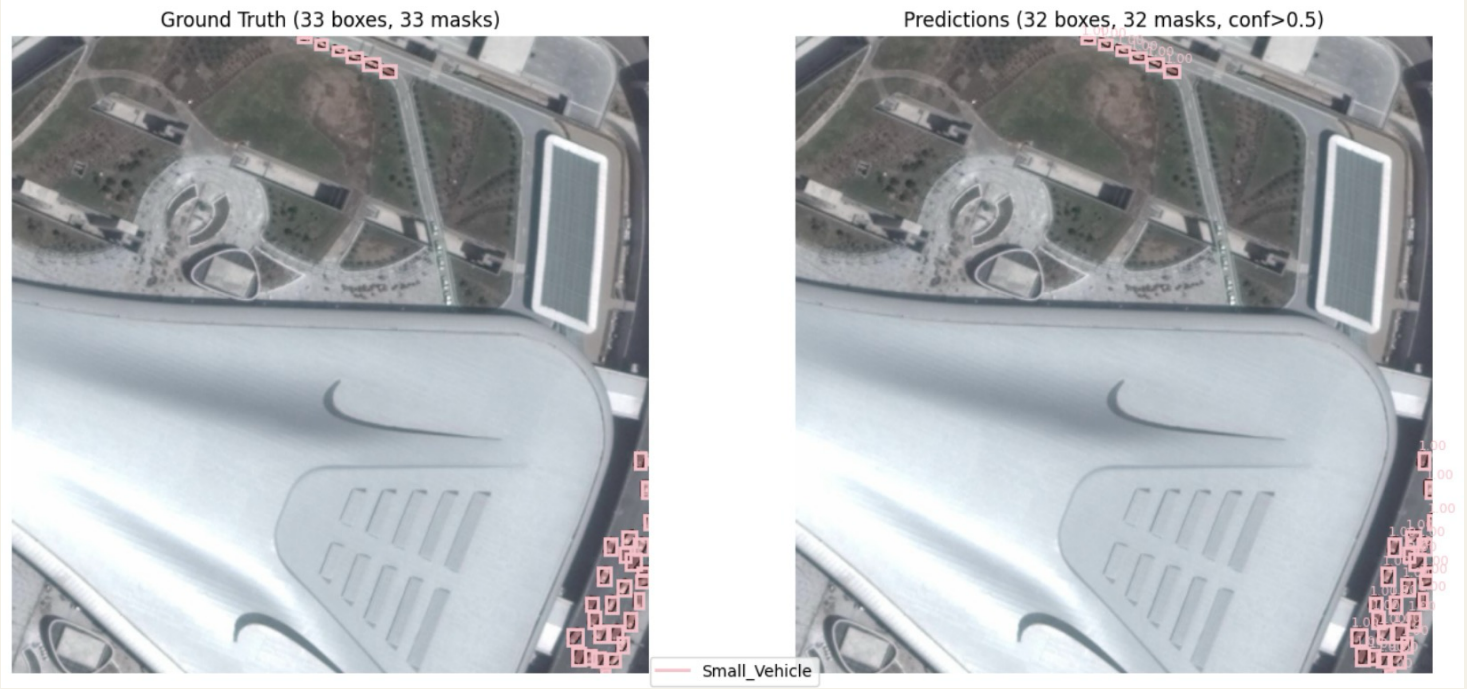


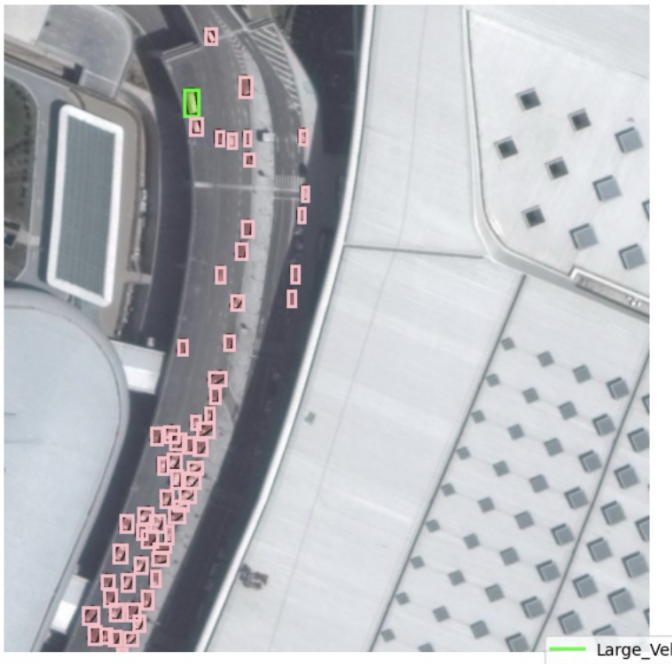
Figure B.1: Weights & Biases experiment tracking dashboard

Appendix C: Overfit test





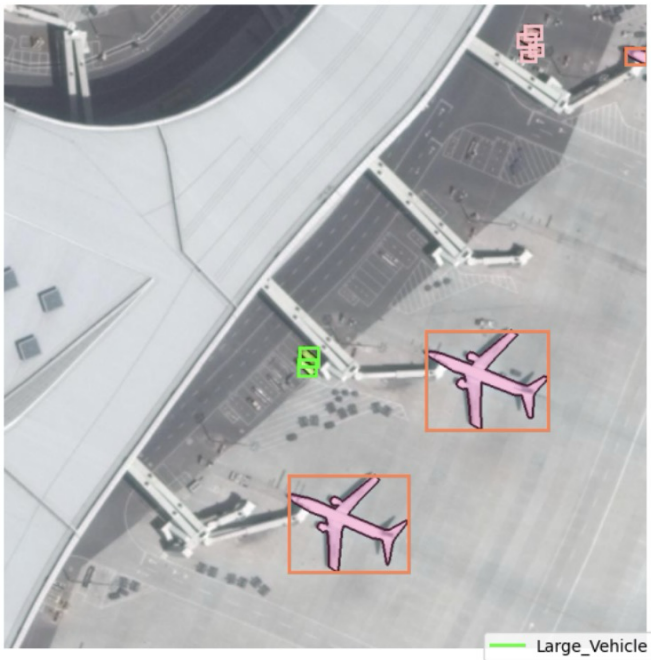
Ground Truth (62 boxes, 62 masks)



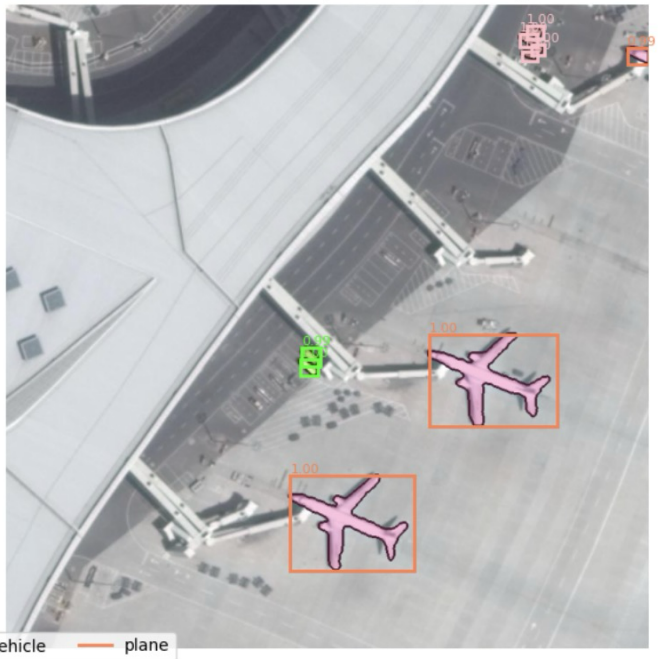
Predictions (62 boxes, 62 masks, conf>0.5)

Large_Vehicle Small_Vehicle

Ground Truth (9 boxes, 9 masks)



Predictions (9 boxes, 9 masks, conf>0.5)



Large_Vehicle Small_Vehicle plane