CVPDL HW1 Report

R14921114 紀敦翊

1 Model Description

1.1 Detector

I adopt a lightweight YOLOv11-nano–style one-stage detector with a CSP-like backbone, FPN/PAN neck, and a decoupled, anchor-free head. The head predicts class scores and box distributions (for DFL) at multiple pyramid levels (P3–P5). Non-maximum suppression (NMS) is applied at inference. Then, I disable mosaic/mixup in the last 10% of epochs (`close_mosaic=0.1`) to reduce train–val drift.

1.2 Architecture

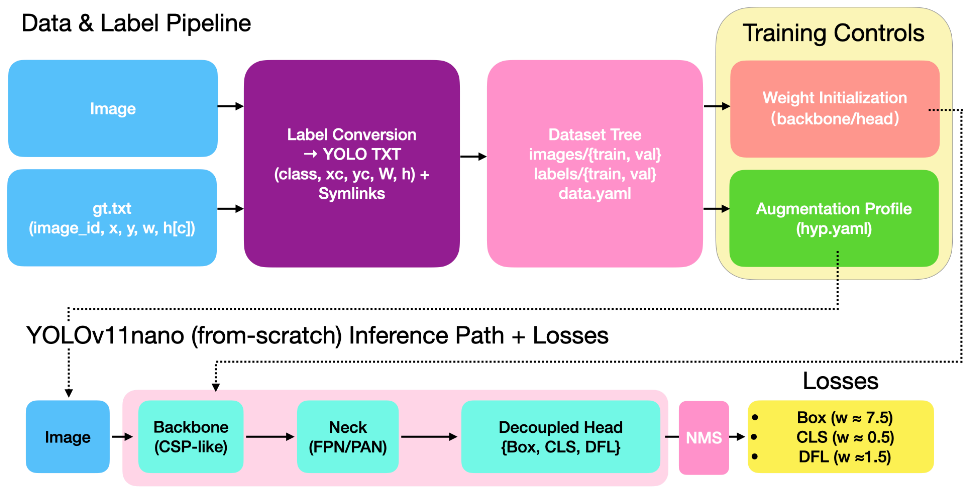

Data & Label Pipeline
Image
gt.txt (image_id, x, y, w, h[c])
Label Conversion → YOLO TXT (class, xc, yc, W, h) + Symlinks
Dataset Tree images/{train, val} labels/{train, val} data.yaml
Training Controls
Weight Initialization (backbone/head)
Augmentation Profile (hyp.yaml)
YOLOv11nano (from-scratch) Inference Path + Losses
Image
Backbone (CSP-like)
Neck (FPN/PAN)
Decoupled Head {Box, CLS, DFL}
NMS
Losses
Box (w ≈ 7.5)
CLS (w ≈ 0.5)
DFL (w ≈1.5)

Figure 1: Architecture (Drawn with Keynote)

The script first converts `gt.txt` into YOLO labels and builds a minimal dataset via **symlinks** under `images/train,val` and `labels/train,val`, writing `data.yaml` automatically. Model initialization targets the network itself: we train **YOLOv11-nano** from scratch (pretrained = False) and, if provided, optionally load backbone-only weights while keeping the decoupled detection head randomly initialized. The augmentation profile (`hyp.yaml`) is applied by the train dataloader before the backbone—mosaic/mixup, color jitter, and light geometric transforms affect both images and boxes, while val/test use raw images. The forward path is Image → Backbone (CSP-like) → Neck (FPN/PAN) → Decoupled Head Box, CLS, DFL, optimized with the standard YOLO multi-task losses as set in `hyp.yaml`.

1.3 Losses

I use the standard Ultralytics/YOLO loss family:

- Box loss: IoU-based localization loss (CIoU/GIoU depending on version).
- DFL: Distribution Focal Loss for precise box regression.
- CLS loss: BCE for class scores (single class).

The code config uses loss weights ≈ `box:7.5`, `cls:0.5`, `dfl:1.5`.

1.4 Key modifications vs. stock YOLO

- Dataset is built on-the-fly from a provided `gt.txt` (custom conversion + symlink tree).
- Optional augmentation profile (custom `hyp.yaml`) and a mosaic/mixup closing schedule in late epochs (`close_mosaic`), to stabilize final training.
- A post-training parameter tuning stage (`tune_para.py`) that grid-searches inference knobs (image size, conf-thresh, NMS IoU, max detections, TTA) on the validation split before freezing the final inference config.

2 Implementation Details

2.1 Data & Preprocessing

- **Input images.** All training images are read from `--images_dir`.
- **Ground truth format.** `gt.txt` contains per-image boxes. During conversion we generate YOLO TXT labels (one file per image) with normalized center-format:

  $$x_c = \frac{x + \frac{w}{2}}{W}, y_c = \frac{y + \frac{h}{2}}{H}, w_n = \frac{w}{W}, h_n = \frac{h}{H}$$

  where (x, y) is the top-left in pixels, w, h are box size in pixels, and W, H are image width/height.
- **Dataset layout.** The script builds a lightweight tree with symlinks:

  ```
  workdir/
      data.yaml
      images/{train,val}/
      labels/{train,val}/
  ```

  A deterministic split is created with `--val_ratio` and `--seed`.
- **Loader transforms.** Standard Ultralytics preprocessing (letterbox resize to `imgsz`, RGB, float in [0,1], CHW). Training uses multi-scale pyramids internally via the model (P3–P5).

2.2 Model & Training Config

- **Architecture.** YOLOv11-nano config (CSP-like backbone → FPN/PAN neck → decoupled head Box, CLS, DFL). Anchor-free heads regress box distributions.
- **Initialization.** Detection heads are randomly initialized; no pretrained detection weights are used. (Backbone remains non-pretrained in the submitted baseline.)
- **Epochs/batch/image size.** Script defaults: `epochs=100`, `batch=16`, `imgsz=640`.

- **Optimizer & LR schedule.** Ultralytics SGD/AdamW variants (per version); cosine decay controlled via `lr0` and `lrf`.
- **Core hyperparameters** (`hyp_text` in `train.py`).
- **Augmentation.** When `--use_aug` is set, the script writes a custom `hyp.yaml` enabling mosaic + mixup (CutMix-like), plus the default Ultralytics geometric/color jitters. We optionally disable mosaic/mixup for the last 20–30% epochs via `close_mosaic` to reduce distribution shift between train and val/inference.
- **Training controls.** AMP mixed precision, dataloader workers, reproducible seed, checkpointing to `workdir/train/weights/`.

2.3 Post-training Parameter Tuning & Inference

- **Validation-guided tuning (`tune_para.py`).** We grid-search on the validation split to maximize mAP@50. The best tuple (`imgsz, conf, iou, max_det, TTA`) is recorded and reused for test-time inference.
- **Final inference (`infer.py`).** Loads `best.pt`, applies the tuned parameters, runs NMS, and emits a CSV in the required format.The class index is fixed to 0 (single-class). The script preserves the exact row order of the provided sample CSV.

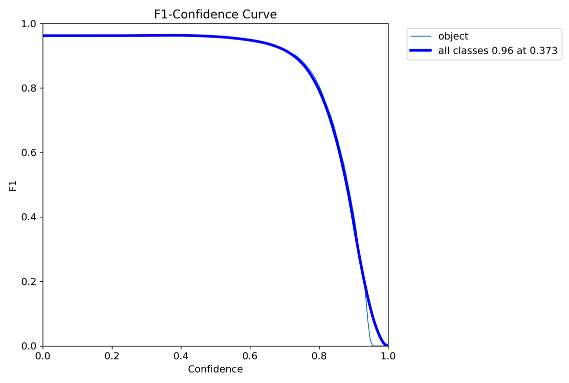
3 Result Analysis

3.1 Quantitative improvements

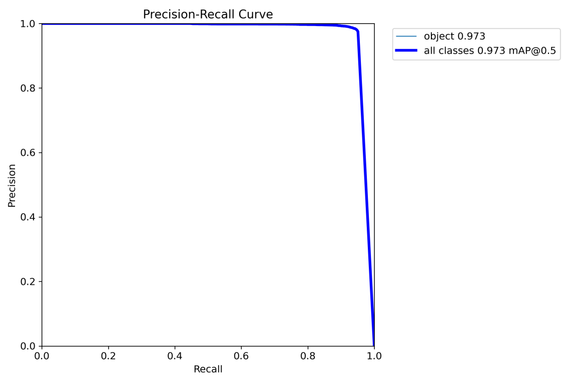
augmentation	image size	epoch	val.	mAP50 of val.
w/o	640	120	0.2	0.987
w/	640	120	0.2	0.984
w/	960	120	0.2	0.99

Table 1: Caption

3.2 Visualization



(a) BoxF1_curve



(b) BoxPR_curve

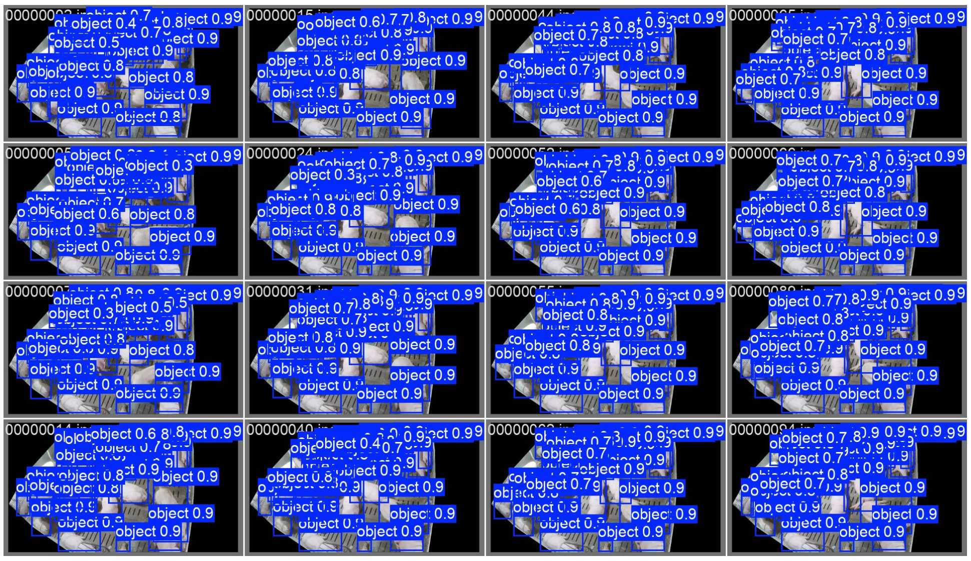


Figure 3: val_batch0_pred

- BoxPR_curve —Precision–Recall Curve The PR curve stays near the top-right corner, indicating consistently high precision across most recall levels; the area yields AP@0.5 = 0.973 (overall P 0.98, R 0.95 from the eval run). Degradation appears only as recall approaches 1.0, suggesting few false positives at practical operating points.
- BoxF1_curve —F1 vs. Confidence The curve peaks at F1 0.96 around confidence 0.37, which marks the optimal threshold range for balancing precision and recall. Increasing the threshold much beyond 0.8 sharply reduces F1 (recall drops), while lowering it far below 0.3 offers little recall gain but risks more false positives.
- val_batch0_pred —Qualitative Predictions (Validation Batch) Representative detections on the validation set under our inference settings show tight boxes and dense, consistent hits with high scores. Only minor overlaps remain in crowded regions, indicating NMS is largely effective while preserving recall. This visual outcome agrees with the strong PR/F1 results above.

4 Short conclusion

I implemented a lightweight YOLOv11-nano detector with a CSP-like backbone and a decoupled, anchor-free head. A minimal dataset pipeline converts gt.txt to YOLO labels and closes mosaic/mixup in the last 10% epochs to reduce train–val drift. A small inference-level tuning (imgsz↑, moderate conf/iou, TTA) adds few points further, aligning with the PR and F1 curves. The remaining errors concentrate on dense overlaps and very small objects.