

EVAL #2

● RETAIL ANALYSIS USING NEURAL NETWORKS

Nathan Little

Colin Kirby



Table of Contents



Project Recap



Baseline
Performance



Improvement
Techniques



Before / After
Improvements



Results
Summary



SOTA
Comparisons



Team &
Contributions



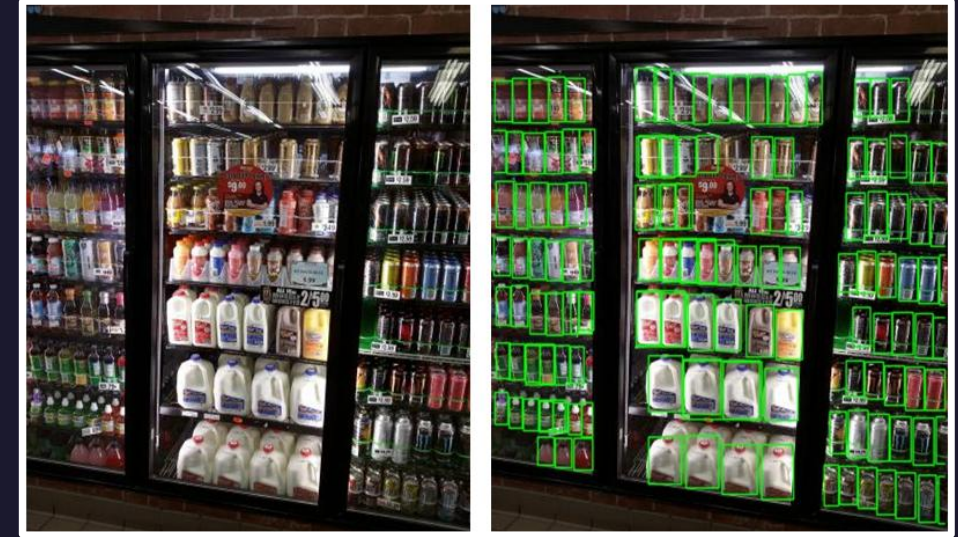
Conclusion



Project Recap – ShelfVision

Our Goal :

- Detect & Localize retail products on shelf images.
- Handle dense, cluttered scenes with overlapping objects.
- Train a custom object detection model on the SKU-110K dataset.
- Improve anchor coverage, training stability, and detection accuracy.

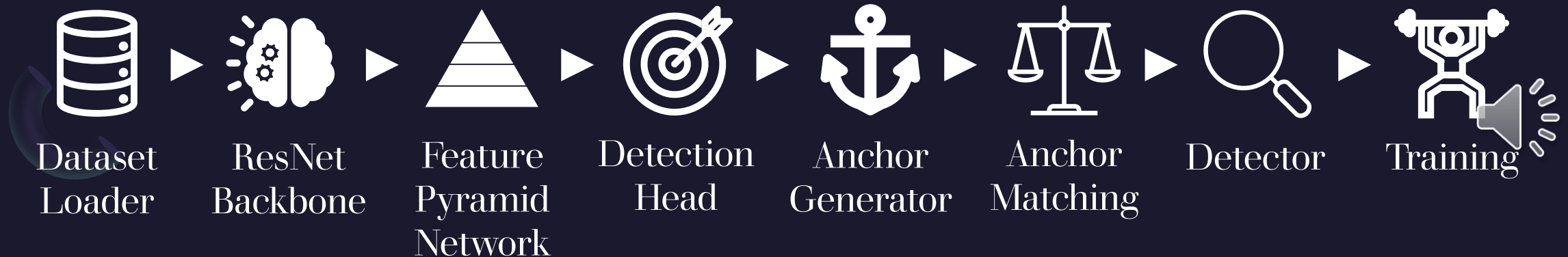


Left: Original image of a grocery shelf.

Right: Detected items with bounding boxes.

Source: "Retail Store Item Detection using YOLOv5"

Pipeline Overview (Simplified) :



Baseline Performance : *Old EVAL1*

Eval #1 – Initial Set Up

- Most metrics were zero or near-zero.
- High box clutter, low overlap → ineffective coverage.
- Resulted in meaningless Precision / Recall Plots.
- Goal back then : “Make anything show up.”

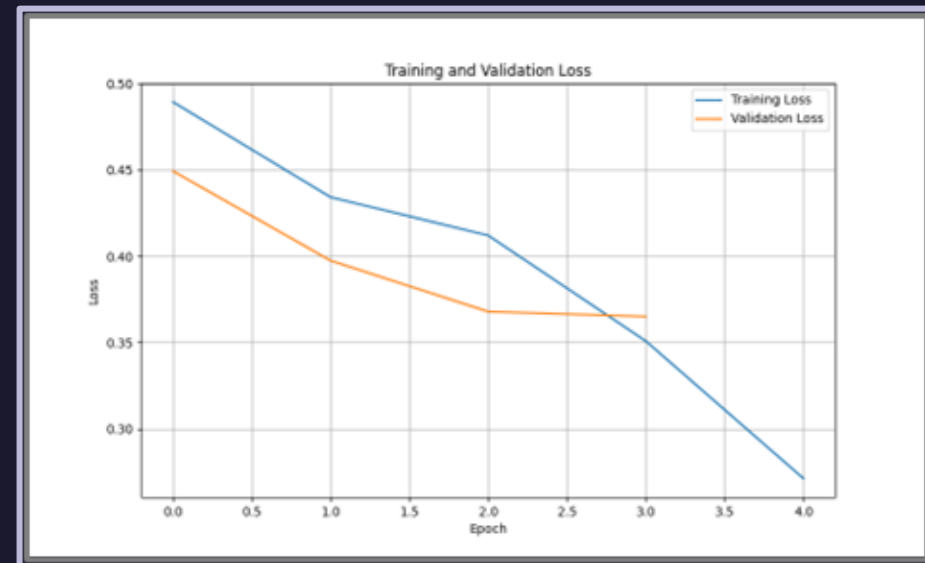


Figure: Old baseline loss — values unreliable due to noisy, random outputs.

Why We Needed a New Baseline

- Detected boxes existed, but didn't align with objects.
- We couldn't tell if any improvement was actually an improvement.
- Needed clean loss curves and IoU-based training stability.
- Set up a new baseline with better anchors + loss visibility.



Baseline Performance : *New Baseline*

New Baseline – Stable Training Run

- Improved architecture + better anchor tuning.
- Loss trends are more meaningful :
 - Classification Loss : Very Low early on
 - Box Regression Loss : Still High, but Steadily Decreasing
- Provides a solid launch point for evaluating future improvements.

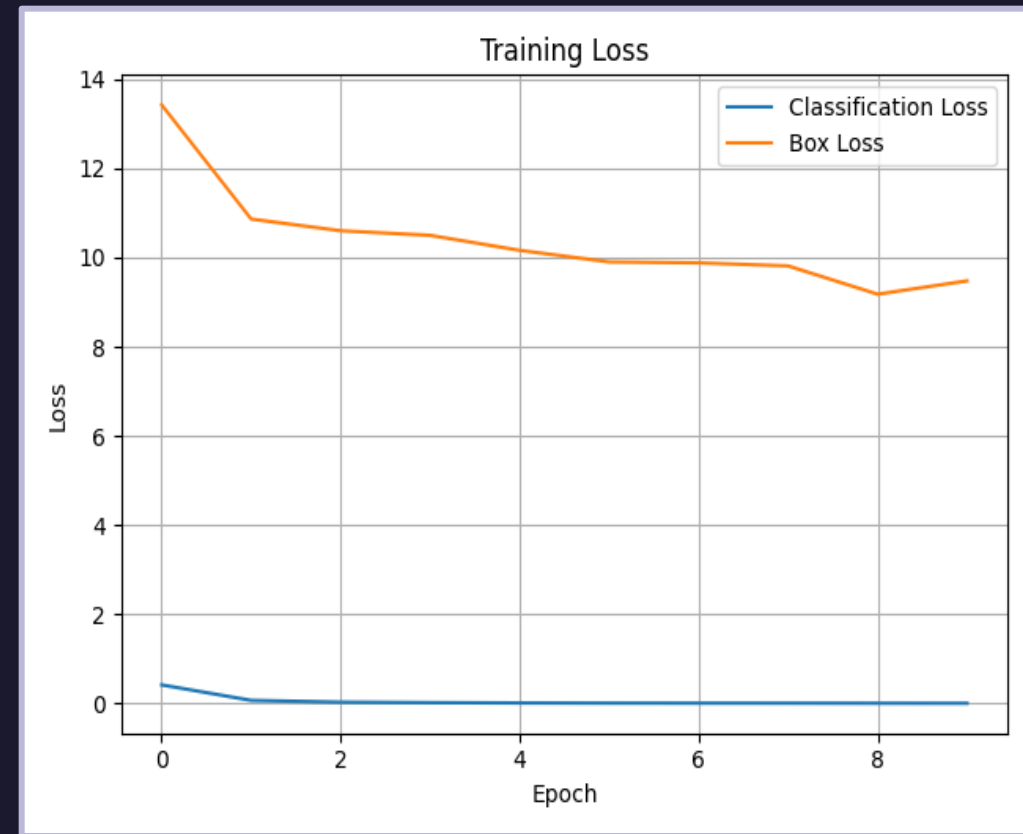
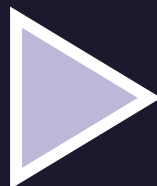


Figure: New baseline training loss. Classification loss (blue) quickly drops, while box loss (orange) decreases slowly, showing potential for better localization.

Improvement Techniques Overview

1. Anchor Tuning Fixes →

Adjusted anchor scales and placement to better match ground truth boxes.

2. Improved Matching Logic →

Rewrote anchor matching with dynamic IoU and best match assignment.

3. Box Delta Debugging →

Fixed errors in how predicted deltas were applied to anchors.

4. Memory & Batch Tuning →

Reduced batch size and added subset config to avoid crashes.

5. Loss Balancing & Schedule Optimization →

Reweight losses, extended warmup, and eased LR decay for more stable training.



1. Anchor Tuning Fixes

The Problem :

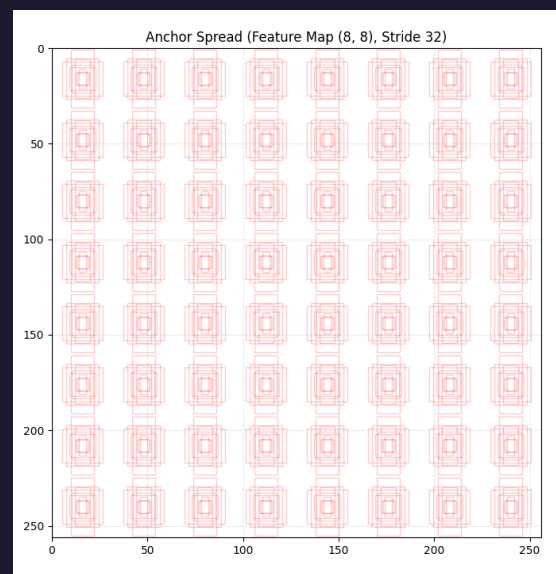
- Anchors were too small, too centered, or clipped at image edges.
- Most anchors didn't overlap with any ground truth box → poor learning.

What We Changed :

- Increased Base Sizes → better coverage across object sizes.
- Refined Scales & Aspect Ratios.
- Anchors now span wider shapes and areas, aligned with SKU-110K product layout.
- Added Stride-Aware Centering.

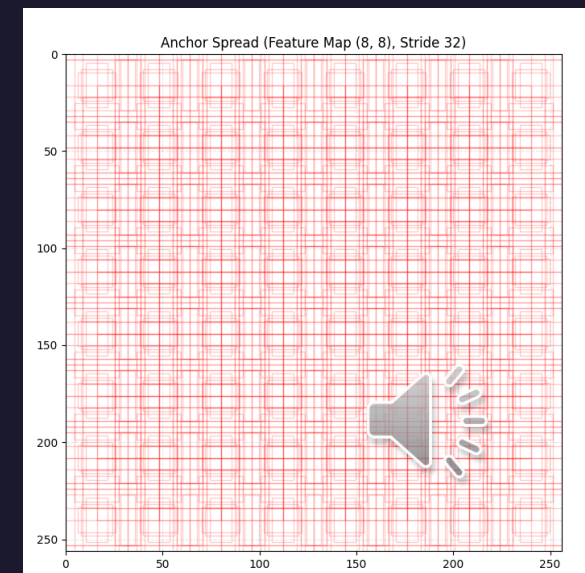
```
# Create Anchor Generator w/ Improved Configuration.  
self.anchor_generator = AnchorGenerator(  
    base_sizes=[64, 128, 256, 512],  
    scales=[0.5, 1.0, 1.5],  
    aspect_ratios=[0.5, 1.0, 2.0],  
)
```

Figure: Snippet of Anchor Generator Initialization.



**Before Adjusting
Scales & Aspect
Ratios**

**After Adjusting
Scales & Aspect
Ratios**



2. Improved Matching Logic

The Problem :

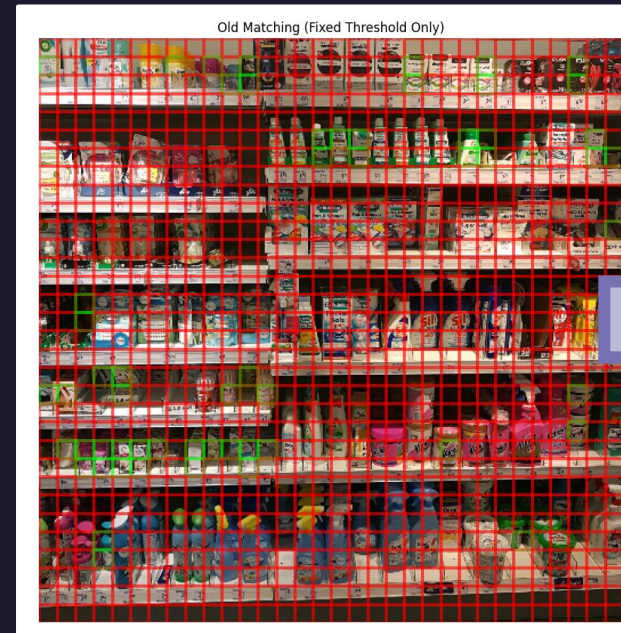
- Anchors weren't properly assigned to GT boxes.
- Many GT boxes had no positive anchor matches.
- Resulted in zero mAP / F1 even after long training runs.

```
# Match Anchors to GT Boxes.  
max_iou, max_idx = ious.max(dim=1)  
pos_mask = max_iou >= iou_threshold  
matched_labels[pos_mask] = gt_labels[max_idx[pos_mask]]  
  
# Force Assign Best Anchor to Each GT (even at Low IoU).  
gt_max_ious, gt_best_anchor_idxs = ious.max(dim=0)  
matched_labels[gt_best_anchor_idxs] = gt_labels
```

Figure: Snippet of Anchor Matching Logic..

What We Changed :

- Introduced IoU-base matching with tunable threshold.
- Forced best-match assignment per GT box → guarantees every object is represented



Before IoU-base matching.



After IoU-base matching.

3. Box Delta Debugging

Before Fix :

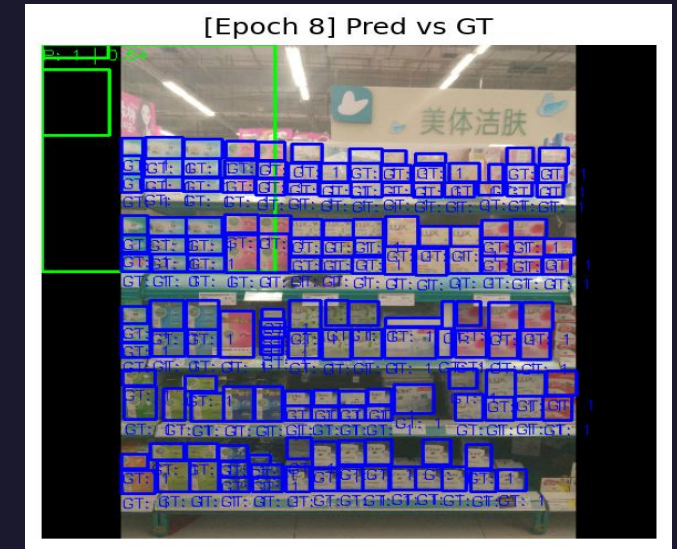
- Box Δ 's were applied without proper clamping or validation.
- Causing box predictions to explode towards image edges or shrink to near-zero size.

Fixed Made :

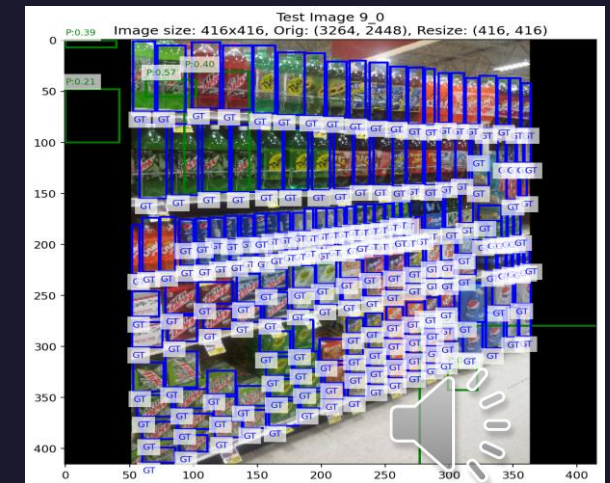
- Clamped predicted delates to safe values.
- Clipped boxes to stay within image boundaries.
- Filtered boxes with heigh/width too small or large.

Result :

- Predictions now stay within valid bounds.
- Reduced false positives from massive or corrupted box shapes.



Before : Predictions Sitting on Edges & Are Too Big.



After : Predictions stay within more applicable size range.

4. Memory & Batch Tuning

The Problem :

- Training w/ Full Dataset + Large Batch Size → CUDA Out of Memory Errors.
- Anchor tuning increased total anchor count → Higher Mem Load during Training.
- Debugging / Testing was slow due to full-data loading.

Fixes Made :

- Reduced Batch Sizes during Training in our config files.
- Utilized a subset size to enable quick training / debug runs.
- Controlled number of anchors per level via smarter scales + ARs.

Result :

- Resolved OOM Crashes.
- Enable faster iterations for testing.

```
[DEBUG] Generated 172032 anchors for level p3  
[DEBUG] Generated 43008 anchors for level p4  
[DEBUG] Generated 10752 anchors for level p5
```



```
[DEBUG] Generated 36864 anchors for level p3  
[DEBUG] Generated 9216 anchors for level p4  
[DEBUG] Generated 2304 anchors for level p5
```

Figure: Reduction in total anchors per level after tuning anchor scales and aspect ratios — significantly lowered memory usage and improved efficiency.

```
torch.OutOfMemoryError: CUDA out of memory. Tried to allocate 12.00 MiB. GPU 0 has a total capacity of 8.00 GiB of which 0 bytes is free. Of the allocated memory 14.64 GiB is allocated by PyTorch, and 93.44 MiB is reserved by PyTorch but unallocated. If reserved but unallocated memory is large try setting PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True to avoid fragmentation. See documentation for Memory Management (https://pytorch.org/docs/stable/notes/cuda.html#environment-variables)
```

Figure: Snippet of OOM Error Message..

5. Loss Balancing & Scheduling Optimization

The Problem :

- Classification & Box Losses were on very different scales, leading to imbalanced learning.
- Model would converge too fast due to aggressive learning rate drops.
- Early training was unstable – the model couldn't learn effectively right away.

```
# Add Loss Weighting.  
cls_loss = outputs["cls_loss"]  
box_loss = outputs["box_loss"]  
loss = cls_loss + 0.1 * box_loss
```

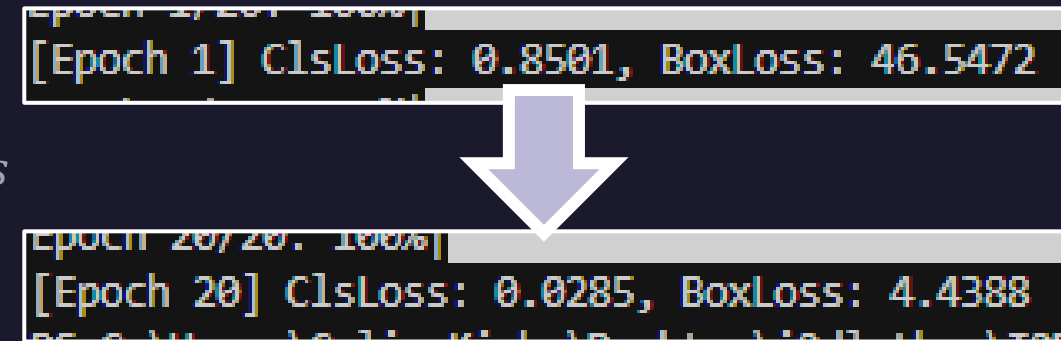
Figure : Snippet of Added Loss Weighting.

What We Changed :

- Dynamic Loss Weighting $\rightarrow loss = cls_{loss} + 0.1 * box_{loss}$
- Warmup Extended \rightarrow 3 Epochs for smoother ramp-up.
- Gentler LR Decay. (0.2 \rightarrow 0.1)

Result :

- Training now more stable, smoother convergence with fewer spikes.



Loss trends across training. The baseline model stalled around a box loss of 10 — our improved setup pushed it down substantially with better learning dynamics.

Before / After Improvements

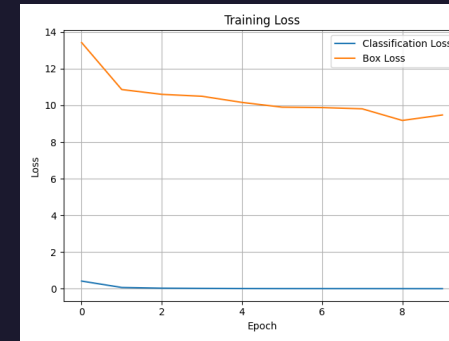
Before Improvements Training / Testing Behavior :

- Box loss : ~ 9.8 ,
- Classification loss : ~ 0.1
- Precision : 3.5%, Recall : 13.5%, mAP : 1.4%
- Although metrics seemed okay, this was misleading — the model produced 55,676 predictions for 14,439 objects, relying on quantity over quality.
- These over-predictions created false positives that happened to overlap, inflating IoU and F1 without true understanding.

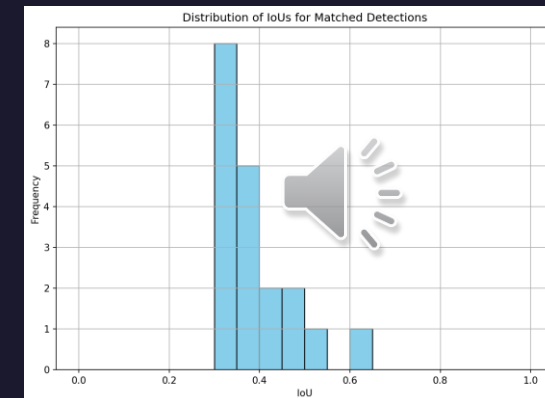
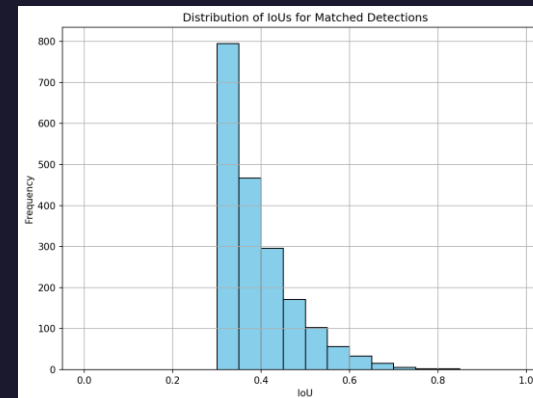
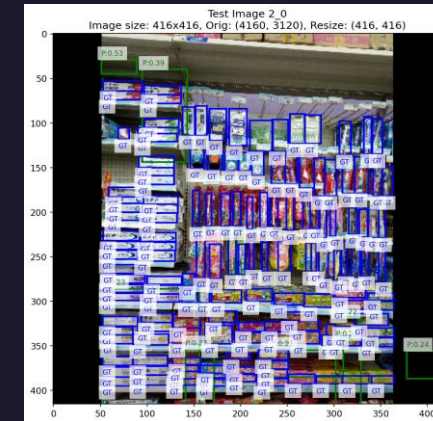
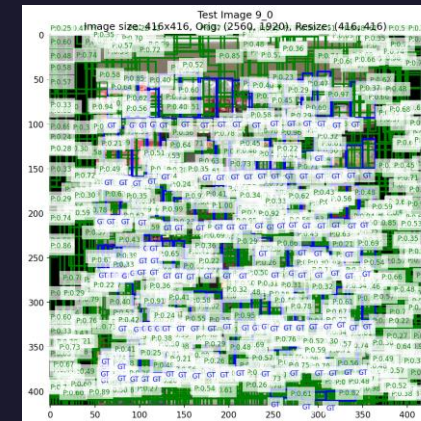
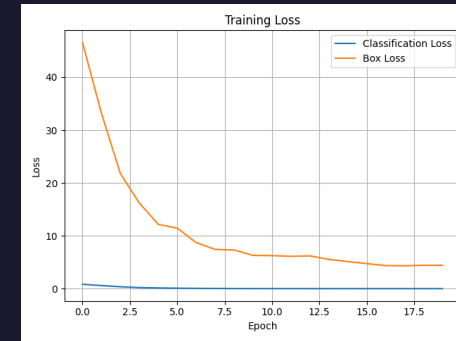
After Improvements Training / Testing Behavior :

- Box loss improved significantly (well below 5)
- Precision dropped to 2.2%, Recall to 0.1%, but this time the model only made 861 predictions.
- This reflects a more conservative, high-confidence behavior — fewer detections, but closer to meaningful learning.
- This is a trade-off during training: fewer, more confident predictions → eventually tuned to recall more.

Before Improvements



After Improvements



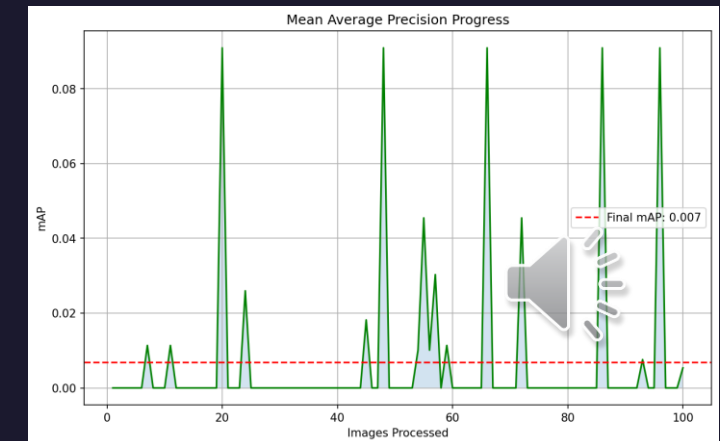
Current Model Results – ShelfVision Summary

What This Means :

- Detection is working (non-zero TP), but we're still struggling with recall and false positives.
- Bounding box alignment has improved (IoU ~ 0.38), but many predictions still fall below confidence or NMS thresholds.
- ShelfVision now generalizes across shelf layouts — major improvement from Eval 1 (empty outputs).
- Still behind fine-tuned SOTA (YOLOv5), but progress is clear.
- These results form the basis for comparing with both:
 - YOLOv5 pretrained (COCO)
 - YOLOv5 fine-tuned on SKU110K

Metric	ShelfVision	Eval I
Precision	0.0221	0.00
Recall	0.0013	0.00
F1 Score	0.0025	0.00
Avg IoU	0.3875	0.00
mAP	0.0069	0.00
True Positives	19	0.00
False Positives	842	0.00
False Negatives	14,420	0.00

Figure: mAP progression over 100 images, final mAP ≈ 0.007 with high variance.



YOLOv5 (COCO Pretrained) vs. ShelfVision

Explanation :

- The YoloV5 that we ran, lacks any exposure to retail shelf-style layouts and object density.
- As a result, almost all detections are filtered out post-NMS or do NOT overlap with GTs.
- YOLOv5 fails due to zero retail domain adaptation — ShelfVision shows early learning despite low recall.
- Despite lower confidence scores, ShelfVision shows clear improvement with meaningful IoUs and non-zero mAP — proving it's learning retail-specific patterns.

Metric	YoloV5	ShelfVision
Precision	0.00	0.0221
Recall	0.00	0.0013
F1 Score	0.00	0.0025
Avg IoU	0.00	0.3875
mAP	0.00	0.0069
True Positives	0	19
False Positives	342	842
False Negatives	14,885	14,420

Note : ShelfVision results are from early-stage training. While performance is still low, the model is beginning to learn SKU-110K-specific patterns—later epochs continue to improve mAP and reduce false positives.





YOLOv5 (Fine-Tuned) vs. ShelfVision

Explanation :

- The YOLOv5 fine-tuned greatly outperforms our ShelfVision across all metrics.
- Looking at YOLO's metrics you can tell the model is much more efficient at learning to localize and classify shelf products with strong consistency.

Why does ShelfVision underperform?

- Smaller customer architecture – Designed for faster inference & lower memory footprint.
- Fewer hyperparameter optimizations.
- Might not leverage multi-scale detection like YOLO's FPN-based head.

Metric	YoloV5	ShelfVision
Precision	0.00	0.0221
Recal	0.00	0.0013
F1 Score	0.00	0.0025
Avg IoU	0.00	0.3875
mAP	0.00	0.0069
True Positives	0	19
False Positives	342	842
False Negatives	14,885	14,420
Predictions		

Team Contributions

Task	Colin Kirby	Nathan Little
Model Architecture & Pipeline	70% – Led coding of backbone, FPN, detection head, anchors	30% – Assisted in module integration
Evaluation & Debugging	30% – Supported validation fixes, ran tests	70% – Focused on identifying root issues (e.g., 0 mAP)
Visualization & Coverage Tools	60% – Built anchor/debug visualizers	40% – Interpreted results, proposed changes
Training Tuning & Stability	50% – Adjusted loss tracking, early stopping	50% – Tuned config (batch size, subset, etc.)
Presentation Writing	40% – Designed visuals and layout	60% – Wrote improvement descriptions and analysis



Conclusion & Next Steps

- ShelfVision now produces valid detections with improved IoU (~ 0.38), a major step up from Eval 1.
- Precision and Recall are still low due to confidence filtering and bounding box quality.
- Fine-tuned YOLOv5 significantly outperforms our model, highlighting the gap in robustness and training scale.
- Despite limited metrics, the model shows meaningful progress in generalizing to real shelf layouts.

Next Steps :

- Perform deeper analysis on false positives and filtered predictions.
- Explore better **threshold tuning**, **data augmentation**, and **post-processing**.
- Begin drafting the final report — ⚠ our team has not started it yet. ⚠

