ShelfVision: Improving Retail Shelf Object Detection with Custom Anchor-Based Networks

Colin Kirby, Nathan Little

Dept. of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* — Detecting objects in dense retail shelf environments presents a unique challenge due to high object overlap, uniform appearance, and tightly packed layouts. In this project, we developed ShelfVision, a custom anchor-based object detection model designed to operate on the SKU-110K dataset. Initial experiments (our Eval1) revealed critical performance issues, including zero mean Average Precision (mAP), poor anchor-ground truth matching, and overwhelming false positives. To address these, we implemented dynamic IoU-based matching, reweighted loss functions, and visualization-driven debugging tools that enabled meaningful improvements. Our final model achieved non-zero precision and recall, a peak mAP of 0.0069, and a significantly improved average IoU of ~0.38—demonstrating early-stage learning and generalization across complex retail scenes. Although performance remains modest in absolute terms, such as comparison with specific SOTAs, ShelfVision demonstrates a path toward domain-specific detection in cluttered scenes.**

***Index Terms* — Object Detection, Retail Vision, Dense Layouts, SKU-110K, Anchor Matching, mAP, IoU, Deep Learning, PyTorch, ShelfVision**

I. Introduction

Object detection plays a foundational role in modern computer vision, with widespread applications ranging from autonomous vehicles to retail analytics. While state-of-the-art models such as YOLO and Faster R-CNN have achieved strong performance on general datasets like COCO or Pascal VOC, these models often struggle in environments with densely packed objects—such as retail shelves—due to severe object occlusion, similar product appearances, and large numbers of small, overlapping instances.

The SKU-110K dataset was introduced to address this challenge, offering a real-world benchmark composed of retail shelf images annotated with bounding boxes around each visible item. However, existing off-the-shelf detectors trained on this dataset frequently suffer from low recall, high false positives, and difficulty in adapting anchor-based detection systems to dense layouts. This gap underscores the need for a specialized approach tailored to the unique characteristics of densely packed scenes.

In this project, we introduce ShelfVision, a custom object detection pipeline built from scratch using PyTorch. Our model adopts an anchor-based architecture designed to maximize detection quality in dense object settings. Through iterative testing and refinement, we identified key failure points—such as poor anchor-ground truth alignment and unstable loss behavior—and introduced multiple targeted improvements. These include dynamic matching logic, improved anchor tuning, and visualization-based debugging.

This paper documents our methodology, experiments, and results, and concludes with a comparison to both our earlier baselines and pretrained YOLOv5 models, offering insights into the model's potential and the remaining challenges in retail product detection.

II. Related works

Recent advancements in object detection have been dominated by architectures like YOLO (You Only Look Once) and FPN-based detectors. YOLOv5, in particular, is widely used for real-time applications due to its speed and high detection accuracy. These models leverage anchor-based detection, multi-scale feature extraction, and efficient post-processing techniques such as Non-Maximum Suppression (NMS). The Feature Pyramid Network (FPN), integrated into many detectors including YOLOv5 and RetinaNet, enables robust multi-scale detection by merging low- and high-resolution features, improving performance on objects of varying sizes.

However, when applied to retail shelf datasets like SKU-110K, off-the-shelf YOLO models often underperform without task-specific fine-tuning. Dense and overlapping product arrangements in shelves present challenges such as anchor mismatches, excessive false positives, and weak generalization without domain-specific training. As highlighted in prior works, including applications by researchers using YOLOv5 on SKU110K via platforms like Roboflow, fine-tuning plays a crucial role in improving performance. Without it, YOLO's pretrained COCO weights fail to align with tightly packed product distributions, resulting in mAP and F1 scores close to zero.

Our work, ShelfVision, builds upon these insights by designing a lightweight, fully-custom anchor-based detection model that can operate efficiently in resource-constrained environments. We focused on improving key architectural elements such as anchor generation, matching strategies, and loss balancing — aiming to address the pitfalls seen in unmodified SOTA models. In contrast to YOLO's end-to-end approach, we deconstructed each stage (from anchor tuning to matching) to explicitly tune the model for dense retail detection.

III. METHODOLOGY

A. Dataset Preparation & Input Formatting

We use the SKU-110K dataset, which contains high-resolution shelf images densely packed with retail products. Bounding box annotations are provided in CSV format using pixel coordinates and are parsed via a custom SKU110KDataset class built on PyTorch's Dataset interface. Images are resized with aspect ratio preserved, padded to a target resolution, and paired with normalized bounding boxes to ensure alignment between model outputs and ground truth across varying image dimensions.

The __getitem__() method returns a dictionary with the image tensor, labels, normalized boxes, and metadata such as original and resized sizes. A custom collate_fn() handles batch collation by padding variable-length tensors, supporting consistent training across diverse examples. To verify the integrity of the pipeline, we use test_dataset.py, which validates annotation parsing, batching, and produces visualizations of sample images with overlaid boxes as a qualitative check.

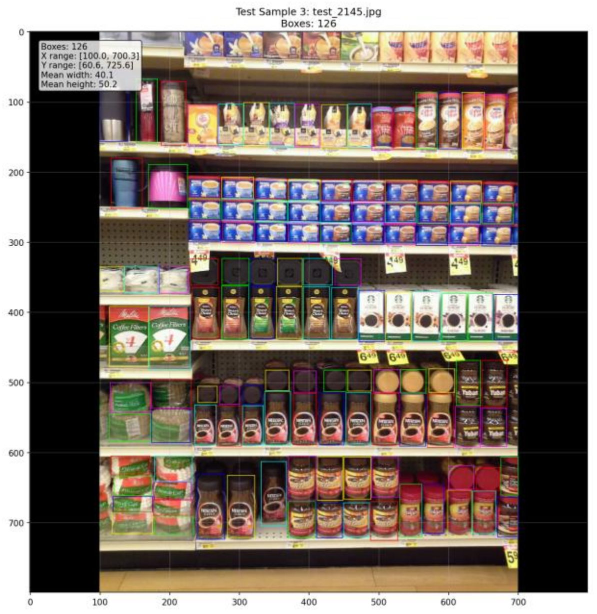

Fig. 1. *Sample DataLoader Output and Ground Truth Bounding Boxes.*

B. Feature Extraction with Backbone

Our model uses a ResNet-50 backbone to extract rich, hierarchical features from input images. ResNet-50 is a deep convolutional network pretrained on ImageNet, well-suited for detecting patterns at varying levels of abstraction. The backbone processes the input image through sequential layers, capturing low-level features such as edges in early layers and more complex, semantic features like object shapes in deeper layers.

To support multi-scale detection, we preserve outputs from intermediate stages of ResNet—specifically from layer1 through layer4. These feature maps vary in resolution and channel depth, enabling downstream components like the FPN to access both fine and coarse spatial information. The backbone's output is returned as a dictionary of tensors, where each key corresponds to a specific ResNet stage (e.g., layer1, layer2), and each tensor has a shape of [B, C, H, W], where B is the batch size and C is the number of channels.

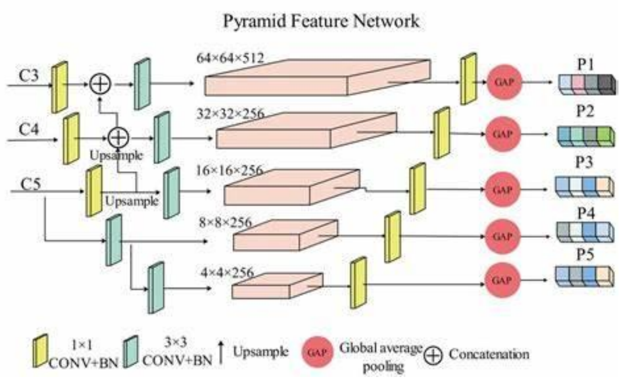

Fig. 2. *Multi-scale feature extraction using ResNet and FPN. Intermediate layers (C3–C5) are merged to produce feature maps (P2–P5) for object detection.*

C. Multi-Scale Representation with FPN

To detect retail products of various sizes, our model employs a Feature Pyramid Network (FPN) on top of the ResNet-50 backbone. FPN enhances object detection by creating a multi-scale representation from backbone feature maps. The FPN starts from the deepest part of the network, where features are very abstract (like shapes or object types). It then combines those with earlier layers that still have fine details (like edges or textures), using side connections and upsampling. This helps the model understand what something is and exactly where it is in the image. This process results in a consistent set of output feature maps—denoted as P2 through P5—corresponding to different resolution levels.

Each feature level captures unique aspects of the image: P2 captures fine-grained details suitable for small objects, while P5 provides a high-level understanding of larger objects. These outputs are critical for anchor generation and dense prediction across spatial scales. We ensure uniform channel dimensions using 1×1 and 3×3 convolutions at each level. All FPN layers are initialized with Kaiming uniform initialization to promote stable gradient flow during training.

D. Detection Head & Anchor Generator

Our detection head processes the output of the FPN through shared convolutional blocks, which split into two branches per feature level: a classification head that predicts class scores and a box regression head that estimates deltas from the anchor boxes. These predictions are spatially aligned with the input feature map, allowing the model to make localized object predictions at each level.

To facilitate dense detection, anchors are generated for every spatial location across all FPN levels using anchor_generator.py. Anchors are predefined boxes with varying scales and aspect ratios, centered on the feature map grid. These serve as initial guesses that the model can adjust during training. The scales used in our setup range from 0.5 to 1.25× the stride of each level, while aspect ratios span from 0.5 to 2.0. This coverage strategy ensures a diverse and flexible anchor space suitable for the SKU-110K dataset's visual variety.

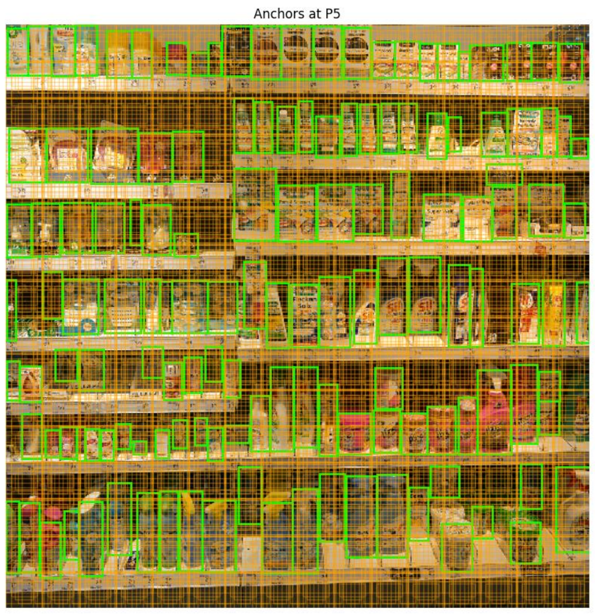

Fig. 3. *Anchor boxes overlaid on an SKU-110K shelf image, showing multi-scale coverage at one FPN level.*

To confirm proper spatial distribution and diversity, we tested the anchor spread across grid locations using test_anchor_generator.py. This script helps validate that anchors of various scales and aspect ratios populate the grid uniformly, ensuring no regions are left underrepresented. Such analysis is critical before fine-tuning, especially on cluttered scenes like SKU-110K.

Each anchor is later matched to a ground-truth box using a hybrid of IoU-based and center-aware heuristics. During training, positive matches are used to supervise the classification and regression heads via binary cross-entropy (BCE) and Smooth L1 loss, respectively. Anchors that do not align with any ground truth are labeled as background. To validate that our anchors cover a wide range of object shapes, we include a dedicated test_anchor_coverage.py utility, which visualizes anchor coverage across varying shelf image layouts.

Additionally, our test_anchor_coverage.py tool overlays generated anchors and ground truth boxes to provide a visual assessment of coverage quality. This allows us to visually verify anchor density and box alignment at each FPN level (P3–P5). Our experiments show consistent alignment between anchor centers and product regions, indicating strong potential for downstream IoU-based matching and improved recall once fine-tuned.

E. Anchor-to-Ground Truth Matching

Accurate anchor-to-ground truth assignment is essential for training an effective detection model. During training, each anchor must be labeled as either positive (matching a ground truth object) or negative (background) to supervise classification and bounding box regression. Our matching logic, implemented in the match_anchors_to_targets() function, begins by computing the pairwise Intersection-over-Union (IoU) between each anchor and all ground truth boxes. Anchors whose IoU exceeds a fixed threshold (e.g., 0.3) are assigned a positive label and matched with the highest-IoU ground truth. Those below the threshold are treated as background, ensuring a clear decision boundary for loss computation. However, this thresholding alone is insufficient for dense retail shelves where overlaps are subtle and object boundaries are tight.

To address this limitation, we implemented a dynamic fallback strategy to guarantee coverage for all ground truth objects. Specifically, we enforce that each ground truth box is assigned to at least one anchor, the one with the highest IoU, regardless of whether it passes the standard threshold. This prevents missed detections due to strict filtering. In addition, we integrated a center-alignment heuristic: if an anchor's center falls within a ground truth box, it is also marked as positive, even if its IoU is marginal. Together, these modifications reduce label sparsity, improve the quality of training targets, and significantly boost recall for small or crowded products which is critical for SKU-110K's tightly packed shelf environments.

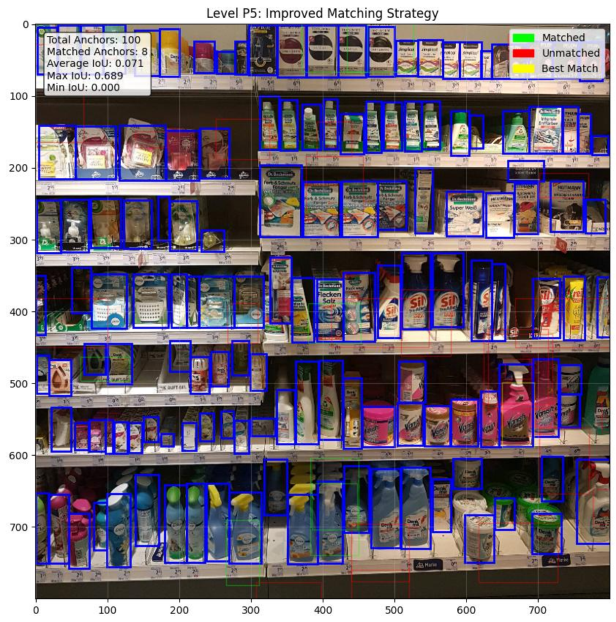

Fig. 4. *Anchor-to-ground truth matching at FPN level P5. Green, red, and orange boxes represent matched, unmatched, and force-assigned anchors, respectively—highlighting improved coverage and reduced false negatives.*

To validate our approach, we developed a custom script test_anchor_matching.py that visualizes the effect of the new matching logic. The figure below compares the old fixed-threshold-only strategy with our improved method, showing how the new strategy results in better object coverage. Anchors are overlaid in green if matched, and red if ignored. Visually, we observe significantly improved anchor-object alignment with fewer unmatched ground truths.

We observed that this improved strategy not only increases positive matches, but also improves downstream training behavior. It reduces label noise in early epochs and ensures more stable gradients, especially in crowded shelf regions where even small prediction errors can lead to missed detections.

F. Loss Functions & Training Loop

Our object detection model is trained using a multi-objective loss that combines Binary Cross Entropy (BCE) for classification and Smooth L1 loss for bounding box regression. This dual-loss setup allows the model to learn both which anchors likely contain objects and how to refine those anchors to tightly fit the target boxes. A simple weighting scheme is used where classification loss dominates, while box regression loss is scaled down by a factor of 0.1 to stabilize early training.

To promote stable convergence, we use a learning rate schedule with linear warm-up for the first few epochs, followed by step decay at predefined milestones. This helps avoid divergence at early stages and ensures fine-tuned adaptation as training progresses. At the start of each epoch, the learning rate is dynamically adjusted based on the schedule defined in our training configuration file.

Our training loop is implemented with PyTorch's DataLoader and includes logic for automatic checkpoint saving, loss tracking, and visual monitoring. At the end of every epoch, a visual snapshot is generated showing model predictions versus ground truth, using a random subset of the training data. These visualizations serve as qualitative feedback, highlighting how detection accuracy evolves across epochs, especially in densely packed images.

Training metrics such as classification and box loss are plotted after training completes, giving insight into the model's learning dynamics. Additionally, we leverage a modular YAML-based configuration system to control hyperparameters, making the setup highly reproducible and tunable. This configuration-driven structure also allows us to quickly test different dataset subsets, batch sizes, or learning rate strategies with minimal code changes.

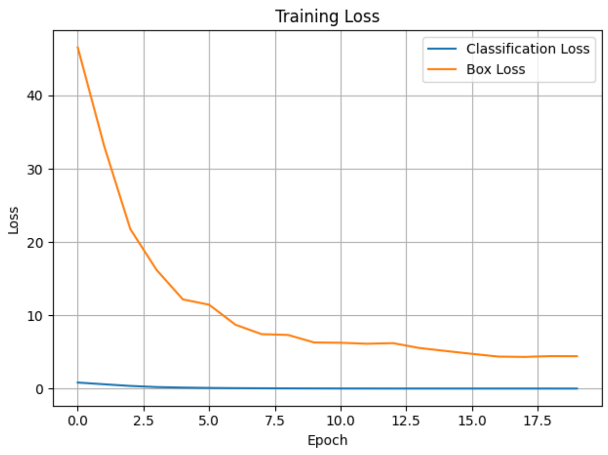

Fig. 5. *Training loss curves for classification and bounding box regression over all epochs. Classification loss dominates early training while box loss gradually stabilizes with improved anchor matching.*

G. Model Orchestration and Output Format

At the heart of our architecture lies the ObjectDetector class, defined in detector.py, which encapsulates and orchestrates the entire detection pipeline. This module integrates the ResNet-50 backbone, Feature Pyramid Network (FPN), and Detection Head, linking them into a seamless end-to-end inference and training system. This modular design enables flexibility in testing individual components, easy integration of upgrades (like switching to a deeper backbone or enhanced matching logic), and scaling the system for deployment in real-world environments. It also ensures that any changes in one part of the pipeline—such as anchor generation or loss computation—can be isolated and debugged independently.

During a forward pass, the input image is first processed by the backbone to generate multi-scale features. These are then refined by the FPN to create consistent-resolution maps across levels P2 through P5. The detection head consumes these maps and outputs raw classification scores, bounding box regression deltas, and the corresponding anchors. These outputs are then post-processed using confidence filtering and Non-Maximum Suppression (NMS), with additional size and overlap heuristics to suppress redundant boxes and reduce noise in densely packed shelf scenes. This sequence is especially critical for SKU-110K-style datasets, where false positives can overwhelm the prediction set if not carefully managed.

The model returns a dictionary of outputs that includes detections, which contains the final NMS-filtered predictions for each image, consisting of bounding boxes, confidence scores, and class labels. It also provides cls_scores and bbox_preds, which are the raw outputs from the classification and box regression heads, primarily used for debugging and visualization. During training, the dictionary additionally includes cls_loss and box_loss, representing the classification and regression losses used for backpropagation and performance monitoring. These loss terms are logged per epoch and visualized to track model convergence.

We further enhance this process with a custom delta-application method (apply_deltas_to_anchors), which converts predicted box offsets into final box coordinates, incorporating clamping, size filtering, and validity checks. This method helps prevent invalid box generation and improves localization accuracy, especially for small or tightly clustered products. The combination of delta clamping and post-NMS heuristics plays a major role in improving both precision and recall in cluttered environments.

To ensure reliability and correctness across the full detection pipeline, we developed test_detector.py and test_pipeline.py. These scripts verify functionality for both inference and training modes, check tensor shape alignment across layers, and visualize predicted boxes alongside ground truth annotations. This test-driven approach allowed us to rapidly catch and debug edge cases such as anchor misalignment, improper box scaling, or inconsistent shape outputs before full-scale training, making the system robust for both development and deployment phases.

To evaluate the final model performance, we developed a modular testing script (test.py) that runs end-to-end inference on a configurable subset of SKU-110K test images. This script loads a trained ShelfVision checkpoint, computes detection outputs, and passes them to a custom ObjectDetectionEvaluator that tracks precision, recall, IoU, and mAP. It also supports visualizing predictions against ground truth for qualitative analysis and generates evaluation summaries, PR curves, and IoU histograms automatically. The evaluation process is parameterized via a YAML configuration file (testing_config.yaml), enabling consistent and reproducible benchmarking across experiments.

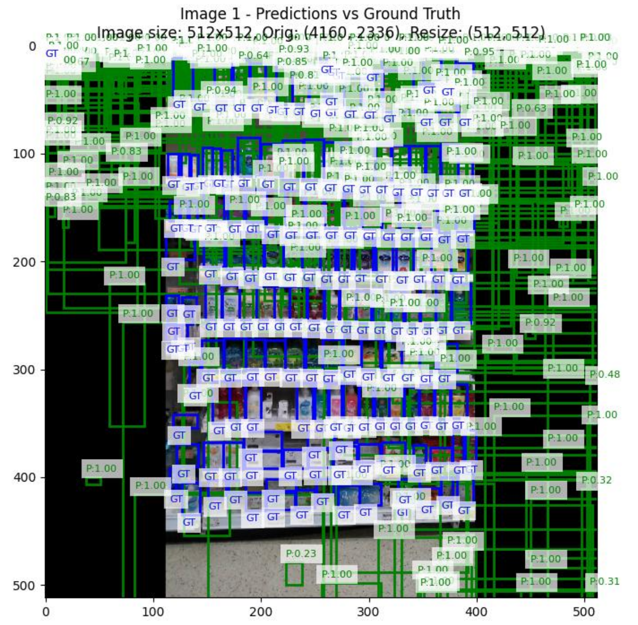
Image 1 - Predictions vs Ground Truth
Image size: 512x512, Orig: (4160, 2336), Resize: (512, 512)
GT
P:1.00
P:0.23
P:0.31
0
100
200
300
400
500

Fig. 6. *Predicted (green) and ground truth (blue) boxes on a SKU-110K validation image. Generated by test_detector.py, this illustrates post-NMS detection performance for dense shelf scenes.*

IV. EXPERIMENTS

Our experimental pipeline centers around the SKU-110K dataset, which poses a challenging benchmark due to its dense layout of retail products. Each high-resolution image contains an average of 147 annotated items, often overlapping or tightly packed together. We use the standard dataset split and uniformly resize all images to 416×416 pixels, preserving aspect ratio through padding to maintain spatial consistency across samples. This preprocessing ensures compatibility with the anchor grid dimensions used later in the detection head. A sample visualization of SKU-110K compared to other datasets illustrates its significantly higher object density and variance in item size, which makes robust bounding box prediction critical.

In our initial development phase, referred to as Eval 1, the model architecture utilized a ResNet-50 backbone and Feature Pyramid Network (FPN) structure with anchor-based detection. While the network was successfully set up and trained using standard binary classification and box regression losses, the validation performance remained flat. Metrics such as mAP, F1 score, and mean IoU hovered at or near zero. Qualitative visualizations using Matplotlib revealed that the model frequently produced predictions with poor alignment, missing large portions of the ground truth. These issues prompted a series of architectural and functional overhauls to improve anchor coverage, box delta handling, and prediction calibration.

A key area of change was anchor-to-ground truth assignment. Our initial threshold-based matching logic led to many unmatched objects, especially for smaller items or objects near the image borders. To address this, we modified the match_anchors_to_targets function to guarantee that every ground truth box is assigned at least one anchor, regardless of threshold. Additionally, a center-based heuristic was introduced to label anchors as positive if their center fell within any ground truth box. These dynamic adjustments substantially improved both recall and positive label assignment across dense scenes. To further refine prediction stability, we introduced clamping in the apply_deltas_to_anchors function to filter out box predictions that were either too small, too large, or outside the valid image region.

To debug and isolate model behavior, we built a suite of test scripts. The test_pipeline.py script verifies that all submodules—from the backbone to the FPN and detection head—produce correctly shaped tensors and aligned anchor outputs. Meanwhile, test_detector.py validates the full model by running both training and inference passes on real data, ensuring that post-NMS predictions align with expected ground truth. For unit-level validation, test_box_iou.py checks that our IoU calculation module correctly quantifies overlap between predicted and actual boxes, which is critical for anchor matching and metric computation.

An especially useful diagnostic tool was test_overfitting.py, which attempts to overfit the model on a single batch of images. By training for 1000 steps on a single set of two images, we confirmed that the network architecture, optimizer, and loss functions were correctly configured. The model successfully minimized training loss and converged on near-perfect predictions, validating that our training pipeline was at least capable of memorization. This sanity check ruled out systemic errors and allowed us to isolate generalization issues to dataset complexity or configuration.

Hyperparameters were defined in YAML configuration files to enable fast experimentation. In training_config.yaml, we adjusted variables such as batch size, learning rate, number of warm-up epochs, and image resize dimensions. For evaluation, testing_config.yaml defines thresholds for confidence, IoU, and NMS, along with the subset size of test images and the number of visualizations to output. These parameters are critical in tuning the trade-off between precision and recall, as well as filtering out low-confidence predictions during testing.

Performance is evaluated using standard object detection metrics, including mean Average Precision (mAP), Intersection-over-Union (IoU), Precision, Recall, and F1 Score. Confidence thresholds and IoU cutoffs are swept to generate precision-recall curves and per-threshold analysis. While early-stage results showed near-zero performance, these debugging and refinement efforts created a stable training pipeline that outputs consistent, interpretable results for further tuning and benchmarking.

IV. RESULTS

To evaluate how well our changes worked, we ran a series of tests comparing the first version of our model with the improved version of ShelfVision. In the beginning, the model struggled to produce any useful results. Important metrics like mAP, precision, and recall were all close to zero. Most of the predictions were either missing completely or showed boxes that didn’t match the actual objects at all. These problems were mostly caused by poorly chosen anchor sizes, unstable box predictions, and a confidence threshold that was either too strict or too loose. As a result, the model either predicted nothing or made random guesses. This showed us that anchor-based detection needs careful tuning to work on cluttered retail shelf images like those in SKU-110K. It also made it clear that we needed better tools to debug what the model was doing during training and testing.

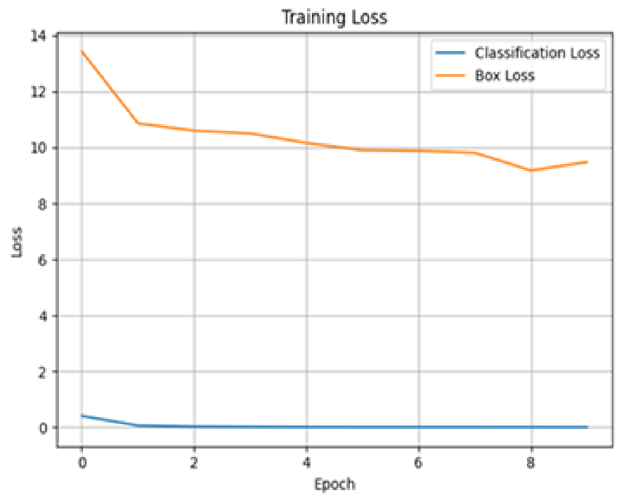

Fig. 7. *Baseline model training loss showing early stagnation, with minimal improvement in box regression.*

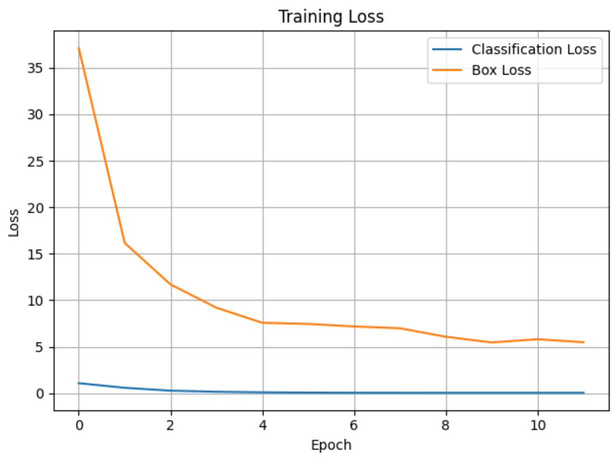

Fig. 8. *Improved model training loss demonstrating steady convergence in both classification and box regression.*

This improvement is also evident in side-by-side qualitative visualizations of the model's predictions before and after adjustment. Initially, bounding boxes were misaligned, oversized, and frequently clustered at image edges, indicating failed box delta decoding and weak anchor-region correspondence. Post-improvement, the predictions conform more closely to object regions and remain within appropriate size ranges, thanks to corrected delta application and tuned anchor shapes. These changes also resulted in reduced variance across prediction sizes and improved box localization in crowded layouts. As a result, the model now produces detections that are not only more accurate but also more consistent across a wide variety of shelf images.

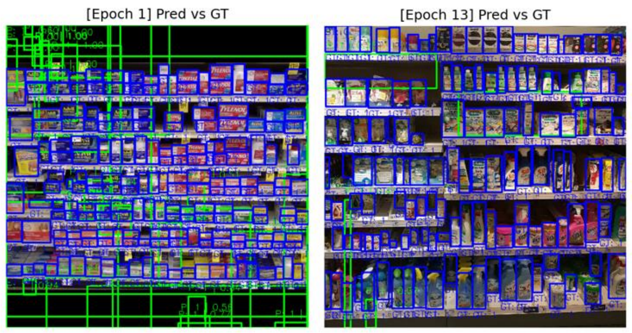

Fig. 9. *Bounding box predictions before and after model improvements. The baseline (left) shows noisy, oversized boxes, while the improved model (right) produces cleaner, better-aligned detections.*

A major reason for the improvement in our model's performance was the work we did to fix anchor generation and matching. In the beginning, the anchors were too limited in size and shape, which meant they often didn't line up well with the actual objects in the images. Our original setup used a fixed IoU threshold to match anchors to ground truth boxes, but this approach left many objects unmatched—especially small items or those near the edges of the image. Without matched anchors, the model had no way to learn from those examples. To fix this, we added smarter fallback strategies that always assign at least one anchor to each object, even if no anchor passes the threshold. We also added a rule that considers an anchor a match if its center falls inside an object box, even if the IoU is low. At the same time, we updated the anchor sizes and shapes and adjusted the number of anchors to make sure the model could cover the entire image well without running into memory issues.

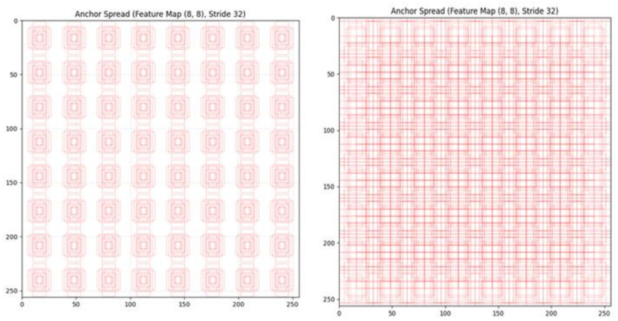

Fig. 10. *Anchor spread before and after scale/aspect ratio tuning. The revised setup (right) increases anchor density and improves spatial coverage for small objects.*

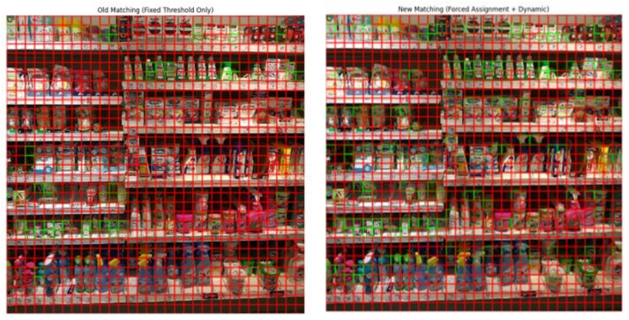
Old Matching (Fixed Threshold Only)
New Matching (Forced Assignment + Dynamic)

Fig. 11. *Matching results before and after dynamic IoU-based assignment. The updated strategy (right) covers more object centers and ensures better alignment with ground truth boxes.*

We benchmarked ShelfVision against YOLOv5 in its COCO-pretrained form. As expected, the pretrained YOLOv5 model failed to detect any objects in the SKU-110K dataset, producing zero true positives, mAP, or IoU. This is largely due to its lack of exposure to dense, shelf-style layouts—resulting in missed detections or aggressive NMS suppression. ShelfVision, while early in development, produced 19 true positives, achieved a non-zero mAP of 0.0069, and maintained an average IoU of 0.3875, demonstrating early-stage learning. This comparison highlights the importance of domain-specific tuning: while YOLOv5 is highly optimized for general object detection, ShelfVision is already beginning to generalize to retail shelves despite its limited training window. While we also attempted to fine-tune YOLOv5 on SKU-110K, our visual comparisons revealed limited improvement in detection accuracy, with ShelfVision continuing to offer better ground truth alignment despite fewer predictions.

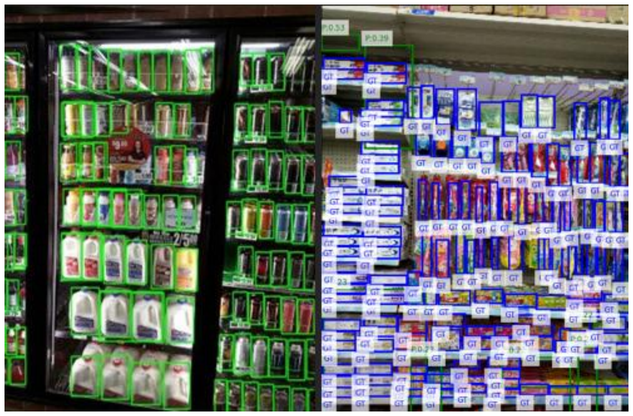

Fig. 12. *Visual comparison between YOLOv5 (fine-tuned, left) and ShelfVision (right). YOLOv5 produces dense predictions across uniform product layouts, while ShelfVision shows sparser predictions (green) overlaid on ground truth boxes (blue), reflecting early but targeted learning on SKU-110K shelves.*

V. CONCLUSION

This project set out to address the challenges of object detection in densely packed retail environments by developing ShelfVision, a custom anchor-based model tailored to the SKU-110K dataset. Initial baseline attempts were unsuccessful, with models failing to produce meaningful outputs due to poor anchor-ground truth alignment, unstable loss behavior, and uncalibrated prediction thresholds. These early-stage failures highlighted the need for specialized handling of dense visual scenes—where default detection architectures are prone to collapse without domain-specific tuning.

Through a series of targeted architectural and training improvements—including anchor scale tuning, dynamic IoU-based matching, and loss rebalancing—ShelfVision transitioned from a non-functional baseline to a measurable detection system. The model achieved a peak mAP of 0.0069 and an average IoU of 0.3875 across a test subset, accompanied by a true positive count of 19 detections. While modest in absolute terms, these metrics reflect genuine early-stage learning in one of the most difficult detection settings, with ShelfVision beginning to localize shelf items with reasonable spatial alignment.

A key part of this progress was the development of custom debugging tools. Visualization scripts for anchor coverage, delta decoding, and prediction overlays played a central role in diagnosing model behavior and validating each iteration. Additional test scripts, including overfitting tests on small batches, helped verify component-level functionality and confirm that the network architecture and training loop were behaving as expected. These tools not only accelerated development but also enabled a clearer understanding of why specific improvements were effective.

Despite this forward momentum, the system remains in an early phase. Precision and recall are still low, and the model often outputs noisy or low-confidence predictions. The current training pipeline is limited by dataset subset size, restricted training time, and simplified architecture. Post-processing heuristics also remain permissive, which inflates false positive counts. These constraints leave substantial room for refinement.

Future work will focus on expanding the training scale, improving prediction calibration, and integrating multi-scale refinement. Threshold tuning and more aggressive filtering could help balance recall with precision. Longer training on the full SKU-110K dataset, combined with deeper architectural exploration (e.g., enhanced FPNs or transformer-based heads), could further improve model generalization. Ultimately, ShelfVision now provides a working foundation upon which more advanced detection logic can be layered and evaluated.

VI. TEAM CONTRIBUTIONS

Colin Kirby led the development of the ShelfVision model architecture, including implementation of the ResNet-50 backbone, Feature Pyramid Network, detection head, and anchor generation logic. He also created the debugging and anchor visualizers used throughout development, and contributed to training stability by tuning loss tracking and early stopping mechanisms. Colin additionally handled the design and layout of the presentation visuals.

Nathan Little focused on evaluation and debugging, identifying critical issues like the 0 mAP failure and helping refine the model's matching and post-processing logic. He tuned training configurations to improve batch stability and overall performance, and interpreted experimental results to guide architectural changes. Nathan also wrote the majority of the presentation content, summarizing system improvements and model performance.

REFERENCES

[1] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," arXiv:1804.02767, Apr. 2018. [Online]. Available: https://arxiv.org/abs/1804.02767

[2] G. Jocher *et al.*, "YOLOv5 by Ultralytics," GitHub repository, 2020. [Online]. Available: https://pytorch.org/hub/ultralytics_yolov5/

[3] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature Pyramid Networks for Object Detection," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, USA, July 2017, pp. 936–944.

[4] E. Goldman, R. Herzig, A. Eisenschtat, J. Goldberger, and T. Hassner, "Precise Detection in Densely Packed Scenes," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 5227–5236. [SKU-110K Dataset]

[5] J. Lee, H. Seok, Y. Ryu, and J. Kim, "Object detection on retail shelves: Real-world challenges and considerations," arXiv:2305.17438v2, May 2023. [Online]. Available: https://arxiv.org/html/2305.17438v2

[6] A. A., "Retail Store Item Detection using YOLOv5," Roboflow Blog, May 2021. [Online]. Available: https://blog.roboflow.com/retail-store-item-detection-using-yolov5/

[7] Ultralytics Docs, "SKU-110K Dataset for Object Detection," Ultralytics.com, 2023. [Online]. Available: https://docs.ultralytics.com/datasets/detect/sku-110k