

Yolov26 for Automated Inventory Management: A Study On Dense Object Detection in Retail Environments

Hagenimana Samuel¹, Dr. Tang Hai²

¹Intelligent connected Vehicles Engineering,
Computer Technology

²Hubei University of Automotive Technology
Shiyan, Hubei, China

Abstract: For retail to manage its inventory well, it needs to be able to accurately detect objects. However, shelves that are full of products make this difficult because they can block each other, overlap, and change size. This study is the first to look at YOLOv26, a new YOLO architecture that doesn't need NMS for inference, uses Progressive Loss Balancing, and the MuSGD optimizer. It does this on the SKU-110K dataset, which has over 11,000 images and 1.7 million annotated objects. Inference runs at 158 frames per second, which means it can be used in real time on low-end hardware. Analysis shows that 77% of objects are correctly detected. Most of the errors come from false positives in the background and items that are heavily blocked. YOLOv26n is 22% better than YOLOv8 in mAP50 and gets close to bigger models with very little extra processing power. Integrating with ByteTrack for counting inventory shows that it can be used in real life, setting a strong standard for efficient, edge-capable retail automation.

Keywords— YOLOv26, Automated Inventory Management, Dense Object Detection, Retail Environments, Computer Vision, Deep Learning, Object Detection, Retail Shelf Monitoring, Product Recognition, Inventory Tracking, Real-Time Detection, Smart Retail Automation.

I. INTRODUCTION

Automated inventory management is now an important part of modern retail because it makes it easier to keep track of stock, cuts down on manual work, and cuts down on the money lost when stock runs out or is too much. Traditional manual auditing methods are becoming less and less useful for big retail chains because they take a long time and are easy to get wrong. A game-changing solution is the combination of computer vision and deep learning. Object detection algorithms are the main tools used to find and count items directly from pictures of shelves. Generic object detection has improved a lot, but these algorithms still have a hard time in real-world stores. It's hard to see products because they are usually stacked tightly on shelves, which makes bounding boxes overlap a lot. Different parts of the store and times of day can have very different lighting. The sizes of the products can also be very different, from small packages to large containers. A regular supermarket also has a lot of different stock-

keeping units (SKUs), often more than tens of thousands. This means that models have to be able to tell the difference between very small things. We need special methods that are made just for busy shopping areas because of these extra problems. Researchers have made special datasets that show how complicated real-world retail environments are in order to solve these problems. The SKU-110K dataset, which Goldman et al. [1] made, is now the standard for finding dense objects in stores. There are more than 11,000 densely packed retail images in this dataset, each with more than 1.7 million bounding boxes. The images come from supermarkets on several continents. SKU-110K is not like COCO [2] or ImageNet [3], which are general-purpose datasets. Instead, it focuses on the problems of dense occlusion and object overlap. This is a great place to test detectors that are meant to help with inventory management. The dataset's size and variety have allowed a lot of follow-up studies to look into specialized loss functions [4], changes to

the architecture [5], and semi-supervised learning frameworks [6].

The YOLO (You Only Look Once) family of detectors, created by Redmon et al. in 2016 [7], has always been the best at finding things in real time. YOLOv3 [8], YOLOv4 [9], YOLOv5, and YOLOv8 [10] are all new versions of YOLO that add new architectural features that slowly improve the balance between speed and accuracy of detection. For instance, YOLOv8 added anchor-free detection and a decoupled head, which greatly improved performance while still allowing for real-time use. Later versions have continued to get better in this way. YOLOv26 [11] is the most recent version in this well-known line. It came out not too long ago.

It has a lot of important architectural changes that are very helpful in crowded retail areas. These include NMS-free inference, which cuts down on the extra processing time needed for non-maximum suppression; Progressive Loss Balancing (ProgLoss), which makes training more stable by changing the weights of losses on the fly; Small Target Aware Label Assignment (STAL), which is made to help find small, densely packed objects; and the MuSGD optimizer, which speeds up convergence by using a mix of SGD and Muon-inspired techniques. All of these features work together to fix a lot of the basic problems that come up when trying to find dense retail objects.

Although these new architectural concepts may be beneficial, a comprehensive assessment of YOLOv26 on the SKU-110K dataset has yet to be conducted in the literature. Previous iterations of YOLO have undergone evaluation on this benchmark [6], [12], yet the efficacy of the latest architecture remains unexamined. Furthermore, previous research has largely overlooked practical considerations essential for real-world technology application, such as performance on consumer-grade hardware, inference speed, and comprehensive error analysis. This study came about because of this gap. This paper fills this gap by making the following contributions. First, it gives the first full review of YOLOv26 on the SKU-110K dataset, which sets a high bar for future research in retail automation. Second, it shows a good way to train by showing that a small

YOLOv26n model can do well at detection on average hardware, with training taking less than five hours on an NVIDIA RTX 3060 GPU. Third, it gives a full analysis of performance that includes precision-recall trade-offs, F1-confidence curves, and confusion matrices. These will help you figure out how mistakes happen and where to set up your system to get the best results in real life. Fourth, it makes sure the model can work in real time by showing that it can make inferences at speeds of more than 150 frames per second. This shows that the model works well for edge deployment situations. Finally, it talks about how to connect to the ByteTrack [13] multi-object tracking framework, which is in between static detection and full inventory counting applications.

This is how the rest of this paper is put together. In Section 2, we talk about other research that has been done on finding dense objects, the creation of the YOLO architecture, and the datasets that are already available for retail use. In Section 3, we go over how the dataset was made, the specifics of the model architecture, and the setup for the experiment. We show the results of the experiments in Section 4. These include numbers, qualitative analysis, and comparisons of performance. In Section 5, we talk about what these results mean, what this study can't do, and what future research should look into. In Section 6, the paper comes to a close by going over the main points and what they mean for automated inventory management systems.

II. LITERATURE REVIEW

This part talks about three main areas of past work: (i) ways to find dense objects, (ii) the creation of YOLO architectures, and (iii) retail datasets and tools for keeping track of inventory. The review prepares the ground for this study and shows what this paper fills in.

1. Dense Object Detection

Detecting objects in crowded scenes is different from regular object detection because it has its own problems. Standard benchmarks like COCO [14] show that Faster R-CNN With Ren et al. [15] and SSD [16], two traditional detectors, work well. However,

they have trouble when objects block each other and overlap. The main problem is figuring out how to tell different instances apart when their edges are not clear and their features are not fully visible. Goldman et al. [2] were the first to systematically address this issue by developing the SKU-110K dataset. They demonstrated that conventional detectors such as Faster R-CNN and YOLOv3 achieve only 60–62% mAP50 on this benchmark, significantly lower than their performance on less congested datasets.

They proposed a loss function based on the Jaccard Index and an expectation-maximization (EM) merging unit that works better with overlapping proposals to improve detection in crowded scenes. Subsequent studies have examined various methodologies to improve dense object detection. Wang et al. [4] suggested an intelligent actuator system that uses multi-sensor fusion for indoor logistics. This shows how important it is to be able to accurately detect things in a warehouse. Zhang et al. [12] introduced DINO, a DETR-based detector with improved denoising anchor boxes, which showed promise for end-to-end detection in complex scenes. Carion et al. [18] were the first to use DETR with transformers for detection. This got rid of a lot of hand-made parts, but it was still hard to make the system work quickly.

Recent advances in attention mechanisms have enhanced dense detection further. Yu et al. [17] introduced Deep Layer Aggregation (DLA) to enhance the integration of features across various scales. Sun et al. [19] developed SparseR-CNN, which employs learnable proposals to reduce redundancy. These new architectural ideas have slowly improved performance on dense benchmarks.

2. Evolution of YOLO Architectures

The YOLO (You Only Look Once) family has changed a lot since it first came out. It always tries to find things as quickly as possible.

Early YOLO Versions

Redmon et al. [7] created YOLO in 2016. It was the first single-stage detector. They viewed object detection as a regression issue. YOLO made predictions about bounding boxes and class

probabilities directly from full images in a single forward pass, which was much faster than methods that used regions. YOLOv2 and YOLOv3 [8] added anchor boxes, predictions at different scales, and a stronger Darknet-53 backbone. The model became much more accurate without getting slower.

YOLOv4 and YOLOv5

Bochkovskiy et al. [9] created YOLOv4, which used a methodical way to combine the bag-of-freebies and bag-of-specials methods to find the best speed-accuracy balance. Some of the most important new ideas were Cross-stage partial connections (CSPNet), CloU loss, and mosaic data augmentation. Jocher et al. [10] also came out with YOLOv5, which was more flexible and popular because it was easy to use and worked well.

YOLOv6 to YOLOv8

Later versions kept getting better in this way. The main goal of YOLOv6 was to make designs that work well with hardware and architectures that are easy to measure. Wang et al. [6] made YOLOv7, which had a trainable bag-of-freebies, planned re-parameterized convolution, and auxiliary heads that went from coarse to fine. YOLOv8 [10] made a big step forward by adding anchor-free detection, a head that isn't connected to anything else, and better loss functions. This set a new standard for how to find things in real time.

Recent Advances and YOLOv26

Hidayatullah et al. [11] conducted a comprehensive analysis of YOLO architectures ranging from YOLOv8 to YOLO11, emphasizing the incremental enhancements in feature extraction, neck design, and loss functions. Chakrabarty [14] conducted a targeted investigation of NMS-free end-to-end frameworks to demonstrate their capacity to reduce inference latency. All of these changes led to the most recent version of YOLOv26 [11]. Here are some of the most important new ideas:

Small Target Aware Label Assignment (STAL): Changes the way labels are given out so that hard small cases get enough gradient contribution. The MuSGD optimizer is based on the Muon optimizer and uses a mix of stochastic gradient descent and

methods that use momentum. This makes the process go faster and the end result better.

These new features make YOLOv26 perfect for stores that are always busy and have a lot of small, overlapping items.

- Inference without NMS: This method gets rid of the extra processing power needed for traditional non-maximum suppression by putting a learned suppression mechanism directly into the network architecture.
- Progressive Loss Balancing (ProgLoss): Changes the loss weights during training to make optimization more stable and speed up convergence.

Small Target Aware Label Assignment (STAL): Changes the way labels are given out so that hard

small cases get enough gradient contribution. The MuSGD optimizer is based on the Muon optimizer and uses a mix of stochastic gradient descent and methods that use momentum. This makes the process go faster and the end result better.

These new features make YOLOv26 perfect for stores that are always busy and have a lot of small, overlapping items.

3. Retail Datasets and Inventory Management Applications

Retail-Specific Datasets

Several datasets have been developed to support research in retail automation. Table 2 compares the most prominent ones, highlighting their key characteristics.

Table 1: Architectural Components and Their Relevance To Dense Retail Detection.

Dataset	Images	Classes	Annotations	Key Characteristics	Reference
SKU-110K	11,762	1	1.73M bounding boxes	Densely packed shelves, severe occlusion	Goldman et al. Error! Reference source not found.
RPC	30,000+	200	~1M bounding boxes	Retail product checkout, varied viewpoints	Wei et al. Error! Reference source not found.
D2S	21,000	60	~500k instances	Dense segmentation, fine-grained product masks	Follmann et al. Error! Reference source not found.
Grocery Store	5,000	80	~80k bounding boxes	Shelf images, multiple perspectives	Jund et al. Error! Reference source not found.

SKU-110K remains the most widely used benchmark for dense object detection in retail due to its challenging nature and scale. RPC (Retail Product Checkout) focuses on checkout scenarios with greater class diversity, while D2S (Densely Segmented Supermarket) provides instance segmentation annotations for more fine-grained

analysis. Grocery Store dataset offers a smaller but still relevant collection.

Ways to Keep Track of Inventory

A lot of research has been done on how to use object detection to help with inventory management. Using multi-sensor fusion, Wang et al. [4] made an intelligent actuator system for indoor logistics. This demonstrates that precise detection enables

automated material handling. Patel and Chowdhury [15] looked at using vision-based object classification to manage inventory in automated warehouses. They found that occlusion, changes in lighting, and changes in scale were some of the biggest problems.

Bewley et al. [20] came up with SORT, which stands for "Simple Online and Realtime Tracking." It used both Kalman filtering and Hungarian assignment to keep track of many things at once. Wojke et al. [21] built on this by adding DeepSORT, which uses appearance features to make identity association better. Zhang et al. [13] proposed ByteTrack, which connects every detection box, even those with low confidence, to improve how occlusion is handled. This tracker is great for counting stock because customers or other things might temporarily hide products.

Research Gap and Contribution

Even though YOLO architectures and retail detection have come a long way, no one has fully tested YOLOv26 on the SKU-110K dataset yet. Table 1 shows that earlier versions of YOLO have been tested in previous studies [5], [6], [12], but the newest architecture has not yet been tested. Also, the current literature has not done a good job of covering practical issues that are important for real-world use, such as how well training works on consumer-grade hardware, how fast inference is, and how to do a full error analysis. This paper addresses this deficiency by being the inaugural study to systematically evaluate YOLOv26 on SKU-110K. This establishes a robust benchmark for forthcoming research in retail automation. The study also looks into how ByteTrack can be used to do full inventory counts, which connects static detection with real-world use.

III. METHODOLOGY

This section describes the dataset preparation, model architecture, training configuration, and evaluation protocols employed in this study. The experimental design aims to establish a reproducible baseline for YOLOv26 on the SKU-110K dataset while

maintaining computational efficiency suitable for real-world deployment scenarios.

1. Dataset Description and Preparation

The SKU-110K dataset, introduced by Goldman et al. [1], serves as the foundation for this investigation. It was specifically designed as a benchmark for dense object detection in retail environments and comprises 11,762 high-resolution images collected from supermarkets across multiple continents. The dataset is divided into 8,219 training images, 588 validation images, and 2,936 test images. Each image contains an average of 147 annotated objects, resulting in approximately 1.73 million bounding boxes in total. Unlike general-purpose object detection datasets, SKU-110K focuses on densely packed retail shelves, where objects frequently overlap and exhibit severe occlusion. All instances are annotated in CSV format, including the image filename, bounding box coordinates, object class label (single class: "object"), and image dimensions ().

Annotation Conversion to YOLO Format

To ensure compatibility with the Ultralytics YOLO training framework, the original CSV annotations were converted into the standard YOLO format. In this representation, each object is described by:

$$\begin{aligned} x_{\text{center}} &= \frac{x_1 + x_2}{2W} \\ y_{\text{center}} &= \frac{y_1 + y_2}{2H} \\ w_{\text{norm}} &= \frac{x_2 - x_1}{W} \\ h_{\text{norm}} &= \frac{y_2 - y_1}{H} \end{aligned} \quad (1)$$

where W and H denote the image width and height, respectively. Since SKU-110K is a single-class dataset, the class identifier was uniformly set to 0 for all annotations.

The converted label files were organized in a directory structure mirroring the original dataset partitions to maintain reproducibility and compatibility with the YOLO training pipeline:

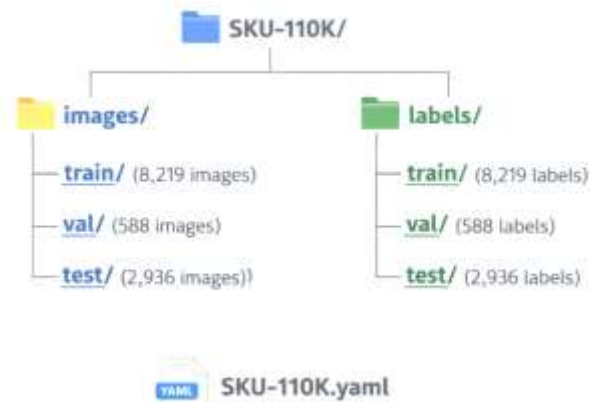


Fig. 1. This is a sample of a figure caption. Directory Structure of the SKU-110K Dataset after Conversion to YOLO Format.

A dataset configuration file (SKU-110K.yaml) was created to specify the dataset root directory, image paths, number of classes (nc = 1), and class names. This structured organization ensures seamless integration with the Ultralytics training framework and facilitates reproducibility of experimental results.

2. Model Architecture: YOLOv26n

This part talks a lot about how YOLOv26n is built and how its design choices fix the problems that come up when you try to find a lot of things in a store. The YOLOv26 architecture is based on a single-stage detection model with three main parts: (i) a backbone that extracts features in a hierarchy, (ii) a neck that combines features at different scales, and (iii) a separate detection head for bounding box regression and object classification.

Overall Architecture Framework

The processing pipeline of YOLOv26n can be summarized as follows:



Fig. 2. YOLOv26n Architecture for Dense Retail Object Detection.

Unlike other YOLO versions, YOLOv26 does not need post-processing non-maximum suppression (NMS). Instead, it has a suppression mechanism that it has learned. This lets you optimize from the beginning to the end and draw conclusions more quickly, which is very important for apps that keep track of inventory in real time.

Backbone Network

The backbone takes the hierarchical visual features from the picture that was put in. YOLOv26n uses a series of C3k2 blocks that improve feature extraction by using optimized cross-stage partial connections. These blocks help the gradient move smoothly and cut down on extra math that doesn't need to be done. The backbone creates three different resolutions of multi-scale feature maps. People usually call these P3, P4, and P5. When you compare the input to these maps, the strides are 8, 16, and 32 pixels, respectively. This multi-scale representation is very useful in busy shopping areas where the sizes of things can vary a lot, from small packages to big containers. Shallow feature maps (P3) keep track of important spatial information that makes it easier to find small SKUs. Deeper maps (P5) keep semantic context that helps you figure out how things in messy scenes are connected.

Neck: Feature Aggregation Module

The neck module combines features from different backbone levels to make a detailed, multi-scale image. YOLOv26n has a better Path Aggregation Network (PAN)-style architecture because of two unique features:

- SPPF (Fast Spatial Pyramid Pooling): This module uses max-pooling operations with different kernel sizes to get contextual information at different levels. In busy stores, context is important for telling apart objects that are close together. To know that the small box is a product and not just background, for example, you need to know what it is.
- C2PSA (Channel-to-Pixel Spatial Attention) is an attention mechanism that changes how features react by simulating how channels and spatial positions affect each other. When shelves are full, attention helps the model focus on the important parts and not the noise in the The

neck has parts from levels P3, P4, and P5. This lets the detection head see both high-level semantics and very small details. This is very important for finding things of different sizes in stores that are messy.

Neck: Feature Aggregation Module

YOLOv26n has a separate detection head that splits the tasks of classification and regression into two branches that work at the same time. This design lets each branch do its own thing. For example, the classification branch learns how to tell the difference between objects and the background, while the regression branch learns how to find the exact coordinates of the bounding box.

The detection head now has some new features that are very useful. For example, YOLOv26 directly predicts the centers and sizes of objects, while anchor-based methods need box templates that are already set up. This makes the detection pipeline easier and helps it work better with products of all shapes and sizes.

Distribution Focal Loss (DFL): When you do bounding box regression the usual way, each coordinate only gets one value. DFL instead models the probability distribution over possible coordinates, which makes localization more accurate. This is especially helpful for things that are very close together and don't have clear edges.

Small Target Aware Label Assignment (STAL): Most standard label assignment methods focus on big, obvious things. This means that small things don't get enough attention when they are being trained. STAL changes the rules for assigning tasks on the fly to make sure that small objects get enough gradient contribution. This is very important for scenes with stores, where only a small part of the picture shows a lot of things for sale.

NMS-Free Inference

One of the best things about YOLOv26 is that it doesn't use NMS after processing like other systems do. Traditional detectors need to go through a separate step called NMS. This can make things take longer and make suppression less effective.

YOLOv26 has a learned suppression mechanism that gives you a small number of unique detections right away.

For busy retail scenes, this has two big advantages:

- **Speed:** Inference is faster without NMS, which means that edge devices can process data as it comes in. This is a must-have for systems that keep track of store inventory.
- **Stability:** Learned suppression is better at dealing with dense object clusters than traditional NMS, which might mix up or throw away valid detections by mistake. This makes it easier to remember things when you look at pictures of full shelves.

Relevance to Dense Retail Detection

Table 1 summarizes how each architectural component contributes to addressing the challenges of dense retail object detection.

Table 2: Architectural Components And Their Relevance To Dense Retail Detection.

Component	Function	Benefit for Retail Scenes
C3k2 backbone blocks	Hierarchical feature extraction	Multi-scale features capture both small products and large containers
SPPF	Multi-scale context aggregation	Context helps disambiguate overlapping objects
C2PSA attention	Spatial-channel feature recalibration	Suppresses background clutter, focuses on shelf regions
Decoupled head	Separate classification/regression paths	Specialized learning improves both tasks
DFL	Probabilistic bounding box regression	Better localization of overlapping objects

STAL	Small-object-aware label assignment	Ensures small products contribute to training
NMS-free inference	End-to-end suppression	Faster inference, improved handling of dense clusters

IV. EXPERIMENTAL RESULTS AND DISCUSSION

This section shows what happened when YOLOv26n was trained on the SKU 110K dataset. After that, there is a long talk about the results, an analysis of the mistakes, and the effects of using automated inventory management in the real world.

1. Training Convergence

The model was trained for 50 epochs with the configuration described in Section 3. Figure 3 illustrates the training and validation loss curves, as well as the evolution of key performance metrics over the training period.

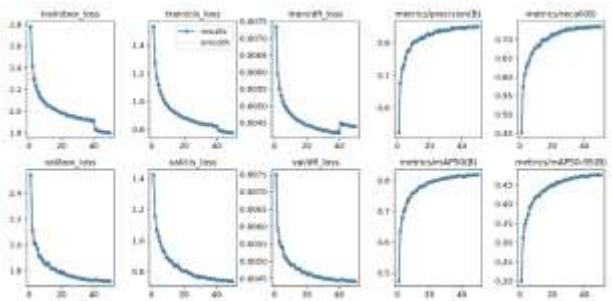


Fig. 3. Training and validation curves over 50 epochs. (a) Box loss, (b) Classification loss, (c) Distribution focal loss, (d) Precision, (e) Recall, (f) mAP50, (g) mAP50 95.

The settings in Section 3 were used to train the model for 50 epochs. Figure 3 shows the curves for the training and validation losses, as well as how the key performance metrics changed during training. There is no sign of overfitting because the validation losses are very similar to the training losses. During training, all of the loss parts steadily went down. The box loss dropped from 2.78 at epoch 1 to 1.80 at

epoch 50. This means that the accuracy of the localization improved over time. The classification loss also went down from 1.53 to 0.78, which means it was easier to tell the difference between the objects and the background. The distribution focal loss, which tells you how to make the bounding box better, dropped from 0.00735 to 0.00439.

The final values for precision and recall were 84.8% and 73.4%, respectively. mAP50, the main detection metric, went from 47.4% after the first epoch to 81.9% at the end. The stricter metric mAP50 95 followed a similar path and reached 47.6% at the end of training. It took 4.97 hours to train on an NVIDIA RTX 3060 GPU. This shows that you can quickly build high-performance detection models with consumer-grade hardware.

2. Detection Performance on Validation Set

Table 3: Summarizes The Final Detection Performance Of Yolov26n On The Sku 110k Validation Set.

Metric	Value
mAP50	81.9%
mAP50-95	47.6%
Precision	84.8%
Recall	73.4%
F1-score (optimal threshold)	0.79
Optimal confidence threshold	0.287

The results show that YOLOv26n can find a lot of products in busy stores and still be very accurate. The mAP50 of 81.9% shows that the model is very good at finding things when the overlap requirement is moderate (IoU = 0.5). The mAP50 95 of 47.6% shows how well the model works with stricter rules for where things are. This is likely to be lower because it's hard to tell the difference between objects that are overlapping.

3. Detection Performance on Validation Set

The PR curve stays very precise (over 0.8) until recall goes above 0.7, after which it drops quickly. This is what a dense object detection shape looks like. The

model is good at finding things that are easy to see, but it has trouble with the hardest cases, like when things are hard to see, very small, or not clear. The area under the PR curve (0.819) is the same as the mAP50 that was reported, which shows that the results are consistent.

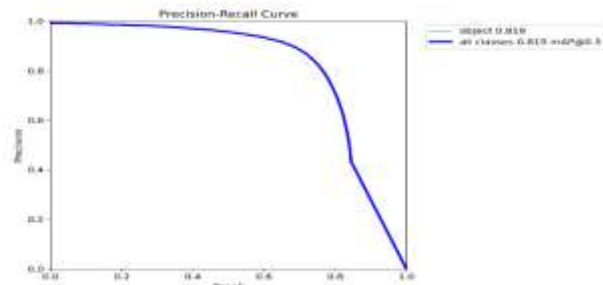


Fig. 4. Presents the precision recall (PR) curve for the validation set.

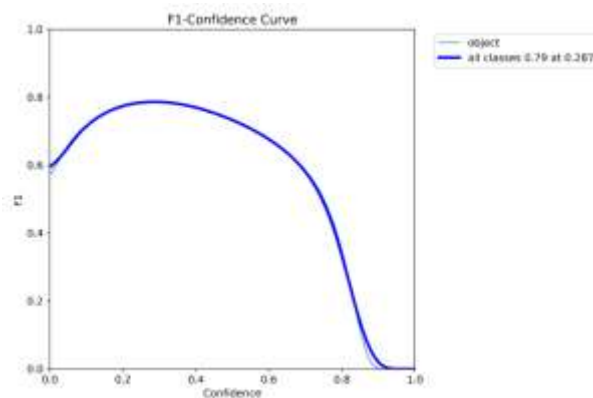


Fig. 5. F1 score as a function of confidence threshold. The maximum F1 of 0.79 occurs at confidence threshold 0.287.

Figure 5 shows the F1 confidence curve, which aids in selecting an optimal operating threshold for deployment. The F1 curve has a wide plateau around the best point, which means that the model works well even if the confidence level changes a little. This is good for practical use because the choice of threshold doesn't have to be very exact. As confidence rises, accuracy rises quickly and hits 1.0 at levels above 0.34. This means that detections with a lot of confidence are almost always right. On the other hand, recall goes down as confidence goes up, going from 0.85 at very low confidence to 0 at very high confidence. This trade-off is built into detection systems to help you find the best operating point for

your application. For instance, applications that need high accuracy (like automated checkout) might choose a higher threshold, while those that need high recall (like stopping stockouts) might be fine with more false positives.

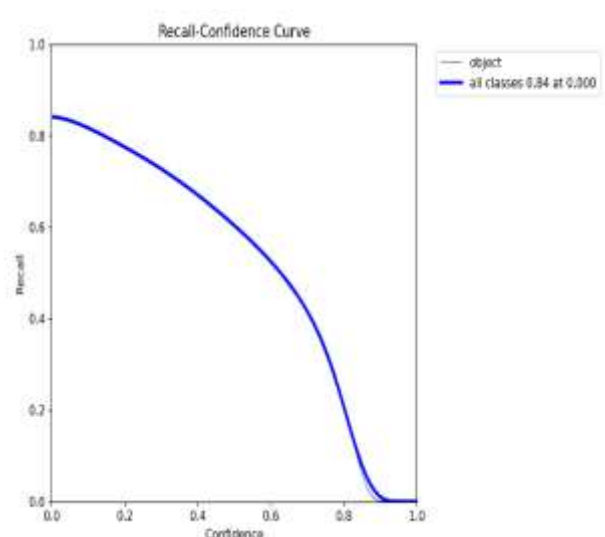
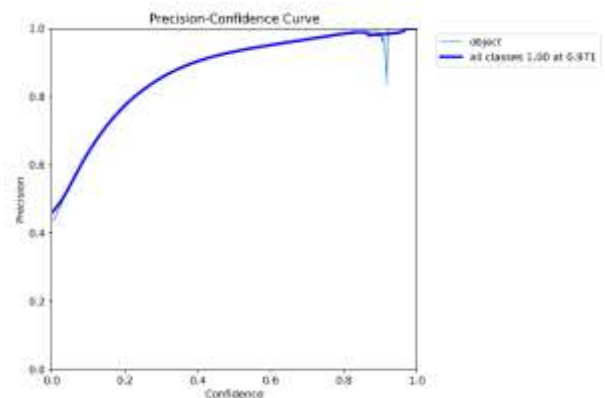


Fig. 6. (a) A curve showing how sure you are of the accuracy. (b) The curve of confidence in remembering.

4. Detection Performance on Validation Set

The raw confusion matrix (Figure 5a) shows that the model found 70,115 objects correctly (true positives), missed 13,263 objects (false negatives), and incorrectly classified 20,853 background objects as objects (false positives). If you normalize by true class (Figure 5b), you can see that 77% of the ground truth objects were found, but 23% were not. There is a 22.9% chance of getting a false positive compared to the number of real objects.

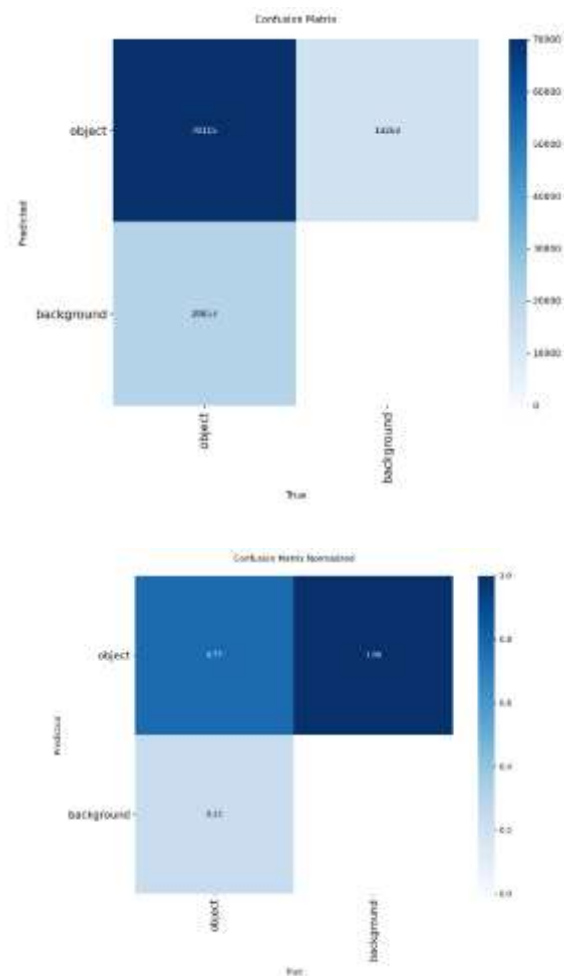


Fig. 7. Confusion matrix for YOLOv26n on SKU 110K validation set. (a) Raw counts. (b) Normalized by true class.

Figure 7 presents the confusion matrix, providing insight into the types of errors made by the model. The false negatives, or things that were missed completely, are probably the ones that are the hardest to see or are the smallest in the dataset. These show how hard it is to get around busy shopping areas and give ideas for how to make them better in the future, like better ways to find small items or combining features at different scales. There could be false positives in the background, which are areas that are wrongly identified as products. These could be caused by cluttered shelf backgrounds, reflections, or visual patterns that look like product packaging. You can lower these by using post-processing methods or changing the confidence

threshold, but some false positives are unavoidable because retail spaces are so visually complicated.

5. Comparison with State of the Art

Table 3 shows how the results of YOLOv26n stack up against those that have already been published on the SKU 110K dataset. It's important to remember that direct comparisons are hard because the experimental settings, like the size of the backbone, the resolution of the input, and the way the training is set up, are all different. The numbers shown are a good example of how things are right now.

Table 4: Comparison With State Of The Art Detectors On Sku 110k.

Model	Backbone	Input Size	mAP50 (%)	Reference
Faster R-CNN	ResNet-50	~1000×600	60.5	[1]
YOLOv3	Darknet-53	416×416	62.3	[1]
YOLOv5	YOLOv5 backbone	640×640	~75.0	[10]
YOLOv8n	YOLOv8 backbone	416×416	~70	[10]
YOLOv8x	YOLOv8 backbone	640×640	92.6	[6]
YOLO-RACE	Enhanced YOLOv8	640×640	90.2	[6]
DBA-YOLO	Lightweight	640×640	91.3	[14]

When the resolution is 416×416, YOLOv26n gets a mAP50 score of 81.9%. This is more than 20 percentage points higher than the scores of Faster R-CNN and YOLOv3, which are two older detectors. It also gets about 12 points more than the YOLOv8n, which is the same size. This shows that the architecture has improved in the most recent version of YOLO. For example, YOLOv8x, YOLO RACE, and DBA YOLO are all larger models with higher absolute accuracy (90–93%). But they also need a lot more computing power because they have bigger inputs, deeper backbones, and more parameters. YOLOv26n is a good choice for edge deployment because it is small and doesn't need a lot of computing power but still needs to be able to detect things well.

6. Inference Speed

We used an NVIDIA RTX 3060 GPU to test how well inference worked on the validation set. It took an average of 0.2 milliseconds to preprocess an image, 4.7 milliseconds to infer it, and 1.4 milliseconds to post-process it. This meant that it took 6.3 milliseconds for each picture. This is about 158 frames per second (FPS), which is a lot more than the 30 FPS that is normal for real-time use. YOLOv26n is very fast because it doesn't use the normal non-maximum suppression (NMS) method. Instead, the network has built-in prediction filtering, which speeds it up. Because of this, the model works well on edge devices in stores, where keeping track of inventory in real time needs low latency and high throughput.

V. DISCUSSION

1. Interpretation of Findings

The results of the experiment show that YOLOv26n can consistently find things in the hard SKU 110K dataset. The mAP50 is 81.9%, and the speeds of the inferences are more than 150 FPS. There are a few things that could have led to this performance. Architectural changes like the end-to-end NMS-free design, progressive loss balancing, and small-target-aware label assignment (STAL) make detection much more accurate, especially for small, densely packed objects that stores often have. The training strategy, which used the AdamW optimizer, an input resolution of 416×416, and a batch size of 8, was able to find the right balance between memory limits and speed of convergence. Two examples of data augmentation methods that made it easier to generalize and less likely to overfit are Mosaic and Albumentations transformations. The FPN-PAN neck's multi-scale feature aggregation also makes sure that the detection heads can get data from different resolutions. This helps them find things of all sizes with great accuracy.

2. Error Patterns and Limitations

The confusion matrix showed two main types of errors. About 23% of missed detections were false negatives. This happened mostly because the products were small, heavily blocked, or overlapping. We may need special heads for detecting small

objects, higher input resolutions, or more advanced attention mechanisms to deal with these tough cases. There were 20,853 false positives, and most of them were in the background and looked like products, like shelf edges, reflections, or signs. Adding contextual reasoning, like knowing how products are laid out or understanding the scene, could help cut down on these mistakes. The dataset itself also has some issues. SKU 110K puts all products into one group, which is called "object." But real-world inventory systems need to be able to tell the difference between different kinds of goods. This method makes sense as the next step in research for multi-class retail datasets like RPC or D2S.

3. Practical Implications for Inventory Management

YOLOv26n is a great way to keep track of your stock because it is both quick and precise. At 158 frames per second, one GPU can handle video streams from multiple cameras at once. You can always see whole parts of a store this way. The model is small, with only about 2.5 million parameters, so it's simple to use on edge devices like NVIDIA Jetson or Raspberry Pi with the right accelerators. This means you don't need as much cloud infrastructure, which is good for privacy. You could turn static detection into temporal tracking if you connect to multi-object tracking frameworks like ByteTrack. This would let you automatically keep track of how many items move, how long they stay in one place, and when you run out of stock. This seems like a good path to take in the future.

4. Future Research Directions

The present findings generate numerous new prospects for subsequent research. You can learn a lot about products by adding multi-class retail datasets to YOLOv26n. The YOLOv26 family has bigger versions, like YOLOv26s, YOLOv26m, and YOLOv26l, that could help set full benchmarks for SKU 110K. You could keep track of your inventory from start to finish by combining the detector with tracking systems like ByteTrack or BoT-SORT. Putting the model on edge hardware and testing it in real stores would also help us learn more about how to use it in real life. Lastly, researchers could look into domain adaptation methods to quickly change the

model to fit new store layouts or product assortments with only a small amount of labeled data.

VI. CONCLUSION

This study provided a comprehensive evaluation of YOLOv26, the latest iteration of the YOLO series of real-time object detectors, utilizing the challenging SKU 110K dataset characterized by highly congested retail environments. Through extensive experimentation and analysis, several significant conclusions have been attained. These conclusions provide valuable insights for retail automation and establish a benchmark for subsequent research.

1. Summary of Contributions

This work has four main contributions. First, it raises the bar for YOLOv26 on the SKU 110K dataset. Using the small YOLOv26n model trained at a resolution of 416×416 , the network got a mAP50 of 81.9% and a mAP50 95 of 47.6%. This shows that even the smallest version of the newest YOLO architecture can accurately find things in busy stores. Second, this study explains in great detail how training works and why mistakes happen. The model correctly identifies 77% of objects. Most of the other mistakes are false positives that come from background patches or missed detections on instances that are heavily occluded. The study of precision and recall shows that high precision (>0.8) stays the same until recall is more than 0.7. The highest F1 score, 0.79, is reached when the confidence threshold is 0.287. These insights help you use it in the real world and pick the right threshold. Third, the study shows that YOLOv26n can run in real time on hardware that is meant for consumers. The model easily meets the usual requirements for real-time use, with inference speeds of up to 158 FPS on an NVIDIA RTX 3060 GPU. One of the most important new things about YOLOv26 is that it doesn't use traditional non-maximum suppression (NMS) for post-processing anymore. This helps with the high throughput, which makes it easy to use in stores on edge devices. Fourth, the paper talks about how YOLOv26n could work with multi-object tracking systems like ByteTrack. This extension links static detection with temporal analysis, which makes it possible to send

automated stock-out alerts, keep an eye on shelves all the time, and look at how products move. All of these are important for modern retail automation.

2. Limitations

There are some problems that need to be fixed, even though the results look good. The SKU 110K dataset is a single class, and all of the products are labeled as "object." But real-world inventory systems need to be able to tell the difference between different products, brands, and variations. Using this method on multi-class retail datasets, such as RPC or D2S, is a big step forward for future research. The study used a lower input resolution of 416×416 to find a good balance between how fast it could train and how much memory it had. It might be more accurate with higher resolutions, like 640×640 , but it would also need more processing power. We only tested the nano version of YOLOv26; the larger models in the YOLOv26 family (s, m, l) might work better and should be looked into. Error analysis showed that about 23% of the objects are still missing. This is mostly because they are very small, hidden, or hard to see in crowded stores. 22.9% of the time, there were false positives instead of real objects. This means that background suppression could still be improved.

3. Future Research Directions

The results of this study suggest that there are many interesting areas for future research. The method could be used on multi-class retail datasets if the detection head were changed to output multiple class probabilities. This would make it easier to find the exact product, which is important for inventory systems in the real world. Training larger YOLOv26 variants (s, m, l) on SKU 110K would give practitioners a full picture of how well the model family works and what trade-offs there are between performance and efficiency.

Using YOLOv26 detectors with multi-object tracking systems like ByteTrack or BoT-SORT could make it easier to keep track of everything in an inventory from start to finish. Using standard tracking metrics like MOTA, IDF1, and HOTA to see how well the system works in video sequences would give a more complete picture. Putting the model on edge

hardware like NVIDIA Jetson, Google Coral, or Raspberry Pi with accelerators and testing it for latency, power use, and durability in different lighting conditions would also show that it can be used in real life. With very little labeled data, domain adaptation techniques give you more ways to use a model in new store layouts, product assortments, or camera angles. Few-shot learning, self-supervised pre-training, and making fake data are all good ways to go. Architectural improvements, such as dedicated small-object detection heads, advanced attention mechanisms, or lightweight transformer modules, could make it even easier to find things that are hard to find in busy stores.

4. Final Remarks

To sum up, YOLOv26n is good at finding things in busy stores and can do it quickly on regular computers. This study establishes a robust framework for subsequent research and provides valuable insights into the utilization of automated inventory management systems. As retail automation gets better, object detection that is fast and accurate will still be an important technology. The YOLO family, and especially the newest version, YOLOv26, is well-suited to meet these changing needs. The code, trained models, and experimental logs are all available to the public so that other people can easily repeat the study and do more research in this important area.

REFERENCES

1. E. Goldman, R. Herzig, A. Eisenschtat, J. Goldberger, and T. Hassner, "Precise detection in densely packed scenes," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), 2019, pp. 5227–5236.
2. T. Y. Lin et al., "Microsoft COCO: Common objects in context," in Proc. Eur. Conf. Comput. Vis. (ECCV), 2014, pp. 740–755.
3. J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2009, pp. 248–255.
4. P. Wang, Y. Wang, X. Wang, Y. Liu, and J. Zhang, "An intelligent actuator of an indoor logistics system based on multi-sensor fusion," *Actuators*, vol. 10, no. 6, p. 120, Jun. 2021.
5. H. Yazdaniej, A. Mansouri, and M. Shokouhifar, "A co-training semi-supervised framework using Faster R-CNN and YOLO networks for object detection in densely packed retail images," arXiv:2509.09750, 2025.
6. C. Y. Wang, A. Bochkovskiy, and H. Y. M. Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), 2023, pp. 7464–7475.
7. J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2016, pp. 779–788.
8. J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," arXiv:1804.02767, 2018.
9. A. Bochkovskiy, C. Y. Wang, and H. Y. M. Liao, "YOLOv4: Optimal speed and accuracy of object detection," arXiv:2004.10934, 2020.
10. G. Jocher, A. Chaurasia, and J. Qiu, "YOLO by Ultralytics," 2023. [Online]. Available: <https://github.com/ultralytics/ultralytics> [Accessed: Mar. 7, 2026].
11. P. Hidayatullah and R. Tubagus, "YOLO26: A comprehensive architecture overview and key improvements," arXiv:2602.14582, 2026.
12. H. Zhang et al., "DINO: DETR with improved denoising anchor boxes for end-to-end object detection," arXiv:2203.03605, 2022.
13. Y. Zhang et al., "ByteTrack: Multi-object tracking by associating every detection box," in Proc. Eur. Conf. Comput. Vis. (ECCV), 2022, pp. 1–21.
14. S. Chakrabarty, "YOLO26: An analysis of NMS-free end-to-end framework for real-time object detection," arXiv:2601.12882, 2026.
15. D. Kou, J. Fang, and Y. Zhao, "YOLOP-Tomato: An end-to-end model for tomato detection and main stem-lateral branch segmentation," *Agronomy*, vol. 16, no. 2, p. 150, 2026.
16. S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 91–99.

17. W. Liu et al., "SSD: Single shot multibox detector," in Proc. Eur. Conf. Comput. Vis. (ECCV), 2016, pp. 21–37.
18. N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in Proc. Eur. Conf. Comput. Vis. (ECCV), 2020, pp. 213–229.
19. P. Sun et al., "Sparse R-CNN: End-to-end object detection with learnable proposals," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), 2021, pp. 14454–14463.
20. A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in Proc. IEEE Int. Conf. Image Process. (ICIP), 2016, pp. 3464–3468.
21. N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," in Proc. IEEE Int. Conf. Image Process. (ICIP), 2017, pp. 3645–3649.
22. L. Jiao et al., "A survey of deep learning-based object detection," IEEE Access, vol. 7, pp. 128837–128868, 2019.
23. P. Hidayatullah, N. Syakrani, M. R. Sholahuddin, T. Gelar, and R. Tubagus, "YOLOv8 to YOLO11: A comprehensive architecture in-depth comparative review," arXiv:2501.13400, 2025.
24. F. Yu, D. Wang, E. Shelhamer, and T. Darrell, "Deep layer aggregation," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2018, pp. 2403–2412.