DeepACO: A Robust Deep Learning-based Automatic Checkout System

Long Hoang Pham, *Member, IEEE*, Duong Nguyen-Ngoc Tran, *Graduate Student Member, IEEE*,
Huy-Hung Nguyen, Tai Huu-Phuong Tran, Hyung-Joon Jeon, Hyung-Min Jeon,
and Jae Wook Jeon, *Senior Member, IEEE*,

Department of Electrical and Computer Engineering
Sungkyunkwan University, South Korea
{phlong, duongtran, huyhung91, taithp, joonjeon, hmjeon, jwjeon}@skku.edu *

Abstract

The retail industry has seen an increasing growth of artificial intelligence and computer vision applications. Of the various topics, automatic checkout (ACO) in retail stores or supermarkets has emerged as one of the critical tasks in this area. Several problems stem from real-world scenarios such as object occlusion, blurring from scanning motion, and similarity in scanned items. Moreover, the challenge also comes from the difficulty of collecting training images that reflect the realistic checkout scenarios due to continuous updates of the products. This paper proposes a deep learning-based automatic checkout system (DeepACO) to recognize, localize, track, and count products as they move along a retail check-out conveyor belt. The DeepACO follows the detect-and-track approach, i.e., applying trackers on detected bounding boxes. It also provides a completed pipeline for generating large training datasets under various environments from synthetic data. The proposed system has been evaluated on the 2022 AI City Challenge Track 4 benchmark. Compared to other state-of-the-art solutions, it has shown outstanding results, achieving top-2 on the test-set A with the F1 score of 0.4783.

1. Introduction

Recently, there has been a new revolution in the retail industry through the adoption of artificial intelligence (AI) and computer vision (CV) applications. Automatic checkout (ACO) is one of the critical problems in this area. When a customer puts their selected products on the checkout counter, an ideal ACO system is expected to accurately recognize each of these products and return a complete shopping list at one glance. When developing a vision- and deep learning-based ACO system, several problems must be considered, such as object occlusion, motion blur, items similarity, and the cost of miss-detection and miss-classification. Furthermore, the problem also comes from the large scale and the fine-grained nature of the product categories and the difficulty of collecting training images that reflect the realistic checkout scenarios due to continuous updates of the products. Overall, accuracy, stability, and effectivity are three vital concerns given real-world ACO systems.

This paper proposes a robust vision- and deep learning-based automatic checkout system (DeepACO) for the retail checkout process. The framework follows a detect-and-track pipeline which first detects candidate bounding boxes, then tracks persisted bounding boxes and assign unique IDs. The tracked bounding boxes are observed throughout the region of interest (ROI) and are counted when they move out of the ROI. To cope with the 2022 AI City Challenge Track 4's evaluation metric, the counting timestamp is recorded as the first time the whole object resides entirely inside the ROI. The DeepACO implements a general pipeline that accommodates different detection (and even instance segmentation) and tracking methods. However, Scaled-YOLOv4 [25] and SORT [1] are implemented and tested in this paper. The effectiveness of the proposed DeepACO framework has been evaluated in the 2022 AI City Challenge Track 4 [17]. A pipeline to generate training data for detection and tracking models is implemented to obtain training data. The generation algorithm randomly places and blends 116,500 synthetic images (patches) of the products over background images under various lighting environments. The results of DeepACO overcome other state-of-the-art (SOTA) methods and achieve the top-2 ranking with the F1 score of 0.4783.

In brief, the main contributions are as follows:

- A general framework for retail product detection and

*This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (2021-0-01364, An intelligent system for 24/7 real-time traffic surveillance on edge devices)

counting is built.

- A pipeline to generate training data (bounding boxes and instance masks) from scanned samples of the products.

- Extensive experiments demonstrated the effectiveness of the DeepACO, achieving the second place in the 2022 AI City Challenge Track 4 [17].

The rest of the paper is as follows. Section 2 lists related works. Section 3 discusses the data generation pipeline. Section 4 elaborates the DeepACO framework. Section 5 describes the experimental results. Finally, Section 6 concludes the findings in this paper.

2. Related Works

2.1. Object Detection

Moving object detection is one of the most fundamental tasks in computer vision. In the early years, moving objects in the image are detected using the background subtraction methods [9]. Then, objects' features are extracted by applying SIFT [15] or HOG [5] descriptors. Other methods use traditional classification models such as Support Vector Machines [10] or Random Forest [3]. However, This approach is prone to a high error rate because of the variety in objects' appearances and scale, as also noise and illumination.

The recent progress of deep neural network architectures has provided more reliable object detection methods. It avoids manual feature extraction and uses a data-driven approach that automatically allows machines to learn feature expressions. Two-stage detection architectures divide the detection process into the region proposal and the classification stage, and the well-known models are R-CNN [8], Fast R-CNN [7], Faster R-CNN [22]. On the other hand, one-stage detectors contain a single feed-forward, a fully convolutional network that directly provides the bounding boxes and the object classification. The widely used models are SSD [14] and YOLO [21] with variants such as YOLOv4 [2], YOLOv5 [11], or Scaled-YOLOv4 [25], etc. Most models are trained from scratch using either MS-COCO Detection Challenge [12], ImageNet Large Scale Visual Recognition Challenge [23], or PASCAL VOC Challenges [6].

2.2. Object Tracking

Multiple Object Tracking (MOT) plays an essential role in video-based application systems. Many existing MOTs are built as the post-processing task of detection models. The tracking could be run offline in traffic analysis or online, running real-time processing simultaneously with the camera or video input frame. When running offline, the search uses the detection results over the entire frame sequence for the offline methods and then performs global optimizations. The standard offline methods have structure as the graph model, which can be enhanced by using minimum cost flow [26], and subgraph decomposition [24].

On the other hand, the online method follows the tracking-by-detection paradigm. This approach uses the current and previous frames to link detection results while maintaining spatial and temporal consistency. Kalman Filter-based [1], and Neural Network [27,29] have been proposed to perform feature associations between tracking objects and new detection. These methods require no training and allow for fast-speed tracking.

2.3. Retail Product Datasets

Recent years have seen an increasing number of new datasets related to retail checkout. Some of the relevant datasets are listed as follows.

- **RPC** [28] is a large-scale retail product checkout dataset. It contains 83,739 images and 200 product categories with 367,935 annotated objects. It includes single-product images taken in controlled environment and multi-product checkout images taken at the checkout counter.

- **RP2K** [20] is a large-scale retail product dataset for fine-grained image classification. It has 350,000 images of more than 2000 different retail products. All images are captured manually in physical retail stores with natural lighting, matching the scenario of real applications.

- **AI City Challenge Retail Checkout** [17] is a dataset consisting of 116,500 synthetic images and several video clips from over 100 different merchandise items. The synthetic images are created from 3D scanned object models and will be used for training.

3. Data Generation

The key of an ACO system is a recognition system that can accurately predict the presence and count of each product in an arbitrary product combination. Usually, such a recognition system is trained with the images captured in the same environment as the deployment scenario. In the context of the ACO problem, the training image should be the one taken at the checkout counter, which captures a combination of multiple product instances. However, due to many product categories and the continuous update of the stock list, it is infeasible to learn the recognition model by enumerating all the product combinations.

A more practical solution is to collect examplar images of each product taken in a controlled environment. Hence, the data generation process for the ACO system is defined as

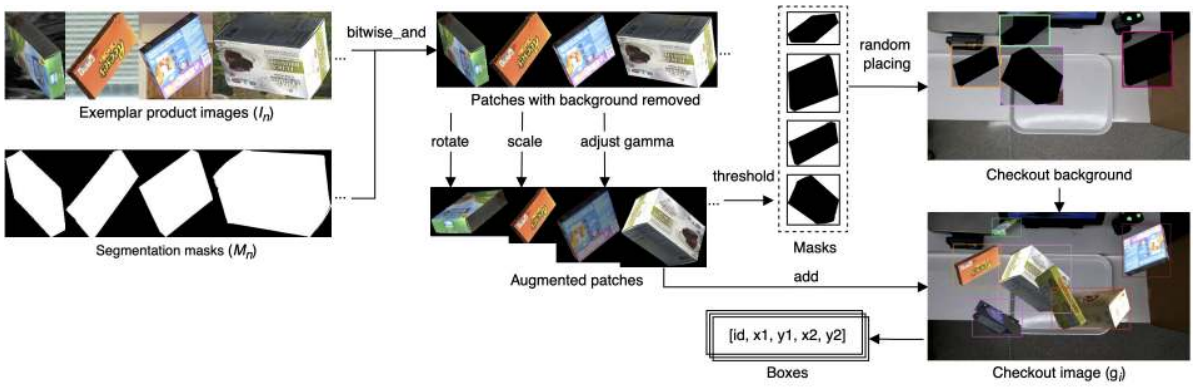


Figure 1. Checkout image generation process. Exemplar images are augmented (rotation, scaling, gamma adjustment) and are randomly placed on the checkout background image so that generated bounding boxes' IoUs do not exceed clutter ratio c.

Figure 2. Sampled checkout images of three clutter levels. From left to right: easy, medium, hard modes

follows. Given a set of products $\mathbf{P} = \{p_i\}$, a single-product exemplar image set $\mathbf{S} = \{(I_s, M_s, y_s)|y_s \in P\}$ is first collected, where l_s is a single-product image, M_s is the binary mask with the product's pixels marked in white, and y_s is its associated product ID/category. In this paper, the single-product exemplar set $\mathbf{S}_{aic} = \langle 116,500 \text{ images}; 116 \text{ ids} \rangle$ from the 2022 AI City Challenge Track 4 [17] is used. The exemplar images are created from 3D scanned products [30]. Example of $\mathbf{S}_{aic}$ can be shown in Fig. 1.

The recognition system can be trained on the synthetically generated checkout images by randomly choosing and placing $\mathbf{S}_{aic}$ on the checkout background with random orientations, occlusion, complex clutter, and lighting conditions. The checkout image set $\mathbf{G} = \{g_i\}$ is constructed using the following equation:

$$g_i = p\left(N, a, s, g, c\right) \tag{1}$$

where $\mathbf{N} = \{(I_n, M_n, y_n)\} \in \mathbf{S}_{aic}$ is a random chosen subset of n exemplar images. (a, s, g, c) are augmentation parameters applied on each I_n. Values of augmentation parameters are randomly chosen among the ranges defined in Table 1. Fig. 1 illustrates the detailed steps of checkout images generation process. The process starts with a "bitwise and" operation between I_n and M_n to remove the background in the exemplar images. Then augmentation operations (rotation, scaling, and gamma adjustment) are applied on each patch using hyperparameters (a, s, g). Next, patches are inverted thresholded to obtain masks. Masks are randomly placed on the background image (extracted from a real-world checkout video) so that bounding boxes IoUs do not exceed the clutter ratio c. Examples of generated checkout images with three different clutter levels can be found in Fig. 2.

Variable	Description	Value ranges
n	Number of products per image	$[7, 12]$
a	Rotation angle	$[0, 360]$
s	Scale ratio	$[0.8, 1.2]$
g	Gamma adjustment	$[0.8, 1.0]$
c	Clutter ratio	$[0.1, 0.5]$

Table 1. Augmentation parameters for checkout image generation.

4. DeepACO

The DeepACO system consists of three components: a detector, a tracker, and a counter. As the name suggests, deep learning-based detectors and trackers are used. Different counters can be built based to suit various application requirements. An overview of the DeepACO system can be seen in Fig. 3.

4.1. Object Detector

The DeepACO layouts a flexible detection pipeline that allows different object detection models to be interchangeable. As shown in Fig. 3, models only need to comply with the batch-ized input and output formats. Hence, given a batch of RGB images $\mathbf{I} = \{I_i|t \leq i \leq t + B\}$ captured at time $[t, t + B]$ (or $\mathbf{I} \in \mathbb{R}^{B \times C \times H \times W}$), the detector returns a batch of detections $\mathbf{D} = \{D_i \in \mathbb{R}^{P \times F}|t \leq i \leq t + B\}$ where B is the batch size, P_i is the number of detections in batch i, and F is a list of features $[x1, y1, x2, y2, cls, conf]$ (top left x and y, bottom right x and y, class id, and confidence score). Each detection is incorporated with the corresponding time step index (i.e, frame index), keeping the output in a timely order.

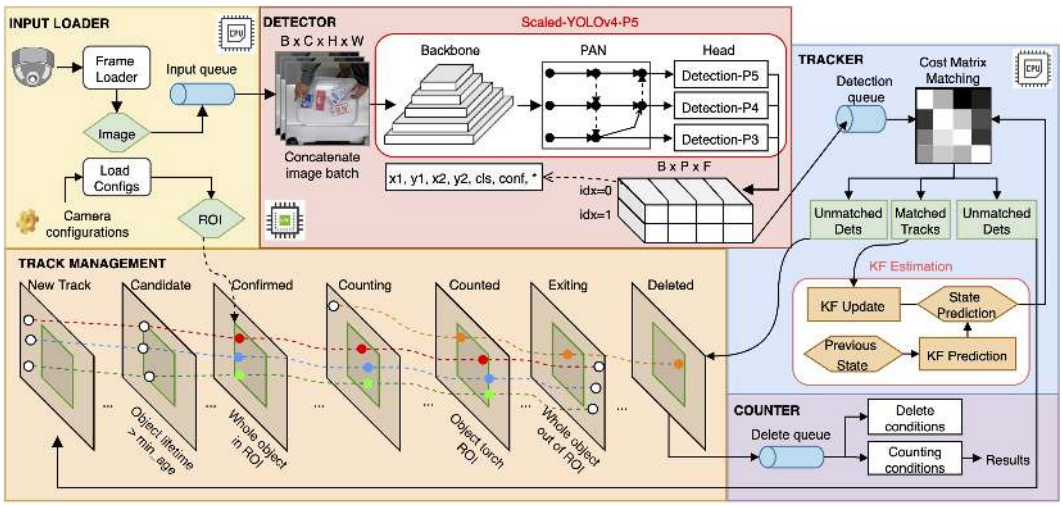


Figure 3. Sampled checkout images of three clutter levels.

Scaled-YOLOv4 [25] is an improvement of the YOLOv4 [2] object detection by applying the Cross Stage Process (CSP) design over key components, including the Darknet backbone and Path Aggregation Network (PAN) [13] neck. Additionally, up-and-down scaling is applied to YOLOv4-large to achieve high accuracy of 51.8%, 54.5%, and 55.5% mAP (Scaled-YOLOv4-P5, P6, and P7, respectively) on the MS-COCO test set while reaching real-time performance at 43, 32, and 17 FPS respectively. Regarding scalability, variants of Scaled-YOLOv4 models can be deployed to a wide variety of cloud GPU or low-end GPU devices. In Fig. 3, Scaled-YOLOv4-P5 [25] is implemented as the baseline detector.

The training of Scaled-YOLOv4 detectors consists of 15,642 synthetically generated checkout images from Section 3. The standard split of 80%-20% training-testing is used to finetune the MS-COCO pretrained models. The same training configuration as in [25] is used. Additionally, basic augmentation (rotate, translate, crop, etc.) and advanced augmentation (RandAugmentation [4], and mosaic [2]) are also employed. Unintentionally, a double augmentation operation is performed considering augmentation operations during checkout image generation. Hence, the resultant models can avoid overfitting and facilitate multi-scale adaptability. The models are trained for 50 epochs with Stochastic Gradient Descent (SGD), in which the best weights evaluated on the test set are selected for inference.

4.2. Hand Estimator

One of the requirements for the 2022 AI City Challenge Track 4 [17] is to identify hand-handled products by customers. Hence, the ability to perceive the shape and motion of hands can be a vital component in improving product

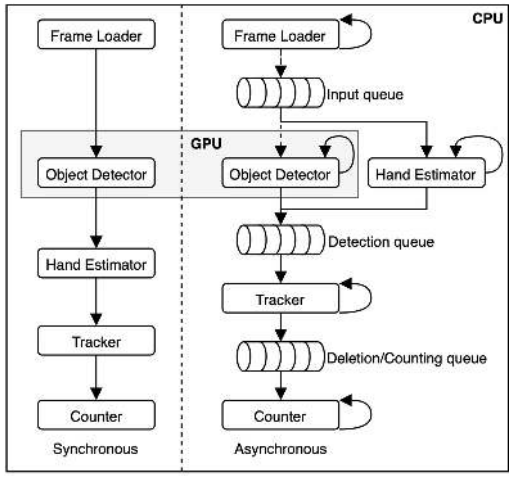
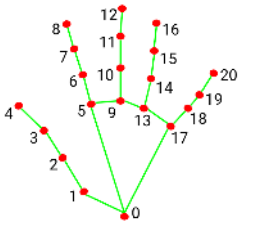


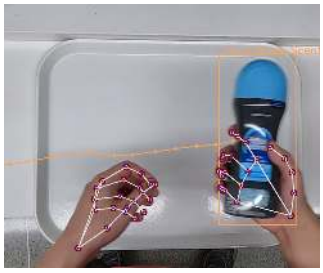
Figure 4. Synchronous and asynchronous processing pipelines are implemented. The synchronous pipeline is mainly used for developing and debugging the system, while the asynchronous pipeline is used for the final solution.

identification accuracy. Robust real-time hand perception is a challenging computer vision task, as hands often occlude themselves or each other (e.g., finger/palm occlusions and handshakes) and lack high contrast patterns. The hand estimation module is implemented using the MediaPipe framework [16] consisting of multiple models working together: a palm detection model that operates on the entire image and returns an oriented hand bounding box; and a hand landmark model that runs on the cropped image region defined by the palm detector and produces high-fidelity 3D hand keypoints.

First, a palm detector using SSD model [14] is trained



(a) 21 hand landmarks defined in MediaPipe [16].



(b) Example of hand-handling product. Certain hand landmarks points are located inside the object's bounding box.

Figure 5. Using hand landmarks to identify hand-handling products.

instead of a hand detector since estimating bounding boxes of rigid objects like palms and fists is significantly simpler than detecting hands with articulated fingers. In addition, as palms are smaller objects, the non-maximum suppression algorithm works well even for two-hand self-occlusion cases. Moreover, palms can be modeled using square bounding boxes, ignoring other aspect ratios and reducing the number of anchors by a factor of 3-5.

After the palm detection over the whole image, the subsequent hand landmark model performs precise keypoint localization of 21 hand-knuckle coordinates inside the detected hand regions via regression. The model learns a consistent internal hand pose representation and is robust even to partially visible hands and self-occlusions. In addition, the cropped palm regions can also be generated based on the hand landmarks identified in the previous frame in the hand estimation pipeline. Only when the landmark model could no longer identify hand presence is palm detection invoked to relocalize the hand. An example of the hand landmarks (coordinates) can be found in Fig. 5a.

The hand estimation module runs in parallel with the object detection module, as shown in Fig. 4. A pretrained weight provided by [16] is applied directly without any fine-tuning. The localization results from both the object detector 4.1 and hand estimator are used by the tracker to filter out non-hand-handling products. Examples of hand-handling products can also be found in Figs. 5b and **??**. On the remark, the hand estimation module is optionally implemented to cope with the requirement of the 2022 AI City Challenge Track 4 [17]. In other use cases, this module can be neglected.

Algorithm 1 Ray-casting algorithm

Require: $l \in \mathbf{L}$, hand landmark point
count $\leftarrow 0$
for each side $\in$ polygon **do**
  if *ray_intersects_segment*$(l, side)$ **then**
    count $\leftarrow$ count + 1
  end if
end for
if *is_odd(count)* **then**
  return *inside*
else
  return *outside*
end if

Algorithm 2 Hand-handling identification algorithm

Require: **L**, set of hand landmarks (42 points)
Require: b, object's bounding box
touches $\leftarrow 0$
for each $l \in \mathbf{L}$ **do**
  if *ray_casting_algorithm*(l, b) **then**
    touches $\leftarrow$ *touches* + 1
  end if
end for
if *touches* $\geq$ *min_touches* **then**
  return *True*
else
  return *False*
end if

Algorithm 3 State switching algorithm

Require: **T**, list of tracks
for each $t \in \mathbf{T}$ **do**
  if t_{state} is $new \wedge len(t) \geq 3$ **then**
    $t_{state} \leftarrow candidate$
  else if t_{state} is $candidate \wedge R(t) \geq 0$ **then**
    $t_{state} \leftarrow confirmed$
  else if t_{state} is $confirmed \wedge R(t) \leq 0$ **then**
    $t_{state} \leftarrow counting$
  else if t_{state} is $counting \wedge hand\_handling(t)$ **then**
    if $R(t) < 0 \vee t_{age} \geq max\_age$ **then**
      $t_{state} \leftarrow counted$
    else
      $t_{state} \leftarrow deleted$
    end if
  else if t_{state} is $counted \wedge R(t) > 0$ **then**
    $t_{state} \leftarrow exiting$
  else if t_{state} is $exiting \wedge t_{age} > max\_age$ **then**
    $t_{state} \leftarrow deleted$
  end if
end for

4.3. Tracker

Tracking is an essential step in improving object recognition in video processing. The tracking module assigns a unique id to a detected object when it enters the ROI. It associates the currently tracking vehicles with newly detected objects and maintains the unique id when moving through the camera. Hence, each object should only be identified

once.

Tracker. After obtaining the object detection results, i.e., bounding boxes, the SORT [1] method is used as the online multi-object tracker. Since the camera is mounted above the checkout counter and facing straight down, instead of a heavier one like DeepSORT [29], a simple tracker like SORT is memory sufficiently, reducing the overall processing speed. The SORT method uses Kalman Filter for motion prediction and the Hungarian algorithm for tracks assignment. The Kalman filter uses the unmatched detection results to initialize the tracking state as a new target and the matched detection results to update the existing target's tracking state. The state-space in each target is defined in the dimensional state space $(u, v, s, r, \dot{u}, \dot{v}, \dot{s})$, where u and v stand for the horizontal and vertical pixel 2D location of the center of the target. The scale s and r represent the scale (area) and the aspect ratio of the object's bounding box, respectively. Standard Kalman filter with constant velocity motion and linear observation model assign target object the tracklet k, which is used for following counting task. When assigning new detection results to exist targets, the shape of each target's bounding box is estimated by predicting its unique position in the current frame. The allocation cost matrix is then calculated as the Intersection-over-Union (IoU) distance between each detection and all predicted bounding boxes of the existing target. The IoU distance as the cost criteria for optimal matching originates from the SORT [1] where it has significantly demonstrated its fast and effective performance when used alongside the Kalman filter. Thus, for a detection D_i, its cost of being matched with any tracklet T_j is computed using IoU that is thresholded by IoU = 0.3 at minimum. The assignment is solved optimally using the Hungarian algorithm.

Tracking State. Track management controls the object's existence in the whole program. The extension of the original track management of SORT [1] is one of the significant contributions in the proposed DeepACO system. When an object travels through the camera's field of view, several stages can be defined as shown in Fig. 3. The tracking states are defined as in Table 2

The full state switching algorithm is described in Algorithm 3. The transition from the "Counting" to the "Counted" stage is the most important. Products are only counted if they are hand-handled. Through the observation suggested by Fig. 5b, a straightforward way to identify hand-handled products is by counting hand landmarks residing inside the detected bounding boxes. Hence, the ray-casting algorithm finds whether a hand landmark is inside or outside a simple polygon, i.e., a bounding box. The number of intersections for a ray passing from the polygon's exterior to any point, if odd, shows that the point lies inside the polygon. If it is even, the point lies outside the polygon; this test also works in three dimensions. Algorithms 1 and 2 shows the detail of the ray-casting and hand-handling identification algorithms respectively.

State	Description
New Track	A new track is created for the newly detected object. This track has a length of 1
Candidate	The new track has been successfully observed for several consecutive frames making it a candidate product. The object at this state is still located outside the ROI
Confirmed	The product moves inside the ROI. In this state, hand landmarks are used to identify whether the product is hand-handled
Counting	When the product touches the ROI border, it is marked as a counting product
Counted	The product is marked as counted if it satisfies the constraint defined in Alg. 2. Otherwise, it is marked for deletion
Exiting	The product has been counted and completely exited the ROI and is marked to be deleted
Deleted	The original deleting state of SORT [1]

Table 2. Tracklet states.

4.4. Counter

In this paper, to cope with AI City Challenge, a simple counter that records the product recognition timestamp is implemented. The time when the object is marked as "Confirmed" and is wholly entered into the ROI, the associate frame index is saved. Later, when the object is marked for "Counting", the output timestamp is calculated by dividing the saved frame index by the video frame rate, which is 60.

5. Experiments & Discussion

5.1. Implementation Details

The DeepACO system was implemented on a desktop consisting of an Intel Core i7-7700, an NVIDIA GeForce RTX 3090 24GB, and 32GB RAM. The whole system is implemented using a combination of OpenCV [18], PyTorch [19], and Mediapipe [16] libraries. Two different processing pipelines have been implemented: synchronous and asynchronous. As the name suggests, the synchronous processing pipeline runs each component in sequential order. Meanwhile, in asynchronous mode, each module is managed by a unique thread (4 CPU threads and 1 GPU thread). Between two modules, a queue is added to store the intermediate results, removing any bottlenecks. The asynchronous processing pipeline is used for the final solution of the 2022 AI City Challenge Track 4.

5.2. Evaluation Data & Metric

The experiments are performed using the test set provided by the 2022 AI City Challenge [17]. The test set consists of five videos where the camera is mounted above the checkout counter and facing straight down while a customer

Rank	Team ID	Score
1	16	1.0000
2	**94 (Ours)**	**0.4783**
		(Precision=0.4400, Recall=0.5238)
3	104	0.4545
4	165	0.4400
5	66	0.4314
6	76	0.4231
7	117	0.4167
8	4	0.4082
9	9	0.4000
10	55	0.4000

Table 3. The overall ranking on F1 score of the multi-class product counting & recognition for automated retail checkout in the 2022 AI City Challenge Track 4.

is pretending to perform a checkout action by "scanning" objects in front of the counter naturally.

The inference results are evaluated by the F1 score:

$$F_1 = \frac{TP}{TP + 0.5 \times (FP + FN)} \tag{2}$$

To compute the F1 score, a true-positive (TP) identification will be considered when an object was correctly identified within the region of interest, i.e., the object class was accurately determined, and the object was identified within the time that the object was over the white tray. A false-positive (FP) is an identified object that is not a TP identification. Finally, a false-negative (FN) identification is a ground-truth object that was not correctly identified.

5.3. Quantitative Evaluation

Our team achieved a very competitive final F1 score of 0.4745 (second place) in the 2022 AI City Challenge Track 4. The final ranking results of the challenge are shown in Table 3. Nevertheless, the recall is only 0.5238, i.e., only half of the test set ground truth has been identified correctly. It means some of the checkout actions are not defined as "scanning" naturally, or the recorded timestamps are incorrect.

5.4. Speed Performance

Beside F1 score, high-performance processing is also considered in this paper. Therefore, the DeepACO system has been tested on two NVIDIA GPUs: an RTX 3090 and an RTX A6000. Both synchronous and asynchronous processing pipelines have been evaluated. Different tries with different batch sizes and queue sizes are tested for the asynchronous processing pipeline. Tables 4, and 5 summary all the experimental results for Scaled-YOLOv4-P5, and -P7. The results show that implementing Scaled-YOLOv4-P5 with image resolution 448×448, batch size 16, and queue size 1 can satisfy real-time processing. The results also indicate that even with queue size 1, the asynchronous processing pipeline significantly boosts the performance of the DeepACO system by an average of 10 FPS, making it suitable for real-world applications.

6. Conclusion

This paper proposes a vision- and deep learning-based automatic checkout system, i.e., DeepACO. The system provides a robust solution for recognizing, localizing, and counting products during the checkout process. Also, DeepACO implements a convenience detect-and-track framework that can support a wide variety of detection and tracking models, extending the framework's capabilities to handle various real-world applications. Besides the main functionality, data generate pipelines are introduced to obtain many training images under different lighting conditions. The accuracy of DeepACO has been evaluated through the 2022 AI City Challenge Track 4 benchmark, achieving a top-2 rank with an F1 score of 0.4783. Furthermore, the system can reach 30 FPS on average, making it suitable for real-world applications.

References

[1] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft. Simple online and realtime tracking. In *IEEE Int. Conf. Image Process.*, pages 3464–3468, 2016. 1, 2, 6

[2] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao. YOLOv4: Optimal speed and accuracy of object detection. *arXiv*, 2020. 2, 4

[3] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. 2

[4] E. D. Cubuk, B. Zoph, J. Shlens, and Q. Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Adv. Neural Inform. Process. Syst.*, volume 33, pages 18613–18624, 2020. 4

[5] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Conf. Comput. Vis. Pattern Recog.*, volume 1, pages 886–893, 2005. 2

[6] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *Int. J. Comput. Vis.*, 111(1):98–136, 2015. 2

[7] R. Girshick. Fast R-CNN. In *Int. Conf. Comput. Vis.*, pages 1440–1448, 2015. 2

[8] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 580–587, 2014. 2

[9] S. V.-U. Ha, N. M. Chung, H. N. Phan, and C. T. Nguyen. TensorMoG: A tensor-driven gaussian mixture model with dynamic scene adaptation for background modelling. *Sensors*, 20(23), 2020. 2

[10] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf. Support vector machines. *IEEE Intell. Syst. Their Appl.*, 13(4):18–28, 1998. 2

q \ b	1	8	16	32	64
S	20.43 / 17.98	20.58 / 18.05	20.54 / 18.06	20.66 / 18.12	20.48 / 18.07
1	31.45 / 33.49	32.82 / 33.86	**32.97** / **34.03**	32.77 / 33.79	32.37 / 33.87
8	30.60 / 33.57	32.43 / 33.86	30.00 / 33.63	28.93 / 33.87	27.48 / 33.64
16	29.61 / 33.63	29.61 / 33.83	28.51 / 33.88	27.55 / 33.84	26.97 / 33.69
32	29.22 / 33.79	28.87 / 33.64	28.23 / 33.73	27.82 / 33.75	26.88 / 33.83

Table 4. Speed performance of Scaled-YOLOv4-P5 with image resolution 448×448 measured in FPS (RTX 3090 / RTX A6000). The first row indicates synchronous processing pipeline.

q \ b	1	8	16	32	64
S	16.73 / 16.35	16.93 / 16.49	16.93 / 16.41	16.98 / 16.62	16.69 / 16.46
1	28.40 / 32.90	30.50 / 32.50	**30.60** / 33.70	30.50 / 33.80	30.00 / 33.60
8	28.80 / 33.70	29.90 / 33.70	29.20 / **33.90**	26.70 / 33.70	28.00 / 33.70
16	27.80 / 33.90	27.8 / 33.90	27.90 / 33.80	27.50 / 33.80	26.50 / 33.90
32	27.50 / 32.90	27.50 / 33.90	27.20 / 34.00	27.10 / 33.60	27.10 / 33.70

Table 5. Speed performance of Scaled-YOLOv4-P7 with image resolution 896×896 measured in FPS (RTX 3090 / RTX A6000). The first row indicates synchronous processing pipeline.

[11] G. Jocher. ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference, Feb. 2022. 2

[12] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common objects in context. In *Eur. Conf. Comput. Vis.*, volume 8693, pages 740–755, 2014. 2

[13] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia. Path aggregation network for instance segmentation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 8759–8768, 2018. 4

[14] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. SSD: Single shot MultiBox detector. In *Eur. Conf. Comput. Vis.*, volume 9905, pages 21–37, 2016. 2, 4

[15] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vis.*, 60(2):91–110, 2004. 2

[16] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, F. Zhang, C.-L. Chang, M. G. Yong, J. Lee, W.-T. Chang, W. Hua, M. Georg, and M. Grundmann. Mediapipe: A framework for building perception pipelines. *arXiv:1906.08172*, 2019. 4, 5, 6

[17] M. Naphade, S. Wang, D. C. Anastasiu, Z. Tang, M.-C. Chang, Y. Yao, L. Zheng, M. S. Rahman, A. S., Q. Feng, V. Ablavsky, S. Sclaroff, and R. Chellappa. The 6th AI City Challenge. In *IEEE Conf. Comput. Vis. Pattern Recog. Worksh.*, 2022. 1, 2, 3, 4, 5, 6

[18] OpenCV. Open source computer vision library, 2015. 6

[19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Adv. Neural Inform. Process. Syst.*, pages 8024–8035, 2019. 6

[20] J. Peng, C. Xiao, and Y. Li. RP2K: A large-scale retail product dataset for fine-grained image classification. *arXiv:2006.12634*, 2021. 2

[21] J. Redmon and A. Farhadi. YOLO9000: Better, faster, stronger. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 6517–6525, 2017. 2

[22] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Adv. Neural Inform. Process. Syst.*, page 91–99, 2015. 2

[23] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *Int. J. Comput. Vis.*, 115(3):211–252, 2015. 2

[24] S. Tang, B. Andres, M. Andriluka, and B. Schiele. Subgraph decomposition for multi-target tracking. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 5033–5041, 2015. 2

[25] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao. Scaled-YOLOv4: Scaling cross stage partial network. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 13029–13038, 2021. 1, 2, 4

[26] X. Wang, E. Türetken, F. Fleuret, and P. Fua. Tracking interacting objects using intertwined flows. 38(11):2312–2326, 2016. 2

[27] Z. Wang, L. Zheng, Y. Liu, Y. Li, and S. Wang. Towards real-time multi-object tracking. In *Eur. Conf. Comput. Vis.*, page 107–122, 2020. 2

[28] X.-S. Wei, Q. Cui, L. Yang, P. Wang, and L. Liu. RPC: A large-scale retail product checkout dataset. *arXiv:1901.07249*, 2019. 2

[29] N. Wojke, A. Bewley, and D. Paulus. Simple online and realtime tracking with a deep association metric. In *IEEE Int. Conf. Image Process.*, pages 3645–3649, 2017. 2, 6

[30] Y. Yao, L. Zheng, X. Yang, M. Napthade, and T. Gedeon. Attribute descent: Simulating object-centric datasets on the content level and beyond. *arXiv preprint arXiv:2202.14034*, 2022. 3