DACNet: A Deep Automated Checkout Network with Selective Deblurring

Yichen Cai Aoran Jiao

University of Toronto, St George Campus
27 King's College Cir, Toronto, Ontario, Canada
{charles.cai, aoran.jiao}@mail.utoronto.ca

Abstract

Automated checkout systems have become increasingly popular as the state-of-the-art deep learning models are efficient and accurate enough for this to become a reality. However, challenges still exist due to the differences between synthetic training data and real-life products, the blurred product images captured during the checkout process, and discontinuous detections due to product similarities or tracking misses. This paper presents a robust deep learning YOLO-based pipeline, DACNet, that counters the above challenges. During training, data augmentation involving overlaying training images onto expected backgrounds creates a more diverse and accurate training dataset. When inferencing, selective deblurring is also incorporated to enhance the clarity of the items to be recognized while maintaining efficiency. And to improve accuracy further, we introduced a retrospective checking algorithm that analyzes previous detections and corrects any inaccuracies due to flickering detections or incorrect tracking results. Together, this pipeline ensures a network that produces reliable training results and high prediction accuracies even in complex retail environments with multiple items present. The proposed method has been submitted to 2023 AI City Challenge by NVIDIA and achieved a top-3 finish on the test set A with an F1-score of 0.8254. Our code is open sourced here: https://github.com/cycv5/AICityChallenge.

1. Introduction

Powered by the recent development of deep learning neural networks, automated checkout systems are rising in popularity. The systems provide a method for efficient and accurate scanning and counting of products that enables a seamless shopping experience. Ideally, when retail items are placed in a designated area or moved across a camera, images are captured and analyzed. And each item in the images is recognized and counted in order to produce the final price for the customer. However, training such a neural network requires a large and diverse dataset which could be difficult to collect and organize. Therefore, synthetic data generated from 3D scans of retail products can provide a well-organized dataset containing a large number of images within a relatively short timeframe.

Despite its efficiency, training on synthetic data poses multiple challenges to the real-life task of checking out customers. One of the main issues is the gap between the training images and real-life images captured during checkout. A deep learning network also needs to consider blurred images, unstable detections, and missed items due to the movements of the products.

This paper presents a robust pipeline (DACNet) for training a deep-learning neural network and applying it in a retail checkout setting. It uses the YOLOv8 [1, 2] algorithm as the main detector and classifier of the retail items, and StrongSORT [1, 3] as the tracking algorithm.

In short, the main contributions of this paper are as follows:

- **Experiments on Data Augmentation**: Synthetic images are copied to a background that resembles the checkout counter, and minor adjustments are made to the products so that they look as close to real-life products as possible. The results are compared with direct training on the original images, proving that data augmentation plays a crucial role in training.
- **Selective Deblurring**: Motion blur is one of the causes of inaccuracies during inferencing. A deblurring algorithm can reduce the blur considerably and improve the accuracy of classification. Due to the added complexity and processing time, only a few key frames from a video stream will be deblurred for a better prediction.
- **Retrospective rectification**: Neural network detections have their imperfections. Items could have duplicate detections and false detections. A good post-processing algorithm needs to be in place to filter out the noises. The proposed algorithm checks the previously recorded items, specifically their timestamp, on-screen time, and class, to determine if the current recording of an item is valid or not.

Together with accurate hand segmentations and region of interest (ROI) detection, the proposed method is robust and versatile. After training on 116,500 images from 116 classes of retail products, it is evaluated in the 2023 AI City Challenge Track 4. Compared to other state-of-the-art methods, DACNet obtained a top-3 finish with an F1-score of 0.8254.

The rest of the paper is organized as follows. Section 2 reviews the existing related works, Section 3 details the DACNet pipeline and algorithms, Section 4 reports the experimental results and discussions, and finally, Section 5 is the conclusion of this paper.

2. Related Works

We frame the AI City Track 4 challenge as a DTC (Detect-Track-Count) problem. This section introduces previous work done on the DTC problem and how we bridge the gap in the context of retail objects.

2.1. Object Detection

Object detection involves the identification and localization of target objects from background images or videos and is therefore key for this challenge. Traditional computer vision algorithms such as SIFT (Scale-Invariant Feature Transform), SURF (Speeded-Up Robust Features), and BRIEF (Binary Robust Independent Elementary Features) rely heavily on extracted feature descriptors and are used with traditional machine learning algorithms such as SVM and KNN. [4] With the rise of deep learning and the availability of larger datasets, deep learning-based computer vision models such as CNN tend to outperform traditional algorithms. [4, 5] Therefore, we focus on the review of these models such as YOLO (You Only Look Once), RetinaNet, R-CNN, and SSD.

Two popular approaches to object detection are one-stage and two-stage models. One-stage models such as YOLO, SSD, and EfficientDet require a single step in predicting the location and type of the object while two-stage models such as R-CNN use an RPN (Region Proposal Network) to generate the ROI (Region of Interest) in the first stage and refine them in the second stage with classifier and bounding box regression. [6] One-stage detectors tend to be faster but less accurate than two-stage detectors [7, 8].

R-CNN proposed by Girshick et al. [9] is a two-stage model combining region proposals with CNN features reaching an mAP of 53.3% in the VOC 2012 dataset. Fast R-CNN [10] and Faster-RCNN [11] further develop the idea by improving the speed and accuracy of predictions and proposing RPN to share full-image convolutional features and decrease the computation bottleneck.

YOLO is a one-stage, unified, real-time model proposed by Redmon et al. [12] that uses a single NN to predict bounding boxes and probabilities. There are many extensions and improvements on the original YOLO model such as YOLOv3 [13], YOLOv5 [14], YOLOv7 [15], YOLOX [16], and most recently, YOLOv8 by Ultralytics [2].

RetinaNet [17] and SSD [18] are two other popular one-stage models. Lin et al. trained the RetinaNet with a focal loss that retains the speed of one-stage detectors and surpasses the accuracy of two-stage detectors. SSD uses a similar single-shot detector with multi-scale feature maps and convolutional predictors.

2.2. Tracking

SOT (Single object tracking), MOT (Multiple object tracking), and OOT (Online object tracking) are common tracking tasks. SORT (Simple online real-time tracker) is the most suitable method for this challenge. SORT combines common tracking algorithms including Kalman Filter and Hungarian algorithm with a high FPS of 260 Hz [19]. DeepSORT [20] mitigates occlusion and missed frames of the tracked object with a deep association metric learned by a CNN. StrongSORT [3] further improves on the missing association and detection of DeepSORT by introducing an appearance-free link model and Gaussian-smoothed interpolation, achieving state-of-the-art for many public benchmarks. ByteTrack [21] is another MOT model which associates every detection box and uses similarities between tracklets to reduce false positives and negatives for low-score detection bounding boxes.

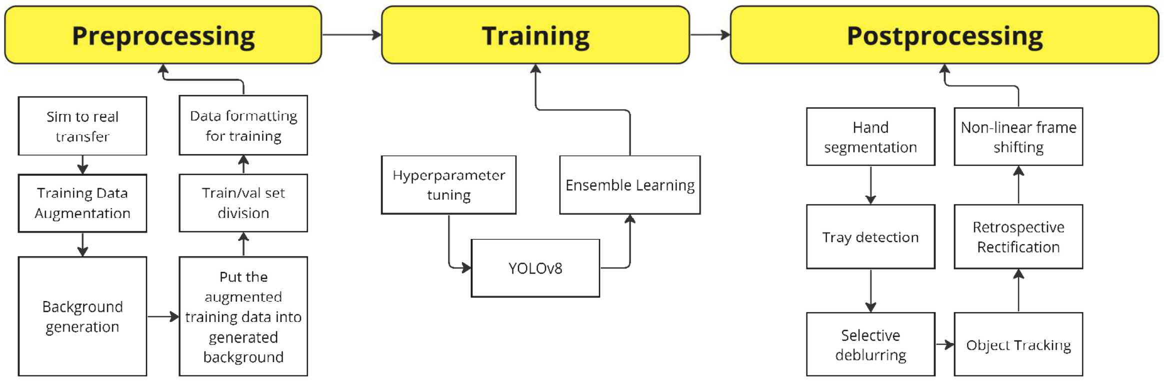

Figure 1: System Architecture Diagram

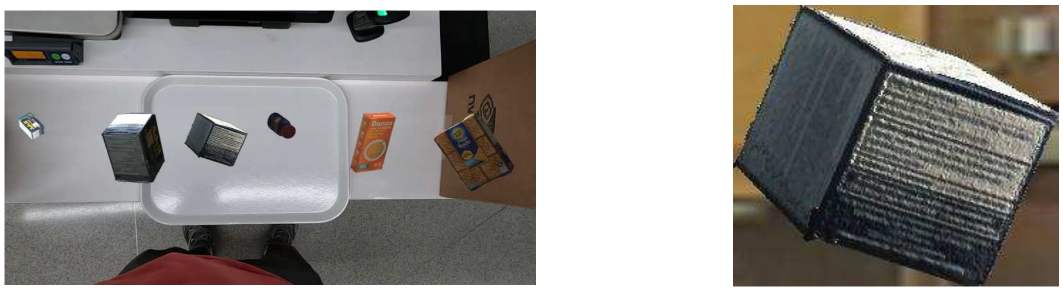
Figure 2: Augmented data VS original synthetic data

2.3. Hand Segmentation

To correctly detect and track objects, it is crucial to avoid false negatives due to occlusions. The retail objects are often occluded by hands during checkout. Therefore, segmenting and inpainting hands would improve the performance of our model. CNN is widely used in this context. Dadashzadeh et al. proposed a two-stage CNN-based HGR-Net which localizes the hand region and segmentation in the first stage and identifies the hand gesture in the second. [22]. There are also large-scale RGB-based datasets for robust hand segmentation and detection training such as Ego2Hands [23].

2.4. Deblurring

Video deblurring is a key step to improving detection accuracy. Motion deblurring and denoising methods consist of more traditional filtering techniques such as the Radon transform and directional filters [24], Haar Wavelet transform [25], super resolution techniques [26], as well as deep learning-based methods which use CNN and data-driven approaches to accumulate information between frames [27, 28].

2.5. Retail Object Recognition Datasets

Public datasets of retail object recognition fuel the development of better models and techniques. The AI City Challenge [29] provides 116,500 synthetic training data under different lighting and angle conditions from more than 100 types of retail items. Other datasets such as the RP2K [30] also consist of a large number of products and rich annotations.

2.6. Bridging the Gap

Our research aims to bridge the gap between theoretical models and their practical applications in this challenge. We evaluate, combine, and improve on the state-of-the-art models and techniques for the DTC problem.

3. DACNet

Our model, DACNet (Deep Automated Checkout Network) is trained and evaluated based on the architecture diagram shown in Figure 1. Our software stack consists of 3 components: preprocessing, training, and postprocessing.

In preprocessing, we address the main challenge of the domain gap between the synthetic training data and the real inference test videos. The simulation-to-real transfer aims to augment the retail products onto realistic generated background images under various lighting conditions, angles, and orientations.

We choose to deploy YOLOv8 as our main model. The training process takes in a YOLOv8 large model, tunes the hyperparameters with the Adam optimizer, and uses ensemble learning to better perform inference.

In postprocessing, we have 6 modules including hand segmentation and inpainting, tray detection, selective deblurring, object tracking, retrospective rectification, and non-linear frame shifting to transform the detection results to the output text file containing the object class ID and time of appearance.

3.1. Data Augmentation for Training

The training data is provided in 116,000 synthetic images generated by Unity. It is key to bridging the domain gap between the simulated training data and the real-world test data. First, to ensure the robustness and diversity of the products in the training data. We perform augmentation by randomly changing the brightness, reflectivity (to simulate different lighting conditions), size, and orientation of the input items, as Figure 2 shows.

Inspired by the One Research Framework [31], random backgrounds with rotations, horizontal, and vertical reflections are generated to fit multiple augmented items onto the same image. This bridges the domain gap between synthetic and real-world data and reduces the total number of training images and potential training time. Random items are placed onto the central region of the background with potential overlaps in between to simulate the occlusion in real life. The coordinates and size of the bounding boxes are extracted from the segmentation masks provided by the original dataset.

The randomness and diversity of the training data augmentation mean we could train multiple models with different versions of the augmented data and perform ensemble learning with different models.

Lastly, the augmented data is divided into training and validation sets and converted into the YOLO data format for training.

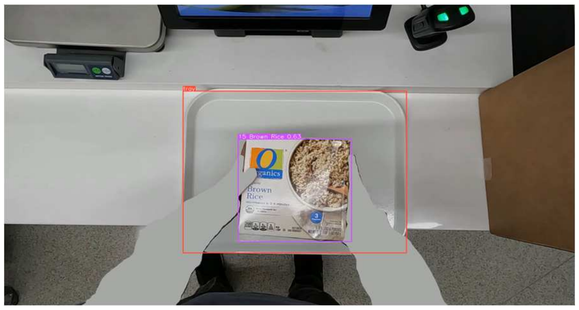

Figure 3: Hands are masked by a color similar to the background.

3.2. Training the Model

The training hyperparameters of our model including learning rate, batch size, and dropouts are fine-tuned by gradient-based stochastic optimizer - Adam [32]. Each model is trained until the validation loss converges and the training loss plateaus. The independently trained models with different variations of augmentation are then combined using ensembled learning and averaging the output confidence score as the final result.

3.3. Preprocessing before Inferencing

Before processing a frame from an input video stream through the backbone classifier, two preprocessing steps are taken – hand segmentation and ROI detection.

When a customer checks out an item, their hands may inadvertently come into contact with the products. This could cause additional noises that impact the prediction accuracy. To mitigate this, we first remove the hands by masking them with a color that is similar to the background. This part of the algorithm is implemented using EgoHOS [33], an egocentric hand segmentation algorithm based on MMSegmentation [34]. EgoHOS is trained on a large set of images containing two hands and is able to output a mask of the hands. In our model, a new class, namely HandSegmentor is created to handle the hand segmentation module. When initialized, it reads the pre-trained model on hand segmentation [33] and prepares the MMSegmentation network for inferencing. Each frame will be processed by the HandSegmentor and output a mask ($Z^{H\times W}$) that represents the hands area with non-zero entries. Then all the non-zero entries are collected and on the original input images, pixels on those indices will be replaced by a predefined color that is close to the background (e.g., [168,168,168]). The result of the hand segmentation will be a tensor that has the same shape as the original ($Z^{H\times W\times 3}$).

The next step is to recognize the region of interest (ROI), which is crucial to accurately count only the products that are being checked out. In 2023 AI City Challenge Track 4, this area is defined by a white tray. The proposed method uses only efficient mathematical methods provided by OpenCV [35] and is applicable to any ROI detection task with a relatively clear boundary. Initially, a default ROI is input to the algorithm as the fail-safe result. Then the detection starts with turning the image into grayscale and applying Gaussian Blur to it. This removes a significant number of noises created by lighting and different shades of colors. A Canny tool is used next to find the edges of the image. Then, an image gradient is calculated using Scharr derivatives to highlight the edges for the final step, flood fill [35]. OpenCV provides a flood fill algorithm that fills the geometry (up until an edge is met) from an input coordinate and returns the resulting rectangle coordinates, representing the ROI. The choice for the input coordinate is based on real-life scenarios, it needs to be within the ROI itself. In our algorithm, we decided to go with the middle of the input video frame first and move to a slightly different location for a second try if the first attempt is unsuccessful (the bounding box is unreasonably small/large given the input image). If the second attempt is also unsuccessful, we would output the default estimate of the ROI. Note that it is unnecessary to do ROI detection every frame as the ROI rarely changes. Thus, we only update the ROI every frame until a non-default, calculated bounding box is found, and after that, it will update every 5 seconds with any new calculated result.

3.4. Image Deblurring

Motion blur is another leading cause of wrong or missed detections of products. Depending on how a product is moved into and out of the camera view, some images captured could be blurry. A fast and powerful deblurring algorithm can help significantly.

NAFNet [36] is one of the state-of-the-art deep learning algorithms that do image restoration efficiently. It eliminates the need for non-linear activation functions in a network. Instead, they are replaced with multiplication or

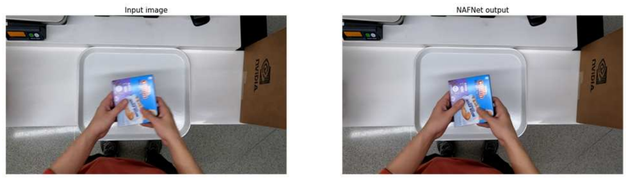

Figure 4: A comparison between the original video frame and the deblurred version by NAFNet.

simply removed. The architecture is mainly a convolutional neural network, with a simplified channel-based attention mechanism and the Simple Gate (an operation that replaces regular non-linear activations).

We used a pre-trained image deblurring model on the REDS (Realistic and Dynamic Scenes) dataset [37] for our purpose. And applying the NAFNet before sending the images to the classifiers enables it to recognize some items that were previously unrecognizable. Again, we need to keep in mind that deblurring takes a large amount of time and space complexity despite the efficient algorithm by NAFNet. Therefore, a selective approach is used. Every time a new detection is found by the backbone algorithms that is within the ROI, the deblurring function will be called once. This ensures that the number of times the deblurring function is called is O(n) where n is the number of items being scanned. This approach keeps the majority of the frames at a low processing time while only around 1% of the frames will be deblurred. Once the deblurring is applied to a frame, its classification result will be weighted higher in the resulting dictionary (introduced in Section 3.5) due to its clarity. This will impact the final calculations in favor of the classification made with a deblurred image.

3.5. Backbone Algorithms

The backbone of our system consists of two major parts, a YOLOv8 classifier, and a StrongSORT tracking algorithm [3]. A video frame first goes into our trained YOLOv8 model, as with all the YOLO models, it only takes one forward pass to determine the locations and types of everything it sees in the frame. For our use case, 116 categories for classification, we decided to utilize a large YOLOv8 variant. With a single Tesla T4 GPU, the inference speed for YOLOv8l alone is around 50ms. Compared to the classic YOLOv5 [38], The v8 model performs 7.96% better in terms of mAP^{50-95} with a similar processing speed (both with the large variant and on the COCO dataset). One of the benefits of using YOLOv8 is its state-of-the-art architecture [2]. For instance, YOLOv8 has anchor-free detections. It predicts the objects directly on the center points of them instead of from known anchor boxes. Those anchor boxes could be misleading, especially used on custom datasets like ours. Additionally, there are a few structural changes to make this new YOLO model efficient and accurate.

Once the YOLOv8 network has its prediction results, it will be fed into the StrongSORT algorithm along with the original image. StrongSORT is an improved DeepSORT algorithm, with around 8.5% improvement in IDF1 on the MOT20 dataset with similar processing speed. One of the main improvements for StrongSORT is its ability to recognize and tackle movements of objects. StrongSORT first uses an ECC (enhanced correlation coefficient maximization) to account for motioned noise caused by movements. Then a modified Kalman filter that emphasizes non-linear motions is used to calculate the weightings during each update across frames. Lastly, for object association, StrongSORT directly includes the motion information (in addition to appearance) for more accurate tracking. Overall, it results in a good tracking algorithm for our use case.

With the two backbone algorithms, we are able to generate a result dictionary in the format of:

{unique_index:(timestamp_in_frame, {class: count})}.

Within the dictionary, keys (unique_index) represent the unique indices that are assigned to each tracked object in the input video of a checkout scenario. The values in this dictionary are tuples. The first element in the tuple is the time stamp (timestamp_in_frame) of the first appearance inside the ROI of this object indicated by the unique_index. The second element is again a dictionary object where the keys are predicted classes of this object and the values are the number of frames in which this object is classified as that class. Classes with higher frame counts indicate a higher probability that this particular object belongs to this class. Classification on deblurred images will receive a boosted frame count (e.g., 1 deblurred frame classification counts as 5 regular frames). Note that there could be multiple entries in this inner dictionary as there could be incorrect classifications.

3.6. Retrospective Rectification and Output

With the resulting dictionary, a final algorithm is applied to it for a better and more consistent result. This algorithm can be separated into 2 stages: backward correction and forward recording. Both backward and forward algorithms process the resulting dictionary from the previous step hence it does not need to look at each frame of the input video, reducing the complexity significantly. The dictionary ordering is kept in a separate list object. A "detection" is defined by an entry in the resulting dictionary. It has a unique index as its key, one timestamp, and possibly multiple classes and frame count for each class. In our rectification algorithm (backward and forward algorithms), for most parts, we only look at the class with the maximum frame count in each detection.

The backward stage starts with the last detections in the resulting dictionary going backward. The idea is for an unstable detection, the tracker could lose track of the item at some point during the time that the item is within ROI. The item could be briefly categorized as something else for a brief time and went back to its correct classification, but the tracker could think of it as a new, unseen item. For each new detection, we would retrospectively check the previous detections. When previous detections within a certain range (and within a certain timestamp) as the exact

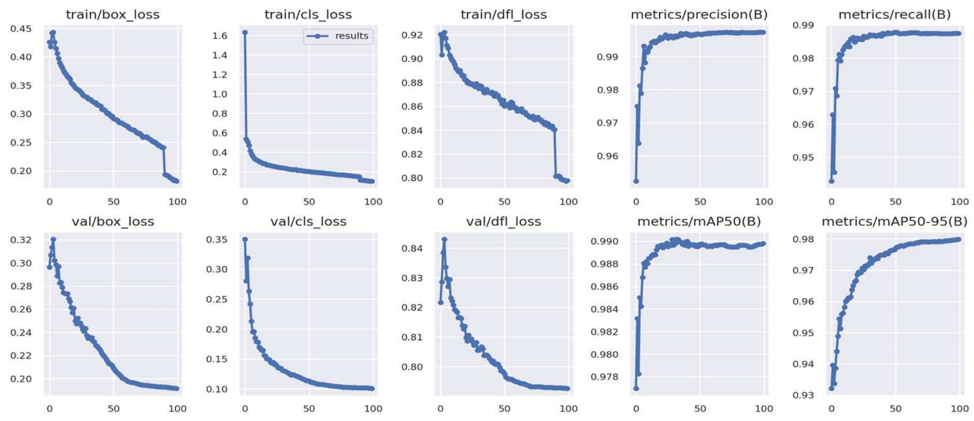

Figure 5: Loss, Precision, Recall, and mAP curves for YOLOv8 training.

same classification, we would consider them as one entry. The algorithm would add the frame count of the later detections to the previous detections and set the later detections' frame count to 0. See Algorithm 1 for details.

The next step is the forward recording (Algorithm 2), which will be the step where the result of the checkout is generated. To avoid extraneous detections, there will be a threshold calculated for a detection to be valid. This threshold is based on the average speed (in frames) of all the items moving across the field of view in the input video, upper and lower bounds are set (arbitrarily and subject to finetuning) to ensure a more generalizable result. The average speed is calculated per video or checkout session. Checkout sessions in real life can easily be segmented by payments from the customers. With the forward algorithm, it loops through the result dictionary, and with any significant detections (frame count larger than the threshold defined above), it will check if the same item is recorded already, determined by the previous recorded items and timestamp. If the current item has not yet been recorded, it will add a new entry to the final result of this checkout session.

Algorithm 1: Backward Checking

```
1   for i from (length_result_dict – 1) to 0:
2       cur_tstamp, cls_dict = timestamp and classes from
            current detection i
3       max_cls = max frame count class in cls_dict
4       for 3 previous detections in result_dict (if exists):
5           prev_tstamp, prev_cls_dict = timestamp and
                classes from previous detection
6           prev_max_cls = max frame count class in
                prev_cls_dict
7           if max_cls == prev_max_cls and (cur_tstamp –
                prev_tstamp – prev_cls_dict[pre_max_cls]) <
                threshold:
8               prev_cls_dict[pre_max_cls] +=
                    cls_dict[max_cls]
9               cls_dict[max_cls] = 0
10              break
```

Algorithm 2: Forward Recording

```
1   for i from 0 to (length_result_dict – 1):
2       cur_tstamp, cls_dict = timestamp and classes from
            current detection
3       max_cls = max frame count class in cls_dict
4       if cls_dict[max_cls] > threshold:
5           if max_clss == previous recorded class:
6               check the timestamp difference, write to output
                    file if it is larger than 18 frames else skip
7           else:
8               write to output file
9           update previous recorded class according to the
    current detection
```

Optionally, one more step can be added before writing to the resulting file. In our use case, we deliberately added paddings to the ROI to have a more generous detection area because we cannot expect customers to always fit the products exactly inside the ROI area within the camera view. Although that gives fewer misses for detections, it does cause the recorded timestamp to be earlier than it should be. This does not affect our rectification algorithms as they only care about the timestamp differentials instead of the absolute timings of the product recognitions. But for a better, more accurate timestamp, a non-linear shift can be applied to the final result. Since the resulting dictionary from the backbone algorithms has the frame count from each detection, we know the speed for each item going across the ROI. Based on this speed, we added a timestamp padding to the result, more specifically, around 1/5 of the time it spends within the ROI.

4. Experiments & Discussion

4.1. Setup

The models are trained on the hardware platform with 2 NVIDIA GeForce RTX 3090 GPUs with 24 GB dedicated

RAM each. The CPU of the platform is Intel (R) Core (TM) i9-9980XE @ 3.00GHz.

4.2. Experiments and Result

The preliminary result of the proposed DACNet is tested on test set A of the 2023 AI City Challenge Track 4. It is evaluated in F1-scores. Each detection from the input test video needs two pieces of information: class and timestamp (in the input video). True positives (TP) are detections with the correct classes and timestamps. False positives (FP) are detections of products recorded in the final output yet not actually present in the test video. Lastly, false negatives (FN) are missing detections of items that appeared in the video.

$$F1-score = \frac{2\times TP}{2\times TP + FP + FN} \qquad (1)$$

We first conducted a simple experiment on the original synthetic dataset provided by the 2023 AI City Challenge with the segmentation masks. We aimed to train a YOLOv8 network for instance segmentation. However, the overall result was not satisfactory (F1-score < 0.1). The instance segmentation task added too much complexity in terms of localization which was unnecessary given our final objective. Also, the original synthetic images have two major disadvantages: out of context and out of scale. The synthetic training data has random backgrounds which caused difficulties for the model when it was tested with the test set. Furthermore, the sizes of products were ignored as they were all stretched to 640*640 for our training.

The above problems can be addressed through our data augmentation step (Section 3.1). With the augmented training data, the result improves drastically. Without the deblurring and the retrospective rectification step, the test accuracy is already better, with an F1-score above 0.5. Given this promising result, we started training for more epochs. Convergence is reached at around 100 epochs as shown in Figure 5, and we decided to employ an early-stopping strategy at 100 epochs. Adding the full pipeline to the test, the model achieved an F1-score of 0.8254, which is the 3rd best result in the public leaderboard of 2023 AI City Challenge Track 4 as shown in Table 1. The progression of our experimental result is recorded in Table 2, note that some recorded F1-scores are based on partial test sets.

Table 1: Top teams in AI City Challenge 2023 Track 4

Rank	Team ID	F1-Score
1	33	0.9792
2	21	0.9787
3 (*)	**13**	**0.8254**
4	1	0.8177
5	23	0.7684
6	200	0.6571

Table 2: Effectiveness of various components of DACNet

Method	F1-Score
YOLOv8 trained on original dataset	<0.1
YOLOv8 trained on the augmented dataset	0.52
YOLOv8 trained on augmented dataset + Hand Segmentation	0.77
YOLOv8 trained on augmented dataset + Hand Segmentation + Selective Deblurring	0.81
YOLOv8 trained on augmented dataset + Hand Segmentation + Selective Deblurring + Retrospective Rectification	**0.8254**

4.3. Discussion

The 3 main contributions proposed in this paper considerably improve the overall result of the pipeline. In particular, the data augmentation enhances the accuracy of the system by bridging the gap between standalone synthetic data and real-life footage from a checkout counter. We would also highlight the importance of maintaining relative sizes between objects during training. In addition, the rectification algorithm refines the object detections by leveraging the temporal relationships and various heuristics, leading to a substantial improvement in the final result.

Despite the promising F1-score, it is worth noting that the training data only consists of synthetic images. As the project is deployed to a real-life platform, more data can be collected during the process hence more accurate datasets can be collected and trained on. Additionally, a possible extension to this project would be integration with barcode scanning to counter the ever-changing packaging and a vast number of products in a retail store.

5. Conclusion

In this paper, we presented a robust pipeline for an automated checkout system that utilizes the state-of-the-art deep learning neural networks like YOLOv8 and StrongSORT. The pipeline also addresses the challenges of training with synthetic data, motion blur, and unstable detection results. It uses some combinations of existing methods as well as some novel algorithms to tackle the above challenges, resulting in good prediction accuracy. That ensures a smooth checkout experience for the customers. And with our DACNet achieving a top-3 finish in the 2023 AI City Challenge Track 4, it proves that it is a strong foundation for a full-scale real-life checkout system that could revolutionize the shopping experience for consumers.

References

[1] M. Broström, "Real-time multi-object tracking and segmentation using Yolov8 with StrongSORT and OSNet (Version 8.0)," [Computer software], DOI: https://doi.org/https://zenodo.org/record/7629840

[2] *YOLO by Ultralytics*. (Version 8.0.0), Ultralytics. [Online]. Available: https://github.com/ultralytics/ultralytics

[3] Y. Du et al., "StrongSORT: Make DeepSORT Great Again," arXiv:2202.13514 [cs], Feb. 2023, Accessed: Mar. 01, 2023. [Online]. Available: https://arxiv.org/abs/2202.13514#

[4] N. O'Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. V. Hernandez, L. Krpalkova, D. Riordan, and J. Walsh, "Deep learning vs. Traditional Computer Vision," *Advances in Intelligent Systems and Computing*, pp. 128–144, 2019.

[5] Lee, A.: Comparing deep neural networks and traditional vision algorithms in mobile robotics. Swarthmore University, 2015.

[6] Lohia, A.; Kadam, K.D.; Joshi, R.R.; Bongale, D.A.M. Bibliometric Analysis of One-Stage and Two-Stage Object Detection. Libr.Philos. Pract. 2021, 4910, 34

[7] P. Soviany and R. T. Ionescu, "Optimizing the trade-off between single-stage and two-stage deep object detectors using image difficulty prediction," *2018 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, 2018.

[8] X. Lu, Q. Li, B. Li, and J. Yan, "Mimicdet: Bridging the gap between one-stage and two-stage object detection," *Computer Vision – ECCV 2020*, pp. 541–557, 2020.

[9] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[10] R. Girshick, "Fast R-CNN," *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015.

[11] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017.

[12] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[13] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767, 2018.

[14] X. Zhu, S. Lyu, X. Wang, and Q. Zhao, "TPH-yolov5: Improved yolov5 based on transformer prediction head for object detection on drone-captured scenarios," *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, 2021.

[15] Wang, C. Y., Boschkovskiy, A., and Liao, H. Y. M. (2022). YOLOv7: Trainable bagof-freebies sets new state-of-the-art for real-time object detectors. arXiv preprint arXiv:2207.02696. doi: 10.48550/arXiv.2207.02696

[16] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. YOLOX: Exceeding YOLO series in 2021. arXiv preprint arXiv:2107.08430, 2021.

[17] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal loss for dense object detection," *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017.

[18] Liu W, Anguelov D, Erhan D, Szegedy C, Reed S, Fu C-Y, Berg AC (2016) SSD: single shot MultiBox detector. In: Leibe B, Matas J, Sebe N, Welling M (eds) Computer vision—ECCV 2016. Springer, Cham, pp 21–37. https://doi.org/10.1007/978-3-319-46448-0_2.

[19] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," *2016 IEEE International Conference on Image Processing (ICIP)*, 2016.

[20] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a Deep Association metric," *2017 IEEE International Conference on Image Processing (ICIP)*, 2017.

[21] Y. Zhang, P. Sun, Y. Jiang, D. Yu, F. Weng, Z. Yuan, P. Luo, W. Liu, and X. Wang, "ByteTrack: Multi-object tracking by associating every detection box," *Lecture Notes in Computer Science*, pp. 1–21, 2022.

[22] A. Dadashzadeh, A. T. Targhi, M. Tahmasbi, and M. Mirmehdi, "HGR-Net: A fusion network for hand gesture segmentation and recognition," *IET Computer Vision*, vol. 13, no. 8, pp. 700–707, 2019.

[23] Fanqing Lin, Brian Price, and Tony Martinez. Ego2hands: A dataset for egocentric two-hand segmentation and detection. In arXiv:2011.07252, 2020.

[24] L. Zhong, S. Cho, D. Metaxas, S. Paris and J. Wang, "Handling Noise in Single Image Deblurring Using Directional Filters," *2013 IEEE Conference on Computer Vision and Pattern Recognition, Portland, OR, USA, 2013*, pp. 612-619, doi: 10.1109/CVPR.2013.85.

[25] Sada M M, Mahesh M G. Image deblurring techniques–a detail review. Int. J. Sci. Res. Sci. Eng. Technol, 2018, 4: 15

[26] T. Yamaguchi, H. Fukuda, R. Furukawa, H. Kawasaki, and P. Sturm, "Video Deblurring and super-resolution technique for multiple moving objects," *Computer Vision – ACCV 2010, pp. 127–140, 2011.*

[27] Su S, Delbracio M, Wang J et al (2017) Deep video-deblurring. *2017 IEEE Conference on Computer Vision and Pattern Recognition*: pp. 1279–1288

[28] K. Zhang, W. Ren, W. Luo, W.-S. Lai, B. Stenger, M.-H. Yang, and H. Li, "Deep image deblurring: A survey" *International Journal of Computer Vision*, pp. 2103–2130, 2022.

[29] M. Naphade, S. Wang, D. C. Anastasiu, Z. Tang, M.-C. Chang, Y. Yao, L. Zheng, M. Shaiqur Rahman, A. Venkatachalapathy, A. Sharma, Q. Feng, V. Ablavsky, S. Sclaroff, P. Chakraborty, A. Li, S. Li, and R. Chellappa, "The 6th AI City Challenge," *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2022.

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

[31] Pham, L. H. (2022). One: One Research Framework. GitHub. https://github.com/phlong3105/one

[32] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[33] L. Zhang, S. Zhou, S. Stent, and J. Shi, "Fine-Grained Egocentric Hand-Object Segmentation: Dataset, Model, and Applications," arXiv:2208.03826 [cs], Aug. 2022, Accessed: Apr. 07, 2023. [Online]. Available: https://arxiv.org/abs/2208.03826

[34] MMSegmentation Contributors. (2020). OpenMMLab Semantic Segmentation Toolbox and Benchmark [Computer software]. https://github.com/open-mmlab/mmsegmentation
[35] OpenCV. Open source computer vision library, 2015.
[36] L. Chen, X. Chu, X. Zhang, and J. Sun, "Simple Baselines for Image Restoration," arXiv:2204.04676 [cs], Aug. 2022, Accessed: Mar. 07, 2023. [Online]. Available: https://arxiv.org/abs/2204.04676v4
[37] N. Challenge, "REDS," *REalistic and Diverse Scenes dataset realistic and dynamic scenes*.
[38] G. Jocher, "ultralytics/yolov5," GitHub, Aug. 21, 2020. https://github.com/ultralytics/yolov5