REAL-TIME OBJECT DETECTION AND TRACKING

Samuele Sermisoni(s212060) *Ruixin Chen(s210312)*
Xianhao Liu(s202724) *Carlos Barragan(s211821)*

https://github.com/Michealxh-L/realtime-object-detection-and-tracking.git

Technical University of Denmark

ABSTRACT

In this paper, we will introduce our project whose goal is to implement an algorithm able to perform object identification and tracking in real-time using as hardware a single board computer such as the Nvidia Jetson Nano. Specifically, we were asked to recognize coke and beer cans rolling on the floor. To do so, we collected the images using a webcam; we augmented our dataset using different geometrical transformations and image blurring; we trained two different deep learning nets (Faster R-CNN and YOLO v5) and made a comparison and evaluation. Then a tracking algorithm based on the Euclidean distance was implemented. Our results are very good with accuracy over 99% and a frame rate over 50Hz.

***Index Terms*— Object Detection, Object Tracking, YOLO, Faster R-CNN, Deep Learning**

1. INTRODUCTION

Nowadays the use of deep learning algorithms is spreading a lot also in the industrial sector, thanks also to the increase in the automation of industrial plants and Industry 4.0. The main problem with these algorithms, however, is that they require high-performance hardware components to function properly (latest generation CPU and GPU). The semiconductor crisis that hit the two-year period 2020-2021 and which is expected to also characterize the whole of 2022 has therefore led to the need to implement lighter algorithms, capable of operating even on low-power hardware.

According to an analysis conducted by McKinsey&Company "feed forward neural networks and convolutional neural networks together have the potential to create between $3.5 trillion and $5.8 trillion in value annually across nine business functions in 19 industries". A large chunk of this value comes from new predictive maintenance technologies ($ 0.5 - $ 0.7 trillion). If through a detailed analysis it is possible to predict a failure or to optimize the maintenance of an industrial plant, the risk of blocking production (with the consequent economic damage) is minimized. An example of "predictive maintenance" is the analysis of the position of objects on conveyor belts. If an object is wrongly rotated it may cause an obstruction with the consequent block of the production line, but if the rotational problem is detected it can be easily fixed by an operator or a machine. These reasons motivated us to implement an algorithm able to perform object identification and tracking in real time capable of running on low-power hardware.

To get intuitive feedback, we applied the algorithm on the scene of detecting and tracking the cans of beer and coke. A video of cans of coke and beer rolling and moving in the floor is recorded as the data to support our training. Two popular real-time object detection algorithms are selected to solve the detection problem. Distance based algorithm is applied for tracking.

This paper is organized as follows: Section 1 introduces the motivation of this project and this paper. Section 2 illustrates the data preparation work of this project that includes how we collected the data as well as how we cleaned and processed the data. Section 3 illustrates why we chose the Faster R-CNN and YOLO v5s algorithms for object detection, how we implemented them, and how we made the comparison. Section 4 illustrates how we implemented object tracking. And Section 5 is the conclusion from the project.

2. DATA PREPARATION

Deep learning models require massive data to train the weights as benchmark models such as MaskRCNN, YOLO, and MobileNet, trained on existing large natural image datasets like ImageNet and COCO. A pre-trained neural network may detect a can of drink from an image, but it will definitely not differentiate between the type or even the beverage brand if it has never been taught to do so. A customised labelled dataset will be a good start for this specific task of training a beer and coke detector and classifier. With data augmentation, the accuracy and robustness of the models can be improved.

2.1. Data Collection

A webcam is utilised to capture the dataset indoor, fixed perpendicular to the ground to take an overhead view of the rolling cans. Two image sequences were recorded as training and test set separately. The movement of the cans in the test set presents faster, more randomised, and thus its scene seems more chaotic. The images are in RGB (three channels) and have 640×480 resolution.

For the training for object detection in supervised learning, it is important to have a good teacher—to annotate the existing objects with their corresponding class labels and compact bounding boxes to indicate their location in the frame. This job was done on CVAT, an open-source, online, interactive image annotation tool for computer vision, where we chose frames within a random time range for annotation. In total, 1609 frames are used to produce the original dataset.

The images in the training set are randomly split into a training and validation set in a ratio $4 : 1$. The random split seed remains the same for all training. The validation set can later tune the learning rate and determine early stopping.

2.2. Data Augmentation

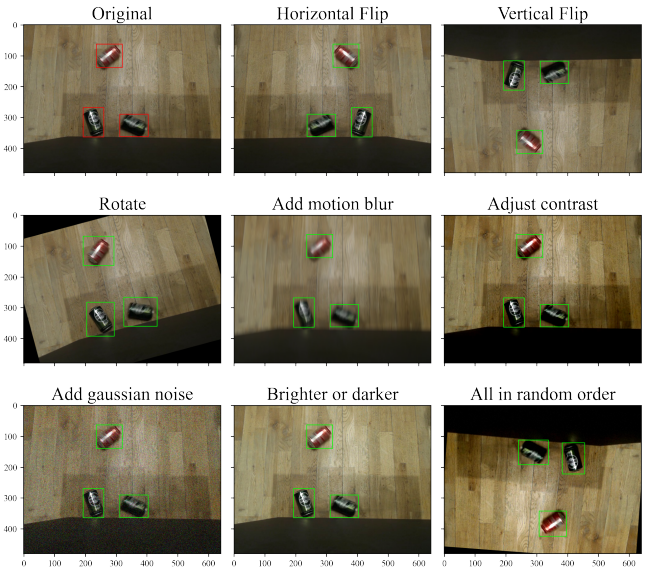

Fig. 1. Data augmentation with different methods in an example frame. Red BBoxes are from the original annotations. Green BBoxes are the transformed.

The overfitting problem arises when training a Faster R-CNN network with limited training data. The additional augmentation was done to images and annotation boxes, which included rotations, flips, shearing, colour channel shifts and motion blurring, see in Fig. 1 in a shuffled order.

Various methods can release the overfitting problem, among which data augmentation is one of the most direct and efficient ways. The object's pose can be manipulated into all possible directions by flipping, rotating, and shearing the image. However, rotating or shearing too much leads to the new bounding box covering much incorrectly targeted space. A small angle can ensure the compactness of the bounding boxes while changing the direction of objects. Multiplication to all channels and contrast adjustment can simulate different illumination. Injecting the right amount of noise is also a fundamental tool for data augmentation.

Particularly, motion blur is introduced into our data augmentation to imitate the object moving fast or the effect under a low frame rate camera. As the latest research, motion blur can enhance the robustness in object detection and captioning[1]. However, there is still a problem applying the motion blur effect in the whole frame, and it looks like the camera is moving but not the rolling cans on the floor; instead, we should blur the labelled objects only.

3. OBJECT DETECTION

The first goal of this project is make the object detection in real-time. To choose the best one from different models, we need to evaluate the performance of different model in standardized rules. We analyze the performance in the aspects of accuracy, speed, and training time. In order to compare the models, we train and test the models in the same environment. Table.1 shows the environment that we run the algorithm. We choose YOLO v5 and Faster R-CNN as models for comparison. Because YOLO v5 is currently the most popular real-time object detection algorithm, while the Faster R-CNN has higher accuracy and the ability to work in real time.

GPU	RTX 3070 Mobile
RAM(GPU)	8G
RAM	16G
Operating System	Ubuntu 18.04 64bit
Computational Performance	16.59 TFLOPS

Table 1. Inference Environment.

3.1. Faster R-CNN Algorithm

In the field of object detection, Faster R-CNN (FRCNN for short) was originally published in NIPS 2015. It is the third iteration in the R-CNN family[2]—which had Ross B. Girshick as author and co-author. Faster R-CNN has integrated feature extraction, proposal extraction, bounding box regression(rect refine) and classification into a network. The comprehensive performance is greatly improved, especially in the detection speed.

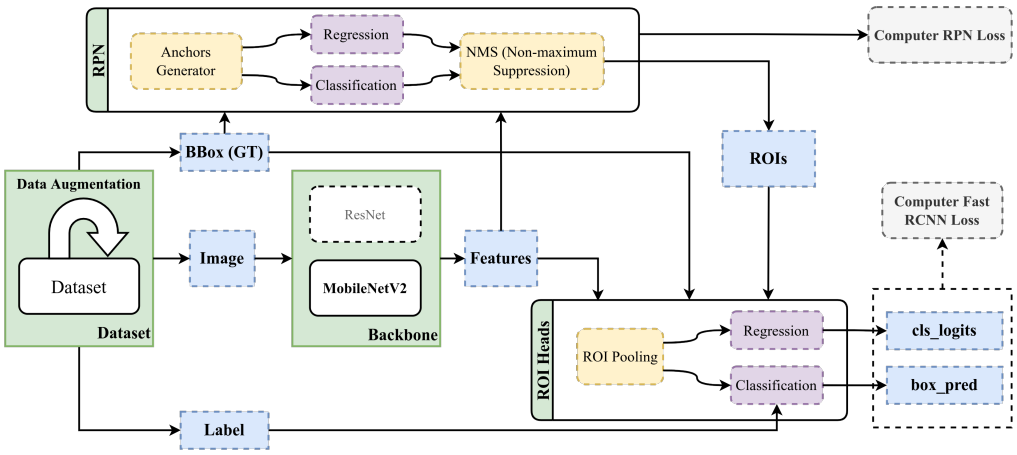

Fig. 2. Faster R-CNN architecture.

We choose MobileNetV2 as the backbone of the Faster R-CNN model, which is designed to run deep networks on personal mobile devices, thanks to its depthwise separable convolutions and inverted residual structure, where the shortcut connections are between the thin bottleneck layers. In our model, MobileNetV2 have been pre-trained with the natural image dataset ImageNet because neural networks can exhibit transfer learning [3], and then fine-tuned with our training set.

Faster R-CNN uses a Region proposal network (RPN) to generate proposals with various scales and ratios. Since in our dataset, the cans' size and ratio are relatively fixed, even when they roll fast and enlarge in the direction of moving due to motion blur. We set the size of the anchor to 50 and the aspect rations to 0.5, 1.0 and 1.5. We tried to freeze the backbone for five epochs before training but got little improvement in accuracy. We found out the FRCNN with a minibatch of four has a better performance than the one with a batch size of eight, similar to the performance of the default setup (the default minibatch size is 2), but have a shorter training time. Taking both real-time performance and accuracy into account, we adjusted the minimum size of the image to be rescaled before feeding it to the backbone. Empirically we set it to be 300 and 480, where 480 is the original size of the frame to achieve different performance—the descaled input leads to a faster FRCNN model but with less accuracy.

3.2. YOLO v5 Algorithm

YOLO v5 is the newest version of YOLO[4] as well as the state-of-the-art architecture for object detection. According to the official documents, compared to YOLO V4, YOLO v5 model is about 88% smaller (27MB vs 244MB) and 180% faster(140FPS vs 50FPS) while with almost the same accuracy.

Fig.3 shows the architecture of YOLO v5. It is a single end-to-end neural network for processing an entire image. It mainly consists of 3 parts, the Backbone for feature extraction, the Neck for getting feature pyramids, and the Head which uses anchor boxes to find the possible bounding boxes and their corresponding class probabilities. The basic work processes are: Split the image into cells. For each cell predicts possible bounding boxes and their corresponding possibility of the box contains an object as well as the class probability of this potential object. Then do Non-maximum Suppression (NMS) to find the optimum one from the constructed bounding boxes.

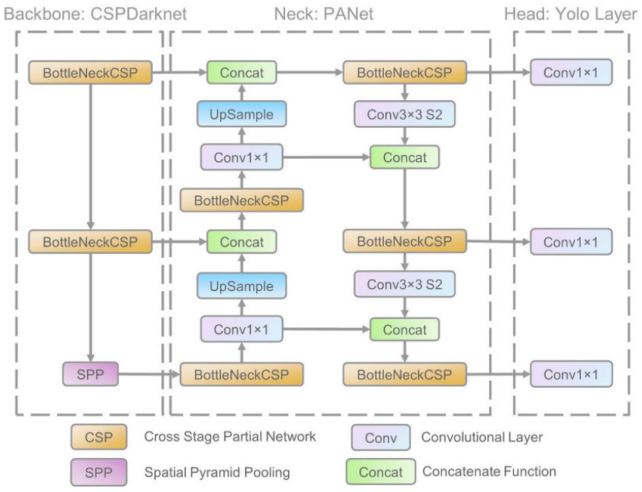

Fig. 3. YOLO v5 architecture.[5]

To implement it on this project, we cloned its original code from Github and followed its official instruction[6] to run it on our local machine. The s version was chosen as the configuration for shortening the time cost of training. The model is trained and tested in the same environment as Faster R-CNN for making the comparison.

From the official instruction and the experiences shared by other users of YOLO v5, We set the training epochs as 300,

and the batch size as 8 for matching the number of computing cores of our GPU to get better computing performance. It cost around 10 hours for training the model which with 99.2% accuracy in the test.

3.3. Model Comparison

YOLO v5s trained to do classification and bounding box regression simultaneously rather than by order. That with the simpler architecture, YOLO v5s runs faster than Faster R-CNN.

	FRCNN (size 300)	**FRCNN (size 480)**	**YOLO v5s**
Speed(FPS)	49.3	36.7	83.3
Accuracy(%)	98.1	98.6	99.2
mAP@[0.5:0.95]	0.765	0.846	0.930
mAP@[0.5]	0.973	0.989	0.995
Training Time(min)	100	120	600

Table 2. Performance of Faster R-CNN and YOLOv5s

To evaluate the performance of the models, we designed and implemented a test function. In the test video, for each object of each frame, if the object was correctly detected with bounding boxes whose IOU is under the threshold, and the object is also correctly classified, we count it as a True case. The ratio of numbers of True cases and the total cases is the accuracy.

Table.2 and Fig. 4 recorded the comparison of Faster R-CNN and YOLO v5s. YOLO v5s is 56% faster than Faster R-CNN while with higher accuracy, which concluded that YOLO v5s is the better choice in our case. But the comparative disadvantage is, YOLO v5s requires much more computing resources to get the well-trained model, which is 6 times more than Faster R-CNN. Thus Faster R-CNN is still a good choice under the limitation of training time or computing resources. Empirically we set the input size of Faster R-CNN to be 300 and 480, where 480 is the original size of the frame to achieve different performance, see in Fig. 5—the descaled input leads to a faster FRCNN model but with less accuracy.

4. OBJECT TRACKING

After having correctly identified and labeled all the cans in the image, they must be assigned a unique name to be able to count and follow them over time. To do so there are different steps to be done. First, the centroid of each object present in the frame is calculated. The identification algorithm outputs the Cartesian coordinates of the upper left (x_l, y_l) and lower right (x_r, y_r) corners. To obtain the coordinates of the centroid it is sufficient to use:

$$x_c = (x_l + x_r)/2 \tag{1}$$

$$y_c = (y_l + y_r)/2 \tag{2}$$

Once all the centroids have been calculated, the Euclidean distance from all the centroids belonging to the previous frame is calculated. In this way each new centroid has an associated distance vector.

$$Dis_{ij} = \sqrt{(x_{c,i} - x_{c,i})^2 + (y_{c,i} - y_{c,i})^2} \tag{3}$$

At this point it is necessary to understand the logic with which to assign the label. The simplest operation is to label the new centroid with the closest old centroid. However, critical situations may arise that need to be resolved. If the label of the nearest has already been used, it passes to that of the next and so on until it is impossible to assign the label. If this happens it means that the new centroid represents a new object and therefore a new label is assigned to it. If a label belonging to the old frame is not used, it means that the object it represents has left the visual field and is therefore eliminated from the list.

Fig.6 shows the result of the application of the object tracking algorithm. The scenes of cans moving very close can be detected and tracked correctly, but we don't have an automatic test method to judge if the tracking is correct.

5. CONCLUSION

In this paper, we have applied two deep learning approaches for object detection and tracking for preventing failures in industrial processes applications. Our approach for the development of object detection was to take high accuracy, the real-time requirement and the hardware limitations into account.

We have successfully developed object detection with both the YOLO V5 and FRCNN algorithms to find which one would be more tailored for our needs. This comparison concluded that the YOLO V5 is more suitable than the FRCNN in terms of accuracy and speed. Other popular algorithms could be involved in the comparison in the future. Augmentation of the data for specific scenarios like blurring the image in a defined direction to simulate the cans' movement would significantly improve the model performance.

Afterwards, object tracking based on the Euclidean distance was developed. The models were trained under annotated bounding-boxes images dataset. As output data, the algorithm gives the inferred bounding-boxes coordinates, which we use to compute the centroid of each object. We repeat this procedure in each subsequent frame, taking into account the computation of the Euclidean distance between each centroid, which also allows us to keep track, add or delete objects in our video.

Further improvements could be made; for instance, implementing a Kalman filter to the algorithm could prevent

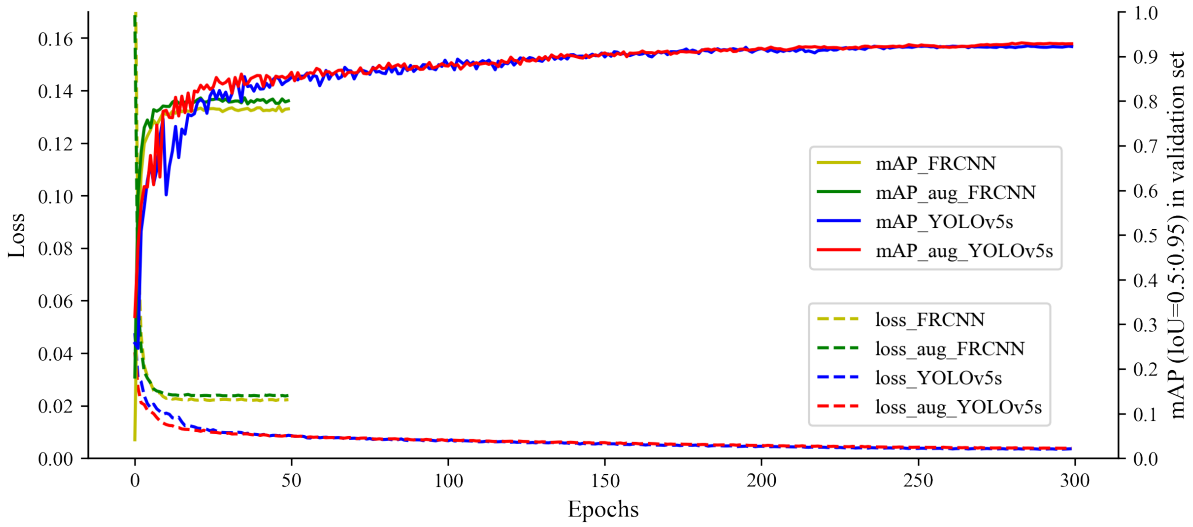

Fig. 4. Training loss and mAP comparison between Faster R-CNN and YOLO v5s

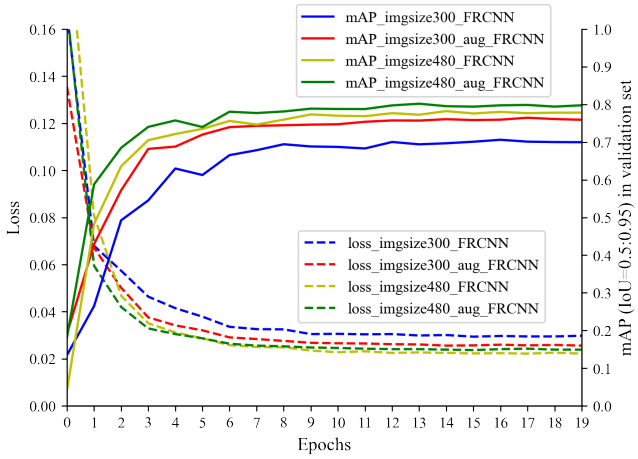

Fig. 5. Training loss and mAP comparison in Faster R-CNN between minimum input size 300 and 480

the problem of occlusion of the objects. Besides, a new test function could be designed to evaluate object tracking performance in terms of accuracy and timeliness.

6. REFERENCES

[1] Shashank Bujimalla, Mahesh Subedar, and Omesh Tickoo, "Data augmentation to improve robustness of image captioning solutions," *CoRR*, vol. abs/2106.05437, 2021.

[2] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian 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, 2016.

[3] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson, "How transferable are features in deep neural networks?," *CoRR*, vol. abs/1411.1792, 2014.

[4] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[5] Aryan Garg, "How to use yolo v5 object detection algorithm for custom object detection," 2021, https://www.analyticsvidhya.com/blog/2021/12/how-to-use-yolo-v5-object-detection-algorithm-for-custom-object-detection-an-example-use-case/ Accessed December 29, 2021.

[6] Glenn Jocher, Alex Stoken, Ayush Chaurasia, Jirka Borovec, NanoCode012, TaoXie, Yonghye Kwon, Kalen Michael, Liu Changyu, Jiacong Fang, Abhiram V, Laughing, tkianai, yxNONG, Piotr Skalski, Adam Hogan, Jebastin Nadar, imyhxy, Lorenzo Mammana, AlexWang1900, Cristi Fati, Diego Montes, Jan Hajek, Laurentiu Diaconu, Mai Thanh Minh, Marc, albinxavi, fatih, oleg, and wanghaoyang0106, "ultralytics/yolov5: v6.0 - YOLOv5n 'Nano' models, Roboflow integration, TensorFlow export, OpenCV DNN support," Oct. 2021.

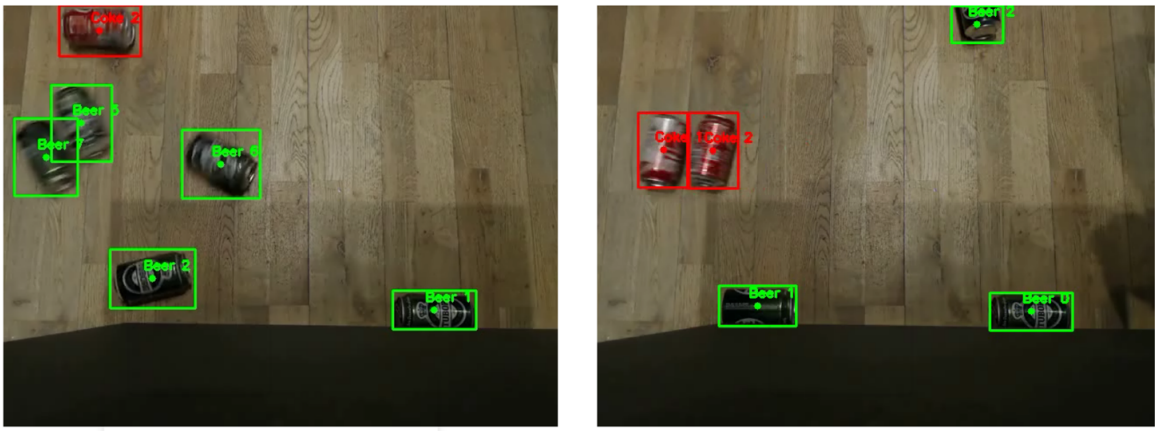

Fig. 6. Object Tracking Demo (Full version see: https://github.com/Michealxh-L/realtime-object-detection-and-tracking.git)