Object detection approaches to be used in embedded system for robots navigation

Ahmad Ali Deeb
Department of Computer Science, Artificial Intelligence, and Control Systems
Bauman Moscow State Technical University
Moscow, Russia
ahmadalideeb3@gmail.com

Farah Shahhoud
Department of computer science, artificial intelligence, and control systems
Bauman Moscow State Technical University
Moscow, Russia
faro7.sh@gmail.com

***Abstract* - This paper investigates the problem of object detection for real-time agents' navigation using embedded systems. In real-world problems, a compromise between accuracy and speed must be found. In this paper, we consider a description of the architecture of different object detection algorithms, such as R-CNN and YOLO, to compare them on different variants of embedded systems using different datasets. As a result, we provide a trade-off study based on accuracy and speed for different object detection algorithms to choose the appropriate one depending on the specific application task.**

***Index Terms* – Robot navigation, object detection, embedded systems, YOLO algorithms, R-CNN algorithms, object semantics.**

I. INTRODUCTION

Target detection has attracted significant attention for autonomous robots due to its notable benefits and recent progress [1]. Target tracking can be used in autonomous vehicles for the development of guidance systems [2]. Pedestrian detection [3], dynamic vehicle detection, and obstacle detection [4] can improve the features of the guiding assistance system. Object recognition technologies for self-driving vehicles have strict requirements in terms of accuracy, unambiguousness, robustness, space demand, and costs [5]. Similarly, object recognition and tracking features in robots can assist in wheeled robots navigation and obstacle avoidance.

Visual navigation systems in service robots can be used in many applications, they are typically deployed in retail, healthcare, warehouse. Others are deployed in more rugged settings, such as in space, defense, agricultural applications, demolition, and for automating dangerous or laborious tasks.

Previously, target detection in robot systems mostly used vision-based target finding algorithms. For example, Raspberry Pi and OpenCV were used to find a target [6]. However, computer vision techniques might provide less accurate results and have issues in predicting unknown future data. On the other hand, machine learning target-detection algorithms can provide a very accurate result, and the model can make predictions from unknown future data. Visual recognition systems involving image classification, localization, and segmentation have accomplished extraordinary research contributions [7]. Moreover, deep learning has made great progress in solving issues in the fields of computer vision, image and video processing, and multimedia [8]. Because of the critical advancements in neural networks, particularly deep learning [9], these visual recognition systems have shown great potential in target tracking.

On-board and off-board ground-based systems are promising platforms in this context. Most of the time, the robot system cannot be equipped with heavy devices due to weight and power consumption. Therefore, off-board ground systems play a vital role. In some cases, communication with the ground station could be impossible due to distance or coverage. An on-board system that can support both weight and power consumption would be a perfect framework for such a situation and environment.

In the present study, we compared various object detection algorithms and different embedded systems to execute those algorithms, based on the comparison we will choose the best algorithm with the best embedded system for object detection in a robot system in real time.

II. IMPLEMENTED OBJECT DETECTION ALGORITHM IN ROBOT SYSTEM

A. Region Proposal Based Framework

The region proposal-based framework, is a two-step process, matches the attentional mechanism of human brain to some extent, which gives a coarse scan of the whole scenario firstly and then focuses on regions of interest [10].

1) Region with CNN (R-CNN): The original paper "Rich feature hierarchies for accurate object detection and semantic segmentation" [11] elaborates one of the first breakthroughs of the use of CNNs in an object detection system called the 'R-CNN' or 'Regions with CNN' that had a much higher object detection performance than other popular methods at the time.

R-CNN generates features in a region using CNN. The algorithm proposed in [11] employed selective search [12] to extract just 2000 regions from the entire input image. These regions are referred to as region proposals RoI and they have a high probability of containing an object. Therefore, instead of classifying big number of regions, just 2000 regions can be worked with as in Fig.1.

Due to the requirement of CNNs to have a fixed input image size, the proposed RoIs are then warped to have a fixed size, then they are fed to a convolution neural network that will extract features from each candidate region.

A classifier like a support vector machine (SVM) [13], [14] will classify the presence of the object within that candidate region proposal based on the extracted features from the previous step.

In addition to predicting the presence of an object within the region proposals, the algorithm also has a bounding-box regressor that predicts four values which are the location and

size of the bounding box that surrounds the object, then filtering with a greedy non-maximum suppression (NMS) [15], [16] to produce final bounding boxes. R-CNN architecture is shown in Fig. 1.

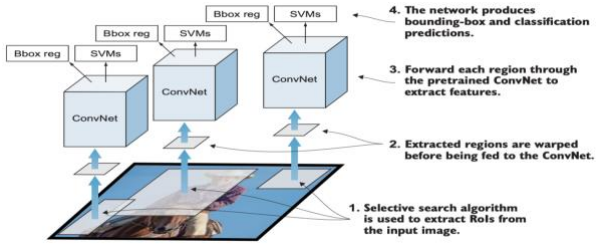

Fig. 1 R-CNN Architecture.

Limitations of R-CNN. It still takes a huge amount of time to train the network as it would have to classify 2000 region proposals per image, it cannot be implemented real time as it takes around 47 seconds for each test image and the selective search algorithm is a fixed algorithm. Therefore, no learning is happening at that stage. This could lead to the generation of bad candidate region proposals.

2) Fast R-CNN: R-CNN model took a huge amount of time to train the network. Girshick et al. [17] built another faster object detection algorithm known as Fast R-CNN to circumvent this problem. Instead of starting with the regions proposal module and then using the feature extraction module, like R-CNN, Fast-RCNN proposes that we apply the CNN feature extractor first to the entire input image and then propose regions. This way, we run only one ConvNet over the entire image instead of 2,000 ConvNets over 2,000 overlapping regions. And the region proposals are generated using other algorithms algorithm such as Edge Boxes [18].

The ConvNet has an extended job to do the classification part as well, this has done by replacing the traditional SVM machine learning algorithm [13], [14] with a softmax layer. This way, single model will perform both tasks: feature extraction and object classification. Fast R-CNN architecture is shown in Fig. 2.

Limitations of Fast R-CNN. Still requires candidate regions as input and the running time of Fast R-CNN is reduced, exposing region proposal computation as a bottleneck.

3) Faster R-CNN: Faster R-CNN is the third iteration of the R-CNN family, developed in 2016 by Shaoqing Ren et al [19]. Similar to Fast R-CNN, the image is provided as an input to a convolutional network that provides a convolutional feature map. Instead of using a selective search algorithm on the feature map to identify the region proposals, a region proposal network (RPN) is used to predict the region proposals as part of the training process.

The architecture of Faster R-CNN can be described using two main networks, first network is region proposal network (RPN)—Selective search is replaced by a ConvNet that proposes RoIs from the last feature maps of the feature extractor to be considered for investigation where the RPN has two outputs, the objectness score (object or no object) and the box location. Second network consists of the typical components of Fast R-CNN. Faster R-CNN architecture is shown in Fig. 3.

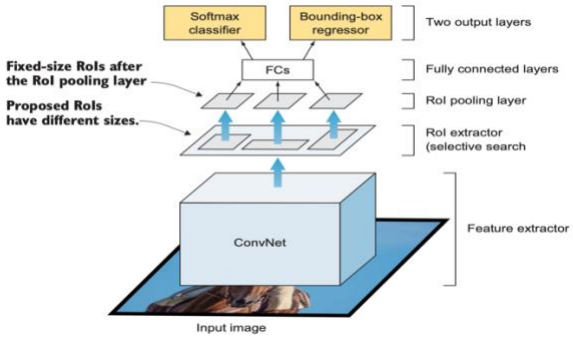

Fig. 2 Fast R-CNN Architecture.

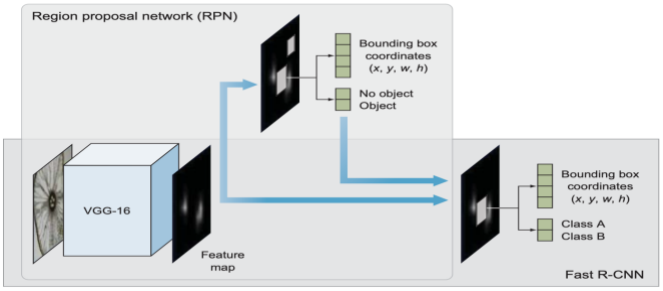

Fig. 3 Faster R-CNN Architecture.

4) Mask R-CNN: Mask R-CNN is an extended version of faster R-CNN for pixel level segmentation. Mask R-CNN [20] works by adding a branch for predicting segmentation masks on each Region of Interest (RoI), in parallel with the existing branch for classification and bounding box regression. The branch is a fully convolutional network FCN [21] on top of a CNN-based feature map. Once these masks are generated, mask R-CNN amalgamates them with the classifications and bounding boxes that result from faster R-CNN. Overall, it generates precise segmentation.

In the second stage of Faster R-CNN, RoI pool is replaced by RoIAlign which helps to preserve spatial information which gets misaligned in case of RoI pool. RoIAlign uses binary interpolation to create a feature map that is of fixed size.

The output from RoIAlign layer is then fed into Mask head, which consists of two convolution layers. It generates mask for each RoI, thus segmenting an image in pixel-to-pixel manner. Mask R-CNN architecture is shown in Fig. 4.

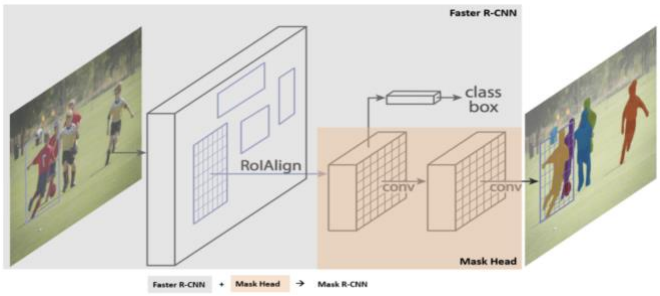

Fig. 4 Mask R-CNN Architecture.

B. Regression/Classification Based Framework

One-step frameworks based on global regression/classification, mapping straightly from image pixels to bounding box coordinates and class probabilities, can reduce time expense [10].

R-CNN object detection systems need to go through two stages to detect the objects. YOLO doesn't need to go through these boring processes. It only needs to look once at the image to detect all the objects and that is why they chose the name (You Only Look Once) and that is actually the reason why YOLO is a very fast model.

1) YOLOv1: YOLO [22] uses an innovative strategy to resolving object detection as a regression problem, it detects bounding box coordinates and class probabilities directly from the image.

YOLO divides the input image into an S × S grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object (Each grid cell predicts only one object). Each grid cell predicts B bounding boxes and confidence scores for those boxes.

These confidence scores reflect how confident the model is that the box contains an object Pr(Object), as well as how accurate is the predicted box by evaluating its overlap with the ground truth bounding box measured by intersection over union $\mathrm{IoU}_{\mathrm{pred}}^{\mathrm{truth}}$.

Each grid cell also predicts C conditional class probabilities, Pr(Classi |Object). These probabilities are conditioned on the grid cell containing an object. We only predict one set of class probabilities per grid cell, regardless of the number of boxes B. YOLOv2 architecture is shown in Fig. 5.

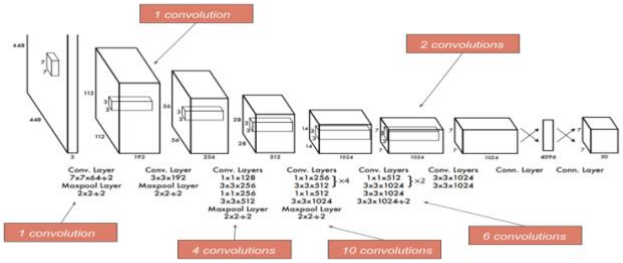

Fig. 5 YOLOv1 Architecture.

Loss function. YOLO uses sum-squared error between the predictions and the ground truth to calculate loss. The loss function composes of the classification loss, the localization loss (errors between the predicted boundary box and the ground truth) and the confidence loss (the objectness of the box).

2) YOLOv2: Since YOLOv1 suffers from localization errors and low recall predictions, the YOLOv2 [23] shows a lot of improvement to increase the speed vs accuracy trade-off.

Design improvement in YOLOv2. Batch normalization, classifying on high resolution inputs, convolutional with anchor boxes, multi-scale training and direct location prediction. YOLOv2 architecture is shown in Fig. 6.

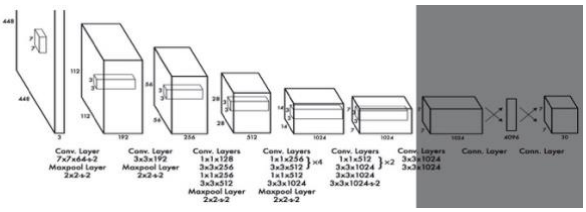

Fig. 6 YOLOv2 Architecture.

3) YOLOv3: YOLOv3 [24] is an improved version of YOLOv2. First, YOLOv3 can give multi-label classification [25] (independent logistic classifiers) to adapt to more complex datasets containing many overlapping labels. Second, YOLOv3 predicts bounding boxes at three different scales by following the idea of feature pyramid network for object detection [26]. The last convolutional layer predicts a 3-d tensor encoding class predictions, objectness, and bounding box. Third, YOLOv3 proposes a deeper and robust feature extractor, called Darknet-53, inspired by ResNet. YOLOv3 architecture is shown in Fig. 7.

Due to the advantages of multi-scale predictions, YOLOv3 can detect small objects even more but has comparatively worse performance on medium and larger sized objects.

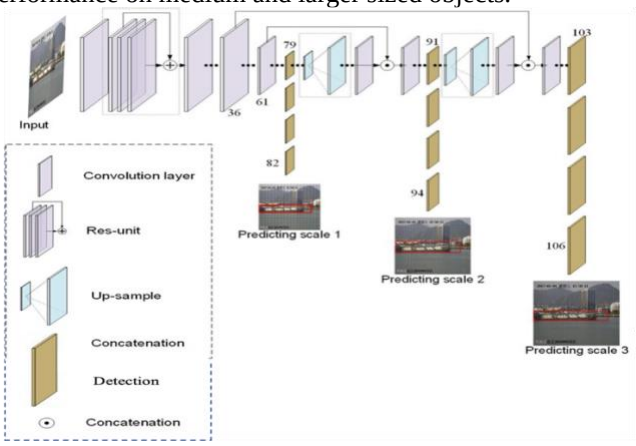

Fig. 7 YOLOv3 Architecture.

4) YOLOv3 Tiny: Tiny YOLOv3 is a lightweight target detection algorithm applied to embedded platforms based on YOLOv3. Therefore, the running speed is significantly increased, but detection accuracy is reduced [27], [28], [29], [30], [31].

Tiny YOLOv3 reduced the YOLOv3 feature detection network darknet-53 to a 13 convolution layers and a 6 Max Pooling layers, Tiny-yolov3 uses the pooling layer instead of YOLOV3's convolutional layer with a step size of 2 to achieve dimensionality reduction. Prediction of bounding boxes occurs at two different feature map scales, which are 13×13, and 26×26 merged with an upsampled 13×13 feature map to predict the target. YOLOv3 Tiny architecture is shown in Fig. 8.

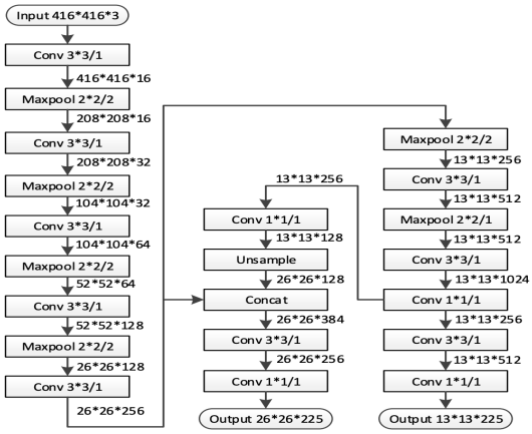

Fig. 8 YOLOv3-Tiny Architecture.

The YOLO family is a series of end-to-end DL models designed for fast object detection, and it was among the first attempts to build a fast real-time object detector. It is one of the fastest algorithms out there. Although the accuracy of the models is close but not as good as R-CNNs, they are popular for object detection because of their detection speed, often demonstrated in real-time video or camera feed input.

III. EXPERIMENTS

A. Embedded systems

Since we are dealing with robots that may be small in size, then we should search for embedded systems that are compact in size, low power consuming and the most important feature is to have high computational performance.

NVidia Jetson devices are embedded AI computing platforms that provide high-performance, low-power computing support for deep learning and computer vision. Together with NVIDIA JetPack™ SDK, these Jetson modules open the door to develop and deploy innovative products across all industries [32].

Jetson is used to deploy a wide range of popular DNN models and ML frameworks to the edge with high performance inferencing, for tasks like real-time classification and object detection, pose estimation, semantic segmentation, and natural language processing (NLP).

Jetson Nano is a small, powerful computer that is able to run multiple neural networks in parallel for applications like image classification, object detection, segmentation, and speech processing. All in an easy-to-use platform that runs in as little as 5 watts [33].

Jetson TX1 is the world's first supercomputer on a module and can provide support for visual computing applications. It is built with the NVidia Maxwell™ architecture and 256 CUDA cores delivering performance of over one teraflop [34].

Jetson TX2 is one of the fastest, most power-efficient embedded AI computing devices. This 7.5-watt supercomputer on a module brings true AI computing at the edge. An NVidia Pascal™ family GPU was used to build it and loaded with 8 GB of memory and 59.7 GB/s of memory bandwidth. It included an assortment of standard equipment interfaces that make it simple to incorporate into a wide scope of hardware [35].

Jetson AGX Xavier has exceeded the limit capabilities of previous Jetson modules to a great extent. In terms of performance and efficiency in deep learning and computer vision, it has surpassed the world's most autonomous machines and advanced robot [36]. This powerful AI computing GPU workstation works under 30 W. It was built around a NVidia Volta™ GPU with Tensor Cores, two NVDLA engines, and an eight-core 64-bit ARM CPU. NVidia Jetson AGX Xavier is the most recent expansion to the Jetson stage [37]. This AI GPU computer can provide unparalleled 32 TeraOPS (TOPS) of the peak computation in a compact 100-mm × 87-mm module form-factor [38]. Xavier's energy efficient module can be deployed in next-level intelligent machines for end-to-end autonomous capabilities. Basic comparison between Jetson modules shows comparison between Jetson modules

TABLE I. BASIC COMPARISON BETWEEN JETSON MODULES

	GPU	CPU	Memory	AI Performance	Power
Jetson Nano	128-core Maxwell	Quad-core ARM A57 @ 1.43 GHz	4 GB 64-bit LPDDR4 25.6 GB/s	472 GFL OPs	5W / 10W
Jetson TX1	256-core NVIDIA Maxwell™ GPU	Quad-Core ARM® Cortex®-A57 MPCore	4GB 64-bit LPDDR4 Memory	1 TFL OPs	Under 10W
Jetson TX2	256-core NVIDIA Pascal™ GPU architecture with 256 NVIDIA CUDA cores	Dual-Core NVIDIA Denver 2 64-Bit CPU Quad-Core ARM® Cortex®-A57 MPCore	8GB 128-bit LPDDR4 Memory 1866 MHx - 59.7 GB/s	1.33 TFL OPs	7.5W / 15W
Jetson Xavier	512-core Volta GPU with Tensor Cores	8-core ARM v8.2 64-bit CPU, 8MB L2 + 4MB L3	32GB 256-Bit LPDDR4x | 137GB/s	16 TFL OPs	10W / 15W / 30W

Since Jetson Xavier embedded system is the most powerful among all other candidates, we will use it to benchmark different object detection algorithms to choose the most suitable for our task.

B. Datasets

We compare various object detection methods on two benchmark datasets, including PASCAL VOC 2007 [39] and Microsoft COCO [40].

The evaluated approaches include R-CNN [11], Fast R-CNN [17], Faster R-CNN [19], Mask R-CNN [20], YOLO [22], YOLOv2 [23], YOLOv3 [24] and YOLOv3 tiny [27], [28], [29], [30], [31].

PASCAL VOC 2007 dataset consists of 20 categories. Microsoft COCO, on the other hand, is composed of more than 300,000 fully segmented images, in which each image has an average of 7 object instances from a total of 80 categories. As there are a lot of less iconic objects with a broad range of scales and a stricter requirement on object localization, this dataset is more challenging than PASCAL 2007.

Dataset statistics shows some comparison of the differences between the PASCAL VOC 2007 and Microsoft COCO.

TABLE II. DATASET STATISTICS

Dataset	*Microsoft COCO*	*PASCAL VOC 2007*
Number of categories	80	20
Number of train-val images	246690	5011
Number of test images	81,434	4952
Number of annotated objects	2500000	24640
Total objects/total number of images	7.6	2.4

Object detection performance is evaluated by average precision AP. For PASCAL VOC 2007, the evaluation terms are Average Precision (AP) in each single category and mean Average Precision (mAP) across all the 20 categories. In COCO dataset AP metric is used for evaluation which is an averaged over multiple Intersection over Union (IoU) values, specifically 10 IoU thresholds of [.50:.05:.95] are used.

AP is also averaged over all categories. Traditionally, this is called "mean average precision" (mAP), but we will denote to it as AP to distinct between COCO and VOC 07 evaluation metrics. So, for COCO we will use AP (averaged across all 10 IoU thresholds and all 80 categories). Averaging over IoUs rewards detectors with better localization.

IV. RESULTS

A. Accuracy comparison

It is important to note that technology is constantly evolving, any comparison can become obsolete quickly.

These experiments are performed in different environments [10], [11], [17], [19], [20], [22], [23], [24], [29], [41], [42], so maybe the results will change a little bit if it is tried to reproduce them. But the purpose of this article is to have a general notion about these methods.

Tables Comparative results on MICROSOFT COCO TEST DEV SET (%), Comparative results on VOC 2007 TEST SET (%), Accuracy of different object detection algorithms on Microsoft COCO and PASCAL VOC 2007 datasets show results of the proposed algorithms on different datasets.

Overall, region proposal-based methods, such as Faster R-CNN and Mask R-CNN, perform better than regression/classfication based approaches like YOLO, due to the fact that quite a lot of localization errors are produced by regression/classification-based approaches.

TABLE III. COMPARATIVE RESULTS ON MICROSOFT COCO TEST DEV SET (%)

Algorithm	backbone	Trained on	AP (%)
R-CNN	—	—	—
Fast R-CNN	Vgg16	COCO train	19.7
Faster R-CNN	Vgg16	COCO trainval	21.7
Mask R-CNN	ResNet-101-FPN	COCO trainval35	39.8
YOLO	—	—	—
YOLOv2	Darknet-19	COCO trainval35	21.6
YOLOv3	Darknet-53	COCO trainval35	33.0
YOLOv3 tiny	Reduced Darknet-53	COCO trainval35	15.3

TABLE IV. COMPARATIVE RESULTS ON VOC 2007 TEST SET (%)

Algorithm	backbone	Trained	mAP (%)
R-CNN	Vgg16	07	66.0
Fast R-CNN	Vgg16	07+12	70.0
Faster R-CNN	Vgg16	07+12	76.4
Mask R-CNN	—	—	—
YOLO	Googlenet	07+12	63.4
YOLOv2	Darknet-19	07+12	78.6
YOLOv3	Darknet-53	07+12	87.4
YOLOv3 tiny	Reduced Darknet-53	07+12	61.3

TABLE V. ACCURACY OF DIFFERENT OBJECT DETECTION ALGORITHMS ON MICROSOFT COCO AND PASCAL VOC 2007 DATASETS

Algorithm	Microsoft COCO AP (%)	PASCAL VOC 2007 mAP (%)
R-CNN	—	66.0
Fast R-CNN	19.7	70.0
Faster R-CNN	21.7	76.4
Mask R-CNN	39.8	—
YOLO	—	63.4
YOLOv2	21.6	78.6
YOLOv3	33.0	87.4
YOLOv3 tiny	15.3	61.3

As YOLOv1 is not skilled in producing object localizations of high IoU, it obtains a very poor result on VOC 2007. However, with the aid of other strategies, such as anchor boxes, BN and fine-grained features, the localization errors are corrected (YOLOv2).

Fig. 9, 10 show algorithms performance on PASCAL VOC 2007 and Microsoft COCO dataset respectively.

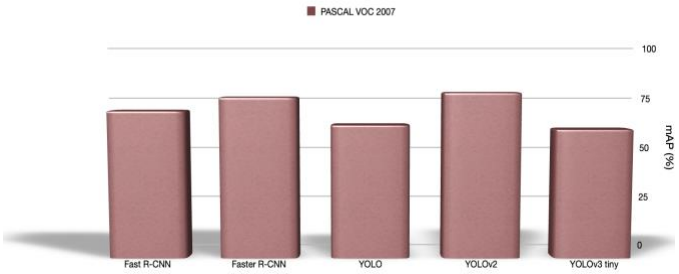

Fig. 9 Accuracy performance of different object detection algorithms on PASCAL VOC 2007 dataset.

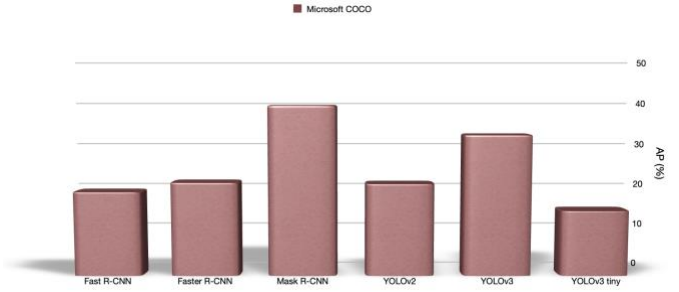

Fig. 10 Accuracy performance of different object detection algorithms on Microsoft COCO dataset.

We can notice that the results on COCO dataset are much worse than those of VOC 2007, and this is due to the existence of a large number of nonstandard small objects.

B. Speed comparison

The processing speed variation experiment was performed for each of the models on three devices. First, the experiments were performed on Jetson TX1 [34], Jetson TX2 [35], and Jetson Xavier that was released for edge-computing [37].

The performance results in the form of frame per seconds FPS are shown in below [43], [44]. This table provides a quantitative comparison between different type of on-board embedded GPU system. However, using this table, one can choose the best algorithm and system for a specific operation.

TABLE VI. PERFORMANCE COMPARISON BETWEEN JETSON MODULES USED FOR TARGET DETECTION

algorithm	*TX1 (FPS)*	*TX2 (FPS)*	*Xavier (FPS)*
Fast R-CNN	—	—	—
Faster R-CNN	—	1	1.3
Mask R-CNN	—	—	—
YOLO			
YOLOv2	3	10	28
YOLOv3	—	4	17
YOLOv3 tiny	9	11	31

Regression based models can usually be processed in real-time at the cost of a drop in accuracy compared with region proposal-based models.

The higher resolution images for the same model have better Map but are slower to process.

The performance and efficiency of Jetson AGX Xavier makes it possible to process all of the components needed for robots to function safely with full autonomy in real-time, including high-performance vision algorithms for real-time perception, navigation, and manipulation.

Fig. 11, 12 show the performance in terms of speed and accuracy for some object detection algorithms that are applicable on NVIDIA Jetson Xavier.

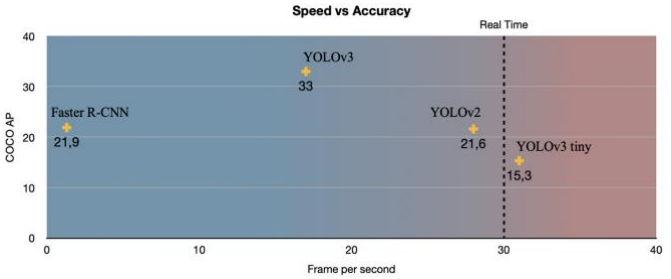

Fig. 11 Object detection algorithms Speed vs accuracy trade-off.

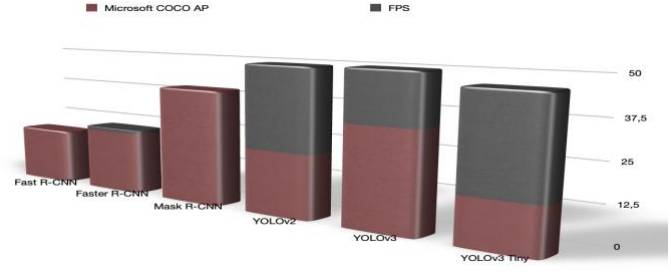

Fig. 12 Performance of different object detection algorithms in terms of speed and accuracy.

V. CONCLUSION

It is difficult to define a fair feature of different object detectors, each case of real life can have different solutions to reach a decision concerning the accuracy and speed. It is necessary to know other factors that affect performance: the type of feature extractor, steps out of the extractor, income resolutions images, strategy coincidence and threshold (as predictions are excluded when calculating the loss), Threshold IoU no maximum suppression ratio of positive anchor and negative, number of proposals, increased data set of training data, using multi-scale images training or testing.

Since the main focus of this paper was on object detection for a robot navigation, we first needed to figure out which algorithm provides faster and more accurate results.

Even though that YOLOv3 tiny is the only algorithm that can be implemented on Jetson Xavier in real-time (more than 30 FPS) but our choice to use YOLOv2 on robot application that demands real-time processing.

The YOLOv2 selection was based on its speed performance that is very close to real-time and its accuracy (21.6% COCO AP) which is 40% better than YOLOv3 tiny (15.3% COCO AP).

In this paper, we have reviewed some object detection algorithms including region proposal approaches like R-CNN family and regression/classification approaches like YOLO versions.

We have also discussed the properties of some on-board embedded GPU system that can be used to perform deep learning processing and the difference between them.

In addition, we performed an experimental comparison on the performance of each algorithm in terms of accuracy based on two datasets Microsoft COCO and PASCAL VOC 2007 and in terms of processing speed by on-board embedded GPU systems.

This paper can be used to determine the appropriate object detection algorithm for a specific task relying on speed/accuracy trade-off.

REFERENCES

[1] Yoon, Y., Gruber, S., Krakow, L., & Pack, D. (2009). Autonomous target detection and localization using cooperative unmanned aerial vehicles. In M. J. Hirsch, C. W. Commander, P. M. Pardalos, & R. Murphey (Eds.), *Optimization and Cooperative Control Strategies* (Vol. 381, pp. 195–205). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-88063-9_12.

[2] Gietelink, O., Ploeg, J., De Schutter, B., & Verhaegen, M. (2006). Development of advanced driver assistance systems with vehicle hardware-in-the-loop simulations. *Vehicle System Dynamics*, *44*(7), 569–590. https://doi.org/10.1080/00423110600563338.

[3] Gerónimo, D., López, A. M., Sappa, A. D., & Graf, T. (2010). Survey of pedestrian detection for advanced driver assistance systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *32*(7), 1239–1258. https://doi.org/10.1109/TPAMI.2009.122.

[4] Ferguson, D., Darms, M., Urmson, C., & Kolski, S., (2008). Detection, prediction, and avoidance of dynamic obstacles in urban environments. IEEE Intelligent Vehicles Symposium, pp. 1149-1154, doi: 10.1109/IVS.2008.4621214.

[5] Hirz, M., & Walzel, B. (2018). Sensor and object recognition technologies for self-driving cars. *Computer-Aided Design and Applications*, *15*(4), 501–508. https://doi.org/10.1080/16864360.2017.1419638.

[6] Ajmal Hinas, Jonathan Roberts, & Felipe Gonzalez. (2017a). Vision-based target finding and inspection of a ground target using a multirotor uav system. *Sensors*, *17*(12), 2929. https://doi.org/10.3390/s17122929.

[7] Pathak, A. R., Pandey, M., & Rautaray, S. (2018). Application of deep learning for object detection. Procedia Computer Science, 132, 1706–1717. https://doi.org/10.1016/j.procs.2018.05.144.

[8] Tijtgat, N., Van Ranst, W., Volckaert, B., Goedeme, T., & De Turck, F. (2017). Embedded real-time object detection for a uav warning system. *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, 2110–2118. https://doi.org/10.1109/ICCVW.2017.247.

[9] Han, S., Shen, W., & Liu, Z. (2016). Deep Drone : Object Detection and Tracking for Smart Drones on Embedded System.

[10] Zhao, Z.-Q., Zheng, P., Xu, S., & Wu, X. (2019). Object detection with deep learning: A review. ArXiv:1807.05511 [Cs]. http://arxiv.org/abs/1807.05511

[11] Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. ArXiv:1311.2524 [Cs]. http://arxiv.org/abs/1311.2524.

[12] Uijlings, J. R. R., van de Sande, K. E. A., Gevers, T., & Smeulders, A. W. M. (2013). Selective search for object recognition. International Journal of Computer Vision, 104(2), 154–171. https://doi.org/10.1007/s11263-013-0620-5.

[13] Drucker, H., Burges, C. JC., Kaufman, L., Smola, A. J., & Vapnik, V. (1997). Support vector regression machines. In Advances in neural information processing systems. 155–161.

[14] Hearst, M. A., Dumais, S. T., Osuna, E., Platt, J., & Scholkopf, B. (1998). Support vector machines. IEEE Intelligent Systems and Their Applications, 13(4), 18–28. https://doi.org/10.1109/5254.708428.

[15] Neubeck, A. & Van Gool, L. (2006). Efficient Non-Maximum Suppression. 18th International Conference on Pattern Recognition (ICPR'06), pp. 850-855, doi: 10.1109/ICPR.2006.479.

[16] Hosang, J., Benenson, R., & Schiele, B. (2017). Learning non-maximum suppression. ArXiv:1705.02950 [Cs]. http://arxiv.org/abs/1705.02950.

[17] Girshick, R. (2015). Fast r-cnn. ArXiv:1504.08083 [Cs]. http://arxiv.org/abs/1504.08083

[18] Zitnick, C. L., & Dollár, P. (2014). Edge boxes: Locating object proposals from edges. In D. Fleet, T. Pajdla, B. Schiele, & T. Tuytelaars (Eds.), Computer Vision – ECCV 2014 (Vol. 8693, pp. 391–405). Springer International Publishing. https://doi.org/10.1007/978-3-319-10602-1_26

[19] Ren, S., He, K., Girshick, R., & Sun, J. (2016). Faster r-cnn: Towards real-time object detection with region proposal networks. ArXiv:1506.01497 [Cs]. http://arxiv.org/abs/1506.01497

[20] He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2018). Mask r-cnn. *ArXiv:1703.06870 [Cs]*. http://arxiv.org/abs/1703.06870.

[21] Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3431–3440. https://doi.org/10.1109/CVPR.2015.7298965.

[22] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 779–788. https://doi.org/10.1109/CVPR.2016.91.

[23] Redmon, J., & Farhadi, A. (2017). Yolo9000: Better, faster, stronger. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 6517–6525. https://doi.org/10.1109/CVPR.2017.690

[24] Joseph Redmon, & Ali Farhadi. (2018). YOLOv3: An Incremental Improvement. *ArXiv:* 1804.02767 *[Cs]*. https://arxiv.org/abs/1804.02767.

[25] Tsoumakas, G., & Katakis, I. (2007). Multi-label classification: An overview. *International Journal of Data Warehousing and Mining*, *3*(3), 1–13. https://doi.org/10.4018/jdwm.2007070101

[26] Lin, T.-Y., Dollar, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017). Feature pyramid networks for object detection. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 936–944. https://doi.org/10.1109/CVPR.2017.106

[27] Ding, S., Long, F., Fan, H., Liu, L., & Wang, Y. (2019). A novel yolov3-tiny network for unmanned airship obstacle detection. *2019 IEEE 8th Data Driven Control and Learning Systems Conference (DDCLS)*, 277–281. https://doi.org/10.1109/DDCLS.2019.8908875.

[28] Mao, Q.-C., Sun, H.-M., Liu, Y.-B., & Jia, R.-S. (2019). Mini-yolov3: Real-time object detector for embedded applications. *IEEE Access*, *7*, 133529–133538. https://doi.org/10.1109/ACCESS.2019.2941547.

[29] Fang, W., Wang, L., & Ren, P. (2020). Tinier-yolo: A real-time object detection method for constrained environments. *IEEE Access*, *8*, 1935–1944. https://doi.org/10.1109/ACCESS.2019.2961959.

[30] Adarsh, P., Rathi, P., & Kumar, M. (2020). YOLO v3-Tiny: Object Detection and Recognition using one stage improved model. *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, 687–694. https://doi.org/10.1109/ICACCS48705.2020.9074315.

[31] Xiao, D., Shan, F., Li, Z., Le, B. T., Liu, X., & Li, X. (2019). A target detection model based on improved tiny-yolov3 under the environment of mining truck. *IEEE Access*, *7*, 123757–123764. https://doi.org/10.1109/ACCESS.2019.2928603.

[32] Meet Jetson, the Platform for AI at the Edge. Available online: https://developer.nvidia.com/embedded-computing (accessed on 28 April 2021).

[33] Jetson Nano Developer Kit. Available online: https://developer.nvidia.com/embedded/jetson-nano-developer-kit (accessed on 28 April 2021).

[34] Jetson TX1 Module. Available online: https://developer.nvidia.com/embedded/buy/jetson-tx1 (accessed on 28 April 2021).

[35] Jetson TX2 Module. Available online: https://developer.nvidia.com/embedded/jetson-tx2 (accessed on 28 April 2021).

[36] NVidia JETSON AGX XAVIER: The AI Platform for Autonomous Machines. Available online: www.nvidia.com/en-us/autonomous-machines/jetson-agx-xavier/ (accessed on 28 April 2021).

[37] Jetson AGX Xavier Developer Kit. Available online: https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit (accessed on 28 April 2021).

[38] NVidia Jetson AGX Xavier Delivers 32 TeraOps for New Era of AI in Robotics. Available online: devblogs.nvidia.com/nvidia-jetson-agx-xavier-32-teraops-ai-robotics/ (accessed on 28 April 2021).

[39] Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., & Zisserman, A. (2010). The pascal visual object classes (Voc) challenge. International Journal of Computer Vision, 88(2), 303–338. https://doi.org/10.1007/s11263-009-0275-4

[40] Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., & Dollár, P. (2015). Microsoft coco: Common objects in context. ArXiv:1405.0312 [Cs]. http://arxiv.org/abs/1405.0312

[41] Shen, Z., Liu, Z., Li, J., Jiang, Y.-G., Chen, Y., & Xue, X. (2019). Object detection from scratch with deep supervision. *ArXiv:1809.09294 [Cs]*. http://arxiv.org/abs/1809.09294

[42] Zhang, F., Luan, J., Xu, Z., & Chen, W. (2020). Detreco: Object-text detection and recognition based on deep neural network. *Mathematical Problems in Engineering*, *2020*, 1–15. https://doi.org/10.1155/2020/2365076

[43] Hossain & Lee. (2019). Deep learning-based real-time multiple-object detection and tracking from aerial imagery via a flying robot with gpu-based embedded devices. *Sensors*, *19*(15), 3371. https://doi.org/10.3390/s19153371.

[44] Murthy, C. B., Hashmi, M. F., Bokde, N. D., & Geem, Z. W. (2020). Investigations of object detection in images/videos using various deep learning techniques and embedded platforms—A comprehensive review. *Applied Sciences*, *10*(9), 3280. https://doi.org/10.3390/app10093280.