



A dynamic discarding technique to increase speed and preserve accuracy for YOLOv3

Ignacio Martinez-Alpiste¹ · Gelayol Golcarenenji¹ · Qi Wang¹ · Jose Maria Alcaraz-Calero¹

Received: 12 July 2020 / Accepted: 21 January 2021 / Published online: 5 March 2021
© The Author(s) 2021

Abstract

This paper proposes an acceleration technique to minimise the unnecessary operations on a state-of-the-art machine learning model and thus to improve the processing speed while maintaining the accuracy. After the study of the main bottlenecks that negatively affect the performance of convolutional neural networks, this paper designs and implements a discarding technique for YOLOv3-based algorithms to increase the speed and maintain accuracy. After applying the discarding technique, YOLOv3 can achieve a 22% of improvement in terms of speed. Moreover, the results of this new discarding technique were tested on Tiny-YOLOv3 with three output layers on an autonomous vehicle for pedestrian detection and it achieved an improvement of 48.7% in speed. The dynamic discarding technique just needs one training process to create the model and thus execute the approach, which preserves accuracy. The improved detector based on the discarding technique is able to readily alert the operator of the autonomous vehicle to take the emergency brake of the vehicle in order to avoid collision and consequently save lives.

Keywords Machine learning · YOLOv3 · Acceleration · Autonomous vehicles

1 Introduction

Object recognition is one of the most challenging problems in the field of computer vision which has been evolved to a very large extent in the last two decades. Both accuracy and speed are essential requirements to successfully detect objects and tackle the difficulties of deployment in real-time systems.

Object detection techniques can be generally classified into two main categories using handcraft features and neural networks. Histogram of Oriented Gradients (HOG) [9] and Viola-Jones(VJ) [50] are effective traditional machine learning techniques widely used in object detection and employed by classification techniques such as SVM [7] and AdaBoost [45] (Adaptive Boosting). To increase the performance of these object detectors, image processing techniques are utilised. Subsequently, two-stage (region proposal technique) and single-step detectors (non-region proposal techniques) were evolved. In the former,

the region of interest is extracted and then the classification is conducted. Popular two-stage techniques include Fast R-CNN [12], Faster R-CNN [44] and R-FCN [8]. The computation is heavy in these methods though. The latter class performs the feature extraction and classification in a single step. Therefore, these methods are computationally better for real-time applications. “You Only Look Once” (YOLO) series [41–43], “You Look Only Twice” [49] and “Single Shot Detector (SSD)” [29] are examples of single-step detectors. Meanwhile, these methods are fast, it could also lead to lower accuracy. YOLOv3 is more accurate and faster than other object detectors algorithm and thus will be the basis for this paper.

This method, however, is still restricted for real-time on-board detection due to its intensive computation and slow speed; hence, a simplified version of YOLOv3 called Tiny-YOLOv3 has been defined. Tiny-YOLOv3 is very fast but with low accuracy specially when it comes to small object detection. Hence, an extra output layer is added to improve the accuracy of the standard Tiny-YOLOv3 [2] for small objects. To accelerate this algorithm while maintaining the obtained accuracy, different methods can be adopted. Generally, there are three categories of accelerating the

✉ Ignacio Martinez-Alpiste
ignacio.alpiste@uws.ac.uk

¹ University of the West of Scotland, Paisley, UK

Convolutional Neural Networks (CNNs), namely structure level, algorithm level and implementation level.

In terms of structure level, network pruning and knowledge distillation are performed to decrease the redundancy in weights. Switching to 16-bit fixed-point presentation can also increase the frame rates when dealing with redundancy in representation. These methods, however, slightly reduce the accuracy [5]. At the implementation level, hardware technologies such as Graphics Processing Unit (GPU), field-programmable gate array (FPGA), and application-specific integrated circuit (ASIC) are generally employed to accelerate the CNN algorithms against common CPUs. However, technologies such as FPGA are expensive and difficult to program. At algorithm level, efficient models are designed to accelerate the CNN algorithms [57], which is the focus of this paper. Hence, in this paper, we propose an acceleration technique at algorithm level by discarding output layers on standard YOLOv3 and Tiny-YOLOv3 with an extra output layer embedded on a NVIDIA Jetson Xavier platform.

The proposed technique in object detection algorithms to maintain speed/accuracy trade-off is expected to achieve real-time detection for critical applications. The proposed acceleration technique at algorithm level will dynamically discard output layers on demand for standard YOLOv3 and Tiny-YOLOv3 embedded on an NVIDIA Jetson Xavier platform. One of such critical use cases is object detection in autonomous vehicles. For instance, object detection is a critical capability for autonomous cars to be aware of the objects in their vicinity and be able to detect, recognise and track objects, especially pedestrians from distance without delay to avoid any accidents. However, pedestrian detection is very challenging on autonomous vehicles (AVs). There have been at least five major accidents caused by AVs hitting objects and one was found to be involved in a pedestrian death. This was due to necessary information not available on time to the pedestrian detector [33]. Therefore, there is a strong need to minimise unnecessary operations to improve the processing speed and fulfil low-latency criterion while maintaining high accuracy of pedestrian detectors deployed in AVs applications. Besides, these applications are executed in low-consumption GPUs plugged inside the vehicles, therefore, light and portable detection networks are decisive.

For this reason, our approach, as a promising robust technique, has been tested as the brain of the self-driving cars to improve the pedestrian detection speed while ensuring high accuracy suitable for self-driving scenarios where a speed/accuracy trade-off is imperative. To this end, our main contributions are summarised as follows:

1. Study of the main bottlenecks that negatively affect the CNN execution.
2. Design of a novel acceleration technique to further improve the speed while maintaining the accuracy and mitigating the bottlenecks of CNN execution.
3. Study of the compatibility of different machine learning platforms with the proposed acceleration technique.
4. Implementation of the proposed acceleration technique at algorithm level on the suitable machine learning platform.
5. Deployment of the system in a small-form AI platform such as NVIDIA Jetson Xavier.
6. Testing of different input resolutions to find the best trade-off between inference time and accuracy in standard Tiny-YOLOv3 with extra output layer using the proposed discarding technique for pedestrian detection in AV applications.

The rest of the paper is arranged as follows. Section II presents the related work. Section III explains the design of the proposed acceleration technique. This is followed by the experimental set up and empirical results in Section IV. Section V concludes the article.

2 Related Work

This section focuses on a description of the machine learning algorithm used in this publication. In addition, an extensive research work has been done to increase the knowledge of previously published works based on the improvement of CNN execution and pedestrian detection for autonomous vehicles.

2.1 YOLOv3

As stated previously, standard YOLOv3 is the enhanced version of the original YOLO algorithm in both accuracy and speed. It implements Darknet-53 architecture as its backbone and three various scales for detection also named as “output layers” or “heads”. It is superior to methods such as SSD, YOLO, YOLOv2 and Faster RCNN when trained on standardised datasets such as Common Objects in Context (COCO) in terms of performance. Standard Tiny-YOLOv3 is a simplified version of YOLOv3 with less number of convolution layers with higher speed but lower accuracy.

To increase the ability of object detection in standard Tiny-YOLOv3, it was modified by adding an extra output layer to increase the probability of small object detection [2, 14]. An “output layer” in object detection is a layer which provides information such as the coordinates of the object, the object class, and the probability of the detected object (confidence). Standard YOLOv3 and Tiny-YOLOv3

with an extra output layer provide three output layers, each one destined to detect at different scales or ranges of pixels. The first output layer detects big objects, the second is responsible for detection of medium objects and the third is able to detect small objects.

As autonomous vehicles applications require light-weight algorithms to be executed in low-computation and low-powered GPUs, we selected Tiny-YOLOv3 with three output layers because it balances the portability with high accuracy and, after the application of our technique, high speed. Other publications such as [31, 56, 58] also base their work in Tiny-YOLOv3 due to its trade-off.

2.2 Analysis of previous works

This subsection provides an overview of various works in the literature related to techniques used for CNN acceleration or improvements and pedestrian detection in autonomous vehicles.

2.2.1 Convolutional neural networks improvements

Convolutional neural networks are high-demanding algorithms in terms of computational resources, power-consumption and data. Table 1 collects several state-of-the-art

scientific papers based on CNN improvements in terms of speed or accuracy. As a case in point, in some studies such as [38, 46], the accuracy was improved to a large extent in AV use cases; however, the detection was not in real-time. In [23, 26, 51], the PGA accelerator was used to improve the speed which is a costly solution. Besides, in [26], the improvement in execution speed (by 11.5%) was compared against the execution on CPU which is usually the slowest hardware to execute CNNs. In another study, Murakami et al. [51] compared their approach against other software implementations (CPU) while their experiments were run hardware acceleration. In addition, the AlexNet [22] was trained with MNIST [24] dataset to compare the accuracy of their approach with other CNNs. In another study, Yanyan Feng et al. [11] highlighted an improvement in the accuracy by 8.9% on GONET dataset [16]; nevertheless, the speed was not specified. In another approach, Zongyue Wang et al. [52] trained their pruning approach with CIFAR [21] dataset and improved the fps by 1.6s when running on the cloud. Xue et al. [53] also presented a simulated results claiming a total improvement of 43% in accuracy. Zhang et al. [55] proposed a new platform called Caffeine for FPGA hardware, and the execution speed was improved $\times 29$ in performance. [25] and [6] focus their research on vehicle detection which has a key impact in

Table 1 Comparison table of previous work based on common acceleration approaches and pedestrian detection

Ref	Objective	Algorithm	Environment	Platform	Accuracy	FPS	Acceleration
[38]	Traffic sign	F-RCNN+ZF+VGG	NG	NG	+58–78%	No RT	Classifier
[46]	Vehicle logo	Custom CNN	GPU	Matlab	99.1%	No RT	Hyper-parameter
[23]	MNIST	Custom CNN	FPGA	Vivado	98.66%	4938 cc	Hardware
[26]	NG	AlexNet	FPGA	NG	NG	$\times 11.5$	Hardware
[51]	MNIST	NASH-CNN	FPGA	Vivado	“Highest”	$\times 15.35$	Hardware
[11]	GONET	AlexNet	GPU	Caffe	+4–8.9%	NG	Algorithm
[52]	CIFAR	Custom CNN	GPU	PyTorch	Less error	$\times 1.6$	Prune and Cloud
[53]	NG	AlexNet	NG	NG	Total improve	43%	Pipeline
[55]	NG	VGG16+AlexNet	FPGA	Vivado	NG	$\times 29$	Platform
[25]	Vehicles	Tiny-YOLOv3-SPP	GPU	Darknet	+7.2%	63	Algorithm
[6]	Vehicles	Tiny-YOLO-based	Nvidia Xavier	NG	+4.6%	69.74	Algorithm
[40]	Dumpsters	Google inception	NG	Tensorflow	88%	NG	Algorithm
[10]	Pedestrians	RPN + R-FCN	NVIDIA	Caffee	+35%	NG	Algorithm
[54]	Pedestrian	Improved YOLOv2	Nvidia	Caffe	43.33%	20	Algorithm
[34]	Pedestrian	SSD	PC	Caffe	54.4%	24	FPGA
[20]	Pedestrian	Customised CNN	NG	NG	95%	NG	Image classification
[48]	Pedestrian	Customised CNN	NG	NG	73.8%	10+	Algorithm
[15]	Pedestrian	HOG+SVM	ITX board	NG	91.56%	50	Zynq SoC
[27]	Pedestrian	YOLO	NG	NG	87.4%	NG	Hardware
[19]	Pedestrian	YOLO2	PYNQ	NG	79.6%	15.15	ZYNQ-SoC
[4]	Pedestrian	Tiny YOLOv3	Jetson XT2	Tensor flow	NG	31.1	Algorithm
TP	Pedestrian	YOLOv3	Nvidia Xavier	PyTorch	YOLOv3	+48.7%	Algorithm

TP This Paper, RT Real-Time, cc clock cycles, NG Not Given

autonomous driving field. Both, based their development in YOLOv3-based algorithm achieving improvements in the accuracy of at least 7% more. Finally, [35] improves the detection of ships in deep waters by the application of different augmentation techniques for the dataset.

2.2.2 Pedestrian detection

Pedestrian detection has gained attention over the last decade. However, the pedestrian detector is not efficient enough for self-driving applications due to real-time requirements while preserving the accuracy. Therefore, there is a need to accelerate CNNs for self-driving scenarios. This section provides a review of techniques used for CNN acceleration related to pedestrian detection in autonomous vehicles.

Table 1 (below [40]) presents and compares the published results of using different acceleration techniques for pedestrian and dumpsters detection. As a case in point, Moussawi et al [34], used FPGA to accelerate deep learning for pedestrian detection on self-driving cars. In this study, firstly, they improved the SSD-based pedestrian detector by modifying the hyper-parameters in the algorithm and secondly, they used a new FPGA design to accelerate the model on the Altera Arria 10 platform. Using Field Programmable Gate Arrays (FPGAs) are more power-efficient compared to GPUs but more expensive and not easy to be used. In another study, Kim et al. [20] proposed a pedestrian detector to improve the detection speed through reducing the classification computation by the presence or absence of a pedestrian and the estimation position in the pre-processing stage. Although they achieved a very high accuracy performance(95%), the execution speed was not specified. A combination of Squeeznet-like backbone with paralleled dilated convolutions was proposed by Trembl et al. [48] to accelerate the segmentation for AVs. The segmentation accuracy was higher than other networks and fast enough for embedded devices. However, it did not fulfil speed requirements when embedded on computing devices such as NVIDIA Jetson Xavier. Hemmati et al. [15], accelerated the combination of HOG and SVM on ITX board using Zynq SoC which is by using an extra hardware. In [19, 27], an improved hardware design was also used to accelerate the pedestrian detector. Instead of detecting pedestrians, [40] detects dumpsters which is a very typical obstacle in autonomous driving. With their improvement they achieve higher accuracy although not better speed because is not specified. [10, 40] has similar flaws when presenting the results in their manuscript, they show an increment of the accuracy although not of the speed. This may lead to the assumption that they increased accuracy by decreasing speed, which is acceptable for AV use cases. On the opposite approach,

[4, 54] provides an increment of speed, although the accuracy is not given which also leads to the assumption that they tip the balance in favour of the speed by sacrificing the accuracy.

In summary, most of the above studies has used different hardware design to accelerate the object detector which are expensive and not straightforward to be used. In addition, the studies with algorithm modification are not fast enough for our use-case. For the first time, the paper provides a cost effective algorithm modification to minimise unnecessary operations to accelerate the algorithm while maintaining the standard YOLOv3 accuracy perfectly suitable for AV environments.

3 Design

This section defines the proposed design to reduce the complexity of Convolutional Neural Networks without losing accuracy. First, the CNN bottlenecks tackled by our design are delineated. Two algorithms adopted in this design are standard YOLOv3 which is the state-of-the-art in object detection algorithms and standard Tiny-YOLOv3 with an extra output layer to increase the accuracy of small object detection [2, 14]. Our approach was tested on a self-driving scenario using Tiny-YOLOv3 (3l) thanks to its suitability for constrained environments.

3.1 Bottlenecks in convolutional neural networks

CNNs are made of neurons that are able to learn weights and biases. Each layer receives an input matrix and applies filters to extract features. Currently, CNN works with images that are represented in three dimensions (Width \times Height \times Depth (RGB)). Generally, CNN will transform 3D images to an output of 2D feature map of the filter. As this process is computationally expensive, it is defined as one of the bottlenecks which directly affects the speed

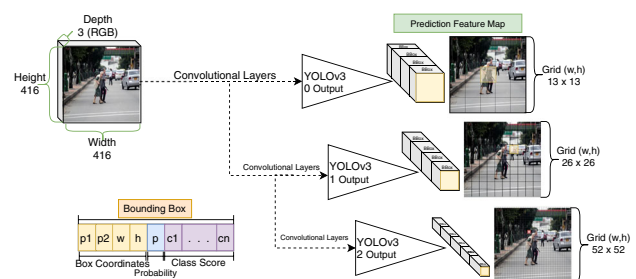


Fig. 1 Presentation of most consuming tasks for a convolutional neural network: number of convolutional layers and number of predictions (GridSize \times GridSize \times 3)

factor. The other consuming task is the number of predictions that the algorithm is able to perform.

Depending on the number of output layers of a CNN-based algorithm, there will be different number of bounding box predictions. Figure 1 shows an image with an standard YOLOv3 input size (416x416px). Over the execution of the network, the image will be down-scaled based the *stride* factor. To make possible different scale detection, the stride will vary for each output layer; the greater the stride, the coarser the output detection grid. Obtaining more grid cells means to have more bounding boxes to predict (usually 3 per grid cell). Thus, performing detection for small objects signifies greater consumption of the resources and thereby lower execution speed. Table 2 shows the stride applied to each output layer and the resultant grid size (also in Fig. 1). Although the total number of predictions are represented in Table 2, the number is theoretical, for instance, many of those 507 predictions able to perform YOLO at the first output layer will detect the same object, therefore, techniques such as Non-Maximum-Suppression are used to select the best prediction. As stated, the number of possible predictions affects the inference time, however, the number of objects detected does not supposes an overload to the algorithm speed. Already being theoretically described in two main bottlenecks of CNNs, Fig. 2 confirms our assumption mathematically. FLOPS (Billion of Floating Point Operations per Second) [1] is a metric commonly used to define the computational complexity of an algorithm; the more BFLOPS, the more computation for the algorithm. Parameters *fsize* and *filter* are directly related to power consumption of convolution operations. Regarding predictions, *out_h*, and *out_w* represent the output grid size (13, 26, 52) depending on the output layer. BFLOPS will be one of the key metrics obtained in Sect. 4) when executing the proposed approach.

Our approach will tackle both bottlenecks mentioned previously to process an image by any type of YOLOv3 algorithm without sacrificing the accuracy while reducing the inference time.

Table 2 Number of predictions regarding each output layer

Size	Layer	Input size	Stride	Grid	Predictions
Big	0	416 × 416	32	13 × 13	507
Medium	1	416 × 416	16	26 × 26	2028
Small	2	416 × 416	8	52 × 52	8112
All	All	416 × 416	All	All	10,647

Predictions = GridSize × GridSize × 3

$$\begin{aligned}
 f &= filters = (\#classes + 5) \times 3 \\
 fsize &: Conv.layer filtersize \\
 c &: channels \\
 out_h &: gridheight \\
 out_w &: gridwidth \\
 BFLOPS &= (2 \times f \times fsize^2 \times c \times out_h \times out_w) / 10^9
 \end{aligned}$$

Fig. 2 Billion of floating point operations per second (BFLOPS) formula

3.2 Network design

Figure 3 illustrated the architecture of YOLOv3 with three output layers. The convolutional layers are shown with red boxes. The operations of the network are shown with arrows. The connections of the convolutional layers are shown in green arrows. The red box indicates the concatenation of YOLO headers in the architecture. Each output layer corresponds to a different scale to detect objects of big (0), medium (1) and small (2) sizes. Due to large number of convolutional layers, the backbone was summarised.

To increase the speed of YOLOv3 algorithm, our design discards the output layers. By applying this approach, the unneeded computation is eliminated. To detect large-scale objects, 59 convolutional layers are used in a feed forward fashion. No up-sampling will occur at this stage and the feature map will be 13 × 13 for an input size of 416. For the medium-scale objects, the discarding will take place at the 79th convolutional layer (circle (A)) shown in yellow. In other words, 80th and 81th convolutional layers will not be considered for detection. This saves computations for the algorithm. Besides, the algorithm will also go through the path with green circle (B) and will be up-sampled (up-sample 1 in Fig. 3) and concatenated with the 61st convolutional layer shown with orange circle (D). Then, the second detection will occur for medium objects. 65 convolutional layers will be evaluated in total for the second output and the feature map will be 26 × 26. In terms of the third output layer for small object detection, the discarding will take place at the 91st convolutional layer in the second branch shown in a blue circle (F). In other words, the discarding will take place at the pink circle (C). The algorithm will go through the second up-sampling in the architecture and will be concatenated with the 36th convolutional layer (circle blue (E)) in the backbone and the feature map will be 52 × 52. Using the discarding technique, 71 convolutional layers with two up-sampling operations are considered against 75 convolutional layers of standard YOLOv3 which slow down the network. In addition, by using this technique, the number of predictions is reduced as some output layers are not considered.

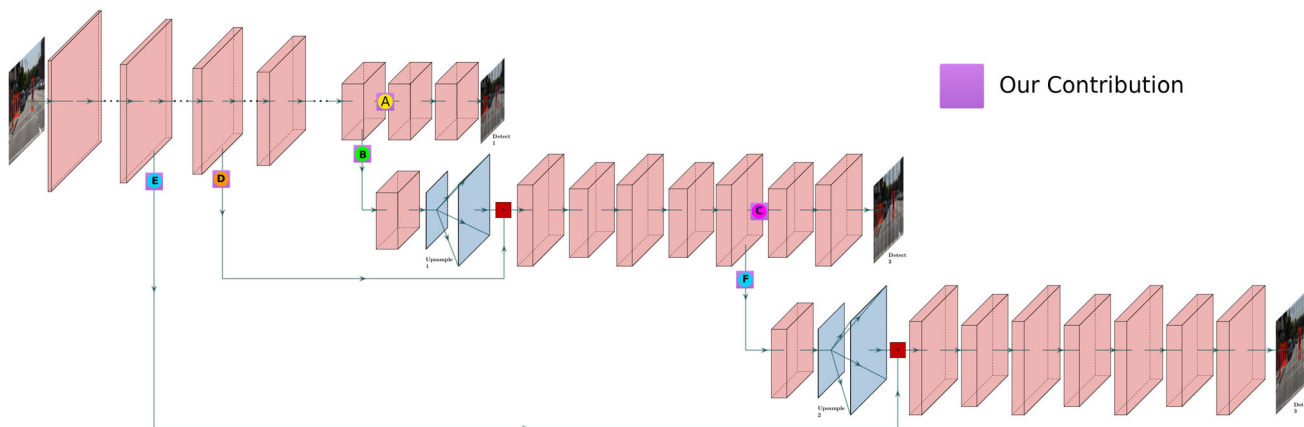


Fig. 3 Design of YOLOv3 with 3 detection layers applying discarding technique. Each colour references where the discarding is done (see Table 4)

Table 3 YOLOv3 configuration after discarding technique

Output	Conv. layer	Upsample	Grid size	Output size
0	59	0	13	202,800
1	65	1	26	811,200
2	71	2	52	3,244,800
0,1	67	1	13,26	1,014,000
0,2	73	2	13,52	3,447,600
1,2	73	2	26,52	4,056,000
All	75	2	13,26,52	4,258,800

Output size = GridSize × GridSize × 3 (#classes + 5)

Table 4 Discarding technique acceleration

YOLOv3 and Tiny-YOLOv3 Acceleration Technique by Discarding Output Layers						
Output Layer	Detection Sizes	A Yellow	B Green	C Purple	D Orange	E/F Blue
0	Big	●	○	○	○	○
1	Medium	○	●	●	●	○
2	Small	○	●	○	●	●
0,1	Big-Med	●	●	●	○	○
0,2	Big-Sma	●	●	○	●	●
1,2	Med-Sma	○	●	●	●	●
All	All	●	●	●	●	●

A summary of final number of convolutional layers for each output layer based on our approach is shown in Table 3. As it can be observed, not only the results of discarding technique were illustrated based on output layers 0, 1 and 2, but also the results of the discarding technique were while others will be evaluated when using a combination of 0,1, 0,2 and 1,2 output layers. Finally,

standard execution of YOLOv3 (not using the discarding technique) is defined as “all”. In addition, the number of up-sampling and the number of predictions was also highlighted in each case. The Standard COCO dataset with 80 classes [28] was considered to calculate the number of predictions [17]. This value will change proportionally with the number of classes used for training.

In order to clarify our approach, Table 4 explains the discarding technique in a simple way by assuming the discarding circles in Fig. 3 as ON/OFF switches (Boolean symbols). If the switch is ON, the convolutional layers will forward the data over that branch. Nevertheless, if the switch is OFF, YOLOv3 will forward the data to other possible paths. For instance, if the circle A is ON, the big human detection will take place. If circles B, C and D are on, the medium human detection will take place and will go through the second branch of the architecture. For small human detection, it will go through the path with circle B, D, E and F. Similarly, the same will happen for dual-scale detection. Without taking the acceleration technique into consideration, all the circles from A to F will be included in the detection.

The proposed approach is based on discarding the paths that YOLOv3 uses to fully execute the CNN-based algorithm. By applying this discarding technique, there is no need to train various models for different scales or output layers. Furthermore, there are no requirements for loading different configurations of CNN-based algorithms in memory and thereby different weights. All these aforementioned advantages are feasible based on the user-demand. If the final user wants to change the output layer, some switches will be ON, while others will remain OFF.

3.3 Network design for tiny-YOLOv3(3l)

Figure 4 illustrates the discarding technique applied to Tiny-YOLOv3 with 3 output layers (3l). In the backbone of standard Tiny-YOLOv3, the original image was down-sampled 6 times with a stride of 2 using max-pooling operations. The number of filters for 3×3 convolution in the backbone is 16, 32, 64, 128, 256, 512, and 1024. As previously stated, our technique is applicable to different types of YOLOv3. Thus, if only the first output of Tiny-YOLOv3 wants to perform detection, just the switch A should be ON. Nevertheless, to detect at second scale, B, C, D must be ON, and the others off. Finally, to detect small objects (3rd output), the algorithm should leave OFF the A and C switches.

Even though the number of convolutional layers is less in Tiny-YOLOv3(3l) than YOLOv3 and the discarding technique would have less effect on the execution speed, it makes a difference when used in constrained environments such as a smartphone or NVIDIA Jetson. Table 5 shows the number of convolutional layers activated when the corresponding output layer(s) is executed. While the number of convolutional layers are reduced, the number of predictions remains constant. However, even with the same number of predictions, less convolutional layers are processed which results in less computational cost. The results of applying the discarding technique on Tiny-YOLOv3 (3l) is summarised in Table 4.

This study focuses on the standard configuration, nevertheless, other researches have increased or decreased YOLO output layers according to their necessities. This is not an issue for the proposed technique, in fact, our optimisation is applicable to any YOLO-based configuration. For instance, the paper demonstrates that the technique is applicable to YOLOv3 and Tiny-YOLOv3 which were selected because their three output layers correspond to object detection at short, medium and large distances.

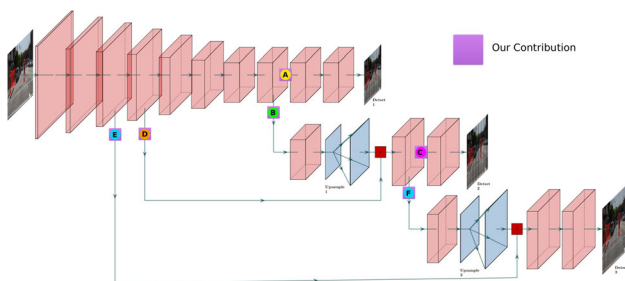


Fig. 4 Design of Tiny-YOLOv3 with 3 detection layers applying dynamic discarding technique

Table 5 Tiny-YOLOv3 configuration after discarding technique

Output L.	Conv. layer	Upsample	Grid size	Output size
0	10	0	13	202,800
1	11	1	26	811,200
2	13	2	52	3,244,800
0,1	13	1	13,26	1,014,000
0,2	15	2	13,52	3,447,600
1,2	14	2	26,52	4,056,000
All	16	2	13,26,52	4,258,800

Output size = GridSize x GridSize x 3 x (#classes + 5)

4 Results

This section is divided into three subsections. First, the experimental setup and the execution environment is explained. Second, the results of applying the dynamic discarding technique on standard YOLOv3 are shown. Finally, the dynamic discarding technique was applied on Tiny-YOLOv3 (3l) in a real environment as a use case for Autonomous Vehicles (AVs) to show the advantages of deploying our approach.

4.1 Experimental setup

As stated previously, both YOLOv3 and Tiny-YOLOv3 (3l) are the baselines to evaluate the discarding technique. As a reference, these algorithms were trained on the standard COCO dataset. To execute CNN models, multiple devices are available in the market. As a case in point, Google TPUs [13] are used as a co-processor to train CNN models on low-power devices. Moreover, the Qualcomm Snapdragon Mobile Platform is a high-end system on chip (SoC) which can be used for machine learning purposes on smartphones and tablets [39]. NVIDIA Jetson Xavier [36] is an off-the-shelf low-power consumption embedded system with 512-core Volta GPU with Tensor Cores which runs on Linux and more than 21 (Tera Operations Per Second) TOPS of computation performance and 32GB of memory RAM. It also supports different modes such as 10 W, 15 W and 30 W (MAXN) to vary the power consumption. Its power capability as well as its portability has tilted the balance of deploying our system on it.

Five state-of-the-art Machine Learning frameworks were examined to select a suitable execution environment for the evaluation of the proposed discarding technique. TensorFlow Lite [47], TensorRT [37], OpenCV [3], PyTorch [18], Snapdragon [39] were deployed and studied taking into account the compatibility with the NVIDIA GPU and with our acceleration technique. The latter is tested based on the method used to execute the YOLO

model on the platform. If the platform just loads and execute the model files (network configuration + weights file), the discarding technique will not have an effect as the network configuration cannot be modified in real-time. However, if the platform allows the implementation of the YOLO network and just loads the training weights, it would take advantage of the dynamic discarding approach. Table 6 compares these ML frameworks, while the compatibilities are displayed in green, the incompatibilities are marked in red. The result reveals that our technique cannot be run on Jetson GPU when using TensorRT and OpenCV frameworks. This drawback hinders the advantage of using a GPU. When it comes to TensorRT framework, the implementation of our discarding technique was not possible due to platform constraints. Snapdragon SDK is another studied framework where the deployment of the discarding technique was not feasible. Although it was not the focus of our study, it is the state-of-the-art in computation for smartphones. Thus, PyTorch was used to deploy the discarding acceleration technique. The project used to implement the discarding technique can be found at [18].

The camera used is a “Logitech Carl Zeiss” able to record at HD 1080p at 30 frames per second. It also has a 90°Field of View (FOV). Finally, all the system was deployed on a “Toyota Prius Plug-in Excel 2018”.

4.2 Pruned-YOLOv3: analysis and results

This subsection focuses on the experiments carried out after implementing and applying our acceleration technique on YOLOv3 in PyTorch. As YOLOv3 is a computationally intensive algorithm, all these results are obtained setting the NVIDIA Jetson Xavier on 30W (MAXN mode). As previously stated, our technique does not compromise the accuracy of the model because it merely removes the unneeded operations of the neural network. Hence, the total accuracy of the model is 55.3 mAP for COCO dataset as stated by the author [17].

Figure 5 shows the cumulative average of the time (inference time) taken to process 1000 frames (416x416px) executing YOLOv3. The time was calculated from the

Table 6 Study of different Machine Learning Framework for implementation of the dynamic discarding technique on YOLOv3

Platform	Exec.Environment	Discarding	Deployment
TF Lite	<i>Jetson CPU</i>	<i>NO</i>	Medium
TensorRT	Jetson GPU	<i>NO</i>	Medium
OpenCV	<i>Jetson CPU</i>	YES	Low
PyTorch	Jetson GPU	YES	Medium
Snapdragon	<i>Qualcomm GPU</i>	<i>NO</i>	Medium

Italic values—Not compatible; **Bold values**—Compatible

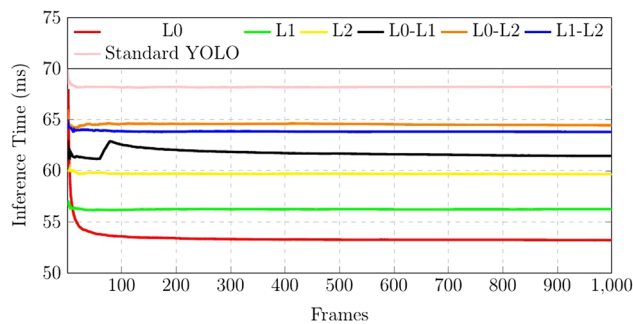


Fig. 5 Speed for each output layer of YOLOv3 or any combinations of output layers against YOLOv3 standard execution. Experiments run in NVIDIA Jetson Xavier at 30 watts

input of the CNN to its output. Seven different experiments were carried out to evaluate the discarding technique: for each output layer, for combination of two output layers and the execution of the standard YOLOv3 without applying the discarding technique. Table 7 shows the results of each output layer and a combination of them. It demonstrates the results based on each output layer depending on the number of convolutional layers executed for each option. It also illustrates the obtained Billion of Floating Point Operations per Second (BFLOPS) of each experiment. The result showed at least 5.5% improvement in speed when executing two output layers. Spotlighting the execution of one output layer, the discarding technique can achieve an improvement of 22% in speed when compared to the standard execution where no discarding technique was utilised.

4.3 Implementation of the approach to a real use case

The previous subsection examines the results obtained after applying our approach to standard YOLOv3. Nevertheless, this algorithm is not suitable for constrained and/or portable environments; As a result, as introduced in Design (Sect. 3), we will benefit from using Tiny-YOLOv3 with 3 output layers. This subsection will apply our approach in a use case for AVs. Nevertheless, this work is not just focused on the AVs field, our approach could also be applied to collision avoidance in ship traffic on rivers, motor vehicles in warehouses, paragliders or drones [31, 32]. The discarding technique is able to choose between different output layers in order to detect pedestrians in real-time, and subsequently take proper action to avoid collision. Although this publication focuses on pedestrians, the system is applicable to any type of objects. Our car was equipped with three basics components: a camera set at the front, a proximity sensor and an NVIDIA Jetson Xavier. The process works as follow: first, the proximity sensor captures the information regarding the distance to the object. Then, the algorithm will choose the

Table 7 YOLOv3 results for different output layers after applying dynamic discarding technique

Layer	Conv.layer	FPS	BFLOPS	Improvement (%)
0	59	18.79	54.436	22
1	65	17.78	58.379	17.6
2	71	16.76	62.410	12.5
0,1	67	16.28	60.062	9.9
0,2	73	15.51	64.093	5.5
1,2	73	15.67	64.181	6.5
All	75	14.66	65.864	0 Base reference

correspondent output layer based on the obtained distance to detect whether the obstacle is a pedestrian. These three different layers correspond to three various scales, the first layer (layer 0) for detecting person being close to the mounted camera on AV, the second layer (layer 1) for person detection at medium scale and the third layer (layer 2) for small person recognition. Figure 6 shows this use case based on human distance from the vehicle and the required time for detection.

To achieve the aim of the study, Tiny-YOLOv3(3l) was trained on COCO dataset with mostly ground-level images. For training, the algorithm was trained setting number of iterations to 160,000. The initial learning rate was set to 0.001. We utilised Stochastic Gradient Descent with Warm Restarts (SGDR) [30] solver with momentum coefficient of 0.9 as learning policy. Finally, we also used a weight decay of 0.0005 to avoid over-fitting. The result of the discarding algorithm was tested with different input sizes including 416, 608, 832 and 1056. For small object detection which also means far distances, the greater the input size the higher the accuracy and lower the execution speed.

Table 8 shows the results of execution speed on different output layers or their combinations. These results are obtained in MAXN mode. The specified BFLOPS calculates the computation needed to execute the algorithm

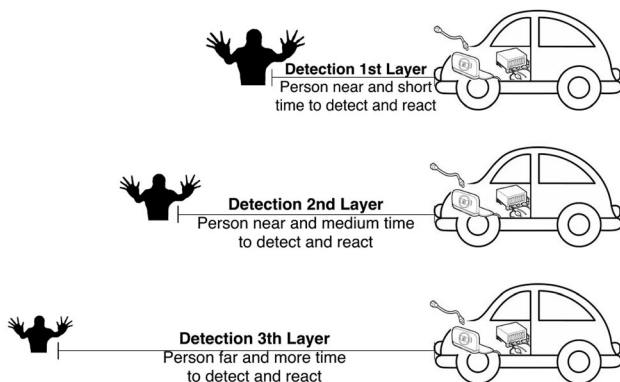


Fig. 6 Discarding technique use case for pedestrian recognition in AVs

regarding the input size and output layers. In addition, the speed improvement was shown when applying our approach in comparison to that of standard Tiny-YOLOv3 when no discarding technique was applied. The result reveals that the FPS was improved with the discarding technique for all the individual detection outputs.

Figure 7 illustrates the speed of Tiny-YOLOv3 versus different input sizes at 30 watts (MAXN) mode of NVIDIA Xavier. As apparent from the results, the speed decreases as the input size increases. Due to the trade-off between speed and accuracy, 832 pixels were chosen as input size for further optimisation and testing of our use case. As it can be observed for input resolution of 832 pixels, the speed for each individual output layer exceeds real-time (30 FPS).

As previously stated, NVIDIA Jetson Xavier supports several modes based on the input power. Different

Table 8 Tiny-YOLOv3 results for different output layers after applying discarding technique on NVIDIA Jetson Xavier in MAXN mode

Input	Layer	FPS	BFLOPS	Improvement (%)
416	0	111.6	4.275	48.7
	1	100	5.128	42.8
	2	88.4	6.855	35.2
	0,1	73.2	5.571	27.8
	0,2	67.7	7.298	15.4
	1,2	69.3	6.944	17.4
	All	57.3	7.386	0 Base reference
	608	0	42	9.132
1		41.3	10.954	32.8
2		40.6	14.643	31.6
0,1		34.7	11.899	19.9
0,2		35.8	15.589	22.6
1,2		35.6	14.832	22
All		27.7	15.778	0 Base reference
832		0	42.2	17.101
	1	39.3	20.511	33
	2	34.6	27.421	23.9
	0,1	32.2	22.283	18.2
	0,2	28.3	29.192	7
	1,2	30	27.774	12.2
	All	26.3	29.546	0 Base reference
	1056	0	29.8	27.548
1		27.3	33.043	35.1
2		22.2	44.174	20.3
0,1		22.8	35.896	22.3
0,2		19.1	47.028	7.1
1,2		19.9	44.743	11.1
All		17.7	47.596	0 Base reference

consumption requirements mean different results on speed when running any machine learning algorithms. Figure 8 shows the results of our approach at different modes: 10, 20 and 30 watts. Besides, the results of executing standard Tiny-YOLOv3-3l with all output layers was compared to the results of each output layers being executed individually. As apparent, our solution outperforms the standard execution in terms of speed. In addition, at 20 watts mode, we would still achieve real-time performance if our camera was recording at 24 fps for output layer 0 and 1. Nevertheless, if we increase the frames per second we will increase also the chances to detect. Two straightforward approaches could be taken to increase even more the speed. First, to reduce the input size of the network, for instance, with an input size of 416×416 our discarding technique surpasses 60 fps threshold. Second, invest in high-spec GPU without power limitations.

The last but not least is to figure out the distance corresponds to each output layer. To measure the distance that each output layer is able to detect, we varied the distance between the camera and the person step-wise by 0.5 m. Being aware of the fact that the pedestrian detection not only depends on the person but also many other factors such as training dataset, weather conditions, the position of the person, clothing, obstacles, we were still able to estimate the required distance to test our use case. Table 9 displays the minimum and maximum distance in meters and the pixel size that the current configuration led to. Based on the obtained results, the minimum size of a human to be detected in the first layer (layer 0) is 67×176 at a maximum distance of 5.5 m. The maximum pixel size of human that can be detected at this layer is 703×460 when the distance is 0.5 m. The minimum size of a human in the second layer (layer 1) is 12×53 at the distance of 31.5 m. For layer 2, the minimum pixel size of human that

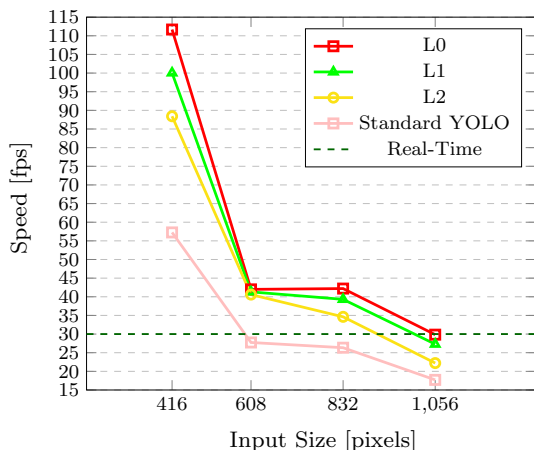


Fig. 7 Speed for Tiny-YOLOv3 model for each output layer depending on the image input size against the standard execution of YOLOv3. Experiments run in NVIDIA Jetson Xavier at MAXN mode

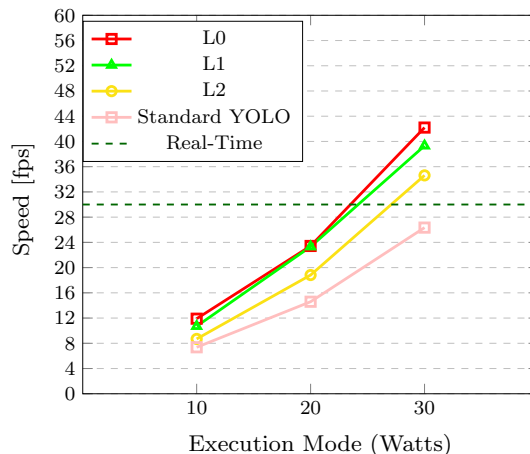


Fig. 8 Speed for Tiny-YOLOv3 model for each output layer at 832 pixels of input size based on the execution mode

can be detected is 9×28 when the distance is 57.5 m. Due to the overlapped distance ranges between different output layers, it is worth considering the execution of two output layers simultaneously.

To evaluate the effectiveness of our approach in preserving the accuracy, a testing dataset was manually collected and labelled from videos recorded in a completely new environment, composed by images never seen before by the algorithm. The testing dataset was divided into three subdatasets based on different distances: short-range (0.5–5.5 m), mid-range (4.5–31.5 m) and long-range (29–57.5 m).

Table 10 compares the standard YOLOv3 (0, 1, 2) executing the three output layers concurrently against the execution of our approach discarding output layers. Each output layers will be tested over the correspondent distance dataset. For short-range distances, the accuracy remains constant when executing all three output layers or just the first output layer when using the discarding technique. For medium-range distances, the accuracy is also preserved no matter which output layers are executed. Finally, for long distances, the aforementioned factor of overlapping distances should be taken into account. Although the execution of just the output layer 2 lowers the accuracy, executing the combination of both output layers 1 and 2,

Table 9 Human detection for each YOLOv3 output layer

Layer	Input Size	FPS	Distance (m)		Pixel Size	
			Min	Max	Max	Min
0	832	42.2	0.5	5.5	703×460	67×176
1	832	39.3	4.5	31.5	77×212	12×53
2	832	34.6	29	57.5	16×63	9×28

Table 10 Accuracy at different distances per execution layer

Layers	Short distance (%)	Mid distance (%)	Long distance (%)
0,1,2	79.9	99.01	98.01
0	79.45	–	–
1	–	98.92	–
2	–	–	82.08
0,1	79.9	99.01	–
1,2	–	98.7	97.01

[0,1,2] is the standard execution of YOLO.

**(a)** First output layer detection of a human at 1.5m of distance.**(b)** Second output layer detection of a human at 22m of distance.**(c)** Third output layer detection of a human at 50.5m of distance.**Fig. 9** Tiny-YOLOv3-3l detection using dynamic discarding technique based on three output layers at different altitudes and environments

mitigates the overlapping issue and thereby preserves the accuracy.

These accuracy results demonstrate that our technique preserves the accuracy of the standard model by dynamically changing the output layers. If the pedestrian is in an “overlapped” or inconsistent distance, the decision to take is executing the two nearest output layers at the same time.

4.4 Qualitative results

Figure 9 shows three pedestrian images. These images were taken from a camera mounted on the car. Each image contains a pedestrian at a different distance and thereby enabling our system to detect using different output layers. Fig. 9a shows a pedestrian at 1.5 m of distance to the car and was detected at output layer 0 based on the results of Table 9. As apparent in Fig. 9b, the pedestrian was detected at output layer 1 due to 22 m of distance to the car. Finally, Fig. 9c shows an accurate detection of a pedestrian at 50.5 m of distance due to the output layer 2 performing the detection.

5 Conclusions

In this paper, we proposed a new acceleration technique to perform efficient deep object detection through output discarding of convolutional layers. To this end, we enforced the discarding to have fewer trainable parameters in comparison with original YOLOv3 and Tiny-YOLOv3, as a promising solution to speed up the neural network while preserving the accuracy. Empirical results show that in the case of YOLOv3, the speed was improved by at least 5.5% when executing two output layers. By executing one output layer, the proposed dynamic discarding technique can achieve a very significant improvement of 22% in speed when compared to the standard execution where no discarding technique is performed. When using Tiny-YOLOv3 with three output layers on a small-form Jetson Xavier PC, the proposed technique can achieve a remarkable maximum improvement of 48.7% in speed. Future work will focus on the improvement of human detection at different conditions to estimate a better range of pixels to recognise them.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing,

adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Alexey AB (2020) Billion floating point operations per second, BFLOPS formula. https://github.com/AlexeyAB/darknet/blob/5b6be00d4b1fd671c20c4c72d2239c924eaa3d4/src/convolutional_layer.c#L406
- Alexey AB (2020) Darknet. <https://github.com/AlexeyAB>
- Bradski G (2000) The OpenCV Library. Dr. Dobb's J: Softw Tools for the Professional Programmer 25(11):120–123
- Chen WH, Kuo HY, Lin YC, Tsai CH (2021) A lightweight pedestrian detection model for edge computing systems. In: Y. Dong, E. Herrera-Viedma, K. Matsui, S. Osumu, A. González Briones, S. Rodríguez González (eds.) Distributed computing and artificial intelligence, 17th international conference. Springer International Publishing, Cham, pp 102–112
- Cheng Y, Wang D, Zhou P, Zhang T (2017) A survey of model compression and acceleration for deep neural networks. Preprint [arXiv:1710.09282](https://arxiv.org/abs/1710.09282)
- Choi J, Chun D, Lee H, Kim H (2020) Uncertainty-based object detector for autonomous driving embedded platforms. In: 2020 2nd IEEE international conference on artificial intelligence circuits and systems (AICAS), pp 16–20. <https://doi.org/10.1109/AICAS48895.2020.9073907>
- Cortes C, Vapnik V (1995) Support-vector networks. *Mach Learn* 20(3):273–297
- Dai J, Li Y, He K, Sun J (2016) R-FCN: object detection via region-based fully convolutional networks. <http://arxiv.org/abs/1605.06409>
- Dalal N, Triggs B (2005) Histograms of oriented gradients for human detection. In: 2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05). IEEE, vol. 1, pp 886–893
- Ding L, Wang Y, Laganière R, Huang D, Fu S (2020) Convolutional neural networks for multispectral pedestrian detection. *Signal Process Image Commun* 82:115764
- Feng Y, Zeng S, Yang Y, Zhou Y, Pan B (2018) Study on the optimization of CNN based on image identification. In: 2018 17th international symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES). IEEE, pp 123–126
- Girshick R (2015) Fast R-CNN. Proceedings of the IEEE international conference on computer vision 2015 inter, pp 1440–1448. <https://doi.org/10.1109/ICCV.2015.169>
- (2020) Google: Cloud Tensor Processing Units (TPUs). <https://cloud.google.com/tpu/docs/tpus>
- He W, Huang Z, Wei Z, Li C, Guo B (2019) TF-yolo: an improved incremental network for real-time object detection. *Appl Sci* 9(16):3225
- Hemmati M, Biglari-Abhari M, Niar S (2019) Adaptive vehicle detection for real-time autonomous driving system. In: 2019 Design, Automation and Test in Europe Conference and Exhibition (DATE). IEEE, pp 1034–1039
- Hirose N, Sadeghian A, Vázquez M, Goebel P, Savarese S (2018) Gonet: a semi-supervised deep learning approach for traversability estimation. In: 2018 IEEE/RSJ international conference on Intelligent Robots and Systems (IROS). IEEE, pp 3044–3051
- Joseph R (2020) YOLO real time object detection. <https://pjreddie.com/darknet/yolo>
- Kathuria A (2020) PyTorch YOLOv3 implementation. <https://github.com/ayoozhkathuria/pytorch-yolo-v3>
- Kim H, Choi K (2019) The implementation of a power efficient bcn-based object detection acceleration on a xilinx FPGA-SOC. In: Proceedings—2019 IEEE international congress on cybernetics: 12th IEEE international conference on internet of things, 15th IEEE international conference on green computing and communications, 12th IEEE international conference on cyber, physical and social computing and 5th IEEE international conference on smart data, iThings/GreenCom/CPSCom/SmartData 2019, pp 240–243. <https://doi.org/10.1109/iThings/GreenCom/CPSCom/SmartData.2019.00060>
- Kim J, Jung WY, Jung H, Han DS (2018) Methodology for improving detection speed of pedestrians in autonomous vehicle by image class classification. In: 2018 IEEE international conference on consumer electronics, ICCE 2018-Jan, pp 1–2. <https://doi.org/10.1109/ICCE.2018.8326252>
- Krizhevsky A, Hinton G et al (2009) Learning multiple layers of features from tiny images. University of Toronto
- Krizhevsky A, Sutskever I, Hinton GE (2017) ImageNet classification with deep convolutional neural networks. *Commun ACM* 60(6):84–90. <https://doi.org/10.1145/3065386>
- Kyriakos A, Kitsakis V, Louropoulos A, Papatheofanous EA, Patronas I, Reisis D (2019) High performance accelerator for cnn applications. In: 2019 29th international symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS). IEEE, pp 135–140
- LeCun Y, Cortes C (2010) MNIST handwritten digit database. <http://yann.lecun.com/exdb/mnist/>
- Li Q, Garg S, Nie J, Li X, Liu RW, Cao Z, Hossain MS (2020) A highly efficient vehicle taillight detection approach based on deep learning. *IEEE transactions on intelligent transportation systems*, pp 1–11. <https://doi.org/10.1109/TITS.2020.3027421>
- Li S, Wen W, Wang Y, Han S, Chen Y, Li H (2017) An FPGA design framework for CNN sparsification and acceleration. In: 2017 IEEE 25th annual international symposium on Field-Programmable Custom Computing Machines (FCCM). IEEE, p 28
- Lin SC, Zhang Y, Hsu CH, Skach M, Haque ME, Tang L, Mars J (2018) The architectural implications of autonomous driving: constraints and acceleration. *ACM SIGPLAN Not* 53(2):751–766. <https://doi.org/10.1145/3173162.3173191>
- Lin TY et al (2014) Microsoft COCO: common objects in context. In: Fleet D, Pajdla T, Schiele B, Tuytelaars T (eds) Computer vision – ECCV 2014. ECCV 2014. Lecture Notes in Computer Science, vol 8693. Springer, Cham. https://doi.org/10.1007/978-3-319-10602-1_48
- Liu W, Anguelov D, Erhan D, Szegedy C, Reed S, Fu CY, Berg AC (2016) SSD: Single shot multibox detector. In: Lecture notes in computer science (including subseries Lecture Notes in artificial intelligence and lecture notes in bioinformatics) 9905 LNCS, pp 21–37. https://doi.org/10.1007/978-3-319-46448-0_2
- Loshchilov I, Hutter F (2019) SGDR: stochastic gradient descent with warm restarts. In: 5th international conference on learning representations, ICLR 2017—conference track proceedings, pp 1–16
- Martinez-Alpiste I, Golcarenrenji G, Wang Q, Alcaraz Calero JM (2020) Altitude-adaptive and cost-effective object recognition in an integrated smartphone and uav system. In: 2020 European

- Conference on Networks and Communications (EuCNC), pp 316–320
32. Martínez-Alpiste I, Casaseca-de-la Higuera P, Alcaraz-Calero JM, Grecos C, Wang Q (2019) Smartphone-based object recognition with embedded machine learning intelligence for unmanned aerial vehicles. *J Field Robot*. <https://doi.org/10.1002/rob.21921>
 33. Miethig B, Liu A, Habibi S, Mohrenschildt MV (2019) Leveraging thermal imaging for autonomous driving. In: ITEC 2019—2019 IEEE transportation electrification conference and expo. <https://doi.org/10.1109/ITEC.2019.8790493>
 34. Moussawi A, Haddad K, Chahine A (2018) An FPGA-accelerated design for deep learning pedestrian detection in self-driving vehicles. <http://arxiv.org/abs/1809.05879>
 35. Nie X, Yang M, Liu RW (2019) Deep neural network-based robust ship detection under different weather conditions. In: 2019 IEEE Intelligent Transportation Systems Conference (ITSC), pp 47–52. <https://doi.org/10.1109/ITSC.2019.8917475>
 36. (2020) NVIDIA: NVIDIA Jetson Xavier. <https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit>
 37. (2020) NVIDIA: NVIDIA TensorRT. <https://developer.nvidia.com/tensorrt>
 38. Qiao K, Gu H, Liu J, Liu P (2017) Optimization of traffic sign detection and classification based on faster r-cnn. In: 2017 International Conference on Computer Technology, Electronics and Communication (ICCTEC), pp 608–611. IEEE
 39. (2020) Qualcomm: snapdragon neural processing engine SDK. <https://developer.qualcomm.com/docs/snpe/overview.html>
 40. Ramírez Díaz I, Cuesta-Infante A, Pantrigo J, Montemayor AS, Moreno J, Alonso V, Anguita G, Palombarani L (2020) Convolutional neural networks for computer vision-based detection and recognition of dumpsters. *Neural Comput Appl*. <https://doi.org/10.1007/s00521-018-3390-8>
 41. Redmon J, Divvala S, Girshick R, Farhadi A (2016) You only look once: unified, real-time object detection. In: Proceedings of the IEEE computer society conference on computer vision and pattern recognition 2016-Dec, pp 779–788. <https://doi.org/10.1109/CVPR.2016.91>
 42. Redmon J, Farhadi A (2017) YOLO9000: Better, faster, stronger. In: Proceedings—30th IEEE conference on computer vision and pattern recognition, CVPR 2017 (2017-Jan), 6517–6525. <https://doi.org/10.1109/CVPR.2017.690>
 43. Redmon J, Farhadi A (2018) YOLOv3: an incremental improvement. <http://arxiv.org/abs/1804.02767>
 44. Ren S, He K, Girshick R, Sun J (2017) Faster R-CNN: towards real-time object detection with region proposal networks. *IEEE Trans Pattern Anal Mach Intell* 39(6):1137–1149. <https://doi.org/10.1109/TPAMI.2016.2577031>
 45. Schapire RE (2013) Explaining AdaBoost. In: Schölkopf B, Luo Z, Vovk V (eds) *Empirical Inference*. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-41136-6_5
 46. Soon FC, Khaw HY, Chuah JH, Kanesan J (2018) Hyper-parameters optimisation of deep CNN architecture for vehicle logo recognition. *IET Intell Transp Syst* 12(8):939–946
 47. TensorFlow: tensorflow lite (2019). <https://www.tensorflow.org/lite/guide>
 48. Trembl M, Arjona-Medina J, Unterthiner T, Durgesh R, Friedmann F, Schuberth P, Mayr A, Heusel M, Hofmarcher M, Widrich M, Nessler B, Hochreiter S (2016) Speeding up semantic segmentation for autonomous driving
 49. Van Etten A (2018) You Only look twice: rapid multi-scale object detection in satellite imagery. <http://arxiv.org/abs/1805.09512>
 50. Viola P, Jones MJ (2004) Robust real-time face detection. *Int J Comput Vision* 57(2):137–154
 51. Vu TH, Murakami R, Okuyama Y, Abdallah AB (2018) Efficient optimization and hardware acceleration of cnns towards the design of a scalable neuro inspired architecture in hardware. In: 2018 IEEE international conference on big data and smart computing (BigComp). IEEE, pp 326–332
 52. Wang Z, Lin S, Xie J, Lin Y (2019) Pruning blocks for CNN compression and acceleration via online ensemble distillation. *IEEE Access* 7:175703–175716
 53. Xue C, Cao S, Jiang R, Yang H (2018) A reconfigurable pipelined architecture for convolutional neural network acceleration. In: 2018 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, pp 1–5
 54. Yang Z, Li J, Li H (2018). Real-time pedestrian detection for autonomous driving. In: 2018 International Conference on Intelligent Autonomous Systems (ICoIAS). IEEE, pp 9–13
 55. Zhang C, Sun G, Fang Z, Zhou P, Pan P, Cong J (2018) Caffeine: toward uniformed representation and acceleration for deep convolutional neural networks. *IEEE Trans Comput Aided Des Integr Circuits Syst* 38(11):2072–2085
 56. Zhang P, Zhong Y (2019) Li X (2019) Slimyolov3: narrower, faster and better for real-time UAV applications. In: IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)
 57. Zhang Q, Zhang M, Chen T, Sun Z, Ma Y, Yu B (2019) Recent advances in convolutional neural network acceleration. *Neurocomputing* 323:37–51
 58. Zhao H, Zhou Y, Zhang L, Peng Y, Hu X, Peng H, Cai X (2020) Mixed yolov3-lite: a lightweight real-time object detection method. *Sensors* 20:1861. <https://doi.org/10.3390/s20071861>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.