Journal of
Low Power Electronics and Applications

Article

Efficient GEMM Implementation for Vision-Based Object Detection in Autonomous Driving Applications

Fatima Zahra Guerrouj [1,2,*], Sergio Rodríguez Flórez [1], Mohamed Abouzahir [2], Abdelhafid El Ouardi [1] and Mustapha Ramzi [2]

[1] SATIE Laboratory, Université Paris-Saclay, ENS Paris-Saclay, Scientific Research National Center (CNRS), Av. des Sciences Bâtiment 660, 91190 Gif-sur-Yvette, France; sergio.rodriguez@universite-paris-saclay.fr (S.R.F.); abdelhafid.elouardi@universite-paris-saclay.fr (A.E.O.)
[2] Systems Analysis, Information Processing and Industrial Management Laboratory, Higher School of Technology of Salé, Mohamed V University in Rabat, Rabat 8007.N.U, Morocco; mohamed.abouzahir@est.um5.ac.ma (M.A.); mustapha.ramzi@um5.ac.ma (M.R.)
* Correspondence: fatima-zahra.guerrouj@universite-paris-saclay.fr

Abstract: Convolutional Neural Networks (CNNs) have been incredibly effective for object detection tasks. YOLOv4 is a state-of-the-art object detection algorithm designed for embedded systems. It is based on YOLOv3 and has improved accuracy, speed, and robustness. However, deploying CNNs on embedded systems such as Field Programmable Gate Arrays (FPGAs) is difficult due to their limited resources. To address this issue, FPGA-based CNN architectures have been developed to improve the resource utilization of CNNs, resulting in improved accuracy and speed. This paper examines the use of General Matrix Multiplication Operations (GEMM) to accelerate the execution of YOLOv4 on embedded systems. It reviews the most recent GEMM implementations and evaluates their accuracy and robustness. It also discusses the challenges of deploying YOLOv4 on autonomous vehicle datasets. Finally, the paper presents a case study demonstrating the successful implementation of YOLOv4 on an Intel Arria 10 embedded system using GEMM.

Keywords: YOLOv4; GEMM; FPGA; autonomous driving

Citation: Guerrouj, F.Z.; Rodríguez Flórez, S.; Abouzahir, M.; El Ouardi, A.; Ramzi, M. Efficient GEMM Implementation for Vision-Based Object Detection in Autonomous Driving Applications. *J. Low Power Electron. Appl.* **2023**, *13*, 40. https://doi.org/10.3390/jlpea13020040

Academic Editor: Andrea Calimera

Received: 18 April 2023
Revised: 20 May 2023
Accepted: 31 May 2023
Published: 6 June 2023

1. Introduction

In Advanced Driver Assistance Systems (ADAS), the object detection model should be able to detect various types of objects, such as vehicles, pedestrians, cyclists, and road signs, in a variety of conditions, such as day and night, rain, and snow. Thus, it is essential to choose an accurate and fast model to help the driver to make safer decisions and to avoid accidents [1]. However, several algorithms are proposed to detect objects, including single-shot detector (SSD) [2], Faster R-CNN [3], Region-based Convolutional Neural Networks (R-CNN) [4], and You-Only-Look-Once (YOLO) [5]. Among these models, YOLOv4 has achieved high accuracy and speed on various object detection tasks, outperforming the other models. Moreover, FPGAs are becoming more prevalent in ADAS due to their ability to rapidly develop complex systems and accurate data processing from multiple sources in real-time. FPGAs also require less power than traditional microprocessors, which makes them ideal for ADAS because they can be used to reduce system power consumption [6]. In addition, FPGAs are cost-effective and can be adapted to meet the needs of different ADAS applications.

Our research aims to evaluate the robustness and effectiveness of YOLOv4 on large autonomous driving datasets. Comprehensive analysis, performance evaluation, and dedicated FPGA architecture design will advance ADAS technologies and facilitate efficient and accurate object detection. The main contributions of this paper are outlined as follows:

- Comprehensive analysis: We conduct a thorough investigation of the algorithmic complexity of YOLOv4 on the KITTI and Self Driving Car datasets, considering

various input sizes. This analysis provides a detailed understanding of the algorithm's computational requirements and helps identify potential areas for optimization.
- Performance evaluation: In addition to analyzing the algorithm's complexity, we evaluate the performance of YOLOv4 on self-driving datasets. This evaluation encompasses accuracy assessment, where we quantify the detection precision and recall rates achieved by YOLOv4 on the autonomous driving datasets.
- FPGA-based GEMM implementation: To optimize the computational efficiency of YOLOv4, we propose a novel FPGA architecture specifically designed for implementing the GEMM operation within the algorithm.

This paper is organized in the following sections. Section 2 introduces the related work. Section 3 presents the autonomous driving datasets and provides an overview of the YOLOv4 algorithm and its architecture. Section 4 outlines the experiment setting and describes the evaluation metrics employed to assess the performance of the algorithm. Section 5 presents the results of the study and offers an analysis of the findings. In Section 6, we propose an FPGA-based GEMM for YOLOv4 and evaluate its effectiveness on two autonomous driving datasets. Finally, we summarize our findings and draw some outlook directions for future research in Section 7.

2. Related Work

Several implementations of CNN and GEMM for YOLOv4 have been proposed to fulfill high ADAS requirements. Most of the implementations are based on the PyTorch framework and exploit the power of GPUs for fast inference and training. Among these implementations, the paper [7] proposes a new methodology for the efficient execution of CNN inference on embedded CPU-GPUs MPSoCs. This methodology takes full advantage of task (pipeline) and data-level parallelism in a CNN to obtain high-throughput execution. This proposed methodology is evaluated by mapping real-world CNNs onto an NVIDIA Jetson MPSoC with integrated CPU-GPUs. Their results show 20% higher throughput than the leading TensorRT DL framework for NVIDIA Jetson MPSoCs. Similarly, Ref. [8] discusses adapting the YOLOv4 neural network detector architecture to multispectral pedestrian detection, evaluating its accuracy and efficiency in automotive applications on Nvidia RTX 3080. Experiments on the KAIST dataset demonstrate that YOLOv4 can achieve high accuracy while retaining low latency of 30 ms and low false negatives, making it suitable for real-time decision-making. The experiments also show that thermal and visible light images can be merged in the modern YOLOv4 architecture without additional computational cost, assuming the two images are aligned and synchronized. Another implementation on an embedded GPU was done by [9], which introduces a new object detection model, YOffleNet, based on YOLOv4, but with a much lighter network architecture that uses ShuffleNet modules instead of CSP DenseNet. Experiments with the KITTI dataset showed that YOffleNet is compressed by 4.7 times, achieving a speed of 46 FPS on an embedded GPU system (NVIDIA Jetson AGX Xavier). Despite the high compression ratio, the accuracy is only slightly reduced to 85.8% mAP, which is only 2.6% lower than YOLOv4. Although, GPUs can provide inference times that meet the needs of advanced driver assistance systems (ADAS), they consume a significant amount of energy, which can be prohibitive in embedded CNN applications [10].

Thus, alternative hardware solutions and optimization techniques are needed to ensure that the power consumption of the embedded CNNs remains within acceptable limits while meeting the performance requirements of ADAS. In particular, an optimized version of YOLOv4 has been proposed by [11], which presents hardware acceleration of the YOLOv4 object detection algorithm on the Xilinx Zynq-7000 system-on-chip (SoC). The proposed implementation enables efficient resource utilization of 43.6% of LUTs, 43.04% of Flip-flops, 82.17% of BRAMs, 79% of DSPs, 189.14 GOP/s throughput, and 10.36 W power consumption. The implementation also achieves a frame rate of 30 FPS with the MS-COCO reference dataset, but does not provide the evaluation accuracy of the algorithm.

The Vivado HLS tool is used to synthesize and implement the object detection algorithm on the SoC.

A different work for accelerating convolution operations is proposed by [12]. This work proposes an FPGA-based architecture to accelerate convolution operations to provide AI functionality to edge devices. The architecture is implemented in Verilog and can be used with different FPGA devices. It can compute simultaneously on multiple feature channels and filter cores, increasing computational parallelism. Their experiences show that the architecture can achieve 0.224 GOPS on a single core when used on a single edge computing FPGA board and 4.48 GOPS when fully utilized. Likewise, this paper [13] explores neural-network-based FPGA acceleration by accelerating general matrix multiplication (GEMM). They propose a design that exploits 8-bit quantization to improve bandwidth while preserving model accuracy, leveraging FPGAs for model parallelization and data reuse for high-performance, low-latency neural network inference. The design is implemented on Xilinx-based Zynq SoCs and tested on the MNIST dataset. The proposed method produces faster acceleration while maintaining nearly identical performance to the unquantized method.

Additionally, the paper [14] presents a low-memory GEMM algorithm for CNN inference on Xilinx Virtex UltraScale+ VU9P FPGAs. The authors design a new accelerator based on systolic networks and a three-level cache to increase parallelism and save bandwidth. The design is evaluated on MobileNet V1 and Inception V4, and the results show excellent performance in terms of throughput and latency with a reduction in memory usage of 21–44%. Additionally, an automated framework for implementing hardware-accelerated DNN architectures is proposed by [15] on Xilinx ZYNQ SoCs for Edge computing applications. The framework combines the scalability of custom hardware with optimization strategies to improve performance per watt while maintaining accuracy. The proposed framework provides an end-to-end solution that facilitates the efficient deployment of topologies on FPGAs. Experiments show that the architectures developed by the framework provide the best trade-off between performance, power consumption, and system cost. As a result, the low-power DNN topologies generated for the MNIST database achieved a high throughput of 3626 FPS and power consumption of only 0.266 W.

Much of the research on FPGA-based accelerators focusing on efficiency and cost-effectiveness by utilizing lightweight object detection models and small datasets. These approaches aim to simplify development and enable real-time applications. However, our research takes a distinct approach by emphasizing the utilization of the full-weight YOLOv4 object detection algorithm on extensive autonomous driving datasets. This choice enables us to explore the algorithm's suitability for Advanced Driver Assistance Systems applications demanding high detection accuracy. The outcomes of our work hold paramount importance, as they will yield invaluable insights into the suitability of YOLOv4 for Advanced Driver Assistance Systems (ADAS) applications demanding high detection accuracy. By focusing on the robustness and efficacy of YOLOv4 on large-scale autonomous driving datasets, our research aims to significantly contribute to the advancement of ADAS technologies and their practical implementation.

3. Materials and Methodology

In autonomous driving applications, object detection models are typically based on deep-learning neural networks. Object detection is an essential part of autonomous driving, enabling autonomous cars to recognize and react to their environment. One of the most challenging aspects of autonomous driving is detecting and recognizing objects on the road, including vehicles, pedestrians, traffic signs, and traffic lights [16]. Several object detection models exist in the literature, including YOLO, SSD, and Faster RCNN, among others. However, to determine the most suitable model for autonomous driving applications, it is essential to evaluate the accuracy of these models. This paper presents an evaluation of several state-of-the-art object detection models on the COCO dataset. We compare the models using the mean Average Precision (mAP) metric and present the results to determine the most accurate model for the given tasks. In addition, we focus on the model that provides the most satisfactory performance on the COCO dataset. Additionally, we evaluate it over two popular autonomous driving datasets, KITTI, and Self Driving Car.

3.1. Vision-Based Object Detection Algorithm

Object detection models use a combination of object classifiers and object localizers to accurately identify and localize objects within an image or video [17]. Convolutional neural networks (CNNs) are used to train these models to recognize and localize objects of specific types, such as cars, people, and animals, within an image or video. Object detection models are applicable to various applications, including self-driving cars, automated surveillance systems, facial recognition systems, and medical image analysis [18]. In this experiment, we evaluate different object detection models using DL Workbench, to facilitate assessing and using these models [19]. Table 1 presents information on several object detection models. We evaluated the mAP of these models on the Deep Learning Workbench framework released by Intel. This framework contains pre-trained object detection and classification models. It allows users to evaluate model accuracy and inference time using virtual CPUs and GPUs on multiple datasets, such as COCO, a benchmark dataset commonly used for object detection tasks, including a large number of images with a variety of objects in different contexts. The mAP metric computes the average accuracy for each object class and averages the average accuracy of all classes. The third column refer to the number of parameters, which is the sum of all weights and biases of the neural network. For example, YOLOv4 has over 60 million trainable parameters. The results show that the YOLO series models outperform the other models on the COCO dataset, and YOLOv4 is the most accurate, with an mAP of 77.40%. This signifies that the YOLO models are very effective for object detection on this dataset. Furthermore, YOLOv4 has gained a significant accuracy improvement over previous versions of YOLO and other object detection models, thus making it a prominent option for autonomous driving applications.

Table 1. Performance analysis of object detection models on COCO.

Object Detection Models	Year	N° of Parameters ($\times 10^6$)	Model Size (MB)	Backbone	mAP %
RetinaNet	2017	65.13	146	Resnet + FPN [1]	33.15
Rfcn-resnet-101	2016	53.46	206	Resnet-101	28.40
SSD-mobilenet-v1	2017	6.80	27	Mobilenet	23.33
SSD-mobilenet-v1-fpn	2017	36.18	119	Mobilenet + FPN	35.54
SSD-mobilenet-v2	2019	16.81	65	Mobilenet	24.95
SSDlite-mobilenet-v2	2018	4.47	18	Mobilenet	24.29
SSD-resnet50-v1-fpn	2017	56.93	198	Resnet-50 + FPN	35.01
Faster-RCNN-resnet-50	2015	29.16	112	Resnet-50	30
Faster-RCNN-resnet-101	2015	48.12	185	Resnet-101	35.72
Faster-RCNN-inception-v2	2015	13.30	52	Inception	26.24
YOLO-v2	2016	50.95	195	Darknet-19	53.15
YOLO-v2-tiny	2016	11.23	43	Darknet-19	29.11
YOLO-v3	2018	61.92	237	Darknet-53	67.7
YOLO-v3-tiny	2018	8.84	33	Darknet-19	33.1
YOLO-v4	2020	64.33	247	Cspdarknet-53	77.40
YOLO-v4-tiny	2020	6.05	24	Cspdark-53-tiny	46.3

[1] Feature Pyramid Networks.

3.2. Datasets

The KITTI dataset [20] is well-known for training and testing autonomous driving algorithms. It includes images, labels, and calibration data acquired from the onboard camera, lidar, and radar sensors of a vehicle driven on German public roads. The collection includes 7481 images of various locations, such as city streets, highways, and rural locations. In addition, the dataset structures road participants in eight different classes (car, van, truck, pedestrians, cycle, tram, and misc.). Calibration information is required to convert sensor values to real-world coordinates. The dataset is suitable for a wide range of applications, including object detection, semantic segmentation, and 3D reconstruction.

The Self Driving Car dataset [21] comprises driving in Mountain View, California, and surrounding cities during daylight hours. It contains 97,942 labels divided into 11 classes, such as biker, car, pedestrian, traffic lights, trafficLight-Green, trafficLight-GreenLeft, trafficLight-Red, trafficLight-RedLeft, trafficLight-Yellow, trafficLight-YellowLeft, truck, and 15,000 images captured by a Point Grey camera at full HD resolution (1920 × 1200 px) at 2 Hz.

The KITTI and Self Driving Car datasets are ideally suited to evaluate the performance of YOLOv4 in the autonomous driving domain. These datasets contain images of environments, such as highways and city streets, which is common in autonomous driving applications. In addition, they provide many annotated images, which can be used to train and accurately evaluate the object detection capabilities of YOLOv4. Furthermore, the images in these datasets are specifically designed to simulate real-world driving scenarios as shown in Figures 1 and 2. They can be used to evaluate the accuracy of YOLOv4 under various driving conditions. We summarize the number of classes and the data structure of the two datasets in Table 2.

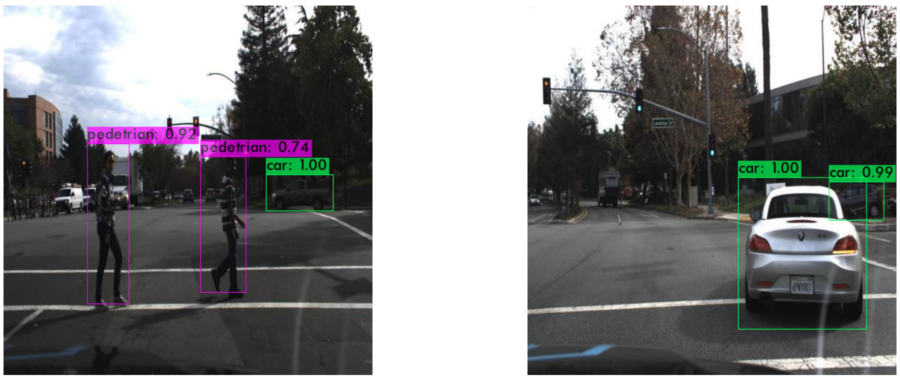


Figure 1. Self Driving Car dataset images.

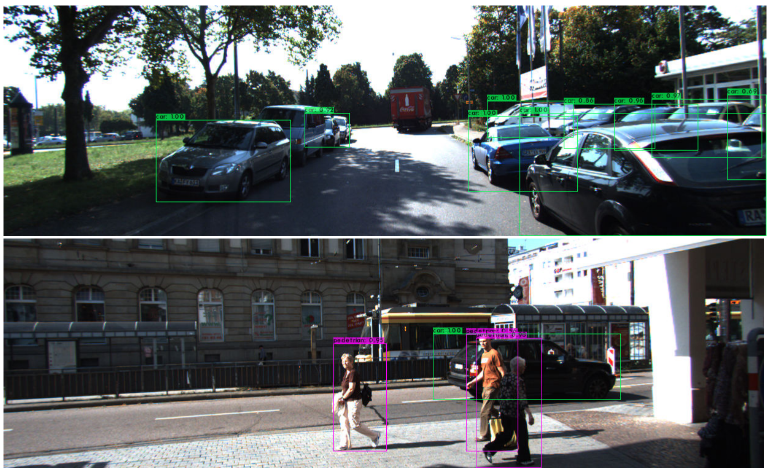


Figure 2. KITTI dataset images.

Table 2. Datasets description.

Dataset	Training Set	Testing Set	Total	Class
KITTI [20]	5237	2244	7481	8
Self Driving Car [21]	10,500	4500	15,000	11

3.3. YOLOv4 Workflow

YOLOv4 has several advantages over other models. First and foremost, YOLOv4 is more accurate than the majority of other models. It has achieved a mean average precision of 77.40 on COCO, which is significantly higher than other models. Second, YOLOv4 is also more efficient than most other models. It can process up to 65 frames per second on a single GPU and up to 18 frames per second on a single CPU [22], making YOLOv4 much faster than other models. Finally, YOLOv4 is also more flexible than other models. It is an excellent choice for various applications, especially for autonomous driving.

YOLOv4 is a state-of-the-art object detection architecture developed by [22]. It is based on the well-known YOLOv3 architecture [23] and is a powerful deep-learning technique for object detection. The main difference between YOLOv3 and YOLOv4 is the addition of the SPP-Net and PANet layers that help to improve the accuracy of the model. The YOLOv4 workflow proceeds through three main components, as shown in Figure 3 the feature extractor, the neck, and the head. The YOLOv4 workflow starts with the input image, which is pre-processed to a standard size, and then goes through a feature extractor as the backbone of the network to extract the feature maps. It consists of a new network architecture called CSPDarknet-53, which contains 53 convolutional layers. Adding these layers improves the accuracy and speed of the model. Afterward, the neck processes the features extracted from the feature extractor, consisting of convolutional layers, batch normalization, and pooling. Finally, the head is the final output layer that contains the classification and regression heads for feature detection. It is typically a fully connected layer. Moreover, the model applies non-maximum suppression to these predictions to remove redundant detections and refine the final set of bounding boxes. In addition, YOLOv4 includes various advanced techniques to improve its performance, including a weighted loss function, spatial pyramid pooling, and a Mish activation function.

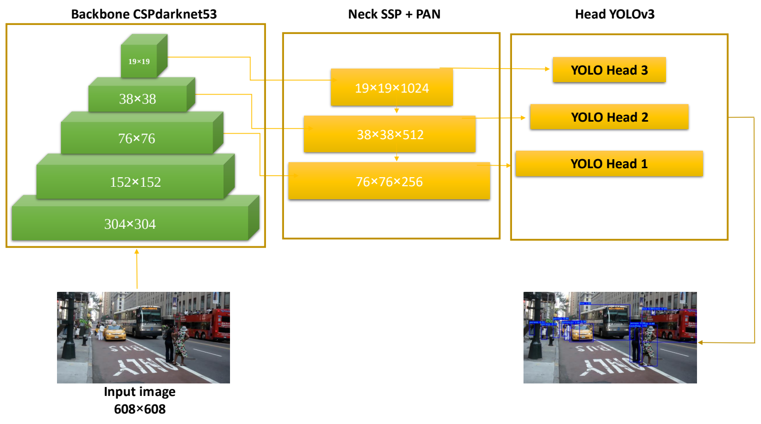


Figure 3. YOLOv4 architecture.

4. Experiment

YOLOv4 is a version of the YOLO family of object detection models [24]. Several metrics are used to evaluate the performance of the YOLOv4 model, such as mAP, precision, and recall [25]. The mAP is a popular metric to measure the precision of object detection models at different confidence levels, calculating the average precision–recall curve. Among the positive detections, precision measures the number of true positives detected. Conversely, recall measures how many true positives the model detects out of all the true positive instances.

For autonomous driving scenarios, accuracy in object detection is critical for ensuring passenger safety. We must, therefore, rigorously evaluate YOLOv4's performance metrics to ensure they meet the required level of accuracy.

4.1. Experimental Settings

YOLOv4 uses the Adam optimizer, a stochastic gradient descent algorithm using adaptive learning rates to update the model parameters. A learning rate 0.0013 was selected to ensure gradient descent converged toward the optimal solution [26]. To address weight decay, we adopted a weight attenuation value of 0.0005 from previous works such as [27–29]. We utilized momentum with a default value of 0.949, as is typical in most deep learning models. This technique was implemented to expedite the optimization process. Additionally, we fixed the batch size to 64, which is adequate for GPU training. A total of 6000 iterations were chosen to establish stability and prevent overfitting. Through experimentation, we observed that the accuracy of the model reaches a stable state beyond this point. We aimed to balance reliable accuracy and avoid overfitting by setting an appropriate iteration limit. Finally, the input size was fixed at 608 $\times$ 608 px to maximize model precision. These carefully selected hyperparameters allowed us to achieve optimal model performance, as demonstrated by our results. The model was trained on an Nvidia Quadro RTX 6000 GPU with a total memory of 24 GB and 4608 CUDA cores. The instructions for implementing YOLOv4 on custom data are available at [22]. For the training phase, the darknet framework was used. It is an open-source neural network framework based on C and CUDA. In this experiment, we use the following metrics: mean average precision, precision, recall, and average precision (AP).

4.2. Evaluation Metrics

Precision [30], recall [30], average precision, and mean average precision were employed to evaluate the model accuracy performance. The above metrics are calculated as follows:

The average precision metric is the weighted average of the precision scores achieved at each threshold on the PR curve, with an increase in recall over the previous threshold used as a weighting factor. Average precision is high when precision P and recall R are high, and low when one or the other is low over a range of confidence threshold values. AP varies between 0 and 1.

$$AP = \sum_{k=0,0.1,0.2,\ldots,1} [R(k) - R(k-1)] * P(k) \tag{1}$$

where k refers to the threshold, and P and R refer to precision and recall, respectively.

The mean average precision is obtained by taking the average of the APs of all the classes.

$$mAP = \frac{1}{m}\sum_{i=1}^{m} AP_i \tag{2}$$

The range of i starts with 1 to m (m is the number of classes).

5. Performance Evaluation

Using YOLOv4 in autonomous driving scenarios requires evaluating its relevance on specific autonomous driving datasets under different resolutions. In addition to evaluating its accuracy on autonomous driving datasets, it is also essential to measure the processing time of the model to identify the most computationally intensive operations. This information is essential to understand the model's performance and improve its efficiency for practical applications.

5.1. Experimental Results on KITTI and Self Driving Car Dataset

Yolov4 was thoroughly evaluated on several benchmark datasets and achieved excellent accuracy. On the MS COCO object detection dataset [31], Yolov4 achieved a mean average precision of 45.6%. On the Open Images V4 dataset [32], Yolov4 achieved an average accuracy score of 39.7%. Alternatively, Yolov4 was evaluated on the PASCAL VOC dataset, achieving an mAP of 71.9% [33].

In this work, we evaluate Yolov4 on two autonomous car datasets with three classes relevant to autonomous navigation (pedestrian, bicycle, car) under different input resolutions 608 × 608 px, 512 × 512 px, and 320 × 320 px.

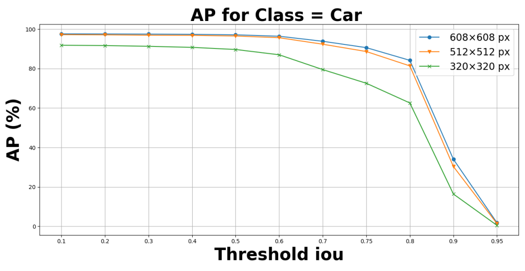
Table 3 shows the performance of YOLOv4 on the KITTI and Self-Driving Car datasets at three different resolutions. The results reveal that the model performs well overall, with an mAP of up to 89.4% at the highest resolution, 608 × 608 px. However, the results vary considerably depending on the resolution used. The model achieved a mAP of 74.18% and 87.51% at the lowest resolution of 320 × 320 px and the highest resolution of 512 × 512 px, respectively. This result indicates that the model is more efficient when it receives more information, as higher resolution allows for more accurate object detection. Furthermore, the model detected objects in the KITTI dataset with high accuracy, as demonstrated by the mAP score.

Table 3. The mean average precision (mAP) of the KITTI and Self Driving Car datasets at the three levels of resolution.

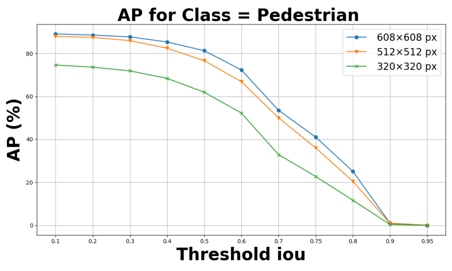
Datasets	**KITTI**			**Self Driving Car**		
Resolution	608 × 608	512 × 512	320 × 320	608 × 608	512 × 512	320 × 320
mAP (%)	89.40	87.51	74.18	58.19	57.61	47.15

For evaluating the Self Driving Car dataset, the model achieves an mAP of 58.19%, 57.61%, and 47.15% at a resolution of 608 × 608 px, 512 × 512 px, and 320 × 320 px, respectively. The results suggest that the model does not yield better performance. Several different factors might cause this. First, as mentioned in Section 3, the resolution of this dataset is 1920 × 1200 px, and the current version of YOLOv4 may not be equipped to handle higher-resolution images. The model may not have sufficient capacity to process higher-resolution images without a significant increase in computational cost. Second, the dataset includes many duplicate bounding boxes for the same object, resulting in poor performance.

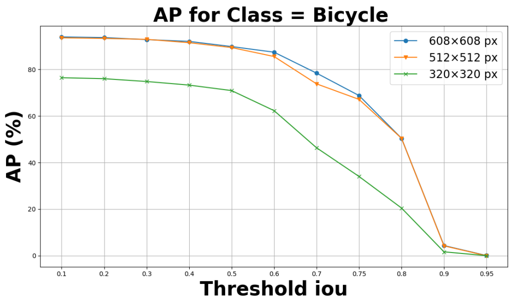
Furthermore, the performance evaluation for the three classes, as shown in Figure 4, shows that the model on KITTI with a resolution of 608 × 608 px achieves an average accuracy (AP) of 97.15% for car detection, 81.25% for pedestrian detection, and 89.79% for bicycle detection. Similarly, for the self-driving car (see Figure 5), the model achieves an average accuracy (AP) of 77.85%, 51.05%, and 45.66% for car, pedestrian and bicycle detection, respectively. We can notice that the class "Car" is more accurate than the other classes due to the fact that it occupies a large number of labels in both databases. In addition, when reducing the input size from 608 × 608 px to 320 × 320 px, we observed a proportional decrease in the average accuracy for all three classes. In the case of the KITTI dataset, as shown in Figure 4, the average accuracy for the car class decreased by approximately 7.44%, resulting in a value of 89.71%. Additionally, for the pedestrian class, the average accuracy decreased by about 19.31% to reach 61.94%, while for the bicycle class, it decreased by about 18.9% to reach 70.89%. Similarly, for the Self Driving Car dataset, as shown in Figure 5, we observed a decrease in the average accuracy for the car class by about 5.93%, resulting in a value of 71.92%. The pedestrian class experienced a decrease of approximately 14.39% to 36.66%, and the bicycle class decreased by approximately 12.79% to 32.87%. These results highlight the impact of scaling the input size on the accuracy performance of the model. The results indicate that the model is able to accurately identify multiple classes simultaneously, making it a viable option for multi-class object detection tasks.



(**a**)

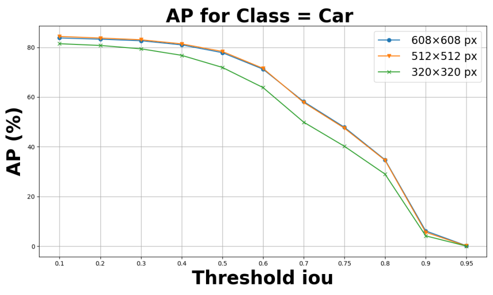


(**b**)

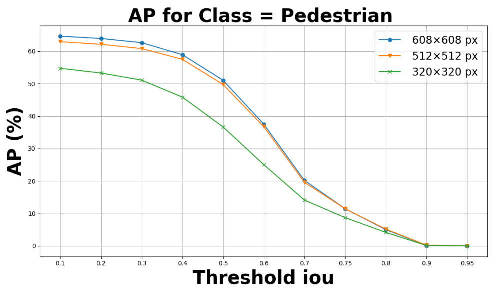


(**c**)

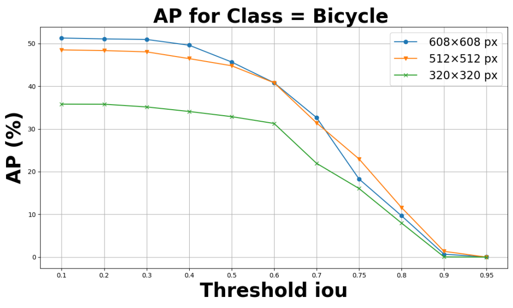
Figure 4. Average Precision (AP) of the three classes, car, pedestrian and bicycle at different resolutions. In blue 608 × 608 px, in orange 512 × 512 px and in green 320 × 320 px on KITTI dataset. (**a**) Class = Car, (**b**) Class = Pedestrian, (**c**) Class = Bicycle.



(**a**)



(**b**)



(**c**)

Figure 5. Average Precision (AP) of the three classes with different resolutions. In blue 608 × 608 px, in orange 512 × 512 px and in green 320 × 320 px on Self Driving Car dataset. (**a**) Class = Car, (**b**) Class = Pedestrian, (**c**) Class = Bicycle.

Overall, YOLOv4 performed better on the KITTI dataset with the three different resolutions tested than on the Self-Driving Car dataset. Using the highest resolution improved performance, as expected. However, YOLOv4 is not suitable for all autonomous driving datasets.

5.2. Processing Time Evaluation

YOLOv4 extracts features from an input image using a series of convolutional layers, which are followed by a batch normalization layer, which reduces overfitting. Yolov4 also uses Mish activation, a hybrid of ReLU6 and Softplus activation functions. These operations are used to predict the objects in the image and compute the confidence score of each object. In this work, we examine the primary operations of YOLOv4 to identify the most computationally intensive operations and propose an optimization on a high-performance FPGA using high-level synthesis. We divide YOLOv4 into different functional blocks to evaluate the processing time of each layer, such as the convolution layer, batch normalization layer, max-pooling layer, activation functions layer, and other layers. We conducted this experiment on an Intel Xeon workstation with 12 cores, 24 threads, and a frequency of 3.50 GHz. The graph below shows the processing time in milliseconds (ms) consumed for each operation in YOLOv4.

As depicted in Figure 6, with a 608 × 608 px input, YOLOv4 takes up to 2177.066 ms to detect objects in the image, where the convolution layer takes about 1381.71 ms, the batch normalization 38.49 ms, the activation functions 305.23 ms, and the remaining time belongs to the max-pooling layers and other operations such as such as maximum-non-suppression and cIOU. Convolution is very compute-intensive, especially when dealing with large images. It takes up to 65% of the total time of the algorithm, followed by "Other" operations, which take up the remaining time. Processes such as YOLO and max-pooling prove to be relatively light. In YOLOv4, the GEMM method is used to accelerate the computation of the convolution operation. It works by decomposing a large matrix multiplication into smaller operations. Thus, the calculations can be performed in parallel, which reduces the total time of the operation. It also reduces memory requirements by allowing the reuse of intermediate results.

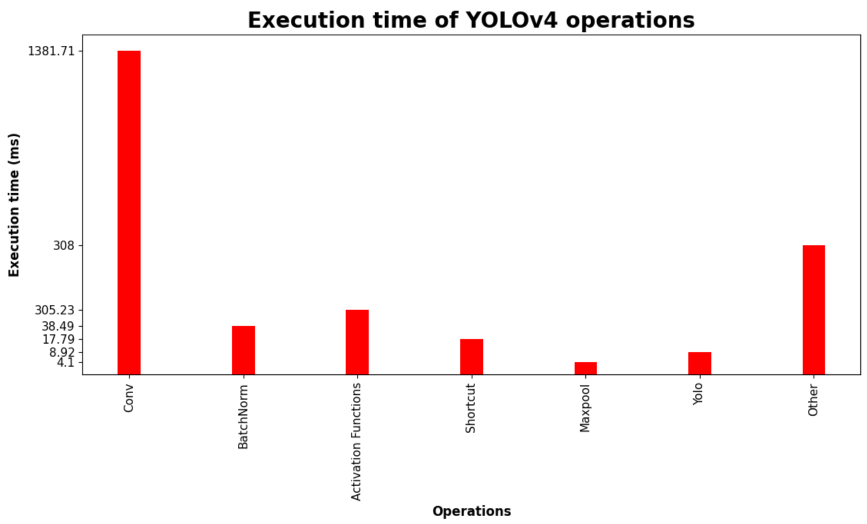


Figure 6. Execution time of YOLOv4 operations.

6. Accelerator Design

Accelerating the GEMM operation of YOLOv4 on FPGA is a promising way to improve the performance of the algorithm. FPGAs can provide higher computational throughput, lower power consumption, and lower latency than CPUs and GPUs. This can be achieved through high-level synthesis tools, custom hardware accelerators, and optimization techniques such as pipelining and loop unrolling.

The FPGA-based GEMM acceleration architecture for YOLOv4 is proposed in Figure 7 to improve the object detection speed. It involves using the direct memory access (DMA) to transfer a feature map of size HxWxC and the associated weight parameters of size KxKxCxM to the on-chip memory. The feature map is then extended into C vectors using the im2col function, and the results are sent to the Processing Units (PE) for efficient parallel multiplication and accumulation. Following this, the results are converted back into a matrix of MxHxW as the output feature map, which can then be used for further analysis or processing.

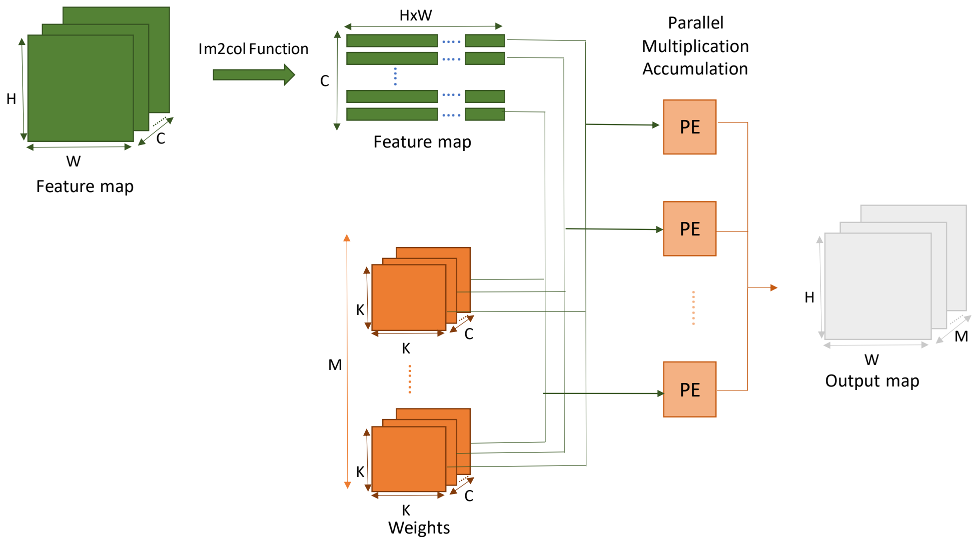


Figure 7. FPGA-based GEMM architecture.

Results and Discussion

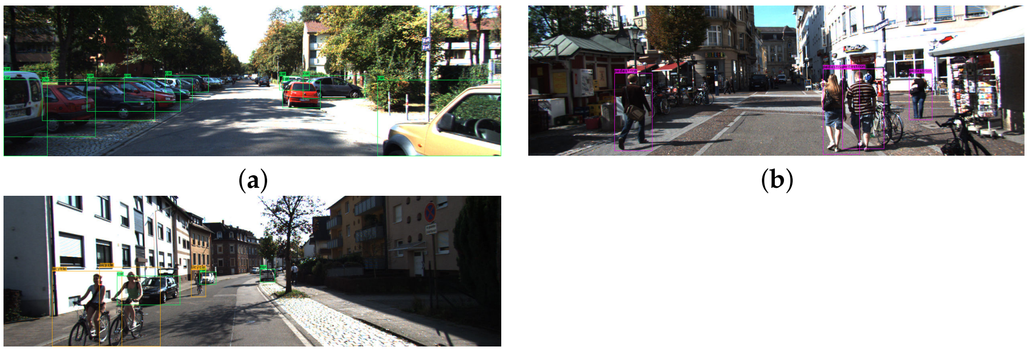
YOLOv4 GEMM operations are implemented on DE5a-Net DDR4, enabling high-speed data processing with high-performance Intel Arria 10 FPGAs. The board includes 16 GB of DDR4 memory running at over 75 Gbps, allowing data transfer up to 7.876 Gbps over PCIe Gen 3 $\times$ 8 between the board and the host PC [34]. In addition to these features, the host PC is equipped with an 8-core Intel Xeon Silver 4108 processor that runs at a basic frequency of 1.8 GHz. The host has 32 GB of DDR4-2666 ECC SDRAM memory (2 $\times$ 16 GB).

Table 4 shows the resource usage for the GEMM implementation on FPGA. Resource utilization includes using logic resources such as LUTs, registers and RAM blocks, and DSP blocks. From the data in the table, it can be seen that the GEMM implementation on FPGA is relatively efficient regarding resource utilization. The utilization of logic resources such as LUTs and registers is relatively low, at about 12% and 12%, respectively. DSP block utilization is about 6%, while memory utilization is relatively high, at about 19%.

Table 4. Resource usage summary.

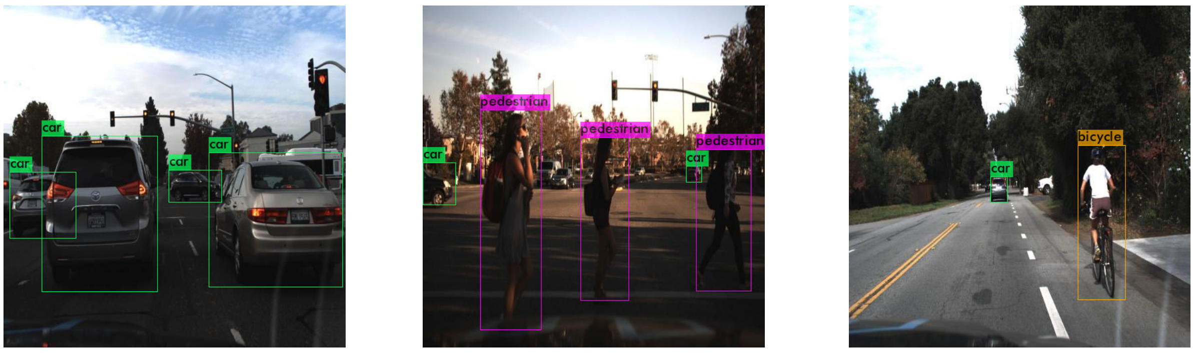
Resource	Usage
Logic utilization	24%
ALUTs	12%
Dedicated logic registers	12%
Memory blocks	19%
DSP blocks	6%

To evaluate the effectiveness of our FPGA-based approach, we conducted experiments on two popular object detection datasets: KITTI, as shown in Figure 8, and Self Driving Car, as shown in Figure 9. YOLOv4 convolution layers are implemented on a DE5a-Net DDR4 with a high-performance Intel Arria 10 FPGA. The communication between the FPGA and the host is done via PCIe. The convolution process is performed on the FPGA, and the results are transferred to the host PC to continue the YOLOv4 process, and then Figures 8 and 9 are displayed. Our results indicate that our FPGA architecture is well-suited for object detection tasks and can efficiently detect and classify objects such as cars, pedestrians, and bicycles. Overall, our results highlight the potential of FPGA-based implementations to perform effective object detection in real-world applications.



(**a**) (**b**) (**c**)

Figure 8. Detecting autonomous vehicle objects using FPGA-based GEMM of YOLOv4 on KITTI dataset. (**a**) Cars, (**b**) pedestrians, (**c**) bicycles.



(**a**) (**b**) (**c**)

Figure 9. Detecting autonomous vehicle objects using FPGA-based GEMM of YOLOv4 on Self Driving Car dataset. (**a**) Cars, (**b**) pedestrians, (**c**) bicycles.

Furthermore, the results of our proposed GEMM implementation on FPGA are promising and demonstrate its effectiveness in terms of performance. As shown in Figure 10, with a resolution of 608 × 608 px, 512 × 512 px, and 320 × 320 px, our GEMM implementation achieved a mAP of 89.40%, 87.51%, and 74.18% on KITTI, respectively—and 58.19, 57.61, and 47.15% on the Self Driving Car dataset. Moreover, the performance evaluation for the three classes, as illustrated in Figure 11, the model on KITTI with a resolution of 608 × 608 px achieves an average precision of 97.15% for car detection, 81.25% for pedestrian detection and 89.79% for bicycle detection. At 512 × 512 px resolution, the model achieves an average accuracy of 96.56% for car detection, 76.61% for pedestrian detection, and 89.37% for bicycle detection. Additionally, with a resolution of 320 × 320 px, the model achieves an AP of 89.71% for car detection, 61.94% for pedestrian detection, and 70.89% for bicycle detection. Similarly, for Self Driving Car, as shown in Figure 12, with 608 × 608 px, the model achieves an AP of 77.85% for car detection, 51.05% for pedestrian detection, and 45.66% for bicycle detection. For a resolution of 512 × 512 px, the model obtains an AP of 78.33% for car detection, 49.72% for pedestrian detection, and 44.78% for bicycle detection. Additionally, with a resolution of 320 × 320 px, the model achieves an average accuracy of 71.92% for car detection, 36.66% for pedestrian detection, and 32.87% for bicycle detection.

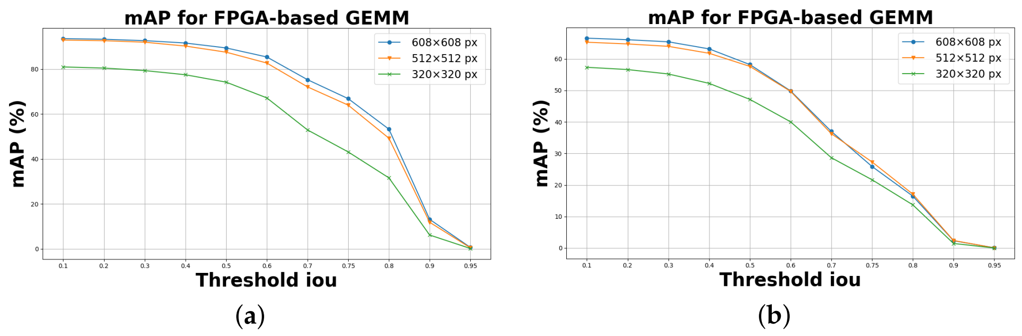


Figure 10. Evaluating the mean average precision for FPFA-based GEMM of YOLOv4 on two autonomous driving datasets. (**a**) mAP for FPGA-based GEMM on KITTI dataset. (**b**) mAP for FPGA-based GEMM on Self Driving Car dataset.

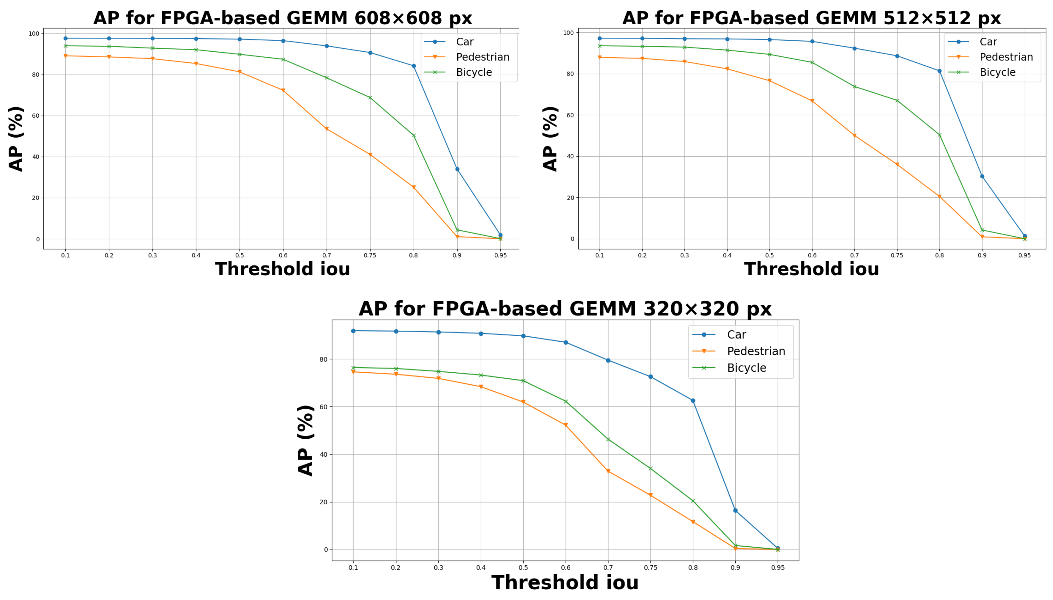


Figure 11. Average precision of three classes on KITTI dataset for FPGA-based GEMM of YOLOv4 (blue: car, green: pedestrian, orange: bicycle).

Our research results demonstrate the effectiveness of our proposed FPGA architecture for implementing the GEMM operation in YOLOv4 without affecting the accuracy. Our approach provides detection results comparable to those obtained with GPU implementations in Section 5 while offering significant performance advantages.

These results demonstrate that the proposed GEMM implementation is an efficient way to process data in an FPGA environment and can be successfully used for object detection in autonomous driving applications. Additionally, our proposed implementation of GEMM on FPGA achieved a runtime of 38 ms with a cadence of 100 Hz on the KITTI dataset, which can reduce power consumption and improve the overall performance of embedded systems. This processing time could be reduced by using optimization techniques such as shift register pipelining, and channels. These techniques can significantly improve the performance of computing systems by increasing the speed of data transfer and processing.

Overall, our results demonstrate potential generalizability to other versions of the YOLO family, including the recently released YOLOv8, which is available on GitHub [35]. In addition, YOLOv8 and YOLOv4 share the same convolutional blocks, enabling us to translate our study to YOLOv8. This transferability demonstrates the robustness and applicability of our research within the broader YOLO framework, allowing us to further explore and validate our findings within the context of YOLOv8. Furthermore, our results highlight the potential of FPGA-based implementations to perform effective object detection in real-world applications.

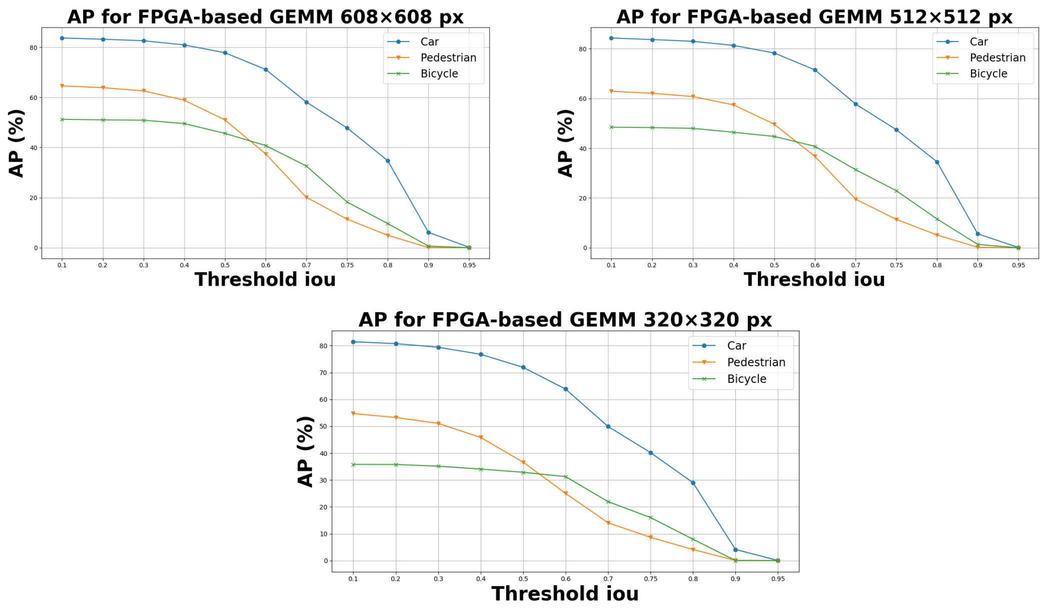


Figure 12. Average precision of three classes on Self Driving Car dataset for FPGA-based GEMM of YOLOv4 (blue: car, green: pedestrian, orange: bicycle).

7. Conclusions

This paper investigates the utilization of Field Programmable Gate Arrays for implementing YOLOv4 GEMM for autonomous driving applications. Our study analyzes the performance of YOLOv4 on two prominent autonomous driving datasets, namely KITTI and Self Driving Car. Notably, we have assessed the performance using various input sizes, specifically 608 × 608 px, 512 × 512 px, and 320 × 320 px. Furthermore, we have compared the detection performance achieved by GEMM operations implemented on an FPGA compared to those executed on a GPU. The outcomes derived from our investigation effectively demonstrate the immense potential of FPGAs in implementing YOLOv4 while preserving model accuracy.

In future research, there are several avenues to further explore the potential of programmable gate arrays in accelerating YOLOv4. It would be interesting to further analyze the performance of other versions of YOLO, such as YOLOv6, YOLOv7, and YOLOv8, on other autonomous driving datasets, such as BDD and Cityscapes, to gain a complete understanding of the effectiveness of these models. Alternatively, considering the complexity inherent in YOLOv4 as a neural network architecture, it becomes interesting to investigate the application of FPGA acceleration to other network components. This could involve exploring novel techniques for optimizing convolutional operations on FPGAs, including systolic networks, data preprocessing approaches, and parallelization techniques. Additionally, conducting comparative studies to evaluate the performance of optimized YOLOv4 implementations on FPGAs against CPUs, GPUs, and other FPGAs would provide invaluable information.

Author Contributions: All authors contributed to the editing and improvement of the manuscript. F.Z.G. developed and implemented the methodology for this article. Conceptualization, F.Z.G., S.R.F. and A.E.O.; data curation, F.Z.G.; software, S.R.F., A.E.O. and M.A.; validation, S.R.F., A.E.O., M.A. and M.R.; supervision, S.R.F., A.E.O., M.A. and M.R.; resources, all authors. All authors analyzed the results and conducted literature reviews. All authors contributed to the design and development of the experiments. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially funded by the Ministry of Europe and Foreign Affairs, (Eiffel grant number: 116724T), and by the National Center for Scientific and Technical Research of Morocco (grant number: 30UM5R2021).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Wei, J.; He, J.; Zhou, Y.; Chen, K.; Tang, Z.; Xiong, Z. Enhanced object detection with deep convolutional neural networks for advanced driving assistance. *IEEE Trans. Intell. Transp. Syst.* **2019**, *21*, 1572–1583. [CrossRef]
2. Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.Y.; Berg, A.C. Ssd: Single shot multibox detector. In Proceedings of the Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, 11–14 October 2016; pp. 21–37.
3. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. *Adv. Neural Inf. Process. Syst.* **2015**, *28*, 91–99. [CrossRef] [PubMed]
4. Bharati, P.; Pramanik, A. Deep learning techniques—R-CNN to mask R-CNN: A survey. In *Computational Intelligence in Pattern Recognition: Proceedings of CIPR 2019*; Springer: Singapore, 2020; pp. 657–668.
5. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You only look once: Unified, real-time object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 779–788.
6. Zeng, K.; Ma, Q.; Wu, J.W.; Chen, Z.; Shen, T.; Yan, C. FPGA-based accelerator for object detection: A comprehensive survey. *J. Supercomput.* **2022**, *78*, 14096–14136. [CrossRef]
7. Minakova, S.; Tang, E.; Stefanov, T. Combining task-and data-level parallelism for high-throughput cnn inference on embedded cpus-gpus mpsocs. In Proceedings of the Embedded Computer Systems: Architectures, Modeling, and Simulation: 20th International Conference, Athens, Greece, 5–9 July 2020; pp. 18–35.
8. Roszyk, K.; Nowicki, M.R.; Skrzypczyński, P. Adopting the YOLOv4 architecture for low-latency multispectral pedestrian detection in autonomous driving. *Sensors* **2022**, *22*, 1082. [CrossRef] [PubMed]
9. Sim, I.; Lim, J.H.; Jang, Y.W.; You, J.; Oh, S.; Kim, Y.K. Developing a compressed object detection model based on YOLOv4 for deployment on embedded GPU platform of autonomous system. *arXiv* **2021**, arXiv:2108.00392.
10. Nurvitadhi, E.; Sheffield, D.; Sim, J.; Mishra, A.; Venkatesh, G.; Marr, D. Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, and ASIC. In Proceedings of the 2016 International Conference on Field-Programmable Technology, Xi'an, China, 7–9 December 2016; pp. 77–84.
11. Babu, P.; Parthasarathy, E. Hardware acceleration for object detection using YOLOv4 algorithm on Xilinx Zynq platform. *J. Real-Time Image Process.* **2022**, *19*, 931–940. [CrossRef]
12. Pham-Dinh, T.; Bach-Gia, B.; Luu-Trinh, L.; Nguyen-Dinh, M.; Pham-Duc, H.; Bui-Anh, K.; Nguyen, X.Q.; Pham-Quoc, C. An FPGA-Based Solution for Convolution Operation Acceleration. In *Intelligence of Things: Technologies and Applications*; Nguyen, N.T., Dao, N.N., Pham, Q.D., Le, H.A., Eds.; Springer International Publishing: Cham, Switzerland, 2022.
13. Sudrajat, M.R.D.; Adiono, T.; Syafalni, I. GEMM-Based Quantized Neural Network FPGA Accelerator Design. In Proceedings of the 2019 International Symposium on Electronics and Smart Devices, Bali, Indonesia, 8–9 October 2019; pp. 1–5.
14. Zhang, W.; Jiang, M.; Luo, G. Evaluating low-memory GEMMs for convolutional neural network inference on FPGAS. In Proceedings of the 2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines, Fayetteville, AR, USA, 3–6 May 2020; pp. 28–32.
15. Belabed, T.; Coutinho, M.G.F.; Fernandes, M.A.; Sakuyama, C.V.; Souani, C. User driven FPGA-based design automated framework of deep neural networks for low-power low-cost edge computing. *IEEE Access* **2021**, *9*, 89162–89180. [CrossRef]
16. Feng, D.; Harakeh, A.; Waslander, S.L.; Dietmayer, K. A review and comparative study on probabilistic object detection in autonomous driving. *IEEE Trans. Intell. Transp. Syst.* **2021**, *23*, 9961–9980. [CrossRef]
17. Shetty, A.K.; Saha, I.; Sanghvi, R.M.; Save, S.A.; Patel, Y.J. A review: Object detection models. In Proceedings of the 2021 6th International Conference for Convergence in Technology, Pune, India, 2–4 April 2021; pp. 1–8.
18. Ren, J.; Wang, Y. Overview of object detection algorithms using convolutional neural networks. *J. Comput. Commun.* **2022**, *10*, 115–132.
19. Andriyanov, N.; Papakostas, G. Optimization and Benchmarking of Convolutional Networks with Quantization and OpenVINO in Baggage Image Recognition. In Proceedings of the 2022 VIII International Conference on Information Technology and Nanotechnology, Samara, Russia, 23–27 May 2022; pp. 1–4.
20. Geiger, A.; Lenz, P.; Urtasun, R. Are we ready for autonomous driving? the kitti vision benchmark suite. In Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, 16–21 June 2012; pp. 3354–3361.
21. Roboflow Self Driving Car Dataset. Available online: https://public.roboflow.com/object-detection/self-driving-car (accessed on 26 February 2023).
22. Bochkovskiy, A.; Wang, C.Y.; Liao, H.Y.M. Yolov4: Optimal speed and accuracy of object detection. *arXiv* **2020**, arXiv:2004.10934.
23. Redmon, J.; Farhadi, A. Yolov3: An incremental improvement. *arXiv* **2020**, arXiv:1804.02767.
24. Jiang, P.; Ergu, D.; Liu, F.; Cai, Y.; Ma, B. A Review of Yolo algorithm developments. *Procedia Comput. Sci.* **2022**, *199*, 1066–1073. [CrossRef]
25. Zaidi, S.S.A.; Ansari, M.S.; Aslam, A.; Kanwal, N.; Asghar, M.; Lee, B. A survey of modern deep learning based object detection models. *Digit. Signal Process.* **2022**, *136*, 103514. [CrossRef]
26. Lian, X.; Zhang, C.; Zhang, H.; Hsieh, C.J.; Zhang, W.; Liu, J. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 1–11.
27. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. *Commun. ACM* **2017**, *60*, 84–90. [CrossRef]
28. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556.

29. Chollet, F. Xception: Deep learning with depthwise separable convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 1251–1258.
30. Fränti, P.; Mariescu-Istodor, R. Soft precision and recall. *Pattern Recognit. Lett.* **2023**, *167*, 115–121. [CrossRef]
31. Lin, T.Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; Zitnick, C.L. Microsoft coco: Common objects in context. In Proceedings of the Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, 6–12 September 2014; pp. 740–755.
32. Kuznetsova, A.; Rom, H.; Alldrin, N.; Uijlings, J.; Krasin, I.; Pont-Tuset, J.; Kamali, S.; Popov, S.; Malloci, M.; Kolesnikov, A.; et al. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. *Int. J. Comput. Vis.* **2020**, *128*, 1956–1981. [CrossRef]
33. Everingham, M.; Van Gool, L.; Williams, C.K.; Winn, J.; Zisserman, A. The pascal visual object classes (voc) challenge. *Int. J. Comput. Vis.* **2010**, *88*, 303–338. [CrossRef]
34. Guerrouj, F.Z.; Abouzahir, M.; Ramzi, M.; Abdali, E.M. Analysis of the acceleration of deep learning inference models on a heterogeneous architecture based on OpenVINO. In Proceedings of the 2021 4th International Symposium on Advanced Electrical and Communication Technologies, Alkhobar, Saudi Arabia, 6–8 December 2021; pp. 1–5.
35. Jocher, G.; Chaurasia, A.; Qiu, J. YOLO by Ultralytics. Available online: https://github.com/ultralytics/ultralytics (accessed on 19 May 2023).