

Implementation of a perception system for autonomous vehicles using a detection-segmentation network in SoC FPGA

Maciej Baczmanski¹, Mateusz Wasala¹ , and Tomasz Kryjak¹ 

Embedded Vision Systems Group, Computer Vision Laboratory,
Department of Automatic Control and Robotics,
AGH University of Krakow, Al. Mickiewicza 30, 30-059 Krakow, Poland
mbaczmanski@student.agh.edu.pl
{mateusz.wasala, tomasz.kryjak}@agh.edu.pl

Abstract. Perception and control systems for autonomous vehicles are an active area of scientific and industrial research. These solutions should be characterised by both high efficiency in recognising obstacles and other environmental elements in different road conditions, real-time capability, and energy efficiency. Achieving such functionality requires an appropriate algorithm and a suitable computing platform. In this paper, we have used the MultiTaskV3 detection-segmentation network as the basis for a perception system that can perform both functionalities within a single architecture. It was appropriately trained, quantised, and implemented on the AMD Xilinx Kria KV260 Vision AI embedded platform. By using this device, it was possible to parallelise and accelerate the computations. Furthermore, the whole system consumes relatively little power compared to a CPU-based implementation (an average of 5 watts, compared to the minimum of 55 watts for weaker CPUs, and the small size (119mm x 140mm x 36mm) of the platform allows it to be used in devices where the amount of space available is limited. It also achieves an accuracy higher than 97% of mAP (mean average precision) for object detection and above 90% of mIoU (mean intersection over union) score for image segmentation. The article also details the design of the Mecanum wheel vehicle, which was used to test the proposed solution in a mock-up city.

Keywords: detection-segmentation neural network, perception, embedded AI, SoC FPGA, eGPU, Vitis AI, Mecanum wheel vehicle

1 Introduction

Today, we are witnessing the rapid development of advanced mobile robotics, including autonomous cars and drones (unmanned aerial vehicles, UAV). This would not be possible without advances in the implementation of perception and control systems, including the use of deep neural networks (DNN). DNNs make

it possible to achieve high accuracy, but memory and computational complexity remain significant challenges. In order to meet the requirements of mobile platforms, i.e. low latency and low energy consumption, it becomes necessary to use specialised hardware platforms such as SoC FPGAs (System on Chip Field Programmable Gate Arrays) or eGPUs (embedded Graphic Processing Units). These solutions also have the advantage of relatively small size and weight. It is also worth noting that a major challenge is the reliability analysis of network-based solutions, including their explainability [1]. This is of particular importance when traffic safety, for example, depends on DNNs detections or control.

In perception systems, two basic tasks can be roughly distinguished: object detection and segmentation (semantic and instance). Object detection is the marking of objects belonging to the considered classes (e.g. cars, pedestrians, cyclists, traffic signs, etc.) in the image with bounding boxes or sometimes binary masks. Semantic segmentation involves assigning to each pixel a label that tells what object it belongs to (e.g. drivable area, horizontal road sign, vegetation, buildings, persistent, or sky). Instance segmentation, on the other hand, allows different labels to be given to pixels belonging to two separate objects of the same class (e.g. two pedestrians). It should be noted that object detection is a simpler and thus computationally less complex task. A typical solution /changeusingthat uses DNNs is the YOLO (You Only Look Once) family of algorithms [2]. In contrast, segmentation, especially of instances to obtain similar information, is much more complex – requiring both longer learning and inference. U-Nets [3] are typically used for semantic segmentation and Mask R-CNN-based [4] solutions for instances.

For autonomous vehicle perception systems, the tasks of detection and segmentation appear together. For objects such as pedestrians, vehicles, bicycles, vertical road signs, or traffic lights, the use of detection is sufficient. However, for the detection of drivable area or horizontal road signs (including pedestrian crossings), it is better to use segmentation. Hence, detection-segmentation networks have been proposed in the literature, which combine the advantages of both approaches and, at the same time, thanks to a common backbone (encoder), are characterised by lower computational complexity and an easier learning process than instance segmentation approaches. A detection-segmentation network, in addition to the aforementioned backbone, consists of a segmentation head and several detection heads. Examples of such networks are YOLOP [5], HybridNets [6] and MultiTask V3 [7] discussed in Section 2.

Taking into account the properties of the detection segmentation networks discussed above, we decided to use this solution as the basis for the perception system of our autonomous vehicle model. We used the MultiTask V3 network, which we implemented and deployed on two embedded platforms: SoC FPGA Kria KV260 and an eGPU (NVIDIA Jetson Nano and Xavier NX). The experiments performed showed that detection-segmentation networks represent a good compromise between accuracy, performance, and power consumption. We also discussed the design of the Mecanum wheeled vehicle used. To the best of our knowledge, this is the first paper that discusses the hardware implementation of

a perception system based on a detection-segmentation network implemented in an SoC FPGA, the results of which were applied to the control of an autonomous vehicle model.

The remainder of this paper is structured as follows. In Section 2 we discuss the relevant prior works on detection-segmentation networks and DNNs acceleration on SoC FPGA. Section 3 discusses the methods used, including the hardware implementation of the considered DNNs, and the design of the autonomous vehicle model. The results obtained are summarised in Section 4. The paper ends with conclusions and a discussion of possible future research.

2 Previous work

Three types of deep neural networks can be distinguished in current vision systems: detection, segmentation, and detection-segmentation. As mentioned in the introduction, detection-segmentation networks represent a compromise between the accuracy of instance segmentation and the speed of simple detection and are therefore an interesting solution for autonomous vehicle perception systems. Several architectures of detection-segmentation networks have been proposed in the literature.

The first is YOLOP [5]. It allows object detection and segmentation of drivable area and horizontal road markings. It consists of a common encoder and 3 separate decoders (one for detection and two for segmentation). It has been trained and evaluated on the popular *BDD100k* dataset [8]. The second is HybridNets [6], which is very similar to YOLOP in terms of functionality. It consists of 4 components: encoder (EfficientNet V2 architecture), neck, detection head (inspired by YOLOv4), and segmentation head. The *BDD100k* dataset was also used for training and evaluation. The third architecture, used in this work, is the MultiTask V3 [7] proposed by AMD Xilinx. It is worth noting that it is included in the Vitis AI library as a demonstrator of its capabilities, but to our knowledge, it has not been described in a scientific publication. Details of its construction are presented in Section 3.1. Unlike YOLOP and HybridNets, it also includes a depth estimation module. However, it has not been evaluated on a publicly available dataset.

The topic of hardware acceleration of deep neural networks, especially for embedded computing, is the subject of intense academic and industrial research due to its very high practical importance. A whole spectrum of solutions is encountered, from dedicated chips for AI acceleration (e.g. Intel Neural Compute Stick, Google Coral, Tesla FSD Chip), through programmable SoC FPGAs to eGPU platforms. A detailed overview of the solutions is beyond the scope of this article, and we refer interested readers, for example, to the review [9] or the work [10].

In this work, we have chosen to use an SoC FPGA platform and also run the selected network on an eGPU platform for comparison. Reprogrammable devices have been a proven platform for implementing vision algorithms for years, which was the main reason for our choice. In addition, they tend to have lower power

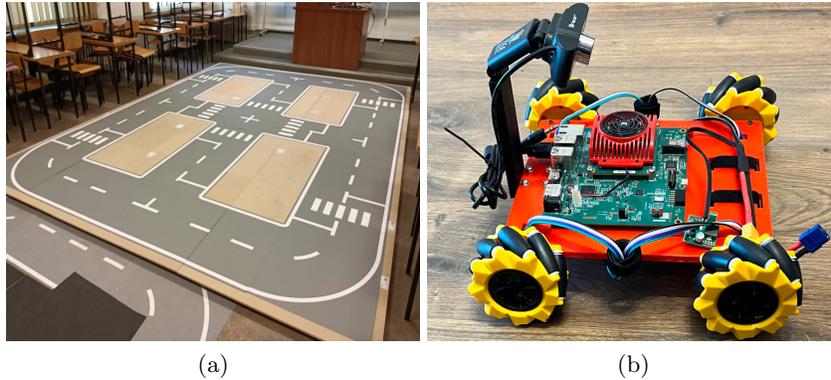


Fig. 1: The mock-up of a city made by us (a) and the model of an autonomous vehicle (b) with Mecanum wheels and all equipment.

consumption than eGPUs. Of the available detection-segmentation networks, we chose MultiTask V3 for two reasons. First, from our previous experiments, it had the highest efficiency and relatively low computational complexity for our scenario. Second, it was well-prepared by AMD Xilinx for acceleration in SoC FPGAs, which facilitated its use in the target perception and control system.

3 Implementation of the perception and control system

The starting point for our research was the FPT'22 [11] competition, the aim of which is to create a model of an autonomous vehicle capable of driving according to the road traffic rules in a mock-up city. Figure 1a shows the used mock-up city. It is equipped with horizontal markings (traffic lanes, pedestrian crossings), traffic lights, figures imitating pedestrians, and various objects (obstacles) to be avoided on the road. Thanks to this test environment, it is possible to evaluate the perception and control system of an autonomous vehicle. The research presented can be divided into four phases: the design and construction of an autonomous vehicle equipped with Mecanum wheels, the design of electronics and assembly equipment, the implementation of the perception and control algorithm on the AMD Xilinx Kria KV260 platform, and the programming of a low-level algorithm to control the motors for the Mecanum wheels. The most important part of the work is the implementation of the perception and control system. It uses a detection-segmentation deep convolutional neural network architecture that is parallelised, quantised, and accelerated on an embedded SoC FPGA platform. On the other hand, the Mecanum wheels allow for precise manoeuvring, and the detection-segmentation network provides the necessary information about obstacles and other elements of the environment. In addition, the PID controller implemented in the motor controllers ensures stable driving, which is essential for the safety of the vehicle.

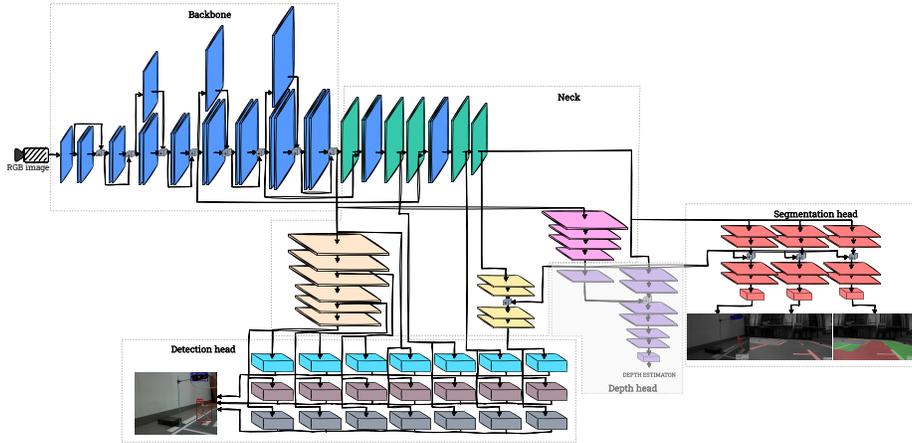


Fig. 2: Scheme of the MultiTask V3 deep neural network, showing layers of neurons grouped into sections. An input image is processed within successive layers to extract features. The features are used to generate output data: detections, segmentation, and also a depth map.

3.1 Detection-segmentation network in SoC FPGA

MultiTask V3 is a deep convolutional neural network, designed by the developers of Vitis AI (AMD Xilinx) as part of an open source library made available for the development process [7]¹. Its architecture is shown in Figure 2 and allows the simultaneous execution of five tasks: detection, three types of segmentation, and depth estimation (not used in this work).

The segmentation part of the architecture is divided into three branches. Each branch can focus on a different task, such as segmenting detected objects, lanes (drivable area), or road markings. This approach makes it easier to prepare training sets, as these can be separated from each other, allowing a pixel to be classified in more than one class (e.g. a road marking should still be detected as a lane). The additional use of detection means that an in-depth analysis of detected objects (e.g. in terms of shape or occupied area in the image) is optional and performed only in special cases. The MultiTask V3 network architecture consists of several elements. First, the input image is transferred to the *Backbone* segment, which is used for feature extraction. This is based on the ResNet-18 convolutional neural network. Then, thanks to the use of encoders and convolutional layers, the *Neck* segment allows further feature extraction and the combination of low-level and high-level features. The features obtained are transferred to the appropriate branches: *Detection*, *Depth*, and *Segmentation* heads. In them, again, thanks to the use of convolution, activation operations, and normalisation, the corresponding result tensors are generated.

¹ MultiTask V3 has not been described in a published scientific paper.

Due to the specificity of the project and the high complexity of the training set for depth estimation, the *Depth* head training was not considered. For the remaining branches, three training sets were prepared, one common for object detection and segmentation and two for drivable area segmentation and road markings. The data for the training sets were obtained from recordings made on a city mock-up, which made it possible to prepare them strictly for the assumed task. 250 photos were obtained for the set containing the detected objects and 500 photos for the set showing the drivable area. The images were then manually labelled using the LabelMe software. The generated datasets were converted into a format compatible with the framework used to train the network. The framework is open source, based on Python, uses the PyTorch libraries, and is published in the Vitis AI libraries. As the software was written for older versions of the libraries and Python, corrections had to be made in order for the code to run properly. Once the modifications had been made, the software was launched using the prepared datasets. The model was trained using the GTX 1060 M GPU on sets split 80/20 between training and validation. The training was stopped after 450 epochs if there was no improvement in network performance.

The next step was to quantise the network model so that it could be run on an embedded SoC FPGA platform. This was done using the software described above. The quantisation is based on the `vai_p_pytorch` API provided by AMD Xilinx. Finally, the model was compiled into an architecture-compatible format using the `vai_c_xir` program, also provided by AMD Xilinx.

The final detection-segmentation model has been launched on the Kria KV260 SoC FPGA platform [12]. Kria is designed for the development of advanced image processing applications, allowing the acceleration of neural networks thanks to the use of DPU (Deep Processing Unit). The platform's operating system is Ubuntu, with PYNQ software installed, which allows a program to be created in Python on notebooks using the DPU overlay. In addition, by using the WiFi USB adapter and modifying the operating system's network settings, it is possible to communicate with the platform via SSH (Secure Shell) and through the Jupyter Notebook server created, allowing the algorithm to be executed and its operation to be analysed in real-time. This communication also makes it possible to continuously monitor the consumption of resources and the performance of the algorithm. Thanks to the libraries used, it is possible to collect image frames from a connected USB camera with a resolution of 512×320 pixels, convert them into the network input tensor, and then analyse the output tensors using methods from the OpenCV library. The implemented algorithm imports the necessary libraries and defines data pre-processing and processing functions.

3.2 Vehicle control algorithm

The algorithm captures the last frame from the USB camera, pre-processes it (size, colour space), and converts it into tensors, which are then fed into the MultiTaskv3 neural network. The network returns tensors which are then converted into masks: segmentation of detected objects, segmentation of drivable area, segmentation of road markings, and bounding boxes of detected objects.

The received data is then analysed: first, it is checked that the pedestrian or obstacle is not in the ROI, which is defined as a short distance in front of the vehicle. In the case of a pedestrian, the vehicle should stop, and in the case of an obstacle, the overtaking manoeuvre should be initiated. The lines are then checked. The detection of a continuous cross-line marking triggers a vehicle stop. Based on the sideline, it is possible to determine the trajectory of movement. If the sideline is not in the ROI – on the left side of the image, the segmentation of the drivable area allows checking if the vehicle is at an intersection or in a curve, which means it needs to turn. Based on the results of the analysis, a trajectory is determined and transmitted to the Arduino microcontroller, which controls the motors. The loop then returns to the initial step and continues indefinitely.

3.3 Hardware setup

The electronics project consisted of placing the Arduino Nano Every microcontroller, based on the ATmega4809, on the breadboard, allowing the use of hardware interrupts on any pin. The microcontroller is directly connected to the motor encoders and four Pololu DRV8838 motor controllers, which allow control using the PWM (Pulse-width Modulation) signal. The power section consists of a LiPo package and step-down converters: 12V for the FPGA platform and 6V for the motors. The microcontroller communicates and is powered via a USB connection to the FPGA platform. The motor control was programmed on the microcontroller in the language provided by Arduino, based on C++. The program receives the set values from the FPGA platform through the UART protocol in the format V_x, V_y, ω , where V_x is the longitudinal velocity vector, V_y is the transverse velocity vector, and ω is the given angular velocity of rotation relative to the geometric centre of the vehicle. From the above values, the angular velocities set values for each of the motors are determined. The rotation of each wheel changes the signals on the encoder connected to it. Using hardware interrupts, it is possible to determine the angle that each of the motors has turned, which is counted in the counter assigned to it, and stored in the cache. The interrupt timer has been implemented in the program, which calls the function exactly every 0.1 seconds. This function retrieves the current counter reading and compares it with the previous one. This is used to determine the angular velocity, the previous values of which are also stored and differentiated for the purposes of the PID (Proportional Integral Derivative) controller. Then, for each motor, the control set for the given speed, the control error and its differential are determined, which makes it possible to determine the P and D terms of the PID controller. The values obtained are used to determine the filling of the PWM signal sent to the motor controllers. The program runs in an infinite loop, and in asynchronous mode, the microcontroller is constantly waiting for a new reference to be sent.

In order to better adapt the vehicle to the dimensions of the city mock-up, all its elements were made using 3D printing technology, such as adapters for the motors to mount the wheels, USB camera holder, base platform adapted to mount the motors, cameras, electronics, power supply, and the main computing

platform. Four Pololu HP micromotors with 150:1 gears and encoders were attached to the base platform, on which the shaft was mounted using Mecanum 80mm diameter wheels with adapters. On the underside of the platform is a breadboard with electronics to control the motors and a 14.8V nominal LiPo pack. At the top of the chassis is a computer platform and a USB camera mount. Figure 1b shows the model of the autonomous vehicle described above.

4 Evaluation of the detection-segmentation network

The first experiment was to compare the quality (efficiency, accuracy) of network model inference before and after quantisation. The tests were performed using the libraries provided by AMD Xilinx, discussed earlier. Each branch was evaluated on the test set and the results are summarised in Tables 1, 2, 3 and 4. As can be seen, quantisation resulted in a slight quality decrease (of the order of less than one per cent). This means, therefore, that the model used by the SoC FPGA platform will behave almost identically to the one run on a PC equipped with a graphics card in the environment provided by AMD Xilinx. 3.1.

To test the efficiency and cost-effectiveness of the proposed solution, a series of performance tests were carried out on the Kria KV260 platform. The input to the algorithm was a pre-prepared dataset derived from footage recorded on a mock-up of the city. During operation, the use of the quad-core Cortex-A53 processor clocked at 1.3 GHz used in the platform, the use of RAM (Random Access Memory) and CMA (Contiguous Memory Allocator), and the power consumption of the SOM (System on Module) platform were checked. The results are shown in Table 5. It is worth noting that the platform makes full use of one CPU core. According to the manufacturer’s documentation, it is possible to run the algorithm using multithreading, but this would involve higher power consumption. The results show that the platform consumes only around 5W of power when running, which allows it to be considered energy efficient.

In order to compare the performance of the platform used, the inference time of the MultiTask V3 network and the execution time of one iteration of the algorithm was examined. The same algorithm was then run on the NVIDIA Jetson Nano and NVIDIA Jetson Xavier NX eGPU platforms, using the pre-quantisation model and the PyTorch library to run the network. The results of the algorithm’s efficiency on the platforms are shown in Table 6.

Experiments show that the Kria KV260 platform has demonstrated the best performance in its power consumption class. In terms of processing speed, it clearly outperforms the NVIDIA Jetson Nano platform, with the same power consumption. It also runs faster than the NVIDIA Jetson Xavier NX platform in 10W consumption mode. Only when using the 20W consumption mode does the NX platform achieve approximately 0.5 fps (frames per second) more, but at the cost of four times higher power consumption.

The achieved processing speed of almost 5 FPS is sufficient for the algorithm to make a decision in a satisfactory time. However, the results show that the

Table 1: Comparison of results for object detection (mAP – mean Average Precision).

Quantisation state	mAP ₅₀ [%]	mAP ₇₀ [%]	mAP ₇₅ [%]
Before	99.4	99.4	97.2
After	99.3	99.3	97.0

Table 2: Comparison of results for drivable area segmentation (MIoU – Mean IoU, IoU – Intersection over Union).

Quantisation state	MIoU [%]	IoU [%]	
		Background	Drivable area
Before	97.31	97.88	96.75
After	97.29	97.86	96.72

Table 3: Comparison of results for lane segmentation (MIoU – Mean IoU, IoU – Intersection over Union).

Quantisation state	MIoU [%]	IoU [%]	
		Background	Lanes
Before	90.72	99.04	82.40
After	90.69	99.04	82.33

Table 4: Comparison of results for object segmentation (MIoU – Mean IoU, IoU – Intersection over Union).

Quantisation state	MIoU [%]	IoU [%]					
		Background	Pedestrian	Amber Light	Red Light	Green Light	Obstacle
Before	96.52	99.85	88.69	93.90	95.13	94.66	94.88
After	92.08	99.81	88.69	92.56	90.49	89.45	91.46

application of deep neural networks on energy-efficient embedded platforms is still a significant challenge.

To sum up. The best results were obtained on the Kria KV260 SoC FPGA platform. The SoC FPGA platform allows us to obtain satisfactory results in terms of accuracy, efficiency, and power consumption. It should be noted that the currently implemented algorithm is still under development, and the results show that it would be beneficial to focus more on code optimisation and system reconfiguration to utilise all CPU cores. This could slightly increase power consumption, but even 10W of consumption can be considered low for a platform that would be the most important element of an autonomous car. The code used in the experiments described is available at https://github.com/vision-agh/mt_kria.

Table 5: Comparison of resource consumption on the Kria KV260 platform.

Resource usage	CPU cores				RAM	CMA	Power
	CPU ₀	CPU ₁	CPU ₂	CPU ₃			
Used	85 %	22 %	3 %	3 %	38 %	6 %	4.95 W

Table 6: Comparison of algorithm’s performance on different computing platforms.

Embedded platform	Power [W]	Speed [fps]	Execution time [s]	Model Inference time [s]
Kria KV260	5	4.85	0.206	0.073
Nvidia Jetson Nano	5	2.07	0.483	0.223
Nvidia Jetson Xavier NX	10	4.35	0.230	0.093
	20	5.48	0.182	0.068

5 Conclusion

In this paper, we have discussed the implementation of a perception system for autonomous vehicles using a detection-segmentation network deployed in an SoC FPGA. We have presented the process of preparing a custom dataset according to the requirements of the FPT’22 competition and the training of a neural network model. We have also given a detailed description of the construction of a Mecanum wheel-based autonomous vehicle model, focusing on mechanical and electrical aspects. A fully autonomous control algorithm has been implemented and run on the discussed platform, as well as on two eGPUs. Several experiments have been performed, showing the efficiency and low power consumption of the proposed solution, which supports our thesis that the FPGA Kria KV260 using the MultiTask V3 neural network is a suitable solution for autonomous cars and robots with limited space and resources.

In future work, we will first refactor the code to further improve its efficiency. We also plan to test the vehicle model on the mock-up. Secondly, we will try to use the *weakly supervised learning* and *self-supervised learning* methods, which, in the case of an atypical, custom dataset, would allow a significant reduction in the labelling process of the learning data. We would also like to consider adding modules for depth estimation and optical flow, as these are often used in autonomous vehicle perception systems.

Acknowledgements

The work presented in this paper was supported by the programme “Excellence initiative - research university” for the AGH University of Krakow.

References

1. Minh, D., Wang, H.X., Li, Y.F. et al. "Explainable artificial intelligence: a comprehensive review". *Artif Intell Rev* 55, 3503–3568 (2022). <https://doi.org/10.1007/s10462-021-10088-y>
2. Terven, J. and Cordova-Esparza, D.: "A Comprehensive Review of YOLO: From YOLOv1 to YOLOv8 and Beyond", 2023, <https://doi.org/10.48550/arXiv.2304.00501>
3. Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." *Medical Image Computing and Computer-Assisted Intervention—MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III* 18. Springer International Publishing, 2015.
4. He, K., Gkioxari, G., Dollár, P. and Girshick, R.: "Mask R-CNN," 2017 IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 2017, pp. 2980-2988, doi: 10.1109/ICCV.2017.322.
5. Wu, Dong, et al. "Yolop: You only look once for panoptic driving perception." *Machine Intelligence Research* (2022): 1-13.
6. Vu, Dat, Bao Ngo, and Hung Phan. "Hybridnets: End-to-end perception network." *arXiv preprint arXiv:2203.09035* (2022).
7. "Vitis AI Library User Guide (UG1354)", <https://docs.xilinx.com/r/en-US/ug1354-xilinx-ai-sdk/MultiTask-V3>, accessed 18.05.2023
8. Yu, Fisher et al. "BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning." 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2018): 2633-2642.
9. K. P. Seng and L. -M. Ang, "Embedded Intelligence: State-of-the-Art and Research Challenges," in *IEEE Access*, vol. 10, pp. 59236-59258, 2022, doi: 10.1109/ACCESS.2022.3175574.
10. Wu, R.; Guo, X.; Du, J.; Li, J.: "Accelerating Neural Network Inference on FPGA-Based Platforms—A Survey". *Electronics* 2021, 10, 1025. <https://doi.org/10.3390/electronics10091025>
11. "The International Conference on Field-Programmable Technology (FPT'21) FPGA Design Competition". <https://wp.rs.cs.okayama-u.ac.jp/design-contest-fpt2022/>, accessed 16.05.2023.
12. "Kria KV260 Vision AI Starter Kit Product Brief" <https://www.xilinx.com/content/dam/xilinx/publications/product-briefs/xilinx-kv260-product-brief.pdf>, accessed 18.05.2023.