IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 6 October 2022, accepted 28 November 2022, date of publication 16 December 2022,
date of current version 23 December 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3230282

SURVEY

A Survey of FPGA-Based Vision Systems for Autonomous Cars

DAVID CASTELLS-RUFAS[1], VINH NGO[1], JUAN BORREGO-CARAZO[1,2], MARC CODINA[1], CARLES SANCHEZ[3], DEBORA GIL[3], AND JORDI CARRABINA[1]

[1]Department of Microelectronics and Electronics Systems, Universitat Autònoma de Barcelona, 08193 Cerdanyola del Vallés, Spain
[2]R+D, Kostal Eléctrica SA, 08181 Barcelona, Spain
[3]Computer Vision Center—Department of Computer Science, Universitat Autònoma de Barcelona, 08193 Cerdanyola del Vallés, Spain

Corresponding author: David Castells-Rufas (david.castells@uab.cat)

This work was supported by grants PID2021-126776OB-C21, RTI2018-095209-B-C21, RTI2018-095209-B-C22 funded by the Ministerio de Ciencia e Innovación (MCIN) from Spain with grant number MCIN/AEI/10.13039/501100011033 and by "European Regional Development Fund (ERDF) A way of making Europ"; Agència de Gestió d'Ajuts Universitaris i de Recerca grant numbers 2017-SGR-1624 and Centres de Recerca de Catalunya (CERCA) Programme/Generalitat de Catalunya. Juan Borrego is supported by the Industrial PhD program from the Generalitat de Catalunya with grant number 2018-DI-043.

ABSTRACT On the road to making self-driving cars a reality, academic and industrial researchers are working hard to continue to increase safety while meeting technical and regulatory constraints Understanding the surrounding environment is a fundamental task in self-driving cars. It requires combining complex computer vision algorithms. Although state-of-the-art algorithms achieve good accuracy, their implementations often require powerful computing platforms with high power consumption. In some cases, the processing speed does not meet real-time constraints. FPGA platforms are often used to implement a category of latency-critical algorithms that demand maximum performance and energy efficiency. Since self-driving car computer vision functions fall into this category, one could expect to see a wide adoption of FPGAs in autonomous cars. In this paper, we survey the computer vision FPGA-based works from the literature targeting automotive applications over the last decade. Based on the survey, we identify the strengths and weaknesses of FPGAs in this domain and future research opportunities and challenges.

INDEX TERMS Autonomous automobile, computer vision, field programmable gate arrays, reconfigurable architectures.

I. INTRODUCTION

Safety is always the most important concern for the human beings who interact on the roads as drivers, bikers, cyclists, or pedestrians. Most traffic accidents are caused by drivers. To reduce them, governments have strengthened legislation by introducing measures such as speed limits, breathalyzer tests, airbags, etc. Despite the exceptional reduction of fatalities during 2020 due to the pandemic [1], there is a background trend towards an increase in the number of accidents. According to the World Health Organization [2], 1.35 million died on the road in 2016. One of the solutions to address this problem is the adoption of Advanced Driving Assistance Systems (ADAS) technologies.

The associate editor coordinating the review of this manuscript and approving it for publication was Yongjie Li.

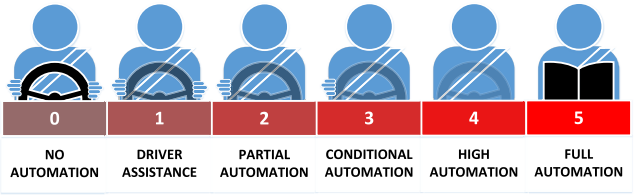


FIGURE 1. Levels of autonomous driving as defined by SAE [3].

ADAS help or replace humans to control the different actuators of a car. The most important actuators are the throttle pedal, the brake, the gearbox, and the steering wheel, but automation also happens in less critical ones like light controls, windscreen wipers, door lockers, etc. The computing systems that take the control of the car need to sense the

surrounding world and create a perception model so that they can analyze the situation to make the right decisions to undertake and complete a mission. The different functions of a self-driving car can be modeled as interdependent different control loops managed at different hierarchical levels. Many research works have presented implementations of these loops with practical constraints. The key requirements for self-driving car computing systems are accuracy, latency, energy efficiency, and economic cost. Accuracy and latency have an impact on safety, while energy efficiency has an impact on autonomy. Currently, a great amount of research is aimed at improving the accuracy of the algorithms required to reach the level 5 from the automation levels defined by the Society of American Engineers (SAE) in [3] and depicted in figure 1. Other aspects will likely receive more attention in the future.

Self-driving functions can be implemented in several computing platforms such as ASICs, CPUs, DSPs, GPUs, FPGAs, or in heterogeneous platforms composed by any combination of aforementioned platforms. One of the most distinct capabilities of FPGAs is that concurrent tasks can be parallelized to a great extent by using techniques such as hardware partitioning or hardware pipelining. Moreover, the latency of some FPGA-based designs can be deterministic [4], [5], which is often a requirement for the car's control system, especially for safety-critical functions. As depicted in 2, FPGAs are typically more energy efficient than other platforms except for ASICs, which usually have higher Non-Recurring Engineering (NRE) costs [6]. However, they provide higher flexibility and a significant lower time-to-market. Other platforms such as CPUs, GPUS, and DSPs provide higher flexibility thanks to a more intensive use of external memory to store intermediate results. This is also the main driver of their generally higher energy consumption. For neural network inference accelerators, FPGA implementations can be up to 10× more energy efficient than GPU ones [7], [8].

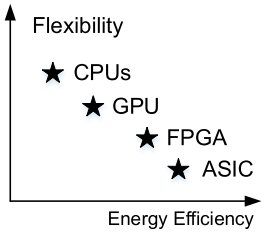


FIGURE 2. Simplification of the Flexibility vs. Energy Efficiency trade-off of computing platforms.

In this paper, we present a survey on Computer Vision (CV) algorithms for autonomous cars using FPGAs. We limit the scope of this paper to CV systems since we expect that image sensors will become dominant in future cars. Traffic signs, traffic lights, or braking/turning lights can only be recognized by image sensor-based systems. Other sensor technologies such as laser imaging detection and ranging (LIDAR) could be also integrated in future cars, but they are still facing some challenges in cost and performance in adverse conditions [9].

To the best of our knowledge, this is the first attempt to survey the use of FPGAs for vision-based ADAS functions, although there have been related recent surveys. For instance, Wan [10] surveys the use of FPGA in robotic computing. Badue [11] and Yurtsever [12] survey the main functional components of self-driving cars. Dewangan [13] focus on vision-based ADAS functions, and [14] provides details about their hardware implementations of with special focus on CNNs. Archad [15] and Ayachi [16] survey the use of FPGAs to implement convolutional neural networks (CNNs). Our survey has specific comparisons on key performance metrics among various state-of-the-art implementations of ADAS problems, such as object detection, pedestrian detection, lane detection, and traffic sign recognition. The contribution of this paper includes:

- An analysis of vision-based problems on autonomous cars, key approaches and challenges.
- A survey of state-of-the-art FPGA-based implementations in the last ten years for those problems and gap indications.
- And a detailed exploration of the basic algorithms or techniques that are used in the FPGAs-based implementations.

After the list of acronyms used through the paper in table 1, the next section presents an overview of FPGA technology. Section III gives the overview of CV systems used in self-driving cars. After that, section IV explores the ''hardware friendly'' algorithms employed in such applications. Section V lists out the state-of-the-art implementations and compares their performance. A look at some future challenges for FPGA-based accelerators for CV processing on autonomous cars is given in section VI before concluding.

II. OVERVIEW OF FPGA TECHNOLOGY

After their humble beginnings serving as glue logic to connect the important chips on a PCB, nowadays, FPGAs are considered full-fledged computing platforms and a valid alternative to be used in high-end cars despite of their high cost.

The spatial computing paradigm can take profit of FPGA architectures for many highly parallel applications that have an important dataflow component, or high memory locality [17]. Not all applications can benefit from full FPGA implementations, but even in many of these cases, FPGAs can still offer some advantages when used in conjunction with host CPUs as accelerators. The speedup provided by such accelerators is often limited by the sequential part of the algorithms running in the CPU as formalized by the Amdahl's law [18].

Transistor density is a determinant factor of the performance and energy efficiency achieved in FPGA design [19]. FPGA manufacturers have historically been early adopters of new foundry nodes, therefore, benefiting from their latest advances in higher transistor densities and lower power consumption and following the Moore's Law.

TABLE 1. List of acronyms.

Acronym	Description
ADAS	Advanced driving assistance systems
ASIC	Application specific integrated circuit
CGRA	Coarse-grain reconfigurable architecture
CNN	Convolutional neural network
CPU	Central processing unit (processor)
CV	Computer vision
DSP	Digital signal processor/processing
DL	Deep-learning
DLU	Deep-learning processing unit
FF	Flip-flop (register)
FLOP	Floating-point operation
FPGA	Field programmable gate array
FPS	Frames per second
FSM	Finite state machine
GPU	Graphics processing unit
HD	Hamming distance
HDL	Hardware design language
HLS	High level synthesis
LD	Lane detection
LE	Logic element
LUT	Look-up table
ML	Machine learning
NN	Neural network
NoC	Network-on-chip
OD	Obstacle detection
PD	Pedestrian detection
SM	Stereo-matching
SAD	Sum of absolute differences
SoC	System-on-chip
SS	Semantic segmentation
TLD	Traffic lights detection
TLR	Traffic lights recognition
TPU	Tensor processing unit
TSD	Traffic sign detection
TSR	Traffic sign recognition

FPGAs are built by replicating many simple basic logic elements (LEs) and some more complex IPs such as dual-port memories and digital signal processing (DSP) modules. The main components of LEs are Look Up Tables (LUTs) and registers, while the main components of DSPs are adders and multipliers (see figure 3).

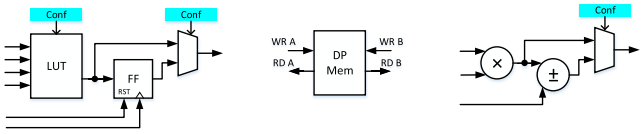


FIGURE 3. Main replicated FPGA components interconnected by a configurable interconnect.

In the late nineties, FPGAs already had thousands of LEs. To put it in perspective, a simple 32 bits Reduced Instruction Set Computing (RISC) processor needs few thousand LEs to be implemented. With the new millennium, it became possible to configure a subset of logic blocs to work as a processor and use the rest of the available logic to implement other processing units or system interfaces that could communicate among them and with the processor using buses or other simple mechanisms (directly, dual-port memories, etc.). Nowadays, the logic density is high enough to implement hundreds of simple processors in the same device. Internal memory can exceed the order of several dozens of MB [7]. The availability of low-latency local memories is very important for many CV applications where data locality can be exploited. On other tasks, the total bandwidth to global external memory is a key factor. Major FPGA manufacturers are addressing these requirements by integrating new multi-ported memories such as HBM [20]. The theoretical peak performance of the latest FPGAs can be up to several TFLOP/s. Although these estimations are based on assuming a full usage of all available computing resources, some works have successfully reached the TFLOP/s range [21].

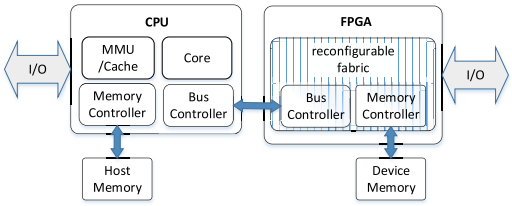


FIGURE 4. Typical structure of an FPGA-based accelerator.

A common structure of a FPGA-based accelerator is shown in Fig. 4. In this structure, the circuits implemented inside the FPGA typically process input data that was previously transferred from the host to a memory in the accelerator board through a PCI-express (PCIe) bus. Output data must be transferred back to the host by similar methods. Thanks to the FPGA flexibility, FPGA accelerator manufacturers can include the most convenient types of on-board memory. DDR4 and recently HBM2 SDRAM are the typical choices if large bandwidth and capacity is required. QDR SRAM is the usual choice for low latency. Communications overhead is often reduced by overlapping communication and computation using an interlaced strategy. For certain applications, this approach is limited by the bandwidth of the PCIe bus and the memory duplication required by the explicit memory transfer operations. Both problems are addressed by new breadth of devices with direct access to virtual memory addresses of the host computer through high speed buses such as OpenCAPI, QPI, and CLX, while providing cache coherency.

With the fast widespread of artificial intelligence (AI) algorithms FPGA manufacturers have seen an industrial increased demand for some long-standing requirements from academia, such as better floating point support, coarse-grain array structures, and network-on-chip (NoC) communications inside the chip. In [22], Li analyzes with detail these recent innovations fostered by AI. Xilinx addresses the coarse-grain and NoC features with its VERSAL adaptive compute acceleration platform (ACAP). Intel addresses NoC with Hyperflex and coarse-grain with higher complexity blocks, like floating point DSPs, and chiplets.

On the design tool-chain side, hardware design is generally approached by design abstractions such as finite state machines (FSM) and dataflow processing. System-level Synthesis (SLS) and High-level Synthesis (HLS) tools are used

by designers to implement their systems onto FPGAs shortening design and verification time. From a software perspective, SLS and HLS hide some implementation details (bit-wise operations, clock management, etc.) which require hardware design experience. HLS/SLS tools describe the generated circuits in a hardware description language (HDL) or a high abstraction language (such as System C, Chisel, etc.), ready for further steps (back-end or low-level synthesis). The language used by an HLS tool to specify the input depends on the specific compiler, being C/C++ one of the most commonly used [23].

III. COMPUTER VISION SYSTEMS FOR AUTONOMOUS CARS

CV systems process data acquired from image sensors. In CMOS imagers, the electric charge induced by the light irradiated over the pixel area is integrated by a capacitor during the exposure time window. Imagers can be classified according to their acquisition process. In frame-based sensors, output images are produced at a certain frequency (frame rate). In event-based sensors, pixels are individually produced as they reach certain requirements (such as reaching a light-integration threshold). The vast majority of CV research is based on frame-based sensors, although there is a growing interest in event-based CV algorithms [12], [24]. Imagers can also be classified according to the wavelength of the light they acquire. They can work in the visible and non-visible wavelength ranges (infrared or X-ray). Consumer sensors typically work in the red, green, and blue visible bands.

Monochrome sensors are sometimes used in cars due to its lower bandwidth requirements helping to reduce the workload of the image processing. Infrared or Multi-spectral imagers are rarely used in cars, but they could bring some benefits. For instance, infrared sensors can be used to detect pedestrians at night [25]. Another important factor is the dynamic range. Autonomous vehicles must work in extreme light conditions ranging from a very sunny day to a dark night condition. Most automotive imagers are designed to work with a high dynamic range (HDR). The wider range is obtained by either using a higher number of bits per pixel, using logarithmic sensors response imagers, or combining consecutive images obtained by multiple exposures.

After acquiring images, CV algorithms usually analyze and transform the obtained information to create a more compact description of some features of the scene. The nature of the process output depends on every specific problem. In some cases the output can be pixel level information, in other cases the output can be associated to a group of pixels. In self-driving cars, to understand street scenes, algorithms need to locate, classify objects, and estimate some of their features, like distance to the car, whether they are moving, etc. Many of these tasks are increasingly implemented with Deep-Learning (DL) approaches [26], and more particularly, using deep Convolutional Neural Networks (CNNs) [27]. However, the computational complexity and the power consumption of DL approaches are highly demanding compared with classical machine learning (ML), and could have an impact on driving range of autonomous cars.

Some early prototypes on self-driving cars were consuming more the 2,000 W on computing alone [28]. More recent analysis [29], [30], [31] estimate that the power consumption of the computing platform can be below 200 W. Considering that the typical power consumption of the rest of the car is in the order of 10 to 20 kW; and that the power consumption of heating, ventilation, and air conditioning systems (HVAC) is generally around 500W [30], this value seems reasonable. Nevertheless, current computing power estimations are based on the execution of software stacks that are not at level 5 of autonomy, as defined by the National Highway Traffic Safety Administration (NHTSA) in the US. Full autonomy (level 5) might require much larger neural networks, with higher power requirements, potentially having a higher impact on the driving range.

Another concern with DL algorithms is the bias of models after training with specific datasets (as raised in raised [32]). This bias leads to an accuracy degradation when the input data is totally new, which could cause future traffic accidents.

Figure 5 illustrates the common functions required for an autonomous car and their dependencies. The top half of the figure describes the safety-critical functions which control the steering, the throttle, the brake, and the car's lights. The motion control module replaces the driver in controlling the steering wheel, the throttle, and the brake pedal. It operates with instructions from the maneuver planning algorithm, which defines the final and specific path that the car will be traversing. The path can be determined by the parking module or the trajectory planning module depending on the context. The output of these two modules should contain concrete instructions on how to move. They both get the input data from the two modules: (1) obstacle detection and (2) law enforcement described as follows:

- Obstacles could be pedestrians, vehicles, barriers, walls or other objects. They are detected by analyzing the surrounding environment provided by the surrounding model module that is built upon the fusing data from HD map, depth sensor, proximity sensor, and visual sensors.
- Traffic law enforcement module must determine the most appropriate path and speed. It obtains data from the outputs of lane detection, traffic sign recognition, and traffic lights recognition.

The trajectory planning module also requires information from the route planning and position module. The route planning module provides possible routes to the destination, based on the current position and traffic information. The current position could be calculated from the GPS, odometry and speed information.

The bottom half of Fig. 5 describes the other autonomous less critical functions. The ambient model module perceives the outside environment regarding light intensity and other special weather conditions such as rain, fog or snow. The light control module will activate the appropriate car's light

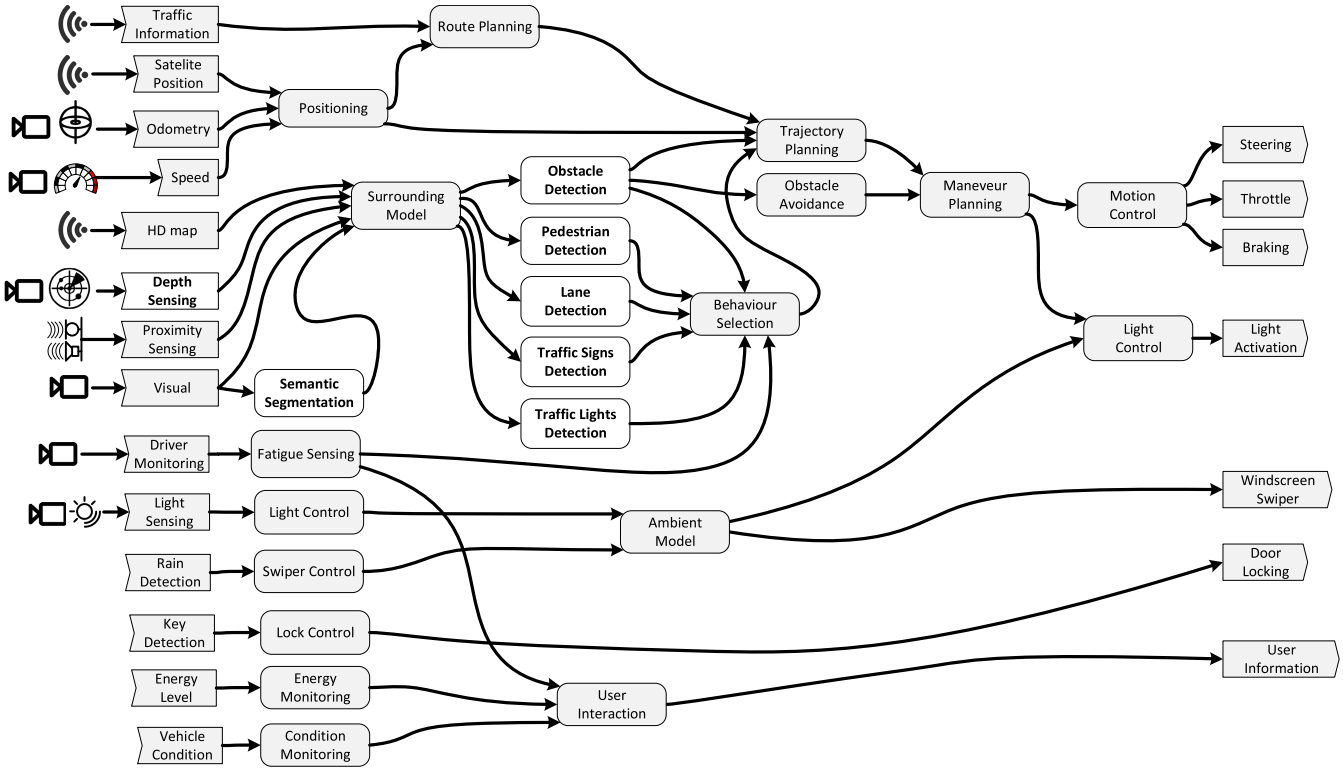


FIGURE 5. Dataflow diagram including some functions required for a self-driving car. Sensors are depicted in the left side, while actuators are depicted in the right side. The functions analyzed in more detail in this survey are pictured as rounded white boxes.

for each light or weather condition. This module also controls the turning or stopping of lights depending on the output of the Maneuver planning module. Similarly, the Wiper control module bases on the rain sensor information to activate the windscreen wiper. The lock control module will lock the doors when it detects a key is in place. And finally, the user interaction module could warn the driver regarding energy level, vehicle condition by graphic displaying or even alarming. Fatigue sensing helps to detect the drowsiness of a driver based on a camera mounted inside the car.

To identify the self-driving functions were FPGAs have more potential we have collected a number of research works from the literature describing FPGA implementations on that function. The collected papers are listed in table 2. Although the list is not complete, it actually shows the order of magnitude of the number of works for every function. The number of research papers on lane departure warning, obstacle detection, traffic sign recognition, and pedestrian detection is larger than other less critical functions. Some of these functions are also identified by [87] as fundamental for autonomous cars. This shows the interest of the FPGA research community to contribute with novel designs for the especially compute-intensive functions. Less compute-demanding functions are less addressed by FPGA research. Each of these problems is analyzed in the following sections to describe the problem itself and the most successful approaches to tackle them.

TABLE 2. Autonomous car problems with autonomy level and their FPGA-based implementations.

Problems	Required from level	Implementations
Driver drowsiness/fatigue	1	[33], [34]
Lane departure warning	2	[35]–[41]
Depth Sensing	2	[42]–[48]
Traffic sign recognition	3	[49]–[54]
Traffic lights recognition	3	[55], [56]
Pedestrian detection	3	[57]–[65]
Semantic segmentation	3	[66]–[71]
Object detection	3	[72]–[76]
Parking assistance	1	[77]
Trajectory planning	3	[78]
Steering control	2	[79]
Throttle control	2	[80]
Cruise control	2	[81]
Brake control	1	[82]
Internal communication	1	[83]
External communication	3	[84]
Sound system	1	[85], [86]

An important factor to consider in the scope of self-driving cars is the processing throughput usually measured in frames per second (FPS). The behaviour of the car depends on the outputs of these functions. The distance travelled by a car moving at a speed of 120 km/h during a frame period when processing images at 30 FPS is 1.1 m. This is not negligible value, and control algorithms could be sensible to those values. Thus, it is desirable to have the highest possible FPS.

A. EVALUATION METRICS

As stated in previous sections CV algorithms extract high level information from images. The obtained information can be associated to a sequence of images, to a single frame, to regions of an image, or even to individual pixels of the image. These different scopes need different methods to evaluate the accuracy of the methods so that an objective comparison among them is possible.

In many cases we deal with binary-class or multi-class classification problems. In those cases an error occurs when an item is classified as the wrong class. Binary classification problems are typically measured with precision (eq. 1), recall (eq. 2),, and f-measure (3).

$$Prec = \frac{TP}{TP + FP} \tag{1}$$

$$Rec = \frac{TP}{TP + FN} \tag{2}$$

$$F_\beta = (1 + \beta^2)\frac{Prec \cdot Rec}{(\beta^2 \cdot Prec) + Rec} \tag{3}$$

Many binary classification problems are affected by an algorithm parameter (e.g. some threshold value). In these cases, varying the threshold results on different values of precision and recall and precision can be expressed as a function of recall *precision*(*recall*). Precision vs. recall curves capture the relation between the two variables as recall swings from 0 to 1. In this situation, a common metric is the average precision (AP) as described by equation 4.

$$AP = \int_0^1 Prec(Rec) \cdot dRec \tag{4}$$

For multi-class classification, the detection performance is commonly measured by mean average precision (mAP), which is the mean of the AP of different classes.

$$mAP = \frac{1}{n}\sum_{k=1}^{n} AP_k \tag{5}$$

When not dealing with classification problems the error might be measured as the absolute difference of the prediction with respect to the ground-truth. A problem then, is to decide whether a relative value is is more meaningful than an absolute one. Two popular metrics are the root mean square error (RMSE, eq. 6) and the absolute relative error (eq. 7).

$$RMSE = \sqrt{\frac{1}{N}\sum_{i \in N}(D_i - GT_i)^2} \tag{6}$$

$$AbsRelError = \frac{1}{|\,N\,|}\sum_{i \in N}\frac{|\,D_i - GT_i\,|}{GT_i} \tag{7}$$

Another option is to use threshold criteria to determine if a value must be considered correct or incorrect. Then, accuracy can be computed by just counting the number of correct values from the total values (as in classification problems).

$$Accuracy = \frac{\sum_{i \in N}[|\,D_i - GT_i\,| > th]}{N} \tag{8}$$

When working with image regions ground truth annotations can be provided as a collection of pixel coordinates or a simpler description of an area such as a bounding box. However the boundary between a correct estimation and a wrong one is not so clear as in the pixel-level case. A common solution is to evaluate the intersection of the ground truth region and the estimated region over the union of both areas (see figure 6). The resulting value is a percentage of similarity between the regions.

In multi-class or multi-instance cases the mean of the IoU for every class or instance can be computed. Again, another common option in instance detection is to consider the values of IoU over a certain threshold (e.g. 50%) as correct estimations.

$$IoU = \frac{|D \cap GT|}{|D \cup GT|} \tag{9}$$

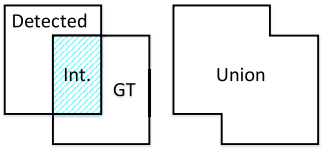


FIGURE 6. Example of the Intersection of a detected area with a ground-truth recording (left) and the area of their union (right).

In many CV classification problems we deal with imbalanced sets, i.e. the number of positives is orders of magnitude lower than the number of negatives (or vice versa). This is the case for detectors using an sliding window approach. Only a tiny fraction (if any) of the tested windows should report a positive. Performance in such cases is typically measured as false negative rate (FNR), also known as miss-rate, as a function of the false positive per window (FPPW) or false positive per image (FPPI).

B. DEPTH ESTIMATION

Depth map generation could be considered an early process required for more complex subsequent functions like obstacle detection. Although having less accuracy than LIDAR, depth estimation from stereo image sensors has a lower cost. Depth is inferred by matching regions of the image from one of the cameras into the other, as they collect images from two viewpoints of the same scene. Figure 7 depicts an example. In particular, the depth $Z(i, j)$ of each pixel, (i, j), can be calculated as the inverse of the disparity (distance) between image pixels corresponding to the projection of the same 3D point, as in eq. 10.

$$Z(i,j) = \frac{Bf}{d(i,j)} \tag{10}$$

where B is the distance between the two cameras, f is the focal length of the cameras and $d(i, j)$ is the disparity value of the pixel. The computation of disparity maps [88] can be split in 5 main stages: 1) rectification of the input images; 2) computation of the matching cost; 3) cost aggregation; 4) disparity computation/optimization, and 5) disparity refinement.

FIGURE 7. Example of the disparity map computed using SGM from stereo images provided in [89].

Matching pixels are usually obtained by searching horizontally in a disparity range and comparing image features of rectified images [90]. Image features can be made up of pixel intensities or more complex visual features like edges, HOG [91] or SIFT [92] features. Cost computation and aggregation are closely linked. Finally, disparity refinement aims at regularizing the disparity map, while handling object occlusion where matching can not be defined. The largest differences across methods concern the implementation of the later steps. Existing approaches can be categorized into local, global and semi-local methods.

Local methods aggregate the matching costs in support regions surrounding each pixel (usually using a weighted average) [92]. The disparity value that gives the lowest difference between support regions is set as the disparity value of the center pixel of the support region in a winner-takes-all approach (WTA).

Global approaches formulate aggregation and disparity computation as a global optimization problem [92]. The disparity map is computed as the minimum of a function defined as a weighted sum of the matching cost of all pixels and a regularizing penalty term that encourages neighboring pixels to have similar disparities.

Local algorithms prioritize speed over accuracy while global algorithms prioritize accuracy over speed. Semi-local approaches [93] offer the best compromise between accuracy and speed. They efficiently compute a global map from the initial local costs by simplifying the smoothness costs so that it can be optimized along different directions separately. This way, the processing can be parallelized and computed in real-time in GPU and FPGA-based systems.

Depth maps can also be inferred from monocular cameras. However, the accuracy of monocular depth sensing is still not parallel to that achieved by stereo vision [94]. Depth estimation can be tested using several benchmarks (see tables 3 and 4), which are typically annotated with labels coming from LIDAR, as it has better accuracy. Since estimating depth is not a classification problem, depth estimation datasets use numeric error metrics such as RMSE, AbsRelError, and others more adapted to the problem. For instance, the popular 2012 KITTI benchmark uses the Out-Noc metric, which an accuracy measure (see 8) with $th = 3$ and considering N from non occluded pixels.

TABLE 3. Stereo depth estimation.

Dataset	Year	Best Result	Metric	Score ↓
KITTI 2012 [95]	2012	LEAStereo [96]	Out-Noc	1.13%
Middlebury v3 [97]	2014	RAFT-Stereo [98]	Avg	4.74
KITTI 2015 [99]	2015	LEAStereo [96]	D1-all	1.65 %
DrivingStereo [100]	2019	ES-Net [101]	D1-all	2.42 %

TABLE 4. Monocular depth estimation.

Dataset	Year	Best Result	Metric	Score ↓
NYU-Depth V2 [102]	2012	BinsFormer [103]	RMSE	0.330
KITTI Eigen Split [104]	2014	BinsFormer [103]	Abs. rel. error	0.052 %
DDAD [105]	2020	SGDepth [106]	Abs. rel. error	0.098 %

As illustrated by table 3, current best methods are based on DL architectures. A common approach for stereo matching DL architectures (as used in LEAStereo [96]) is the use of two networks. The first one is used to generate a 4D feature map from input images. The second one is used to create a 3D cost map from which the final disparity value is selected. ES-Net tries to improve accuracy by using matching at different scales and using a mask to estimate occluded areas. And, RAFT-Stereo [98] uses an iterative approach, using RNNs, to refine disparity predictions.

On the other hand, DL approaches based on monocular images (see table 4) tend to use regression, or classification networks, or a combination of both (such as in [103]).

C. SEMANTIC SEGMENTATION

Semantic segmentation is the algorithm that assigns a label among a known set of class labels to every pixel of the input image. The classes can be anything meaningful for the context of interest. For the road context, they are usually pedestrian, cyclist, road, sidewalk, traffic sign, traffic light, etc.

As surveyed by the works [107], [108], there are different strategies to build a semantic segmentation systems for self-driving vehicles. The input of the system are the images from cameras. Depth information coming from stereo sensors, LIDAR, and HD maps can be fused to help segmenting the scene and to improve system's accuracy. In this survey, we focus on the approach that only uses input images from cameras. The type of expected output from semantic segmentation is illustrated by the example shown in Fig. 8.

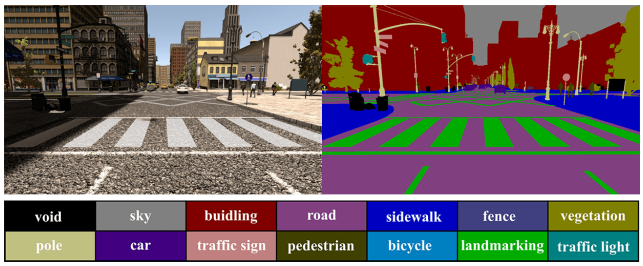


FIGURE 8. Semantic segmenation example from the Synthia syhthetic dataset [109].

TABLE 5. Semantic segmentation datasets.

Dataset	Year	Best Result	Metric	Score ↑
Camvid [127]	2009	DeepLabV3Plus + SDCNetAug [128]	mIoU	81.7 %
Pascal VOC 2012 [129]	2012	EfficientNet-L2 +NAS-FPN [130]	mIoU	90.5 %
Cityscapes [131]	2016	Multi Scale Spatial Attention [132]	mIoU	86.2 %
Kitti [133]	2018	DeepLabV3Plus + SDCNetAug [128]	mIoU	72.8 %

The most commonly used datasets to test semantic segmentation algorithms are listed in table 5. Mean IoU is commonly used to indicate the accuracy of a semantic segmentation system. It is the mean IoU of all the classes of the dataset. The initial approaches to do image segmentation were based on hand crafted methods that were exploiting low-level features of the image. Some examples are [110], [111], and [112]. The success of CNNs for classification allowed to progress to semantic segmentation, starting with seminal woks like Fully Convolutional Network (FCN) [113]. Since then, most semantic segmentation models have been based on CNNs, and can be divided into two type of networks: dilated and encoder-decoder.

Dilated networks [114], [115] benefit from avoiding downsampling and upsampling and, thus, maintaining the same resolution, incurring however in higher latency. Outstanding examples are PSPNet [116], or DeepLab V3 [117], which with its atrous pooling pyramid achieves a mIoU of 83.5% on Cityscapes and 82.9% on Camvid. Recently, attention has also been used to boost performance in an efficient manner with channel [118] or spatial [119] mechanisms.

Contrarily, encoder-decoder architectures focus on downsampling and upsampling to attain semantic and spatial knowledge. Works like U-Net [120] or Segnet [121] stand out for the use of skip connections, as well others such as RefineNet [122] or HRNet [123], which employs different branches to conserve the high resolution.

Recently, there has also been an increased interest into real-time and lightweight networks, which combine methods from real-time approaches, such as ICNet [118], with its cascade network, and lightweight operations, like depthwise separable convolutions [124] or shuffle cells [125], common in other tasks. An outstanding work in such line are BiseNet and BiseNet-V2 [126], which deliver an efficient solution for semantic segmentation.

D. LANE DETECTION

Lane detection (LD) is an algorithm that detects and extracts lanes on roads at a distance of several tens of meters ahead [134]. It is the key algorithm in many other applications such as lane departure warning systems (LDWS), advanced cruise control (ACC), lane keeping assist (LKA), lane centering assist (LCA), lane change assist, and turning assist. Vision sensors are likely used in lane detection systems since lane marks and are created for human vision. LD is a quite mature function integrated in many cars in the market. The ISO 17361:2017 standard [135] describes the specification for performance and testing procedures for LDWS. Although it does not require an especific frame processing time, it requires a distance threshold before lane departure alarm is issued.

Besides curved lanes, there are other challenges that a lane detection system must solve as such as occlusion, bad weather conditions (foggy, rainy, snowy), light condition (night time, in a tunnel), and bad quality of the road or the lane marks. LD systems must be able to detect landmarks on the road surface but should also be able to infer the drivable lanes, and determine the current lane (ego lane).

Table 6 lists some popular datasets for lane detection together with current best performing designs. Even more datasets for LD are described in [136]. Some are focused on the recognizing of the landmarks, while some are focused on the recognition of the whole lane area. In fact, selecting the appropriate ground-truth annotations becomes a challenge. Some annotation possibilities are illustrated in figure 9. In some datasets the ground-truth is provided at pixel level, by associating a class for landmarks, and another for the rest. In other cases, a multiclass annotation is provided to describe different types of marks such as lane marks, zebra lines, etc. In some cases the ground-truth is provided as a mathematical function that describes the lane lines.

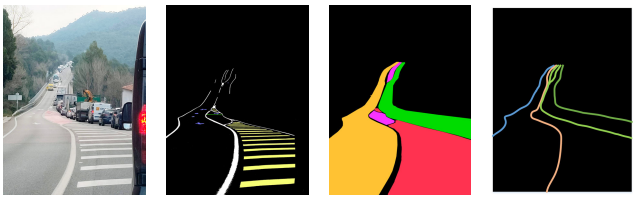


FIGURE 9. Fragment of a challenging scenario for lane detection with few straight lines, multiple road marks, and occlusions caused by traffic jam. Possible annotation types from left to right: pixel-level annotation of the lane and land marks, pixel-level annotation of the drivable areas, and functional descriptions of the lane marks.

When using the classification approach, the problem of LD becomes very similar to Semantic Segmentation. Thus, CNN-based methods are also generaly used. Spatial CNN (SCNN) [143] try to reinforce spatial information via inter

TABLE 6. Lane detection datasets. With respect to the type, LM denotes labeling of lane marks at the pixel level, Rd. denotes labeling of the whole road lane at pixel level, $f(x)$ denotes lane mark function labeling.

Dataset	Year	Type	Best Result	Metric	Score ↑
Caltech [137]	2008	$f(x)$	DNet-CNet [138]	F_1	94.6 %
KITTI [139]	2013	Rd.	DFM-RTFNet [140]	AP	94.05 %
TuSimple [141]	2017	$f(x)$	FOLOLane [142]	TPR	96.92 %
CULane [143]	2018	$f(x)$	CondLaneNet [144]	F_1	79.48 %
LLAMAS [145]	2019	LM	LaneAF [146]	F_1	96.0 %
BDD100K [147]	2019	Rd. / LM	ENet-SAD [148]	Acc.	36.56 %

layer propagation of the CNN. Self Attention Distillation (SAD) [148] use attention maps which are distilled from the lower layers of the network during training. Other approaches (like Enet [149]) use dilated convolutions to extract dense features on any arbitrary resolution.

When using curve fitting approaches, some networks are combined with HT line detection, such as [150]. But better than lines, the usual goal is to get a polynomial description of the lane lines. In this case, regression networks are typically used, such as in [151].

However, many DNN-based approaches are generally not friendly to embedded platforms due to their excessive number of parameters and computational cost. Classic machine learning approaches using hand-craft features are often less demanding and better suited for resource-constrained execution platforms. They are typically based on a pipeline with the following functions: Edge segmentation, line detection, and line tracking often using kalman filters. HT line detection does not accurately detect curved lanes, however it can be used if a curved line is approximated by a number of connected straigh lines as in [152]. Besides, HT can incorporate a lane model to detect curved lanes. Jung [153] proposes a linear-parabolic model. A linear model is used for the near field and a parabolic one estimates the far field.

E. TRAFFIC SIGN DETECTION AND RECOGNITION

Traffic sign detection (TSD) is the process of detecting traffic signs in an image, and traffic sign recognition (TSR) is the process of recognizing their meaning. Similar to LD, TSD and TSR are already present in some cars in the market. They are some of the applications that use image sensors as color is such a meaningful feature of the signs. For instance, red is typically associated with prohibitory signs. There are many challenges to getting a high recognition rate, such as the effect of lighting conditions, motion blur, fading of signs due to environmental conditions, rotation, and occlusion of signs [154]. Another important challenge is the large variety of traffic signs as many countries have specific traffic sign designs [155]. Once signs are detected and recognized, another challenge is to identify which ones are relevant to the car.

Some of these challenges can be observed in figure 10. The scene from a Russian street at twilight contains many visible signs at different distances. In addition to the illumination problems and a large number of visible signs, notice that one of the most important signs of the scene is the stop sign (CTOП), which is very different from the sign used in US and EU.

FIGURE 10. Multiple signs on both sides at the far sight, including information, warning, and prohibitory signs. b) Two attaching prohibitory signs and an information sign with a secondary sign attached.

TSD and TSR systems that can endure multiple illumination conditions and any weather conditions that can be found in any place on the earth should be trained in a large dataset. Tables 7 and 8 list some of the available datasets for detection and recognition respectively. Early datasets were covering restricted regions, such as Belgium, Germany, and US. Later datasets have covered a much larger geographic area and covered multiple weather and light conditions.

TABLE 7. Traffic sign detection datasets.

Dataset	Year	Best Result	Metric	Score ↑
LISA [156]	2012	ACF [157]	mAP	97 %
GTSDB [158]	2013	R-CNN Casc. [159]	F-Measure	94.42 %
Belgium TS [160]	2014	DRMAN [161]	AUC	93.34 %
CURE-TSD [162]	2017	DFR-TSD [163]	Precision	91.13 %

TABLE 8. Traffic sign recognition datasets.

Dataset	Year	Best Result	Metric	Score ↑
STSD [164]	2011	Mask R-CNN [165]	mAP	95.2 %
GTSRB [166]	2012	CNN with Spatial Transformers [167]	Accuracy	99.71 %
Belgium TS [168]	2013	Inception.v3 [169]	Accuracy	99.18 %
CURE-TSR [170]	2017	iELMNet [171]	Accuracy	98.63 %

Quite often, in TSD, a sample is considered positive if the confidence value is bigger than a given threshold. But

a positive is only a true positive if the intersection over union (IoU) between the detected box and the ground truth is higher than a chosen threshold (e.g. 50%). Otherwise, the detection is a false positive.

Therefore, the precision (eq. 1) and recall (eq. 2) values depend on the confidence and the IoU threshold. Different detection models are usually compared at the same IoU. During the AP calculation, the confidence threshold is selectively varied to determine the corresponding recall and precision. The AP is the average of these values of precision. This is the AP of a single class, the average AP of different classes gives mAP (eq. 5).

TSR methods have been recently surveyed in [172]. Classic machine-learning approaches work in a two-phase process that separates detection from recognition. The detection process starts with the segmentation step. The two most popular features for segmenting images in this application are color and edge. Once segmented, the recognition is done by extracting features and classifying them among a large number of classes.

Since RGB color space is sensitive to lighting conditions, images are typically converted to color spaces such as Hue Saturation Intensity (HSI) or Hue Saturation Value (HSV). However, the authors in [173] suggest that this is an unnecessary transformation as it does not provide a significant benefit compared with a normalized RGB space.

Color thresholding is often used to segment the input image and detect colors such as red, blue, yellow, and white. For edge segmentation, Sobel and Canny are the most widely used. Canny is typically preferred over Sobel due to its higher robustness at the cost of higher complexity.

In the recognition phase, the commonly used features are SURF [174], HOG [91], and Haar-like features [175]. A classifier will then take the feature as input to recognize the type of sign detected.

Classifiers such as SVM [176] and cascade classifier [175] are commonly used since they provide good accuracy at reasonable computing cost. The use of temporal information is a common refinement to reject some false positives caused by noise and increase accuracy. This is typically done by tracking sign instances with Kalman filters [156].

The classic methods were already saturating the initial datasets [168]. However, with the much larger sign base and expanded lighting and weather conditions of new datasets classic methods were not providing so good results and TSD and TSR increasingly adopted DL methods, specially CNNs.

One of the first works is [177], where they use a mix of color processing and a small CNN with a fixed filter part for detection and a learning component for classification. Over time, the structures have grown in complexity and new methods have been incorporated. Examples of such new methods are GANs [178], Transformers [167], extreme learning [171], and autoencoders [161] for recognition, and bounding box regression [179], [180] or multi-box non maximum suppression for detection. Currently, a common strategy extended through most TSD and TSR CNN applications is to employ a pre-trained CNN backbone like Inception [169] or DenseNet [171], among others. Another common procedure is the combination of detection with other tasks such as segmentation [165], for example, through the addition of multi-task heads [181], [182].

However, recently research has increased trying to compact networks for TSD and TSR [183], mainly due to the high computational costs of detection [165] and recognition. In the case of detection, YOLO (You Only Look Once), and ulterior versions, is a common component for fast detection [184], [185], [186]. In general, efficient NN techniques, such as efficient modules or quantization are being incorporated incrementally to TSD and TSR networks [187], [188], [189] and networks are being, in an easily increasing manner, ported to hardware accelerator platform [190], [191]. For the interested reader, there are reviews available in the literature [155], [156], [167].

Although the results from 8 suggest that TSR is a solved problem, an empirical test of market solutions suggests the opposite [192], as slight alterations of traffic signs (as found in reality) cause a significant drop of accuracy.

F. TRAFFIC LIGHTS RECOGNITION

Similar to TSR, traffic light detection (TLD) and recognition (TLR) are the processes of detecting traffic lights in a scene and recognizing their meaning, which is associated with their color and blinking state. TLD and TLR are also based on image sensors as color information is fundamental.

Fig. 11 illustrates some cases in which TLR needs to detect and recognize the traffic lights. A survey on vision-based TLD and TLR is presented in [193], but it is previous to the extraordinary DL advances from recent years.

FIGURE 11. Traffic lights from DriveU dataset.

Classic machine learning methods used to solve the traffic light detection problem by an sliding window scan to extract HOG features to later classify them [194]. However, such approaches suffer from large computation and multi-scale problems. To reduce the complexity, some subsets of the input image can selected with simpler methods and then checked with more complex ones [195].

These methods have been surpassed by the use of DL models, specially CNNs. Using CNNs, more features can be extracted compared with HOG and more non-linear

feature combinations can be learned compared with simpler machine learning classifiers. Most implementations aiming to real-time execution use 2D CNNs. One exception is the work of [196] that generates a feature descriptor of 1D which is feed as an input to a CNN. R-CNNs and Faster R-CNN are also used.

As in other domains, datasets (see table 9) have been expanded to include multiple traffic light types, and different lighting and weather conditions. Traffic light detection has been reported to be more challenging at night [197] as they can be confused with other kind of lights (including the lights from other cars). Starting from LISA, now most traffic lights datasets combine day and night scenes. Latest DL models achieve very high accuracies but require a significant computing power.

TABLE 9. Traffic light recognition datasets.

Dataset	Year	Best Result	Metric	Score ↑
LISA [198]	2015	2D CNN [199]	$F1_m$	99.92
Bosch Small [200]	2017	2D CNN [199]	$F1_m$	89.44 %
DriveU [201]	2018	Faster R-CNN [202]	mAP	90 %

G. OBSTACLE DETECTION

Obstacle detection (OD) is the application that identifies obstacles present in the drivable area that could become a hazard to the vehicle. OD is the foundation to implement more complex systems such as collision avoidance, situation analysis, cruise-control, and path planning. Camera-based systems usually bring higher spatial resolution than other active sensor systems such as LIDAR, RADAR, or ultrasonic sensors. Figure 12 shows an example with some obstacles on the road. Some might be expected (such as other cars, or pot-holes) while other objects (like a chair in the middle of the road) might be much less expected in the road context. A survey on real-time OD has categorized these systems into four different models based on the clustering strategy [203]. OD is very similar to semantic segmentation.

Depth information is often used to improve the accuracy of the systems. Based on the assumption that depth should be equal or larger while going from bottom to top in any specific column of an image, an obstacle can be detected only by analyzing the depth map. Nevertheless, this simple approach could not handle noisy data from depth maps. In [204], the authors proposed to use this strategy to detect free space on road scenes based on occupancy maps [205]. In an occupancy map, each cell represents the likelihood of occupancy at its coordinate and, ideally, only objects lying above the road are registered as obstacles. A dynamic programming technique is used to obtain the line separating free space from obstacles on the road. The heights of obstacles are also determined by the same dynamic programming approach.

Some obstacles can be static, but others (like a football ball crossing the street) could move. Tracking moving objects is also important to prevent collisions. Optical flow (OF),

FIGURE 12. Example of obstacles than can be found on a Vietnamese road. In the left part there is a car. Some meters away in the ego-lane there is a big piece of wood on the road. In the far distance there are a group of vehicles stopped.

also known as scene flow [203], is a popular technique to detect moving objects. The algorithm tracks the movement by comparing temporal continuous images but the problem is that everything is moving in the scene. To solve this issue, you have to find unexpected movement vectors. The OF algorithm has higher computational cost than stereo matching since it searches for corresponding points both horizontally and vertically. OF is usually fused with stereo matching to improve the robustness of the obstacle detection algorithm. This combined technique is referred as 6D-vision [206], in which velocities of moving pixels are tracked. Because of the high computational complexity of tracking 3D pixels, [207] proposed to represent an image using stixels instead of pixels. Stixels are rectangular vertical sticks representing objects on the street and provide a significant reduction of data volume.

Some specific datasets for obstacle detection are available: Lost and Found [208], Street Hazards [209] and RoadObstacle21 [210]). They typically provide a sequences of images and a corresponding depth map from the scene. Labels are commonly annotated at the pixel level, by assigning a class. However, because of the limited number of images of some of these datasets, many works also work with generic object detection datasets (like Pascal VOC [129] and COCO [211]).

As seen in the table 10, best performing obstacle detection systems are also commonly implemented with DL models.

TABLE 10. Obstacle detection datasets.

Dataset	Year	Best Result	Metric	Score ↑
Microsoft COCO [211]	2014	Faster R-CNN [212]	mAP	73.2 %
Pascal VOC [129]	2015	SSD ResNet50 [213]	mAP	74.1 %
Lost and Found [208]	2016	MergeNet [214]	IDR	82.05 %
StreetHazards [209]	2019	PAnS [215]	AUROC	91.1 %
RoadObstacle21 [210]	2021	Max. Entropy [216]	AUPRC	85.1 %

Faster R-CNN networks, are derived from R-CNN and follow a two step process. In a first step, a network proposes a region of interest. Then, for each proposed region a CNN is used to classify the region and perform a bounding box regression. Two step networks can be slow.

Single shot detection (SSD) networks follow a single step approach, providing a faster solution. A SSD network has two components the backbone, which is a pre-trained image classification network working as a feature extractor, and the

head, which performs the final classification for subdivisions of the image. In [213], the backbone is based on a ResNet50.

The MergeNet [214] works with 3 different networks, the first one works in Stripes (groups of columns), the second works with the whole image Context, an the third one combines the result from the previous ones.

Prototypical anomaly segmentation [215] is based on associating a confidence value with the predicted classification output. The idea is that low confidence will be related with unexpected objects. To effectively extract class-prototypes from the network, they use a cosine classifier, i.e. a classifier which uses cosine similarity between the input features and the class weights as class scores.

A useful technique to detect anomalies is using a method to evaluate out-of-distribution samples (OoD). In many classification networks the last layer is composed by a SoftMax layer, that outputs probabilities of the samples belonging to each class. By assessing the entropy of the probabilities of this layer we can infer how sure we are that a sample belongs to one of the training classes. If one class is clearly detected (probability close to 1) the entropy of the layer will be close to zero. Maximum entropy will happen with unexpected samples. This is the idea of [216] which is derived from DeepLabv3+.

H. PEDESTRIAN DETECTION

Pedestrian detection (PD) can be considered a sub-problem of OD, but because of its importance for safety, it is usually considered separately. As many visual functions, PD can be quite challenging because of partial occlusions, distance of the targets, illumination conditions and dynamic unexpected behaviours of the pedestrians.

Figure 13 shows a group of people in a street of Buenos Aires, Argentina, where some of these challenges are illustrated. Pedestrians appear in scene images with different sizes, the farther they are the smaller they appear and the closer they are the bigger they appear. Detecting pedestrians when they are relatively far is very important. If they are detected only when they are very close the is a high risk of hitting them since the processing latency will not allow to stop the car with sufficient margin.

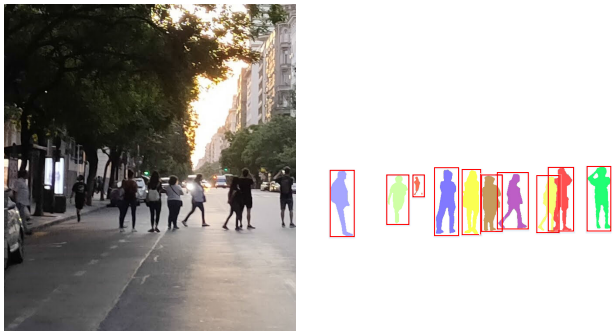


FIGURE 13. Example of a potential PD annotation labels. A self-driving car should detect them all, not only the ones in the very front of the car, but also the ones in the sidewalk, as they could jump into the drivable area.

TABLE 11. Pedestrian detection datasets. *MR* stands for Reasonable *MR*, MR^2 for Reasonable MR^2.

Dataset	Year	Best Result	Metric	Score
INRIA [91]	2004	CNN [217]	Accuracy ↑	98.4 %
Caltech [218]	2009	F2DNet [219]	MR^{-2} ↓	2.2 %
CityPersons [220]	2017	F2DNet [219]	MR^{-2} ↓	8.7

PD datasets (see table 11) usually use a metric which is based on a miss rate (MR) with respect to pedestrians having a size in pixels larger than a given threshold (typically 50 pixels, corresponding to 60 meters as calculated in [221]). In the INRIA dataset [91], one of the first available datasets for PD, the size of the persons in the images are rather large making it no so illustrative of the autonomous vehicle scenario. Later datasets (such as Caltech and CityPersons) have been considering smaller person sizes and more adapted to the real road challenges.

One of the first successful algorithms for pedestrian detection was based on HOG features and SVM classification. In the work from Dollar [218], the authors justified that measuring the miss-rate based on false positives per image (FFPI) is more suitable for a detection system, whereas the miss rate based on false positives per window (FFPW) is often used to evaluate classifiers. With FFPI, in 2005, the HOG algorithm was reported to be among the best algorithms surveyed with a miss rate of 68% [218] and 10.4% miss rate at 10^{-4} FFPW [91] in the Caltech and INRIA datasets respectively.

However, nowadays the best state-of-the-art algorithms [32] are based on NNs and the $MR?{-}2$ metric is used. The value of $MR?{-}2$ summarizes the performance of the detector using a log-average miss-rate. The calculation method is the average miss-rate under 9 FPPI values (range [0.01, 1.0]). The lower the score, the better the performance.

Best performing networks are Pedestron [32] and F2DNet [219]. Pedestron uses cascade mask R-CNN, which extends Cascade R-CNN to instance segmentation by adding a mask head to the cascade. F2DNet is a two stage detector with focal attention.

IV. BASIC ALGORITHMS

Many CV algorithms are based on basic image processing algorithms. These functional building blocks are often applied to many pixels of the input images and can consume a relevant share of of total required computing power. Their ubiquity and performance requirements makes them good candidates for FPGA implementations. In this section, we will briefly describe the main aspects of these algorithms so that, in section V, we can understand better how FPGAs take profit of their features to provide efficient implementations.

A. LOW-LEVEL ALGORITHMS

Many CV pipelines start processing the acquired images with algorithms working at pixel level or small region level. Those are often considered low-level algorithms by their simplicity

in terms of description. However, they might require high processing power as they are applied to many pixels.

1) COLOR SPACE TRANSFORMATION

Most color image sensors acquire 3 color channels (red, green, blue) in a Bayer pattern layout. To get RGB information for every pixel of the image bilinear or bicubic interpolation of missing color values is required. Some algorithms require to compute color similarity to segment the pixels of a given color. Other color spaces, like YUV, HSV, XYZ, or LAB are better than RGB for those purposes. Some color space conversions can be done by simple linear transformations. For instance, the conversion from RGB to YUV is defined by equation 11.

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.14713 & -0.28886 & 0.436 \\ 0.615 & -0.51499 & -0.10001 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (11)$$

Other color space conversions require slightly more complex non-linear transformations. For instance, the conversion of RGB to HSV color space is defined by equation 12.

$$\begin{aligned} \Delta &= \max(R,G,B) - \min(R,G,B) \\ V &= \max(R,G,B) \\ S &= \begin{cases} 0 & \text{if } V = 0 \\ \dfrac{\Delta}{V} & otherwise \end{cases} \\ H &= \begin{cases} 0 & \text{if } \Delta = 0 \\ \dfrac{G-B}{\Delta} \mod 6 & \text{if } V = R \\ \dfrac{B-R}{\Delta} + 2 & \text{if } V = G \\ \dfrac{R-G}{\Delta} + 4 & \text{if } V = B \end{cases} \end{aligned} \quad (12)$$

In most color spaces other than RGB, one of the components is associated with luminance. This is usually used to convert do greyscale. Another interesting pixel-wise operation that can be considered a color space conversion is binarization, which consists of reducing the required number of bits to store color to just 1, thus producing black and white images. A popular binarization algorithm is Otsu [110], but it requires to compute an Histogram of the image, which makes it difficult to implement using a streaming architecture.

2) LINEAR BLOCK OPERATIONS

Linear block operations on images are often applied in the form of convolutions. The general expression of a convolution is shown in equation 13, where w is a kernel matrix which is applied to every pixel of the input image f.

$$g(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} w(dx,dy) f(x+dx, y+dy) \quad (13)$$

The size of the kernel and their elements can be varied to adapt it to various purposes. Convolutions can be also used as filters. For instance, in Gaussian filters the values of the matrix follow a Gaussian distribution. They can be used to smooth the image. Gabor Filters banks, with their orientation aspect, are useful to detect some features independently of rotation transformations. Derivative filters such as Sobel filters can be used for edge detection and are less compute intensive than Canny edge detectors.

Convolutions are the basic building block of Convolutional Neural Networks (CNN), which have revolutionized CV in the last decade after AlexNet [222], and more than 30 years after their were initially proposed [223].

3) NON-LINEAR BLOCK OPERATIONS

In some cases, non-linear operations are interesting. Some of them are low level morphological operations which usually work on binary images, such as erosion, dilation, opening, closing, top hat, and others. Other operations are fill holes, Non-linear filters, like median filter (see eq. 14), or rank filters are also common.

$$m(x,y) = \underset{dx\in[-a,a], dy\in[-b,b]}{median} \{f(x+dx, y+dy)\} \quad (14)$$

Non-linear operations are common in classic machine learning based CV methods, but also they are part of DL approaches. One common step of CNNs is the max-pooling step, which downscales an image by selecting the max values from windows from it.

4) BLOCK SIMILARITY

In some occasions, CV algorithms have to check the similarity between two regions of an image. Block similarity is usually performed using either sum of abosulute differences (SAD), sum of squared differences (SSD), or any varian of cross-correlation function such as the normalized cross-correlation (NCC).

Block similarity is used, for instance, in stereo-matching, where patterns extracted from left image $P \in I_{left}$ are used to find the most similar region $R \in I_{right}$. In this example, SAD would be computed as eq. 15.

$$SAD(P,R) = \sum_x \sum_y |P(x,y) - R(x,y)| \quad (15)$$

Another interesting distance operation is the Hausdorff distance, which measures the maximum distance between any points from two sets (see eq. 16). This is sometimes used with binary images in template matching.

$$d_H(X,Y) = \max \left\{ \sup_{x\in X} d(x,Y), \sup_{y\in Y} d(X,y) \right\} \quad (16)$$

A more complete reference of block distance operations are described in [224].

B. FEATURE EXTRACTION

Feature extraction is the process that transforms input data in a more compact or meaningful representation so that subsequent processing steps can use them for detection or classification. Feature extraction methods are often compute

intensive and can be based on any combination of characteristics of the input images such as color, texture, shape, position, dominant edges, orientation, etc.

1) MATHEMATICAL TRANSFORMATION FEATURES

Some features are obtained by a mathematical transformation. The discrete cosine transform (DCT) [225], for instance, transforms from the spatial domain to the frequency domain and, because of the usual low frequency components found in real-world images, is highly used in image coding and compression.

Some transformations operate at the region level and are designed to obtain a feature vector of the region that is invariant to affine image transformations and, at the same time, is distinctive. If those requirements are met, features can be are useful to identify image regions and find correspondences between similar images. This is the case for feature extraction algorithms like Haar-like [175], HOG [91], SIFT [226], SURF [227], or ORB [228].

HOG features are obtained from image regions by computing a weighted histogram of the gradient orientation. The contribution of each gradient to each bin of the histogram is based on the magnitude of the gradient (see equation 17). The computation of HOG features is demanding as it involves to compute the square root to obtain the magnitude of the gradient and the arc-tangent to recover its orientation.

$$h_{bin} = \sum_{|\angle\vec{g}-\varphi_{bin}|\leq\pi/n} |\vec{g}| \cdot \frac{|\angle\vec{g}-\varphi_{bin}|}{\pi/n} \tag{17}$$

On the other hand, Haar-like features are much less demanding. They are based on adding the weighted values of two rectangular regions from the image I (see equation 18). The weights are typically 1 and -1 and the patterns are selected adequately depending on the object to detect. The examples of figure 14 could be used for car detection.

$$f = w_0 \sum_{p\in R_0} I(p) + w_1 \sum_{p\in R_1} I(p) \tag{18}$$

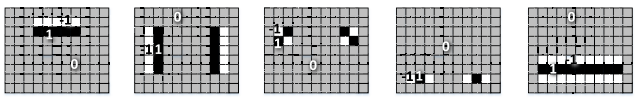


FIGURE 14. Haar-like features that could be used for car detection.

A method to speedup the computation of the sum of rectangular image areas is to first compute the integral image using equation 19, and then combine the values from the corners to obtain the sum.

$$I_{\sum}(x,y) = \sum_{i=0}^{y}\sum_{j=0}^{x} I(i,j) \tag{19}$$

Another popular method to obtain feature vectors is Speeded-Up Robust Features (SURF) [174]. An image region can be described by a Hessian matrix, which is build by a using Laplacian of Gaussians in different directions. When the determinant of the Hessian is positive it is more likely that the region can be uniquely identified. A σ parameter can be used to modulate the Gaussian function and effectively control the scale of the region. This is illustrated by (20), where $D_{xx}(x,y,\sigma)$ is the Gaussian of the second order derivative on x and scale σ. The maximum value for different values of σ is selected.

$$H(x,y,\sigma) = \begin{bmatrix} D_{xx}(x,y,\sigma) & D_{xy}(x,y,\sigma) \\ D_{xy}(x,y,\sigma) & D_{yy}(x,y,\sigma) \end{bmatrix} \tag{20}$$

The main idea behind SURF is that Laplacians can be substituted by box filters and easily computed using the Integral Image. Box filters (see figure 15) are convolutional filters with discretized values where integral images can be effectively exploited.

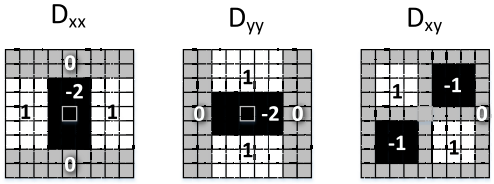


FIGURE 15. Box filters.

Another popular transform is the Census transform [229], which assigns a binary string to every pixel of the image depending on whether the values of neighboring pixels are bigger than the pixel. If we consider a 3×3 kernel, the 8 neighboring pixels produce an 8 bit string. Then the census transform can be expressed as an 8 bit number and the similarity with other 3×3 blocks can be computed with Hamming distance. This simplicity makes it very interesting for FPGA implementations that require region matching, such as stereo matching.

2) MODEL FITTING FEATURES

In other cases, we are interested in detecting the parameters of some models that we assume exist in the image. This is the case of LD where we can assume that we have pixels belonging to lane marks that should be part of straight lines or a polynomial or a higher degree. We can also assume that we will be unable to detect only pixels of the lane marks, so we must tolerate a level of noise, or, in other words, a number of outliers from the estimated model.

A possible solution is to use Hough transform (HT) [230] which is a voting strategy. Some lane detection algorithms use HT for line detection. Candidate pixels to belong to an unknown line are selected. The line equation can be expressed as either $y = m \cdot x + n$ or in polar form as equation 21.

$$\rho = x\cos\theta + y\sin\theta \tag{21}$$

The later form is better to support vertical lines. The principle behind HT is that every candidate point votes for all points meeting the equation in the two-dimensional vote space given

by θ and ρ parameters. The points with the higher values in the voting space will likely define the most probable lines.

Another option is to use the random sample consensus (RANSAC) method [231]. RANSAC is based on randomly selecting several points to define a function equation. The rest of the points are classified as inliers or outliers of the equation defined with the random points. We repeat the procedure several times selecting the equations with less outliers. RANSAC works well if a lot of the points are part of the equation, otherwise the number of outlier might be too large in all cases to give a valuable guess.

C. CLASSIFICATION

In some cases, classification is a final step in a recognition algorithm. There are some classification algorithms that have been implemented in FPGAs. Saidi conveniently surveys them in [232]. We will briefly describe some of them.

Template matching is a simple classifier that works directly on feature vectors. For each class c, we can have several reference feature vectors $T_{c,i}$. As explained in section IV-A4, a similarity check against the reference vectors can be done to select the reference class that gives the minimum difference (see equation 22).

$$c(I) = \underset{\forall c,i}{\operatorname{argmin}} \mathit{Diff}(T_{c,i}, I) \tag{22}$$

Template matching might be used, for instance, in traffic sign recognition, since traffic signs are standardized in shape and size. The algorithm can take the ROIs of the image and compare them with templates in a database to find the closest match. If the database is large, this method may involve a lot of computation.

Support Vector Machine (SVM), also known as Support Vector Networks [176], is another classification technique that attempts to build a hyperplane model (23) in a high dimensional feature space that is able to separate the elements from two classes. SVM can be extended to support multiclass classification, but in its original form addresses the problem of binary class classification.

$$y(\vec{x}) = \vec{w} \cdot \vec{x} + b \tag{23}$$

The advantage of SVM is that it generalizes better than template matching. Its simplicity makes it interesting for FPGA implementations.

Then, there is a family of classifiers that combine simple (or weak) classifiers. Decision Trees, Random Forest or AdaBoost, to name a few. Most classifiers require a training phase in which the system parameters are adjusted to correctly classify the input data. In supervised training, data selection and annotation is a crucial step of the training process, and it can have a large impact on the final accuracy of the system. A recent survey [233] provides more details on the topic. Neural networks are also used for classification, but will be described in more depth in the next subsection.

D. NEURAL NETWORKS

Neural networks can work both for feature extraction and classification. In fact, the boundary between both functions is so thin that they are often perceived as a single block.

Artificial Neural Networks, or simply Neural Networks (NNs), are bio-inspired circuits that try to mimic the function of the neurons in the brain. The function of a neuronal layer $i+1$ is often described by an activation function α on each elements of the result from the product between the input vector $\vec{x_i}$ of layer i and a weight vector $\vec{w}$ plus a bias $\vec{b}$ (see equation 24).

$$x_{i+1} = \alpha(\vec{w_i} \cdot \vec{x_i} + \vec{b_i}) \tag{24}$$

In the intermediate layers of the network, the input vector of a layer is composed by a concatenation of the outputs from the neurons of previous layer. The values of vectors $\vec{w}$ and $\vec{(b)}$ are learned during network training. The activation functions α can be non-linear functions such as sigmoid, hyperbolic tangent, rectified linear unit (ReLu), etc. However, some of these functions suffer from a vanishing gradient problem when training is performed. ReLu is less affected by this effect, and hence has become very popular in many networks.

NN outputs can be used for classification or regression. In regression networks, the outputs can be directly interpreted as function values. In classification networks, each output is typically associated to a class, and the value of the output is associated with the likelihood of the input to belong to that class. An special additional Soft-max layer is usually added to enforce that the outputs sum to 1 and can therefore be interpreted as probabilities.

Before the eclosion of deep-learning, multi-layer perceptron networks were the most widely used NNs. Convolutional neural networks (CNNs) later provided a major step forward in performance for many CV problems [222].

A central component in CNNs are convolution layers that produce a number of output channels associated with a convolution kernel which is applied to the input image. A kernel k is a matrix of (usually) small dimensions that is applied to the input image I as sliding window filter. In the convolution equation 25, x and y the coordinates of the pixels, and the ranges $[-a, a]$ and $[-b, b]$ configure the two-dimensional convoluted region in the original image. The parameters of the kernels are learned during training. This process allows to recognize certain features, patterns or representations in the input of each layer.

$$g(x, y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} I(x + dx, y + dy) \cdot k(dx, dy) \tag{25}$$

An important component in CNNs are pooling layers, which reduce the dimensionality of the output layer by selecting a subset of the input values. The reverse operations (unpooling, and deconvolutions) can also be used to increase the dimensionality of subsequent network layers.

Network architectures can be built for specific purposes by connecting layers of different components and

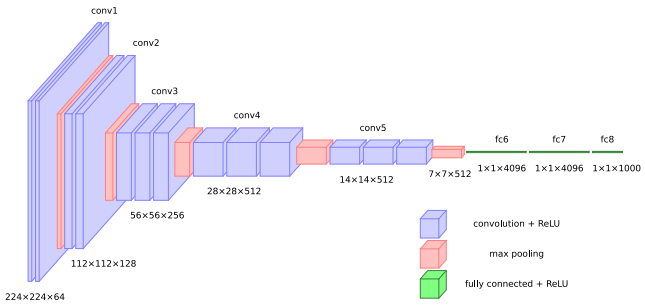


FIGURE 16. Diagram of the architecture of VGG16 [234], a popular CNN classification model.

dimensions. Possible combinations are infinite, but computational demand can easily grow to intractable levels in big networks. Hence, researchers have studied network architectures that achieve good results with reasonable computing demand. Some popular architectures are AlexNet [222], VGG [234], MobileNet [124], ResNet [235], SqueezeNet [236], DenseNet [237], and new ones are continuously proposed.

CNNs have some tolerance but are not totally invariant to scale, rotation and translation differences. To increase their performance under these conditions, special networks (known as transformers [238]) have been proposed that detect the necessary geometric transformation to be performed on the input image so that the following steps can be effectively done under most conditions.

The dimension of the different layers of a network depend on the problems addressed. Classification of whole images can follow a big-to-small funnel approach if the number of final classes is small enough. This can be clearly seen in the figure 16, which depicts the VGG16 NN architecture. When pixel-level output is required, the initial and final dimensionality are the same. The goal then is to reduce the dimensionality as much as possible in the midle of the network as is done in encoder/decoder networks like UNet [120].

In some cases the number of outputs can be variable. This is the case with detectors. Many classic machine learning detector algorithms are based on multiscale sliding windows. For DNNs, this approach is prohibitive. R-CNN [239] narrows down candidate regions using a selective search step. Low level semantic segmentation is performed on the input image using classical methods to obtain candidate bounding boxes. The subset of candidate regions is then used as inputs for classification and bounding box regression.

Some optimizations have been made on this idea. First, in fast R-CNN [240] the input image is directly processed by a convolutional layer and the selective search is done with the output of this layer. In faster R-CNN, an additional NN (region proposal network) is used to propose regions.

Another approach to reduce the number of candidate regions is to work with a predefined grid, as done by SSD [179] and YOLO [241]. For each region on the grid, a bounding box regression and a classification stage is performed.

In instance segmentation, a bounding box is not sufficient and instances are often identified at the pixel level. Hence, an extra network head is required to identify each pixel of the region as belonging to the instance. This exactly what Mask R-CNN [242] networks do, they extend R-CNN to output a class, a bounding box, and a mask at pixel level for each candidate region.

Recent advances to improve the accuracy of instance segmentation include the use of a cascade of multiple networks. Different Mask R-CNN detectors can be trained aimed with different mIoU objectives and then be combined.

NNs can be used straigh away as a whole functional block, but in some cases they are integrated as a building block of more complex designs, such as done in some depth estimation or lane detection algorithms.

V. FPGA BASED IMPLEMENTATIONS

This section compares the experimental results of the published works on vision-based applications for autonomous cars using FPGAs. As seen in previous sections, the best accuracy in many problems is often achieved by DL methods, especially CNNs. However, NNs are often very compute-intensive and the trade-off between accuracy and execution time should be analyzed in detail since the safety of hard real-time systems depends both on the accuracy of the systems and their execution time. As pointed out in previous sections as well, energy efficiency is also a factor to consider to ensure it does not affect significantly the driving range.

A. COMMON DESIGN TECHNIQUES

The flexibility of FPGAs is paid by some overheads in area (number of transistors), capacitance, delay (clock frequency), and leakage current with respect to custom designed integrated circuits (ASICs) that are manufactured in foundries with latest technology nodes [243].

To propose system implementations that can provide a competitive alternative to solutions implemented in other platforms, FPGA designs must overcome this initial overhead and add additional gains. This is generally done by exploiting the spatial computing [244] paradigm, and a set of common techniques that we briefly describe in the following subsections.

1) CUSTOM MEMORY BUFFERS

The flexibility of FPGA allows to implement memory buffers addressing the needs of specific applications. In line buffers (see figure 17) the goal is to act as a custom cache memory, the memory access pattern is regular and high data locality. The trick is that in long shift register we can predict the exact positions were the recently accessed data is located. This allows to avoid many memory accessed since the values are already available in the buffer.

Other custom memories widely used in FPGAs are FIFOs. They allow transparently handle situations of transient mismatch between the data-rate of a data producer and a consumer.

Custom memory strategies in FPGAs are very often using the availble dual-port memories, which allow simultaneous

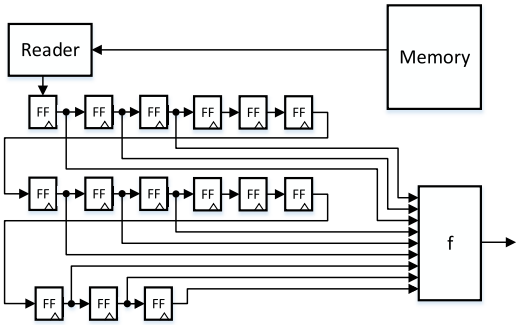


FIGURE 17. An illustration of line buffer and window buffer.

read and write operations in two endpoints. When a higher number of ports are required, arbitration circuits must be added.

2) TASK PARALLELISM

The purpose of task parallelism is to divide a function into small tasks and dedicate a separate computing system to each of them. By assigning independent hardware to execute each task, FPGAs can not only benefit from having multiple concurrent processing units, but also specializing each processing unit to its assigned task.

Pipelining is a form of task parallelism. It can be applied with several granularities in FPGAs. Pipelining requires storage between the computing elements. The nature of FPGA LEs (see figure 3), which are already contain registers for storage, makes them well suited for implementing fine-grained pipelines. At the coarse-grained level, dual-port memories and ping-pong buffers also help to store the intermediate values of task-level pipelining.

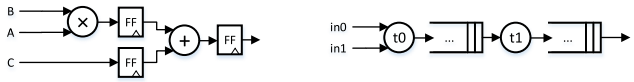


FIGURE 18. Example of task parallelism at the fine-grain (left) and a coarse grain (right) levels.

3) DATA PARALLELISM

When working with independent data, there is the possibility of creating multiple computing units that work with several independent sets of the data simultaneously. If the data storage is centralized, it only make sense to create them if the cost of computing is high enough so that the data non-concurrent accesses represents a tiny fraction of the computing time. Otherwise the speedup achieved by hardware replication is not worth it.

4) NUMBER REPRESENTATION

Floating point numbers are generally used in many algorithms because they offer a flexible way to forget about the valid number range of numbers. The flexibility is paid in higher latency, which becomes a major drawback for iterative algorithms.

On the other hand, integer arithmetic operations (add, sub, mult) are commonly implemented in single-cycle designs. Fixed point arithmetics can be implemented easily using integer operators. So moving an algorithm to fixed-point has a two-sided benefit: better performance (as operations take just 1 cycle) and less resource usage (as integer arithmetic units are much simpler than floating point ones).

Canonical Signed Digital (CSD) representation is another method to make arithmetic operations more compact. It is very useful to minimize the resources used when operating with constant values.

All these techniques can be exploited by FPGAs to reduce resource usage and reduce the latency of iterative algorithms.

5) LOOKUP TABLES

FPGAs are already based on LUTs with a small number of inputs and it is easy to them to reach larger sizes, either by combining LUTs or using embedded memories. Many complex mathematical functions can be optimized by having pre-computed values that are stored in lookup tables (as depicted in 19. With this approach, several discrete values of a complex function f can be retrieved from memory in a few cycles instead of using the high number of cycles required to compute f.

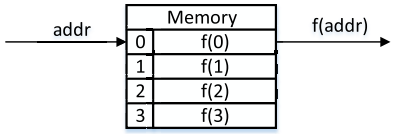


FIGURE 19. Example of a lookup table containing four precomputed values of a complex function f.

A very popular algorithm that benefits from LUTs is CORDIC algorithm [245] to compute vector rotation, and thus trigonometric functions such as cos and sin. A vector rotated by angle β can be computed by multiplying by a rotation matrix (26). We can select a special set of angles γ_i so that their tangent is of the form $\tan(\gamma_i) = \pm 2^i$ and store them on a table. Then any arbitrary rotation by an angle β can be decomposed as a sequence of rotations by the precomputed angles γ_i. The benefit of this approach is that multiplications can be implemented by shift operations, saving a large amount of resources.

$$\begin{bmatrix} x_r \\ y_r \end{bmatrix} = \frac{1}{\sqrt{1+\tan^2\beta}} \begin{bmatrix} 1 & -\tan\beta \\ \tan\beta & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{26}$$

6) SYSTOLIC ARRAYS

Systolic array (SA) architectures were proposed before the birth of FPGAs [246] to favor multiprocessing while minimizing data transfers and improving layout regularity [246]. Computing in SA can be interpreted as pipelining in a higher dimension space, as processing and data is generally

synchronously pumped into a 2D array of processing elements (see figure 20).

Although the systolic array has a limited dimension, many algorithms can be partitioned in smaller blocks adapting to the size of the array to take profit of its higher level of parallelism. Matrix multiplication is one of such algorithms, and neural networks mainly rely on multiplying layer input matrices with layer weights. FPGAs allow to specialize the processing element of the SA. So, for a long time, they have been widely used for many applications, including NNs [247].

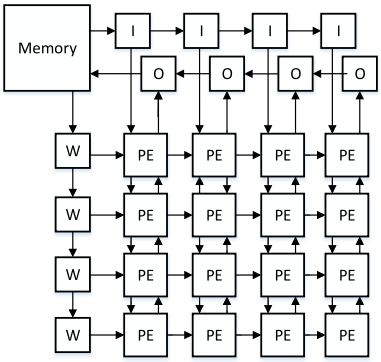


FIGURE 20. Example of a Systolic Array design.

B. IMPLEMENTING NEURAL NETWORKS IN FPGAs

Given the importance of NNs in many of the autonomous car functions, it is interesting to review what methods can be used to implement NNs in FPGAs. This job has already been covered on some recent surveys [7], [15], [248], [249], [250], [251], but we will briefly summarize the main aspects.

Works such as [252], [253], and [254] develop frameworks for efficiently deploying CNN on FPGA platforms. Currently, there is a trend to embed deep CNN models in FPGAs, along with their compression and quantization, as explored in different works ([255], [256], [257]). CNN structures and algorithms suitable for FPGAs are presented in [236], [252], [258], [259], [260], [261], [262], and [263].

The computation required in the inference phase of NNs is dominated by matrix multiplications. While CPUs and GPUs try to exploit vector processing and multiprocessing to speedup matrix multiplication, a new breed of computing platforms relies on SAs to increase performance. The advantage of SAs is that they provide many processing elements that can execute in parallel and reduce the need to store intermediate results in memory. These computing platforms are named either tensor processing units (TPUs), neural processing units (NPUs), or DL processing units (DPUs).

The flexibility of FPGAs allows any architecture to be implemented within the limits of available device resources. Although soft-core processors (Soft-CPUs) are very popular, Soft-GPUs are still rarely used in FPGAs. On the other hand, main FPGA manufacturers provide Soft-DPUs under different marketing names (e.g. OpenVino, Vitis-AI) to accelerate NN inference. Simplification of the NNs is a common step on FPGA NN frameworks to reduce computational load. It generally involves quantization and pruning. The exectutions is orchestrated by the host CPU that sequentially executes the different layers of the NN by invoking the Soft-DPU as a NN accelerator.

An alternative approach is to create NN accelerators from C/C++ code by using HLS such as in [252] and [254]. The goal of an efficient NN FPGA implementation is to execute as many simultaneous operations as possible while reducing the access to external memory. Given the nature of matrix multiplication operations an obvious choice is to make use of the FPGA's multiply-adder resources, as they are heavily used in matrix multiplication and convolutions.

The level of parallelism on NNs is very high. In CNNs, the different elements of a layer output vector are independent and can be computed at the same time. The convolutions for the input channels are totally independent from one another. The whole network processing can be thought as a very deep pipeline. As convolutional kernels are typically applied with an sliding window approach, line buffers can be used to optimize the required memory accesses. However, the different dimensions of the network layers increases the difficulty of producing generic modules for convolutions in FPGAs [257].

In general, the problem of working with CNNs in FPGAs is that big models are rarely fully unrolled in the FPGA because of resource limits and the difficult of creating processing units that are general enough to be reused by different layers of the network. Differences in memory access patterns and the variety of data used by different layers often make it difficult for an implementation to deliver good performance while preserving energy and resource efficiency [254].

Xilinx provides an environment called Vitis-AI that allows the migration of networks described in the most popular frameworks for execution in Soft-DPUs. OpenVino handles a similar process for Intel devices. All these frameworks deal with network optimization, usually quantization and pruning. Other NN implementation methods try to increase the level of network unrolling. In the case of BNN networks, the resource savings due to the reduction of the number of bits of the network parameters and the use of bitwise binary operations make it possible for some networks to be fully implemented inside the FPGA device. The framework FINN [8] enables BNNs designed with Theano to be implemented on Xilinx FPGAs. LogicNets [264] tries to unroll networks described with Pytorch and convert artificial neurons to truth tables during the synthesis process. This makes it possible to implement fully unrolled high-performance networks as well.

The performance, latency, and energy efficiency of FPGA-based NNs is very competitive when networks can be fully unrolled into the device. However, some works [251] suggest that the achieved speedups for non-trivial cases are modest.

C. IMAGE-BASED DEPTH SENSING

Despite the progress in monocular depth estimation, self-driving cars mostly rely on stereo matching to estimate depth because of its superior performance [94].

Many recent stereo matching solutions are based on DL. DL approaches are perceived as a black box where researchers still have no clear explanation on what are the main cues that systems use to infer depth. On the other hand classic machine learning approaches are better understood, which is a good aspect for FPGA platform optimization.

As described in section III-B, depth from stereo images can be inferred from the disparity between regions of the images. Local matching algorithms select the minimum matching cost MC (or maximum similarity) along the epipolar line as equation 27.

$$D_{LM}(x, y) = \min_{\forall d} MC(x, y, d) \tag{27}$$

One of the preferred methods for block similarity is census transform. As it uses binary vectors, the SAD dissimilarity operation is equivalent to hamming distance (HD), which require very simple arithmetic operations. In [43], the authors complement the CT feature vector with the gradient magnitude to increase the accuracy of the matching operation. For, CT-only based approaches, the cost of a disparity on a certain location is computed by equation 28.

$$MC_{L,R}(x, y, d) = HD(CT_L(x, y), CT_R(x - r, y)) \tag{28}$$

Line buffers are generally used to reduce the required memory bandwidth to compute the block features. Nevertheless, the time complexity evaluating the cost for all possible candidate disparities is still $\mathcal{O}(w \times h \times d_{max})$.

To reduce this time complexity, many FPGA implementations ([42], [43], [44], [47]) use a data-parallel approach to unroll the loop along the d values since all matching costs are data independent. Thus, multiple matching cost units (MCU) can be replicated as depicted by figure 21.

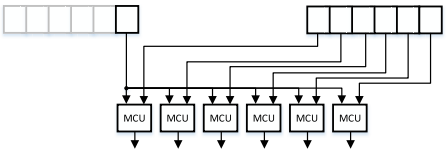


FIGURE 21. Illustrative example of Loop unrolling along d to speedup cost computation in stereo matching.

Many stereo-matching FPGA implementations use a SGM approach, where the MC tridimensional matrix is stored in memory and a dynamic programming approach is used along different paths $r = (r_x, r_y)$ to check the consistency at a global scope. Each path direction r requires its own tridimensional additional tables described by equation 29, where P_1 is a constant cost that penalizes selecting neighbouring disparity values along the path, and P_2 penalizes bigger gaps.

$$\begin{aligned} L_r(x, y, d) = MC(x, y, d) \\ + \min \begin{cases} L_r(x - r_x, y - r_y, d) \\ L_r(x - r_x, y - r_y, d - 1) + P_1 \\ L_r(x - r_x, y - r_y, d + 1) + P_1 \\ \min_{\forall i} L_r(x - r_x, y - r_y, i) + P_2 \end{cases} \\ - \min_{\forall k} L_r(x - r_x, y - r_y, k) \end{aligned} \tag{29}$$

The final disparity value is selected from the one that minimizes the sum of the costs along all considered paths as described by equation 30.

$$D_{SGM}(x, y) = \min_{\forall d} \sum_r L_r(x, y, d) \tag{30}$$

To reduce the memory requirements of the MC cost table, authors in [42] propose to store the k more significant values instead of all possible d elements, and return the P_2 when no element is on the table.

However, this is not enough to reduce the spatial complexity of the algorithm which is still $\mathcal{O}(x, y, d)$ due to data dependencies caused by the multiple paths followed to build the L_r tables. If all paths of figure 22 are computed the algorithm requires to store the whole MC and L_r tables to generate D_{SGM}.

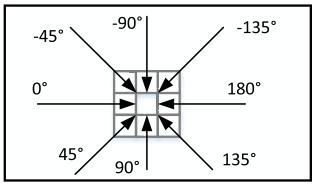


FIGURE 22. Multiple paths of SGM algorithm.

As a row-scan loop is generally used, a clever optimization is to work with a subset of paths that depend on the values computed in previous iterations of the loop (i.e. pixels from either previous row or same row and previous column). Many real-time implementations (like [42], [44], [46], [47]) limit the number of aggregation paths to four or five so that a single pass of the input images is sufficient and memory requirements are reduced to just one row of the matrices.

Best performing FPGA implementations (see 12) use such SGM approach, but there are some exceptions using a local approach that rely on additional features to increase the accuracy. For instance, in [43], the authors combine the gradient magnitude with the census transform to provide a richer matching cost MC. They also compute a vertical aggregation image cost to refine the decision and a final consistency check between left and right disparities.

In [45], the authors use the random PatchMatch approach which is based on a random selection of the disparity, followed by and evaluation of the error and a propagation of best matches to neighbouring locations. The original algorithm is iterative, but the authors successfully propose a non-interative

TABLE 12. Performance of most relevant recent depth sensing implementations on FPGA. The Bad-pixels metric informs of the percentage of pixels having a computed disparity higher than a threshold from the ground-truth.

Ref	Author/Year	d_{max}	Method	FPS	Dataset	Bad pixels ↓
[42]	Schumacher 2014	160	SGM	199	KITTI 12	18.06 %
[43]	Dehnavi 2017	128	LM	50	KITTI 15	9.22 %
[44]	Rahnama 2018	128	SGM	72	KITTI 15	9.9 %
[45]	Zhang 2019	128	LM	60	KITTI 15	6.88 %
[46]	Chen 2020	64	BNN-SGM	114	KITTI 15	6.37 %
[47]	Shrivastava 2020	64	SGM	322	KITTI 15	11.4 %
[48]	Wei 2022	256	SGM	70	KITTI 12	7 %

TABLE 13. Synthesis results of most relevant recent depth sensing implementations on FPGA.

Ref	FPGA	Freq. (MHz)	RAM (kbits)	DSPs	LUTs	FFs	Power (W)
[42]	Virtex-5	91.7	1,076	n.a.	109,072	n.a.	n.a.
[43]	Virtex-6	110	765	896	119,320	242,610	n.a.
[44]	Zynq-7000	100	6,036	n.a.	165,480	177,066	3.94
[45]	Kintex-7	148.5	151	0	53,190	40,980	n.a.
[46]	Stratix-V	91	7,891	0	350,354	57,785	3
[47]	Virtex-7	100	641	0	211,260	n.a.	n.a.
[48]	Zynq Ultr.Scl.+	200	16,164	514	227,677	74,697	10.09

alternative exploiting pipelining with good error rate results. In [44], the authors focus on power efficiency of the implementation using high-level synthesis HLS. They analyze the relation between window width of Census transform and the power consumption to finally obtain a system consuming less than 4W at 72 FPS.

The most accurate implementation on FPGAs [46] achieves an bad pixel rate of 6.37% on KITTI 2015. The main difference in this design is the binary neural network for the matching cost computation. The rest of the system is similar to a SGM-based algorithm. The authors use the common techniques to reduce the hardware cost of a DNN-based model such as: weigth quantization, fixed point numbers, and prunning of channels and layers to reduce the network size. The design runs at a speed of 114 FPS with a power consumption of only 3W, thus, providing an energy efficiency of 38 frames per Joule.

Finally, [47] presents the best throughput. The design exploits data parallelism by computing multiple costs simultaneously, and task parallelism to implement a pipeline for cost aggregation. To solve the data dependencies and increase the initiation interval of the pipelined design the relax the dependency of the terms used in equation 29 to use closely already computed values instead. They argue that the accuracy degradation (to 11.4 % in KITTI 2015) is acceptable.

However, given the trade-off shown in figure 23 and as suggested by Zhao's results [265], the main advantage of FPGAs for this problem is expected to be energy efficiency rather than performance or accuracy.

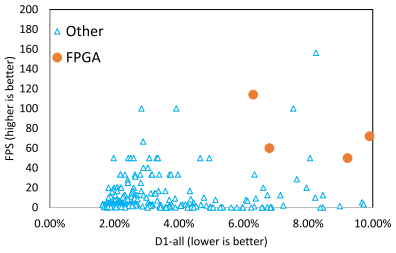


FIGURE 23. Trade-off between FPS and accuracy of stereo matching solutions on the KITTI 2015 dataset. FPGA designs are shown in red circles. Other platform designs (CPU, GPU) are shown in blue triangles.

D. SEMANTIC SEGMENTATION

Semantic segmentation aims to label each pixel in the image with a category or class name. Most FPGA implementations use CNNs. Similar to other CNN-based algorithms, implementations for semantic segmentation on FPGAs typically adopt compression techniques such as pruning and quantization [255]. Also, lightweight network models such as MobileNet [124] are preferred. FPGA-based designs normally use low-bitwidth fixed-point data to save resources. Adders are preferably implemented with LUTs to avoid using DSPs.

TABLE 14. Performance of most relevant semantic segmentation implementations on FPGA.

Ref.	Author/Year	Network	FPS	Dataset	mIoU ↑
[66]	Liu 2018	Unet	17	Cityscapes	61 %
[67]	Sada 2019	MobileNet	139	Cityscapes	64.2 %
[68]	Shimoda 2019	AlexNet	165	Camvid	45 %
[69]	Huang 2020	Segnet	7	Pascal VOC2012	44.7 %
[70]	Miyama 2021	Unet	123	Camvid	67.8 %
[71]	Jia 2021	E-Net	32.9	Cityscapes	63.09

TABLE 15. Synthesis details for most relevant semantic segmentation implementations on FPGA.

Ref.	FPGA	Freq. (MHz)	RAM (kbits)	DSPs	LUTs	FFs	Power (W)
[66]	Zynq ZC706	200	19,620	900	218,600	437,200	9.6
[67]	Zynq ZCU102	300	49,248	1,831	182,000	137,000	24
[68]	Zynq ZCU102	100	20,214	390	105,060	100,508	1.2
[69]	Arria 10	202	34,060	1,515	205,910	n.a.	26
[70]	Alveo U200	300	43,416	882	453,826	685,525	n.a.
[71]	Zynq 7035	n.a.	9,252	689	62,599	192,112	n.a.

A semantic segmentation network usually has deconvolutional layers for upsampling. This operation is usually done in CPUs and GPUs by using transposed convolutions adding zeros to the input image to implement strides and padding. On FPGAs, dynamically modifying the input image is not desirable, as best performance is achieved when taking profit of streaming memory access.

Liu [66] propose an efficient FPGA implementation of deconvolutional layers on FPGAs by computing all the values

of the final output depending on an input pixel, and accumulating the results on an output temporal buffer. This strategy avoids the use of the undesired zero padding, achieving a better energy efficiency with respect to GPUs. The number of operations per second per DSP (GOPS/DSP) of this design is very low compared to the others found in tables 14 and 15.

In [67] and [266], deconvolution is replaced by bi-linear interpolation. The authors use a tiled approach, so they divide the input image into smaller images and perform the segmentation on them. The NN is a MobileNet inspired by PSPNet NN implemented in a Xilinx ZCU102. The design achieves an accuracy (mIoU) of 64.2% running at 139 FPS for 1024×512 input images while consuming 24 W of power. The authors compress the model to fit in the FPGA's on-chip memory and, therefore, avoid external memory accesses, which induce high latency and energy consumption. They selectively prune the layers that have high number of operations and weights. Besides, pruning is filter-wise (not layer-wise) to make the number of non-zero weights equal between filters. This improve the speed of the convolutional pipeline since the load is balanced among filters. To store the sparse matrix in block RAMs, the authors need an overhead of additional memory for row, column, and channel information of the non-zero weights in the matrix. This overhead is small compared to the memory for non-zero weights since the sparse ratio is high. Another popular method for CNN compression is quantization, which is also done in this design. The authors use 8-bit integer for the whole inference pipeline.

The same authors map an AlexNet-based FCN to the same FPGA platform [68]. FCN differs from CNN in the last layer, where the fully connected layer is replaced by a convolutional layer. Some techniques used in [67] and [266] are also applied, including filter-wise pruning, bi-linear interpolation, coordinate memory overhead. In this case, the evaluation is on Camvid dataset. The FPGA-based implementation obtains 45% mIoU, which is nearly equal to its corresponding dense model.

Huang [69] maps Segnet onto FPGA using OpenCL-based HLS. However, the throughput of the system is only 7 FPS. The design achieves 44.7% mIoU on PASCAL VOC 2012 dataset. The authors explore the design space by varying the number of input features and kernels to be paralellized.

Miyama [70] implements 3-bit Unet-based network and achieves 67.8% mIoU on Camvid dataset. The inference pipeline only accesses internal memory. The design achieves a high number of operations per cyle by implementing the 3-bit addition and multiplication in LUTs instead of DSPs. The authors also adopt bilinear interpolations and convolutions to replace transposed convolutions. As a result, the NN reaches a frame rate of 123 FPS.

In summary, most FPGA NN implementations meet the real-time execution requirement but have an accuracy below the 70 % mIoU. As depicted in figure 24, there is still a significant gap between the accuracy provided by implementations on other platforms but, as in the previous case, they generally provide a better energy efficiency.

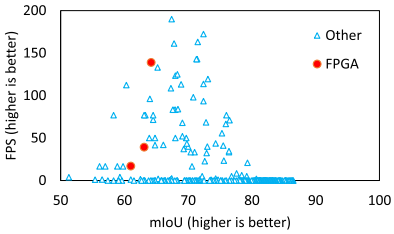


FIGURE 24. Trade-off between FPS and mIoU in semantic segmentation on Cityscapes dataset.

E. LANE DETECTION

The LD implementations on FPGA tend to follow a simple pipeline based on edge detection, ROI segmentation, binarization, HT, and a final verification phase to detect lanes. In the simple case, LD does not use temporal information but it can be used to improve accuracy as lane marks can be occluded or non-visible during some frames. This is usually done by implementing lane tracking with Kalman filters (KF) [37], [39].

Some FPGA implementations reduce the computational burden by ignoring some regions of the scene where the road lanes are highly unlikely to appear, such as upper part of the image above the horizon. In [37], they try to reduce the computational burden of subsequent steps by only considering pixels with certain gradient orientations. However this comes at the cost of computing the gradient.

Some of these algorithmically simple designs (such as [36], [37], [39], and [40]) achieve real-time performance at different image resolutions. Although these works describe a small loss of accuracy compared to the state of the art on other computing platforms, the reality is that they generally detect a limited number of lanes (usually two) and if they contrast the accuracy with a known dataset, they do not do this using the latest more challenging datasets. Instead, they tend to use the simpler Caltech dataset. As can be seen in table 16, even in that case, they do not provide the F_1 metric, and use accuracy instead.

Most implementations adopt an streaming architecture using line buffers and a Sobel 2D convolution kernel to generate an edge image.

The voting space for HT requires to loop on θ to get the corresponding values for ρ, as described by (21). We see that all the implementations in table 16 use HT or its improved version. Most of the implementations restrict the θ range to narrow down the searching space and, therefore, improve the detection speed. This restriction is reasonable because the purpose is to detect the lanes in front of the car, not an arbitrary line. Furthermore, the θ range is divided into smaller intervals, and different parallel units will process on different intervals. Many implementations try to implement

Algorithm 1 Hough Transform for Line Detection

Input: B, a binary image with detected edges. n_{lines}, the number of most voted lines to detect.

Output: R, a list of the parameters (ρ, θ) from the n_{lines} most voted lines.

1: $V \leftarrow 0$
2: **for all** y **do**
3: **for all** x **do** ▷ for all B image pixels
4: **for** $\theta \leftarrow 0$ **to** θ **step** θ_{inc} **do**
5: $\rho \leftarrow x \cdot \cos\theta + y \cdot \sin\theta$
6: $V(\hat{\rho}, \hat{\theta}) \leftarrow V(\hat{\rho}, \hat{\theta}) + 1$
7: **end for**
8: **end for**
9: **end for**
10: $R \leftarrow \text{argmax}_{n_{lines}}(V)$

a parallel voting strategy by unrolling the voting loop (line 4 in algorithm 1). Loop unrolling would be ineffective if the voting space is stored in a single shared memory because memory accesses would collide. To allow effective parallelization multiple separate memory blocks must be used to allow independent access to the voting memory. Then, loop unrolling is limited by the number of simultaneous memory blocks used to store the voting space (as depicted in Fig. 25).

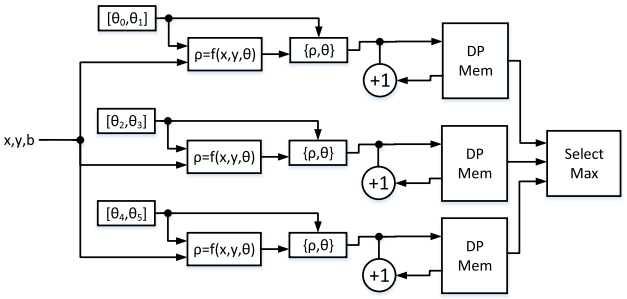


FIGURE 25. Voting strategy on FPGAs.

Look-up tables are also used to avoid the complexity of computing sin and cos functions and reduce the latency of the calculation to one clock cycle. In [39], the θ range is [10°,170°], and it is divided into 8 intervals of 20°. Eight processing pipelines are instantiated to calculate the ρ values, vote, and find the peak values as the lane candidates. The limitation of this design is that only one candidate is found for each interval. In [36], the authors instead divide the θ into two ranges [1°, 65°] and [-65°, -1°], corresponding to left and right lanes. Horizontal lines in front of the car (ex. $\theta = 90°$) are not considered. The range is divided into 13 intervals, 5° each. The division of the θ range into small intervals probably is the reason why this implementation achieves the highest throughput in Table 16.

Working with a predefined discrete number of θ values allows to store the $\cos\theta$ and $\sin\theta$ in lookup tables to reduce the computation load, this is a common technique in several works [36], [38], [39].

On-chip dual-port RAMs can be used to implement the voting memory and avoid using external memory (as done in [39]). The voting process includes reading, incrementing, and writing back to the memory. These operations happen in a single clock cycle, reading at the rising edge and writing at the falling edge.

In [39], the authors also improve the algorithm by moving the coordinate origin to the vanishing point where lanes intersect. Thanks to that, only edge points having $\rho = 0$ are stored in the voting memory. This algorithmic change relies on the precision of the estimated vanishing point. In this case, they use CORDIC IPs to ensure a good estimation accuracy. This requires some extra resources but it reduces the memory usage for voting.

Malmir [40] improves the accuracy of the system by adding stripe detection in the HT step. Lane marks are stripes, which can be identified by two parallel lines running close. The idea is that lines that are not stripes can be rejected. This work achieves the best performance in term of throughput among the implementations in table 16 with 60 fps on 1280 × 720 images.

TABLE 16. Performance lane detection FPGA.

Ref.	Author/Year	Method	FPS	Dataset	F-1
[39]	An 2013	SF+HT+KF	40	n.a.	n.a.
[36]	Zhao 2014	Otsu+HT	160	n.a.	n.a.
[37]	Xiao 2016	Sobel+HT+KF	24	Caltech	n.a.
[38]	Guan 2017	HT	18	n.a.	n.a.
[40]	Malmir 2019	Sobel+HT+SD	60	Caltech	n.a.
[41]	Zhan 2019	Color Segmentation + binarization	104	n.a.	n.a.

TABLE 17. Synthesis results lane detection FPGA.

Ref.	FPGA	Freq. (MHz)	RAM (kbits)	DSPs	LUTs	FFs	Power (W)
[39]	Spartan-3	80	1,656	21	4,632	4,966	n.a.
[36]	Kintex-7	74	2,160	31	10,081	8,405	n.a.
[37]	Cyclone III	n.a.	n.a.	n.a.	n.a	n.a.	n.a.
[38]	Stratix IV	200	1,566	16	2,656	1,459	n.a.
[40]	Kintex-7	160	8,478	170	91,050	66,186	n.a.
[41]	ZYNQ7035	100	n.a.	n.a.	n.a.	n.a.	n.a.

Most FPGA-based LD implementations are based on HT. As described in [267], this is an important drawback as HT limits the maximum possible reachable accuracy. Moreover, as seen in table 26, existing works do not pay enough attention at thoroughly contrasting accuracy using demanding recent datasets. To the best of our knowledge, there is no FPGA work presenting a NN-based LD that validates its results against a challenging public dataset.

As depicted in figure 26, solutions implemented in other computing platforms (CPU+GPU) already provide the required accuracy at real-time. FPGAs tend to be good at

energy efficiency and latency [268] but existing works do no analyze the power consumption aspect enough.

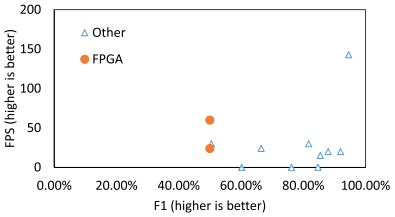


FIGURE 26. Lane detection trade-off on FPS vs Accuracy.

F. TRAFFIC SIGN RECOGNITION

Similar to LD algorithms, TSD and TSR FPGA implementations tend to use the following techniques to increase performance and save resources: line buffers, fixed-point representation, pipelining, data parallel processing, and avoidance of external memory access.

Recognizing all traffic signs from a large dataset using classical machine-learning methods is still very challenging. The hardware implementations in [44], [51], and [52] can only recognize a small amount of sign types and they are not confronted against a large real-world traffic-sign dataset. Some authors move to a software-based approach [50], [53] on FPGA to increase the accuracy at the cost of decreasing the the processing speed.

CNN-based implementations tend to achieve better performance. For instance, the model in [54] has a recognition rate of 96.5% at 36 FPS on GTSRB dataset. This accuracy is competitive with the state-of-the-art model [269] which obtains 99.61% recognition rate.

Correctly identifying color is very important for traffic sign recognition, as it has a semantic meaning. The clustering of similar colors is simpler when using color spaces other than RGB, such as HSV, HSI, or YUV. In Table 18, [49] only uses FPGA programmable logic to accelerate the HSV-based color segmentation. The authors choose to implement HSV over HSI because it requires less divisions. The final implementation can only work at 0.2 FPS since the detection and classification are implemented in software.

Since the signs appear at different distances and poses in the scene, TSD and TSR algorithms should be invariant to affine transformations. Rotation and scale invariant features such as SURF are used in [50] and [51], while ORB features are used in [53]. In the case of SURF, the algorithm can make use of the integral image $I_{\sum}$ described by equation (19). While the integral images provides an efficient way to compute box filters, it present an additional advantage for FPGA implementations since it can be computed using a pixel-streaming architecture such as that shown in 28. A similar design is used in [270].

To reduce the computational complexity, Zhao [51] performs additional optimizations. First they convert floating-point into fixed-point data with acceptable loss in accuracy. Secondly, the SURF algorithm is adapted to

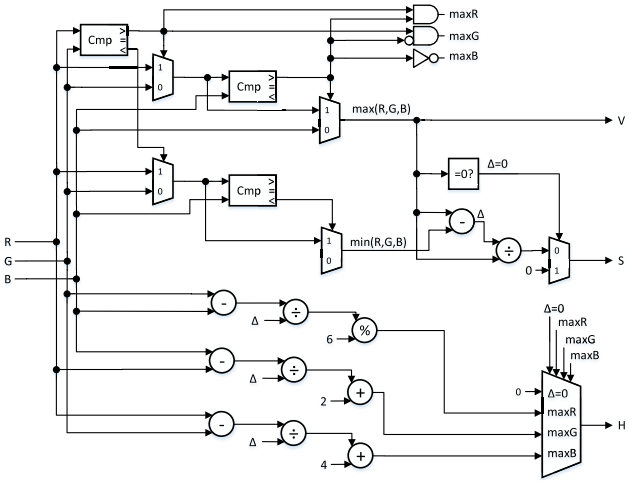


FIGURE 27. Logic diagram of an RGB to HSV color space conversion. Registers are normally used in a pipelined design. Here are not included to simplify the diagram.

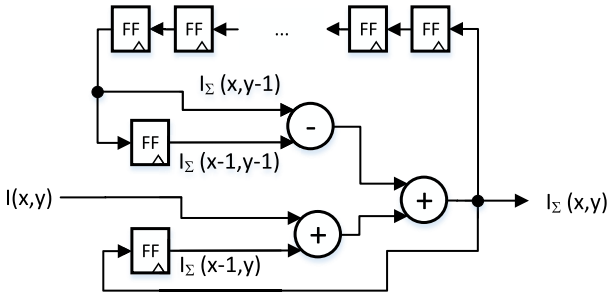


FIGURE 28. Integral Image Circuit.

the characteristics of the sign recognition application. The authors restrict the number of scales to two because most signs are quite small in the image. The dominant direction of each interest point is set upright to simplify the computation. The final implementation achieves up to 60 FPS throughput with 800 × 600 input images. SURF-based TSR is also implemented in software in [50]. But the throughput of this design is only 1 FPS for 320 × 240 input frames. This paper reports the accuracy of 87.19% based on a private database.

The work in [52] uses a different approach to implement TSR. It uses grayscale input images instead of color to reduce the overall system cost. The critical path in the pipeline are the cascade classifiers based on Haar-like features. The authors rearrange the numerical operations in the feature extraction module to reduce the amount of memory accesses and thus efficiently use the limited memory bandwidth. The approach reduces the number of operations 1.77× and the number of memory access 3.38×. Besides, the authors propose other techniques to improve the performance of the system such as shared image storage, adaptive workload distribution, and fast image block integration. The design achieves the highest throughput in table with 126 FPS. This algorithm proves to be more suitable for FPGAs than CPUs and GPUs. The authors show that the accelerator speedups 17.25× and 5.61× over

the CPU and GPU implementations, respectively. Moreover, the energy consumption is reduced 252.44× and 42.68× over the CPU and GPU alternatives. We assume that this type of cascade algorithm is not GPU-friendly.

Farhat et. al. [53] implement the TSR on the hard-processor of a FPGA SoC and obtain 11 FPS throughput. The detection rate is 94% at a recall of 97% on the GTSDB dataset. The recognition accuracy is reported separately of 95% at a recall of 95%. This is the only paper in table 18 implement sign detection on GTSDB dataset. It is more practical because it includes the localization step of the signs in the street scenes before recognition step. The combination of the two phase is 92% which is the product of the precision of the two phases.

In [54], only the recognition step is done on FPGA. The authors propose a resource-efficient CNN for FPGAs. They employ binary weights and 8-bit integer features to port the network onto FPGA. The implementation achieves a recognition accuracy of 96.5% on modified GTSRB dataset. The processing speed is 36 FPS which is applicable for real-time applications.

TABLE 18. Performance traffic sign recognition FPGA.

Ref.	Author/Year	Method	FPS	Dataset	Accuracy
[49]	Russell 2013	CS+Edge+TM	0.2	n.a.	NA
[51]	Zhao 2013	SURF+TM	60	n.a.	NA
[50]	Han 2015	CS+SURF+KNN	1	Private	87.19
[52]	Shi 2017	Cascade	126	n.a.	NA
[53]	Farhat 2018	CS+ORB+TM	11	GTSDB	95.5
[54]	Lechner 2019	CNN	36	GTSRB	96.53

CS = Color segmentation. TM = Template Matching. KNN = k-nearest neighbours. CNN = Convolutional Neural Network. ORB = ORBF.

TABLE 19. Synthesis results traffic sign recognition on FPGA.

Ref.	FPGA	Freq. (MHz)	RAM (kbits)	DSPs	LUTs	FFs	Power (W)
[49]	Zynq-7020	143	n.a.	n.a.	n.a.	n.a.	n.a.
[51]	Kintex-7	126	7,992	244	179,559	2,186	3.4
[50]	Zynq-7020	n.a.	956	n.a.	15,960	5,320	5.2
[52]	Zynq-7045	100	272.5	n.a.	108,581	4,372	n.a.
[53]	Zynq-7020	n.a.	n.a.	n.a.	n.a.	NA	n.a.
[54]	Zynq-7020	100	4,608	0	30,434	24,686	n.a.

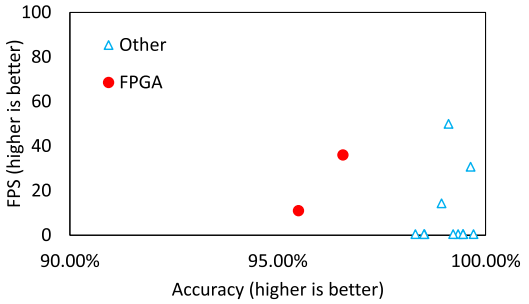


FIGURE 29. Traffic sign recognition results on GTSRB dataset.

As it happens with LD, solutions implemented in other computing platforms (CPU+GPU) already provide the required accuracy at real-time (see figure 29) and power consumption is not analyzed enough.

G. TRAFFIC LIGHTS

The goal of TLD and TLR is not only to detect the presence of traffic lights, but also to recognize their state. Contrary to traffic signs, traffic lights have a dynamic behavior. There are very few works of TLD and TLR in FPGAs. Moreover, the few existing ones do not use well-known datasets, such as LISA. We limit our analysis to only two implementations.

TABLE 20. Performance traffic lights recognition FPGA.

Ref.	Author/Year	Method	FPS	Dataset	Accuracy
[271]	Wu 2019	Haar+AdaBoost	30	n.a.	n.a.
[56]	Fukuchi 2020	CNN	2.5	n.a.	98.3 %

TABLE 21. Synthesis results traffic lights recognition on FPGA.

Ref.	FPGA	Freq. (MHz)	RAM (kbits)	DSPs	LUTs	FFs	Power (W)
[271]	XC7A100	200	288	n.a.	41523	39682	n.a.
[56]	Zynq Ultrascale+ MPSOC	n.a.	900	41	5866	4859	

Wu [271] proposes a TLR system based on an AdaBoost classifier. He uses YCrCb color space conversion with fixed point approximation. The YCrCb conversion is similar to the YUV conversion described by equation (11). Then, he computes an integral image for each channel that are used for the following Haar-feature weak classifiers. The threshold values used by the classifiers are determined in a training process.

In [56], Fukuchi proposes a CNN model. However, the FPGA is used to implement a convolution unit, not the whole NN, which is the reason why the system has a low resource consumption and, worse, a low performance.

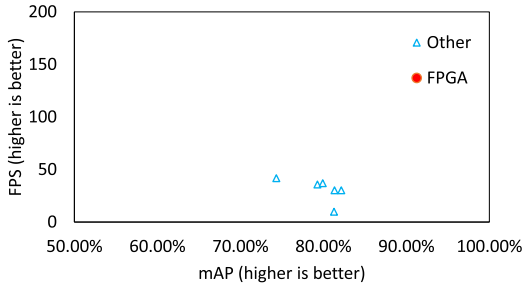


FIGURE 30. Traffic lights recognition results trade-off between accuracy and performance on LISA dataset.

Solutions in alternative platforms reach high levels of accuracy and are close to real-time operation as shown in 30. On the other hand, existing FPGA implementations do not report enough data for a fair comparison.

H. OBSTACLE DETECTION

Obstacles on the road can be of various forms and visual appearances, so OD systems must expect the unexpected. The detection can be based on depth maps, semantic information, or object detection from cameras, which is probably the most popular method. Modern DL-based object detectors are capable of detecting a large number of object classes, an can be used to detect most expected obstacles.

There have been many FPGA-based CNN implementations devoted to object detection published in recent years, surveyed in [7], [272], and [273]. However, the number of implementations explicitly targeting at self-driving cars is limited. One exception is [274], where authors implement an optimized CeNN in FPGA for obstacle detection.

Zeng [273] exhaustively surveys the recent object detection NNs and the design of FPGA-based accelerators for them. CNNs on FPGAs mostly use tiling and loop unrolling techniques to achieve high processing speed, and can also use Winograd convolutions [275] to reduce workload.

Line buffers can also be used to reduce the memory bandwidth required to access input values. With small enough kernels the kernel weight values can be stored in registers or in on-chip memories to reduce the memory bandwidth. To save even more memory space weights and features are quantized and pruned. When the on-chip memory is not enough, loop tiling technique divide the data such as input features, weights, and output feature into tiles.

Tables 22 and 23 compares some CNN-based object detection systems on FPGAs. Most implementations evaluate results with the PASCAL VOC dataset, with the exception of [72] which uses KITTI, and [76] using COCO2014.

TABLE 22. Performance CNN-based object detection implementations on FPGAs.

Ref.	Author/Year	Method	FPS	Dataset	mAP
[72]	Nakahara 2017	VGG11 BCNN	34.9	KITTI	n.a.%
[73]	Nakahara 2018	YOLOv2	40.81	PASCAL VOC	67.6%
[74]	Ma 2018	SSD	16	PASCAL VOC	76.94%
[74]	Ma 2018	SSD	34.5	PASCAL VOC	76.94%
[75]	Nguyen 2019	YOLO2	109.3	PASCAL VOC	64.16%
[76]	Yu 2020	YOLOv3-tiny	1.9	COCO2014	30.9%

TABLE 23. Synthesis results CNN-based object detection implementations on FPGAs.

Ref.	FPGA	Freq. (MHz)	RAM (kbits)	DSPs	LUTs	FFs	Power (W)
[72]	ZCU102	200	4,176	0	179 k	190 k	2.5 [a]
[73]	ZCU102	300	30,708	377	135 k	370K	4.5
[74]	Arria 10	240	51,620	1,518	175 k (ALM)	n.a.	40
[74]	Stratix 10	300	76,880	4,363	532 k(ALM)	n.a.	100
[75]	Virtex-7	200	20,592	272	115 k	115 k	18.29
[76]	Zedboard	100	3,330	160	25.9 k	46.7 k	3.36

[a] dynamic power

In [72], a Binarized CNN [276] is implemented in an FPGA for the task of obstacle detection. BCNN outstands for its lightness as only $+1$ or -1 weights are allowed. The whole network is composed of a sliding window and ROI generator followed by a fully pipelined VGG-11 based B-CNN and non-maximum suppression.

In [73], the same authors as in [72] implement YOLOv2 on Xilinx ZCU102 board [73] using PASCAL VOC dataset. The design's throughput is 40.81 FPS; it outperforms the same design on CPU and GPU. The authors binarize both weights and features of a lightweight YOLOv2 model. The features are all stored in block RAMs to avoid external memory access. The weights are loaded in the cache memory so that the latency of external memory access is hidden. Feature maps are processed in parallel to take advantage of the weights in cache memory. The authors proposed to employ SVMs for localization and classification. There are up to 50 SVMs instantiated in the design, and they use floating-point weights loaded in the cache memory. The power consumption is only 4.5 W, which is the lowest in the Table 23. The paper reports 67.6% mAP, even though the model is binarized. This good accuracy might come from the SVM models with floating-point weights.

Nguyen [75] also implements YOLOv2 and obtains only 64.16% mAP. The design only binarizes the weights. The features are represented by three to six bit-width fixed-point numbers. The entire model is stored in Block RAMs and intermediate data are fully reused across the layers. Therefore, the design minimizes external memory accesses. Parallel computation is on both feature map and weight convolution. The pipeline supports batch processing to improve the design's throughput. This design provides the best throughput, resource efficiency (number of operations per second per kLUT), and energy efficiency (number of operations per Joule).

Instead of YOLOv2, authors in [74] implement SSD on two different FPGAs, Arria 10 and Stratix 10. Normal convolution is executed instead of original dilated convolution which consumes a large number of resources. Besides, the normalization operation is simplified by using an estimated constant scale. Finally, the authors use a single stride step to reuse the hardware although the original SSD has different stride steps across the layers. The authors propose to process one tile at a time. The reason is that the weights and the features for one tile can be fully loaded into memory buffers from external memory and, therefore, the pipeline can overlap the memory access delay with the computational latency. The design uses fixed-point numbers to represent data and the quantization is different between layers. The paper reports a very high accuracy of 76.94 % mAP. The implementation on Stratix 10 provides a higher performance while it provides a better energy efficiency on Arria 10.

YOLOv3-tiny, a light-weight version of the state-of-the-art object detection model YOLOv3, is mapped on a low-end FPGA device in [76]. The tiny model has fewer layers and it is more suitable for resource-constrained platforms. The authors also compress the model using 16-bit fixed point quantization to further decrease computational load

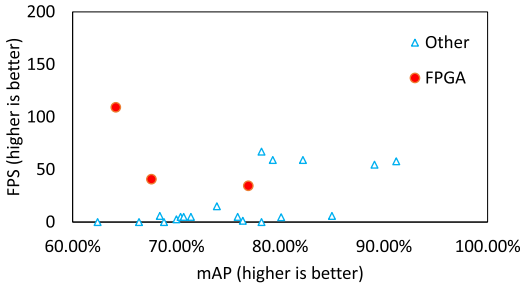


FIGURE 31. Results trade-off between accuracy and performance on PASCAL VOC.

and detection latency. However, the implementation still has to use off-chip memory due to the limited on-chip memory. Therefore, the speed of the design is only 2 FPS and 10 GOPS. The accuracy is also low with 30.9% mAP and this number is measured only on the validation set of the COCO2014 dataset. Anyway, the implementation gives a trend to map this state-of-the-art model onto FPGA platforms.

Figure 31 depicts the performance and accuracy results of object detection implementations based on FPGA and other computing platforms on the PASCAL VOC dataset. The figure illustrates that there are implementations that offer higher performance and higher accuracy than FPGA-based ones.

I. PEDESTRIAN DETECTION

Accuracy in PD is crucial due to the safety concerns. One of the best PD implementations in the literature is currently [32], achieving a miss rate of 1.7 % on Caltech dataset using R-CNNs. The most accurate FPGA implementation based on a NN [63] is providing 11.8% on the same dataset.

Many previous attempts to provide FPGA-based PD are based on classic ML approaches, HOG+SVM [57], [58], [59], [65], HOG+AdaBoost [60], or Haar+SVM [64]. Most works justify sacrificing accuracy to achieve higher throughput. The generation of HOG features can be implemented effectively in a pixel streaming architecture by using line buffers [57]. Once feature vectors are obtained, a SVM classifier can also be implemented effectively by unrolling the loop of the linear equation (23). The normalization of HOG features, the multiscale processing and non-maximum suppression (NMS) phases add some complexity that prevents to use a fully streaming design. HOG requires to compute the orientation of a gradient, this trigonometric function is typically implemented by a pipelined CORDIC block, or by a polynomial approximation [57].

In the SVM classification module, the pipeline rearranges the calculation so that every block histogram passes the pipeline once and the memory does not need to store it. This technique helps to save the memory resource. Finally, to achieve real-time speed for 18 scales, the authors use time-multiplex approach, which processes 6 scales for every input image and obtains the detection result of 18 scales after three input images. This approach also saves the computing resource because there are only 6 pipelines instantiated instead of 18. The implementation achieves the highest throughput in the Table with 132 million pixels per second.

To implement multiscale, designs either opt for a replication of units for different scales, to time-multiplex existing units, or for a combination of both. In any case, image rescaler units are needed to generate the multiple input images. Fixed-point arithmetic is commonly used [64]. In [58], the authors study the effect of reduced bit-width on the accuracy and performance of the algorithm on FPGA. The paper proves that 13-bit fixed-point preserves the same detection accuracy as the original floating-point.

Rettkowski [60] classifies HOG features using AdaBoost classifier instead of SVM. The authors propose to add a binarization step in the normalization phase to reduce resource with an apparent good accuracy, however results are not evaluated on public datasets.

One of the few FPGA-based PD systems using a NN is presented in [63]. Pipelining and parallelism are used, but due to the limited number of DSPs and on-chip memory, the layers are read in chunks from external memory, reducing the throughput of the design, which is the lowest in table 24, but still on the real-time range. The authors quantize the model into 16-bit fixed-point with less than 0.01% loss in accuracy. The implementation achieves a miss rate of 11.8% with a real-time throughput of 24 FPS.

Recently, Ghaffari et. al. [65] propose several algorithmic changes to the original HOG algorithm and implement it on FPGA. First, they replace divisions and multiplications in bin assignment by logarithmic operations. Only subtractions and logarithms are used in this method, and, therefore, the DSPs used for multipliers are reduced. The implementation uses look-up tables to calculate the logarithm of the gradients. Thanks to the optimization techniques, the resource usage is one of the lowest of table 25.

TABLE 24. Performance pedestrian detection FPGA.

Ref.	Author/Year	Method	FPS	Dataset	Accuracy
[57]	Hahnle 2013	HOG + SVM	64	INRIA	16 % (10^{-3})
[58]	Ma 2015	HOG + SVM	68.18	Caltech	44% (0.46)
[59]	Khan 2015	HOG + SVM	42.7	INRIA	13% (10^{-4})
[60]	Rettkowski 2015	HOG + AdaBoost	40	other	90.2% (0.04)
[63]	Moussawi 2018	FDNN + FDNN	24	Caltech	n.a.
[64]	Luo 2019	Haar + SVM	70	INRIA	96.93 %
[65]	Ghaffari 2020	HOG + SVM	115	INRIA	30% (10^{-2})

In [277], a complete dataflow of a recurrent convolutional neural network (R-CNN) in FPGA is proposed for car and pedestrian detection, however accuracy results are incomplete and the system is tested with simulation.

In summary, FPGA-based PD implementations can not be fairly compared with those implemented on other platforms because of the lack of detail on their accuracy analysis. We believe that there is a research gap to propose FPGA implementations that could be competitive with respect to

TABLE 25. Synthesis results pedestrian detection FPGA.

Ref.	FPGA	Freq. (MHz)	RAM (kbits)	DSPs	LUTs	FFs	Power (W)
[57]	Virtex-5	270	1,188	49	5,188	5,176	n.a.
[58]	Virtex-6	150	13,738	190	184,953	208,665	37
[59]	Virtex-6	265	2,851	112	47,424	75,878	19
[60]	Zynq-7000	82.2	0	4	21,297	5,942	n.a.
[63]	Arria 10	n.a	n.a.	n.a.	n.a.	n.a.	n.a.
[64]	Stratix IV	170	197,642	0	6,286	4,566	0.65
[65]	Kintex Ultrascale	150	756	36	7,804	n.a.	n.a.

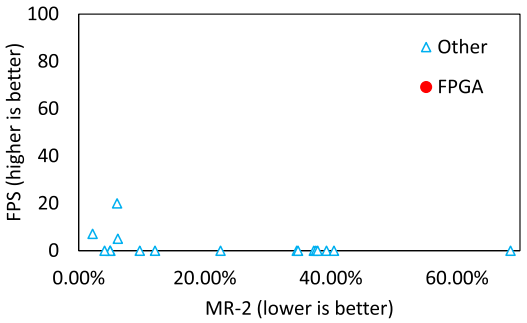


FIGURE 32. FPGA results pedestrian detection FPGA vs Others on Caltech dataset.

those existing on other platforms as figure 32 shows that they hardly reach the real-time requirements with good accuracy.

VI. DISCUSSION

Based on the analyzed information, we can identify some trends and challenges for the use of FPGA-based accelerators in self-driving cars.

A. DEEP-LEARNING TREND

For all functions analyzed, DL models systematically achieve the best accuracies. DL architectures are expected to continue evolving to increase their accuracy and face more complex challenges [278]. Neural architecture search (NAS) can play a role in achieving higher accuracy values at the cost of additional computing power during training. This might reduce the number of players that afford the large investment. It is also expected that NNs in the automotive domain can benefit from working, not only on the spatial, but on the temporal domain. Networks dealing with video streams could benefit from computation performed on previous frames and even reduce their computational demand by doing so. Self-driving cars will include a multitude of sensors. While the research on many of the functions are addressed by only processing data from a single type of sensor, combining data from multiple sensors would contribute to better accuracies. Thus, we expect more future research on multi-modal algorithms combining info from multiple sensors. There is an increasing number of works working with 3D data, and we expect that the future trend keeps increasing the data volume and the network complexity.

To foster these developments also more multimodal datasets will be required, and new metrics will have to consider more abstract evaluation criteria than those currently used. As an idea, we might see benchmarks that report "number of collisions per year" after running the system in a controlled environment. The cost of producing such datasets with real data can become extremely high, so we expect that they are reduced significantly by using realistic simulation frameworks such as CARLA [279].

B. THE MASS DEPLOYMENT OF SELF-DRIVING CARS

After an initial enthusiasm and many optimistic marketing claims, many believe that the mass-deployment of self-driving cars will be rather slow, estimating that 50% of the fleet will be autonomous by 2060 [280]. Besides the needed adaptation of regulation and the insurance business, there will still be many challenges to solve in the coming years [281]. In controlled conditions current technology is performing well. The problem is that there is an endless number of edge conditions that escape from the considered scenarios during algorithm training and testing. Those edge conditions have already resulted in some Level 2 cars being involved in road accidents. Some of them are caused by humans intentionally breaking the law, wild animals, or moving objects. To avoid these situations, the future system should be able to track better moving objects and predict animal and human behaviour. Other complex situations are the result of hard illumination conditions, presence of unexpected objects, or errors from the sensors.

At the algorithmic level, although the CV functions apparently provide a good performance, they are highly dependent on the training dataset. For instance, in [282] the authors describe the problems of PD NN models to generalize when testing on different datasets than those used during training. This will be a serious aspect for worldwide deployment of the technology, as this reliance on the training dataset is also known to create a bias that could drastically reduce the accuracy of the systems when dealing with underrepresented classes.

C. ESTIMATING THE COMPUTATIONAL DEMAND OF SELF-DRIVING CARS

In this section we will try to estimate the required computational power that will be required to execute the future autonomous car functions.

Table 26 illustrates the computational complexity of some of the most accurate NNs for the analyzed autonomous car functions by listing their parameter count and required FLOPs (floating-point operations, not to be confused with Floating-point operations per second). Aggregating the number of FLOPs and considering a real-time execution requirement at 30 FPS, we estimate that a computing platform to execute all functions would require more than 100 TFLOPS (floating-point operations per second) of computing power to execute them all simultaneously. Google TPUv4i already provides a peak performance of 138 TFLOPS [283], but

the company researchers already assume that memory and computing demand for NNs is increasing at a 1.5× factor per year. If we consider that the complexity of the algorithms will keep increasing and the FPS should be increased to achieve higher levels of safety we might be reaching the PFLOPS range.

TABLE 26. Parameter count of different networks.

ADAS function	Network	Parameter Count	FLOPs
SM	LEA-Stereo [96]	1.81 M	782.2 G
SS	EfficientNet [130]	485 M	229 G
LD	RSCM [284]	3.21 M	23.7 G
TSR	Transformers [167]	60 M	1.8 T
TLD	DETLR [285]	7.7 M	81.4 G [a]
TLR	DETLR [285]	55 k	5.2 M
OD	ResNET 50 [286]	53.3 M	322.01 G
PD	YOLOv5-AC [219]	3.37 M	22.3 G

[a] estimated

What are the architectural features of the systems that can provide this level of performance?

1) ARITHMETIC UNITS

Many operations of recent NN models can be implemented as matrix multiplications. These operation can benefit from SIMD or vector processing units, where the same operation is executed over multiple data organized in arrays of elements of a primitive data type. While the bandwidth of the arithmetic unit is fixed, recent architectures allow to select the width of the array elements between several discrete values. The throughput of such units is $T = B/(2O)$, where B is the input bandwidth (in bits per second), and O the width of the array element (in bits per operand). Thus, reducing O is a common technique to increase the performance of the system.

This strategy can be used for either floating point or integer arithmetic. Integer arithmetic is simpler to implement and consumes less hardware resources, but floating point is usually mandatory during training to make gradient descent work. Still, there is some flexibility to select the bit width for mantisa and exponent without significantly affecting the accuracy. Instead of the IEEE 754 standard, Google introduced a 16bits floating point format (bfp16) that reduces the number of bits of the mantissa. Besides augmenting the throughput, this also reduces the required hardware resources since the number of logic gates to implement a multipliers depends quadratically on the number of bits.

During inference, fixed point and more aggressive bit-widths can be used. There are a number of works using int8. Again, this is a source of memory bandwidth reduction and increased performance throughput, and allows to use the saved resources to replicate more arithmetic units.

Another aggressive resource reduction technique impacting accuracy is the use of approximate computing units to perform the required computation [287].

2) MEMORY

The fast execution of large inference NNs is heavily dependent on having a high memory bandwidth to fetch both layer inputs and weights. GPUs have a high memory bandwidth thanks to a multiported access to external high-speed memories. In addition, the have a cache hierarchy that minimize the accesses to external memory. On comparison, FGPAs use to have a lower bandwidth to external memory.

TPUs architectures optimize the memory hierarchy by providing multiported memories (e.g. HBM), complex DMA engines, on-chip memory, and interconnection to to maximize the throughput of the systolic array. In Google's TPUv4i [283] the on-chip memory is rather big 144 MB and provides a bandwidth of 2 TB/s to the systolic array.

Future memory systems for NN accelerators are surveyed by Asad [288]. Even with on-chip memory optimization techniques, the high amount of memory needed by NNs still requires the use of multiple external memory chips. Higher memory density is required. 3D stacking helps to reduce the physical length of the wires to access the memory contributing to a lower memory consumption. Serial and differential communication interfaces also help in reducing the pinout requirements of multiported memories. A drawback of 3D stacking is the increased processing cost, and the higher energy density, that could become a limiting factor. An strategy to reduce external memory requirements is to increase the amount of on-chip memory. A recurrent idea is to try to store network parameters internally, since they are computed during training but not updated during inference. Thus, parameters could be stored permanently on-chip to reduce external accesses. This is specially interesting for NVRAM memories that have a higher density than DRAM and SRAM. Although higher than SRAM, the density of DRAM is limited by the need to create a big enough capacitor that is able to store the charge for a reasonable period of time. Most NVRAM technologies are free from that limitation. ReRAM is one recent NVRAM promising technologies, with a higher density. With a significant higher density, the ultimate aim would be to store all weigths on-chip.

Another concept that could have an impact in future architectures is the in-memory and near-memory computing [289], that promotes to tighly-couple computing cores and memory storage devices or even use memory to perform computing.

3) THE LATENCY PROBLEM

Given the possible high speeds that are expected in self-driving vehicles, low processing latency is fundamental to ensure a safe operation. Distance travelled by a moving object at a constant speed is $X = V \cdot t$. Imagine a situation where two cars have a frontal collision running at 120 km/h. Just one second before the impact, the cars would be 66 meters away from each other. Considering that reaction time of the mechanical parts is not immediate it is extremely important that dangerous situations can be identified and preventing actions are taken as fast as possible.

As we have analyzed, the control loops in self-driving cars involve several complex functions. Security relies in the end-to-end latency, which involve several functions. Actually, the final end-to-end latency will be given by the sum of latencies from the functions on the critical path.

Some naive end-to-end latency analysis measure each individual latency and aggregate them to get the critical path. However, as pointed out by Becker [31], the overhead of communication between the different functions and the effects of their simultaneous excution in the same computing platform have a significant impact that should not be overlooked. The author analyzes the Autoware software [290] latencies, not only in isolation, but end to end, concluding that current software and hardware platforms are not able to deliver the required performance.

Empirical testing of latency is important, especially for NNs architectures and and systems interfacing with external sensors. Data acquisition can already be a determinant issue [291]. For NN models, latency cannot be obtained as the inverse of throughput. Many times, NN reported throughput is the result of applying optimization strategies such as input-batching, and task pipelining using ping-pong buffers. Thus, the latency values are usually higher.

D. CURRENT FPGA OFFERINGS

FPGAs are one of the possible computing platforms that can have a place in self-driving cars. Other alternatives are CPUs, GPUs, DPUs. In this section we analyze what FPGAs are currently offering, and how are they positioned with respect to competing platforms.

1) THE ROLE OF FPGAs IN CURRENT SELF-DRIVING SOFTWARE STACKS

Self-driving cars hardware and software is a very dynamic arena in constant development. There are many players in the game: big software companies, auto-makers, chip manufacturers, and small companies. As self-driving is so complex at the functional level, there have been initiatives to provide a complete software stack to provide autonomous driving. Table 27 collects some of the more representative initiatives. Still, no one is providing level-3 yet.

Many of the stacks are Closed-Source, since they try to become an business asset to provide an advantage against competitors. However, there are also some Open-Source proposals. Apollo Auto [292] is an interesting open-source proposal from Baidu that sits on top of a drive-by-wire car platform and addresses all functional aspects of self-driving cars. It have been quickly evolving since 2017, currently being in v7. Autoware [290] is an open alliance to promote the development of self-driving technologies, it also provides a full software stack.

Another Open-Source solution is OpenPilot [293]. This is a completely different proposal that aims to control an standard car by a drive-by-wire approach using visual sensors. In the core of the system there is an end to end NN called SuperCombo that controls the actuactors directly from the sensor inputs. However, according to [294] OpenPilot has important functionality deficits that would not make it comparable to other modular approaches.

TABLE 27. Software stacks for autonomous driving.

Software	Computing Platform	Comments
Waymo Driver	CPU+GPU	Closed-Source
Cariad	CPU	Closed-Source
Autopilot [295]	CPU+GPU+DPU	Closed-Source
Apollo Auto [292]	CPU+GPU	Open-Source
XPilot [296]	CPU+GPU	Closed-Source
OpenPilot [293]	CPU+GPU	Open-Source
Aurora Driver	CPU+GPU	Closed-Source
NVIDIA Driverworks [297]	CPU+GPU	Development Framework
Autoware [290]	CPU+GPU	Open-Source

2) THE ACCURACY-PERFORMANCE TRADEOFF

While implementations of CV functions in CPUs and GPUs focus on obtaining a good accuracy-performance tradeoff, FPGA implementations often sacrifice accuracy over performance [7], [298], either by selecting simpler algorithms or applying aggressive optimizations that have a negative impact on it. Table 28 lists some relevant works from the literature that show good tradeoffs between accuracy and performance when implementing ADAS functions on FPGA and non-FPGA platforms.

TABLE 28. Accuracy/error and performance (in FPS) achieved by recent works for different functions.

ADAS function	FPGA			Other		
	work	acc./err.	perf.	work	acc./err.	perf.
SM	[46]	↓ 6.37 %	114	[299]	↓ 1.65 %	50
SS	[67]	↑ 64.2 %	139	[300]	↑ 79.3 %	20
LD	[40]	↑ 50 % [a]	60	[138]	↑ 94.6 %	142
TSR	[54]	↑ 96.53 %	36	[301]	↑ 99.89 %	37
TLR	no relevant result			[302]	↑ 82 %	30
OD	[74]	↑ 76.9	34	[303]	↑ 91.2 %	57
PD	[58]	↓ 68.50 %	68	[304]	↓ 6.10 %	20

[a] estimated

↑ and ↓ are used to indicate that either an accuracy metric (higher when better) or an error metric (lower when better) is used.

The gap between the best FPGA implementations and non-FPGA implementations is significant for most functions. It is especially big for pedestrian detection, which is one of the most critical functions. This is because many FPGA implementations are based on HOG+SVM, a method that is significantly less accurate than the best recent DL models.

Accuracy is fundamental for safety-critical functions and should never be sacrificed unless there is a strong reason that justifies it. One such reason could be that the alternative method does not achieve the real-time requirement. But on the light of the results from table 28 this is not the case. Many CPU/GPU implementations are not only more accurate, but also show a higher throughput.

Due to the limitation of memory and computing resources, FPGA-based accelerators normally customize algorithms to hardware-friendly algorithms. For similar reasons, there is a lack of fusion systems on FPGAs as they increase the complexity to a higher level.

3) MEMORY ASPECTS

With small enough NNs, FPGAs can store many parameters in internal resources (FFs or on-chip memories) or even implement constant arithmetic operations to reduce the demand of external memory bandwidth. However, self-driving cars require complex networks with a large number of parameters, and those aggressive strategies cannot be generally applied.

FPGAs also offer a lot of flexibility to create line buffers. They are very useful to reduce latency, increase throughput and energy efficiency in sliding-window processing applications [305]. However, the multiple window and kernel sizes of most NNs make it difficult to leverage these capabilities.

Another aspect that increase the complexity for FPGA implementations is the lack of cache memories. CPU and GPU programmers can address external memory without worrying much data locality because there is a memory hierarchy that increases it by the transparent use of several cache levels. On contrary, FPGA designers have the overhead of explicitly implementing strategies to increase data locality [306].

4) THE DEVELOPMENT BOTTLENECK

The number of research works targeting GPU platform has outnumbered the number of those targeting FPGAs in the last decade. This can be observed by looking at the research papers published from 2010 with the term CNN and either FPGA or GPU (see figure 33). Notice that in recent years there is an order of magnitude difference in favor of GPUs.

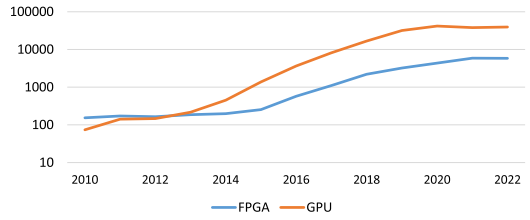


FIGURE 33. Number of yearly works in google scholar results for the queries "FPGA CNN" and "GPU CNN."

One of the reasons of this difference could be attributed to economic cost and availability, but we believe it is more related to the availability of development tools. Many programming frameworks have been proposed for GPUs (CUDA, HMPP, OpenCL, OpenACC, etc.). The software based compilation flow certainly eases the development and test iterative process. Moreover, GPU support has been added to most (if not all) DL frameworks. On the other hand, very few DL frameworks directly support FPGAs.

When it comes to low level implementation of NN primitives, some works focus on providing highly optimized HDL. However, HDL is slow and tedious. A more productive method is to use high level synthesis through OpenCL or existing frameworks NNs such as the one in [307]. HLS tools help developers get to the final result faster with less concern on hardware low-level details. However, HLS-based implementations still can only gain a sub-optimal throughput compared to RTL-based ones.

5) ENERGY EFFICIENCY

Some works of the literature have claimed an FPGA advantage against GPUs in terms of energy efficiency [268]. Table 29 shows that the energy efficiency of FPGA-based implementations is better than GPU-based ones from 2.26× up to 58× across different algorithms. We include the feature size of the foundry process as it is a determinant factor for energy efficiency [19].

TABLE 29. Reported energy efficiency factor between FPGAs and GPUs.

Work	year	Problem	FPGA node	GPU node	Factor
[44]	2018	SM	28 nm	28 nm	36×
[308]	2017	OD (CNN)	20 nm	28 nm	2.26×
[58]	2015	PD (HOG+SVM)	40 nm	28 nm	31×
[73]	2018	OD (CNN)	16 nm	16 nm	43×
[309]	2021	OD (CNN)	16 nm	16 nm	13×
[46]	2020	SM	28 nm	12 nm	58×

In a quick and superficial analysis, one could conclude that FPGAs are much superior in energy efficiency, but a deeper analysis reveals that these differences can not be generalized. In many cases [46], [309], part of this differences can be attributed to the use of fixed point small-width arithmetic (int8) in the FPGA instead of the single precision floating point arithmetic (fp32). In [46], part of the algorithm can benefit from the use of line buffers to do cost aggregation in 4 different directions simultaneously. Shi [308] looks like a more fair comparison, but in this case the foundry process is not the same, being the FPGA one more recent but still not providing a double-digit improvement with respect to the GPU.

From these works we do not know exactly know the number of operations performed. Thus, we can not estimated the OPS/W provided by the platforms. In any case, Wang [310] analyzes several FPGA NN accelerators being below 100 GOPS/W. From our previous estimations the ideal computing platform should be able to deliver 1 PFLOPS within a 200 W energy budget. This gives an objective of 5 TFLOPS/W, and actually the proposal from [310] is in this range.

In any case, energy efficiency is probably the best advantage of FPGAs with respect to CPUs and GPUs, and it is often overlooked in many research works. We believe that there is an important research gap in proposing novel implementations that address this issue.

E. THE CHALLENGES FOR FPGAs IN FUTURE SELF-DRIVING CARS

In this section we will analyze the challenges that could face FPGAs in the self-driving cars domain.

1) THE DEVELOPMENT EFFORT

Self-driving software stacks will require DL experts and software engineers with experience in multi-processing systems. DL models are easily implemented into CPU and GPU platforms as many of the DL frameworks automatically consider them as execution platforms. Therefore, there exist a gap between the ease of deployment between FPGA platforms and other platforms. We believe that a significant effort should be devoted to reduce this gap and make it easier for non-FPGA engineers to leverage the power of existing devices.

2) TECHNOLOGY ADVANTAGE

The simple structure, regularity, and configuration facilities of FPGA devices used to be a good test for new foundry nodes, as so, they were usually using the latest available nodes after the top market drivers (CPU manufacturers) and sometimes even before. However, market demand has increased the diversity of resources that now can be found in a FPGA, increasing their complexity.

As depicted in figure 34, FPGA adoption of new foundry nodes is lagging behind of those for CPUs and GPUs for recent nodes. It is specially relevant how fast TPUs have embraced new nodes since the main challenges of self-driving functions seem to come from the execution of DL models. The higher transistor densities given by new nodes are often not translated in smaller devices, but in higher transistor counts for the same size and, in occasions, even in bigger devices. In FPGAs this translates into more resources, and with more available resources and bigger designs the complexity of place and route algorithms used synthesis is incremented, negatively affecting the design productivity.

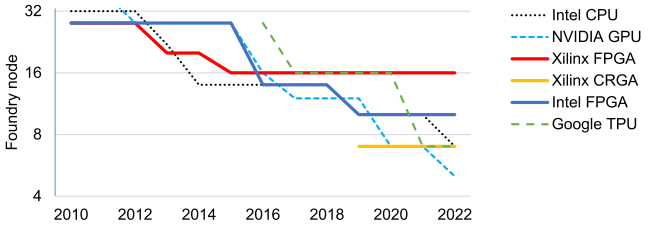


FIGURE 34. Timeline of the access to latest foundry nodes for different devices.

Also releveant, is that Xilinx has devoted their latest used foundry node to their new family of coarse-grain reconfigurable architectures (VERSAL).

3) COMPETING COMPUTING PLATFORMS

Future self-driving car systems will have several computing platform options to explore. There have been some attempts [311] to analyze how several architectures (CPU/GPU/FPGA) can be part of on self-driving cars. However, the landscape is highly dynamic, and the anlysis should be revisited often until a clear option provides a significant advantage. From our previous analysis, we estimate that an ideal computing platform should be able to acquire data from different sensors in real-time, have a computing power close to 1 POPS, and an energy efficiency above 5 TOPS/W.

Current mainstream multi-core processors contain large vector units (e.g. implementing AVX512) that can be leveraged to execute NN models, but they are still far from the former performance and energy efficiency requirements. A new breath of homogeneous many-core processors are addressing AI inference. A prominent example is the Esperanto ET-SoC-1 [312] that contains 1000 RISC-V cores. Pre-silicon estimations were claiming that it will be able to deliver more than 100 TOPS within 20 W of power. This would result in 5 TOPS/W.

Tesla, who is already selling more than 1 million cars per year, decided to develop its own FSD chip containing DPUs, which reaches 144 TOPS with a power consumption of 72 W, giving 2 TOPS/W. Tesla discontinued using chips from NVIDIA and Mobileye. Mobileye EyeQ5 reaches 24 TOPS, with a power consumption of 10 W, so 2.4 TOPS/W.

Some DPUs are already above the 10 TFLOPS/W range [313], [314]. On GPUs, the recent NVIDIA RTX 4090 is providing a significant computing power (82 fp16 TFLOPS), but at a high electric power cost 800 W TPD. Using these theoretical values we would get an energy efficiency of 0.125 TFLOPS/W According to Nvidia, the Drive Orin chip (optimized for automotive) is capable of 250 TOPS with a power consumption of 500 W, giving a 0.5 FLOPS/W.

FPGA manufacturers are integrating higher complexity blocks, actually transitioning from fine-grain to coarse-grain reconfigurable architectures CGRAs. Xilinx VERSAL devices claim 160 (int8) TOPS at less than 180 W. This values would represent an energy efficiency of 1 TOPS/W. On the other hand, some FPGAs designs (thus, using fine grine reconfigurable architectures) claim [310] 5.4 TOPS/W. This is the result of pushing down further the arithmetic precision.

We actually see some convergence in DL architectures. Reduced arithmetic precision, SIMD architectures, and strategies to increase memory locality and reduce external access. If many competing platforms are achieving a similar performance and energy efficiency range, then the dominance will be the result of other aspects, like the development cost, support, and the development ecosystem.

VII. CONCLUSION

The paper captures the state of FPGA implementations for vision-based applications on self-driving cars for the last decade. The vision-based problems are presented and related to the typical functions of autonomous cars. We identify several key problems where FPGAs have been considered as a valid implementing alternative: stereo-matching, semantic segmentation, lane detection, traffic sign recognition, traffic lights recognition, obstacle detection, and pedestrian

detection. The state-of-the-art of each of them is analyzed to identify the approaches, challenges and basic techniques being used in FPGA implementations. Published FPGA works are listed and compared and their achieved accuracy and performance are put in context with respect their competing alternative computing platforms. Based on the survey, we provide the state-of-the-art achievements and we identify the gaps for future research directions on FPGA-based accelerators for autonomous cars.

REFERENCES

[1] Y. J. Yasin, M. Grivna, and F. M. Abu-Zidan, "Global impact of COVID-19 pandemic on road traffic collisions," *World J. Emergency Surgery*, vol. 16, no. 1, pp. 1–14, Dec. 2021.

[2] World Health Organization. (2018). *Global Status Report on Road Safety 2018: Summary (no. who/nmh/nvi/18.20)*. [Online]. Available: https://apps.who.int/iris/bitstream/handle/10665/277370/WHO-NMH-NVI-18.20-eng.pdf

[3] *J3016-Taxonomy and Definitions for Terms Related to on-Road Motor Vehicle Automated Driving Systems*, Surface Vehicle Recommended Practive, SAE, Warrendale, PA, USA, 2016.

[4] S. Shreejith, K. Vipin, S. A. Fahmy, and M. Lukasiewycz, "An approach for redundancy in FlexRay networks using FPGA partial reconfiguration," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2013, pp. 721–724.

[5] J. Ahmad and A. Warren, "FPGA based deterministic latency image acquisition and processing system for automated driving systems," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2018, pp. 1–5.

[6] Y. Chen, T.-J. Yang, J. Emer, and V. Sze, "Understanding the limitations of existing energy-efficient design approaches for deep neural networks," *Energy*, vol. 2, no. L1, pp. 1–3, 2018.

[7] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A survey of FPGA-based neural network accelerator," 2017, *arXiv:1712.08934*.

[8] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, 2017, pp. 65–74.

[9] Y. Li and J. Ibanez-Guzman, "Lidar for autonomous driving: The principles, challenges, and trends for automotive lidar and perception systems," *IEEE Signal Process. Mag.*, vol. 37, no. 4, pp. 50–61, Jul. 2020.

[10] Z. Wan, B. Yu, T. Y. Li, J. Tang, Y. Zhu, Y. Wang, A. Raychowdhury, and S. Liu, "A survey of FPGA-based robotic computing," *IEEE Circuits Syst. Mag.*, vol. 21, no. 2, pp. 48–74, 2021.

[11] C. Badue, R. Guidolini, R. V. Carneiro, P. Azevedo, V. B. Cardoso, A. Forechi, L. Jesus, R. Berriel, T. M. Paixão, F. Mutz, L. de Paula Veronese, T. Oliveira-Santos, and A. F. De Souza, "Self-driving cars: A survey," *Exp. Syst. Appl.*, vol. 165, Mar. 2020, Art. no. 113816.

[12] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE Access*, vol. 8, pp. 58443–58469, 2020.

[13] D. K. Dewangan and S. P. Sahu, "Towards the design of vision-based intelligent vehicle system: Methodologies and challenges," *Evol. Intell.*, pp. 1–42, Mar. 2022. [Online]. Available: https://link.springer.com/article/10.1007/s12065-022-00713-2

[14] X. Feng, Y. Jiang, X. Yang, M. Du, and X. Li, "Computer vision algorithms and hardware implementations: A survey," *Integration*, vol. 69, pp. 309–320, Nov. 2019.

[15] M. A. Arshad, S. Shahriar, and A. Sagahyroon, "On the use of FPGAs to implement CNNs: A brief review," in *Proc. Int. Conf. Comput., Electron. Commun. Eng. (iCCECE)*, Aug. 2020, pp. 230–236.

[16] R. Ayachi, Y. Said, and A. B. Abdelali, "Optimizing neural networks for efficient FPGA implementation: A survey," *Arch. Comput. Methods Eng.*, vol. 28, pp. 4537–4547, Jan. 2021.

[17] A. DeHon, "Trends toward spatial computing architectures," in *Proc. IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers. ISSCC. 1st Ed.*, Feb. 1999, pp. 362–363.

[18] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proc. Spring Joint Comput. Conf. AFIPS (Spring)*, 1967, pp. 483–485.

[19] D. Castells-Rufas, A. Saa-Garriga, and J. Carrabina, "Energy efficiency of many-soft-core processors," in *Proc. Int. Workshop High Perform. Energy Efficient Embedded Syst.*, 2016, pp. 1–8.

[20] Y.-K. Choi, Y. Chi, W. Qiao, N. Samardzic, and J. Cong, "HBM connect: High-performance HLS interconnect for FPGA HBM," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, 2021, pp. 116–126.

[21] E. Nurvitadhi, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. O. G. Hock, Y. T. Liew, K. Srivatsan, D. Moss, S. Subhaschandra, and G. Boudoukh, "Can FPGAs beat GPUs in accelerating next-generation deep neural networks?" in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, Feb. 2017, pp. 1–14, doi: 10.1145/3020078.3021740.

[22] Z. Li, Y. Zhang, J. Wang, and J. Lai, "A survey of FPGA design for AI era," *J. Semiconductors*, vol. 41, no. 2, Feb. 2020, Art. no. 021402.

[23] R. Nane, V.-M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, and K. Bertels, "A survey and evaluation of FPGA high-level synthesis tools," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 35, no. 10, pp. 1591–1604, Oct. 2016.

[24] G. Chen, H. Cao, J. Conradt, H. Tang, F. Rohrbein, and A. Knoll, "Event-based neuromorphic vision for autonomous driving: A paradigm shift for bio-inspired visual sensing and perception," *IEEE Signal Process. Mag.*, vol. 37, no. 4, pp. 34–49, Jul. 2020.

[25] P. Hurney, "Real-time detection of pedestrians in night-time conditions using a vehicle mounted infrared camera," Ph.D. dissertation, Dept. Elect. Electron. Eng., Nat. Univ. Ireland, Galway, Ireland, 2016.

[26] J. Borrego-Carazo, D. Castells-Rufas, E. Biempica, and J. Carrabina, "Resource-constrained machine learning for ADAS: A systematic review," *IEEE Access*, vol. 8, pp. 40573–40598, 2020.

[27] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[28] J. Stewart. (2018). *Self-Driving Cars Use Crazy Amounts of Power, and It's Becoming a Problem*. Accessed: Apr. 9, 2020. [Online]. Available: https://www.wired.com/story/self-driving-cars-power-consumption-nvidia-chip

[29] J. H. Gawron, G. A. Keoleian, R. D. De Kleine, T. J. Wallington, and H. C. Kim, "Life cycle assessment of connected and automated vehicles: Sensing and computing subsystem and vehicle level effects," *Environ. Sci. Technol.*, vol. 52, no. 5, pp. 3249–3256, Mar. 2018.

[30] J. A. Baxter, D. A. Merced, D. J. Costinett, L. M. Tolbert, and B. Ozpineci, "Review of electrical architectures and power requirements for automated vehicles," in *Proc. IEEE Transp. Electrific. Conf. Expo. (ITEC)*, Jun. 2018, pp. 944–949.

[31] P. H. Becker, J. M. Arnau, and A. González, "Demystifying power and performance bottlenecks in autonomous driving systems," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Oct. 2020, pp. 205–215.

[32] I. Hasan, S. Liao, J. Li, S. U. Akram, and L. Shao, "Generalizable pedestrian detection: The elephant in the room," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2021, pp. 11328–11337.

[33] S. Vitabile, A. De Paola, and F. Sorbello, "A real-time non-intrusive FPGA-based drowsiness detection system," *J. Ambient Intell. Humanized Comput.*, vol. 2, no. 4, pp. 251–262, Dec. 2011.

[34] F. Moreno, F. Aparicio, W. Hernández, and J. Páez, "A low-cost real-time FPGA solution for driver drowsiness detection," in *Proc. IECON 29th Annu. Conf. IEEE Ind. Electron. Soc.*, vol. 2, Nov. 2003, pp. 1396–1401.

[35] E. Shang, J. Li, X. An, and H. He, "A real-time lane departure warning system based on FPGA," in *Proc. 14th Int. IEEE Conf. Intell. Transp. Syst. (ITSC)*, Oct. 2011, pp. 1243–1248.

[36] J. Zhao, B. Xie, and X. Huang, "Real-time lane departure and front collision warning system on an FPGA," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, Sep. 2014, pp. 1–5.

[37] J. Xiao, S. Li, and B. Sun, "A real-time system for lane detection based on FPGA and DSP," *Sens. Imag.*, vol. 17, no. 1, pp. 1–13, Dec. 2016.

[38] J. Guan, F. An, X. Zhang, L. Chen, and H. Mattausch, "Real-time straight-line detection for XGA-size videos by Hough transform with parallelized voting procedures," *Sensors*, vol. 17, no. 2, p. 270, Jan. 2017.

[39] X. An, E. Shang, J. Song, J. Li, and H. He, "Real-time lane departure warning system based on a single FPGA," *EURASIP J. Image Video Process.*, vol. 2013, no. 38, pp. 1–18, Jul. 2013.

[40] S. Malmir and M. Shalchian, "Design and FPGA implementation of dual-stage lane detection, based on Hough transform and localized stripe features," *Microprocessors Microsyst.*, vol. 64, pp. 12–22, Feb. 2019.

[41] H. Zhan and L. Chen, "Lane detection image processing algorithm based on FPGA for intelligent vehicle," in *Proc. Chin. Autom. Congr. (CAC)*, Nov. 2019, pp. 1190–1196.

[42] F. Schumacher and T. Greiner, "Matching cost computation algorithm and high speed FPGA architecture for high quality real-time semi global matching stereo vision for road scenes," in *Proc. 17th Int. IEEE Conf. Intell. Transp. Syst. (ITSC)*, Oct. 2014, pp. 3064–3069.

[43] M. Dehnavi and M. Eshghi, "FPGA based real-time on-road stereo vision system," *J. Syst. Archit.*, vol. 81, pp. 32–43, Nov. 2017, doi: 10.1016/j.sysarc.2017.10.002.

[44] O. Rahnama, T. Cavalleri, S. Golodetz, S. Walker, and P. Torr, "R3SGM: Real-time raster-respecting semi-global matching for power-constrained systems," in *Proc. Int. Conf. Field-Programmable Technol. (FPT)*, Dec. 2018, pp. 102–109.

[45] X. Zhang, H. Sun, S. Chen, L. Song, and N. Zheng, "NIPM-sWMF: Toward efficient FPGA design for high-definition large-disparity stereo matching," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 29, no. 5, pp. 1530–1543, May 2019.

[46] G. Chen, Y. Ling, T. He, H. Meng, S. He, Y. Zhang, and K. Huang, "StereoEngine: An FPGA-based accelerator for real-time high-quality stereo estimation with binary neural network," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 11, pp. 4179–4190, Nov. 2020.

[47] S. Shrivastava, Z. Choudhury, S. Khandelwal, and S. Purini, "FPGA accelerator for stereo vision using semi-global matching through dependency relaxation," in *Proc. 30th Int. Conf. Field-Programmable Log. Appl. (FPL)*, no. 1, Aug. 2020, pp. 304–309.

[48] K. Wei, Y. Kuno, M. Arai, and H. Amano, "RT-libSGM: An implementation of a real-time stereo matching system on FPGA," in *Proc. Int. Symp. Highly-Efficient Accel. Reconfigurable Technol.*, Jun. 2022, pp. 1–9.

[49] M. Russell and S. Fischaber, "OpenCV based road sign recognition on Zynq," in *Proc. 11th IEEE Int. Conf. Ind. Informat. (INDIN)*, Jul. 2013, pp. 596–601.

[50] Y. Han, K. Virupakshappa, E. Vitor Silva Pinto, and E. Oruklu, "Hardware/software co-design of a traffic sign recognition system using Zynq FPGAs," *Electronics*, vol. 4, no. 4, pp. 1062–1089, Dec. 2015. [Online]. Available: http://www.mdpi.com/2079-9292/4/4/1062

[51] J. Zhao, S. Zhu, and X. Huang, "Real-time traffic sign detection using SURF features on FPGA," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, Sep. 2013, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/6670350/

[52] W. Shi, X. Li, Z. Yu, and G. Overett, "An FPGA-based hardware accelerator for traffic sign detection," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 4, pp. 1362–1372, Apr. 2017.

[53] W. Farhat, S. Sghaier, H. Faiedh, and C. Souani, "Design of efficient embedded system for road sign recognition," *J. Ambient Intell. Humanized Comput.*, vol. 10, pp. 1–17, Jan. 2018, doi: 10.1007/s12652-017-0673-3.

[54] M. Lechner, A. Jantsch, and S. M. P. Dinakarrao, "ResCoNN: Resource-efficient FPGA-accelerated CNN for traffic sign classification," in *Proc. 10th Int. Green Sustain. Comput. Conf. (IGSC)*, Oct. 2019, pp. 1–6.

[55] K. Wei, K. Honda, and H. Amano, "FPGA design for autonomous vehicle driving using binarized neural networks," in *Proc. Int. Conf. Field-Programmable Technol. (FPT)*, Dec. 2018, pp. 428–431.

[56] T. Fukuchi, M. O. Ikechukwu, and A. B. Abdallah, "Design and optimization of a deep neural network architecture for traffic light detection," in *Proc. 2nd ACM Chapter Int. Conf. Educ. Technol., Lang. Tech. Commun. (ETLTC)*. Les Ulis, France: EDP Sciences, vol. 77, 2020, p. 01002.

[57] M. Hahnle, F. Saxen, M. Hisung, U. Brunsmann, and K. Doll, "FPGA-based real-time pedestrian detection on high-resolution images," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops*, Jun. 2013, pp. 629–635.

[58] X. Ma, W. A. Najjar, and A. K. Roy-Chowdhury, "Evaluation and acceleration of high-throughput fixed-point object detection on FPGAs," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 6, pp. 1051–1062, Jun. 2015.

[59] A. Khan, M. U. K. Khan, M. Bilal, and C.-M. Kyung, "Hardware architecture and optimization of sliding window based pedestrian detection on FPGA for high resolution images by varying local features," in *Proc. IFIP/IEEE Int. Conf. Very Large Scale Integr. (VLSI-SoC)*, Oct. 2015, pp. 142–148.

[60] J. Rettkowski, A. Boutros, and D. Göhringer, "Real-time pedestrian detection on a Xilinx Zynq using the HOG algorithm," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, Dec. 2015, pp. 1–8. [Online]. Available: http://ieeexplore.ieee.org/document/7393339/

[61] C. Kelly, F. M. Siddiqui, B. Bardak, Y. Wu, R. Woods, and K. Rafferty, "FPGA soft-core processors, compiler and hardware optimizations validated using HOG," in *Proc. Int. Symp. Appl. Reconfigurable Comput.*, V. Bonato, C. Bouganis, and M. Gorgon, Eds. Cham, Switzerland: Springer, 2016, pp. 78–90.

[62] V. Ngo, A. Casadevall, M. Codina, D. Castells-Rufas, and J. Carrabina, "A high-performance HOG extractor on FPGA," in *Proc. Workshop High Perform. Energy Efficient Embedded Syst.*, Manchester, U.K., 2018, pp. 1–6.

[63] A. Moussawi, K. Haddad, and A. Chahine, "An FPGA-accelerated design for deep learning pedestrian detection in self-driving vehicles," 2018, *arXiv:1809.05879*.

[64] A. Luo, F. An, X. Zhang, and H.-J. Mattausch, "A hardware-efficient recognition accelerator using Haar-like feature and SVM classifier," *IEEE Access*, vol. 7, pp. 14472–14487, 2019.

[65] S. Ghaffari, P. Soleimani, K. F. Li, and D. W. Capson, "A novel hardware—Software co-design and implementation of the HOG algorithm," *Sensors*, vol. 20, no. 19, p. 5655, Oct. 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/19/5655

[66] S. Liu, H. Fan, X. Niu, H.-C. Ng, Y. Chu, and W. Luk, "Optimizing CNN-based segmentation with deeply customized convolutional and deconvolutional architectures on FPGA," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 3, pp. 1–22, Sep. 2018.

[67] Y. Sada, M. Shimoda, A. Jinguji, and H. Nakahara, "A dataflow pipelining architecture for tile segmentation with a sparse MobileNet on an FPGA," in *Proc. Int. Conf. Field-Programmable Technol. (ICFPT)*, Dec. 2019, pp. 267–270.

[68] M. Shimoda, Y. Sada, and H. Nakahara, "Filter-wise pruning approach to FPGA implementation of fully convolutional network for semantic segmentation," in *Proc. Int. Symp. Appl. Reconfigurable Comput.* Cham, Switzerland: Springer, 2019, pp. 371–386.

[69] H. Huang, Y. Wu, M. Yu, X. Shi, F. Qiao, L. Luo, Q. Wei, and X. Liu, "EDSSA: An encoder–decoder semantic segmentation networks accelerator on OpenCL-based FPGA platform," *Sensors*, vol. 20, no. 14, p. 3969, Jul. 2020.

[70] M. Miyama, "FPGA Implementation of 3-bit quantized CNN for semantic segmentation," *J. Phys., Conf.*, vol. 1729, Jan. 2021, Art. no. 012004, doi: 10.1088/1742-6596/1729/1/012004.

[71] W. Jia, J. Cui, X. Zheng, and Q. Wu, "Design and implementation of real-time semantic segmentation network based on FPGA," in *Proc. 7th Int. Conf. Comput. Artif. Intell.*, Apr. 2021, pp. 321–325.

[72] H. Nakahara, H. Yonekawa, and S. Sato, "An object detector based on multiscale sliding window search using a fully pipelined binarized CNN on an FPGA," in *Proc. Int. Conf. Field Program. Technol. (ICFPT)*, Dec. 2017, pp. 168–175.

[73] H. Nakahara, H. Yonekawa, T. Fujii, and S. Sato, "A lightweight YOLOv2: A binarized CNN with a parallel support vector regression for an FPGA," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, Feb. 2018, pp. 31–40. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3174243.3174266

[74] Y. Ma, T. Zheng, Y. Cao, S. Vrudhula, and J.-S. Seo, "Algorithm-hardware co-design of single shot detector for fast object detection on FPGAs," in *Proc. Int. Conf. Comput.-Aided Design*, Nov. 2018, pp. 1–8.

[75] D. T. Nguyen, T. N. Nguyen, H. Kim, and H. J. Lee, "A high-throughput and power-efficient FPGA implementation of YOLO CNN for object detection," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 8, pp. 1861–1873, Aug. 2019.

[76] Z. Yu and C.-S. Bouganis, "A parameterisable FPGA-tailored architecture for YOLOv3-tiny," in *Applied Reconfigurable Computing. Architectures, Tools, and Applications*, F. Rincón, J. Barba, H. K. H. So, P. Diniz, and J. Caba, Eds. Cham, Switzerland: Springer, 2020, pp. 330–344.

[77] N. Scicluna, E. Gatt, O. Casha, I. Grech, and J. Micallef, "FPGA-based autonomous parking of a car-like robot using fuzzy logic control," in *Proc. 19th IEEE Int. Conf. Electron., Circuits, Syst. (ICECS)*, Dec. 2012, pp. 229–232.

[78] Y. Shimmyo, M. Arakawa, S. Mie, H. Saito, Y. Okuyama, and H. Yomogita, "Implementation of an autonomous driving system for FPT2018 FPGA design competition using the zynqberry processing board," in *Proc. Int. Conf. Field-Programmable Technol. (FPT)*. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers, Dec. 2018, pp. 410–413.

[79] M. Ruba, R. Nemes, F. P. Piglesan, H. Hedesiu, and C. Martis, "Complete elecric power steering system real-time model in the loop simulator," in *Proc. ELEKTRO*, May 2018, pp. 1–6.
[80] Q. Weikang, W. Li, X. Lingjun, and Z. Yifang, "Practical solution for automotive electronic throttle control based on FPGA," in *Proc. 9th Int. Conf. Signal Process.*, Oct. 2008, pp. 453–457.
[81] P. Maji, S. K. Patra, and K. Mahapatra, "Implementation of FPGA based fuzzy PI approximate control for automatic cruise control system," in *Proc. Int. Conf. Circuits, Commun., Control Comput.*, Nov. 2014, pp. 203–206.
[82] D. Thakur and A. Thakare, "Implementation of automatic reverse braking system on FPGA," *Int. J. Electron., Commun. Soft Comput. Sci. Eng.*, vol. 4, pp. 133–136, 2015. [Online]. Available: https://www.proquest.com/openview/b81b9681d0ccbf41438ac84fc3c31a3a/1?pq-origsite=gscholar&cbl=2029261
[83] S. Shreejith and S. A. Fahmy, "Extensible FlexRay communication controller for FPGA-based automotive systems," *IEEE Trans. Veh. Technol.*, vol. 64, no. 2, pp. 453–465, Feb. 2015.
[84] O. Sander, B. Glas, C. Roth, J. Becker, and K. D. Müller-Glaser, "Design of a vehicle-to-vehicle communication system on reconfigurable hardware," in *Proc. Int. Conf. Field-Programmable Technol. (FPT)*, Dec. 2009, pp. 14–21.
[85] H.-Y. Jheng, Y.-H. Chen, S.-J. Ruan, and Z. Qi, "FPGA implementation of high sampling rate in-car non-stationary noise cancellation based on adaptive Wiener filter," in *Proc. IEEE/IFIP 19th Int. Conf. VLSI System-on-Chip*, Oct. 2011, pp. 114–117.
[86] D. F. Syu, S. W. Syu, S. J. Ruan, Y. C. Huang, and C. K. Yang, "FPGA implementation of automatic speech recognition system in a car environment," in *Proc. IEEE 4th Global Conf. Consum. Electron. (GCCE)*. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers, Feb. 2016, pp. 485–486.
[87] L. Liu, S. Lu, R. Zhong, B. Wu, Y. Yao, Q. Zhang, and W. Shi, "Computing systems for autonomous driving: State of the art and challenges," *IEEE Internet Things J.*, vol. 8, no. 8, pp. 6469–6486, Apr. 2021.
[88] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *Int. J. Comput. Vis.*, vol. 47, nos. 1–3, pp. 7–42, Apr. 2002. [Online]. Available: https://www.middlebury.edu/stereo
[89] D. Hernandez-Juarez, A. Chacón, A. Espinosa, D. Vázquez, J. C. Moure, and A. M. López, "Embedded real-time stereo estimation via semi-global matching on the GPU," *Proc. Comput. Sci.*, vol. 80, pp. 143–153, Dec. 2016.
[90] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge, U.K.: Cambridge Univ. Press, Mar. 2003. [Online]. Available: https://www.cambridge.org/core/books/multiple-view-geometry-in-computer-vision/0B6F289C78B2B23F596CAA76D3D43F7A
[91] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, Jun. 2005, vol. 1, no. 1, pp. 886–893.
[92] R. A. Hamzah and H. Ibrahim, "Literature survey on stereo vision disparity map algorithms," *J. Sensors*, vol. 2016, pp. 1–23, Dec. 2016.
[93] H. Hirschmuller and S. Gehrig, "Stereo matching in the presence of sub-pixel calibration errors," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 437–444.
[94] N. Smolyanskiy, A. Kamenev, and S. Birchfield, "On the importance of stereo for accurate depth estimation: An efficient semi-supervised deep neural network approach," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2018, pp. 1007–1015.
[95] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2012, pp. 3354–3361.
[96] X. Cheng, Y. Zhong, M. Harandi, Y. Dai, X. Chang, H. Li, T. Drummond, and Z. Ge, "Hierarchical neural architecture search for deep stereo matching," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, pp. 22158–22169, 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/fc146be0b230d7e0a92e66a6114b840d-Paper.pdf
[97] D. Scharstein, H. Hirschmüller, Y. Kitajima, G. Krathwohl, N. Nešić, X. Wang, and P. Westling, "High-resolution stereo datasets with subpixel-accurate ground truth," in *Proc. German Conf. Pattern Recognit.* Cham, Switzerland: Springer, 2014, pp. 31–42.
[98] L. Lipson, Z. Teed, and J. Deng, "Raft-stereo: Multilevel recurrent field transforms for stereo matching," in *Proc. Int. Conf. 3D Vis. (DV)*, Dec. 2021, pp. 218–227.
[99] M. Menze and A. Geiger, "Object scene flow for autonomous vehicles," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 3061–3070.
[100] G. Yang, X. Song, C. Huang, Z. Deng, J. Shi, and B. Zhou, "DrivingStereo: A large-scale dataset for stereo matching in autonomous driving scenarios," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 899–908.
[101] Z. Huang, T. B. Norris, and P. Wang, "ES-Net: An efficient stereo matching network," 2021, *arXiv:2103.03922*.
[102] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, "Indoor segmentation and support inference from RGBD images," in *Proc. Eur. Conf. Comput. Vis.* Berlin, Germany: Springer, 2012, pp. 746–760.
[103] Z. Li, X. Wang, X. Liu, and J. Jiang, "BinsFormer: Revisiting adaptive bins for monocular depth estimation," 2022, *arXiv:2204.00987*.
[104] D. Eigen, C. Puhrsch, and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," 2014, *arXiv:1406.2283*.
[105] V. Guizilini, R. Ambrus, S. Pillai, A. Raventos, and A. Gaidon, "3D packing for self-supervised monocular depth estimation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 2485–2494.
[106] M. Klingner, J.-A. Termöhlen, J. Mikolajczyk, and T. Fingscheidt, "Self-supervised monocular depth estimation: Solving the dynamic object problem by semantic guidance," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2020, pp. 582–600.
[107] M. Siam, S. Elkerdawy, M. Jagersand, and S. Yogamani, "Deep semantic segmentation for automated driving: Taxonomy, roadmap and challenges," in *Proc. IEEE 20th Int. Conf. Intell. Transp. Syst. (ITSC)*, Oct. 2017, pp. 1–8.
[108] C. C. Kaymak and A. Uçar, "A brief survey and an application of semantic image segmentation for autonomous driving," in *Handbook of Deep Learning Applications*. Cham, Switzerland: Springer, 2019, pp. 161–200.
[109] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez, "The SYNTHIA dataset: A large collection of synthetic images for semantic segmentation of urban scenes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 3234–3243.
[110] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. SMCS-9, no. 1, pp. 62–66, Feb. 1979.
[111] L. Vincent and P. Soille, "Watersheds in digital spaces: An efficient algorithm based on immersion simulations," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 13, no. 6, pp. 583–598, Jun. 1991.
[112] M. Van den Bergh, X. Boix, G. Roig, B. de Capitani, and L. Van Gool, "Seeds: Superpixels extracted via energy-driven sampling," in *Proc. Eur. Conf. Comput. Vis.* Berlin, Germany: Springer, 2012, pp. 13–26.
[113] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 3431–3440.
[114] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected CRFs," 2014, *arXiv:1412.7062*.
[115] C. Yu, J. Wang, C. Gao, G. Yu, C. Shen, and N. Sang, "Context prior for scene segmentation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 12416–12425.
[116] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 2881–2890.
[117] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," 2017, *arXiv:1706.05587*.
[118] H. Zhao, X. Qi, X. Shen, J. Shi, and J. Jia, "ICNet for real-time semantic segmentation on high-resolution images," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 405–420.
[119] H. Zhao, Y. Zhang, S. Liu, J. Shi, C. C. Loy, D. Lin, and J. Jia, "PSANet: Point-wise spatial attention network for scene parsing," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 267–283.
[120] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. Int. Conf. Med. Image Comput. Comput.-Assist. Intervent*, 2015, pp. 234–241.
[121] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A deep convolutional encoder–decoder architecture for image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 12, pp. 2481–2495, Jan. 2017.
[122] G. Lin, A. Milan, C. Shen, and I. Reid, "RefineNet: Multi-path refinement networks for high-resolution semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1925–1934.

[123] J. Wang, K. Sun, T. Cheng, B. Jiang, C. Deng, Y. Zhao, D. Liu, Y. Mu, M. Tan, X. Wang, W. Liu, and B. Xiao, "Deep high-resolution representation learning for visual recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 10, pp. 3349–3364, Oct. 2021.

[124] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

[125] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet V2: Practical guidelines for efficient CNN architecture design," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 116–131.

[126] C. Yu, C. Gao, J. Wang, G. Yu, C. Shen, and N. Sang, "BiSeNet v2: Bilateral network with guided aggregation for real-time semantic segmentation," *Int. J. Comput. Vis.*, vol. 129, no. 11, pp. 3051–3068, Sep. 2021, doi: 10.1007/s11263-021-01515-2.

[127] G. J. Brostow, J. Fauqueur, and R. Cipolla, "Semantic object classes in video: A high-definition ground truth database," *Pattern Recognit. Lett.*, vol. 30, no. 2, pp. 88–97, 2009.

[128] Y. Zhu, K. Sapra, F. A. Reda, K. J. Shih, S. Newsam, A. Tao, and B. Catanzaro, "Improving semantic segmentation via video propagation and label relaxation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 8856–8865.

[129] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal visual object classes challenge: A retrospective," *Int. J. Comput. Vis.*, vol. 111, no. 1, pp. 98–136, Jan. 2015.

[130] B. Zoph, G. Ghiasi, T.-Y. Lin, Y. Cui, H. Liu, E. D. Cubuk, and Q. V. Le, "Rethinking pre-training and self-training," 2020, *arXiv:2006.06882*.

[131] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 3213–3223.

[132] A. Sagar and R. Soundrapandiyan, "Semantic segmentation with multi scale spatial attention for self driving cars," 2020, *arXiv:2007.12685*.

[133] H. A. Alhaija, S. K. Mustikovela, L. Mescheder, A. Geiger, and C. Rother, "Augmented reality meets computer vision: Efficient data generation for urban driving scenes," *Int. J. Comput. Vis.*, vol. 126, no. 9, pp. 961–972, Sep. 2018.

[134] A. B. Hillel, R. Lerner, D. Levi, and G. Raz, "Recent progress in road and lane detection: A survey," *Mach. Vis. Appl.*, vol. 25, no. 3, pp. 727–745, 2014.

[135] *Intelligent Transport Systems—Lane Departure Warning Systems—Performance Requirements and Test Procedures*, document ISO 17361:2017(en), 2017. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso:17361:ed-2:v1:en

[136] S. Shirke and R. Udayakumar, "Lane datasets for lane detection," in *Proc. Int. Conf. Commun. Signal Process. (ICCSP)*, Apr. 2019, pp. 0792–0796.

[137] M. Aly, "Real time detection of lane markers in urban streets," in *Proc. IEEE Intell. Vehicles Symp.*, Jun. 2008, pp. 7–12.

[138] L. Zhang, F. Jiang, J. Yang, B. Kong, A. Hussain, M. Gogate, and K. Dashtipour, "DNet-CNet: A novel cascaded deep network for real-time lane detection and classification," *J. Ambient Intell. Humanized Comput.*, pp. 1–16, Jul. 2022. [Online]. Available: https://link.springer.com/article/10.1007/s12652-022-04346-2

[139] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *Int. J. Robot. Res.*, vol. 32, no. 11, pp. 1231–1237, 2013.

[140] H. Wang, R. Fan, Y. Sun, and M. Liu, "Dynamic fusion module evolves drivable area and road anomaly detection: A benchmark and algorithms," *IEEE Trans. Cybern.*, vol. 52, no. 10, pp. 10750–10760, Oct. 2022.

[141] *TuSimple Dataset*. Accessed: Oct. 6, 2022. [Online]. Available: https://github.com/TuSimple/tusimple-benchmark/wiki

[142] Z. Qu, H. Jin, Y. Zhou, Z. Yang, and W. Zhang, "Focus on local: Detecting lane marker from bottom up via key point," 2021, *arXiv:2105.13680*.

[143] X. Pan, J. Shi, P. Luo, X. Wang, and X. Tang, "Spatial as deep: Spatial CNN for traffic scene understanding," *Proc. AAAI Conf. Artif. Intell.*, Apr. 2018, vol. 32, no. 1, pp. 1–8.

[144] L. Liu, X. Chen, S. Zhu, and P. Tan, "CondLaneNet: A Top-to-down lane detection framework based on conditional convolution," 2021, *arXiv:2105.05003*.

[145] K. Behrendt and R. Soussan, "Unsupervised labeled lane markers using maps," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshop (ICCVW)*, Oct. 2019, pp. 832–839.

[146] H. Abualsaud, S. Liu, D. B. Lu, K. Situ, A. Rangesh, and M. M. Trivedi, "LaneAF: Robust multi-lane detection with affinity fields," *IEEE Robot. Autom. Lett.*, vol. 6, no. 4, pp. 7477–7484, Oct. 2021.

[147] F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhavan, and T. Darrell, "BDD100K: A diverse driving dataset for heterogeneous multitask learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 2636–2645.

[148] Y. Hou, Z. Ma, C. Liu, and C. C. Loy, "Learning lightweight lane detection CNNs by self attention distillation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1013–1021.

[149] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "ENet: A deep neural network architecture for real-time semantic segmentation," 2016, *arXiv:1606.02147*.

[150] F. Chao, S. Yu-Pei, and J. Ya-Jie, "Multi-lane detection based on deep convolutional neural network," *IEEE Access*, vol. 7, pp. 150833–150841, 2019.

[151] D. Neven, B. D. Brabandere, S. Georgoulis, M. Proesmans, and L. V. Gool, "Towards end-to-end lane detection: An instance segmentation approach," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2018, pp. 286–291.

[152] C.-B. Wu, L.-H. Wang, and K.-C. Wang, "Ultra-low complexity block-based lane detection and departure warning system," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 29, no. 2, pp. 582–593, Feb. 2019.

[153] C. R. Jung and C. R. Kelber, "Lane following and lane departure using a linear-parabolic model," *Image Vis. Comput.*, vol. 23, no. 13 pp. 1192–1202, 2005.

[154] P. Phalguni, K. Ganapathi, V. Madumbu, R. Rajendran, and S. David, "Design and implementation of an automatic traffic sign recognition system on TI OMAP-L138," in *Proc. IEEE Int. Conf. Ind. Technol. (ICIT)*, Feb. 2013, pp. 1104–1109.

[155] S. B. Wali, M. A. Abdullah, M. A. Hannan, A. Hussain, S. A. Samad, P. J. Ker, and M. B. Mansor, "Vision-based traffic sign detection and recognition systems: Current trends and challenges," *Sensors*, vol. 19, no. 9, p. 2093, Jan. 2019.

[156] A. Mogelmose, M. M. Trivedi, and T. B. Moeslund, "Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey," *IEEE Trans. Intell. Transp. Syst.*, vol. 13, no. 4, pp. 1484–1497, Dec. 2012.

[157] A. Møgelmose, D. Liu, and M. M. Trivedi, "Detection of us Traffic signs," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 6, pp. 3116–3125, Dec. 2015.

[158] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel, "Detection of traffic signs in real-world images: The German traffic sign detection benchmark," in *Proc. Int. joint Conf. neural Netw. (IJCNN)*, Aug. 2013, pp. 1–8.

[159] J. Zhang, Z. Xie, J. Sun, X. Zou, and J. Wang, "A cascaded R-CNN with multiscale attention and imbalanced samples for traffic sign detection," *IEEE Access*, vol. 8, pp. 29742–29754, 2020.

[160] R. Timofte, K. Zimmermann, and L. Van Gool, "Multi-view traffic sign detection, recognition, and 3D localisation," *Mach. Vis. Appl.*, vol. 25, no. 3, pp. 633–647, Apr. 2014.

[161] Y. Yang, Y. Yu, and Z. Huang, "Traffic sign detection using deep random mapping autoencoder network," in *Proc. IEEE Int. Conf. Inf. Autom. (ICIA)*, Aug. 2018, pp. 911–916.

[162] D. Temel, T. Alshawi, M.-H. Chen, and G. AlRegib, "Challenging environments for traffic sign detection: Reliability assessment under inclement conditions," 2019, *arXiv:1902.06857*.

[163] S. Ahmed, U. Kamal, and M. K. Hasan, "DFR-TSD: A deep learning based framework for robust traffic sign detection under challenging weather conditions," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 6, pp. 5150–5162, Jun. 2022.

[164] F. Larsson and M. Felsberg, "Using Fourier descriptors and spatial models for traffic sign recognition," in *Proc. Scandin. Conf. Image Anal.* Berlin, Germany: Springer, 2011, pp. 238–249.

[165] D. Tabernik and D. Skocaj, "Deep learning for large-scale traffic-sign detection and recognition," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 4, pp. 1427–1440, Apr. 2020.

[166] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural Netw.*, vol. 32, pp. 323–332, Aug. 2012.

[167] A. Arcos-García, J. Álvarez-García, and L. M. Soria-Morillo, "Deep neural network for traffic sign recognition systems: An analysis of spatial transformers and stochastic optimisation methods," *Neural Netw.*, vol. 99, pp. 158–165, Mar. 2018.

[168] M. Mathias, R. Timofte, R. Benenson, and L. Van Gool, "Traffic sign recognition—How far are we from the solution?" in *Proc. Int. Joint Conf. Neural Netw.*, 2013, pp. 1–8.

[169] C. Lin, L. Li, W. Luo, K. C. P. Wang, and J. Guo, "Transfer learning based traffic sign recognition using inception-v3 model," *Periodica Polytechnica Transp. Eng.*, vol. 47, no. 3, pp. 242–250, Aug. 2018.
[170] D. Temel, G. Kwon, M. Prabhushankar, and G. AlRegib, "CURE-TSR: Challenging unreal and real environments for traffic sign recognition," 2017, *arXiv:1712.02463*.
[171] A. Batool, M. W. Nisar, J. H. Shah, A. Rehman, and T. Sadad, "IELMNet: An application for traffic sign recognition using CNN and ELM," in *Proc. 1st Int. Conf. Artif. Intell. Data Anal. (CAIDA)*, 2021, pp. 132–137.
[172] B. Sanyal, R. K. Mohapatra, and R. Dash, "Traffic sign recognition: A survey," in *Proc. Int. Conf. Artif. Intell. Signal Process. (AISP)*, Jan. 2020, pp. 1–6.
[173] H. Gomez-Moreno, S. Maldonado-Bascon, P. Gil-Jimenez, and S. Lafuente-Arroyo, "Goal evaluation of segmentation algorithms for traffic sign recognition," *IEEE Trans. Intell. Transp. Syst.*, vol. 11, no. 4, pp. 917–930, Apr. 2010.
[174] H. Bay, T. Tuytelaars, and L. V. Gool, "SURF: Speeded up robust features," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2006, pp. 404–417.
[175] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. pattern Recognit. (CVPR)*, vol. 1, Dec. 2001, p. 1–8.
[176] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, Jul. 1995.
[177] Y. Wu, Y. Liu, J. Li, H. Liu, and X. Hu, "Traffic sign detection based on convolutional neural networks," in *Proc. Int. Joint Conf. Neural Netw.*, Aug. 2013, pp. 1–7.
[178] J. Li, X. Liang, Y. Wei, T. Xu, J. Feng, and S. Yan, "Perceptual generative adversarial networks for small object detection," in *Proc. IEEE Conf. Comput. Vis. pattern Recognit. (CVPR)*, Jul. 2017, pp. 1222–1230. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2017/html/Li_Perceptual_Generative_Adversarial_CVPR_2017_paper.html
[179] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot MultiBox detector," 2015, *arXiv:1512.02325*.
[180] P. Garg, D. R. Chowdhury, and V. N. More, "Traffic sign recognition and classification using YOLOv2, faster RCNN and SSD," in *Proc. 10th Int. Conf. Comput., Commun. Netw. Technol. (ICCCNT)*, Jul. 2019, pp. 1–5.
[181] A. Dinesh Kumar, "Novel deep learning model for traffic sign detection using capsule networks," 2018, *arXiv:1805.04424*.
[182] L. Abdi and A. Meddeb, "Deep learning traffic sign detection, recognition and augmentation," in *Proc. Symp. Appl. Comput.* New York, NY, USA: Association for Computing Machinery, Apr. 2017, pp. 131–136.
[183] A. Wong, M. J. Shafiee, and M. S. Jules, "MicronNet: A highly compact deep convolutional neural network architecture for real-time embedded traffic sign classification," *IEEE Access*, vol. 6, pp. 59803–59810, 2018.
[184] C. Dewi, R.-C. Chen, Y.-T. Liu, X. Jiang, and K. D. Hartomo, "Yolo v4 for advanced traffic sign recognition with synthetic training data generated by various GAN," *IEEE Access*, vol. 9, pp. 97228–97242, 2021.
[185] J. Zhang, M. Huang, X. Li, and X. Jin, "A real-time Chinese traffic sign detection algorithm based on modified YOLOv2," *Algorithms*, vol. 10, no. 4, p. 127, Nov. 2017. [Online]. Available: https://www.mdpi.com/1999-4893/10/4/127
[186] J. Jiang, S. Bao, W. Shi, and Z. Wei, "Improved traffic sign recognition algorithm based on YOLO V3 algorithm," *J. Comput. Appl.*, vol. 40, pp. 2472–2478, Apr. 2020.
[187] J. Li and Z. Wang, "Real-time traffic sign recognition based on efficient CNNs in the wild," *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 3, pp. 975–984, Mar. 2018.
[188] X. Bangquan and W. X. Xiong, "Real-time embedded traffic sign recognition using efficient convolutional neural network," *IEEE Access*, vol. 7, pp. 53330–53346, 2019.
[189] S. Saha, M. S. Islam, M. A. B. Khaled, and S. Tairin, "An efficient traffic sign recognition approach using a novel deep neural network selection architecture," in *Emerging Technologies in Data Mining and Information Security* (Advances in Intelligent Systems and Computing), A. Abraham, P. Dutta, J. K. Mandal, A. Bhattacharya, and S. Dutta, Eds. Singapore: Springer, 2019, pp. 849–862.
[190] N. S. Artamonov and P. Y. Yakimov, "Towards real-time traffic sign recognition via YOLO on a mobile GPU," *J. Phys., Conf.*, vol. 1096, Sep. 2018, Art. no. 012086, doi: 10.1088/1742-6596/1096/1/012086.
[191] R. Huang, J. Pedoeem, and C. Chen, "YOLO-LITE: A real-time object detection algorithm optimized for non-GPU computers," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2018, pp. 2503–2510.
[192] D. Babić, D. Babić, M. Fiolić, and Ž. Šarić, "Analysis of market-ready traffic sign recognition systems in cars: A test field study," *Energies*, vol. 14, no. 12, p. 3697, Jun. 2021.
[193] M. Diaz, P. Cerri, G. Pirlo, M. A. Ferrer, and D. Impedovo, "A survey on traffic light detection," in *Proc. Int. Conf. Image Anal. Process.* Cham, Switzerland: Springer, 2015, pp. 201–208.
[194] Y. Ji, M. Yang, Z. Lu, and C. Wang, "Integrating visual selective attention model with HOG features for traffic light detection and recognition," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2015, pp. 280–285.
[195] Z. Shi, Z. Zou, and C. Zhang, "Real-time traffic light detection with adaptive background suppression filter," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 3, pp. 690–700, Mar. 2016.
[196] C. Oh, D. Sim, and H. J. Kim, "Traffic light recognition based on one-dimensional convolutional neural network," in *Proc. IEEE 23rd Int. Conf. Intell. Transp. Syst. (ITSC)*, Sep. 2020, pp. 1–6.
[197] M. B. Jensen, M. P. Philipsen, C. Bahnsen, A. Møgelmose, T. B. Moeslund, and M. M. Trivedi, "Traffic light detection at night: Comparison of a learning-based detector and three model-based detectors," in *Proc. Int. Symp. Vis. Comput.* Cham, Switzerland: Springer, 2015, pp. 774–783.
[198] M. P. Philipsen, M. B. Jensen, A. Mogelmose, T. Moeslund, and M. M. Trivedi, "Learning based traffic light detection: Evaluation on challenging dataset," in *Proc. 18th IEEE Intell. Transp. Syst. Conf.*, Sep. 2015, pp. 2341–2345.
[199] H.-K. Kim, K.-Y. Yoo, J. H. Park, and H.-Y. Jung, "Traffic light recognition based on binary semantic segmentation network," *Sensors*, vol. 19, no. 7, p. 1700, Apr. 2019.
[200] K. Behrendt, L. Novak, and R. Botros, "A deep learning approach to traffic lights: Detection, tracking, and classification," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2017, pp. 1370–1377.
[201] A. Fregin, J. Müller, U. Krebel, and K. Dietmayer, "The DriveU traffic light dataset: Introduction and comparison with existing datasets," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 3376–3383.
[202] M. Bach, D. Stumper, and K. Dietmayer, "Deep convolutional traffic light recognition for automated driving," in *Proc. 21st Int. Conf. Intell. Transp. Syst. (ITSC)*, Nov. 2018, pp. 851–858.
[203] N. Bernini, M. Bertozzi, L. Castangia, M. Patander, and M. Sabbatelli, "Real-time obstacle detection using stereo vision for autonomous ground vehicles: A survey," in *Proc. 17th Int. IEEE Conf. Intell. Transp. Syst. (ITSC)*, Oct. 2014, pp. 873–878.
[204] H. Badino, U. Franke, and D. Pfeiffer, "The stixel world—A compact medium level representation of the 3D-world," in *Pattern Recognition*. Berlin, Germany: Springer, 2009, pp. 51–60.
[205] H. Badino, U. Franke, and R. Mester, "Free space computation using stochastic occupancy grids and dynamic programming," in *Proc. Workshop Dyn. Vis. (ICCV)*, vol. 20. Rio de Janeiro, Brazil: Citeseer, Oct. 2007, p. 73.
[206] G. S. U. Franke, C. Rabe, and H. Badino, "6D-vision: Fusion of stereo and motion for robust environment perception," in *Proc. Joint Pattern Recognit. Symp.* Berlin, Germany: Springer, 2005, pp. 216–223.
[207] D. Pfeiffer and U. Franke, "Efficient representation of traffic scenes by means of dynamic stixels," in *Proc. IEEE Intell. Vehicles Symp.*, Jun. 2010, pp. 217–224.
[208] P. Pinggera, S. Ramos, S. Gehrig, U. Franke, C. Rother, and R. Mester, "Lost and found: Detecting small road hazards for self-driving vehicles," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2016, pp. 1099–1106.
[209] D. Hendrycks, S. Basart, M. Mazeika, M. Mostajabi, J. Steinhardt, and D. Song, "Scaling out-of-distribution detection for real-world settings," 2019, *arXiv:1911.11132*. [Online]. Available: https://arxiv.org/abs/1911.11132
[210] R. Chan, K. Lis, S. Uhlemeyer, H. Blum, S. Honari, R. Siegwart, P. Fua, M. Salzmann, and M. Rottmann, "SegmentMeIfYouCan: A benchmark for anomaly segmentation," 2021, *arXiv:2104.14812*.
[211] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dolí, "Microsoft COCO: Common objects in context," in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 740–755.
[212] T. Ahmad and Y. Ma, "Performance evaluation of faster R-CNN for on-road object detection on graphical processing unit and central processing unit," in *Proc. Int. Conf. Intell. Comput.* Cham, Switzerland: Springer, 2019, pp. 99–108.
[213] G. K. Erabati and H. Araujo, "Dynamic obstacle detection in traffic environments," in *Proc. 13th Int. Conf. Distrib. Smart Cameras*, Sep. 2019, pp. 1–2.

[214] K. Gupta, S. A. Javed, V. Gandhi, and K. M. Krishna, "MergeNet: A deep net architecture for small obstacle discovery," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 5856–5862.
[215] D. Fontanel, F. Cermelli, M. Mancini, and B. Caputo, "Detecting anomalies in semantic segmentation with prototypes," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2021, pp. 113–121.
[216] R. Chan, M. Rottmann, and H. Gottschalk, "Entropy maximization and meta classification for out-of-distribution detection in semantic segmentation," 2020, *arXiv:2012.06575*.
[217] K. Pranav and J. Manikandan, "Design and evaluation of a real-time pedestrian detection system for autonomous vehicles," in *Proc. Zooming Innov. Consum. Technol. Conf. (ZINC)*, May 2020, pp. 155–159.
[218] P. Dollar, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: A benchmark," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 304–311.
[219] A. Hannan Khan, M. Munir, L. van Elst, and A. Dengel, "F2DNet: Fast focal detection network for pedestrian detection," 2022, *arXiv:2203.02331*.
[220] S. Zhang, R. Benenson, and B. Schiele, "CityPersons: A diverse dataset for pedestrian detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 3213–3221.
[221] C. Wojek, S. Walk, and B. Schiele, "Multi-cue onboard pedestrian detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 794–801.
[222] A. Krizhevsky, I. Sulskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, vol. 25, 2012, pp. 1–9.
[223] Y. LeCun, "Handwritten digit recognition with a back-propagation network," in *Proc. Adv. Neural Inf. Process. Syst.*, 1990, pp. 396–404.
[224] A. Giachetti, "Matching techniques to compute image motion," *Image Vis. Comput.*, vol. 18, no. 3, pp. 247–260, Feb. 2000.
[225] K. R. Rao and P. Yip, *Discrete Cosine Transform: Algorithms, Advantages, Applications*. Cambridge, MA, USA: Academic, 2014.
[226] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, 2004.
[227] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (SURF)," *Comput. Vis. Image Understand.*, vol. 110, no. 3, pp. 346–359, 2008.
[228] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *Proc. Int. Conf. Comput. Vis.*, Nov. 2011, pp. 2564–2571.
[229] R. Zabih and J. Woodfill, "Non-parametric local transforms for computing visual correspondence," in *Proc. Eur. Conf. Comput. Vis.* Berlin, Germany: Springer, 1994, pp. 151–158.
[230] P. V. Hough, "Method and means for recognizing complex patterns," U.S. Patent 3 069 654, Dec. 18, 1962.
[231] M. A. Fischler and R. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, 1981, doi: 10.1145/358669.358692.
[232] A. Saidi, S. Ben Othman, M. Dhouibi, and S. Ben Saoud, "FPGA-based implementation of classification techniques: A survey," *Integration*, vol. 81, pp. 280–299, Nov. 2021.
[233] P. C. Sen, M. Hajra, and M. Ghosh, "Supervised classification algorithms in machine learning: A survey and review," in *Emerging Technology in Modelling and Graphics*. Singapore: Springer, 2020, pp. 99–111.
[234] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.
[235] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1026–1034.
[236] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and & 0.5MB model size," 2016, *arXiv:1602.07360*.
[237] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4700–4708.
[238] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial transformer networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, Dec. 2015, pp. 2017–2025.
[239] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 580–587.
[240] R. Girshick, "Fast R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1440–1448. [Online]. Available: https://openaccess.thecvf.com/content_iccv_2015/html/Girshick_Fast_R-CNN_ICCV_2015_paper.html
[241] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788.
[242] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis.*, Oct. 2017, pp. 2961–2969.
[243] D. Castells-Rufas, "Scalable parallel architectures on reconfigurable platforms," Ph.D. dissertation, Dept. Microelectron. Electron. Syst., Universitat Autònoma de Barcelona, Bellaterra, Spain, 2016.
[244] M. Budiu, G. Venkataramani, T. Chelcea, and S. C. Goldstein, "Spatial computation," in *Proc. 11th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2004, pp. 14–26.
[245] B. Lakshmi and A. Dhar, "Cordic architectures: A survey," in *Proc. VLSI Design*, vol. 2010, 2010, pp. 1–10.
[246] H. T. Kung and C. E. Leiserson, "Systolic arrays for (VLSI)," Dept. Comput. Sci., Carnegie-Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU-CS-79-103, 1978.
[247] R. Gadea, J. Cerdá, F. Ballester, and A. Mocholí, "Artificial neural network implementation on a single FPGA of a pipelined on-line backpropagation," in *Proc. 13th Int. Symp. Syst. Synth.*, 2000, pp. 225–230.
[248] P. Plagwitz, F. Hannig, M. Strobel, C. Strohmeyer, and J. Teich, "A safari through FPGA-based neural network compilation and design automation flows," in *Proc. IEEE 29th Annu. Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*, May 2021, pp. 10–19.
[249] R. Wu, X. Guo, J. Du, and J. Li, "Accelerating neural network inference on FPGA-based platforms—A survey," *Electronics*, vol. 10, no. 9, p. 1025, Apr. 2021.
[250] S. Mittal, "A survey of FPGA-based accelerators for convolutional neural networks," *Neural Comput. Appl.*, vol. 32, no. 4, pp. 1109–1139, Feb. 2020.
[251] M. Lebedev and P. Belecky, "A survey of open-source tools for FPGA-based inference of artificial neural networks," in *Proc. Ivannikov Memorial Workshop (IVMEM)*, Sep. 2021, pp. 50–56.
[252] D. Wang, J. An, and K. Xu, "PipeCNN: An OpenCL-based FPGA accelerator for large-scale convolution neuron networks," 2016, *arXiv:1611.02450*.
[253] R. DiCecco, G. Lacey, J. Vasiljevic, P. Chow, G. Taylor, and S. Areibi, "Caffeinated FPGAs: FPGA framework for convolutional neural networks," in *Proc. Int. Conf. Field-Programmable Technol. (FPT)*. Piscataway, NJ, USA Institute of Electrical and Electronics Engineers, Dec. 2016, pp. 265–268.
[254] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping CNN onto embedded FPGA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 1, pp. 35–47, Jan. 2017.
[255] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. 4th Int. Conf. Learn. Represent. (ICLR)*, 2016, pp. 1–14.
[256] J. Qiu, S. Song, Y. Wang, H. Yang, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, and N. Xu, "Going deeper with embedded FPGA platform for convolutional neural network," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA)*, 2016, pp. 26–35. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2847263.2847265
[257] R. Zhao, X. Niu, Y. Wu, W. Luk, and Q. Liu, "Optimizing CNN-based object detection algorithms on embedded FPGA platforms," in *Proc. Int. Symp. Appl. Reconfigurable Comput.*, vol. 10216. Berlin, Germany: Springer-Verlag, 2017, pp. 255–267.
[258] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J.-S. Seo, and Y. Cao, "Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays (FPGA)*, pp. 16–25, 2016. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2847263.2847276
[259] D. Gschwend, "ZynqNet: An FPGA-accelerated embedded convolutional neural network," 2020, *arXiv:2005.06892*.
[260] B. Liu, D. Zou, L. Feng, S. Feng, P. Fu, and J. Li, "An FPGA-based CNN accelerator integrating depthwise separable convolution," *Electronics*, vol. 8, no. 3, p. 281, Mar. 2019. [Online]. Available: https://www.mdpi.com/2079-9292/8/3/281
[261] M. P. Véstias, R. P. Duarte, J. T. de Sousa, and H. C. Neto, "Fast convolutional neural networks in low density FPGAs using zero-skipping and weight pruning," *Electronics*, vol. 8, no. 11, p. 1321, Nov. 2019. [Online]. Available: https://www.mdpi.com/2079-9292/8/11/1321

[262] Z. Liu, P. Chow, J. Xu, J. Jiang, Y. Dou, and J. Zhou, "A uniform architecture design for accelerating 2D and 3D CNNs on FPGAs," *Electronics*, vol. 8, no. 1, p. 65, Jan. 2019. [Online]. Available: https://www.mdpi.com/2079-9292/8/1/65
[263] K. Xu, X. Wang, and D. Wang, "A scalable OpenCL-based FPGA accelerator for YOLOv2," in *Proc. IEEE 27th Annu. Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*, Apr. 2019, p. 317.
[264] Y. Umuroglu, Y. Akhauri, N. J. Fraser, and M. Blott, "LogicNets: Co-designed neural networks and circuits for extreme-throughput applications," in *Proc. 30th Int. Conf. Field-Programmable Log. Appl. (FPL)*. Los Alamitos, CA, USA: IEEE Computer Society, Aug. 2020, pp. 291–297.
[265] J. Zhao, T. Liang, L. Feng, W. Ding, S. Sinha, W. Zhang, and S. Shen, "FP-Stereo: Hardware-efficient stereo vision for embedded applications," in *Proc. 30th Int. Conf. Field-Programmable Log. Appl. (FPL)*, 2020, pp. 269–276.
[266] M. Shimoda, Y. Sada, and H. Nakahara, "FPGA-based inter-layer pipelined accelerators for filter-wise weight-balanced sparse fully convolutional networks with overlapped tiling," *J. Signal Process. Syst.*, vol. 93, no. 5, pp. 499–512, May 2021.
[267] Q. Huang and J. Liu, "Practical limitations of lane detection algorithm based on Hough transform in challenging scenarios," *Int. J. Adv. Robotic Syst.*, vol. 18, no. 2, pp. 1–13, 2021.
[268] M. Qasaimeh, K. Denolf, J. Lo, K. Vissers, J. Zambreno, and P. H. Jones, "Comparing energy efficiency of CPU, GPU and FPGA implementations for vision kernels," in *Proc. IEEE Int. Conf. Embedded Softw. Syst. (ICESS)*, Jun. 2019, pp. 1–8.
[269] C. Gamez Serna and Y. Ruichek, "Traffic signs detection and classification for European urban environments," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 10, pp. 4388–4399, Oct. 2020.
[270] W. Chen, S. Ding, Z. Chai, D. He, W. Zhang, G. Zhang, Q. Peng, and W. Luo, "FPGA-based parallel implementation of SURF algorithm," in *Proc. IEEE 22nd Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2016, pp. 308–315.
[271] X.-H. Wu, R. Hu, and Y.-Q. Bao, "Parallelism optimized architecture on FPGA for real-time traffic light detection," *IEEE Access*, vol. 7, pp. 178167–178176, 2019.
[272] K. Abdelouahab, M. Pelcat, J. Sérot, F. Berry, and C. Ferrand, "Accelerating CNN inference on FPGAs: A survey," 2018, *arXiv:1806.01683*.
[273] K. Zeng, Q. Ma, J. W. Wu, Z. Chen, T. Shen, and C. Yan, "FPGA-based accelerator for object detection: A comprehensive survey," *J. Supercomput.*, vol. 78, pp. 1–41, Mar. 2022.
[274] X. Xu, T. Wang, Q. Lu, and Y. Shi, "Resource constrained cellular neural networks for real-time obstacle detection using FPGAs," in *Proc. 19th Int. Symp. Quality Electron. Design (ISQED)*. Washington, DC, USA: IEEE Computer Society, Mar. 2018, pp. 437–440.
[275] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 4013–4021.
[276] T. Simons and D.-J. Lee, "A review of binarized neural networks," *Electronics*, vol. 8, no. 6, p. 661, Jun. 2019. [Online]. Available: https://www.mdpi.com/2079-9292/8/6/661
[277] N. Li, S. Takaki, Y. Tomiokay, and H. Kitazawa, "A multistage dataflow implementation of a deep convolutional neural network based on FPGA for high-speed object recognition," in *Proc. IEEE Southwest Symp. Image Anal. Interpretation (SSIAI)*. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers, Mar. 2016, pp. 165–168.
[278] A. Balasubramaniam and S. Pasricha, "Object detection in autonomous vehicles: Status and open challenges," 2022, *arXiv:2201.07706*.
[279] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proc. 1st Annu. Conf. Robot Learn.*, 2017, pp. 1–16.
[280] T. Litman, "Autonomous vehicle implementation predictions," Victoria Transp. Policy Inst., Victoria, BC, Canada, Tech. Rep., 2022. [Online]. Available: https://vtpi.org/avip.pdf
[281] K. Kirkpatrick, "Still waiting for self-driving cars," *Commun. ACM*, vol. 65, no. 4, pp. 12–14, Apr. 2022.
[282] M. Sha and A. Boukerche, "Performance evaluation of CNN-based pedestrian detectors for autonomous vehicles," *Ad Hoc Netw.*, vol. 128, Apr. 2022, Art. no. 102784.
[283] N. P. Jouppi, D. Hyun Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin, G. Kurian, J. Laudon, S. Li, P. Ma, X. Ma, T. Norrie, N. Patil, S. Prasad, C. Young, Z. Zhou, and D. Patterson, "Ten lessons from three generations shaped Google's TPUv4i : Industrial product," in *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2021, pp. 1–14.
[284] D. Xiao, L. Zhuo, J. Li, and J. Li, "Structure-prior deep neural network for lane detection," *J. Vis. Commun. Image Represent.*, vol. 81, Nov. 2021, Art. no. 103373.
[285] Y. Cai, C. Li, S. Wang, and J. Cheng, "DeLTR: A deep learning based approach to traffic light recognition," in *Proc. Int. Conf. Image Graph.* Cham, Switzerland: Springer, 2019, pp. 604–615.
[286] D. He, Y. Qiu, J. Miao, Z. Zou, K. Li, C. Ren, and G. Shen, "Improved mask R-CNN for obstacle detection of rail transit," *Measurement*, vol. 190, Feb. 2022, Art. no. 110728.
[287] G. Armeniakos, G. Zervakis, D. Soudris, and J. Henkel, "Hardware approximate techniques for deep neural network accelerators: A survey," *ACM Comput. Surveys*, vol. 55, no. 4, pp. 1–36, May 2023.
[288] A. Asad, R. Kaur, and F. Mohammadi, "A survey on memory subsystems for deep neural network accelerators," *Future Internet*, vol. 14, no. 5, p. 146, May 2022.
[289] G. Singh, L. Chelini, S. Corda, A. J. Awan, S. Stuijk, R. Jordans, H. Corporaal, and A.-J. Boonstra, "Near-memory computing: Past, present, and future," *Microprocessors Microsyst.*, vol. 71, Nov. 2019, Art. no. 102868.
[290] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, "An open approach to autonomous vehicles," *IEEE Micro*, vol. 35, no. 6, pp. 60–68, Nov./Dec. 2015.
[291] V. Y. Gudur, S. Maheshwari, S. Bhardwaj, A. Acharyya, and R. Shafik, "Hardware-algorithm codesign for fast and energy efficient approximate string matching on FPGA for computational biology," in *Proc. 44th Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. (EMBC)*, Jul. 2022, pp. 87–90.
[292] H. Fan, F. Zhu, C. Liu, L. Zhang, L. Zhuang, D. Li, W. Zhu, J. Hu, H. Li, and Q. Kong, "Baidu Apollo EM motion planner," 2018, *arXiv:1807.08048*.
[293] CommaAI. *Openpilot*. Accessed: Oct. 6, 2022. [Online]. Available: https://github.com/commaai/openpilot
[294] L. Chen, T. Tang, Z. Cai, Y. Li, P. Wu, H. Li, J. Shi, J. Yan, and Y. Qiao, "Level 2 autonomous driving on a single device: Diving into the devils of openpilot," 2022, *arXiv:2206.08176*.
[295] Tesla. *Autopilot*. Accessed: Oct. 6, 2022. [Online]. Available: https://www.tesla.com/autopilot
[296] Xpeng. *Xpilot*. Accessed: Oct. 6, 2022. [Online]. Available: https://en.xiaopeng.com/g3/xpilot.html
[297] NVIDIA. *Driveworks SDK*. Accessed: Oct. 6, 2022. [Online]. Available: https://developer.nvidia.com/driveworks
[298] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, M. Hasan, B. C. Van Essen, A. A. S. Awwal, and V. K. Asari, "A state-of-the-art survey on deep learning theory and architectures," *Electron*, vol. 8, no. 3, p. 292, Mar. 2019. [Online]. Available: https://www.mdpi.com/2079-9292/8/3/292
[299] V. Tankovich, C. Hane, Y. Zhang, A. Kowdle, S. Fanello, and S. Bouaziz, "HITNet: Hierarchical iterative tile refinement network for real-time stereo matching," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 14362–14372.
[300] I. Krešo, J. Krapac, and S. Šegvić, "Efficient ladder-style DenseNets for semantic segmentation of large images," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 8, pp. 4951–4961, Aug. 2021.
[301] H. H. Aghdam, E. J. Heravi, and D. Puig, "A practical approach for detection and classification of traffic signs using convolutional neural networks," *Robot. Auton. Syst.*, vol. 84, pp. 97–112, Oct. 2016.
[302] Q. Wang, Q. Zhang, X. Liang, Y. Wang, C. Zhou, and V. I. Mikulovich, "Traffic lights detection and recognition method based on the improved YOLOv4 algorithm," *Sensors*, vol. 22, no. 1, p. 200, Dec. 2021.
[303] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "YOLOX: Exceeding Yolo series in 2021," 2021, *arXiv:2107.08430*.
[304] W. Liu, S. Liao, W. Hu, X. Liang, and X. Chen, "Learning efficient single-stage pedestrian detectors by asymptotic localization fitting," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 618–634.
[305] J. Fowers, G. Brown, P. Cooke, and G. Stitt, "A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays (FPGA)*, pp. 47–56, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2145694.2145704
[306] J. Cong, Z. Fang, Y. Hao, P. Wei, C. Hao Yu, C. Zhang, and P. Zhou, "Best-effort FPGA programming: A few steps can go a long way," 2018, *arXiv:1807.01340*.
[307] K. He, B. Liu, Y. Zhang, A. Ling, and D. Gu, "FeCaffe: FPGA-enabled caffe with OpenCL for deep learning training and inference on Intel stratix 10," 2019, *arXiv:1911.08905*.

[308] W. Shi, M. B. Alawieh, X. Li, and H. Yu, "Algorithm and hardware implementation for visual perception system in autonomous vehicle: A survey," *Integration*, vol. 59, pp. 148–156, Sep. 2017, doi: 10.1016/j.vlsi.2017.07.007.

[309] J. Wang and S. Gu, "FPGA implementation of object detection accelerator based on vitis-AI," in *Proc. 11th Int. Conf. Inf. Sci. Technol. (ICIST)*, May 2021, pp. 571–577.

[310] Y. Wang, C. Ding, Z. Li, G. Yuan, S. Liao, X. Ma, B. Yuan, X. Qian, J. Tang, Q. Qiu, and X. Lin, "Towards ultra-high performance and energy efficiency of deep learning systems: An algorithm-hardware co-optimization framework," in *Proc. AAAI Conf. Artif. Intell.*, vol. 32, no. 1, 2018, pp. 1–9.

[311] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, "The architectural implications of autonomous driving: Constraints and acceleration," in *Proc. 23rd Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2018, pp. 751–766.

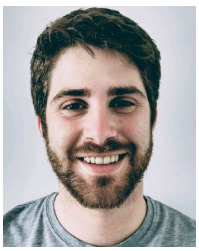
[312] D. R. Ditzel, "Accelerating ML recommendation with over 1,000 RISC-V/tensor processors on esperanto's ET-SoC-1 chip," *IEEE Micro*, vol. 42, no. 3, pp. 31–38, May 2022.

[313] F. Tu, Y. Wang, Z. Wu, L. Liang, Y. Ding, B. Kim, L. Liu, S. Wei, Y. Xie, and S. Yin, "A 28 nm 29.2TFLOPS/W BF16 and 36.5TOPS/W INT8 reconfigurable digital CIM processor with unified FP/INT pipeline and bitwise in-memory booth multiplication for cloud deep learning acceleration," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, vol. 65, Feb. 2022, pp. 1–3.

[314] B. Keller, R. Venkatesan, S. Dai, S. G. Tell, B. Zimmer, W. J. Dally, C. T. Gray, and B. Khailany, "A 17–95.6 TOPS/W deep learning inference accelerator with per-vector scaled 4-bit quantization for transformers in 5 nm," in *Proc. IEEE Symp. VLSI Technol. Circuits (VLSI Technol. Circuits)*, Jun. 2022, pp. 16–17.

DAVID CASTELLS-RUFAS received the B.S., M.S., and Ph.D. degrees in computer science from the Universitat Autònoma de Barcelona (UAB), Spain, in 1994, 2009, and 2016, respectively. From 2003 to 2018, he was worked as a Research Assistant and a Postdoctoral Researcher with the CEPHIS/CAIAC Research Center, UAB, where he was an Adjunct Professor. From 2020 to 2022, he was with the Computer Architecture and Operating Systems Department, UAB, and the Microelectronic and Electronic Systems Department, UAB. Currently, he is with the Barcelona Supercomputing Center (BSC). He founded the technology-based companies Histeresys and Creanium, in 1998 and 2001, respectively. His research interests include high performance computing, reconfigurable systems, and embedded systems.

VINH NGO was born in Thua Thien Hue, Vietnam, in 1983. He received the bachelor's and master's degrees in electrical—electronics from Bach Khoa University, Vietnam, in 2006 and 2008, respectively, and the Ph.D. degree in electronic and telecommunication engineering from the Universitat Autònoma de Barcelona, in 2022. His research interests include chip multiprocessor architectures, high performance microprocessor designs, FPGA-based image processing accelerators, and DRAM memory controller designs.

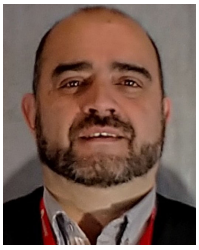


JUAN BORREGO-CARAZO was born in Soria, Spain, in 1993. He received the Graduate degree in physics from the Universitat Autònoma de Barcelona (UAB), Spain, in 2016, and the M.Sc. degree in data science from the University of Barcelona (UB). He is currently pursuing the joint Ph.D. (Industrial) degree with UAB and KOSTAL Eléctrica, S.A.-based in embedding neural networks in resource-constrained devices for gesture recognition. His research interests include deep learning and neural networks, computer vision, and machine learning.

MARC CODINA was born in Barcelona, Spain, in 1983. He received the Graduate degree in computer engineering from the Universitat Autònoma de Barcelona (UAB) and the master's degree in mobile application development from the Universitat Oberta de Catalunya (UOC). He is currently pursuing the Ph.D. degree in electronic and telecommunication engineering with UAB. His research interests include the IoT, embedded and mobile platform technologies, and machine learning.

CARLES SANCHEZ received the Ph.D. degree in computer science. He is currently a Professor at the Computer Science Department, UAB, and a member of the Interactive and Augmented Modelling (IAM) (iam.cvc.uab.es) Research Group, Computer Vision Center (CVC). He has been working in research projects (two as IP and participated in 18) and technological transfer for more than ten years. He has long experience in medical imaging, anatomy modeling, and image processing applied in the development of intelligent radiomic medical support systems. He has coauthored more than 15 papers in journals and over 25 international conferences.

DEBORA GIL received the Ph.D. degree in mathematics from the Computer Science Department, UAB. She is currently a Professor with the Computer Science Department, UAB, and in charge of the Interactive and Augmented Modeling (IAM) (iam.cvc.uab.es) Research Group, Computer Vision Center (CVC). She has a wide multidisciplinary experience and is an expert in mathematical and statistical modeling of heterogeneous data in clinical (diagnosis and intervention) decision support systems. She has coauthored more than 60 articles in journals (41 indexed in JCR) and over 100 international conferences. She has been an IP of several national (seven competitive and five with private companies) and international (the WP Leader in E-pilots, H2020-CS2-CFP08-2018-01, and a Coordinator of H2020 Topiomics-ATTRACT).



JORDI CARRABINA was born in Manresa, Catalonia, in 1963. He received the Graduate degree in physics and the M.S. and Ph.D. degrees in computer science from the Universitat Autònoma de Barcelona (UAB), Catalonia, Spain, in 1986, 1988, and 1991, respectively. In 1986, he joined the National Center for Microelectronics (CNM-CSIC), where he was collaborating, until 1996. Since 1990, he has been an Associate Professor with the Department of Computer Science, UAB. In 2005, he joined the New Microelectronics and Electronic Systems Department, heading the CEPHIS Research Group, recognized as the TECNIO Innovation Technology Center, Catalan Government Agency ACCIO. He is currently a Professor at UAB. He is also teaching in the B.S. and M.Sc. degrees in telecommunications engineering and computer engineering at UAB, the master's degree in embedded systems at UPV-EHU, and coordinating the MsC on IoT for eHealth. His research interests include microelectronic systems oriented to embedded platforms, SoC/NoC architectures, and printed microelectronics.