IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received January 28, 2020, accepted February 23, 2020, date of publication February 27, 2020, date of current version March 9, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2976513

Resource-Constrained Machine Learning for ADAS: A Systematic Review

JUAN BORREGO-CARAZO[1,2], **DAVID CASTELLS-RUFAS**[2], **ERNESTO BIEMPICA**[1], **AND JORDI CARRABINA**[2]
[1]R+D, Kostal Eléctrica, S.A., 08181 Barcelona, Spain
[2]Microelectronics and Electronic Systems Department, Universitat Autònoma de Barcelona, 08193 Cerdanyola del Vallès, Spain

Corresponding author: Juan Borrego-Carazo (juan.borrego@uab.cat)

This work was partly supported by the Spanish Ministry of Science, Innovation and Universities under grant RTI2018-095209-B-C22, and by the Catalan Government under grants 2017SGR1624 and 2018DI30.

ABSTRACT The advent of machine learning (ML) methods for the industry has opened new possibilities in the automotive domain, especially for Advanced Driver Assistance Systems (ADAS). These methods mainly focus on specific problems ranging from traffic sign and light recognition to pedestrian detection. In most cases, the computational resources and power budget found in ADAS systems are constrained while most machine learning methods are computationally intensive. The usual solution consists in adapting the ML models to comply with the memory and real-time (RT) requirements for inference. Some models are easily adapted to resource-constrained hardware, such as Support Vector Machines, while others, like Neural Networks, need more complex processes to fit into the desired hardware. The ADAS hardware (HW platforms) are diverse, from complex MPSoC CPUs down to classical MCUs, DPSs and application-specific FPGAs and ASICs or specific GPU platforms (such as the NVIDIA families Tegra or Jetson). Therefore, there is a tradeoff between the complexity of the ML model implemented and the selected platform that impacts the performance metrics: function results, energy consumption and speed (latency and throughput). In this paper, a survey in the form of systematic review is conducted to analyze the scope of the published research works that embed ML models into resource-constrained implementations for ADAS applications and what are the achievements regarding the ML performance, energy and speed trade-off.

INDEX TERMS Machine learning, embedded software, automotive engineering, GPU, FPGA, ADAS.

I. INTRODUCTION

Safety is increasingly important for drivers and other users of the driving environment such as pedestrians, cyclists, bikers, scooters, etc. Only in 2016, road accidents in the European Union summed up to 25.600 fatalities and 1.4 million people injured [1]. Many measures are being taken by public institutions (improving road quality, enhancing driver consciousness and driving regulations, etc.) to improve these numbers. However, the cause of most of these accidents resides in human errors or distractions. For this reason, automotive firms are pushing up research on improving safety. The cornerstone of these advances are Advanced Driver Assistance Systems (ADAS).

ADAS systems consist in several sensors, processing and actuators components that help drivers to avoid accidents and to drive carefully. Examples of ADAS tasks or systems are Adaptive Cruise Control (ACC), Lane Departure Warning System (LDW) or Intelligent Speed Adaptation (ISA), among others.

In the case of sensing, several sensors such as Lidar, infrared (IR) and visible cameras, are used to collect the required data for each tasks. The next step is to process all that information and compute the desired outputs so that actuators can apply the required commands.

Traditionally, ADAS have been implemented on application-specific complex ad-hoc solutions built for the automotive domain since they suppose a large market. The corresponding algorithms need to be adapted explicitly to every task and the processing pipelines become complex. For example, in the case of Traffic Sign Recognition (TSR) systems, the first step was obtaining images of the traffic scenes from cameras. Then, applying models based on template matching, contour detection or color thresholding along

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

with other processing steps, to determine the desired regions of interest (ROI). Follows the segmentation and classification of every sign and the corresponding actuation. This problem is even more complex if you consider different sign models at different countries.

In previous years, the application of machine learning algorithms to ADAS tasks began to grow as a significant trend in the automotive sector. Machine Learning (ML) and deep learning (DL) models, as for example Artificial Neural Networks (ANN) [2] or the more advanced Convolutional Neural Networks (CNNs) [3], have proven to be an extraordinary solution for such complex tasks. These models have been already tested in ADAS systems for tasks such as vehicle detection [4]. Support Vector Machines (SVMs) are one of the most used methods when classifying complex information since they provide good performance and can be implemented as a lightweight model in resource-constrained platforms (for the inference phase after its training off-line).

In this paper, we refer as performance the metric related to machine learning outcomes for ML tasks (accuracy, precision, etc.). The HW/SW implementation performance will be referred as either speed (latency, throughput, etc.) or energy consumption.

Machine Learning (or Deep Learning) models are computationally intensive (more for the training than for the inference phases) while the processing elements used by the automotive sector are often cost-Resource-Constrained (i.e. microcontroller units or MCUs). Due to the trade-off between performance and resource burdening, there are different options for embedding such complex and heavy (in terms of memory) models on different computing platforms used for different car platforms (segmented according to their costs).

The new ML solutions focus their attention in three options: (1) model adaptation for resource-constrained environments; (2) hardware implementation or acceleration procedures and results; and (3) the use of more powerful hardware. The first case includes model adaptation there are three trends: adapting operations [5], adapting numeric computations and data types [6] and adapting models for scarce resource environments ([7], [8]). The option of HW implementation refers to mapping NNs on programmable processors or ad-hoc hardware such as FPGA or neuromorphic processors together with the explicit memory mappings for efficient resource management ([9], [10]). Finally, the use of more powerful hardware is based in the recent development of embedded platforms with dedicated GPUs, especially devoted to automotive tasks, which accelerate inference of deep learning models. This is the case of NVIDIA Jetson and Tegra families being adopted for the embedded and automotive domain.

The purpose of this paper is to review the research work whose focus is the use of machine learning models for ADAS tasks and their deployments on the resource-constrained embedded platforms addressed to the automotive domain. In order to reach a fair classification of this research, our study is carried out as a systematic review.

II. RELATED WORK

In the past, there have been reviews and surveys devoted to specific elements and problems of the automotive sector. Self-driving cars [11], crowd information and traffic management [12] and intra-vehicular communications [13] are examples found in the vast literature devoted to automotive topics. There are also reviews that focus on ADAS systems such as [14]–[17]. However, these reviews usually focus their attention on topics such as the type of the sensing technology used for each task or what new perspectives are projected to the ADAS sector.

There are also reviews devoted to specific ADAS tasks or subtasks with the corresponding models. For example, the review [18] is devoted to pedestrian detection systems that use far-infrared video as sensing technology. In [19], authors compare different types of sensing and classification models, SVM and Adaboost for vehicle detection. In [20], authors perform a comparative study among different techniques devoted to traffic light and sign recognition and the properties and robustness of each technique or procedure. There are also reviews devoted to specific and complete ADAS tasks: [21] is devoted to Adaptive Cruise Control systems (and the whole system implementation).

Obviously, there are also reviews on machine learning methods ([22], [23]). However, only one review has been found in the intersection of machine learning and ADAS systems. In [24], authors compare different methods and how the data pipeline is generally built when applying machine learning models to ADAS tasks.

As a summary, there has been a lot of work devoted to reviewing, summarizing, and comparing ADAS systems and related research. However, at the best knowledge of the authors, there in not any study covering the three main topics at once: advanced driver assistance systems (ADAS), machine learning models and their implementation on embedded devices.

III. METHODOLOGY

Systematic reviews imply following a strict methodology. This section describes the main components of the review process. First, the definition of the review question and the inclusion/exclusion criteria. Second, the description of the search and screening methodologies.

A. QUESTION

Systematic reviews are quite popular in the medical domain. They start from the main question of the review according to the PICOS eligibility criteria [25]. The PICOS format stands for patient, intervention, comparison, outcomes and study design. In our case, we propose to change patient by object. Thus, the format used is OICOS: object, intervention, comparison, outcomes and study design. The question should follow the specification of each component in order to be clear and concise.

In the present case, specifications are:

- **Object**: A machine or deep learning model, i.e. a model that can be trained, devoted to an ADAS task.
- **Intervention**: embedding the model into a Resource-Constrained platform, by means of model or hardware modifications.
- **Comparison**: performance of the model with other implementations devoted to the same task.
- **Outcome**: performance metric of the model, energy efficiency and speed of model execution (inference), if available.
- **Study design**: experimental design and results.

Hence, the main question of the study is established as: **How a Machine Learning model can be implemented into an embedded or Resource-Constrained platform devoted to ADAS tasks?**From this question, several others stem off:

- Which is the embedded platform?
- Which ML model was used to assess the ADAS task?
- Has the model been modified before embedding it?
- Are both training and inference performed in the embedded platform or only inference?
- Are some models better suited for ADAS tasks? And for Resource-Constrained tasks?

B. ELIGIBILITY CRITERIA

The question itself and the PICOS format leads to the establishment of the inclusion and exclusion criteria.

1) INCLUSION CRITERIA

There are three main requirements to include in the review an article collected from queries on common databases: it has to contain a ML model, it has to execute in an embedded or Resource-Constrained platform and it has to implement an ADAS task.

A machine learning model means that it has been trained regarding a given dataset. The model should be clearly stated and defined. With regard to the execution, the model can be modified, optimized or transformed for its mapping to an embedded platform. Our requirement is that the model should be placed or run in a Resource-Constrained or embedded platform by means of some modifications either in the model or in the hardware. Research works without any modification done on the ML model for its implementation on a resource-constrained platform will also be included in the review. We can exceptionally include studies consisting in the formulation of some modification of ML models or hardware that can be useful but that were not explicitly used for any ADAS task.

We also require the ML model to be tested. Hence, a measure of the obtained performance has to be included in the publication together with the metric selected and the dataset used. Obviously, the implementation has to focus into any specific ADAS task. The ADAS tasks considered in this review are the classified as:

- Vehicle and pedestrian detection
- Driver's state, behavior and identification
- Traffic sign recognition
- Road detection and scene understanding
- Miscellaneous

2) EXCLUSION CRITERIA

There are certain characteristics that will dismiss an article from being accepted in the review. Regarding the model, if the research paper does not include any training or learning, it will not be included. Also, if the overall implementation is carried out in a personal computer or general-purpose machine, the article will be also excluded. If the study lacks any performance test (with the corresponding metric) it will be also rejected.

Considering the topic, exclusion criteria consist in the following list: information management, vehicular ad-hoc networks (VANETs), traffic surveillance, stealing avoidance systems, battery State-of-Charge (SoC) and State-of-Health (SoH). Those studies are not included in our review.

Studies that consist in general tasks such as object detection, that could be implemented for ADAS but which are not explicitly devoted to ADAS will also be dismissed unless they provide a useful improvement applicable to the field. Of course, if it the access to the full text of an article is not possible, it will be excluded from the review. There is not any exclusion criteria regarding the format of publication or the date of publication.

C. SEARCH STRATEGY

The search is carried out by the two principal researchers. Each researcher is in charge of a group of databases or sources of information. The search is performed with the resources made available by the Autonomous University of Barcelona. The first step of the search is the specification of the databases used. As the research is focused in computing and electronics the specific databases are chosen accordingly. The databases used for the search are:

- Multidisciplinary databases:
  - Web of Science
  - Scopus
  - Arxiv
  - Springer On-line
- Specific databases:
  - IEEE Xplore Digital Library
  - ACM Digital Library

For the searches, we use the product set of the keywords in the next two groups in every search engine:

- **Machine learning keywords**: [deep learning, neural networks, RNN, CNN, regression, machine learning, extreme learning, ensemble methods, reinforcement, meta-learning, decision tree, random forest, gradient boost, LSTM, SVM, support vector machine, GRU,

autoencoder, Hopfield, Boltzmann machine, deep belief, Markov, autodiff, recurrent, convolutional]
- **Hardware keywords**: [embedded, SoC, MCU, CPU, microcontroller, chip, processor, wearable, low power, fixed point, device, FPGA, field programmable gated array, FPU, GPU]
- **ADAS keywords**: [vehic∗ OR automo∗ OR car OR ADAS OR "advanced driver assistance system" OR driver]

All searches are carried out regarding title, abstract and keywords in conjunction. If this combination is not possible, the title is used for the search. The main reason for this is the noise avoidance in the search hits. When possible, the boolean AND and OR are used to group searches. The searches were conducted between November 2018 and December 2019. Hence, articles published beyond 01-01-2019, are out of the scope. Any article with a posterior published date has been manually included and for reference purposes.

D. SCREENING STRATEGY

We include in the review all papers that fulfill the inclusion criteria and can be downloaded. The screening methodology is divided in two phases: title/abstract and full text. The first inclusion/rejection phase is carried out taking into account title, abstract and conclusions. It can happen, that the title or the abstract are not sufficiently specific about the content. Hence, if there is any possibility that it can meet the inclusion criteria the corresponding article is included due to the need of not excluding any relevant study.

Once all the articles have passed the first screening, the next one consists of the full text reading. The desired information is retrieved on all papers accepted according to a specific structure. The fields contained in this retrieval are:

- Task
- Embedded or Resource-Constrained platform
- ML/DL algorithm, library and embedding procedure
- Data pipeline
- Dataset characteristics and object of classification/regression
- Task performance, speed and energy

IV. RESULTS

As stated before, the screening process consists in several steps of selection of articles. The results of the different steps are: articles hit in the search, number of accessed articles, deleted duplicates, screen by title, abstract and conclusions. The two final steps of the process are screen by full text reading and, finally, summarizing. Table 1 shows the number of articles associated with each database and the finally accessed number. In the case of Table 2, it shows the number of articles associated with every step of the systematic review and selection procedure.

After the search, screening and selection steps, selected articles and conclusions read and summarized. This section shows and analyzes the published results according to the tasks classification stated in section III.B.1.

TABLE 1. Results of the search in terms of the number of papers and the papers accessed for each database.

Database	No articles hit	No articles accessed
IEEE Xplore	811	805
Arxiv	156	156
ACM Digital Library	109	99
DART	103	78
OATD	31	26
OpenGrey	3	1
Google Scholar	65	35
Scopus	793	399
Springer	280	221
TESEO	71	4
Web of Science	15	15
TOTAL	2437	1839

TABLE 2. Number of articles after each step of the systematic selection and review procedure.

Articles Hit	Accessed	Delete Duplicates	1st Screening	2nd Screening
2437	1839	1561	450	65

A. VEHICLE AND PEDESTRIAN DETECTION

Pedestrian or vehicle detection implementations on embedded devices using machine learning usually show a very structured design: preprocessing, feature extraction and classification. Usually, preprocessing starts with binarization (thresholding), selection of areas-of-interest (AoI) of the image or an image pyramid scheme. After that, feature extraction uses algorithms such as HOG, CoHOG or Haar. Finally, features allow classifying regions with one machine learning classifier, typically SVM. These kinds of systems have an advantage: their easy deployment on embedded platforms since these classifiers require little memory and short time, and preprocessing tasks can easily be implemented on hardware.

With the advent of neural networks and more specifically convolutional networks ([26], [27]), this situation has changed. The performance shown by NNs (compared to traditional pipelines) pushed developers to try to embed them in resource-constrained devices. The problem resides in the fact that those networks are extremely heavier in terms of memory and resource usage. Hence, its embedding is difficult and requires applying several strategies. Among them stand out pruning, quantization and layer fusion. However, even with these techniques, pipelines and required structures are still too heavy to attain real-time performance for functions such as detection and classification of pedestrians and road objects. In this sense, one problem that the systems devoted to object detection and classification usually show is their need to implement one pipeline per task.

Specific networks, such as YOLO [28] were developed to solve that issue. They are usually implemented on GPU-based embedded boards to speed-up inference time. This often incurs in a not optimal energy profile. As a result, developers have also tried to implement those systems in ad-hoc hardware (ASIC or FPGA) which show a more suitable energy profile.

1) ALGORITHM STRUCTURE

The classical approach is to use HOG and SVM for detecting objects together with some previous preprocessing step. In [29], they apply a pyramid of different scale images and create a HOG descriptor vector with histogram normalization by rolling a fixed size window through the image. Finally, they feed these different vectors to the SVM. They report an increase from 16% to 40.1% from changing the number of scales from 1 to 44 (scale factor 1.05). Another example of a pyramid of images is found in [30]. Authors apply region selection to each of the images in the pyramid by selecting central regions related to vanishing point. Feature extraction uses CoHOG. This technique relies on the co-occurrences of gradients in the window and among different images in the pyramid. This boosts the robustness of detection compared to HOG.

Another way of creating features for further classification is shown in [31]. Using a sliding window approach, authors obtain feature vectors, through a covariance matrix decomposition, which are then forwarded to a linear SVM.

Usually both HOG and feature extractor plus SVM can be extended by adding tracking after classification or feature extraction improvements. In [30], authors improve the performance by adding tracking using Kalman filtering for the prediction of the next position. An extended version of the HOG plus SVM pipeline is found in [32] whose authors add detection grouping to delete overlapped windows that are detecting the same pedestrian, and detection tracking to keep detections among frames.

Haar wavelet is another alternative for generating the features used for classification. In [33], authors show an early implementation of vehicle detection and distance estimation for a Forward Collision Warning system. To detect a vehicle, they firstly generate 3 ROIs from different overlapping parts of the image. Then, they apply Sobel edge detector for detecting the vehicle. Finally, a quantized version of Haar wavelet features are extracted from the ROI where the vehicle is detected. Then, these features are fed to a SVM. To enhance detection among frames, they apply tracking through Motion Vector estimation [34] and they apply SOD to the region where the car was detected.

There are also other classification choices instead of applying SVM. In [35], authors start from a pyramid of images and use Adaboost as detector. After detection, they apply tracking with dynamic modelling, using Haar-like features, and shape and symmetry estimation to improve detection rate.

The use of CNNs for both feature extraction and classification is also common although they impose higher computational demands and is less suitable for resource-constrained applications. Hence, most research works focus their attention on reducing the memory and processing requirements of these networks. In [36], authors use a modified version of SqueezeNet [37] but instead of detecting the pedestrian through a bounding box at pixel level, they detect it through a coarser grid to reduce the computational requirements of the model. The size of the grid is related to the size of the last feature map size of the network. Besides modifying the ground truth, no preprocessing is applied to images in order to improve pedestrian detection.

A good example of memory size reduction is found in [38]. Authors use the DeepPed network [39] based on AlexNet. In order to reduce memory utilization and improve latency, they apply two optimization strategies for the network: scalar quantization (based on k-means clustering) and pruning. They study how these strategies affect both convolutional layers and fully connected layers. They find that pruning affects severely the performance of convolutional layers while quantization does not impose such performance burdens. In the case of fully connected layers, both strategies impose similar performance losses. However, in this last case, the combination of both strategies at the same time yields better results. To evaluate the effect of these optimization procedures in the memory requirements they compare the weight of the initial network, 216.94 MB, with the final compressed one, 3.5 MB, which corresponds to a 61.35x compression factor while only adding 0.4% increase in the Log Average Miss Rate.

Data variety and quantity is usually a difficult point for training neural networks. Several strategies exist for improving the initial situation of any specific dataset. In [40], authors use a SSD network [41], with a Wide Residual (WR) network [42] as a base for detecting pedestrians, vehicles and cyclists from different datasets: KITTI, CBCL Streetscenes and Cityscapes. To obtain more data, they apply Restricted Random Sampling, which consists in sampling crops from original images and resizing them maintaining the image size ratio of objects inside. Experiments show that not varying the shape ratio of objects helps to increase performance. Another point when detecting different types of objects is the different number of occurrences of each class. One solution is to apply balanced loss weighting using the relative proportion of objects. In [40], they applied it and improve mAR by 8%.

One way to ease the deployment of CNN is to reduce computational constraints of the original structure as well as other modifications. In [43], authors focus in the design and deployment of a lightweight CNN for pedestrian and vehicle detection. To do so, they use a SSD network, which has less accuracy than R-CNN or R-FCN but is faster, with a lightweight backbone consisting of a MobilenetV2. To increase accuracy without reducing throughput too much, they apply a series of modifications to the original network. First, they increase resolution to account for pedestrians placed in far frames. Second, the also use depth-wise blocks in the SSD head to reduce complexity. Third, they extend the prediction branch with two more heads to account for low and medium sized objects and, finally, they prune the network via random filter sampling. As an improvement, they show that a two stage training from scratch is possible without incurring into the problems of pre-trained backbones, such as domain misalignment [44]. Finally, they apply tracking through frames to improve detection.

Another way to increase performance is to use an ensemble of models. In certain ADAS applications, it is important to have a reduced false positive rate, as for example in Autonomous Emergency Braking (AEB), where a false positive could induce harmful situations. In [4], authors reduce the false positive rate by adding a DNN as an extra model for classification. The base model is already an ensemble, Adaboost, who receives a ROI from a previous detection pipeline formed by binarization, connected component labelling and integral feature channel. The way the models are added (either voting or bagging) is not clearly explained.

An interesting application of vehicle and pedestrian detection is found in [45]. After being able to detect the object, authors apply first a RNN to predict its future position and velocity, and then a Fuzzy Inference System (FIS) to predict the riskiness of the object. The detection uses input images from a stereo camera to extract and track optical flow features. Semi Global Matching (SGM) is in charge of providing depth information and the ego motion module of adjusting the values with respect to the movement of the driving vehicle. ROIs are generated in small portions on the images and only where objects are detected. Any object detected is characterized by 4 variables $St = x, z, \Delta x, \Delta z$: lateral distance, longitudinal distance and velocities in both dimensions. The collection of these 4 variables for N frames, $\{St, St-N\}$, is the input of a Recurrent Neural Network in charge of predicting Q further steps. The whole collection of previous and predicted values $\{St-N, St+Q\}$ is used to predict the riskiness of the object through a FIS. The FIS has different membership functions for each variable and set of rules devised for moving car and pedestrian risk determination. The system is tested with the KITTI dataset and, for their specific task, they create one novel dataset: risky urban scene stereo (RUSS).

2) EMBEDDING METHOD AND PERFORMANCE

As already mentioned, the main problem when implementing neural networks on embedded devices is the real-time constrain often formulated as trade-off between latency and performance. In [36], authors implement a modified version of SqueezeNet (a CNN which takes advantage of 1×1 convolutions and delayed sampling for reducing computational demands) using the S32V234 [46] automotive processor. To further reduce memory requirements and improve latency, the network uses integers with 8-bit representation and arithmetic except the *tanh* activations, which operate as floats. They achieve at most 30 fps of speed, consuming 2W.

An example of the latency constraints that deep neural networks suffer is found in [38]. Their network consists in an Aggregated Channel Features (ACF) detector, an optimized Alexnet-based network and a SVM. The whole optimized system achieves 61x compression factor and a final weight of 3.5 MB. However, using an NVIDIA Jetson TX1, they only achieve 2.4 fps. To increase that value, they reuse the previous detection results for next frames, reaching 10 fps at maximum. Another example of a SqueezeNet modification is [47]. Its authors convert the detection and classification modules of the original network to a joint classification and detection module named ConvDet. This fully convolutional detection allows for a drastic reduction on the number of parameters and a related increase in the possibilities of region proposal. This advance makes the network weigh less and achieve similar or better performance that other heavier networks such as Faster R-CNN [48]. Compared to YOLO, which weights 735 MB, SqueezeDet weighs, in its minimal version, 7.9 MB.

Another example of CNN optimization is found at [49], where a modified version of a Faster R-CNN is implemented on a NVIDIA Tegra TX1. They use pruning and quantization with two objectives: size reduction and throughput increase. Pruning is applied trough threshold set out from observations of the results on fully connected layers. For quantization, authors take advantage of video instructions of the SIMD architecture selected to reduce network weights from 32-bit real values to 8-bit integers. The overall result shows a decrease from 260 Mbytes to 42 Mbytes in size and a reduction in latency from 508 ms to 327 ms per frame. This allows only 3 fps, showing again the burden that CNNs suffer regarding throughput even when using a dedicated embedded board with GPU support.

Another possibility for increasing throughput is the analysis and profile of its components. Authors in [50] increase the speed of a Fast R-CNN by analyzing which layers consume most of the time. They observe that fully connected layers and batch norm are the most costing layers. Hence, they drop batch normalization layers and reduce the number of neurons of fully connected layers which are not directly related to classification. On a PC, they get a throughput increase from 15 to 25 fps while on a NVIDIA Jetson TX1 they only reach 5 fps. Once again, we observe that CNNs are still too heavy to attain real-time performance.

The deployment of a system into the desired platform for its production use is usually a matter of interest. Several methods can help to achieve this deployment. Apart from modifying the model, neural network adapters and optimizers such as OpenVINO [51] or NVIDIA TensorRT [52] can help adapting the model to hardware specificities. In [43], authors use OpenVINO to accelerate and deploy a lightweight CNN implemented on a general purpose CPU but portable to other platforms.

One of the possibilities to overcome the limitation for CNN embedding is network weights quantization. Among the different quantization types, the most extreme one is binary weight quantization which converts floating-point numbers into binary values, either 0 and 1 or -1 and 1. In [53], authors convert two YOLO CNNs, based on Darknet [54] and MobileNet backbones, to binary-weighted networks.

To reduce the accuracy drop, they use two strategies: wise binarization and taught learning. In the wise binarization stage, they train the network binarizing layers in groups. First, the upper layers, and then lower ones in the network graph. Taught learning helps to reduce the accuracy drop by making the binarized network mimic the features obtained from certain layers of a float-valued teacher network. To be able to account for this mimicking during the network training, they added a new loss term based on the difference of the outputs of the same layer in both teacher and student networks. This strategy is possible as long as both networks share the corresponding layer, even if the overall network is not the same. They achieve a reduction from 257 MB to 8.8 MB from Darknet-YOLO with only 2% drop in accuracy.

A more feasible choice among machine learning algorithms for hardware embedding is SVM. An early example of SVM optimization can be found in [55] where authors reformulate SVM to be able to quantize its parameters to reach a feed forward phase with fixed-point values. Fixed-point parameters for inference are saved in LUTs. With 12 bits, they are able to attain same classification error rate as with the floating-point representation version: 5.96%, with the Daimler-Chrysler dataset. Another early example of an embedded implementation of SVM is [33], in which authors implement a Forward Collision system on a dual core DSP processor. In the case of [56], authors implement a pipeline of HOG and SVM in two embedded platforms: a Raspberry Pi 3 and an Odroid C2, obtaining 5 fps and 7 fps respectively for 320 $\times$ 97 pixel images.

There are also fully hardware implementations. In [29], authors optimize a system of image pyramid, followed by HOG and SVM. The main contributions are efficient scale selection and algorithm optimization, parallelism of detectors and image preprocessing. In the HOG step, they bin the gradients without computing the actual angle and use L1 norms instead of L2 for magnitude computation. The histogram normalization is carried with L2 norm and fixed-point dividers (due to 5% drop in performance when using L1). The SVM is trained off-line and its weights are transformed to 4-bit fixed-point numbers. Regarding the scale generation, they choose a scale factor that provides a good trade-off between the number of images to compute and the accuracy performance. They also propose a parallel structure of the different detectors optimized for energy and latency. To reduce memory constraints, they preprocess the image in two ways: first, they quantize pixels to 4-bits and afterwards they use the gradient image instead of the original image to be able to reduce it to 3-bits. They report results of algorithm execution both column- and row-wise. Since classification window is higher than wider, results are far better for column-wise processing. Regarding optimization results, multi-scale boosts accuracy by 2.4x but increases memory usage by 8.8x and energy consumption by 14x. Parallelism of detectors helps reducing energy consumption by 3.4x without affecting accuracy and preprocessing saves 24% energy and 25% memory. All results are obtained on 1080HD videos at 60 fps from the INRIA persons dataset [57] and is implemented using a 45 nm SOI CMOS ASIC technology at a supply voltage of 0.72 V.

Another example of hardware mapping can be found in [31]. Authors compute the covariance matrices of several patches of an image with a FPGA in order to forward feature vectors to a SVM. To obtain the covariance matrices they, compute first and second order integral tensors. For this computation, they profit from the symmetry of the second order integrals to reduce the number of MAC units used, and the parallelism that tensor computation allows to reduce latency. The throughput reached is 132 fps for images of 128 $\times$ 64 pixels. Usually, only linear SVMs are applied due to their simplicity, but in [58] authors apply non-linear SVM with RBF kernel. They map the SVM into a gate-level circuit (with Verilog HDL coding and a 65 nm standard cell library) and they achieve 90 fps on 640 $\times$ 480 pixel images. Another interesting approach applying SVM into hardware is found in [59]. Authors define a training procedure that allows finding an optimal classifier regarding the number of bits used to implement the algorithm in digital hardware.

In [32], we found a good example of algorithm mapping on an heterogeneous platform composed of GPP plus FPGA. HOG and SVM are implemented on the FPGA and the detection and tracking on the ARM processor. To reduce computing time, they implement some optimizations in the detection scheme. For computing HOG norm and orientation, they use the CORDIC algorithm. For computing the SVM prediction vector, they use a LUT. They use 16-bit fixed-point arithmetic for the overall system and they attain 60 fps for 1270 $\times$ 720 pixel images with an energy efficiency of 3.95 GOPS/W A surprising implementation is found in [45], where a neuro-fuzzy inference engine is mapped onto an application-specific complex hardware architecture comprising both analog and digital processing. The system contains a matrix processing unit with a parallel SIMD system (with a flexible numerical representation) to detect objects and provide the level of risk for the car and the driver. The pipeline consists of a semi-global matching processor (SGMP) that obtains a real-time depth map and provides distances to objects. Next, a vector of distances and derivatives of detected objects enters a Recurrent Neural Network. The RNN and the Fuzzy Inference system (FIS) are mapped together onto an analog core implementing the intention-prediction processor (IPP). Weight multiplication is performed by current-steering multiplying-DACs. The activation function is realized with a simple sigmoidal amplifier and the outputs of the network are fed into a transconductance Membership Function (MF) in charge of the intention prediction evaluation. The MF changes its shape to adapt to various risky conditions with 5 control parameters. The digital part performs on-line learning. The overall system has two operation modes: (1) high performance and (2) low power. The SoC is implemented using a 65nm CMOS technology, operates at 30 fps on 720p videos and consumes energy ranging from 0.984 mW to 330 mW.

Another detection SoC implemented for ADAS is [35]. Their authors stablish an Adaboost based detection mechanism with tracking that achieves 60 fps when working at 140 meters (active distance) and 300 fps when working with 30m. The energy consumption of the chip is around 69mW on average with a power efficiency of 3.01TOPS/W.

The language in which NN are described is usually not suitable for its direct mapping onto the target platform. In [62], authors port Yolo and TinyYolo networks to OpenCL (only for inference). They replace kernel implementations such as general matrix-matrix multiplication (GEMM) and simple arithmetic operations (e.g. scaling). Other layers are also replaced by OpenCL homologue implementations. To analyze their results they compare CUDA and OpenCL implementations obtained mapping on different platforms: cuDNN with CUDA on a GTX 1050 and Darknet with OpenCL on AMD R7 APU and AMD Radeon RX 560. The results show the need for a better optimization of the deep learning network for OpenCL. OpenCL implementation for both hardware platforms was 6.4x times slower for TinyYOLO, being the computational part of the network the main responsible of the performance loss.

FPGAs are considered more efficient regarding energy consumption than GP-GPU systems. However, depending on the task and the inference engine, this advantage (and others like throughput) is not fulfilled. In [60], authors compare a NVIDIA Tegra X2 board with a Xilinx Ultrascale regarding throughput and energy consumption. In the case of a complex task such as detection, both platforms show similar results in both parameters. However, in this study, the FPGA platform is not optimized and its performance severely varies for different engines such as PipeCNN [65] or xfDNN [66]. The main problem is that the High Level Synthesis (HLS) tool used for porting CNNs to FPGA platforms is still not as efficient as traditional design. In the case of a simpler task such as classification, a huge difference regarding throughput is observed. The GP-GPU platform achieves around 600 fps while the FPGA achieves only 60 FPS. It must be noted that the FPGA system is not fully optimized for the algorithm. One important point regarding the GP-GPU system in the classification task is that throughput increases proportionally to CPU clock (but not to GPU clock) since the application is memory intensive. Meanwhile, this burden is not observed in the detection task, where throughput also increases with the GPU frequency.

To exploit FPGAs performance, the system has to be well optimized. The authors of [61] carry out a series of simulations for optimizing energy performance of the FPGA for HOG and Adaboost algorithms. They obtain a simulated improvement ranging from 376x to 1896x. They implement approximate computing and quantization in HOG and a Processing in Memory (PIM) hardware accelerator for Haar detector. For Adaboost, they arrange the learning steps in memory blocks. As noted in the case of a GP-GPU platform [60], memory access can become burden for throughput and energy efficiency.

3) PARTIAL CONCLUSIONS

The 23 articles reviewed in this section are classified in two groups regarding their algorithm: classical, which use preprocessing, feature extraction and classification, and NN-based. Algorithms using neural networks shown better performance. However, there are problems in their porting to embedded platforms concerning real-time performance. From the implementation point of view, there are two main options for vehicle and pedestrian detection: software and hardware. While the preprocessing and feature extraction parts are prone to be accelerated using application-specific hardware, classification is usually devised in software. On the other hand, hardware mappings of CNNs is a hot topic both on ASICs and FPGAs. Detection of different objects is a difficult task since it implies detecting first the objects and then classifying them. For this reason, CNNs suffer from higher latency although improvements have been made when using networks such as YOLO. In the other hand, classical classification algorithms, which have a lower memory footprint, together with tailored preprocessing algorithms, show lower latency but also lower performance in terms of accuracy.

A summary of the collected and selected articles is shown in Tables 3 and 4. The information collected shows key factors affecting performance: algorithms and libraries, dataset, embedding strategies, and hardware platforms together with the obtained results regarding performance in the specific task.

B. DRIVER'S STATE, BEHAVIOR AND IDENTIFICATION

Distracted driving was responsible of 9.9% of fatalities in 2015 in the U.S.A. ([67], [68]). Hence, being able to correct driver distractions or alerting the tired driver is of utmost importance to maintain safety in transportation. For this reason, there has been a strong research trying to solve the problem in a safe, non-intrusive and efficient way.

Driver inattention can be divided in two categories: distraction and fatigue. Distraction can be further divided into visual and cognitive distraction. These distinctions are important since each type of lack of awareness requires different types of information to be processed. Following this reasoning, driver distractions can be measured by centering the attention at the driver, at the car or at both. Fatigue, a priori, can only be analyzed regarding the driver. Hence, the first difficulty when trying to determine the level of distraction or fatigue of a driver is the source of information.

Some studies center the analysis of driver's state in physiological signals such as EEG, ECG, SRG or EMG [69]. Retrieving these signals require using sensors that can provide good relation with regards to alerts. However, their degree of intrusiveness makes some of the proposals impractical for real daily life situations. One alternative is to place sensors out of the human body like for instance in the steering wheel, as in [70]. In that case, authors use it for user identification, but it can also be used for driver's state measurement with some limitations. Only the hands are exposed

TABLE 3. Algorithm and data information for vehicle and pedestrian detection articles: task, classification, dataset, embedding and performance. / is used when the field is not given.

Article	Task	Classification Model	Dataset	Embedding	Performance
[38]	Pedestrian detection	CNN	Caltech pedestrian	Quantization (k-means) & pruning	61.35x compression factor & -0.4% Log Average Miss Rate
[60]	Detection	CNN	/	Hardware mapping	/
[61]	Detection	Adaboost	UIUC car detection dataset & INRIA image datasets	Processing in memory	2654x speedup
[30]	Pedestrian detection	SVM	/	/	93.2 % TPR
[56]	Vehicle Detection	SVM	KITTI	/	70 % accuracy
[49]	Vehicle detection	CNN	/	Quantization & Pruning	83.6 % accuracy
[40]	Vehicle, pedestrian, cyclist detection	CNN	KITTI & CBCL & Cityscapes	/	68.1 % mAP
[45]	Pedestrian detection	RNN-FIS	KITTI	Hardware mapping	98.1 %
[43]	Detection	CNN	COCO & Imagenet	Pruning	90.6% AP
[53]	Detection	CNN (BWN)	KITTI	Binarization	76 % mAP
[62]	Detection	CNN (YOLO)	PASCAL VOC 2007&2012	/	78.6 & 57.1 % mAP
[35]	Detection	Adaboost	/	/	>90 % detection rate
[33]	Detection	SVM	/	/	/
[47]	Detection	CNN	KITTI	ConvDet module	76.7 mAP
[36]	Pedestrian Detection	CNN	Caltech pedestrian	8-bit quantization	/
[29]	Pedestrian Detection	SVM	INRIA person	Hardware	39.1 % AP
[4]	Vehicle detection	Adaboost & DNN	/	/	>90% precision
[58]	Detection	SVM	/	Hardware	/
[31]	Pedestrian detection	SVM	INRIA person	Hardware	10% miss rate at 10^{-3} False Positives per Window (FPPW)
[63]	Pedestrian detection	SVM	Daimler-Chrysler	Hardware	5.96% error rate
[32]	Pedestrian Detection	SVM	Caltech Pedestrian Dataset	Hardware	/
[50]	Vehicle Detection	CNN (FAST R-CNN)	ETH [64]	Pruning	>90% accuracy
[55]	Pedestrian detection	SVM	Daimler-Chrysler	/	5.96% Error rate

and hence only signals coming from them can be analyzed. Another approach centered at the driver is to infer its state from cameras inside the car that can determine head pose, gaze and other features. However, this methodology has huge dependence on recording conditions such as lighting or sun glasses and its predictions are not robust in these contexts. To solve this problem, an option is to use other types of cameras such as depth or infrared cameras. Apart from body signals and cameras, there are other options to sense the driver state. In [71], 5 inattentive driving activities are sensed through Doppler shifts by using microphones and speakers. According to the authors, each activity has its own frequency shift footprint that allows its identification.

The other approach to detect the driver state, is to retrieve data from car behavior, such as distance to the front car or the lateral acceleration, and use it to monitor the driving behavior and infer a danger degree [72]. The problem to infer the dangerousness of a driving style or behavior is the lack of a standardized set of rules to define it. Hence, the level inferred is totally devised with logic rules by the programmer or application builder.

The challenge of performing both sensing and classification in real-time is added to the problem of selecting the method to sense distraction or fatigue. Real-time is mandatory since a distraction can occur just in less than second and its consequences can be catastrophic. For this reason, applications tend to use SVM which is a lightweight method compared to deep learning [73]. We also include in this section the driver identification as a topic prior and related to driver state. Driver identification consists in being able to identify different drivers according to certain features. It is intimately related to state determination because both imply recording signals from each different driver or from the car to carry out the objective. Its applications range from avoiding driving the car in some time spans, limiting speed in case of young or old drivers or avoiding not authorized driving.

The following sections focus on the structure of the algorithms for drowsiness, fatigue and distraction found in the review along with the important facts of every method. Later, we examine the platform used to embed the methods and the way researchers do it.

TABLE 4. Implementation information for vehicle and pedestrian detection articles: libraries/languages, platform, hardware, image size, detected classes and speed. The field HW contains the product name, the processor or any information provided. / is used in tables when any data is given in the original article.

Article	Libraries/ Languages	Platform	Hardware	Image Size	Classes	Speed (fps)
[38]	Caffe	Embedded Board	NVIDIA Jetson TK1	640x480	1	2.5 to 10 (depending on num. instances)
[60]	xfcdnn & PipeCNN	Embedded Board & FPGA	NVIDIA Tegra X2 & Xilinx Ultrascalae+	408x408	/	600 (GPU) & 60 (FPGA)
[61]	ORCHARD	Embedded processors	Intel Xeon E5440 & ARM Cortex A53	/	/	/
[30]	/	Embedded processor	TOSHIBA Visconti2 7502	720p	1	/
[56]	Open CV	Embedded Board	Rapberry Pi 3 & Odroid C2	320x97	2	6
[49]	Caffe	Embedded Board	NVIDIA Tegra X1	/	1	33
[40]	/	Embedded Board	NVIDIA Jetson TX1	1242x375	3	/
[45]	/	ASIC	65nm CMOS	720p	1	30
[43]	OpenVINO	/	/	672x384	1	From 40.2 to 63.5
[53]	/	/	/	720p	3	/
[62]	OpenCL	GPU	AMD Embedded RX-421BD	364x480	20	/
[35]	/	ASIC	40nm CMOS	1280x960	1	60 to 300 (depending distance to object)
[33]	/	DSP	ADI-BF561 600MHz dual core	858x525	1	20
[47]	Tensorflow	/	/	1242x375	3	57 (GPU)
[36]	/	Embedded Processor	S32V234 NXP	640x480	1	30
[29]	/	ASIC	Self-made ASIC	1920x1080	1	60
[4]	/	Embedded board	Freescale I.MX6Q	32x32	1	14 to 16
[58]	/	ASIC	65nm CMOS	640x480	2	90
[31]	/	FPGA	Xilinx Virtex-6 LX240T	128x64	1	132
[63]	/	FPGA	Xilinx Spartan–IIE XC2S300E	18x36	1	/
[32]	Vivado	FPGA	Xilinx ZC 702	1270x720	1	60
[50]	Caffe	Embedded Board	NVIDIA Jetson TX1	720x480	1	5

1) ALGORITHMS STRUCTURE

As said, SVM is often the preferred algorithm for classification when dealing with embedded applications. Authors in [69] apply SVM to classify between a drowsy and awaken state. The input consists in two signals, Galvanic Skin Response (GSR) and Electromyography (EMG) and the features derived from them. In order to have a better classification and cleaner features, they previously apply a median filter to those signals. Another case of SVM usage is found in [74]. The authors use Electroencephalographic (EEG) signals to classify the awareness state of a driver. To clean up signals they apply a Chebyshev bandpass filter, transform the signals to the frequency domain by means of a Fast Fourier Transform (FFT) and obtain the power spectral density (PSD). Then, they use a support vector regression (SVR) with a Radial Basis Function (RBF) as kernel to predict the reaction time with respect to the lane departure events generated by the simulator. This case is interesting since the authors use different power metrics of the PSD relating them to the reaction time. However, the regression coefficients indicate a poor correlation between the metrics and the reaction time with the exception of the alpha power that shows $R2 = 0.54$.

Another study where authors use features coming from human biosignals for the classification of the driver state with an SVM is [75]. They define the state of the driver with regards the Abraham-Hicks emotional guidance scale and establish three states: relaxed, stressed and fatigued. The main inputs for the SVM are features derived from electromyographic signals of the upper trapezius muscle, photoplethysmography signals of the earlobe, as well as inertial motion sensing of the head movement. In all cases, some preprocessing is carried out. In the first case, features are derived from the PSD obtained through FFT while in the third one, features are obtained from the application of the complementary filter to the gyroscope readings. The final selection of features to be used at inference is carried out considering the performance of every single feature. Later, they propose the final fusion of the selected ones. Another use of SVM is to give a range or scale of the drowsiness level. In [76], authors use Posterior Probability Model with the Platt algorithm [77] to obtain a degree of drowsiness between 0 and 1 (after binary classification).

The main goal is to obtain meaningful signals correlated to the attention of the driver. Apart from electric body signals, head nodding or eye movement are also considered for obtaining information about the driver state. In [78], authors predict the head pose using of CNNs and RNNs. The input consists of depth maps of the driver and the output is three values for 3D angles: yaw, pitch and roll. The CNN is used to reduce the dimensions of the input features and to provide meaningful information to the RNN, implemented by a LSTM network of two layers, which outputs directly the 3D angle value. The datasets used are the Biwi Kinect Head Pose dataset [79] and the ICT-3DHP database [80] together with data augmentation. The method provides better results than other studies, such as [81] while only using depth and not RGB plus depth. In[82], authors derive a method for obtaining the position of eyes. It consists of face detection by Viola-Jones algorithm with Haar-like features, illumination normalization, eye candidate generation and eye classification with an SVM. The inputs for the classifier are the position of the candidate's eyes and the intensity values. Although the method does not provide an exact evaluation of the driver state, it could help in building a complete system based on gaze detection.

A complete example of a drowsiness detector based on eye closure is found in [83]. This study offers a full pipeline description from image capture to percentage of eye closure (PERCLOS) computation. Authors begin with an image resizing and gray component extraction. Next, face detection is carried out using the Viola-Jones algorithm on integer images and Haar-like feature classifier. Following, eye detection is performed using local binary pattern histogram (LBPH) and its state, closed or opened, is established through a Partial Least Squares regression and SVM for the classification. Finally, they report PERCLOS computation based on the number of blinking among frames. One interesting point of the process is the use of PLS (Partial Least Squares) for dimensionality reduction, which offers better performance than PCA. It only needs 18 support vectors to get an optimal hyperplane whereas the PCA subspace requires 173 support vectors. Thus, the computation time is reduced one order of magnitude (from 42ms to 4ms). According to authors reasoning, the choice of Viola-Jones face detector is based on its robustness and simplicity compared to other detectors.

In general, when the input is too complex or has too much information for a traditional classifier, deep learning is a good option. In [73], authors classify the behavior of the driver among ten different activities using different CNNs. The input for the model is a $64 \times 64 \times 3$ RGB image and the CNNs are the well-known VGG-16, AlexNet, GoogleNet and ResNet. They use dataset recordings obtained through the Carnetsoft [84] driving simulator. The problem of heavy structures, such as the mentioned networks, is that they require a huge number of training samples and that the inference is not as fast as in classical models (i.e. linear SVM).

In [85], authors implement a detection and classification model based on one CNN and knowledge distillation. The input for the first network is an RGB image and the output is the detection of the eyes, mouth and face. The second network takes those features as inputs and classifies people as being in a yawning, normal or drowsy state. For detection, they use Multi-Task Cascaded Convolutional Networks (MTCNN) [86] and the classification network is composed of 4 streams, each one with a similar structure to AlexNet, that take each feature coming from the previous network as input.

Driver identification is often related to driver state or driver behavior analysis since both tasks imply sensing signals from the individual driver or its interaction with any of the elements of the driving context. For example, in [87], authors build a driver identification algorithm based on brake and gas pedal signals, cepstral analysis for feature extraction and ANN for classification. However, these signals can also be used for classifying the behavior of the driver. In this case, it is interesting to note the use of cepstral analysis for feature generation since it is able to transform the components, which are not separable, into a linear sum, providing good separability. In this case, the sensing mechanism is appropriate since it does not interfere with the driving activity as with ECG, EEG or other biological signals. In [70], there is another case of driver identification that is used for driver state classification. Authors sense heart rate signals from the steering wheel. The problem of this methodology is that it requires to have both hands at the wheel, which in some contexts does not happen (e.g. when changing gears). The positive point is that the method is not intrusive regarding the driver and its embedding is well embraced by car component manufacturers.

A novel way of identifying driver activities is found in [71], where authors sense the activities through frequency shifts (Doppler effect) of signals emitted by speakers and received by a microphone. They first convert the signals to the frequency spectrum using SOFT (Sliding-window Overlap Fourier Transformation) and then they reduce the dimensionality by means of PCA. Later, they use SVM binary classifiers assembled in a gradient forest to provide on-line inference and early recognition of inattentive activities. The study stands out the use of Self Organizing Maps for detecting driving modes.

Only one example of driver behavior determination through car driving signals [72] has been found in the review process. Authors record data such as distance to next car or frontal/lateral accelerations to estimate the driving behavior: accelerating, decelerating, moving left or right and more. The identification is carried out by a Hidden Markov Model and its output is further used for a danger level estimation through fuzzy logic rules.

2) EMBEDDING METHOD AND PERFORMANCE

The main problem with drowsiness detection is capturing meaningful signals from the driver without interfering with its attention at the driving environment. In [69] authors capture Galvanic Skin Response (GSR) and Electromyography (EMG) signals and use a Lilypad Arduino for computation with GSR, EMG and BLE interface boards. The main issue is that the driver has to wear the sensors to obtain meaningful signals. This could bother the driver and difficult the driving activity. The captured signals are sent to a LG G Smart-watch where they are analyzed to classify the driver's state. It is not clear how they embed the SVM classification algorithm in the smart-watch. The achieved performance is good, over 90% precision. The problem in this case is the dataset. It only consists in 4 males and 2 females which, with the help of a physician, establish their own state by using the Karolinska Sleepiness Scale (KSS) [88].

In [74] authors develop a EEG sensor that does not need any type of conductive gels what facilitates its use. However, the sensor still needs to be worn by the driver. After preprocessing the signals, they use a SVR obtained from LIBSVM [94] and develop the model in JAVA to be able to embed it in mobile devices. Once again, SVM seems to be the preferred option to make an embedded model. Bluetooth is again used to connect the sensor and mobile platform. This adds latency to the model that is able to provide alert information every 2 s using regression to obtain the reaction time and awareness state. In [75], authors use also an Arduino platforms (plus BLE and sensors shields) to connect to a Google Nexus 5 smartphone running a mobile application. The app uses SVM to predict the emotional state of the driver using a three class (relaxed, stressed, and fatigued) classification. For this purpose, the model reduces the initially considered 36 features to only the top-15 ones in order to fit with the computational capabilities of the Google Nexus 5. The performance of SVM is good with 96.23% accuracy and a battery lifetime of 8 hours, and can be improved by incorporating multiple features extracted from PPG, EMG, and IMU up to 99.52%. Once more, the emotional response paradigm has only been tested on ten subjects, consisting on 10 min baseline, 5 min pre-stimulus, and 5 min post-stimulus measurements.

Again, SVM is preferred due to its easy embedding and the evolution of tools for embedding models on hardware and software. Logistic regression is also easy to embed and can deliver good results as shown in ([95], [96]). This is also the case of neural networks that are becoming easier to embed, thanks to frameworks such as Tensorflow [97] and Theano [98] and specialized embedded hardware, such as NVIDIA Jetson boards or software libraries such as TFLite [99] or TFMicro. One example is [78], where a system containing both CNN and RNN (devoted to head pose estimation) is embedded into NVIDIA Jetson TX1 and TK1 platforms. The input consists of 64 × 64 pixel images and the recording is done using a Kinect camera [100] and the library Libfreekinect. The network consists of three convolution plus max pooling stacks followed by two additional convolution layers and two layers of LSTM cells. The system is able to achieve 19 fps on the TK1 with the use of the GPU alone and 30 fps on the TX1 when they use the cuDNN library and the GPU.

In [73] authors embed four different CNNs, such as VGG-16, into a Jetson TX1. The problem of CNNs is the high latency in inference, 8 fps in ResNet with 64 × 64×3 images, not achieving real-time performance. They compensate this problem by reaching 92% accuracy in tasks where the input information is too big to be processed by any classic algorithm.

As already mentioned, embedded implementations of deep learning try to find ways to simplify the network. In the case of [85] authors opt for deleting two streams from the multi branch CNN that they devoted to drowsiness detection. They also train a bigger network to teach a smaller network (process known as knowledge distillation). The overall system implemented consists of a Multi-Task Cascaded Convolutional Networks (MTCNN) for eyes and mouth detection plus another network, Driver Drowsiness Detection Network (DDDN) with multiple branches, devoted to obtain the drowsiness level. With the branches cut, they reduce redundant information and increase the speed from 6.1 to 13.5 fps on the NVIDIA Jetson TK1. With the compressed model (from knowledge distillation) they lose only 3.6% accuracy (from 94.8% to 91.2%) but attain 14.9 fps. There are two main downsides to this study. First, the dataset, as in many other studies, is not public and results cannot be checked. And second, the latency, despite the optimizations, does not attain real-time performance, what discourages the use of CNNs for embedded applications that require real-time performance.

In [87], authors map an ANN onto a Xilinx's KINTEX-7 family FPGA for driver identification using gas and brake pedal signals. The idea in this case is to map the most computationally intensive part to hardware, the matrix operations of the Multi-Layer Perceptron, and to implement in software the less demanding parts, such as preprocessing with band-pass filters.

In [83], authors implement a full drowsiness detection pipeline as software in a DSP. The full eye detection process consists in image capturing and resizing, face detection, eye detection and eye classification. In this case, the overall systems performs at 3 fps on a TMS320DM6437 DSP with images 288 × 360 pixels. The main bottlenecks is face detection that consumes nearly 72% of the processing time while eye localization takes 21% of the total latency, 888.31 ms.

A strategy for easing the embedding of NNs in MCUs (microcontroller units) is presented in [92]. Authors use a RNN embedded in a smartphone for detecting driving activities. They follow the strategy for quantizing and compressing weights found in [101]. Weights are quantized to 8 bit unsigned integers, inputs are quantized on the fly, the multiplication is accumulated in 32 bit integers and then transformed to float when biases are added and used for the computation of the activation function. With this strategy, they are able to reduce the weight of the model from 412 KB to 77KB without apparent loss in accuracy and decreasing latency. This strategy is expanded in [6] to provide full integer inference. This same strategy is found in TF Lite [99] framework which helps reducing the memory and latency constraints that CNN impose. In [93], authors implement a drowsy classifier with a quantized CNN (MobileNet) to video recordings in a smart-phone. TF Lite provides functions to quantize the networks and an API for calling the networks and implement inference in Resource-Constrained devices easing the task of embedding NNs. However, the problem is still latency that limits the achieved frame rate to 5-10 fps, which is still not valid for many real-time contexts.

TABLE 5. Algorithm and data information for driver state, behavior and identification articles: task, classification, dataset, embedding and performance.

Article	Task	Classification Model	Dataset	Data	Processing	Performance
[69]	Drowsiness state	SVM	Own	EGM&GSR	Median Filter & Threshold & Feature Extraction	90% precision
[78]	Head pose estimation	CNN & RNN (LSTM)	Biwi & ICT-3DHP datasets	Depth and RGB image	/	2.0 MAE
[72]	Driver behavior	HMM/Fuzzy System	Own	GPS&Accelerometer &Camera	Filtering & Downsampling & Normalization & Quantization	99% Accuracy
[74]	Driver state	SVR	Own	EEG	Chebyshev band pass & FFT & PSD	0.932 RMSE
[75]	Driver state	SVM	Own	PPG&EMG&IMU	FFT & PSD	95% Accuracy
[82]	Eye localization	SVM	Yale B[89], AR [90], ORL[91]	RGB images	Normalization & eye candidate generation	93.48 % Accuracy
[85]	Driver state	CNN	Own	RGB images	/	91.2 % Accuracy
[73]	Driver behavior	CNN	Own	RGB image	/	92 % Accuracy
[87]	Driver identification	MLP	Own	Gas and brake pedal signals	Cepstral analysis & FFT & band pass filter	72.0 % Accuracy
[83]	Driver state	SVM	Own	RGB images	Resize & Gray image extraction & V-J face detection & LBPH & PLS regression	99.24% TPR 8.44%FPR
[71]	Driver state	Gradient Model Forest	Own	Microphone, Speaker	SOFT & PCA & SVM	94.80 % Accuracy
[92]	Driver behaviour	RNN	Own	Accelerometers, gyroscopes, magnetometers and GPS	Mean imputing & standardization	88.7 % Accuracy
[76]	Driver state	SVM & Posterior probability	Own	EEG	BPF & FFT	91.2 % Accuracy
[93]	Driver state	CNN	Own	RGB image	/	89.0% Accuracy

3) PARTIAL CONCLUSIONS

This section detailed the 14 papers containing algorithms and its embedding for driver's state and behavior determination. Biometric body signals, such as EEG, GSR, EMG, are widely used. However, these signals often imply intrusive methods that are not suitable for real driving situations. One option to avoid this issue is to place the sensors in the steering wheel. This improves latency since it avoids the use of wireless communication between sensors and the computing platform. However, it implies certain implementation problems for sensing, such as dependence on signals coming from both hands. Another alternative is the use of cameras to sense the driver. In this case, the models require more resources and usually lack of real-time performance. Finally, another way of sensing behavior (but not its state) is through car signals. In this case, the main problem is to determine which actions are related to each behavior.

Regarding models used, SVM is the most popular one due to its lightweight nature and easiness to embed in hardware. CNNs and RNNs are also used even that those structures imply a heavy resource usage, which ends up in low throughput and lack of real-time performance.

One important comment is the lack of a uniform dataset shared among research works. This makes impossible to compare articles and methodologies. Moreover, in the case of biosignals, any public dataset has not been found as also happens in the case of the car signals, while regarding head pose estimation, public datasets are available.

Regarding hardware used, GPUs, with the NVIDIA Jetson family, stand out as one of the most used devices. However, due to the resource demanding nature of CNN, real-time performance is achieved just in rare cases.

The summary of the selected articles is presented in Tables 5 and 6 with the information related to their key factors affecting performance: tasks, NNs, libraries, dataset, embedding strategies, hardware platforms, and results regarding performance.

C. TRAFFIC SIGN RECOGNITION

Driving tasks are restrained to strict norms which affect the behavior and possible actions of the driver. These norms change depending on the conditions of the driving context: number of lanes, amplitude of the hard shoulder, existence of crossings, surroundings and others. Traffic signs and lights are placed by authorities in proper places to adjust the drivers' behavior and to specify the actions permitted or prohibited in every circumstance. It is essential for drivers to be aware of this information in order to act accordingly and drive safely. However, in some occasions, the driver might get distracted or miss the traffic signs and lights. To avoid this situation, Traffic Sign Recognition (TSR) methods capture the information covered in signs and lights and serve it to the driver, thus reducing the probabilities that the driver is unaware of driving regulation context. These methods must be fast enough to provide the information in time but accurate enough to avoid confusing the driver. However, the diversity

TABLE 6. Implementation information for driver state, behavior and identification articles: object classes, libraries/languages, platform, hardware, image size and speed.

Article	Libraries/ Languages	Platform	Hardware	Image Size	Classes	Speed
[69]	/	Smart-watch	LG G smartwatch	/	Awake/Drowsy	/
[78]	Keras	Embedded	NVIDIA Jetson TK1 & TX1	640x480	3d angles: yaw, pitch and roll	19/30 fps
[72]	/	DSP Processor	TI DM3730	/	7 driving events	/
[74]	lib-svm	Smartphone	/	/	Vigilance degree	/
[75]	/	Smartphone	Google Nexus 5	/	Relaxed, stressed and fatigued	/
[82]	/	Embedded processor	INTEL Bay-Trail	/	Eye / Not eye	16 fps
[85]	Tensorflow	Embedded	NVIDIA Jetson TK1	640x480	Normal, yawning, drowsy	14.9 fps
[73]	Tensorflow	Embedded	NVIDIA Jetson TX1	64x64	10 driving behaviours	14 fps
[87]	/	FPGA	XC7k325T Xilinx's KINTEX-7 family	/	n drivers	4.7 µs
[83]	OpenCV & MATLAB	DSP	TMS320DM6437	576x720	PERCLOS Closed or opened eye	3 fps
[71]	lib-svm	Smartphone	HTC Desire G7, ZTE U809, HTC EVO 3D, SAMSUNG Nexus 5	/	5 driving activities	/
[92]	/	Smartphone	/	/	18 driving patters	4.27 ms
[76]	lib-svm	Smartwatch	/	/	Drowsiness level	4 ms
[93]	OpenCV & TFLite	Smartphone	Qualcomm Snapdragon 430 MSM8937	224x224	2 classes: rested and sleep	5 to10 fps

of external conditions, such as rain or fog, or even lack of maintenance of traffic information make the task of recognizing traffic sign information an outstanding challenge.

Generally, there are two sources of traffic sign information: Global Positioning System (GPS) information and TSR methods. GPS information can be useful but has two main downsides that make it inappropriate as main source of information. First, the lack of satellite or connectivity information in some contexts as for example, long tunnels or mountainous areas and, second, the lack of updated information of the context (i.e. maps). TSR methods are based on computer vision and can work everywhere but they have to offer a good trade-off between latency and accuracy. This, however, must not exclude the possibility of combining both types of information: internal, from the car itself, and external, from a GPS system or a database.

In [29], a database of geo-referenced traffic signs is developed concomitant with a TSR system. The idea is to generate a positive synergy between the two sources of traffic sign detection: the TSR system provides new signs and their locations or corrects and updates the database, while the database reports sign status and check for false negatives. Nevertheless, one important problem with mixed recognition systems is the weight of each branch and which has modification rights over the other.

Another important distinction in TSR systems is the common differentiation between traffic lights and traffic signs. Studies tend to focus in one type of traffic signaling element, such as in [102], where only red and green traffic lights are taken into account. Furthermore, there are even splits among traffic signs, for example in studies covering only speed limit signs [103]. The reason behind this object specialization relates to the performance in both sign detection and classification. Usually, characteristics such as color or shape, among others, are used to detect signs. Hence, class diversity is problem for detection, lowering the final classification rate as well as imposing demanding hardware requirements. Keeping an application specific object detector and classifier, eases the procedure. For example, in [104] only triangular traffic signs are modelled.

It is important to notice that there are important and widely used traffic sign dataset, such as the German Traffic Sign Recognition Benchmark (GTSRB) [105] or the Belgium Traffic Sign Dataset (BTSD) [106] although few projects use the original datasets but modified versions or their own dataset. These object specifications and dataset modifications restrict the proper comparisons among studies. Other used datasets are San Diego Traffic Sign (SDTS) dataset [107], the Stereopolis dataset[108], or the Nexar Traffic Light Challenge dataset [109]. In this section, the different approaches found for TSR are reviewed, both from the perspective of algorithms and hardware used.

1) ALGORITHM STRUCTURE

Most TSR algorithms follow the same structure. They have two differentiated steps: (1) detection or generation of the Region of Interest (ROI) for defining the regions in the image where the sign is located; and (2) recognition, where the detected sign is classified in one of the predefined classes.

The input in all studies is an RGB image. In some cases, authors apply preprocessing procedures to the image before the detection process. For example, in [110] authors apply histogram equalization to the ROI of the object detected in order to adjust the intensity of the pixels in the YCrCb color space. In [111], authors apply two preprocessing steps: upper right corner image selection and white balancing. Selecting a specific part of the image is found in some studies, such as in [112], where authors crop 25% from the top and 15% from the bottom part of the image. In [113], the RGB image is converted to the HSV color space, to better account for relationships between colors. Although these procedures are found in the original studies as preprocessing steps, similar methods are defined as part of the detection process in other studies, such as Red Blue Channel Enhancement (RBCE) found in [114]. Simpler strategies are also found, as in [115], where the authors simply convert the image to gray scale.

For the detection step most of the studies use a concatenation of different algorithms until obtaining the desired ROIs. In [116], authors use a pyramid scheme to obtain the same image in different resolutions and then they apply HOG to produce features. For the HOG computation, they use EVE (embedded vision/vector engine), a fixed-point vector coprocessor. The final detection step is carried out by an Adaboost cascade classifier with mean-shift algorithm to group the final candidate ROIs. In [113], authors use one pass blob detection with 4-connectivity to detect the regions of interest. Other studies also apply connected components to obtain the final regions of interest. In [112], authors produce a binary image, based on RB Normalization and thresholds, and then apply Connected Component Labelling. The use of binary images is common to produce the desired regions such as in [117] and [111]. Another useful technique to produce the desired ROIs is to use template or shape matching, as in [111].

The detection process in ([103], [117]) consists in converting the images from RGB to HSV, applying color threshold to obtain a binary map of red points and extracting the contour using structural analysis functions. Final ROIs are extracted based on minimum shape requirements and aspect ratio constraints and are detected with polygonal curve approximation and minimal bounding rectangle functions. They both use binarization and shape matching procedures.

More complex processes have been found, such as in [114], where authors apply Red/Blue Channel Enhancement, followed by Maximally Stable Extremal Regions (MSER) [118] to produce the generated ROIs. Then, they apply region merging and a shape/ratio filtering before obtaining the final ROIs. Another elaborated process is found in [117] where authors apply the following steps to obtain candidate regions: RGB to HSV conversion, color thresholding and contour detection. In [110], authors use a different approach for detecting the regions of interest. They apply Multi Block Local Binary Pattern [119], faster than Haar or HOG. Then, they use six different cascade classifiers to determine the image class. An example of robust traffic light recognition is found in [102]. Authors produce the candidate regions for red and green traffic lights by modelling the traffic light colors in HSL format as 1D Gaussian distributions. After that, the region of interest is further redefined by applying the top-hat algorithm to an extended region and thus making the algorithm robust to light changes. The region is also passed through a filter of shape and ratio thresholds. Another approach to detect regions is to train a cascade of classifiers with Haar-like features as in [115].

Although most of the papers follow the detection plus recognition procedure, there are some only devoted to detection. In [120], a CNN encoder-decoder based on U-Net [121], is used to produce grey scale images. These images are then binarized, pixels are clustered using DBScan [122] and filtered to produce the final regions where traffic lights are found. The loss of the overall algorithm uses DICE measure (as formulated in [123]) for the regions generated.

The case of [104] is a special one, since authors devote the algorithm only to triangular traffic signs using a five step based method: computing gradients of pixels, Harris corner detector, encoding of corners, applying RANSAC for line detection and finally detecting the baseline. Although the performance is not distinguishable from other methods, the RANSAC and line detection methodology makes the method more robust against rotations that can happen due to the inclination of the traffic signals.

In [124], authors use a knowledge based and attention mechanism to detect traffic signs with Recurrent Neural Networks (RNN). They use a SqueezeNet and a fire module to provide future maps for the LSTM cells. These LSTM cells are modified to include an attention mechanism to decide the part of the image to concentrate on. Finally, a reverse Gaussian method is used to apply domain knowledge to the process. After the generation of ROIs, the following steps are the generation of the appropriate features for classification and their feeding (or directly the ROIs) to the final classifier.

Once ROIs were generated, either they directly feed a classifier or they are used to generate the appropriate features for classification. HOG and LBP stand among the most used feature extractors. In [113], after filtering by shape and resizing ROIs, HOG features are extracted and fed to a linear SVM. This structure is also found in other studies: authors in [114] send HOG results to a group of linear SVMs that are used (in a voting process) to obtain the final classification result, and in [112], authors apply HOG and fed it to a soft margin SVM after filtering by shape, resizing and converting[111], they use the final resized ROIs as feature vectors for a linear SVM.

Apart from HOG and LBP, there are other options for creating features for the classification step. For example, in [117], authors first extract the important contents from the image descriptor named Centroid-to-Contour descriptor. It consists in several steps: the computation of the moments of the image and its centroid as well as contour detection. Next, distances from centroid to the points in the contour are computed, sampled and normalized to create the new feature vector. This feature vector is then fed into a linear SVM.

Convolutional Neural Networks are the other preferred classifier. In [116], authors use a CNN for classifying different German traffic signs directly from the ROIs obtained in detection. The network consists in two stacks of convolutions plus max pooling, another convolution layer and two final fully connected layers. Another study that also uses CNNs is [110], where the input for the network is also the ROI from the detection step. The network structure is quite similar to the previous study: convolutions and max pooling followed by fully connected layers for classification. A special case of CNN usage is [107], where a two task network (for both classification and regression) is built in order to classify the shape label and determine the 2D orientation of the signal. The two branches are implemented after a VGG16 or Inception V2 as base network, which generates the features for each of the branches. For the detection branch, they use a modified version of Single Shot Detector (SSD) followed by Non-Maximum Suppression (NMS). Authors apply a template transform to the networks outputs to segment the final sign.

Other classifiers used are self-organizing maps (SOM), as in [125], where the input consists a sort of histogram count of red pixels by row.

2) EMBEDDING METHOD AND PERFORMANCE

Studies found in this review focus on three types of platforms: smart-phones, FPGAs and embedded processors/boards. Embedded boards are the mainstream choice to build traffic sign recognition systems. However, smartphones are keeping pace with them due to their increase in computing capabilities. FPGAs stand as the less used platform probably due to their complex implementation process.

Smartphones have an uprising in computing performance and efficiency but they still have constrained resources compared to personal computers and many embedded platforms. In [107], authors implement a CNN in a Qualcomm Snapdragon 820A processor using the Snapdragon Neural Processing Engine (SNPE) SDK for optimization purposes. Their network was built onto a base or skeleton network plus two branches devoted to pose and shape estimation. They make a series of changes to the original structure to increase throughput, from 1 to 7 fps while only loosing around 6% in performance. Optimizations consist in changing from a heavy base network (VGG-16 to Inception V2 and then SqueezeNet), cropping the image and reducing resolution.

Another study that uses a smart-phone is [110]. Their authors implement a MB-LBP plus CNN system in a Huawei P9 Lite with has an octa-core processor (ARM-Cortex 53 cores). Image resolution is 640×480 pixels and they achieve 10 fps and 94.24% accuracy on GTSRB. For implementing MB-LBP they use OpenCV Training Cascade Library. No clues are given regarding the implementation of the CNN in the ARM processor; neither regarding the embedding option chosen nor on the optimizations carried. However, being the input for the CNN of size 32×32 and having only two fully connected and three convolution layers, it is possible that any optimization was done. In [114] authors implement the whole data processing pipeline plus SVM on an Apple iPhone 6s, which has a dual-core 1.84 GHz Twister processor and 64GB ROM with 2GB RAM. They obtain 94.48% recall at 30 fps on 640×480 pixel images. However, no details are given on the embedding methodology or latency analysis.

Other papers provide some hints about implementation details. For example, in [117] the whole pipeline of the new Centroid to Contour descriptor plus SVM is implemented in two Android platforms using OpenCV and LIBSVM [94] for detection and classification respectively, and Android NDK (Revision 10e) toolset and OpenCV4Android SDK for the embedding. In this case, it is explicitly established that the system is trained apart and only inference is embedded. The main platform used is an Android smartphone equipped with a 2.3 GHz Qualcomm Snapdragon Quad-Core CPU. Authors report accuracies above 95% for all the traffic limit signs and throughputs up to 30 fps while processing 720p video streams.

In [102], there is an example of optimizing the system related to the platform. They embed their ELM plus finite state machine in a Samsung Note 3 smartphone equipped with a quad-core Krait 400 architecture CPU at up to 2.3 GHz per core and an Adreno 330 GPU with a frequency of 450 MHz and 3G RAM. For the conversion of the input image to HSL they use a LUT table and they accelerate the K-ELM algorithm with OpenCL and a CPU-GPU fusion approach. Furthermore, they built a parallel pipeline for processing different ROIs at the same time. The full system is applied on 1280×720 videos for recognizing different types of red and green lights achieving 93.2% mean accuracy and up to 20 fps.

In [115], authors compare a set of classifiers: KNN, SVM, MLP, Optimum Path Forests (OPF), Least squares and Extreme Machine Learning (EML). They decide to choose OPF as the best model to embed in a smartphone according to the trade-off between latency and accuracy.

Concerning implementations on embedded boards, authors of [111] implement their pipeline for TSR in a FriendlyARM Tiny4412 board, which has a 1.5GHz quad-core ARM Cortex-A9 Samsung Exynos4412 as main processor. This board comes with 1GB RAM, 4GB FLASH, and 7-inch capacitive touch screen. They implement SVM with OpenCV and use it for inference on the embedded board. As they report, the most demanding part of their full algorithm corresponds to detection, color segmentation (red, blue) and shape matching. For this part, they implement a multi-thread system using the Qt library: images are divided in four parts and each one is assigned to a thread for color segmentation while for shape matching, every thread manages a candidate. The performance goes from 10-11 fps without threading to 15 fps with it. They obtain a 90.1% accuracy while recognizing 50 different types of traffic signs. In this case, they opted for a hardware related optimization, but model parameter related decisions can also improve the system latency.

The most computationally demanding step of a TSR algorithm is detection. In [112], authors study the impact on performance of adding several thresholds to color segmentation. Their result is that several thresholds produce the best detection rate but it slows down the process. They also optimize HOG by using fixed-point computations and study the effect of the number of bits related to performance. To differentiate the algorithm at the hardware level, they implement two modules: (1) detection on two 640×480 pixel images and (2) recognition module on the ROI inputs of size 32×32. Detection is implemented using OpenCV and classification using the library LIBSVM. Regarding the hardware, they implement the proposed TSR hardware architecture on an ASIC using 90nm CMOS from TSMC. The core size is 0.26mm^2 and the chip size is about 1mm^2. The design operates at 105 MHz clock frequency. They are able to compute one full pass at 135 fps with 91.53% accuracy for the three types of signs classified: prohibitory, danger and mandatory. The overall energy consumption of the system is estimated as 8 mW. This is an important fact since it is the only study providing explicitly energy consumption results. It stands out that, being energy consumption important for embedded automotive applications, almost any study has focused its attention on it.

In [116], authors use a TI's TDA3x SOC (low-power heterogeneous architecture for ADAS) which comes with fixed- and floating-point dual TMS320C66x generation of DSP cores. It also includes a programmable vision acceleration engine (EVE), dual ARM Cortex-M4 cores and an image signal processor (ISP). They implement the image pyramid and HOG in EVE and Adaboost classifier and the CNN in each of the DSP processors. They obtain an overall accuracy of 89.6% and the system has a throughput of 15 fps. That low throughput could be due to the limited processing capabilities of the system, but also due to the overload of heavy demanding algorithmic structures such as CNNs and Adaboost.

Again, few application use FPGA for TSR. In [113], authors try to detect blobs of two different traffic lights: green and red. For that, they split blob detection in two branches with 4 connectivity labelling in only one pass by recording the blob position in a table and the connections between pixels in another table. Finally, they merge both tables to obtain position of blobs. The whole system containing preprocessing plus blob detection plus HOG plus SVM is devoted to green and red traffic light recognition and is implemented in a Xilinx Zynq ZC-702 board with well-balanced workload on FPGA fabric and the on-chip ARM processor. For fast memory access, they use AXI4-stream bus with a video DMA, realizing high-speed from FPGA to frame buffers in DDR memory. They report 92.11% and 94.44% recall for green and traffic lights and, depending on the number of red lights in the picture, and 60 fps (up to 100 fps). Data formats and arithmetic used are not detailed and they only show the System-on-Chip architecture at block level. Authors in [121] map their implementation on a platform that combines a Xilinx Virtex II XC2V1000 FPGA, used in the preprocessing of the images with the Axeon VindAX processor on a PCI development board for the SOM. They claim that signs were perfectly detected and identified under variations of scale (as small as 25 pixels wide), rotation (up to 20%) and occlusion (up to 15–20% of vertical and 5–10% of horizontal occlusion, depending on the area being occluded) at 19-21 fps on VGA images. Their SOM maps have been trained with images from Spain and Czech Republic, while the system has been tested with images from Britain, so they use their own test set with almost 100% success rate.

One point that is not usually tackled in research and which is very important for the automotive sector is safety and robustness of models. An interesting study regarding this topic is [126], where authors analyze the reliability of a CNN devoted to traffic recognition and implemented on a FPGA. The core of the study consists in injecting errors in the data streams to see the robustness of the system. One important conclusion is that pipelined parallel neural network implementations show a lower Architecture Vulnerability Factor (AVF) than timing-multiplexed architectures. The results show that the proposed network and system Failure-In-Time (FIT) would comply with the ASIL-B and C requirements but not with D.

3) PARTIAL CONCLUSIONS

This section analyzed 14 TSR articles regarding their ML pipeline and hardware implementation. Most algorithms divide the task in two parts: detection and classification. In the detection step, techniques such as color thresholding, connected component labelling or shape matching stand out. In the recognition process, two algorithms share the main uses: SVM and CNNs. The detection steps are the most computationally demanding and the main objective for hardware and software optimization.

Regarding hardware, it is surprising the use of smartphones while automotive embedded boards, FPGAs and ASICs, such as the Jetson family, are also used in publications. Both smartphones and embedded boards have problems when dealing with CNN to achieve real-time performance. The preferred algorithms is SVM due to its weight and performance. It surprises that frameworks such as Tensorflow Lite and Tensorflow for Microcontrollers [127] or ARM-NN [128] did not appear in the process of embedding and adapting CNNs to MCUs.

One of the main obstacles for a fair comparison among articles, is the difference in the final recognized or detected objects. Another problem is the different data format used (concerning inputs). All selected articles use images, but their size is different in most cases what severely impacts both ML and speed. It is also surprising that any article has implemented a full pipeline of traffic light and sign recognition. A method that embraces both object types with a remarkable accuracy and decent throughput would be of significant use in industry. Another important note is that all systems are trained externally and are frozen for its inference; any on-line learning or model adaptation has been found to the best

TABLE 7. Algorithm and data information for traffic sign and light recognition articles: task, pre-detection, detection, pre-classification and performance. In this table the columns predetection, detection, preprocess and classification model are in this exact order to reproduce the usual algorithm order. In [107], the reason of performance variation is due to the different optimizations for speed-performance trade-off. In [114], the different values correspond to different datasets. In [120] the performance is given as a threshold because in the paper is not specified for all classes.

Article	Task	Pre-detection	Detection	Pre-classification	Classifi-cation	Dataset	Performance
[113]	Green and red lights	RGB to HSV & Binarization	4-Connectivity Labelling	Shape filter & Resizing&HOG	SVM	Own	93.27 % Recall
[107]	Prohibitive, mandatory and danger signs	Cropping and resizing	CNN & template transform	/	CNN & template transform	STDS & GTSBR	82.8 to 88.4% mAP
[116]	24 traffic signs	Image pyramid	Adaboost	HOG	CNN	TI automotive	/
[104]	Triangular traffic signs	pixel gradient & Harris corner detector & corner encoder	RANSAC	/	/	Stereopolis [108]	81% detection rate
[112]	Prohibitive, mandatory and danger signs	Cropping & RB Norm thresholding	CCL (connect. comp label)	shape filter & resizing & HOG	SVM	GSTBR	90.85% accuracy
[102]	Red and green lights	HSL / Geometric color region filtering / Top-hat transform	Shape ratios	HOG-LBP	k-ELM + FSM	Own	93.0% F1
[114]	35 traffic signs	RBCE	MSER & shape filtering & merging	HOG	SVM	GTSBR	94.48% recall
[110]	43 traffic signs	/	MB-LBP	Histogram equalization	CNN	GTSBR & BTSD &Own	94.0%,92.3%, 78.34% accuracy
[111]	50 traffic signs	Color segmentation and thresholding	Shape matching	Binarization	SVM	Own	90.1% accuracy
[117]	10 speed limit signs	HSV & Thresholding & Contour detection	CCL	CtC	SVM	Own	> 95.0% F1
[120]	Traffic light detection	CNN & binarization	DBScan & cluster filtering	/	/	Own (from Nexar)	92.22% DICE
[115]	9 speed sign classes	Gray scale	Cascade classifier & Haar features	Adaptive Thres. & Size filtering & Resizing	OPF	Own	99.65% accuracy
[124]	13 class traffic signs	/	SqueezeNet & KB-RANN & Reversed Gaussian	/	/	BTSD	81.0% mAP
[125]	8 classes	Color segmentation	Size and number threshold	/	SOM	Own	99%

TABLE 8. Implementation information for traffic sign and light recognition articles: tasks, libraries/languages, platform, hardware, image size and speed. In [116] two values are given for speed since two processors were tested.

Article	TRS Task	Libraries/Languages	Platform	Hardware	Image Size	Speed (fps)
[113]	Green and red lights	/	FPGA	Xilinx Zynq ZC-702	1024x768	100
[107]	Prohibitive, mandatory and danger signs	Caffe / SNPE1 SDK	Embedded	Qualcomm Snapdragon TM 820A	1280x720	0.2-7
[116]	24 traffic signs	TI's Vision SDK	Embedded	TI's TDA3x SOC	/	15
[104]	Triangular traffic signs	RTMaps2 / OpenCV	Embedded	/	/	30
[112]	Prohibitive, mandatory and danger signs	LIBSVM	ASIC	APR on TSMC 90nm CMOS	1360x800	135
[102]	Red and green lights	OpenCL	Smartphone	Samsung Note 3	1280x720	20
[114]	35 traffic signs	/	Smartphone	Iphone6	640x480	20
[110]	43 traffic signs	Tensorflow	Smartphone	Huawei P9 Lite	640x480	10
[111]	50 traffic signs	OpenCV / Qtlibrary	Embedded	FriendlyARM Tiny4412	640x480	15
[117]	10 speed limit signs	OpenCV / OpenCV4Android	Embedded	Radxa Rock Pro & ASUS PF500KL	640x480	44/32
[120]	Traffic light detection	Keras	Embedded	NVIDIA Jetson TX2	256x455	15
[115]	9 speed sign classes	C	Smartphone	Android 2.5 GHz 2GB RAM	35x35	25
[124]	13 class traffic signs	C	Embedded	NVIDIA Jetson TX1	542x412	10
[125]	8 classes	Modular Map Design/	FPGA	Xilinx Virtex II XC2V1000	640x480	20

knowledge of authors. To facilitate a general view of the differences among articles,a summary and comparison of the articles is presented in Tables 7 and 8.

D. ROAD DETECTION AND SCENE UNDERSTANDING

Scene understanding refers generally to the segmentation and detection of the components of the driving environment to produce an accurate representation. Although it is not directly an ADAS function, scene understanding can help other ADAS tasks such as collision avoidance or lane change functions by providing the required information or directly as a part of that task. In some cases, instead of pursuing a whole scene segmentation, only the road is detected or segmented. Traditionally, road was detected using manually extracted features that model the road through polygons, color of lane markers or template matching [129]. Lately, researchers have also adopted deep learning as the main technique to solve road detection.

Due to nature of the road segmentation and scene understanding tasks and the limited amount of works found, we present together the deep learning methodologies used, as well as their optimization, embedding and achieved performance.

Full scene understanding requires its semantic segmentation, what can be directly performed with CNNs. This process imposes high computational requirements (hundreds of GOPS or TOPS). Again, due to the limited platform resources, CNNs have to be adapted or modified to reduce their memory and processing requirements. Another option is to modify the network for its direct hardware mapping, which is a good option if low energy and/or high throughput are desired.

In [130], authors determine 3 techniques that can help implementing a CNN in a Resource-Constrained device: quantization, layer fusion and sparse multiplication. Their task is to determine the free drivable space. For this purpose they take a pre-trained network, ResNet10, with the Cityscapes dataset [131] and modify it for its embedding in a TDA2x SoC from Texas instruments. Quantization helps to achieve low-size networks and decrease latency. It basically consists in the conversion of floating-point weights and activations to fixed-point equivalents. There are two different ways to do it: (1) by fine-tuning/training the network in a quantized-aware mode or (2) converting the operations and values to fixed-point after the network is trained (in this case some operations such as activations remain in floating-point). Afterwards, they apply layer fusion consisting in grouping different operations in the same process. For example, 2D convolution and max pooling can be grouped what saves the need to store intermediate results in memory, or a given input is loaded into the internal memory only once when several convolutions use that same input, as in Inception modules [132].

In general, this type of fused operations helps to reduce memory bandwidth and improve speed. Finally, the last technique implemented is sparse multiplications. It consists in avoiding carrying out the explicit computation when a value in a matrix is 0 or close to it. Sparse or light matrices can be obtained by applying regularization during training, by means, for example, of L2 or L1 norms. This helps reduce the latency since it reduces the amount of multiply-accumulate (MAC) operations. In the case of [130], authors achieve a speed-up of 3.93x by applying sparse convolution with only a 1.79 % drop in mIoU (mean intersection over union). Another important point in this paper is the description of the process for embedding a deep learning network in an embedded system. Usually, only inference is carried out in Resource-Constrained devices. Hence, the usual pipeline is to train the network in a suited environment, such as a personal computer with a GPU. Then convert the model to an appropriate format for its embedding (for example with Tensorflow Lite [99] or Glow [133]) and finally, use a library for calling the adapted graph and infer the results (for example the Texas Instruments Deep learning library -TIDL-).

Another possibility is to apply transfer learning from a bigger network to a smaller one. In ()[134], authors apply transfer learning to obtain a network of equivalent accuracy but with only 0.5% memory footprint for weights. They find that using Softmax probabilities to generate the loss for the smaller network or target network improves results compared to using the labels. They also find that training the network with a Balanced Gradient Contribution (BGC) improves their results. This latter case implies taking into account, in the computation loss, the different frequencies of appearance for each of the segmented classes, related with the problem of data scarcity for semantic segmentation. They prove that training a network with a mix of densely and sparsely annotated data altogether with a weighting loss strategy helps solving the diverging nature of training when dealing with multi-domain differently annotated data.

Another way to embed CNNs is to downgrade the computational cost of operations such as convolutional layers. Using depth-wise convolutions, as introduced by [5], to produce lightweight CNNs, the authors in [135] are able to embed a SDD [41] (modified version) devoted to multi-scale object identification. The main point of depth-wise separable convolution is to divide the standard convolution in two distinct operations. The first one applies a filter to each channel without changing them while the second applies a point-wise convolution to combine the output of the previous operation. Formula 1 shows the relationship between computational complexity of standard convolutions and depth-wise separable convolution, where N is the number of output channels and D_k is the kernel size of filters. The formula refers to the relative complexity of depth-wise convolutions with regard to standard convolutions. As observed, computational complexity is reduced.

$$\frac{1}{N} + \frac{1}{D_k^2} \tag{1}$$

In [136], authors develop a network devoted to lane position and orientation and suited for embedding. They embed it in a NVIDIA Drive PX and achieve 100 fps with 240 × 260 resolution images. It is outstanding that the network is really small for a vision task: two stacks of convolutions, followed by normalization and pooling, and two stacks of fully connected layers followed by drop-out. They use images from lateral cameras and artificially generated images. In [137], they propose a segmentation and detection multi-task system.

TABLE 9. Information for road detection and scene understanding: task, dataset, embedding strategy, the hardware, performance, image size and latency.

Article	Task	Dataset	Embedding strategy	Hardware	Performance	Image size	Latency (fps)
[130]	Finding drivable space	Cityscapes	Quantized inference & layer fusion & sparse CNN	TI TDA3x SOC	95.90 % MIOU	1024x512	20
[138]	Road detection	Cityscapes & KITTI	Hardware mapping	0.45mm^2 ASIC 32nm	91.60 mAP	1080P	241
[136]	Lane position and orientation	Own	Caffee	NVIDIA Drive PX	89.32% accuracy	240x360	100
[137]	Object and drivable terrain detection	KITTI	/	NVIDIA Jetson TX2	82.1 % mAP	1392x512	5.32
[134]	Road scene semantic segmentation	MDRS3&KITTI& RMPTMP&GTSRB&GSV	Knowledge transfer	/	71.8% mean per class accuracy	240x180	/
[135]	Multi-scale object identification	Own	Depth-Wise Conv	Google Pixel	60 % precision detection	/	2
[139]	Lane Change Detection	Own	Direct SVM	Google Nexus4, Nubia Z7 Max	96.4% Accuracy	/	/
[140]	Lane Change Detection	Own	Specialized Library (FANN [142])	Raspberry Pi 3 Model B	53 % RMSE	/	18.06 ms

The key of their approach, in order to achieve lower latency than competitors, is to use a backbone which gives shared features at different scales. The general structure consists of a shared encoder and specific task decoders. The encoder is based on Inception V-2 and the decoders are, for segmentation, a FCN [141] derived network, and for detection, a SSD based network. With this shared framework, authors achieve a latency of 5.32 fps with the standard KITTI dataset resolution on a NVIDIA Jetson TX2. It has to be noted that this result is far from real-time processing.

In [138], authors map a CNN into a chip area of 0.45mm^2 and the energy consumption of only 80mW and 241 fps throughput at 1080p. The CNN is devoted to road segmentation (the road is the only part of the image segmented). Their implementation has two main characteristics that help improving their results and mapping the network to hardware. First, as inputs, together with images they add 2 extra feature channels related to row coordinate and column coordinate, since road is found usually in the lower part of the image. And second, they use the same number of neurons at every layer to be able to reuse the same hardware for all layers.

Another task inside road understanding is lane change detection, which in fact is a useful task for semi-autonomous cars. It helps the system to know, altogether with scene understanding, where the car is and when changes its position. In [139], authors detect lane changes of the car driven using accelerometer and gyroscope sensors altogether with a pre-trained SVM. The implementation uses SVMLIB [94] and it is implemented in two smartphones: Google Nexus 4 / Nubia Z7 Max. In this case, it is interesting to analyze how they use a server together with GPS to provide better resolution of the localization of the car with constrained K-means clustering, establishing a mixed embedded plus cloud-based solution. Another application to lane change detection is found in [140]. They detect roll angles and lane changes using a Raspberry Pi 3 model B together with a fully connected artificial network implemented with the FANN [142] library. For detecting changes, they use both acceleration values and angle change rates.

1) PARTIAL CONLUSIONS

Road detection and scene understanding applications, found in 8 articles, is mainly carried out with deep learning techniques such as CNN with their corresponding computational burden. Among the techniques applied to embed CNNs in resource-constrained devices to achieve real-time performance stand out quantization, layer fusion, convolution optimization and knowledge transfer. As for implementation, NVIDIA Jetson boards and application-specific hardware achieve better performance in both latency and energy while losing adaptability.

For this task, certain homogeneity exists among articles regarding the dataset and performance metrics. This fact allows a better cross comparison between articles in terms of their network, hardware or embedding strategy. Table 8 shows the selected articles and related data regarding performance and implementation characteristics.

E. MISCELLANEOUS

Previously addressed ADAS tasks correspond to well defined problems. However, other 5 articles comply with the requirements and constraints of the review but do not belongto any of those specific and established topics. These articles are summarized in this section by commenting each article individually.

One important aspect of deep learning applications for automotive systems is functional safety. ISO26262 establishes requisites and guidelines but does not specify software requirements for machine learning applications [143] that are relevant for its real-case application. In [144], authors study the robustness of a lightweight application devoted to road segmentation. They focus on code and control flow errors but also on data processing errors. They also

TABLE 10. **Algorithm and data information for the miscellaneous section: task, classifier, dataset, preprocessing and performance.**

Article	Task	Classification Model	Preprocessing	Dataset	Performance
[145]	Gesture recognition	FSM	Cultural Alg.	Own	77% Accuracy
[146]	Word spotting	HMM	Spectral subtraction	Own	93% Accuracy
[147]	Empty Space detection (Parking)	CNN	/	PKLot & CNRPark	99% Accuracy
[148]	Parking Line Warning	CNN	/	Own	99.13 % recall
[149]	Parking Maneuver Assist	GRBF	EKF SLAM	Own	6% out of bag error

TABLE 11. **Implementation information for the miscellaneous section: platform, hardware used, language and libraries, classes and the speed achieved.**

Article	Platform	Hardware	Libraries/ Languages	Image Size	Object classes	Speed (fps)
[145]	Smartwatch	Sony Smartwatch 3 axis	F#/xamarin toolset	/	6 Wrist gestures	/
[146]	Embedded baord	SuperH RISC Engine	/	/	1856 word vocabulary	38.1 ms
[147]	Embedded Board	NVIDIA Jetson TX1	Tensorflow	256x256	Vehicle/Not vehicle	142 fps
[148]	Embedded board	NVIDIA Jetson TX2	Keras	224x224	Line segmentation	5 fps
[149]	Embedded board	ODROID	C++	/	Maneuvers definition	/

investigate the effects of numerical representation, concluding that fixed-point numbers offer more robustness towards faults because they do not compress numbers around 0. Finally, they also investigate the effect of faults depending on the layer where they occur. The conclusion is that few studies have been carried out on fault injection for machine learning applications and that regulatory constraints should incorporate these methodologies in order to bring standardization into place.

One topic that has not appeared along this review is HMI (Human Machine Interface). Automotive industry is changing the way in which the user interacts with the car: from push and rotational buttons to smart devices. In [145], authors develop a gesture recognition application based on data coming from smartwatch sensors (gyroscope and accelerometer). The recognized gestures include twisting, swiping, tapping and others. Their purpose is to navigate through a hierarchical menu with two special gestures defined to enter and exit the gesture recognition mode. This helps to reduce battery consumption since only the specific sensors are used when the system looks for a gesture. To identify pre-defined gestures they use Finite State Machines and Cultural Algorithms. The use of FSMs has the advantage of having linear time complexity compared to other algorithms such as HMM, DTW or NN and also, that the number of variables does not increment exponentially as, for example, in HMM.

Following these lines, as classical mechanical buttons are getting off automotive surfaces and voice and gesture recognition technologies are in place, it has been a surprise not finding many articles devoted to voice recognition embedded in automotive systems. The reason might be that voice recognition studies are not focused directly into automotive contexts but the opposite, automotive is seen as a specific application and studies focus in other, easier to manage, contexts. Another probable cause might be that embedding machine learning models for voice recognition is considered not enough interesting since a good connection to a server can be found in most automotive situations. The most probable reason, however, is a conjunction of the previous given causes: studies covering speech recognition to embedded devices in the automotive context are not common. In this review, only one article has been found regarding embedded voice recognition for automotive systems, [146], and dates back to 2002. In it, authors apply Spectral Subtraction (SS) for removing noise in the signal and semi-continuous Hidden Markov Models for the recognition of different words. To reduce latency and maintain accuracy, they implement different strategies such as converting SS to fixed-point computations and transforming Gaussian distributions to partial spaces represented by small numbers of code words.

Looking for a place to park is a common time-wasting activity in big cities. Hence, it surprises that parking assistance has not appeared profusely in the articles hit by the review. Smart parking solutions reduce the time spent seeking a parking space and are one of the requirements for smart city plans since it would reduce traffic density and pollution, and increase welfare of drivers. In [147], they propose a network of cameras for parking space detection linked to a server and a web application (stacked into goggle maps) being able to indicate free parking spaces nearby. Detection of free parking spaces is carried out with a NVIDIA Jetson TX1 board, a custom light CNN and two datasets PKLot [150] and CNRPark [151] achieving good results. 99% accuracy and 142 fps. In [148], they implement a Parking Line Warning System based on VGG-16 [152] and a NVIDIA Jetson TX2 by using 4 mounted cameras as an AVM (Around View Monitoring).

Another topic related to parking would be maneuvering assistance during parking. In [149], they build an autonomous maneuvering system for parking. They use vehicle dynamics based on Extended Kalman filters and SLAM to provide feature vectors for detecting segmentation points of the movement during parking. To identify such points they use a General Radial Basis Function (GRBF) classifier that, due to its fuzzy-like interpretability and lightweight structure, is a good choice for Resource-Constrained devices. The summary of key points of the articles in this last section is found in Tables 10 and 11.

V. CONCLUSION

This study performs a systematic review about Resource-Constrained machine learning applications for ADAS. The originally proposed question is *How a Machine Learning model can be implemented into an embedded or Resource-Constrained platform devoted to ADAS tasks?* Covering and to answer this question. At the same time, we have also tried to answer some of the secondary questions that stemmed from the main question: which type of models were used, how were they modified to fit in every selected hardware platform from a diversity of them, among others.

The review and collection of information has been conducted satisfactorily and structured in five major ADAS tasks: (1) vehicle and pedestrian detection; (2) driver's state, behavior and identification; (3) traffic sign recognition; (4) road segmentation and understanding and (5) miscellaneous. From the initial search that obtained 1839 research papers accessed, we end up with 23, 14, 14, 8 and 5 papers respectively for the previous tasks relevant for ML and embedded solutions.

The cross comparison between studies that would be the next step of a systematic review, has been difficult in some cases and failed in most of them. The main difficulty is that, unlike medical studies, there is not a common or shared method for carrying out the study. For example, if the purpose is to measure the improvement of a certain type of quantization or compression method and compare it to other methods, the ideal situation would be to have all the other variables unchanged (ceteris paribus). But this is not the common situation where the dataset, hardware, labelling and classification objects change among studies

Nevertheless, this impossibility to conduct a fair cross comparison among articles it is not an impediment to obtain satisfactory information regarding the primary and secondary questions. As a whole, we have seen how researchers embed machine learning models into Resource-Constrained platforms and devoted to ADAS tasks, which was our original intention with this review. The conclusion is that there are two key decisions.

The first one is model modification or selection. SVM is often chosen because, once trained, is a lightweight model and does not need any modification. Other deep learning models, such as CNNs, represent more computing demanding models and several techniques appeared to adapt these networks to resource-constrained devices: quantization, layer fusion, model compression and pruning among others.

The second decision refers to the implementation platform. In this case, three main groups appeared: embedded boards, ASIC/FPGAs and smartphones. Regarding the embedded boards, stand out those which have an internal GPU, such as the NVIDIA Jetson or Tegra family. This eases the deployment and adaptation of deep learning models.

Regarding ASICs and FPGAs, they offer the possibility to map the network to the hardware thus improving both speed and energy savings. Last, there has been an increase in the usage of smartphones for ADAS applications since they use low-power embedded devices (that have often related families for the embedded domain). Although smartphones are not precisely scarce resource, models such as deep neural networks still need to be adapted. For this reason, it has been useful to consider them in the review, since many of the techniques previously commented apply for this case.

Lastly, it has been possible to see how researchers often structure their algorithms: preprocessing, ML model and application-specific task. Preprocessing is a highly image- or sensor-dependent step as well as the dataset selection. For example, in the case of traffic light recognition, color segmentation is recursively used while in drivers' state, wearables are mainly used. When cameras are used, a preprocessing step is added to reduce image size and most commonly used models are SVM or CNN.

Some final recommendations from this review, oriented to improve faster and clearer research on ML for ADAS, refer to the need to: first, establish common grounds for same tasks to get fair cross comparison and to avoid misguiding results, and second, report speed and energy for embedded implementations.

REFERENCES

[1] *Annual Accident Report 2018*, European Commission, Eur. Road Saf. Observatory, Brussels, Belgium, 2018.

[2] Y. LeCun, L. D. Jackel, L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. A. Müller, E. Säckinger, P. Simard, and V. Vapnik, "Learning algorithms for classification: A comparison on handwritten digit recognition," *Neural Netw., Stat. Mech. Perspective*, vol. 261, p. 276, Jan. 1995.

[3] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, Dec. 1989.

[4] J. S. Shin, U. T. Kim, D. K. Lee, S. J. Park, S. J. Oh, and T. J. Yun, "Real-time vehicle detection using deep learning scheme on embedded system," in *Proc. Int. Conf. Ubiquitous Future Netw. (ICUFN)*, 2017, vol. 3, no. 1, pp. 272–274.

[5] 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*. [Online]. Available: http://arxiv.org/abs/1704.04861

[6] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2704–2713.

[7] I. Fedorov, R. P. Adams, M. Mattina, and P. N. Whatmough, "SpArSe: Sparse architecture search for CNNs on resource-constrained micro-controllers," 2019, *arXiv:1905.12107*. [Online]. Available: http://arxiv.org/abs/1905.12107

[8] C. Louizos, K. Ullrich, and M. Welling, "Bayesian compression for deep learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 3288–3298.

[9] P.-H. Pham, D. Jelaca, C. Farabet, B. Martini, Y. LeCun, and E. Culurciello, "NeuFlow: Dataflow vision processing system-on-a-chip," in *Proc. IEEE 55th Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Aug. 2012, pp. 1044–1047.

[10] L. Cavigelli and L. Benini, "Origami: A 803-GOp/s/W convolutional network accelerator," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 27, no. 11, pp. 2461–2475, Nov. 2017.

[11] C. Badue, R. Guidolini, R. V. Carneiro, P. Azevedo, V. B. Cardoso, A. Forechi, L. Jesus, R. Berriel, T. Paixão, F. Mutz, L. Veronese, T. Oliveira-Santos, and A. F. De Souza, "Self-driving cars: A survey," 2019, *arXiv:1901.04407*. [Online]. Available: http://arxiv.org/abs/1901.04407

[12] E. Mitleton-Kelly, I. Deschenaux, C. Maag, M. Fullerton, and N. Celikkaya, "Enhancing crowd evacuation and traffic management through AmI technologies: A review of the literature," in *Co-Evolution of Intelligent Socio-Technical Systems*. Berlin, Germany: Springer, 2013, pp. 19–41.

[13] S. Tuohy, M. Glavin, C. Hughes, E. Jones, M. Trivedi, and L. Kilmartin, "Intra-vehicle networks: A review," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 2, pp. 534–545, Apr. 2015.

[14] K. Bengler, K. Dietmayer, B. Farber, M. Maurer, C. Stiller, and H. Winner, "Three decades of driver assistance systems: Review and future perspectives," *IEEE Intell. Transp. Syst. Mag.*, vol. 6, no. 4, pp. 6–22, 2014.

[15] A. Ziebinski, R. Cupek, D. Grzechca, and L. Chruszczyk, "Review of advanced driver assistance systems (ADAS)," *AIP Conf.*, vol. 1906, no. 1, 2017, Art. no. 120002.

[16] A. Ziebinski, R. Cupek, H. Erdogan, and S. Waechter, "A survey of ADAS technologies for the future perspective of sensor fusion," in *Proc. Int. Conf. Comput. Collective Intell.*, 2016, pp. 135–146.

[17] R. Okuda, Y. Kajiwara, and K. Terashima, "A survey of technical trend of ADAS and autonomous driving," in *Proc. Int. Symp. VLSI Technol., Syst. Appl. (VLSI-TSA)*, Apr. 2014, pp. 1–4.

[18] P. Hurney, F. Morgan, M. Glavin, E. Jones, and P. Waldron, "Review of pedestrian detection techniques in automotive far-infrared video," *IET Intell. Transp. Syst.*, vol. 9, no. 8, pp. 824–832, Oct. 2015.

[19] J. J. Antony and M. Suchetha, "Vision based vehicle detection: A literature review," *Int. J. Appl. Eng. Res.*, vol. 11, no. 5, pp. 3128–3133, 2016.

[20] S. Wali, "Comparative survey on traffic sign detection and recognition: A review," *Przegląd Elektrotechniczny*, vol. 1, no. 12, pp. 40–44, Dec. 2015.

[21] L. Xiao and F. Gao, "A comprehensive review of the development of adaptive cruise control systems," *Vehicle Syst. Dyn.*, vol. 48, no. 10, pp. 1167–1192, Oct. 2010.

[22] T. G. Dietterich, "Machine learning for sequential data: A review," in *Proc. Joint IAPR Int. Workshops Stat. Techn. Pattern Recognit. (SPR), Struct. Syntactic Pattern Recognit. (SSPR)*, 2002, pp. 15–30.

[23] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," *Emerg. Artif. Intell. Appl. Comput. Eng.*, vol. 160, pp. 3–24, Jun. 2007.

[24] A. Moujahid, M. ElAraki Tantaoui, M. D. Hina, A. Soukane, A. Ortalda, A. ElKhadimi, and A. Ramdane-Cherif, "Machine learning techniques in ADAS: A review," in *Proc. Int. Conf. Adv. Comput. Commun. Eng. (ICACCE)*, Jun. 2018, pp. 235–242.

[25] A. M. Methley, S. Campbell, C. Chew-Graham, R. McNally, and S. Cheraghi-Sohi, "PICO, PICOS and SPIDER: A comparison study of specificity and sensitivity in three search tools for qualitative systematic reviews," *BMC Health Services Res.*, vol. 14, no. 1, p. 579, Nov. 2014.

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[27] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," M.S. thesis, Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, 2009.

[28] 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.

[29] A. Suleiman and V. Sze, "An energy-efficient hardware implementation of HOG-based object detection at 1080HD 60 fps with multi-scale support," *J. Signal Process. Syst.*, vol. 84, no. 3, pp. 325–337, 2016.

[30] C. Yi, Q. Sheng, Q. Zhou, and H. Wang, "Pedestrian detection based on Visconti2 7502," in *Proc. IEEE 17th Int. Conf. Commun. Technol. (ICCT)*, Oct. 2017, pp. 1513–1518.

[31] S. Martelli, D. Tosato, M. Cristani, and V. Murino, "Fast FPGA-based architecture for pedestrian detection based on covariance matrices," in *Proc. 18th IEEE Int. Conf. Image Process.*, Sep. 2011, pp. 389–392.

[32] B. Meus, T. Kryjak, and M. Gorgon, "Embedded vision system for pedestrian detection based on HOG+SVM and use of motion information implemented in Zynq heterogeneous device," in *Proc. Signal Process., Algorithms, Archit., Arrangements, Appl. (SPA)*, Sep. 2017, pp. 406–411.

[33] C.-C. Lin, C.-W. Lin, D.-C. Huang, and Y.-H. Chen, "Design a support vector machine-based intelligent system for vehicle driving safety warning," in *Proc. 11th Int. IEEE Conf. Intell. Transp. Syst. (ITSC)*, Oct. 2008, pp. 938–943.

[34] B. Furht, J. Greenberg, and R. Westwater, *Motion Estimation Algorithms for Video Compression*, vol. 379. Berlin, Germany: Springer, 2012.

[35] Y.-M. Tsai, T.-J. Yang, C.-C. Tsai, K.-Y. Huang, and L.-G. Chen, "A 69 mW 140-meter/60 fps and 60-meter/300 fps intelligent vision SoC for versatile automotive applications," in *IEEE Symp. VLSI Circuits, Dig. Tech. Paper*, Jun. 2012, pp. 152–153.

[36] R. Verbickas, R. Laganiere, D. Laroche, C. Zhu, X. Xu, and A. Ors, "SqueezeMap: Fast pedestrian detection on a low-power automotive processor using efficient convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jul. 2017, pp. 463–471.

[37] 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 <0.5 MB model size," 2016, *arXiv:1602.07360*. [Online]. Available: https://arxiv.org/abs/1602.07360

[38] D. Tome, L. Bondi, L. Baroffio, S. Tubaro, E. Plebani, and D. Pau, "Reduced memory region based deep convolutional neural network detection," in *Proc. IEEE 6th Int. Conf. Consum. Electron.-Berlin (ICCE-Berlin)*, Sep. 2016, pp. 15–19.

[39] D. Tomè, F. Monti, L. Baroffio, L. Bondi, M. Tagliasacchi, and S. Tubaro, "Deep convolutional neural networks for pedestrian detection," *Signal Process., Image Commun.*, vol. 47, pp. 482–489, Sep. 2016.

[40] H. E. Kim, Y. Lee, H. Kim, and X. Cui, "Domain-specific data augmentation for on-road object detection based on a deep neural network," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2017, pp. 103–108.

[41] W. Liu, "SSD: Single shot multibox detector wei," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 21–37.

[42] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *Proc. Brit. Mach. Vis. Conf.*, 2016, pp. 87.1–87.12.

[43] A. Kozlov and D. Osokin, "Development of real-time ADAS object detector for deployment on CPU," in *Proc. SAI Intell. Syst. Conf.*, 2019, pp. 740–750.

[44] Z. Shen, Z. Liu, J. Li, Y.-G. Jiang, Y. Chen, and X. Xue, "DSOD: Learning deeply supervised object detectors from scratch," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 1937–1945.

[45] K. J. Lee, K. Bong, C. Kim, J. Jang, H. Kim, J. Lee, K.-R. Lee, G. Kim, and H.-J. Yoo, "14.2 a 502GOPS and 0.984 mW dual-mode ADAS SoC with RNN-FIS engine for intention prediction in automotive black-box system," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, vol. 1, Jan. 2016, pp. 256–257.

[46] NXP. *S32V234: Vision Processor for Front and Surround View Camera, Machine Learning and Sensor Fusion*. [Online]. Available: https://www.nxp.com/products/processors-and-microcontrollers/arm-processors/s32v2-vision-mpus-/vision-processor-for-front-and-surround-view-camera-machine-learning-and-sensor-fusion:S32V234

[47] B. Wu, A. Wan, F. Iandola, P. H. Jin, and K. Keutzer, "SqueezeDet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jul. 2017, pp. 446–454.

[48] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017.

[49] B. Kim, Y. Jeon, H. Park, D. Han, and Y. Baek, "Design and implementation of the vehicular camera system using deep neural network compression," in *Proc. 1st Int. Workshop Deep Learn. Mobile Syst. Appl. (EMDL)*, vol. 17, 2017, pp. 25–30.

[50] F.-A. Chang, C.-C. Tsai, C.-K. Tseng, and J.-I. Guo, "Embedded multiple object detection based on deep learning technique for advanced driver assistance system," in *Proc. IEEE 60th Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Aug. 2017, pp. 172–175.

[51] Intel. *OpenVINO Toolkit*. Accessed: Oct. 21, 2019. [Online]. Available: https://software.intel.com/en-us/openvino-toolkit

[52] *NVIDIA TensorRT*. Accessed: Oct. 21, 2019. [Online]. Available: https://developer.nvidia.com/tensorrt

[53] J. Xu, P. Wang, H. Yang, and A. M. López, "Training a binary weight object detector by knowledge transfer for autonomous driving," 2018, *arXiv:1804.06332*. [Online]. Available: http://arxiv.org/abs/1804.06332

[54] J. Redmon. (2013). *Darknet: Open Source Neural Networks in C*. Accessed: Oct. 21, 2019. [Online]. Available: http://pjreddie.com/darknet/

[55] E. Alba, D. Anguita, A. Ghio, and S. Ridella, "Using variable neighborhood search to improve the support vector machine performance in embedded automotive applications," in *Proc. IEEE Int. Joint Conf. Neural Netw., IEEE World Congr. Comput. Intell.*, Jun. 2008, pp. 984–988.

[56] D. Nazir, M. Fizza, A. Waseem, and S. Khan, "Vehicle detection on embedded single board computers," in *Proc. 7th Int. Conf. Comput. Commun. Eng. (ICCCE)*, Sep. 2018, pp. 480–485.

[57] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2005, pp. 886–893.

[58] S. Kim, S. Lee, and K. Cho, "Design of high-speed support vector machine circuit for driver assistance system," in *Proc. Int. SoC Design Conf. (ISOCC)*, Nov. 2012, pp. 45–48.

[59] D. Anguita, A. Ghio, S. Pischiutta, and S. Ridella, "A hardware-friendly support vector machine for embedded automotive applications," in *Proc. Int. Joint Conf. Neural Netw.*, Aug. 2007, pp. 1360–1364.

[60] G. Brilli, P. Burgio, and M. Bertogna, "Convolutional neural networks on embedded automotive platforms: A qualitative comparison," in *Proc. Int. Conf. High Perform. Comput. Simulation (HPCS)*, Jul. 2018, pp. 496–499.

[61] Y. Kim, M. Imani, and T. Rosing, "ORCHARD: Visual object recognition accelerator based on approximate in-memory processing," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des. (ICCAD)*, Nov. 2017, pp. 25–32.

[62] Y. Koo, C. You, and S. Kim, "OpenCL-darknet: An OpenCL implementation for object detection," in *Proc. IEEE Int. Conf. Big Data Smart Comput. (BigComp)*, Jan. 2018, pp. 631–634.

[63] A. Ghio and S. Pischiutta, "A support vector machine based pedestrian recognition system on resource-limited hardware architectures," in *Proc. Ph.D Res. Microelectron. Electron. Conf.*, no. 3, Jul. 2007, pp. 161–163.

[64] A. Ess, B. Leibe, and L. Van Gool, "Depth and appearance for mobile scene analysis," in *Proc. IEEE 11th Int. Conf. Comput. Vis.*, Oct. 2007, pp. 1–8.

[65] D. Wang, K. Xu, and D. Jiang, "PipeCNN: An OpenCL-based open-source FPGA accelerator for convolution neural networks," in *Proc. Int. Conf. Field Program. Technol. (ICFPT)*, Dec. 2017, pp. 279–282.

[66] XLINX Inc. (2018). *xfDNN*. Accessed: Oct. 21, 2019. [Online]. Available: https://www.xilinx.com/support/documentation/white_papers/wp504-accel-dnns.pdf

[67] *Traffic Safety Facts*, U.S.D. Transp.-Nat. Highway Traffic Saf. Admin., Washington, DC, USA, 2016.

[68] *Distracted Driving*, U.S.D. Transp.-Nat. Highway Traffic Saf. Admin., Washington, DC, USA, 2016.

[69] L. Boon-Leng, L. Dae-Seok, and L. Boon-Giin, "Mobile-based wearable-type of driver fatigue detection by GSR and EMG," in *Proc. IEEE Region Conf. (TENCON)*, Nov. 2015, pp. 1–4.

[70] A. Lourenço, A. P. Alves, C. Carreiras, R. P. Duarte, and A. Fre, "CardioWheel: ECG biometrics on the steering wheel," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*. Cham, Switzerland: Springer, Sep. 2015, pp. 267–270.

[71] X. Xu, J. Yu, Y. Chen, Y. Zhu, S. Qian, and M. Li, "Leveraging audio signals for early recognition of inattentive driving with smartphones," *IEEE Trans. Mobile Comput.*, vol. 17, no. 7, pp. 1553–1567, Jul. 2018.

[72] B.-F. Wu, Y.-H. Chen, C.-H. Yeh, and Y.-F. Li, "Reasoning-based framework for driving safety monitoring using driving event recognition," *IEEE Trans. Intell. Transp. Syst.*, vol. 14, no. 3, pp. 1231–1241, Sep. 2013.

[73] D. Tran, H. Manh Do, W. Sheng, H. Bai, and G. Chowdhary, "Real-time detection of distracted driving based on deep learning," *IET Intell. Transp. Syst.*, vol. 12, no. 10, pp. 1210–1219, Dec. 2018.

[74] C.-T. Lin, C.-H. Chuang, C.-S. Huang, S.-F. Tsai, S.-W. Lu, Y.-H. Chen, and L.-W. Ko, "Wireless and wearable EEG system for evaluating driver vigilance," *IEEE Trans. Biomed. Circuits Syst.*, vol. 8, no. 2, pp. 165–176, Apr. 2014.

[75] B. G. Lee, T. W. Chong, B. L. Lee, H. J. Park, Y. N. Kim, and B. Kim, "Wearable mobile-based emotional response-monitoring system for drivers," *IEEE Trans. Human-Machine Syst.*, vol. 47, no. 5, pp. 636–649, Oct. 2017.

[76] G. Li, B.-L. Lee, and W.-Y. Chung, "Smartwatch-based wearable EEG system for driver drowsiness detection," *IEEE Sensors J.*, vol. 15, no. 12, pp. 7169–7180, Dec. 2015.

[77] J. C. Platt, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," in *Advances in Large Margin Classifiers*. Cambridge, MA, USA: MIT Press, 1999, pp. 61–74.

[78] G. Borghi, R. Gasparini, R. Vezzani, and R. Cucchiara, "Embedded recurrent network for head pose estimation in car," in *Proc. 4th IEEE Intell. Vehicles Symp.*, Jun. 2017, pp. 1503–1508.

[79] G. Fanelli, M. Dantone, J. Gall, A. Fossati, and L. Van Gool, "Random forests for real time 3D face analysis," *Int. J. Comput. Vis.*, vol. 101, no. 3, pp. 437–458, Aug. 2012.

[80] M. Venturelli, G. Borghi, R. Vezzani, and R. Cucchiara, "From depth data to head pose estimation: A siamese approach," 2017, *arXiv:1703.03624*. [Online]. Available: http://arxiv.org/abs/1703.03624

[81] A. Saeed and A. Al-Hamadi, "Boosted human head pose estimation using kinect camera," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2015, pp. 1752–1756.

[82] Z. Xia, W. Zhang, F. Tan, X. Feng, and A. Hadid, "An accurate eye localization approach for smart embedded system," in *Proc. 6th Int. Conf. Image Process. Theory, Tools Appl. (IPTA)*, Dec. 2016, pp. 1–5.

[83] K. Selvakumar, J. Jerome, K. Rajamani, and N. Shankar, "Real-time vision based driver drowsiness detection using partial least squares analysis," *J. Signal Process. Syst.*, vol. 85, no. 2, pp. 263–274, Dec. 2015.

[84] Carnetsoft. (2004). *CarnetSoft. Driving Simulators*. Accessed: Oct. 21, 2019. [Online]. Available: https://cs-driving-simulator.com

[85] B. Reddy, Y.-H. Kim, S. Yun, C. Seo, and J. Jang, "Real-time driver drowsiness detection for embedded system using model compression of deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jul. 2017, pp. 438–445.

[86] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," *IEEE Signal Process. Lett.*, vol. 23, no. 10, pp. 1499–1503, Oct. 2016.

[87] I. del Campo, R. Finker, M. V. Martinez, J. Echanobe, and F. Doctor, "A real-time driver identification system based on artificial neural networks and cepstral analysis," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2014, pp. 1848–1855.

[88] A. A. Putilov and O. G. Donskaya, "Construction and validation of the EEG analogues of the karolinska sleepiness scale based on the karolinska drowsiness test," *Clin. Neurophysiol.*, vol. 124, no. 7, pp. 1346–1352, Jul. 2013.

[89] K.-C. Lee, J. Ho, and D. J. Kriegman, "Acquiring linear subspaces for face recognition under variable lighting," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 5, pp. 684–698, May 2005.

[90] A. R. Martinez and R. Benavente, "The AR face database," Comput. Vis. Center, Barcelona, Spain, Tech. Rep. 24, 1999, vol. 24.

[91] F. S. Samaria and A. C. Harter, "Parameterisation of a stochastic model for human face identification," in *Proc. IEEE Workshop Appl. Comput. Vis.*, Dec. 1994, pp. 138–142.

[92] X. Xu, S. Yin, and P. Ouyang, "Fast and low-power behavior analysis on vehicles using smartphones," in *Proc. 6th Int. Symp. Next Gener. Electron. (ISNE)*, May 2017, pp. 1–4.

[93] M. García-García, A. Caplier, and M. Rombaut, "Sleep deprivation detection for real-time driver monitoring using deep learning," in *Proc. Int. Conf. Image Anal. Recognit.*, in Lecture Notes in Computer Science, vol. 10882, 2018, pp. 435–442.

[94] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, pp. 27:1–27:27, Apr. 2011.

[95] L.-W. Ko, W.-K. Lai, W.-G. Liang, C.-H. Chuang, S.-W. Lu, Y.-C. Lu, T.-Y. Hsiung, H.-H. Wu, and C.-T. Lin, "Single channel wireless EEG device for real-time fatigue level detection," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2015, pp. 1–5.

[96] A. Samir, B. A. Mohamed, and B. A. Abdelhakim, "Detection of driver drowsiness based on the Viola & Jones method and logistic regression analysis," in *Proc. Medit. Symp. Smart City Appl. (SCAMS)*, vol. 17, 2017, pp. 1–6.

[97] M. Abadi, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2016, pp. 265–283.

[98] T. Theano Development Team *et al.*, "Theano: A Python framework for fast computation of mathematical expressions," 2016, *arXiv:1605.02688*. [Online]. Available: http://arxiv.org/abs/1605.02688

[99] Tensorflow. *Tensorflow Lite*. Accessed: Dec. 1, 2019. [Online]. Available: https://www.tensorflow.org/lite

[100] Z. Zhang, "Microsoft kinect sensor and its effect," *IEEE Multimedia-Mag.*, vol. 19, no. 2, pp. 4–10, Feb. 2012.

[101] R. Alvarez, R. Prabhavalkar, and A. Bakhtin, "On the efficient representation and execution of deep acoustic models," in *Proc. Annu. Conf. Int. Speech Commun. Assoc. (INTERSPEECH)*, vols. 8–12, Sep. 2016, pp. 2746–2750.

[102] W. Liu, S. Li, J. Lv, B. Yu, T. Zhou, H. Yuan, and H. Zhao, "Real-time traffic light recognition based on smartphone platforms," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 27, no. 5, pp. 1118–1131, May 2017.

[103] C.-Y. Tsai, H.-C. Liao, and Y.-C. Feng, "A novel translation, rotation, and scale-invariant shape description method for real-time speed-limit sign recognition," in *Proc. Int. Conf. Adv. Mater. Sci. Eng. (ICAMSE)*, Nov. 2016, pp. 486–488.

[104] M. Boumediene, C. Cudel, M. Basset, and A. Ouamri, "Triangular traffic signs detection based on RSLD algorithm," *Mach. Vis. Appl.*, vol. 24, no. 8, pp. 1721–1732, Aug. 2013.

[105] 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.

[106] 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. (IJCNN)*, Aug. 2013, pp. 1–8.

[107] H. S. Lee and K. Kim, "Simultaneous traffic sign detection and boundary estimation using convolutional neural network," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 5, pp. 1652–1663, May 2018.

[108] R. Belaroussi, P. Foucher, J.-P. Tarel, B. Soheilian, P. Charbonnier, and N. Paparoditis, "Road sign detection in images: A case study," in *Proc. 20th Int. Conf. Pattern Recognit.*, Aug. 2010, pp. 484–488.

[109] Nexar. (2016). *Nexar Traffic Light ChallengeDataset*. Accessed: Oct. 21, 2019. [Online]. Available: https://challenge.getnexar.com/challenge-1

[110] H. Novais and A. R. Fernandes, "Community based repository for geo-referenced traffic signs," in *Proc. Encontro Português Computação Gráfica Interação (EPCGI)*, Oct. 2017, pp. 1–8.

[111] T. Q. Vinh, "Real-time traffic sign detection and recognition system based on friendlyARM Tiny4412 board," in *Proc. Int. Conf. Commun., Manage. Telecommun. (ComManTel)*, Dec. 2015, pp. 142–146.

[112] H.-M. Weng and C.-T. Chiu, "Resource efficient hardware implementation for real-time traffic sign recognition," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, Apr. 2018, pp. 1120–1124.

[113] Y. Zhou, Z. Chen, and X. Huang, "A system-on-chip FPGA design for real-time traffic signal recognition system," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2016, pp. 1778–1781.

[114] P.-C. Shih, C.-Y. Tsai, and C.-F. Hsu, "An efficient automatic traffic sign detection and recognition method for smartphones," in *Proc. 10th Int. Congr. Image Signal Process., Biomed. Eng. Informat. (CISP-BMEI)*, Oct. 2017, pp. 1–5.

[115] S. L. Gomes, E. de S. Rebouças, E. C. Neto, J. P. Papa, V. H. C. de Albuquerque, P. P. R. Filho, and J. M. R. S. Tavares, "Embedded real-time speed limit sign recognition using image processing and machine learning techniques," *Neural Comput. Appl.*, vol. 28, no. S1, pp. 573–584, 2017.

[116] S. Jagannathan, K. Desappan, P. Swami, M. Mathew, S. Nagori, K. Chitnis, Y. Marathe, D. Poddar, and S. Narayanan, "Efficient object detection and classification on low power embedded systems," in *Proc. IEEE Int. Conf. Consum. Electron. (ICCE)*, Jan. 2017, pp. 233–234.

[117] C.-Y. Tsai, H.-C. Liao, and K.-J. Hsu, "Real-time embedded implementation of robust speed-limit sign recognition using a novel centroid-to-contour description method," *IET Comput. Vis.*, vol. 11, no. 6, pp. 407–414, Sep. 2017.

[118] J. Matas, O. Chum, M. Urban, and T. Pajdla, "Robust wide-baseline stereo from maximally stable extremal regions," *Image Vis. Comput.*, vol. 22, no. 10, pp. 761–767, Sep. 2004.

[119] S. Liao, X. Zhu, Z. Lei, L. Zhang, and S. Z. Li, "Learning multi-scale block local binary patterns for face recognition," in *Proc. Int. Conf. Biometrics*, 2007, pp. 828–837.

[120] D. Yudin and D. Slavioglo, "Usage of fully convolutional network with clustering for traffic light detection," in *Proc. 7th Medit. Conf. Embedded Comput. (MECO)*, Jun. 2018, vol. 169, nos. 3–4, pp. 1–6.

[121] 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.*, in Lecture Notes in Computer Science, vol. 9351, 2015, pp. 234–241.

[122] M. Ester, H. P. Kriegel, J. Sander, and X. Xu, "Density-based clustering algorithms for discovering clusters," in *Proc. 2nd Int. Conf. Knowl. Discovery Data Mining (KDD)*, Portland, OR, USA, 1996, pp. 2–4.

[123] T. A. Sørensen, "A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on danish commons," *Det Kongelige Danske Videnskabernes Selskab Biologiske Skrifter*, vol. 5, no. 4, p. 42, 1948.

[124] K. Yi, K. Z. Jian, S. Chen, Y. Yang, and N. Zheng, "Knowledge-based recurrent attentive neural network for small object detection," 2018, *arXiv:1803.05263*. [Online]. Available: https://arxiv.org/abs/1803.05263

[125] M. S. Prieto and A. R. Allen, "Using self-organising maps in the detection and recognition of road signs," *Image Vis. Comput.*, vol. 27, no. 6, pp. 673–683, May 2009.

[126] I. C. Lopes, F. Benevenuti, F. L. Kastensmidt, A. A. Susin, and P. Rech, "Reliability analysis on case-study traffic sign convolutional neural network on APSoC," in *Proc. IEEE 19th Latin-Amer. Test Symp. (LATS)*, Mar. 2018, pp. 1–6.

[127] TensorFlow. (2018). *TensorFlow for Microcontrollers*. Accessed: Oct. 21, 2019. [Online]. Available: https://www.tensorflow.org/lite/microcontrollers/overview

[128] ARM. *ARM NN*. Accessed: Oct. 21, 2019. [Online]. Available: https://www.arm.com/products/silicon-ip-cpu/machine-learning/arm-nn

[129] A. Küçükmanisa, G. Tarım, and O. Urhan, "Real-time illumination and shadow invariant lane detection on mobile platform," *J. Real-Time Image Process.*, vol. 16, no. 5, pp. 1781–1794, Apr. 2017.

[130] M. Mody, D. Kumar, P. Swami, M. Mathew, and S. Nagori, "Low cost and power CNN/deep learning solution for automated driving," in *Proc. 19th Int. Symp. Qual. Electron. Design (ISQED)*, Mar. 2018, pp. 432–436.

[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] C. Szegedy, C. S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-ResNet and the impact of residual connections on learning," in *Proc. 31st AAAI Conf. Artif. Intell.* AAAI Press, 2017, pp. 4278–4284.

[133] N. Rotem, J. Fix, S. Abdulrasool, G. Catron, S. Deng, R. Dzhabarov, N. Gibson, J. Hegeman, M. Lele, R. Levenstein, J. Montgomery, B. Maher, S. Nadathur, J. Olesen, J. Park, A. Rakhov, M. Smelyanskiy, and M. Wang, "Glow: Graph lowering compiler techniques for neural networks," 2018, *arXiv:1805.00907*. [Online]. Available: https://arxiv.org/abs/1805.00907

[134] G. Ros, S. Stent, P. F. Alcantarilla, and T. Watanabe, "Training constrained deconvolutional networks for road scene semantic segmentation," 2016, *arXiv:1604.01545*. [Online]. Available: https://arxiv.org/abs/1604.01545

[135] Y. Gu, Q. Wang, and S. Kamijo, "Intelligent driving data recorder in smartphone using deep neural network-based speedometer and scene understanding," *IEEE Sensors J.*, vol. 19, no. 1, pp. 287–296, Jan. 2019.

[136] A. Gurghian, T. Koduri, S. V. Bailur, K. J. Carey, and V. N. Murali, "DeepLanes: End-to-end lane position estimation using deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2016, pp. 38–45.

[137] M. Oeljeklaus, F. Hoffmann, and T. Bertram, "A fast multi-task CNN for spatial understanding of traffic scenes," in *Proc. 21st Int. Conf. Intell. Transp. Syst. (ITSC)*, Nov. 2018, pp. 2825–2830.

[138] Y. Zhou, Y. Lyu, and X. Huang, "RoadNet: An 80-mW hardware accelerator for road detection," *IEEE Embedded Syst. Lett.*, vol. 11, no. 1, pp. 21–24, Mar. 2019.

[139] S. Zhu, X. Wang, Z. Zhang, X. Tian, and X. Wang, "Lane-level vehicular localization utilizing smartphones," in *Proc. IEEE 84th Veh. Technol. Conf. (VTC-Fall)*, Sep. 2016, pp. 1–5.

[140] J. G. Guzmán, L. P. González, J. P. Redondo, M. M. Martínez, and M. L. Boada, "Real-time vehicle roll angle estimation based on neural networks in IoT low-cost devices," *Sensors*, vol. 18, no. 7, pp. 1–21, Jul. 2018.

[141] Y. Zhang, Z. Qiu, T. Yao, D. Liu, and T. Mei, "Fully convolutional adaptation networks for semantic segmentation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 6810–6818.

[142] S. Nissen *et al.*, "Implementation of a fast artificial neural network library (FANN)," Dept. Comput. Sci., Univ. Copenhagen (DIKU), Copenhagen, Denmark, Tech. Rep. 31, 2003, vol. 31, p. 29.

[143] R. Salay, R. Queiroz, and K. Czarnecki, "An analysis of ISO 26262: Using machine learning safely in automotive software," 2017, *arXiv:1709.02435*. [Online]. Available: http://arxiv.org/abs/1709.02435

[144] R. Krutsch and R. Schlagenhaft, "Diagnostic mechanism and robustness of safety relevant automotive deep convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jul. 2017, pp. 378–385.

[145] F. Waris and R. G. Reynolds, "Using cultural algorithms to improve wearable device gesture recognition performance," in *Proc. IEEE Symp. Ser. Comput. Intell.*, Dec. 2015, pp. 625–633.

[146] N. Hataoka, H. Kokubo, Y. Obuchi, and A. Amano, "Compact and robust speech recognition for embedded use on microprocessors," in *Proc. IEEE Workshop Multimedia Signal Process. (MMSP)*, Dec. 2002, pp. 288–291.

[147] H. Bura, N. Lin, N. Kumar, S. Malekar, S. Nagaraj, and K. Liu, "An edge based smart parking solution using camera networks and deep learning," in *Proc. IEEE Int. Conf. Cognit. Comput. (ICCC)*, Jul. 2018, pp. 17–24.

[148] D. Lee, J.-S. Lee, S. Lee, and S.-C. Kee, "The real-time implementation for the parking line departure warning system," in *Proc. 3rd IEEE Int. Conf. Intell. Transp. Eng. (ICITE)*, Sep. 2018, pp. 236–240.

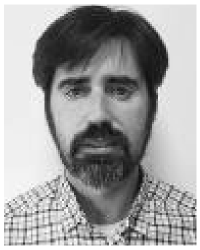
[149] G. Notomista and M. Botsch, "Maneuver segmentation for autonomous parking based on ensemble learning," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2015, pp. 1–8.

[150] P. R. L. Almeida, L. S. Oliveira, A. S. Britto, E. J. Silva, and A. L. Koerich, "PKLot—A robust dataset for parking lot classi cation the PKLot dataset," *Expert Syst. Appl.*, vol. 42, no. 11, pp. 1–6, 2015.

[151] G. Amato, F. Carrara, F. Falchi, C. Gennaro, C. Meghini, and C. Vairo, "Deep learning for decentralized parking lot occupancy detection," *Expert Syst. Appl.*, vol. 72, pp. 327–334, Apr. 2017.

[152] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*. [Online]. Available: http://arxiv.org/abs/1409.1556

JUAN BORREGO-CARAZO was born in Soria, Spain, in 1993. He graduated 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 industrial Ph.D. degree with UAB and KOSTAL Eléctrica, S.A., based in embedding neural networks in resource-constrained devices for gesture recognition. His main interests include deep learning and neural networks, computer vision, and machine learning.



DAVID CASTELLS-RUFAS was born in Manresa, Spain, in 1971. He 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.

Since 2003, he has been working as a Researcher with the CEPHIS Research Group, Microelectronics and Electronic Systems Department, UAB, where he also teaches as a Lecturer. He founded the technology-based companies Histeresys and Creanium, in 1998 and 2001, respectively. His research interests include reconfigurable systems, high performance embedded systems, and computing architectures.

ERNESTO BIEMPICA was born in Barcelona, Spain, in 1973. He graduated in electronic engineering from the Universitat Politècnica de Catalunya (UPC), Spain, in 1997, the M.B.A. degree from the Eada Business School, in 2009, and the M.Sc. degree in micro and manoelectronics from the Universitat Autònoma de Barcelona (UAB), in 2011. He is currently the R&D Manager of KOSTAL Eléctrica, S.A.

JORDI CARRABINA was born in Manresa, Catalonia, in 1963. He graduated in physics from the Universitat Autònoma de Barcelona (UAB), Catalonia, Spain, in 1986, and received the M.S. and Ph.D. degrees in computer science from UAB, in 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 from the Catalan Government Agency ACCIO. He is currently teaching in the B.S. and M.Sc. degrees of telecommunications engineering and computer engineering at UAB, the master's of embedded systems at UPV-EHU, and coordinating the New MsC Degree on IoT for eHealth. His main interests are microelectronic systems oriented to embedded platforms, SoC/NoC architectures, and printed microelectronics.