IEEE Open Journal of
Circuits and Systems

Received 14 February 2022; revised 29 April 2022; accepted 9 May 2022. Date of publication 12 May 2022; date of current version 25 May 2022.

Digital Object Identifier 10.1109/OJCAS.2022.3174632

A Low-Rank CNN Architecture for Real-Time Semantic Segmentation in Visual SLAM Applications

LAURA FALASCHETTI (Member, IEEE), LORENZO MANONI (Graduate Student Member, IEEE), AND CLAUDIO TURCHETTI (Life Member, IEEE)

Department of Information Engineering, Università Politecnica delle Marche, 60131 Ancona, Italy

This article was recommended by Associate Editor A. Akbari.

CORRESPONDING AUTHOR: L. FALASCHETTI (e-mail: l.falaschetti@univpm.it)

This work was supported by the Università Politecnica delle Marche.

ABSTRACT Real-time semantic segmentation on embedded devices has recently enjoyed significant gain in popularity, due to the increasing interest in smart vehicles and smart robots. In particular, with the emergence of autonomous driving, low latency and computation-intensive operations lead to new challenges for vehicles and robots, such as excessive computing power and energy consumption. The aim of this paper is to address semantic segmentation, one of the most critical tasks for the perception of the environment, and its implementation in a low power core, by preserving the required performance of accuracy and low complexity. To reach this goal a low-rank convolutional neural network (CNN) architecture for real-time semantic segmentation is proposed. The main contributions of this paper are: *i)* a tensor decomposition technique has been applied to the kernel of a generic convolutional layer, *ii)* three versions of an optimized architecture, that combines UNet and ResNet models, have been derived to explore the trade-off between model complexity and accuracy, *iii)* the low-rank CNN architectures have been implemented in a Raspberry Pi 4 and NVIDIA Jetson Nano 2 GB embedded platforms, as severe benchmarks to meet the low-power, low-cost requirements, and in the high-cost GPU NVIDIA Tesla P100 PCIe 16 GB to meet the best performance.

INDEX TERMS Embedded systems, semantic segmentation, smart robots, smart vehicles, tensor decomposition.

I. INTRODUCTION

SEMANTIC image segmentation (SIS) is an important task in computer vision that separates an image into several semantically meaningful parts and classifies them into one of the pre-established objects [1]. Given an image, a semantic segmentation algorithm is expected to predict dense labels for all pixels in the image by assigning each image pixel to a category label corresponding to an object.

In the past there has been an active interest for semantic segmentation in several applications including augmented reality [2], image editing [3], medical imaging [4] and significant progress has been made to solve this problem in these fields.

Recently, regarding semantic segmentation as an important task that can help deep understanding of scene, objects and human, a great attention has been devoted to such a task in autonomous driving for smart vehicles and smart robots.

In this context Simultaneous Localization and Mapping (SLAM) [5], [6] is essential for both vehicles and robots to achieve environment detection and autonomous navigation. The core issue of SLAM is to build-up or updating a map in an unknown environment while simultaneously estimate the agent's location within it.

Two different approaches are generally used for SLAM, namely visual SLAM [7] and laser SLAM [8] according to the sensors used to acquire information about the

environment. Visual SLAM utilizes the images captured by a camera, while laser SLAM uses a LIDAR laser scan. Laser SLAM is preferred owing to its higher environmental recognition accuracy, however the intensive computation required, especially for resource-constrained robots, represents a key obstacle for the widespread deployment of such an approach.

To make autonomous driving universally adopted, the major challenge is to simultaneously enable such computation intensive task on a low-power computing system at an affordable price [9], thus visual SLAM is the most suitable solution for this purpose.

Semantic segmentation, that aims at labeling categories at the pixel level of an image, is an established task to understand as much as possible the surrounding scene, and is a fundamental task in visual SLAM since enables to estimate semantic 3D map. However SIS is one of the most critical activities in SLAM, that demands intensive computing operations, thus mainly affecting the low latency requirement in autonomous driving systems. Besides, even though speed of SLAM is a fundamental aspect in car navigation systems (CNS) [10]–[19], cost and low-power restrictions are significant constraints in robot navigation systems (RNS) [10], [20]–[30]. In order to meet those non overlapping performance, different hardware setups are in general adopted for SLAM implementation: GPU (possible low-power) is the right choice for CNS [15], while low-power embedded CPU is the most appropriate solution for RNS [23], [25], [28], [30].

Following the above motivations, this paper focuses on real-time semantic segmentation in visual SLAM and its implementation in low power cores, CPU for RNS and GPU for CNS, by preserving the required performance of accuracy and low-complexity.

A critical aspect of semantic segmentation is that it requires combining dense pixel-level accuracy with multi-scale contextual information. Indeed, at a local level (a few pixels wide), two small image patches belonging to different classes can be misclassified with the same label.

Before the unchallengeable success of convolutional networks (CNNs), most of the semantic segmentation techniques developed in the previous decade, were based on hand-crafted features. Typically, a classifier such as Boosting [31], [32], Random Forest [33] or Support Vector Machines [34], was used to predict the class probability of the center pixel in a patch. Subsequently, a richer information from context was incorporated using many different techniques, e.g., second-order pooling [35], fully connected conditional random fields (CRFs) [36], multi scale CRFs [37], associate hierarchical CRFs [38].

Convolutional Neural Networks (CNNs), initially designed for classification tasks, were subsequently adopted to segmentation [39] by replacing fully connected layers with fully convolutional layers, thus resulting in an encoder-decoder architecture named fully convolutional network (FCN). After this seminal approach, a large number of methods based on the FCN architecture have been proposed [40]–[55].

A fully convolutional network is able to recover the spatial information that is lost in typical classification networks due to the fixed dimension of fully connected layers at the end of these networks. FCN architecture is composed by two sections: the encoder network and the decoder network. The encoder network produces a low resolution representation of the input image using a down-sampling technique. In general this section is built by adapting the convolutional layers of a typical classification network (e.g., AlexNet, VGG-16). The decoder network may have different architectures and is used for up-sampling the coarse feature map produced by the last layer of the encoder. FCN architecture requires a large number of trainable parameters in the encoder network and a small decoder network. Thus, the overall large size of this network makes it hard to achieve an optimal trade-off between accuracy and computational resources in real-time semantic segmentation.

Since the emergence of FCNs a large varieties of architectures have been proposed and the main approaches that represent the state-of-the-art in semantic segmentation are: U-Net [56], DeepLab [57], SegNet [41], ENet [58], ICNet [59], ERFNet [60], BiSeNet [61], Fast-SCNN [62], SwiftNetRN [63], FCHarDNet [64]. Some of these networks aim at addressing the accuracy problem while others improve the performance in terms of storage cost, computational time and power consumption. Unfortunately none of the aforementioned architectures is able to take into account the key requirements of storage cost, accuracy, inference time and low power for real-time semantic segmentation in a vehicle.

The aim of this paper is to propose a low-rank CNN architecture for real-time semantic segmentation, suitable to be implemented in a visual SLAM framework that is able: *i)* to address all those issues (storage cost, accuracy, inference time, low power) and *ii)* to outperform in terms of these key aspects all of the state-of-the-arts architectures.

The main contributions of the paper are summarized as follows.

- A CNN compression technique based on CANDECOMP/PARAFAC (CP) tensor decomposition, has been analysed and adopted to reduce the complexity of a convolutional layer.
- An optimized architecture which combines U-Net and ResNet models has been proposed.
- The network has been compressed with the CP decomposition technique to reduce the computational complexity.
- The low-rank CNN architecture so derived, has been implemented in two embedded platforms: a Raspberry Pi 4 (CPU) and an NVIDIA Jetson Nano 2 GB (GPU). These two platforms have been chosen in order to meet the different performance required by autonomous robot navigation and car navigation systems. Indeed, in the former cost and low power are the main performance required, while in the latter speed of SLAM algorithm is a fundamental aspect. Additionally, to meet the best

performance the network has been implemented in the high-cost GPU NVIDIA Tesla P100 PCIe 16 GB.

The rest of the paper is organized as follows. Section II summarizes the related work and Section III describes the proposed work. Section IV analyses and applies the CP decomposition to a convolutional layer. Section V derives a new low-rank CNN architecture called UNet-ResNet. Section VI is devoted to experiments conducted in three different platforms: Raspberry Pi 4, GPU NVIDIA Jetson Nano 2 GB and GPU NVIDIA Tesla P100 PCIe 16 GB, to compare the performance of LR-UNet-ResNet with the state-of-the-art.

II. RELATED WORK

A. LITERATURE REVIEW

Semantic image segmentation is an active research field [65]–[69] in which a large amount of methods have been proposed over the last years to improve the performance of this image technique in view of the extensive application prospects. Before the advent of deep neural networks several methods based on classifying pixel independently have been used. In [70], semantic texton forests have been introduced that are randomized decision forests which use only simple pixel comparisons on local image patches, then performing both an implicit hierarchical clustering into semantic textons and an explicit local classification of the patch category. An approach that integrates motion and appearance-based features for object recognition and segmentation of road scenes has been proposed in [71]. The motion-based features and appearance-based features (textons, colour, location and HOG descriptors) are combined within a boosting framework that automatically selects the most discriminative features for each object. The work [72] can be viewed as an integration of object class segmentation methods and object detection approaches. The model used is a conditional random field defined on pixels, segments and objects, then a global energy function which combines results from sliding window detectors, low-level pixel-based unary and pairwise relations, has been adopted. Successively, other approaches have been addressed to produce high quality unaries by trying to predict the labels for all the pixels in a patch instead of only in central pixel. Inspired by this idea [73] provided a novel way to incorporate joint statistics about the local label neighborhood in the random forest framework. The approach adopted in [74] differs from other previous existing solutions for the use of dense depth maps recovered via multi-view stereo matching techniques as cues to achieve accurate scene parsing. A combination of popular hand designed features and spatio-temporal super-pixelization is used in [75] to obtain higher accuracy in labelling image regions. The availability of RGB-D sensors (color + depth) and the release of several Kinect datasets [76] showed the usefulness of the depth channel to improve segmentation. Improvements were made in [77] by extracting RGB-D rich features at low-level and by combining, for contextual modeling, two strategies, one using segmentation trees, and the other using superpixel Markov random fields.

More recently the success of deep convolutional neural networks has prompted many researchers to exploit their feature learning capabilities for classifying images at pixel level. In [78] a feed-forward neural network approach which can take into account long range label dependencies in the scenes while controlling the capacity of scene, was suggested. A new form of convolutional neural network that combines the strengths of CNNs and conditional random fields (CRFs)-based probabilistic graphical modeling, has been introduced in [43]. Contrary to previous existing approach posing semantic segmentation as a single task of region-based classification, the architecture in [79] decouples classification and segmentation: labels associated with an image are identified by a classification network, and binary segmentation is subsequently performed by segmentation network.

Recent approaches to semantic segmentation are based on convolutional encoder-decoder architectures where the encoder generates low-resolution image features and the decoder upsamples features to segmentation maps with per-pixel class scores. The pioneer method used to perform end-to-end segmentation is the fully convolutional network (FCN) [39], in which the last fully connected layer of traditional architectures, such as VGG [80] and AlexNet [81], is converted into a fully convolutional layer. The interesting idea of this approach is that a simple interpolation filter is employed for deconvolution and only the CNN part of the network is fine-tuned to learn deconvolution indirectly. To overcome the absence of real deconvolution in FCN, a different strategy is used in [82] in which a multi-layer deconvolution layer, composed of deconvolution, unpooling, and rectified linear unit (ReLU) layers, is learned. A large number of methods choose the architecture of FCN as their baseline and the predictive performance of FCN has been improved further by adopting a large variety of different solutions.

B. STATE-OF-THE-ART TECHNIQUES

Convolutional neural networks (CNNs) are powerful visual methods that have proven to be particularly suitable for solving whole-image classification tasks. Nevertheless, in recent years it has been shown that CNNs can also be adopted to perform dense predictions for per-pixel tasks such as semantic segmentation. Here we summarize the main techniques that represent the state-of-the-art in semantic segmentation and that will be used for a comparison with the architecture proposed in this paper.

1) FCN

Fully convolutional networks (FCNs) [39], are built by transforming fully connected layers of a CNN into convolutional layers. The network so obtained is called convolutionalized network. In this way typical classification networks (i.e.,

AlexNet, VGG-16) are first converted into fully convolutional networks and then a transposed convolution layer (also called up-sampling layer) is appended to the end of the convolutional networks. The transposed convolution layer is used for up-sampling the output feature map obtained by the last layer of the initial CNN architecture, to produce a dense prediction of the input image.

2) U-NET

U-Net architecture firstly proposed in [56] for biomedical image segmentation, modifies and extends FCN architecture such that it works with very few training images and yields more precise segmentations. It consists of a contracting path and an expansive path. The former (contracting path) has the typical architecture of a convolutional network: repeated application of convolutions, each followed by a rectified linear unit (ReLU) and a pooling operation for downsampling. The latter (expansive path) consists of the repeated application of an upsampling applied to the feature map, followed by a convolution ("up convolution"), that halves the numbers of feature channels.

3) DEEPLAB

DeepLab model [46] was proposed to overcome some limitations in the application of CNNs to semantic segmentation: reduced feature resolution, existence of objects at multiple scales, and reduced localization accuracy due to CNN invariance. To address the first issue, convolution with upsampled filters, or 'atrous convolution', is derived as a powerful tool in dense prediction tasks. More specifically, the downsampling operator from the last few max pooling layers of CNNs are removed and instead the filters in subsequent convolutional layers are upsampled, resulting in feature maps computed to a higher sampling rate. Secondly the atrous spatial pyramid pooling (ASPP) technique, a computationally efficient scheme of resampling a given feature layer at multiple rates prior to convolution, is adopted to robustly segment objects at multiple scales. Finally to remove the invariance to spatial transformations, inherently limiting the spatial accuracy of a CNN, a fully connected Conditional Random Field (CRF) is applied to capture fine details. In order to capture the contextual information at multiple scales, DeepLabv3 [83] applies several parallel atrous convolutions with different rates and DeepLabv3+ [57] extends DeepLabv3 by adding a simple effective decoder module to refine the segmentation.

4) SEGNET

SegNet [41] is a deep convolutional encoder-decoder architecture that consists of an encoder network, and a corresponding decoder network followed by a pixel-wise classification layer. The encoder is topologically identical to the convolution layers in VGG-16, except for the fully connected layers that are removed. The novelty of SegNet is the decoder network which is formed by a hierarchy of decoders one corresponding to the other. A decoder uses the max-pooling indices received from the corresponding encoder to perform upsampling of their input feature maps. It then performs convolution with a trainable filter bank to densify the feature map.

5) ENET

The architecture named ENet (Efficient Neural Network) [58] was created specifically for tasks requiring a low latency operation. The architecture is based on the concept of 'deep residual learning', that was introduced in [84] to solve the degradation problem of deeper networks: with the network depth increasing, accuracy is saturated and degrades rapidly. The architecture is formed by stacked layers, named bottleneck modules, each of which asymptotically approximates a residual function instead of a generic mapping. Each conv layer is either a regular, dilated for full convolution (also known as deconvolution), and the activations are zero padded to match the number of feature maps.

6) ICNET

Image Cascade Network (ICNet) [59] is mainly focused on the challenging task of real-time semantic segmentation, to make semantic segmentation fast while not sacrificing too much quality. To this end the image at the input of the ICNet is downsampled by factors of 2 and 4, forming a cascade input to medium-and-high-resolution branches. To get high quality segmentation, medium and high resolution branches help recover and refine the coarse prediction. To limit the number of parameters, light weighted convolution layers are adopted in higher resolution branches, while output feature maps from low and medium resolution are fused in the high-resolution branch by a cascade-feature-fusion unit.

7) ERFNET

ERFNet (Efficient Residual Factorized Network) architecture [60] is based on the concept of residual layer [84] which has the property of allowing convolutional layers to approximate residual functions. This technique significantly reduces the degradation problem present in architectures that stack a large amount of layers. Two different versions are commonly used to implement a residual layer: the non-bottleneck design with two 3×3 convolutions and the bottleneck design with two 1×1 convolutions at the input and output ends, and a 3×3 convolution in the middle. As these two layers compete each other in terms of computational resources and accuracy, in the ERFNet the non-bottleneck residual module is redesigned in a more optimal way by entirely using convolutions with $1D$ filters.

8) BISENET

Bilateral Segmentation Network (BiSeNet) [61] has been proposed to achieve real-time inference speed while preserving segmentation performance. This architecture is composed of two parts: Spatial path (SP) and Context path (CP). SP is used to preserve the spatial size of the original

TABLE 1. Summary of advantages and disadvantages of the different architecture described in Section II-B.

Model	Pros	Cons	Platform Used
FCN	converts an existing CNN architecture constructed for classification to a fully convolutional network	absence of real deconvolution layer	NVIDIA Tesla K40c
U-Net	upsampling layers increase the resolution of output	introduces extra computation, most spatial information lost cannot be recovered	NVIDIA Titan
DeepLab	recovers full resolution features maps by atrous (dilated) convolution	aims at enhancing segmentation accuracy alone, memory and computational expensive	NVIDIA Titan X
SegNet	the decoder uses pooling indices computed in the encoder to improve efficiency	only efficient in terms of memory and computational time, computational expensive	NVIDIA Titan with cuDNN v3 acceleration
ENet	optimized for fast inference and high accuracy	low accuracy	NVIDIA Titan X, NVIDIA Jetson TX1 (embedded)
ICNet	it can run at high resolution 1024×2018	memory expensive	NVIDIA Titan X
ERFNet	allows stacking a large amount of layers	memory expensive	NVIDIA Titan X, NVIDIA Jetson TX1 (embedded)
BiSeNet	uncouples spatial information from contextual information	memory and computational expensive	NVIDIA Titan XP
Fast-SCNN	uses two-branch networks to combine global context with spatial details	memory expensive	NVIDIA Titan X, NVIDIA Titan XP
SwiftNet	leverages light-weight upsampling with lateral connections as the most cost-effective solution to restore the prediction resolution	memory expensive	NVIDIA GTX 1080Ti, NVIDIA Jetson TX2 (embedded)
HarDNet	uses an efficient sparsification scheme that reduces the concatenation cost, low MACs and memory traffic	memory expensive	NVIDIA TitanV, NVIDIA GTX1080, NVIDIA Jetson Nano (embedded)

input image and encode its spatial information. This path contains three layers and extracts 1/8 of the original image, thus encoding rich spatial information. The CP downsamples the input feature map quickly to obtain a large receptive field, thus encoding high level semantic information. Finally a Feature Fusion Module (FFM) and an Attention Refinement Module (ARM) are used for the function of the two paths and refinement of final prediction.

9) FAST-SCNN

Fast Segmentation Convolutional Neural Network (Fast-SCNN) [62], is inspired by existing two-branch methods for fast segmentation of high resolution images, and is particularly suited to efficient computation on embedded devices with low memory. Fast-SCNN uses three modules: a learning to downsample module, a coarse global feature model and a feature fusion module. A learning to downsample module computes low-level features for multiple resolution branches simultaneously. The global feature extractor module is aimed at capturing the global context. The feature fusion model is used for the fusion of the two branches.

10) RECENT EFFICIENT MODELS

Several efficient models for dense prediction have recently been proposed for real-time prediction on mobile platforms such as cars, drones, and various kind of robots.

Instead of using custom lightweight architectures to decrease computational complexity, in [63] an alternative approach which achieves a significantly better performance, has been proposed. The method has been conceived around three basic building blocks. i) Recognition encoder. A pre-trained lightweight architecture (ResNet-18 [84] or MobileNet V2 [85]) has been used as the main segmentation encoder. ii) Upsampling decoder. To leverage lightweight upsampling, lateral connections as the most cost-effective solution to restore prediction resolution, is used. iii) Module for increasing the receptive field. A spatial pyramid pooling (SPP) block has been adopted to enlarge the receptive field by fusing shared features at multiple resolution.

The Harmonic Densely Connected Network to achieve high efficiency in terms of both low MACs and memory traffic, has been recently proposed [64]. This new network architecture is based on a Densely Connected Network and uses an efficient sparsification scheme that reduces the concatenation cost significantly better than that achieved by logDenseNet. The proposed connection pattern forms a group of layers called a Harmonic Dense Block (HDB), which is followed by a Conv 1×1 layer as a transition. DenseNet employees a bottleneck layer before every Conv 3×3 layer to enhance the parameter efficiency.

For easy reference Table 1 summarizes the most relevant aspects, as well as advantages and disadvantages of architectures previously discussed.

III. PROPOSED METHOD

Although all the networks previously discussed are able to obtain a good accuracy, the performance achieved both in terms of storage cost and computational complexity do not suffice for embedded MCUs.

The main goal of the proposed work is to prove that a novel CNN architecture can be derived, to achieve an optimal trade-off between accuracy and computational resources. To reach this goal we proceed as follows:

- The CANDECOMP/PARAFAC (CP) decomposition technique has been applied to the kernel of a generic convolutional layer. In this way the layer is decomposed as two 1×1 convolutions at the input and output ends, and a $D \times D$ convolution in the middle. As a result a reduction of both the parameter number and operation number is obtained.
- An optimized architecture which combines U-Net structure and ResNet residual blocks, has been defined. Three versions, i.e., V1, V2 and V3, have been derived to explore the trade-off between model complexity and segmentation accuracy.
- All those models were initially trained to get the desired accuracy and were compressed using the CP decomposition technique. To this end an iterative compression fine-tuning algorithm was used to compensate for the loss of accuracy due to compression.
- The three models were implemented on Google Colaboratory using TensorFlow v. 2.4.1 and TensorFlow Keras v. 2.4.0 as a backend in Python v. 3.7.10. The networks were trained for 100 epochs on the CamVid dataset by using an Adam optimizer with categorical cross entropy as loss function.
- The low-rank CNN architectures so derived, have been implemented in two embedded platforms: a Raspberry Pi 4 (CPU) and an NVIDIA Jetson Nano 2 GB (GPU).

IV. TENSOR DECOMPOSITION FOR CNN COMPRESSION

CNN accuracy is achieved at the expense of high computational complexity, thus adopting CNN compression techniques to reduce the inference time in real-time SIS is essential. Among them, pruning, quantization and low-rank tensor decomposition are the most common.

Pruning starts by learning the weights via normal training, then proceeds by removing the small-weight connections [86]–[88].

Quantization reduces the number of bits used to represent data [89], [90].

Tensor decomposition is a multilinear approximation that reduces a full-rank tensor to a low-rank tensor, i.e., that requires a reduced number of data and operations to be represented by preserving accuracy [91]–[93].

Since CNN models involve convolutional operations on banks of filters represented by tensors, the size of which can be very large, thus tensor decomposition is a useful technique to reduce complexity of CNN implementation [89], [94], and it will be applied in the following to derive a high performance CNN architecture.

Although pruning and quantization are valuable techniques for CNN comparison, they don't affect the network architecture, thus they are irrelevant in defining a new optimized CNN model.

It is well known that the numbers of an N-th order $(I \times I \times \ldots I)$ tensor, I^N, scales exponentially with the tensor order N. Among different techniques available to reduce computational cost and storage cost of tensor models for an N-th order tensor for which the storage requirement of raw data is $O(I^N)$, the most effective are CANDECOMP/PARAFAC (CP) decomposition and Tucker decomposition (TKD) [92]. The number of parameters in a CP decomposition reduces to NIR while TKD reduces the size of a given data tensor to $(NIR+R^N)$, where R represents the number of terms in the decomposition. Thus in order to minimize CNN complexity, CP decomposition will be used since it is able to outperform TKD in terms of compression rate.

This method has shown to be very effective to reduce redundancy of the neural network parameters, however it suffers for a degeneracy phenomena when applied to approximate a tensor of relatively high rank. The degeneracy of CP has been extensively studied in the past [95], [96] and is a common phenomena in tensors which have non unique CP and determines stability problems during the training stage of a CNN. Recently the Error Preserving Correction method (EPC) [97] and a variant of EPC [98] that introduce a correction to the decomposition have been proposed to get a more stable decomposition. To avoid this degeneracy effect we will apply the CP method to tensors of not too high rank.

A. CP DECOMPOSITION

The CANDECOMP/PARAFAC (CP) decomposition [92] is a multilinear transformation [93] that approximates a tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_n}$ with a tensor $\hat{\mathcal{A}}$ that is a linear combination of R rank-1 tensor in the form

$$\hat{\mathcal{A}} = \sum_{r=1}^{R} \lambda_r \, a_r^{(1)} \circ a_r^{(2)} \circ \ldots \circ a_r^{(n)} \tag{1}$$

where the symbol "$\circ$" represents the outer product [91], $a_r^{(1)}, a_r^{(2)}, \ldots$ are the column vectors of the matrices $A_1 = [a_1^{(1)}, \ldots, a_R^{(1)}] \in \mathbb{R}^{n_1 \times R}$, $A_2 = [a_1^{(2)}, \ldots, a_R^{(2)}] \in \mathbb{R}^{n_2 \times R}$, $\ldots$ respectively and λ_r are coefficients to be determined. The matrices $A_1, A_2, \ldots$ and $\Lambda = diag(\lambda_1, \ldots, \lambda_R)$ are derived by solving for a given value of R the optimization problem

$$\min_{\hat{\mathcal{A}}} = \left\| \mathcal{A} - \hat{\mathcal{A}} \right\|_F \tag{2}$$

where $\|\cdot\|_F$ denotes the Frobenius norm. Denoting with $\mathcal{A}_{(1)}, \mathcal{A}_{(2)}, \ldots$ and $\hat{\mathcal{A}}_{(1)}, \hat{\mathcal{A}}_{(2)}, \ldots$ the modal unfoldings (or matricizations) of tensors $\mathcal{A}$ and $\hat{\mathcal{A}}$ respectively, it can be proven that the error in (2) can be rewritten as

$$\left\| \mathcal{A} - \hat{\mathcal{A}} \right\|_F = \left\| \mathcal{A}_{(1)} - \hat{\mathcal{A}}_{(1)} \right\|_F = \left\| \mathcal{A}_{(2)} - \hat{\mathcal{A}}_{(2)} \right\|_F = \ldots. \tag{3}$$

Taking into account these constraints, the minimization problem stated by (2) is equivalent to a multilinear least-square problem, which can be solved by minimizing iteratively any one of the errors between the matrices in (3) until the convergence is reached, using the so-called multilinear least-square (MLS) technique.

B. CNN COMPRESSION

In this section we want to show that the CP decomposition technique previously described, can be profitably used to compress a CNN network in order to reduce both the storage cost and the computational cost.

To this end let us refer to a generic convolutional layer

$$\mathcal{Y}_{h',w',t} = \sum_{s=1}^{S} \sum_{i,j} \mathcal{X}_{h_i,w_j,s}\, \mathcal{K}_{i,j,s,t} \tag{4}$$

where $\mathcal{X} \in \mathbb{R}^{I_x \times I_y \times S}$ is the input tensor, $\mathcal{Y} \in \mathbb{R}^{O_x \times O_y \times T}$ the output tensor and $\mathcal{K} \in \mathbb{R}^{D \times D \times S \times T}$ the kernel of the filter. Denoting with s the stride of the layer, then $O_{x,y} = I_{x,y}/s$ and the number of operations, both additions and multiplications, are given by

$$N_{ops} = O_x O_y D^2 ST = \frac{I_x I_y}{s^2} D^2 ST = \frac{I_x I_y}{s^2} N_{params} \tag{5}$$

where N_{params} is the number of parameters in the weight tensor.

Using the CP decomposition the kernel $\mathcal{K}$ can be approximated by the 1-rank tensor

$$\hat{\mathcal{K}}_{i,j,s,t} = \sum_{r=1}^{R} a_{i,r} \circ b_{j,r} \circ c_{s,r} \circ d_{t,r} \tag{6}$$

so that (6) becomes

$$\mathcal{Y}_{h',w',t} \cong \sum_{r=1}^{R} \sum_{s=1}^{S} \sum_{i,j} \mathcal{X}_{h_i,w_j,s}\, \mathcal{Q}_{i,j,r}\, c_{s,r}\, d_{t,r} \tag{7}$$

where $\mathcal{Q}_{i,j,r} = a_{i,r}\, b_{j,r}$ is the generic term of tensor $\mathcal{Q} \in \mathbb{R}^{D \times D \times R}$.

Defining the tensors $\mathcal{W}$ and $\mathcal{Z}$, then (7) can be interpreted as a sequence of three convolutions

$$\mathcal{W}_{h_i,w_j,r} = \sum_{s=1}^{S} \mathcal{X}_{h_i,w_j,s}\, c_{s,r} \tag{8}$$

$$\mathcal{Z}_{h',w',r} = \sum_{i,j} \mathcal{W}_{h_i,w_j,s}\, a_{i,j,r} \tag{9}$$

$$\mathcal{Y}_{h',w',t} = \sum_{r=1}^{R} \mathcal{Z}_{h',w',r}\, d_{t,r}, \tag{10}$$

two 1×1 convolutions (8), (9) at the input and output ends respectively, and a $D \times D$ convolution in the middle.

Comparing (4) and (7) it results that the tensor $\mathcal{K}_{i,j,s,t} \in \mathbb{R}^{D \times D \times S \times T}$ in (4) is decomposed in (7) as the combination of tensors $c_{s,r} \in \mathbb{R}^{S \times R}$, $d_{r,t} \in \mathbb{R}^{R \times T}$, $\mathcal{Q}_{i,j,r} \in \mathbb{R}^{D \times D \times R}$ and this correspondence is schematically depicted in Fig. 1.

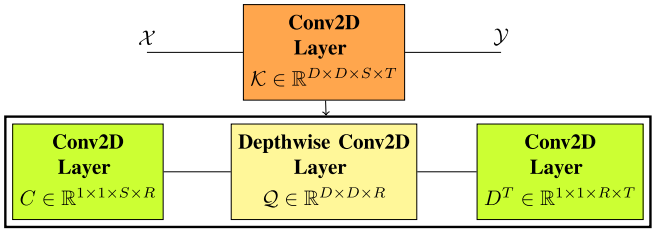

FIGURE 1. The effect of CP decomposition on Conv2D Layer.

Denoting by c_{params} and c_{ops} the reduction factors of the parameters number and Floating Point Operations number, it is easy to show that:

$$c_{params} = \frac{R\left(D^2 + S + T\right)}{D^2 ST} \tag{11}$$

$$c_{ops} = \frac{s^2 RS + RT + RD^2}{D^2 ST}. \tag{12}$$

V. LOW-RANK CNN ARCHITECTURE

The design flow to derive the low-rank CNN architecture based on CP decomposition comprises two main stages.

- First, an optimized architecture called UNet-ResNet, or full rank (FR) UNet-ResNet, which combines U-Net [56] structure with ResNet [99] residual-blocks technique was proposed.
  Three different versions V1, V2 and V3 of the model have been proposed in order to better explore the tradeoff between model complexity and segmentation accuracy.
  These models were initially trained to get the desired accuracy.
- Next, these trained networks were decomposed with the CP decomposition technique to reduce the computational complexity and to achieve acceptable values for inference times. An iterative compression fine-tuning technique was used to compensate for the loss of accuracy due to network decomposition. The networks so obtained were called low rank UNet-ResNet (LR-UNet-ResNet) V1, V2 and V3.

A. UNET-RESNET ARCHITECTURE

The design of the UNet-ResNet was based on the U-Net architecture [56], truncated to three stages as depicted in Fig. 2.

In using this architecture however, a large number of encoder/decoder layers and block channels would be required to achieve a satisfactory learning accuracy, that corresponds to a high-complex model unsuitable to be implemented in an embedded processor. To overcome this issue, the residual-blocks of ResNet architecture [99] have been adopted to implement the encoder/decoder layers, instead of simple convolutional layers. These blocks, which are depicted in Fig. 3, are formed by two convolutional blocks in the main path, the first of which may include a downsampling of the features. In this case a convolutional layer is added to the residual path

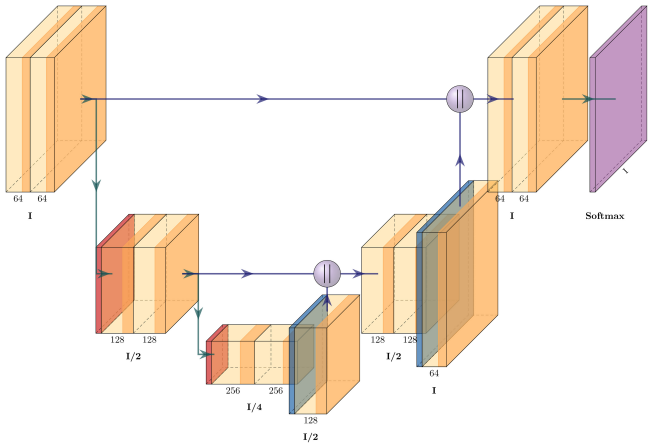

FIGURE 2. U-Net architecture truncated to three stages.

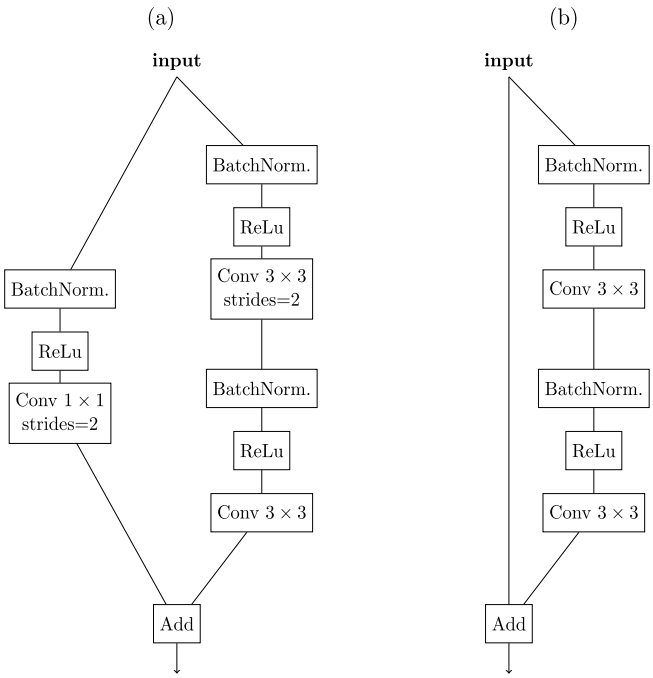

FIGURE 3. Residual blocks of ResNet architecture with (a) and without (b) downsampling.

in order to adjust features dimension for the final addition, otherwise residual-path is simply an identity.

A similar approach, based on the combination of UNet and ResNet architectures, has been used in [100] to solve a binary sea-land semantic segmentation task. However, instead of using simple convolutions in the upsampling process the proposed model uses residual blocks for both encoding/decoding stages since an accurate design of the decoder architecture process has shown to be crucial for segmentation accuracy [82].

Instead of starting with totally random initialized weights, a pretrained set of weights on Imagenet dataset [101] were used to achieve better segmentation performance.

TABLE 2. Summary of UNet-ResNet V1 architecture. Here *c* is the number of output channels, *s* is the stride of the block.

Block	c	s	Output name	Output shape
conv2d 7×7	64	2	-	$240 \times 180 \times 64$
maxpool	64	2	X_0	$120 \times 90 \times 64$
residual block	64	2	-	$60 \times 45 \times 64$
residual block	64	1	X_1	$60 \times 45 \times 64$
residual block	128	2	-	$30 \times 23 \times 128$
residual block	128	1	-	$30 \times 23 \times 128$
decoder residual block	64	2	-	$60 \times 46 \times 64$
decoder residual block	64	1	Y_1	$60 \times 46 \times 64$
concat(Y_1, X_1)	-	-	-	
decoder residual block	64	2	-	$120 \times 90 \times 64$
decoder residual block	64	1	Y_2	$120 \times 90 \times 64$
concat(Y_2, X_0)	-	-	-	
conv2d 1×1	12	1	-	$120 \times 90 \times 12$
Upsample	12	4	-	$480 \times 360 \times 12$
Softmax	-	-	Y_{pred}	$480 \times 360 \times 12$

UNet-ResNet V1 architecture is reported in Fig. 4 and described in Table 2. A first reduction of the features dimension is obtained with a convolutional layer and a max-pooling, followed by two stages which are formed by two residual-blocks each, the first of whom is responsible for the downsampling.

The decoder stages are made by the residual-blocks of the encoder without downsampling, the first of whom is followed by a bilinear interpolation upsampling. A factor 4 of upsampling is recovered at the output of the network.

With the aim of achieving better results in terms of computational efficiency without a high loss of accuracy, the stride of the initial maxpooling was set to $s = 4$ and a third stage with 2 residual-blocks was added to the encoder and decoder. With these modifications the UNet-ResNet V2 architecture described in Table 3 was obtained.

An intermediate version V3 might be derived choosing $s = 2$ for the second residual-block, instead of using a downsampling factor of 4 for the maxpooling. In this manner the features learnt by the first residual-block are of the same dimension as in V1, that corresponds to a more accurate segmentation than V2 since less information is lost. On the other hand complexity is higher than V2 but lower than V1. The architecture of the network UNet-ResNet V3 so derived can be found in Table 4.

In order to better understand the properties of the networks previously discussed it is worth to explore the similarity between low-rank UNet-ResNet architecture and Depthwise Separable models [102], [103]. The basic idea in these latter models is to replace a convolutional layer with a factorized version that splits convolution into two separate layers. The first layer, called depthwise convolution, performs lightweight filtering by applying a simple convolutional filter per input channel. The second layer, called pointwise convolution, performs a 1×1 convolution. A

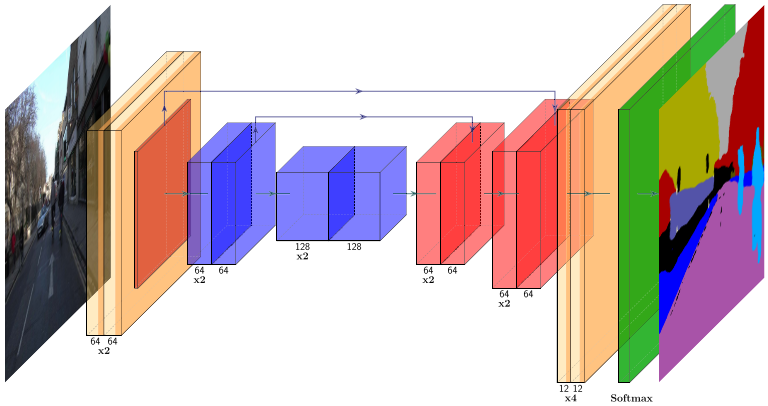

FIGURE 4. UNet-ResNet V1 architecture.

TABLE 3. Summary of UNet-ResNet V2 architecture.

Block	c	s	Output name	Output shape
conv2d 7×7	64	2	-	$240 \times 180 \times 64$
maxpool	64	4	X_0	$60 \times 45 \times 64$
residual block	64	2	-	$30 \times 23 \times 64$
residual block	64	1	X_1	$30 \times 23 \times 64$
residual block	128	2	-	$15 \times 12 \times 128$
residual block	128	1	X_2	$15 \times 12 \times 128$
residual block	256	2	-	$8 \times 6 \times 256$
residual block	256	1	-	$8 \times 6 \times 256$
decoder residual block	128	2	-	$16 \times 12 \times 128$
decoder residual block	128	1	Y_1	$16 \times 12 \times 128$
concat(Y_1, X_2)	-	-	-	
decoder residal block	64	2	-	$30 \times 24 \times 64$
decoder residal block	64	1	Y_2	$30 \times 24 \times 64$
concat(Y_2, X_1)	-	-	-	
decoder residual block	64	2	-	$60 \times 46 \times 64$
decoder residual block	64	1	Y_3	$60 \times 46 \times 64$
concat(Y_3, X_0)	-	-	-	
conv2d 1×1	12	1	-	$120 \times 90 \times 12$
Upsample	12	8	-	$480 \times 360 \times 12$
Softmax	-	-	Y_{pred}	$480 \times 360 \times 12$

TABLE 4. Summary of UNet-ResNet V3 architecture.

Block	c	s	Output name	Output shape
conv2d 7×7	64	2	-	$240 \times 180 \times 64$
maxpool	64	2	X_0	$120 \times 90 \times 64$
residual block	64	2	-	$60 \times 45 \times 64$
residual block	64	2	X_1	$30 \times 23 \times 64$
residual block	128	2	-	$15 \times 12 \times 128$
residual block	128	1	X_2	$15 \times 12 \times 128$
residual block	256	2	-	$8 \times 6 \times 256$
residual block	256	1	-	$8 \times 6 \times 256$
decoder residual block	128	2	-	$16 \times 12 \times 128$
decoder residual block	128	1	Y_1	$16 \times 12 \times 128$
concat(Y_1, X_2)	-	-	-	
decoder residual block	64	2	-	$30 \times 24 \times 64$
decoder residual block	64	1	Y_2	$30 \times 24 \times 64$
concat(Y_2, X_1)	-	-	-	
decoder residual block	64	2	-	$60 \times 46 \times 64$
decoder residual block	64	2	Y_3	$120 \times 92 \times 64$
concat(Y_3, X_0)	-	-	-	
conv2d 1×1	12	1	-	$120 \times 90 \times 12$
Upsample	12	4	-	$480 \times 360 \times 12$
Softmax	-	-	Y_{pred}	$480 \times 360 \times 12$

modified version of this model, used in the MobileNet V2 architecture [85], is based on the so called 'residual bottleneck layer' (rbl), that comprises two 1×1 convolutions at the input and output ends, and a 3×3 convolution in the middle. Although rbl resembles the layer achieved by CP decomposition described in Section IV, the resulting networks, LR UNet-ResNet and MobileNet V2, can be quite different. In fact, the LR UNet-ResNet is obtained starting from the FR UNet-ResNet that was initially trained to get the desired accuracy and then, once decomposed with the CP technique, is retrained to restore the accuracy. Differently MobileNet is trained without benefit of a pre-training/finetuning scheme applied to a full-rank/low-rank dimensionality reduction technique as was done in the proposed approach.

VI. EXPERIMENTAL RESULTS

A. DATASET

We evaluated the performance of UNet-ResNet (V1, V2, and V3) on the Cambridge-driving Labeled Video Database (CamVid) dataset [104]–[106]. This dataset is often used in (real-time) semantic segmentation research, particularly for road scene segmentation and autonomous driving applications. The CamVid dataset is a publicly available cloud segmentation dataset that contains a collection of videos with object class semantic labels, completed with metadata. The database provides ground truth labels that associate each pixel with one of 32 semantic classes that are relevant in a driving environment: animal, archway, bicyclist, bridge, building, car (sedan/wagon), cart/luggage/pram, child, column/pole, fence, lane markings drivable, lane markings

TABLE 5. Comparison of the full-rank UNet-ResNet V1, V2 and V3.

	Test accuracy [%]	mIoU [%]	Storage cost [MB]	Parameters number	MFLOPs
UNet-ResNet V1	86.58	56.67	12.663	1077k	5209
UNet-ResNet V2	85.54	53.94	46.620	4033k	2340
UNet-ResNet V3	85.93	54.88	46.726	4041k	2711

non-drivable, misc text, motorcycle/scooter, other moving, parking block, pedestrian, road (drivable surface), road shoulder, sidewalk, sign / symbol, sky, SUV / pickup truck, traffic cone, traffic light, train, tree, truck / bus, tunnel, vegetation misc, void, wall. The "void" label indicates an area which is ambiguous or irrelevant in this context. The dataset is split up as follows: 369 training, 100 validation and 232 testing pair images (original RBG images and segmentation masks). The original frame resolution for this dataset is 960×720.

In this experimentation, 11 foreground classes are selected: *bicyclist, building, car, fence, pedestrian, pole, road, sidewalk, sign, sky, tree*. This approach is commonly used in literature [41], [58], [59], [63] as it better reflects the statistically significant classes of the dataset [106]. To reduce 32 classes into 11, multiple classes from the original dataset are grouped together, e.g., "car" is a combination of "car", "SUV / pickup truck", "truck / bus", "train", and "other moving". By adding the *void* class, which is treated as background, we have 12 classes in total. The original frame resolution has been downsampled to 480×360.

Fig. 6 shows some examples of this dataset. The first row is for the raw input image, and the second row is for the ground truth.

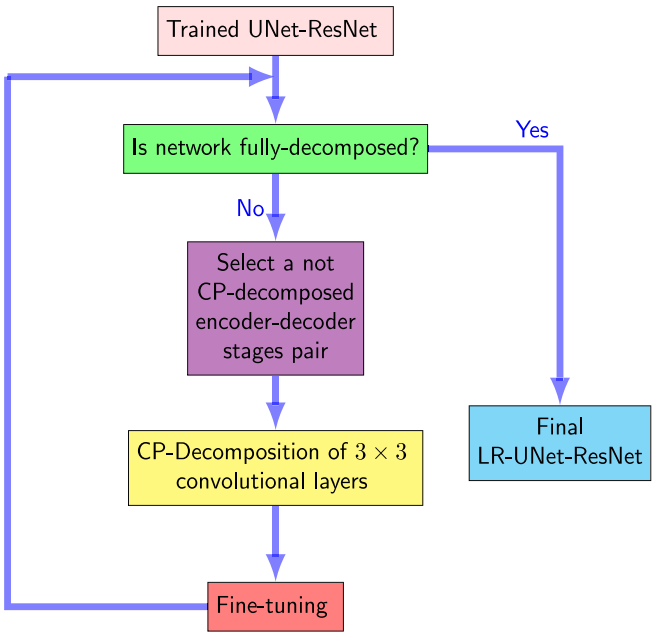

FIGURE 5. Design flow to derive final network LR-UNet-ResNet.

B. TRAINING OF UNET-RESNET ARCHITECTURE

The proposed models were trained on the dataset described in the previous subsection by minimizing a categorical-crossentropy loss with the Adam optimizer, by using a learning rate of 0.001 by applying an exponential decay with a rate of 0.985 for 100 epochs and a batch size of 3.

Model performances were evaluated using standard segmentation metrics, i.e., accuracy and mean Intersection-Over-Union (mIoU) also known as Jaccard Index.

On the other hand models' suitability in a low-cost, low-power embedded system was measured with storage cost (MB), parameters number (complexity), and number of FLOPs (Floating Point Operations). The storage cost has been computed using the Python API `os.path.getsize()`, which returns the file size in bytes, where, in this case, the file corresponds to the CNN model in h5 format.

Table 5 provides the results of full-rank UNet-ResNet V1, V2, V3 for the specified metrics.

All the models achieve good segmentation accuracy and mIoU. UNet-ResNet V1 has the highest accuracy but its complexity is considerably higher than V2 and V3. UNet-ResNet V2 has the minimum complexity but the worst segmentation performance. This loss is caused by the details missed by increasing the initial downsampling factor. The gap is partially recovered by UNet-ResNet V3.

It can be noticed from Table 5 that a high number of FLOPs and storage cost are required, resulting for an unpractical model to be implemented on an embedded device for the discussed application.

For this purpose hence it is essential to dramatically reduce model complexity without losing too much segmentation accuracy contemporarily.

Thus the trained models were compressed with CP-decomposition applied to all but the 1×1 convolutional layers (which cannot be further decomposed properly) according to (8), (9) and (10).

Particularly, an iterative compression & fine-tuning technique was used to derive the final LR-UNet-ResNet, which is synthesized by Fig. 5. At each iteration all 3×3 layers of a stage in the encoder and the corresponding ones in the decoder are decomposed, then the network is re-trained to recover the loss of accuracy.

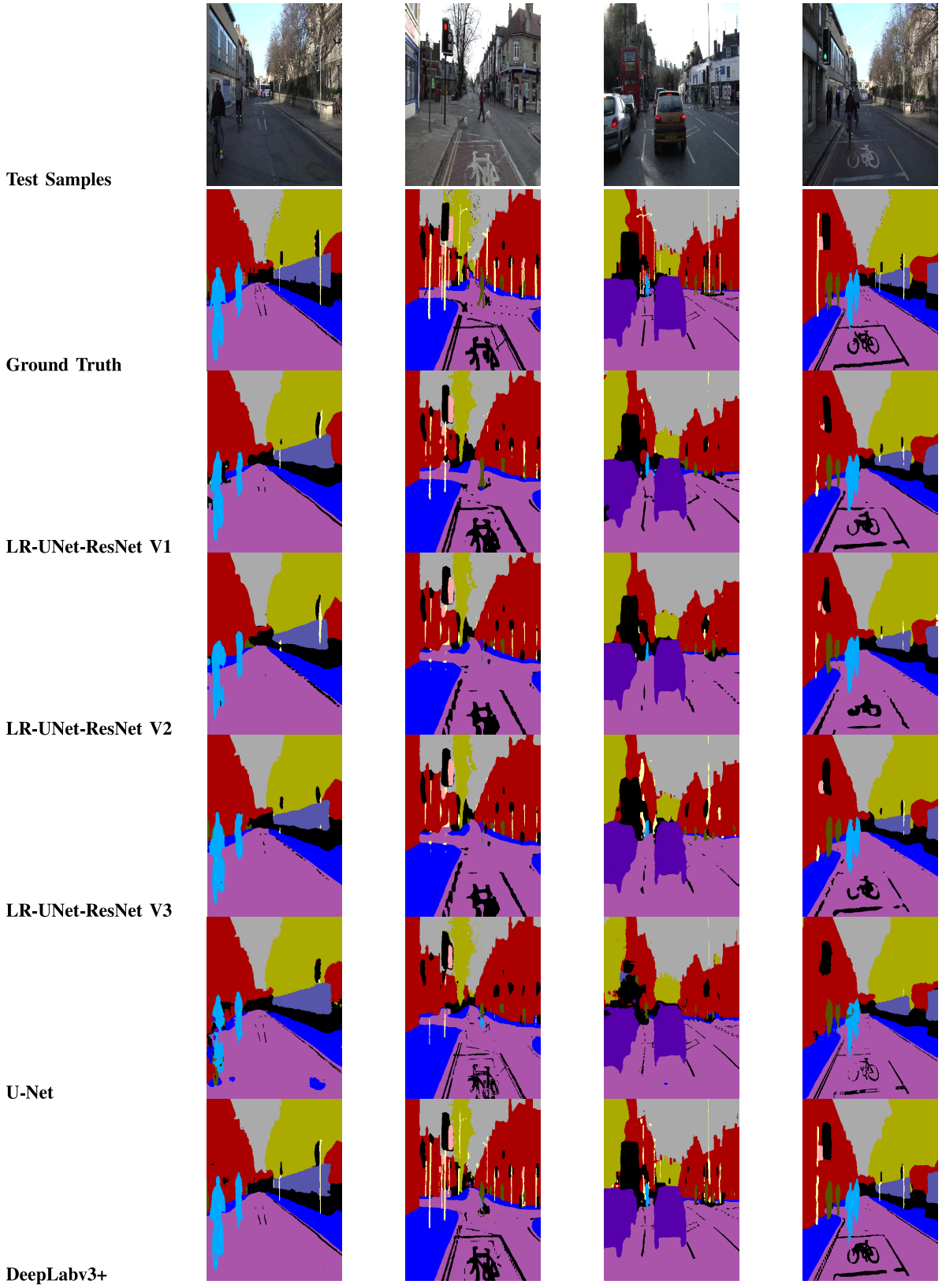

FIGURE 6. Qualitative assessment of segmentation on the CamVid test set (color code: maroon - *building*, citrus - *tree*, gray - *sky*, indigo - *car*, oriental pink - *sign*, cannon pink - *road*, olive - *pedestrian*, purple - *fence*, pine glade - *pole*, blue - *sidewalk*, deep sky blue - *bicyclist*, and black - *void*).

This method is much more stable than a simple full-compression & re-training, since at each step accuracy degradation is much lower and easy to be compensated.

At the final iteration the initial 7×7 layer is decomposed separately since it extracts the main features of the network, hence it is highly responsible for the segmentation quality.

TABLE 6. Compression parameters for UNet-ResNet V1.

	Compression factor	Optimizer	Learning rate	Epochs
stage #2	0.1	Adam	0.0005	150
stage #1	0.07	Adam	0.001	200
conv2d 7×7	0.07	Adam	0.001	200

TABLE 7. Compression parameters UNet-ResNet V2.

	Compression factor	Optimizer	Learning rate	Epochs
stage #3	0.1	Adam	0.0005	100
stage #2	0.1	Adam	0.0005	100
stage #1	0.1	Adam	0.001	150
conv2d 7×7	0.1	Adam	0.001	150

TABLE 8. Compression parameters for UNet-ResNet V3.

	Compression factor	Optimizer	Learning rate	Epochs
stage #3	0.1	Adam	0.0005	100
stage #2	0.1	Adam	0.0005	100
stage #1	0.1	Adam	0.001	150
conv2d 7×7	0.1	Adam	0.001	150

The parameters used in the described method such as the layers compression factor, learning rate and number of epochs used for the fine-tuning can be found in Table 6, 7, 8 for the three models respectively.

The compression factors were chosen accurately low so that we could obtain sufficiently low inference times when implementing models on an embedded device. Particularly we had to select lower compression factors for UNet-ResNet V1 than for V2 and V3 since V1 has a much higher complexity.

Table 9 provides the outcomes for the low-rank models resulting from the proposed compression technique.

All the models lose no more than 1.1% of accuracy with respect to their full-rank counterparts, moreover a considerably high gain in number of parameters, storage cost and computational cost is achieved. LR-UNet-ResNet V1 has no longer the best segmentation performances, this is obviously due to the higher compression applied. LR-UNet-ResNet V3 achieves the best accuracy and mIoU, but with a much lower gain in computational cost with respect to V1 and V2. LR-UNet-ResNet V2 has still the lowest complexity and worst segmentation accuracy.

C. TESTING ON EMBEDDED PLATFORMS

In order to validate the suitability of the low-rank CNN architecture previously discussed for the real-time semantic segmentation, some experiments on the low-cost, low-power embedded platforms Raspberry Pi 4 and NVIDIA Jetson Nano 2 GB, have been conducted. Besides, to meet the best performance the CNN architecture has been implemented in the high-cost GPU NVIDIA Tesla P100 PCIe 16 GB.

The former two platforms have been chosen to distinguish between the performance required by autonomous robot navigation and car navigation systems. In autonomous robot navigation systems cost and low power are the main performance required, since typically a robot proceeds at a moderate speed. Thus in this case the Raspberry Pi 4 represents a low-cost, low-power hardware solution suitable for this purpose [10], [20]–[30]. The Raspberry Pi 4 model B is based on a quad-core Cortex-A72 (ARM v8) 64-bit SoC clocked at 1.5 GHz and 4 GB of LPDDR4 SDRAM. As this board consumes 3 W when idle and 6 W under load on average,[1] it is representative of typical low-power systems. Conversely for car navigation systems low latency of SLAM algorithm is the primary performance required, due to the higher speed of a car. The NVIDIA Jetson Nano 2 GB is a compromise choice to meet the requirements of speed and low cost in this case. The NVIDIA Jetson Nano 2 GB is based on a quad-core ARM A57 at 1.43 GHz, a NVIDIA Maxwell GPU with 128 core and 2 GB of LPDDR4 SDRAM.

The experiments aim to compare the performance of LR-UNet-ResNet (V1, V2, and V3) with those achieved by the state-of-the-art networks. In particular, the following semantic segmentation architectures have been considered: UNet-MobileNetV2, U-Net [56], DeepLabv3+ [57], SegNet [41], ENet [58], ICNet [59], ERFNet [60], BiSeNet [61], Fast-SCNN [62], SwiftNetRN-18 [63], FCHarDNet-68 [64].

For the experiments on the three platforms the same metrics defined in Section VI-B together with the inference time and the number of recognized frames per second (FPS) were used as performance of the network.

All models were implemented on Google Colaboratory (GPU runtime) using TensorFlow v. 2.4.1 and TensorFlow Keras v. 2.4.0 as a backend in Python v. 3.7.10. The networks were trained for 100 epochs on the CamVid dataset partitioned as reported in previous section, by using an Adam optimizer with categorical cross entropy as loss function, a learning rate of 0.02 and a batch size of 4. All models were saved in Hierarchical Data Format version 5 (HDF5) binary data format (.h5). HDF5 is a grid file format to store structured data, that is ideal for storing multi-dimensional arrays of numbers. Keras saves models in this format, so that the weights and model configuration can be easily stored in a single file.

To run the evaluation code on Raspberry Pi 4, we used TensorFlow/Keras v. 2.4.0 with Python v. 3.7.3 on Raspbian 10 (Buster) operating system. The results for all tensors were obtained with 32-bit floating point precision (h5 model).

To perform inference on Jetson Nano 2 GB the NVIDIA TensorRT,[2] an SDK for high-performance deep learning inference on NVIDIA hardware, is used and the Keras h5 models were then converted in Open Neural Network Exchange (ONNX) format. To compute

1. https://www.raspberrypi.org/documentation/hardware/raspberrypi/power/README.md https://www.pidramble.com/wiki/benchmarks/power-consumption
2. https://developer.nvidia.com/tensorrt

TABLE 9. Comparison of the LR-UNet-ResNet V1, V2 and V3.

	Test accuracy [%]	mIoU [%]	Storage cost [MB]	Parameters number	MFLOPs
LR-UNet-ResNet V1	85.46	55.35	2.034	130k	587.04
LR-UNet-ResNet V2	84.78	52.72	6.584	508k	319.12
LR-UNet-ResNet V3	85.09	53.66	6.689	478k	452.58

TABLE 10. Comparison of proposed network with the state-of-the-art methods on the CamVid 11 road class segmentation problem in terms of storage cost, accuracy and inference time - performance on the embedded platform Raspberry Pi 4 - *Keras h5 model.*

Model	Storage cost [MB]	Compression factor	Parameters number	MFLOPs	Test accuracy [%]	Inference time CPU [s]	FPS CPU
LR-UNet-ResNet V1	**2.034**	8	**130531**	**587.04**	85.45	**0.591**	**1.70**
LR-UNet-ResNet V2	6.584	8	508 487	**319.12**	84.78	**0.461**	**2.17**
LR-UNet-ResNet V3	6.689	8	478 463	**452.58**	85.08	**0.524**	**1.91**
UNet-MobileNetV2	49.353	1	4 215 444	8840.64	83.91	2.845	0.35
U-Net [56]	**3.818**	1	**303876**	7362.83	83.55	1.114	0.89
DeepLabv3+ [57]	472.909	1	41 255 900	68 780.63	**90.13**	6.544	0.15
SegNet [41]	32.840	1	2 848 748	97 646.51	84.53	6.385	0.15
ENet [58]	5.923	1	372 208	2630.39	83.93	1.148	0.87
ICNet [59]	78.044	1	6 741 900	5376.45	83.14	1.096	0.91
ERFNet [60]	24.645	1	2 070 328	17 739.99	86.91	2.087	0.47
BiSeNet [61]	203.008	1	26 533 848	36 342.10	87.95	3.783	0.26
Fast-SCNN [62]	21.080	1	1 797 212	1620.32	83.24	0.613	1.63
SwiftNetRN-18 [63]	15.981	1	1 361 858	6147.74	83.22	0.916	1.09
FCHarDNet-68 [64]	11.463	1	1 365 864	2542.84	85.27	1.040	0.96

inference, ONNX models with FP32 precision and opset 13 have been generated and the Jetson module was run with maximum performance, which is 10W for Jetson Nano. Particularly, NVIDIA JetPack v. 4.6 that includes TensorRT v. 8.0.1 and CUDA v. 10.2 has been used.

To evaluate the performance on Raspberry Pi 4 platform, Table 10 reports the results achieved for storage cost, compression, parameter numbers (complexity), FLOPs, accuracy, inference time, and FPS, while Table 11 reports both the IoU for each class and the mIoU. The performance obtained on NVIDIA Jetson Nano 2 GB are shown in Table 12 and Table 13.

As can be seen all the LR-UNet-ResNet networks proposed in this paper outperform the other networks in terms of FLOPs, inference time and FPS. Besides, the LR-UNet-ResNet V1 has the best performance also in terms of storage cost and parameter numbers.

Comparing the performance obtained with the three versions of the LR-UNet-ResNet, the following considerations can be made.

- If the shortest inference time is the main requirement, without worrying about learning accuracy, thus the architecture V2 is the better choice.
- The solution V1 ensures the best performance in terms of storage cost.
- The architecture V3 gives a trade-off between inference time and accuracy.

About the accuracy and mIOU, the values achieved with LR-UNet-ResNet are just a few percentage below the best performances obtained with other networks, but this does not constitute a real issue for the goal, since this lack of accuracy is compensated by the greater number of frame per second the LR-UNet-ResNets are able to perform.

To a complete treatment, a comparison with other compression techniques like pruning methods was conducted. The pruning methods can be grouped into two main categories: weight pruning (WP) [107], [108] and filter (channel) pruning (FP) [109]. WP is a pruning method that discards the individual weights with low values by using a fine-grained approach, resulting in a sparse network without affecting prediction performance. However, sparse networks require to be implemented in a specialized hardware in order to obtain an actual reduction of computation cost and inference time. FP-based methods prune filters or channels within the convolution layers that give a low contribute to the total energy of the weights tensor. By removing whole filters in

TABLE 11. Comparison of proposed network with the state-of-the-art methods on the CamVid 11 road class segmentation problem in terms of IoU for each class - performance on the embedded platform Raspberry Pi 4 - *Keras h5 model.*

Model	Building	Tree	Sky	Car	Sign	Road	Pedestrian	Fence	Pole	Sidewalk	Bicyclist	mIoU
LR-UNet-ResNet V1	76.82	69.57	90.53	69.06	26.50	90.32	19.44	42.15	13.08	73.93	37.44	55.35
LR-UNet-ResNet V2	76.56	69.96	89.99	66.92	21.97	88.45	16.02	38.30	5.33	71.76	34.68	52.72
LR-UNet-ResNet V3	77.04	70.48	89.99	68.30	21.59	89.13	19.23	38.29	8.93	72.64	34.66	53.66
UNet-MobileNetV2	72.75	66.55	91.21	65.77	19.90	91.76	21.42	40.40	18.74	74.83	39.46	54.80
U-Net [56]	72.28	68.77	88.17	62.99	13.09	89.06	15.44	44.98	3.96	70.54	29.01	50.75
DeepLabv3+ [57]	85.08	77.65	92.08	81.30	33.44	92.99	37.35	57.95	27.45	82.41	50.29	**65.27**
SegNet [41]	73.29	69.84	91.13	61.77	25.89	90.41	23.42	41.97	16.80	74.39	37.30	55.11
ENet [58]	72.45	64.18	90.28	67.17	25.39	89.83	18.21	38.77	15.21	71.41	39.22	53.83
ICNet [59]	71.00	67.27	88.32	61.40	27.91	89.25	18.15	39.78	6.24	72.73	36.58	52.60
ERFNet [60]	77.62	70.82	91.56	75.81	32.83	92.17	28.33	45.85	27.35	78.91	40.55	60.16
BiSeNet [61]	82.00	75.55	90.95	75.69	28.35	90.61	24.00	53.61	8.60	79.21	41.99	59.14
Fast-SCNN [62]	73.43	68.69	89.06	63.11	20.33	86.96	15.98	38.12	4.86	69.39	31.86	51.07
SwiftNetRN-18 [63]	72.38	66.95	89.42	59.17	25.58	88.70	14.56	42.12	10.06	70.96	32.04	51.99
FCHarDNet-68 [64]	74.06	68.82	91.31	71.50	21.98	90.99	19.76	45.99	17.49	74.27	40.14	56.03

TABLE 12. Comparison of proposed network with the state-of-the-art methods on the CamVid 11 road class segmentation problem in terms of storage cost, accuracy and inference time - performance on the embedded platform NVIDIA Jetson Nano 2 GB (GPU) - *ONNX model.*

Model	Storage cost [MB]	Compression factor	Parameters number	MFLOPs	Test accuracy [%]	Inference time GPU [s]	FPS GPU
LR-UNet-ResNet V1	**2.034**	8	**130531**	**587.04**	85.46	**0.045**	**22.02**
LR-UNet-ResNet V2	6.584	8	508 487	**319.12**	84.78	**0.039**	**25.20**
LR-UNet-ResNet V3	6.689	8	478 463	**452.58**	85.09	**0.044**	**22.65**
UNet-MobileNetV2	49.353	1	4 215 444	8840.64	83.91	0.170	5.85
U-Net [56]	**3.818**	1	**303876**	7362.83	83.56	0.101	9.82
DeepLabv3+ [57]	472.909	1	41 255 900	68 780.63	**90.13**	2.451	0.41
SegNet [41]	32.840	1	2 848 748	97 646.51	84.54	0.415	2.41
ENet [58]	5.923	1	372 208	2630.39	83.94	0.094	10.59
ICNet [59]	78.044	1	6 741 900	5376.45	83.14	0.098	10.20
ERFNet [60]	24.645	1	2 070 328	17 739.99	86.92	0.147	6.78
BiSeNet [61]	203.008	1	26 533 848	36 342.10	87.95	2.531	0.40
Fast-SCNN [62]	21.080	1	1 797 212	1620.32	83.24	0.047	21.17
SwiftNetRN-18 [63]	15.981	1	1 361 858	6147.74	83.22	0.059	16.71
FCHarDNet-68 [64]	11.463	1	1 365 864	2542.84	85.27	0.073	13.70

the network together with their connecting feature maps, the computation costs are reduced significantly without sparse connectivity patterns.

Table 14 shows the results achieved by applying WP and FP compression methods to the UNet-ResNet (V1, V2, and V3). In the first two rows of each model we reported the full rank and low rank versions of the proposed networks for convenience. The compression factor is almost the same for all the compressed networks. As you can see the CP method outperforms both weight and filters pruning in terms of accuracy. As far as the inference time is concerned filters method is able to obtain the best performance. However, it is worth to notice that while a wide range of optimization techniques have been proposed to perform standard convolutions, such as fast fourier transform (FFT) [110], winograd (Winograd) [111]

TABLE 13. Comparison of proposed network with the state-of-the-art methods on the CamVid 11 road class segmentation problem in terms of IoU for each class - performance on the embedded platform NVIDIA Jetson Nano 2 GB (GPU) - *ONNX model.*

Model	Building	Tree	Sky	Car	Sign	Road	Pedestrian	Fence	Pole	Sidewalk	Bicyclist	mIoU
LR-UNet-ResNet V1	76.82	69.57	90.53	69.06	26.50	90.32	19.44	42.15	13.08	73.93	37.44	55.35
LR-UNet-ResNet V2	76.56	69.96	89.99	66.92	21.97	88.45	16.02	38.30	5.33	71.76	34.68	52.72
LR-UNet-ResNet V3	77.04	70.48	89.99	68.30	21.59	89.13	19.23	38.29	8.93	72.64	34.66	53.66
UNet-MobileNetV2	72.75	66.55	91.21	65.77	19.90	91.76	21.42	40.40	18.74	74.83	39.46	54.80
U-Net [56]	72.28	68.77	88.17	62.99	13.09	89.06	15.44	44.98	3.96	70.54	29.01	50.75
DeepLabv3+ [57]	85.08	77.65	92.08	81.30	33.44	92.99	37.35	57.95	27.45	82.41	50.29	**65.27**
SegNet [41]	73.29	69.84	91.13	61.77	25.89	90.41	23.42	41.97	16.80	74.39	37.30	55.11
ENet [58]	72.45	64.18	90.28	67.17	25.39	89.83	18.21	38.77	15.21	71.41	39.22	53.83
ICNet [59]	71.00	67.27	88.32	61.40	27.91	89.25	18.15	39.78	6.24	72.73	36.58	52.60
ERFNet [60]	77.62	70.82	91.56	75.81	32.83	92.17	28.33	45.85	27.35	78.91	40.55	60.16
BiSeNet [61]	82.00	75.55	90.95	75.69	28.35	90.61	24.00	53.61	8.60	79.21	41.99	59.14
Fast-SCNN [62]	73.43	68.69	89.06	63.11	20.33	86.96	15.98	38.12	4.86	69.39	31.86	51.07
SwiftNetRN-18 [63]	72.38	66.95	89.42	59.17	25.58	88.70	14.56	42.12	10.06	70.96	32.04	51.99
FCHarDNet-68 [64]	74.06	68.82	91.31	71.50	21.98	90.99	19.76	45.99	17.49	74.27	40.14	56.03

TABLE 14. Comparison of the UNet-ResNet V1, V2, V3 models with the state-of-the-art compression methods - performance on the embedded platforms: Raspberry Pi 4 (CPU) and NVIDIA Jetson Nano 2 GB (GPU) - *Keras h5 model for Raspberry Pi 4 and ONNX model for NVIDIA Jetson Nano 2 GB.*

Model	Pruning method	Compression factor	Storage cost [MB]	Parameters number	MFLOPs	Test accuracy [%]	mIoU	Inference time CPU [s]	FPS CPU	Inference time GPU [s]	FPS GPU
UNet-ResNet V1	FR	1	12.663	1 077 497	5209.37	86.58	56.67	0.859	1.16	0.053	18.77
	LR	8	2.034	130 531	587.04	85.46	55.35	0.591	1.69	0.045	22.02
	weights	8	12.663	137 212	5209.37	84.98	54.10	0.808	1.23	0.053	18.60
	filters	9	1.755	123 223	556.47	82.24	49.62	0.472	2.12	0.035	28.57
UNet-ResNet V2	FR	1	46.619	4 033 273	2340.46	85.54	53.94	0.569	1.76	0.041	24.20
	LR	8	6.584	508 487	319.12	84.78	52.72	0.461	2.17	0.039	25.20
	weights	8	46.62	501 807	2340.46	84.17	51.52	0.540	1.85	0.041	24.11
	filters	8	5.959	478 463	252.98	82.05	48.79	0.361	2.77	0.030	33.09
UNet-ResNet V3	FR	1	46.725	4 041 465	2711.12	85.93	54.88	0.653	1.53	0.046	21.70
	LR	8	6.689	478 463	452.58	85.09	53.66	0.524	1.91	0.044	22.65
	weights	8	46.725	509 999	2711.12	84.75	52.96	0.650	1.53	0.046	21.53
	filters	8	6.067	486 655	312.03	81.98	49.03	0.395	2.53	0.032	30.39

and general matrix multiplication (GEMM) [112], these solutions offer little benefit for depthwise convolutions. This is because such techniques are designed to optimize arithmetic computation, but not memory access latency, which often dominates the execution time of depthwise convolution [113], [114] due to its lower arithmetic operations compared to a standard convolution. Thus, these optimizated techniques used to implement efficiently filters pruning method are the major reason for the best performance obtained in terms of inference time. Nevertheless, the accuracy achieved with this method is very low in comparison to the other techniques. Therefore, the proposed networks ensure the best compromise between accuracy and inference time, thus showing suitable performance for semantic segmentation task both in autonomous driving vehicles and robots.

Fig. 6 shows the qualitative comparisons of the LR-UNet-ResNet (V1, V2, and V3) predictions with the state-of-the-art networks. For this comparison, the architectures U-Net and DeepLabv3+ have been chosen, since, according to Table 10, they outperform the selected state-of-the-art networks in terms of storage cost and accuracy, respectively.

Besides, in order to show the capability of the proposed CNN architecture to reach low/ultra-low latency with suitable hardware, Table 15 and Table 16 report the performance achieved on desktop (Intel Core i7-6800K CPU with 3.40 GHz and 32 GB of RAM and GPU NVIDIA Tesla P100 PCIe 16 GB). As expected storage cost, compression, complexity and accuracy keep unchanged, while inference time is scaled by about an order of magnitude.

TABLE 15. Comparison of proposed network with the state-of-the-art methods on the CamVid 11 road class segmentation problem in terms of storage cost, accuracy and inference time - performance on desktop - *Keras h5 model for CPU using TensorFlow and ONNX model for GPU using NVIDIA TensorRT.*

Model	Storage cost [MB]	Compression factor	Parameters number	MFLOPs	Test accuracy [%]	Inference time CPU [s]	FPS CPU	Inference time GPU [s]	FPS GPU
LR-UNet-ResNet V1	**2.034**	8	**130531**	**587.04**	85.46	0.091	10.95	**0.0028**	**359.54**
LR-UNet-ResNet V2	6.584	8	508 487	**319.12**	84.78	**0.071**	**14.08**	**0.0028**	**354.96**
LR-UNet-ResNet V3	6.689	8	478 463	**452.58**	85.09	**0.080**	**12.46**	**0.0033**	**303.44**
UNet-MobileNetV2	49.353	1	4 215 444	8840.64	83.91	0.465	2.15	0.0075	134.04
U-Net [56]	**3.818**	1	**303876**	7362.83	83.55	0.188	5.31	0.0036	277.28
DeepLabv3+ [57]	472.909	1	41 255 900	68 780.63	**90.13**	0.896	1.11	0.0193	51.81
SegNet [41]	32.840	1	2 848 748	97 646.51	84.54	0.926	1.07	0.0109	91.85
ENet [58]	5.923	1	372 208	2630.39	83.94	0.132	7.57	0.0053	189.35
ICNet [59]	78.044	1	6 741 900	5376.45	83.14	0.155	6.45	0.0052	192.51
ERFNet [60]	24.645	1	2 070 328	17 739.99	86.91	0.264	3.78	0.0061	163.10
BiSeNet [61]	203.008	1	26 533 848	36 342.10	87.96	0.510	1.96	0.0116	85.99
Fast-SCNN [62]	21.080	1	1 797 212	1620.32	83.24	0.087	11.49	0.0035	288.96
SwiftNetRN-18 [63]	15.981	1	1 361 858	6147.74	83.22	0.091	10.96	0.0056	179.71
FCHarDNet-68 [64]	11.463	1	1 365 864	2542.84	85.27	0.135	7.40	0.0076	131.66

TABLE 16. Comparison of proposed network with the state-of-the-art methods on the CamVid 11 road class segmentation problem in terms of IoU for each class - performance on desktop - *Keras h5 model.*

Model	Building	Tree	Sky	Car	Sign	Road	Pedestrian	Fence	Pole	Sidewalk	Bicyclist	mIoU
LR-UNet-ResNet V1	76.82	69.52	90.53	69.06	26.50	90.32	19.44	42.15	13.08	73.93	37.44	55.35
LR-UNet-ResNet V2	76.59	69.96	89.99	66.92	21.97	88.45	16.02	38.30	5.33	71.76	34.68	52.72
LR-UNet-ResNet V3	77.04	70.48	89.99	68.30	21.59	89.13	19.23	38.29	8.93	72.64	34.66	53.66
UNet-MobileNetV2	72.75	66.55	91.21	65.77	19.90	91.76	21.43	40.43	18.74	74.82	39.47	54.80
U-Net [56]	72.28	68.77	88.17	62.99	13.09	89.06	15.44	44.98	3.96	70.54	29.01	50.75
DeepLabv3+ [57]	85.08	77.65	92.08	81.30	33.44	92.99	37.35	57.95	27.45	82.41	50.29	**65.27**
SegNet [41]	73.29	69.84	91.13	61.77	25.89	90.41	23.42	41.97	16.80	74.39	37.30	55.11
ENet [58]	72.40	64.20	90.30	67.20	25.40	89.80	18.20	38.80	15.20	74.10	39.20	53.80
ICNet [59]	71.00	67.27	88.32	61.40	27.91	89.25	18.15	39.78	6.24	72.73	36.58	52.60
ERFNet [60]	77.62	70.82	91.56	75.81	32.83	92.17	28.33	45.85	27.35	78.91	40.55	60.16
BiSeNet [61]	82.00	75.55	90.95	75.69	28.35	90.61	24.00	53.61	8.60	79.21	41.99	59.14
Fast-SCNN [62]	73.43	68.69	89.06	63.11	20.33	86.96	15.98	30.12	4.86	69.39	31.86	51.07
SwiftNetRN-18 [63]	72.38	66.95	89.42	59.17	25.58	88.70	14.56	42.12	10.06	70.96	32.04	51.99
FCHarDNet-68 [64]	74.06	68.82	91.31	71.50	21.98	90.99	19.76	45.99	17.49	74.27	40.14	56.03

VII. CONCLUSION

One of the main challenges in autonomous driving for smart vehicles and smart robots, is to implement the real-time functional modules, such as location, perception and so on, on a low-power embedded platform at an affordable price. Semantic segmentation is one of the most critical tasks of autonomous driving, since it requires massive computation and storage resources as well as fast real-time performance. This paper shows that adopting a CNN compression technique based on tensor decomposition, and an architecture that combines the U-Net structure and the ResNet residual blocks, a new architecture named UNet-ResNet that is suitable for real-time semantic segmentation, can be derived. The networks proposed in this paper outperform state-of-the-art networks both in terms of complexity, and accuracy, as well as inference time and storage cost. To demonstrate the superiority of the proposed approach a large experimentation on the low-cost, low-power Raspberry Pi 4 and on two GPUs platforms have been conducted.

REFERENCES

[1] N. Atif, M. Bhuyan, and S. Ahamed, "A review on semantic segmentation from a modern perspective," in *Proc. Int. Conf. Elect. Electron. Comput. Eng. (UPCON)*, 2019, pp. 1–6.

[2] Z.-W. Hong *et al.*, "Virtual-to-real: Learning to control in visual semantic segmentation," in *Proc. 27th Int. Joint Conf. Artif. Intell. (IJCAI)*, Jul. 2018, pp. 4912–4920.

[3] C. Gao, X. Zhang, and H. Wang, "A combined method for multi-class image semantic segmentation," *IEEE Trans. Consum. Electron.*, vol. 58, no. 2, pp. 596–604, May 2012.

[4] L. Chen *et al.*, "DRINet for medical image segmentation," *IEEE Trans. Med. Imag.*, vol. 37, no. 11, pp. 2453–2462, Nov. 2018.

[5] R. Li, S. Wang, and D. Gu, "Ongoing evolution of visual SLAM from geometry to deep learning: Challenges and opportunities," *Cogn. Comput.*, vol. 10, no. 6, pp. 875–889, 2018.

[6] C. Cadena *et al.*, "Past, present, and future of simultaneous Localization and mapping: Toward the robust-perception age," *IEEE Trans. Robot.*, vol. 32, no. 6, pp. 1309–1332, Dec. 2016.

[7] N. Engelhard, F. Endres, J. Hess, J. Sturm, and W. Burgard, "Real-time 3D visual SLAM with a hand-held RGB-D camera," in *Proc. RGB-D Workshop 3D Perception Robot. Eur. Robot. Forum*, vol. 180. Vasteras, Sweden, 2011, pp. 1–15.

[8] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2D LIDAR SLAM," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2016, pp. 1271–1278.

[9] J. Tang, S. Liu, L. Liu, B. Yu, and W. Shi, "LoPECS: A low-power edge computing system for real-time autonomous driving services," *IEEE Access*, vol. 8, pp. 30467–30479, 2020.

[10] O. C. B. Silveira, J. G. O. C. de Melo, L. A. S. Moreira, J. B. N. G. Pinto, L. R. L. Rodrigues, and P. F. F. Rosa, "Evaluating a visual simultaneous Localization and mapping solution on embedded platforms," in *Proc. IEEE 29th Int. Symp. Ind. Electron. (ISIE)*, 2020, pp. 530–535.

[11] T. Peng, D. Zhang, D. L. N. Hettiarachchi, and J. Loomis, "An evaluation of embedded GPU systems for visual SLAM algorithms," *Electron. Imag.*, vol. 2020, no. 6, pp. 325–331, 2020.

[12] T. Peng, D. Zhang, R. Liu, V. K. Asari, and J. S. Loomis, "Evaluating the power efficiency of visual SLAM on embedded GPU systems," in *Proc. IEEE Nat. Aerosp. Electron. Conf. (NAECON)*, 2019, pp. 117–121.

[13] R. Giubilato, S. Chiodini, M. Pertile, and S. Debei, "An evaluation of ROS-compatible stereo visual SLAM methods on a nVidia Jetson TX2," *Measurement*, vol. 140, pp. 161–170, Jul. 2019.

[14] S. Aldegheri, N. Bombieri, D. D. Bloisi, and A. Farinelli, "Data flow ORB-SLAM for real-time performance on embedded GPU boards," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2019, pp. 5370–5375.

[15] J. Tang, L. Ericson, J. Folkesson, and P. Jensfelt, "GCNv2: Efficient correspondence prediction for real-time SLAM," *IEEE Robot. Autom. Lett.*, vol. 4, no. 4, pp. 3505–3512, Oct. 2019.

[16] A. Bokovoy, K. Muravyev, and K. Yakovlev, "Real-time vision-based depth reconstruction with NVidia Jetson," in *Proc. Eur. Conf. Mobile Robots (ECMR)*, 2019, pp. 1–6.

[17] T. Ma, N. Bai, W. Shi, L. Wang, and T. Wu, "Research and application of visual SLAM based on embedded GPU," in *Proc. Int. Conf. Heterogeneous Netw. Qual. Rel. Security Robustness*, 2020, pp. 3–21.

[18] T. Ma *et al.*, "Research on the application of visual SLAM in embedded GPU," *Wireless Commun. Mobile Comput.*, vol. 2021, Jun. 2021, Art. no. 6691262.

[19] J. Jeon, S. Jung, E. Lee, D. Choi, and H. Myung, "Run your visual-inertial Odometry on NVIDIA Jetson: Benchmark tests on a micro aerial vehicle," *IEEE Robot. Autom. Lett.*, vol. 6, no. 3, pp. 5332–5339, Jul. 2021.

[20] P. Phueakthong and J. Varagul, "A development of mobile robot based on ROS2 for navigation application," in *Proc. Int. Electron. Symp. (IES)*, 2021, pp. 517–520.

[21] A. Torresani, F. Menna, R. Battisti, and F. Remondino, "A V-SLAM guided and portable system for photogrammetric applications," *Remote Sens.*, vol. 13, no. 12, p. 2351, 2021.

[22] P. Huang, L. Zeng, K. Luo, J. Guo, Z. Zhou, and X. Chen, "ColaSLAM: Real-time multi-robot collaborative laser SLAM via edge computing," in *Proc. IEEE/CIC Int. Conf. Commun. China (ICCC)*, 2021, pp. 242–247.

[23] M. Liao, D. Wang, and H. Yang, "Deploy indoor 2D laser SLAM on a Raspberry Pi-based mobile robot," in *Proc. 11th Int. Conf. Intell. Human–Mach. Syst. Cybern. (IHMSC)*, vol. 2, 2019, pp. 7–10.

[24] K. Krinkin, E. Stotskaya, and Y. Stotskiy, "Design and implementation Raspberry Pi-based omni-wheel mobile robot," in *Proc. Artif. Intell. Nat. Lang. Inf. Extraction Soc. Media Web Search FRUCT Conf. (AINL-ISMW FRUCT)*, 2015, pp. 39–45.

[25] L. D. S. Pinto, L. E. S. A. Filho, L. Mariga, C. L. N. Júnior, and W. C. Cunha, "EKF-SLAM with autonomous exploration using a low cost robot," in *Proc. IEEE Int. Syst. Conf. (SysCon)*, 2021, pp. 1–7.

[26] J. Lomps, A. Lind, and A. Hadachi, "Evaluation of the robustness of visual SLAM methods in different environments," 2020, *arxiv.abs/2009.05427*.

[27] Y. K. Tee and Y. C. Han, "LiDAR-based 2D SLAM for mobile robot in an indoor environment: A review," in *Proc. Int. Conf. Green Energy Comput. Sustain. Technol. (GECOST)*, 2021, pp. 1–7.

[28] H. A. Miranto, A. N. Jati, and C. Setianingsih, "Realization of point cloud maps using ROS & visual sensor on Raspberry Pi 3D based mobile robot," in *Proc. 4th Int. Conf. Inf. Technol. Inf. Syst. Elect. Eng. (ICITISEE)*, 2019, pp. 517–522.

[29] F. Zhen, G. Yanning, H. Peng, and Z. Shaojiang, "The implementation of visual odometer based on Raspberry Pi and robot operating system," in *Proc. IEEE CSAA Guid. Navig. Control Conf. (CGNCC)*, 2018, pp. 1–6.

[30] D. T. Son, M. T. Anh, D. D. Tu, L. Van Chuong, T. H. Cuong, and H. S. Phuong, "The practice of mapping-based navigation system for indoor robot with RPLIDAR and Raspberry Pi," in *Proc. Int. Conf. System Sci. Eng. (ICSSE)*, 2021, pp. 279–282.

[31] Z. Tu, "Auto-context and its application to high-level vision tasks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2008, pp. 1–8.

[32] J. Shotton, J. Winn, C. Rother, and A. Criminisi, "TextonBoost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context," *Int. J. Comput. Vis.*, vol. 81, no. 1, pp. 2–23, 2009.

[33] M. Johnson and J. Shotton, *Semantic Texton Forests*. Berlin, Germany: Springer, 2010, pp. 173–203.

[34] B. Fulkerson, A. Vedaldi, and S. Soatto, "Class segmentation and object localization with superpixel neighborhoods," in *Proc. IEEE 12th Int. Conf. Comput. Vis.*, 2009, pp. 670–677.

[35] J. Carreira, R. Caseiro, J. Batista, and C. Sminchisescu, "Semantic segmentation with second-order pooling," in *Proc. Comput. Vis. ECCV*, 2012, pp. 430–443.

[36] P. Krähenbühl and V. Koltun, "Efficient inference in fully connected CRFs with Gaussian edge potentials," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 24, 2011, pp. 109–117.

[37] X. He, R. S. Zemel, and M. A. Carreira-Perpinan, "Multiscale conditional random fields for image labeling," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, vol. 2, 2004, pp. 695–702.

[38] L. Ladický, C. Russell, P. Kohli, and P. H. S. Torr, "Associative hierarchical CRFs for object class image segmentation," in *Proc. IEEE 12th Int. Conf. Comput. Vis.*, 2009, pp. 739–746.

[39] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2015, pp. 3431–3440.

[40] W. Liu, A. Rabinovich, and A. C. Berg, "ParseNet: Looking wider to see better," 2015, *arxiv.abs/1506.04579*.

[41] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A deep convolutional encoder–decoder architecture for image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 12, pp. 2481–2495, Dec. 2017.

[42] A. Kendall, V. Badrinarayanan, and R. Cipolla, "Bayesian SegNet: Model uncertainty in deep convolutional encoder–decoder architectures for scene understanding," 2015, *arxiv.abs/1511.02680*.

[43] S. Zheng *et al.*, "Conditional random fields as recurrent neural networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2015, pp. 1529–1537.

[44] A. Arnab, S. Jayasumana, S. Zheng, and P. H. Torr, "Higher order conditional random fields in deep neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 524–540.

[45] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.

[46] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "DeepLab: Semantic image segmentation with deep convolutional nets, Atrous convolution, and fully connected CRFs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 834–848, Apr. 2018.

[47] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected CRFs," 2014, *arXiv:1412.7062*.

[48] L. Chen, Y. Yang, J. Wang, W. Xu, and A. L. Yuille, "Attention to scale: Scale-aware semantic image segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 3640–3649.

[49] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," 2015, *arXiv:1511.07122*.

[50] Z. Wu, C. Shen, and A. van den Hengel, "High-performance semantic segmentation using very deep fully convolutional networks," 2016, *arxiv.abs/1604.04339*.

[51] Z. Wu, C. Shen, and A. van den Hengel, "Wider or deeper: Revisiting the ResNet model for visual recognition," 2016, *arxiv.abs/1611.10080*.

[52] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 2881–2890.

[53] C. Hazirbas, L. Ma, C. Domokos, and D. Cremers, "FuseNet: Incorporating depth into semantic segmentation via fusion-based CNN architecture," in *Proc. Comput. Vis. ACCV*, 2017, pp. 213–228.

[54] A. Valada, G. Oliveira, T. Brox, and W. Burgard, "Towards robust semantic segmentation using deep fusion," in *Proc. Robot. Sci. Syst. (RSS) Workshop*, vol. 114, 2016, p. 9.

[55] A. Valada, J. Vertens, A. Dhall, and W. Burgard, "AdapNet: Adaptive semantic segmentation in adverse environmental conditions," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2017, pp. 4644–4651.

[56] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Proc. Med. Image Comput. Comput.-Assist. Intervent. (MICCAI)*, 2015, pp. 234–241.

[57] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder–decoder with Atrous separable convolution for semantic image segmentation," in *Proc. Comput. Vis. ECCV*, 2018, pp. 833–851.

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

[59] H. Zhao, X. Qi, X. Shen, J. Shi, and J. Jia, "ICNet for real-time semantic segmentation on high-resolution images," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 405–420.

[60] E. Romera, J. M. Álvarez, L. M. Bergasa, and R. Arroyo, "ERFNet: Efficient residual factorized ConvNet for real-time semantic segmentation," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 1, pp. 263–272, Jan. 2018.

[61] C. Yu, J. Wang, C. Peng, C. Gao, G. Yu, and N. Sang, "BiSeNet: Bilateral segmentation network for real-time semantic segmentation," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 325–341.

[62] R. P. K. Poudel, S. Liwicki, and R. Cipolla, "Fast-SCNN: Fast semantic segmentation network," 2019, *arxiv.abs/1902.04502*.

[63] M. Oršic, I. Krešo, P. Bevandic, and S. Šegvic, "In defense of pre-trained ImageNet architectures for real-time semantic segmentation of road-driving images," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 12599–12608.

[64] P. Chao, C.-Y. Kao, Y.-S. Ruan, C.-H. Huang, and Y.-L. Lin, "HarDNet: A low memory traffic network," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 3552–3561.

[65] P. Yin, R. Yuan, Y. Cheng, and Q. Wu, "Deep guidance network for biomedical image segmentation," *IEEE Access*, vol. 8, pp. 116106–116116, 2020.

[66] L. Jing, Y. Chen, and Y. Tian, "Coarse-to-fine semantic segmentation from image-level labels," in *Proc. CVPR*, 2018, pp. 1–10.

[67] S. Pan, Y. Tao, C. Nie, and Y. Chong, "PEGNet: Progressive edge guidance network for semantic segmentation of remote sensing images," *IEEE Geosci. Remote Sens. Lett.*, vol. 18, no. 4, pp. 637–641, Apr. 2021.

[68] L. V. Tran and H.-Y. Lin, "BiLuNetICP: A deep neural network for object semantic segmentation and 6D pose recognition," *IEEE Sensors J.*, vol. 21, no. 10, pp. 11748–11757, May 2021.

[69] G. Dong, Y. Yan, C. Shen, and H. Wang, "Real-time high-performance semantic image segmentation of urban street scenes," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 6, pp. 3258–3274, Jun. 2021.

[70] J. Shotton, M. Johnson, and R. Cipolla, "Semantic texton forests for image categorization and segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2008, pp. 1–8.

[71] P. Sturgess, K. Alahari, L. Ladicky, and P. Torr, "Combining appearance and structure from motion features for road scene understanding," in *Proc. BMVC*, Sep. 2009, pp. 1–11.

[72] Ľ. Ladický, P. Sturgess, K. Alahari, C. Russell, and P. H. S. Torr, "What, where and how many? Combining object detectors and CRFs," in *Proc. Comput. Vis. ECCV*, 2010, pp. 424–437.

[73] P. Kontschieder, S. R. Bulò, H. Bischof, and M. Pelillo, "Structured class-labels in random forests for semantic image labelling," in *Proc. Int. Conf. Comput. Vis.*, 2011, pp. 2190–2197.

[74] C. Zhang, L. Wang, and R. Yang, "Semantic segmentation of urban scenes using dense depth maps," in *Proc. Comput. Vis. (ECCV)*, 2010, pp. 708–721.

[75] J. Tighe and S. Lazebnik, "SuperParsing: Scalable nonparametric image parsing with Superpixels," in *Proc. Comput. Vis. (ECCV)*, 2010, pp. 352–365.

[76] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, "Indoor segmentation and support inference from RGBD images," in *Proc. Comput. Vis. (ECCV)*, 2012, pp. 746–760.

[77] X. Ren, L. Bo, and D. Fox, "RGB-(D) scene labeling: Features and algorithms," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2012, pp. 2759–2766.

[78] P. Pinheiro and R. Collobert, "Recurrent convolutional neural networks for scene labeling," in *Proc. 31st Int. Conf. Mach. Learn.*, vol. 32, Jun. 2014, pp. 82–90.

[79] S. Hong, H. Noh, and B. Han, "Decoupled deep neural network for semi-supervised semantic segmentation," 2015, *arxiv.abs/1506.04924*.

[80] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. ICLR*, 2015, pp. 1–6.

[81] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 25, 2012, pp. 1106–1114.

[82] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *Proc. ICCV*, 2015, pp. 1520–1528.

[83] L. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking Atrous convolution for semantic image segmentation," 2017, *arxiv.abs/1706.05587*.

[84] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 770–778.

[85] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4510–4520.

[86] Y. L. Cun, J. S. Denker, and S. A. Solla, *Optimal Brain Damage*. San Francisco, CA, USA: Morgan Kaufmann, 1990, pp. 598–605.

[87] S. J. Hanson and L. Y. Pratt, "Comparing biases for minimal network construction with back-propagation," in *Proc. 1st Int. Conf. Neural Inf. Process. Syst. (NIPS)*, 1988, pp. 177–185.

[88] N. Ström, "Phoneme probability estimation with dynamic sparsely connected artificial neural networks," *Free Speech J.*, vol. 5, nos. 1–41, p. 2, 1997.

[89] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proc. IEEE*, vol. 108, no. 4, pp. 485–532, Apr. 2020.

[90] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to $+1$ or -1," 2016, *arxiv.abs/1602.02830*.

[91] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Rev.*, vol. 51, no. 3, pp. 455–500, Sep. 2009.

[92] A. Cichocki *et al.*, "Tensor decompositions for signal processing applications: From two-way to multiway component analysis," *IEEE Signal Process. Mag.*, vol. 32, no. 2, pp. 145–163, Mar. 2015.

[93] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, "Tensor decomposition for signal processing and machine learning," *IEEE Trans. Signal Process.*, vol. 65, no. 13, pp. 3551–3582, Jul. 2017.

[94] S. Lin, R. Ji, C. Chen, D. Tao, and J. Luo, "Holistic CNN compression via low-rank decomposition with knowledge transfer," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 12, pp. 2889–2905, Dec. 2019.

[95] A. Cichocki, N. Lee, I. Oseledets, A.-H. Phan, Q. Zhao, and D. P. Mandic, "Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions," *Found. Trends Mach. Learn.*, vol. 9, nos. 4–5, pp. 249–429, Dec. 2016.

[96] R. A. Harshman, "The problem and nature of degenerate solutions or decompositions of 3-way arrays," in *Proc. Talk Tensor Decompositions Workshop*, Palo Alto, CA, USA, 2004, p. 79.

[97] A.-H. Phan, P. Tichavský, and A. Cichocki, "Error preserving correction: A method for CP decomposition at a target error bound," *IEEE Trans. Signal Process.*, vol. 67, no. 5, pp. 1175–1190, Mar. 2019.

[98] A.-H. Phan *et al.*, "Stable low-rank tensor decomposition for compression of convolutional neural network," in *Proc. Comput. Vis. ECCV*, 2020, pp. 522–539.

[99] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015, *arxiv.abs/1512.03385*.

[100] Z. Chu, T. Tian, R. Feng, and L. Wang, "Sea-land segmentation with RES-UNet and fully connected CRF," in *Proc. IEEE Int. Geosci. Remote Sens. Symp. (IGARSS)*, 2019, pp. 3840–3843.
[101] O. Russakovsky *et al.*, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015.
[102] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arxiv.abs/1704.04861*.
[103] F. Chollet, "XCeption: Deep learning with depthwise separable convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2017, pp. 1800–1807.
[104] "Motion-based segmentation and recognition dataset." [Online]. Available: http://mi.eng.cam.ac.uk/research/projects/VideoRec/CamVid/(Accessed: Jun. 10, 2021).
[105] G. J. Brostow, J. Shotton, J. Fauqueur, and R. Cipolla, "Segmentation and recognition using structure from motion point clouds," in *Proc. Comput. Vis. ECCV*, 2008, pp. 44–57.
[106] G. J. Brostow, J. Fauqueur, and R. Cipolla, "Semantic object classes in video: A high-definition ground truth database," *Pattern Recognit. Lett.*, vol. 30, no. 2, pp. 88–97, Jan. 2009.
[107] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," 2015, *arXiv.abs/1506.02626*.
[108] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding," in *Proc. 4th Int. Conf. Learn. Rep. (ICLR)*, May 2016, pp. 1–6.
[109] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient ConvNets," in *Proc. ICLR*, 2017, pp. 1–13.
[110] Z. Li *et al.*, "AutoFFT: A template-based FFT codes auto-generation framework for ARM and X86 CPUs," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal. (SC)*, 2019, pp. 1–15.
[111] D. Yan, W. Wang, and X. Chu, *Optimizing Batched Winograd Convolution on GPUs*. New York, NY, USA: Assoc. Comput. Mach., 2020, pp. 32–44.
[112] A. Vasudevan, A. Anderson, and D. Gregg, "Parallel multi channel convolution using general matrix multiplication," in *Proc. IEEE 28th Int. Conf. Appl. Specif. Syst. Archit. Process. (ASAP)*, 2017, pp. 19–24.
[113] G. Lu, W. Zhang, and Z. Wang, "Optimizing Depthwise separable convolution operations on GPUs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 1, pp. 70–87, Jan. 2022.
[114] "NVIDIA, CUDA C++ best practices guide." [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html (Accessed: Apr. 23, 2022).

LAURA FALASCHETTI (Member, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees in electronic engineering from the Università Politecnica delle Marche, Ancona, Italy, in 2008, 2012, and 2016, respectively. In 2017, she joined the Department of Information Engineering, Università Politecnica delle Marche as a Postdoctoral Research Fellow, where she is currently a Postdoctoral Researcher and an Assistant Professor of Electronic Systems. Her current research interests include embedded systems, machine learning, neural networks, manifold learning, pattern recognition, signal processing, image processing, speech processing, and biosignal analysis.

LORENZO MANONI (Graduate Student Member, IEEE) received the B.Sc. and M.Sc. degrees in electronics engineering and the Ph.D. degree in information engineering from the Università Politecnica delle Marche, Ancona, Italy, in 2015, 2018, and 2022, respectively, where he is currently a Research Fellow with the Department of Information Engineering. His current research interests include signal processing, embedded systems, machine learning, convolutional neural networks, algorithms analysis and design, and biosignal analysis.

CLAUDIO TURCHETTI (Life Member, IEEE) received the Laurea degree in electronics engineering from the University of Ancona, Ancona, Italy, in 1979. He joined the Università Politecnica delle Marche, Ancona, in 1980, where he was the Head of the Department of Electronics, Artificial Intelligence and Telecommunications for five years and is currently a Full Professor of Micro-Nanoelectronics and Design of Embedded Systems. He has published more than 160 journal and conference papers, and two books. The most relevant papers were published in IEEE JOURNAL OF SOLID-STATE CIRCUITS, IEEE TRANSACTIONS ON ELECTRON DEVICES, IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, IEEE TRANSACTIONS ON SIGNAL PROCESSING, IEEE TRANSACTIONS ON CYBERNETICS, IEEE JOURNAL OF BIOMEDICAL AND HEALTH INFORMATICS, IEEE TRANSACTIONS ON CONSUMER ELECTRONICS, and *Information Sciences*. He has held a variety of positions as a Project Leader in several applied research programs developed in cooperation with small, large, and multinational companies in the field of microelectronics. His current research interests include statistical device modeling, RF integrated circuits, device modeling at nanoscale, computational intelligence, signal processing, pattern recognition, system identification, machine learning, and neural networks. He has served as a program committee member for several conferences and as a reviewer for several scientific journals. He is a member of the IEEE, Computational Intelligence and Signal processing Society. He has been an Expert Consultant of the Ministero dell'Università e Ricerca.