MiniNet: An Efficient Semantic Segmentation ConvNet for Real-time Robotic Applications

Iñigo Alonso, Luis Riazuelo, and Ana C. Murillo

Abstract—Efficient models for semantic segmentation, in terms of memory, speed, and computation, could boost many robotic applications with strong computational and temporal restrictions. This work presents a detailed analysis of different techniques for efficient semantic segmentation. Following this analysis, we have developed a novel architecture, *MiniNet-v2*, an enhanced version of *MiniNet* [1]. *MiniNet-v2* is built considering the best option depending on CPU or GPU availability. It reaches comparable accuracy to the state-of-the-art models but uses less memory and computational resources. We validate and analyze the details of our architecture through a comprehensive set of experiments on public benchmarks (Cityscapes, Camvid, and COCO-Text datasets), showing its benefits over relevant prior work. Our experiments include a sample application where these models can boost existing robotic applications.

Index Terms—Deep learning, Efficient models, Semantic segmentation, Scene understanding

I. INTRODUCTION

Research on novel efficient deep learning models is increasing the number of robotic applications that can make use of learning-based solutions. In particular, more efficient semantic segmentation models have brought a lot of attention [3], [4], [6], [33]. Semantic segmentation models compute semantic label probabilities for every image pixel, providing very rich information about the context and details within the captured scene. This task is essential for a quick scene understanding. As with many other deep learning-based solutions, it is unfeasible to run many of the top-performing models on CPUs, or even at low-powered GPUs, at the high rates required in many real applications.

CNN models usually require to have high-end GPUs to run inferences in near real-time. Unfortunately, these GPUs are often not available in robotic platforms or are not affordable for some applications running on small robots, drones or mobile phones. Even for autonomous vehicles, we cannot expect them to hold one high-end GPU for each task it has to perform. Therefore, there are many restrictions to take into account to deploy deep learning-based techniques in real-world applications. Available computational resources, power consumption and time restrictions are some of the significant constraints that robotic applications usually face. In the last years, we have seen an increasing interest in deep learning solutions for real-time applications on low-power GPUs. Several solutions [22], [24], [32] get results close to top-performing methods at much lower computational and energy cost.

I. Alonso, L. Riazuelo and A.C. Murillo are with the DIIS - I3A, Universidad de Zaragoza, Spain. {inigo, riazuelo, acm}@unizar.es
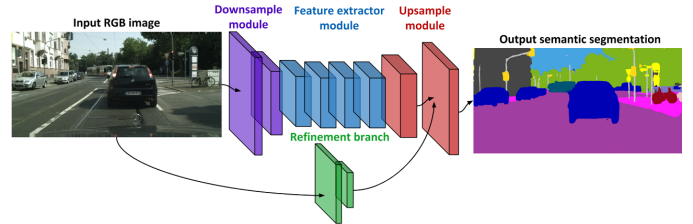

Fig. 1. MiniNet-v2: overview of the proposed architecture.

This work presents an study of efficient techniques for efficient semantic segmentation that yielded to our novel architecture for efficient semantic segmentation. *MiniNet-v2* (summarized in Fig. 1 and detailed in Table I) presents a better accuracy vs resource requirements trade-off than the state-of-the-art. It is an improved version of *MiniNet*, firstly introduced in [1]. The main improvements come from the use of our proposed multi-dilation depthwise convolution and the use of an additional convolutional branch for retrieving fine-grained information instead of using skip connections. The new architecture presented here has two main configurations: *MiniNet-v2* is focused on more general set-ups with GPU availability and *MiniNet-v2-cpu* enables many applications and tasks to run semantic segmentation models without requiring GPU on-board. For instance, we show its usefulness for V-SLAM smart key-frame selection, demonstrated in [1].

Our experiments evaluate how different CNN techniques and operations affect relevant semantic segmentation metrics, namely: GFlops (computation), the number of parameters and model size (memory), execution time (both on CPU and GPU) and Mean Intersection over Union (accuracy of the segmentation). We show that *MiniNet-v2* is more efficient overall than the state-of-the-art regarding those metrics, especially in the required memory, and still gets comparable performance on well known public datasets from autonomous driving environments, Cityscapes [7] and Camvid [5].

II. RELATED WORK

Techniques to improve CNNs efficiency: Many works lately focus on reducing CNNs memory and computational requirements, which directly affects energy consumption and execution time. We next group some of the most relevant ideas into several categories depending on what they focus on.

Some approaches focus their contributions on the *training phase*. The most common strategy is joint training and distillation [11], [23]. These techniques rely on two models,

one larger and not focused on efficiency which can get top-performing results, and a corresponding small and efficient model. The contribution relies on how to transfer the knowledge from the accurate model to the efficient one.

Other works focus their attention on carefully choosing the *parameter data types*. Quantized models [13] or binary networks [8] are recent works studying the effect of using less precise data types on the accuracy vs. memory trade-off and on the computation required.

Increased efficiency can also be achieved through *post-processing methods* such as pruning [21]. These methods study how to reduce a trained CNN without losing accuracy.

Other works study novel network *operations* such as depthwise separable convolutions [27] and dilated convolutions [9]. Finally, other works propose novel CNN architectures directly focused on the target task. In particular, for efficient segmentation we find architectures such as Enet [22] or ERFNet [24]. These works propose different modules and strategies to perform semantic segmentation more efficiently.

Our work is more closely related to the last two groups of prior work: novel network operations and novel CNN architectures. Sec. VI includes a more detailed analysis of existing network operations to improve efficiency on semantic segmentation architectures.

Semantic segmentation architectures: CNNs for semantic segmentation typically follow an encoder-decoder structure: the encoder learns features while reducing the resolution and the decoder upsamples the learned features and maps them into the segmentation result. FCN [17] is one of the early works following a fully convolutional design for semantic segmentation. They propose to add a single decoder layer at the end of a classification CNN. Their results show that just upsampling the encoder features was enough to learn the semantic segmentation. SegNet [3] follows a symmetric encoder-decoder structure achieved by adding upsampling layers, i.e., unpooling. More recent works improve these earlier segmentation architectures by adding novel operations or modules proposed initially within CNNs architectures for classification tasks. FC-DenseNet [14] follows the DenseNet work [12] using dense modules. PSPNet [33] uses ResNet [10] as its encoder and introduces the Pyramid Pooling Module, which is incorporated at the end of the CNN allowing to learn effective global contextual priors. Deeplab-v3 [6] is one of the top-performing architectures for segmentation and makes use of two powerful operations: the depthwise separable convolutions [27] and the atrous (or dilated) convolutions [9].

The semantic segmentation methods discussed so far have achieved impressive results but are not designed for computationally limited scenarios. Sometimes they rely on costly post-processing methods, e.g., CRFs or multi-scale inference, and they present too high computational requirements and inference time for embedded platforms.

Efficient segmentation architectures: As efficient semantic segmentation architectures are concerned, ENet [22] sets up certain basis which following works, such as ERFNet [24], ICNet [32], ThunderNet [30] or GUN [18], have built upon.

These architectures perform early downsampling of the feature maps so that most of the learning is performed at a low resolution (e.g., 1/8 of the input size). In these architectures, the computation is focused on the encoder which is in charge of learning the features and the decoder just upsamples them. In ERFNet [24], the authors propose an architecture inspired by Enet, but including the use of factorized convolutions [2], which reduce the number of learning parameters. ICNet [32] is a three-branch architecture that learns parameters at different resolutions and then joins the branches to compute the final result. GUN [18] is a two-branched architecture that also works at different resolutions but, differently from previous architectures, shares weights at early stages. It also propose a guided upampling operation, latter improved in Mazzini et al. [19]. Li et al. [16] perform an auto-search approach in stead of manual implemented modules to find an architecture with good speed-performance trade-off.

In the following section, we analyze and evaluate different techniques that can help building more efficient architectures.

III. Techniques for efficient semantic segmentation CNNs

This section presents a compilation of relevant ideas to build efficient semantic segmentation architectures and discusses their main insights.

A. Convolutional layers

Convolutional layers are very relevant to our work since they are the computation core in CNNs. Apart from the standard convolutional layer, there are other layers that perform the convolution in a different way reducing the required number of parameters and operations. We consider the *factorized convolutions* and the *depthwise separable convolutions* [27] as the most relevant approximations for our goals. These ideas have been proved to perform very similar or even better than standard convolutions [24]. Figure 2 shows a comparison of the different types of convolutional layers.

Factorized convolution: This convolutional layer consists of performing two consecutive convolutions factorizing the convolutional kernel. Let $W \in \mathbb{R}^{C_i \times d \times d \times C_o}$ denote the learning parameters of a 2D convolutional layer, where C_i is the number of input channels, C_o is the number of output channels and $d \times d$ represents the kernel size of each convolution. A standard 2D convolution has $d \times d \times C_i \times C_o$ learning parameters. In contrast, a *factorized convolution* layer has two convolutions of $1 \times d$ and $d \times 1$ kernels, leading to $2 \times d \times C_i \times C_o$ learning parameters. When setting a 3×3 kernel size, the parameter reduction comes to a 33% with respect to the standard convolutional layer.

Depthwise separable convolution: It consists of factorizing the standard convolution into two separate convolutions. The first convolution is called the *depthwise convolution*. It performs a spatial convolution independently for each input channel, i.e., each output channel is only computed by one input channel in contrast to standard convolutions where all input channels are used for each output channel. Therefore, this *depthwise convolution* has $C_i \times d \times d$ parameters (where
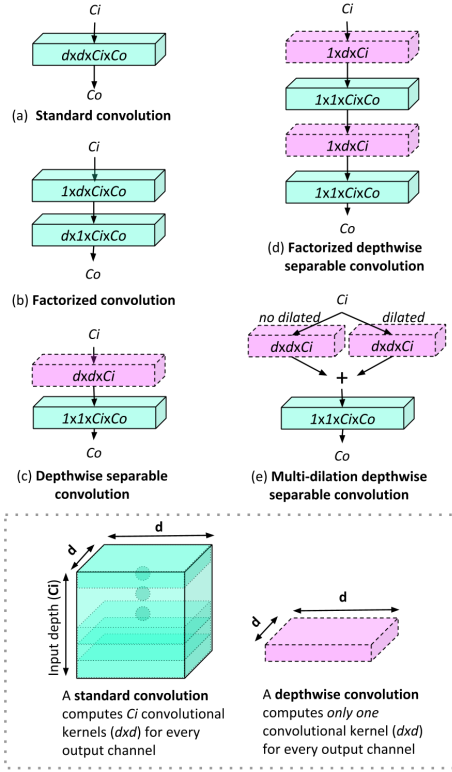
Fig. 2. Depiction of the different types of convolutions considered in this work, including required number of parameters on each step of the convolution (e.g., in $d \times d \times C_o \times C_i$: C_i is the number of input channels, C_o is the number of output channels and d is the kernel size).

$C_i = C_o$). The second convolution is called the *pointwise convolution* which consists in performing a 1×1 convolution, projecting the output channels by the depthwise convolution onto a new channel space, combining the output of the *depthwise convolution*. It has $C_i \times C_o$ learning parameters. Thus, the total learning parameters is $C_i \times d \times d + C_i \times C_o$. Setting a 3×3 kernel size and 10^3 number of kernels leads to a reduction of 88% of the learning parameters with respect to the standard convolutional layer. Thus, the *depthwise separable convolution* reduces, even more, the number of parameters than the *factorized convolution*.

Combining both ideas, *depthwise separable convolution* and *factorized convolution*, yields to $(2 \times C_i \times d + C_i \times C_o)$ learning parameters, which leads to a reduction of 88% of learning parameters with respect to a standard convolutional layer.

Atrous convolution: Atrous or dilated convolutions [9] introduce the dilatation rate r. This dilatation rate defines the stride between two adjacent kernel values. Therefore, a 3x3 kernel with $r = 2$ will have the same field-of-view as a 5x5 kernel, while only using 9 parameters instead of 25. When $r = 1$ there is no dilation and it is just a standard convolution.

This type of convolution is very important regarding efficient semantic segmentation because it allows lowering the number of layers and parameters [22], [24], [32] while being able to cover the same field-of-view. Nevertheless, this type of convolutions are not fully established in semantic segmentation architectures because big dilation rates lose local information.

Multi-dilation depthwise separable convolution: We introduce this new convolutional layer, which consists of two parallel depthwise convolutions, one with a dilation rate $r = 1$ and the other one with $r \geq 1$. Then, their outputs are added and then a pointwise convolution is applied as Fig. 2 shows.

This convolutional layer improves the performance by efficiently learning both local and global spatial relationships. It learns a larger variety of kernels with less number of layers thanks to the dilation rate. This layer only adds $C_i \times d \times d$ parameters with respect to the standard depthwise separable convolution. Therefore, it still leads to a reduction of 87% of the learning parameters with respect to the standard convolutional layer.

B. Other efficient techniques to explore

There are additional techniques that can improve efficiency in semantic segmentation architectures, which we discuss next.

Retrieving fine-grained information: Encoder-decoder architectures for CNNs reduce the resolution of the input image (between $\times 8$ and $\times 32$) when learning the features in the encoder. Working at low resolutions hinders the CNN to get detailed outputs because it does not learn well fine-grained information. Another fact that makes difficult to learn fine-grained information is stacking too many convolutional layers, because deep features encode the image context rather than local and spatial information. There are two common strategies to deal with this issue.

Several works [14], [26] connect early layers to final layers, in order to keep high-resolution information and recover fine-grained information without the need of adding more layers to the network. Nevertheless, these skip connections have some drawbacks: early layers have to extract fine-grained information for enhancing the output and useful low-level features for the encoder, which, made at the same time, may hinder the optimization.

Another strategy towards the same goal is to have a very light independent branch of convolutions for spatial information preservation [31]. This second strategy does not force early layers to extract different types of features, but adds more computational cost. Section VI includes a quantitative comparison of these strategies.

Reducing output resolution: Working on high resolutions implies a high computational cost. Therefore, apart from working at low resolutions at the encoder, another way to save computational resources is to perform the semantic segmentation predictions on a lower resolution than the input resolution removing last convolution layers and upsampling layers. This strategy has been shown to provide significant computational saving with little loss in accuracy [6].

IV. PROPOSED EFFICIENT SEMANTIC SEGMENTATION ARCHITECTURE

This section describes our architecture for efficient semantic segmentation: *MiniNet-v2*[1]. It is inspired by ERFNet [24] and our design choices are based on the benefits and drawbacks of the techniques discussed in previous section.

[1]Code at https://github.com/Shathe/MiniNet-v2

A. Main blocks

The main blocks are shown in Table I, where the whole architecture is defined. The key ingredient and the core of MiniNet-v2 architecture is its *convolutional module*. It consists of a $3x3$ depthwise separable convolution followed by a residual connection. If the module is applied on the Feature Extractor block, we use multi-dilation depthwise separable convolution instead and add a dropout of 0.25 after the convolution. All *downsample* operations consist of a max-pool operation concatenated with a strided convolution and all *upsample* operations are transposed convolutions.

1) Downsample block: This block quickly downsamples the features. It consists of combining downsample operations with convolutional modules.

2) Feature extractor block: This block is the main part of the encoder. It consists of several consecutive convolutional modules with different dilation rates.

3) Refinement block: This block extracts spatial and high resolution features performing two downsample operations to the input image. This block goal is to extract additional features that can help to refine the features previously learned in the feature extractor block.

4) Upsample block: This block upsamples the feature extractor block output. Then, the refinement block and this block are added. The last part of this block consists of several convolutional modules without dilation rate and upsample operations. The output resolution is half of the input's resolution.

B. MiniNet-v2

This new architecture enhances our previous one, firstly introduced in [1], and gets similar performance than state-of-the-art models with much larger number of parameters and memory requirements. We propose two different configurations: the main architecture, MiniNet-v2, defined in Table I, and a smaller version, MiniNet-v2-cpu, built for CPU applications. This smaller version is the same architecture removing these convolutional modules: from m3 to m10, from m16 to m25 and from m28 to m29.

MiniNet-v2 has 0.52M parameters and the model takes up to 2.02MB. At 1024x512 resolution, it works at 50fps in a TitanXp and 4fps in an Intel i5-8600k, and performs 12.89 GFLOPs (floating point operations) and 6.45 GMACs (multiply–accumulate operations), getting 70.4% MIoU on the Cityscapes benchmark.

MiniNet-v2-cpu has 0.27M parameters and the model takes up to 1.06MB. At 512x256 resolution, it works at 250fps in a TitanXp and 30fps in an Intel i5-8600k, and performs 1.68 GFLOPs and 0.84 GMACs, getting 59.9% MIoU on the Cityscapes benchmark. This smaller version gets lower segmentation accuracy but is more efficient due to smaller input resolution and the reduction on the number of convolutional layers.

V. EVALUATION

We use the following metrics and datasets for the evaluation.

TABLE I
MiniNet-v2 ARCHITECTURE FOR AN INPUT SIZE OF 1024x512.

Block	Name	Type	Input	Output size
Downsample	d1	downsampling	image	512x256x16
	d2	downsampling	d1	256x128x64
	m1	rate=1	d2	256x128x64
	m2	rate=1	m1	256x128x64
	m3	rate=1	m2	256x128x64
	m4	rate=1	m3	256x128x64
	m5	rate=1	m4	256x128x64
	m6	rate=1	m5	256x128x64
	m7	rate=1	m6	256x128x64
	m8	rate=1	m7	256x128x64
	m9	rate=1	m8	256x128x64
	m10	rate=1	m9	256x128x64
	d3	downsampling	m10	128x64x128
Feature extractor	m10	rate=1	d3	128x64x128
	m11	rate=2	m10	128x64x128
	m12	rate=1	m11	128x64x128
	m13	rate=4	m12	128x64x128
	m14	rate=1	m13	128x64x128
	m15	rate=8	m14	128x64x128
	m16	rate=1	m15	128x64x128
	m17	rate=16	m16	128x64x128
	m18	rate=1	m17	128x64x128
	m19	rate=1	m18	128x64x128
	m20	rate=1	m19	128x64x128
	m21	rate=2	m20	128x64x128
	m22	rate=1	m21	128x64x128
	m23	rate=4	m22	128x64x128
	m24	rate=1	m23	128x64x128
	m25	rate=8	m24	128x64x128
Ref	d4	downsampling	image	512x256x16
	d5	downsampling	d4	256x128x64
Upsample	up1	upsampling	m25	256x128x64
	m26	rate=1	up1 + d5	256x128x64
	m27	rate=1	m26	256x128x64
	m28	rate=1	m27	256x128x64
	m29	rate=1	m28	256x128x64
	output	upsampling	m29	512x256xN

rate: stands for the convolutional dilation rate.

A. Metrics

- **Execution time.** Inference time both on GPU (on Titan Xp using PyTorch framework) and CPU (on Intel i5-8600k).
- **Memory.** Number of parameters of the CNN and the required memory for the model (MB).
- **Computation.** GFlops (Giga Floating Point Operations) of a forward step.
- **MIoU** (Mean Intersection over Union). This is the most common metric for semantic segmentation tasks.

B. Datasets

- **Cityscapes** [7]. An urban-scene understanding dataset widely adopted to evaluate semantic segmentation approaches. It consists of 19998 coarsely annotated images and 5000 fine-annotated images (split into 2975 images for training, 500 images for validation, and 1525 images for testing). Test set evaluation is performed by submitting the test predictions on the official benchmark server.
- **Camvid** [5]. An autonomous driving dataset frequently used to train existing state-of-the-art approaches for urban areas image segmentation. It consists of 367 training images, 101 validation images, and 233 test images.
- **COCO-Text** [28]. A text detection dataset based on the MS COCO dataset. It contains over 173k text annotations in over 63k images. The dataset provides bounding boxes for the text detection task, but we convert them into pixel labels, i.e., binary segmentation (text vs non-text).

VI. ANALYSIS OF MININET-V2 COMPONENTS

This section evaluates the effects of the described techniques in Sec. III and justifies our design of *MiniNet-v2*.

TABLE II
PERFORMANCE OF FACTORIZED (ERFNET) AND NON-FACTORIZED
(MININET-V2) CONVOLUTIONS USING DEPTHWISE SEPARABLE
CONVOLUTIONS (D) OR STANDARD CONVOLUTIONS (S).

Configuration	GPU (ms)	CPU (ms)	Params (M)	GFlops	Memory (MB)	MIoU
ERFNet-S [24]	10	88	2.06	13.42	7.95	58.79
ERFNet-D	10	62	**0.49**	**3.20**	**1.93**	58.36
MiniNet-v2-S	**9**	109	3.02	19.68	11.55	**58.89**
MiniNet-v2-D	**9**	**61**	**0.49**	3.28	1.97	58.51

TABLE III
PERFORMANCE COMPARISON BETWEEN SKIP CONNECTION FROM EARLY
LAYERS (S) VERSUS A LIGHT ADDITIONAL BRANCH (I).

Configuration	GPU (ms)	CPU (ms)	Params (M)	GFlops	Memory (MB)	MIoU
MiniNet-v2-D	9	**61**	**0.49**	**3.28**	1.97	58.51
MiniNet-v2-D-S	9	62	**0.49**	**3.28**	1.97	56.89
MiniNet-v2-D-I	9	62	**0.49**	3.42	2.00	**58.71**

A. Training protocol and implementation details

The training protocol is the same for all our experiments. We train the different CNN configurations on the Cityscapes data (only fine–annotated images) for 250 epochs with a batch size of 12. We use Adam optimizer with an initial learning rate of 10^{-3} and polynomial learning rate decay schedule which power is set to 0.9. We use a weight decay of 2×10^{-4}. We use horizontal flips and shifts for data augmentation. As similar architectures, we perform the training optimization via back-propagation of the loss. The loss is calculated as the sum of all per-pixel losses, through parameter gradients using the common soft-max cross entropy loss function.

For these experiments, MiniNet-v2 components are evaluated using the Cityscapes validation set with input resolution of 512x256 (half of its original input size).

B. Techniques for efficient semantic segmentation CNNs

In the following experiments, we refer as *MiniNet-v2* (basic version) to our initial ERFNet modification consisting of replacing each ERFNet convolutional module with two *MiniNet-v2* convolutional modules (explained in Sec.IV).

1) Factorized convolutions and depthwise separable convolutions: This is the most important experiment because it is focused on variations on convolution layers, which are the main ingredient in CNNs. This experiment compares standard convolutions (S), depthwise separable convolutions (D) and multi-dilation separable convolutions (M). It also compares factorized convolutions used by ERFNet with respect to non-factorized convolutions used by *MiniNet-v2*.

Table II summarizes the results of this evaluation. The first thing to note is that standard convolutions (S) have too many parameters, FLOPs and execution time compared to depthwise convolutions. Regarding factorized convolutions, they take more execution time and present worse segmentation performance (MIoU) than non-factorized convolutions, while having the advantage of running slightly fewer FLOPs.

The approach we select for our architecture (and used in the following experiments), MiniNet-v2-D, shows several enhancements over the original ERFNet architecture (ERFNet-S): it reduces the generated computation (GFlops) by a 75% and the memory cost (number of parameters) by a 75%.

2) Retrieving fine-grained information: This experiment compares two different strategies to preserve high-resolution information and recover fine-grained information from the input image.

We consider two different configurations. In the first setup (S), we sum the output of the first downsampling operation and the output of the penultimate upsample operation. The second configuration, a new convolutional branch (I), performs two downsample operations to the input image and then adds it to the penultimate upsample operation.

Table III confirms our hypothesis on the detrimental effect of the first skip connection configuration (S). This effect is caused by forcing early layers to perform two jobs at the same time: early layers have to extract fine-grained information for enhancing the output and useful features for the rest of the encoder. This is especially important in small and efficient networks where there are very few layers and fewer parameters. In contrast, the additional convolutional branch (I) allows the network to extract fine-grained information without any negative effect. This information allows the network to get more accurate outputs, especially in contours, which is beneficial to classes with small size like traffic lights.

3) Atrous convolutions: Atrous or dilated convolutions allow adjusting the kernel field-of-view to capture multi-scale information. Nevertheless, they increase the execution time. In this experiment, we evaluate four different configurations: same dilation rates as ERFNet (E); double dilation rates than ERFNet (2); no dilated convolutions, (0); using minimum dilation rates needed to reach a field-of-view equal to the feature extractor input resolution (1). We also evaluate our proposed convolutional layer, multi-dilation depthwise separable convolution, which performs two depthwise convolutions in parallel: one with dilation rate and another one without it.

As Table IV shows, decreasing the dilation rates harms the MIoU performance while it does not improve much the efficiency. This is particularly noticeable when there are not many convolutional layers, as it usually happens in efficient architectures. This is due to a lack of context information. Increasing too much the dilation rates has the opposite effect, it lacks local information. Our architecture uses multi-dilation separable convolutions, which outperform the other methods. The improvement is mostly due to the effect of learning global and local context at the same time, thanks to the two parallel depthwise convolutions with different dilation rates.

4) Reducing output resolution: This experiment evaluates the benefits of reducing the output resolution with respect to the input's resolution. We compare the MiniNet-v2-D-I-M configuration (the best so far) with MiniNet-v2-D-I-M-R, which removes the penultimate upsampling operation and the four last *MiniNet-v2* convolutional modules that were performed at that last resolution. We only decreased the resolution ×2 to avoid loosing too much information.

TABLE IV
PERFORMANCE WHEN VARYING DILATION RATES: ERFNET DEFAULT DILATION (E), DOUBLING ERFNET DILATION (2), NO DILATION (0), MINIMUM DILATION FOR WHOLE FIELD-OF-VIEW (1) OR USING MULTI-DILATION DEPTHWISE CONVOLUTIONS (M).

Configuration	GPU (ms)	CPU (ms)	Params (M)	GFlops	Memory (MB)	MIoU
MiniNet-v2-D-I-E	**9**	61	**0.49**	**3.42**	**2.00**	58.71
MiniNet-v2-D-I-0	**9**	**55**	**0.49**	**3.42**	**2.00**	57.91
MiniNet-v2-D-I-1	**9**	60	**0.49**	**3.42**	**2.00**	58.81
MiniNet-v2-D-I-2	**9**	64	**0.49**	**3.42**	**2.00**	57.21
MiniNet-v2-D-I-M	10	69	0.53	3.66	2.10	**59.36**

TABLE V
PERFORMANCE WHEN REDUCING THE OUTPUT RESOLUTION (R).

Configuration	GPU (ms)	CPU (ms)	Params (M)	GFlops	Memory (MB)	MIoU
MiniNet-v2-D-I-M	10	74	0.53	3.66	2.10	**59.36**
MiniNet-v2-D-I-M-R	**8**	**54**	**0.52**	**3.22**	**2.02**	59.08

Table V shows that the execution time has been drastically reduced, 33% for GPU and 22% for CPU, because higher resolutions take most of the computation. Besides, the drop in the MIoU performance is not very significant. For these reasons, our proposed *MiniNet-v2* architecture follows this approach. This last configuration, MiniNet-v2-D-I-M-R, is the *MiniNet-v2* final architecture.

VII. BENCHMARK EVALUATION

This section compares our work to current state-of-the-art on different semantic segmentation problems, using Cityscapes, Camvid, and COCO-Text datasets.

A. Multi-class segmentation

Training protocol: For the Cityscapes dataset, differently from the previous section, we jointly train on the coarse-annotated and fine-annotated data. For all datasets, we train for 1M iterations with a batch size of 6, an initial learning rate of 1×10^{-3}, polynomial learning rate decay schedule which power is set to 0.9 and weight decay of 2×10^{-4}. We use horizontal flips, random scaling ($\times 0.5$, $\times 2$) and horizontal shifts for data augmentation. For the optimization we use the cross entropy loss function. To account for class imbalance, we use the median frequency class balancing, as applied in SegNet [3]. To smooth the resulting class weights, we propose to apply a power operation: $w_c = (\frac{f_{median}}{f_c})^i$, with f_c being the frequency of class c and f_{median} the median of all frequencies. We set i to 0.12. When computing the loss, instead of resizing the labels to match the output shape of our network, we resized the output for not losing information in the labeled image.

Evaluation: Table VI shows the results on the test set of the Cityscapes dataset. *MiniNet-v2* at 1024x512 resolution uses $\times 4$ less memory, parameters and FLOPs than ERFNet (our main baseline) while getting higher performance (MIoU). Our architecture gets state-of-the-art performance (MIoU) for efficient semantic segmentation, similar to ICNet or Li et al. with pretraining, but with the lowest memory requirements reported. At this resolution, it also gets better performance

TABLE VI
EVALUATION ON THE **TEST SET** OF THE **CITYSCAPES** DATASET.

Method	GPU ms (type)	Input Resolution	Params (M)	Memory (MB)	GFlops	MIoU w pt	MIoU w/o pt
Li et al. [15]	7* (1080ti)	1536x768	2.9**	—	—	71.4	—
ICNet [32]	30 (TX)	2048x1024	6.7	—	—	—	70.6
GUN [18]	27 (TXp)	1024x512	19.0	—	58.7	70.4	—
ERFNet [24]	20 (TXp)	1024x512	2.1	7.95	53.78	69.7	68.0
Mazzini et al. [19]	9 (TXp)	1024x512	1.4	—	—	68.9	63.7
CGNet [29]	56 (V100)	2048x1024	0.5	—	—	—	64.8
ThunderNet [30]	10 (TX)	1024x512	4.7	—	—	64.0	—
ESPNet [20]	9 (TX)	512x256	0.4	1.46	4.5	—	60.3
ENet [22]	13 (TX)	1024x512	0.4	1.64	8.72	—	58.3
SegNet-basic [3]	60 (TX)	480x360	29.5	112.40	286.03	—	56.1
MiniNet [1]	5 (TXp)	512x256	3.1	12.35	1.06	—	40.7
MiniNet-v2	20 (TXp), 11 (2080ti)	1024x512	0.5	2.02	12.89	—	70.5
MiniNet-v2	9 (TXp)	512x256	0.5	2.02	3.22	—	64.7
MiniNet-v2-cpu	4 (TXp)	512x256	0.3	1.06	1.68	—	59.3

w pt: with pretraining on Imagenet; TX: TitanX; TXP: TitanXP
* Reported using TensorRT. 25 ms using Caffe Time.
** Only encoder meassure is reported.

than other networks that require many more parameters like GUN, ThunderNet or SegNet. When it comes to *MiniNet-v2* at 512x256 resolution, the MIoU performance is higher (+5 IoU) than ESPNet and ESPNet using similar number of parameters, memory, FLOPs and runtime. The MIoU reached is also higher than ThuderNet's (pretrained) and Mazzini et al. (without pretraining) MIoU, with similar efficiency metrics. Our *MiniNet-v2-cpu* architecture, which is focused on CPU applications, gets similar MIoU than ENet and ESPNet while presenting better efficiency metrics.

Table VII shows a comparison of our models with the state-of-the-art using the Camvid benchmark. *MiniNet-v2* at 960x720 resolution presents a little higher MIoU than ICNet and ERFNet, and it is much more efficient in terms of both memory and speed. Comparing methods at 480x360 resolution, *MiniNet-v2* gets higher MIoU than most of the methods (Enet, SegNet, FC-DenseNet56) and similar results than other methods that use many more parameters, like FC-DenseNet103. Considering only the fastest approaches, *MiniNet-v2-cpu* gets better performance (+8.6% MioU) than ENet and earlier MiniNet (+18.6% MioU) and presents much lower memory requirements. It also beats some relevant architectures, like FC-DenseNet56, in all the metrics. *MiniNet-v2-cpu* gets similar performance than ENet in the Cityscapes dataset, but, interestingly, it gets significantly better performance in the Camvid data. This could mean that our architecture learns better on smaller datasets, a common case in robotics. Note that this dataset is $\times 8$ smaller than the Cityscapes dataset and therefore, it is more sensitive to pre-training: MiniNet-v2 goes from 69% MIoU to 76% in the Camvid data if pretrained on Cityscapes.

B. Binary segmentation

Previous experiments have shown how *MiniNet-v2* gets similar or better results than state-of-the-art models while being more efficient regarding several metrics. However, *MiniNet-v2-cpu* is 10 MIoU points below when evaluated in the multi-class benchmarks from the two previous experiments. Differently from previous experiments performed on datasets with more semantic classes, wow we consider binary text segmentation. We show that for simpler semantic segmentation tasks,

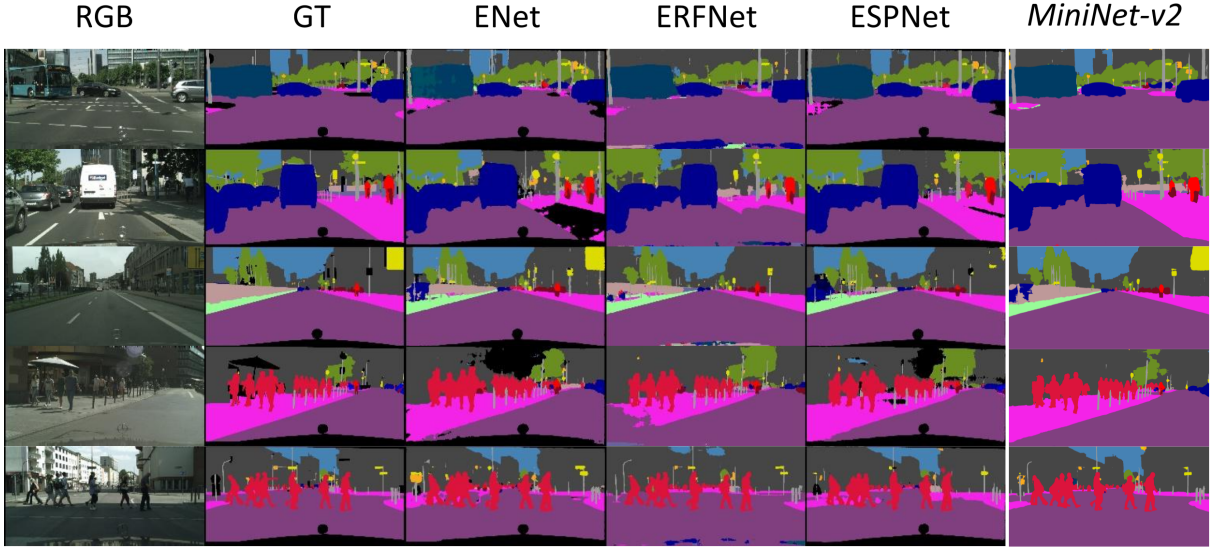| RGB	GT	ENet	ERFNet	ESPNet	*MiniNet-v2*

Fig. 3. Qualitative results on the Cityscapes validation dataset. We show that *MiniNet-v2* achieves really accurate segmentation results, even better than state-of-the-art methods. Different colors mean different classes (except black, which is the ignore/void class). Visual results taken from [20] except ours.

TABLE VII
EVALUATION ON THE **TEST SET** OF THE **CAMVID** DATASET.

Method	GPU ms (type)	Input Resolution	Params (M)	Memory (MB)	GFlops	MIoU w pt	MIoU w/o pt
Mazzini et al. [19]	4(TXp)	480x360	1.4	—	—	68.7	—
ERFNet [25]	28 (TXp)	960x720	2.1	7.95	70.90	—	68.3
ICNet [32]	36 (TX)	960x720	6.7	—	—	—	67.1
FC-DenseNet103 [14]	109 (TXp)	480x360	9.4	35.77	139.43	—	66.9
FC-DenseNet56 [14]	70 (TXp)	480x360	1.5	5.29	61.75	—	58.9
SegNet-basic [3]	60 (TX)	480x360	29.5	112.4	286.03	55.6	—
ENet [22]	6 (TX)	480x360	0.4	1.64	2.87	—	51.3
MiniNet [1]	5 (TXp)	512x256	3.1	12.35	1.06	—	41.3
MiniNet-v2	28 (TXp), 15 (2080ti)	960x720	0.5	2.02	17.01	—	69.0
MiniNet-v2	9 (TXp)	480x360	0.5	2.02	4.22	—	66.1
MiniNet-v2-cpu	5 (TXp)	480x360	0.3	1.06	2.22	—	59.9

w pt: with pretraining on Imagenet. TX: TitanX; TXP: TitanXP

TABLE VIII
BINARY SEGMENTATION RESULTS ON THE COCO-TEXT DATASET.

Method	GPU ms (type)	Input Resolution	Params (M)	Memory (MB)	GFlops	MIoU
DeeplabV3+ [6]	32 (TXp)	512x512	41.1	—	51.44	32.29
ERFNet [24]	20 (TXp)	1024x512	2.1	7.95	53.78	29.27
MiniNet [1]	5 (TXp)	512x256	3.1	12.35	1.06	27.63
MiniNet-v2	20 (TXp), 11 (2080ti)	1024x512	0.5	2.02	12.89	30.78
MiniNet-v2	8 (TXp)	512x256	0.5	2.02	3.22	29.02
MiniNet-v2-cpu	4 (TXp)	512x256	0.3	1.06	1.68	28.82

There is no pretraining run for any of the methods

MiniNet-v2-cpu also gets similar results than top-performing CNNs, dropping only 2 MIoU points from *MiniNet-v2*.

For this experiment, we use the COCO-Text dataset [28] (a subset of machine printed and legible text images). We trained four architectures: MiniNet-v2-cpu, MiniNet-v2, Deeplabv3+ (top-performing generic semantic segmentation approach) and ERFNet (the state-of-the-art for low-power GPUs), using the same training protocol as in previous experiments.

Table VIII shows that our approach reaches comparable performance to state-of-the-art architectures (even including those not focused on efficiency like Deeplab) when training for simpler segmentation tasks. Even *MiniNet-v2-cpu* performs similar to top methods, while running at 250fps on GPU and 30fps on CPU.

C. Efficient keyframe selection example

MiniNet architectures facilitate the use of semantic segmentation in robotic platforms that require fast pre-processing of the current view content. For example, to decide whether to transmit the frame or not to other robots or to select the most representative frame to store from a sub-sequence.

We refer to [1] for a more detailed set of experiments that demonstrate how even the semantic segmentation of our simpler model, *MiniNet*, is enough for real practical applications. There *MiniNet* is used to enhance visual SLAM keyframe selection in a realistic experiment. MiniNet is trained to quickly segment regions in the image likely to have the target scene class (in that example, scene text). Fig. 4 shows an example of the advantages of the key-frame selection using our proposed MiniNet.

VIII. CONCLUSIONS

This work presents our novel architecture for efficient semantic segmentation, *MiniNet-v2*, and its variation *MiniNet-v2-cpu*, improving our earlier work in *MiniNet*. These models pave the way for applications in robotics or related embedded systems that require quick visual scene understanding steps. Our model design is based on the conclusions from the presented study about the most relevant techniques to build efficient CNNs. We analyze the trade-offs provided by several techniques and operations for efficient semantic segmentation. Our results show that our proposed multi-dilation depthwise separable convolutions, the use of an additional convolutional branch instead of skip connections and reducing the output resolution, are key steps to achieve a good trade-off between accuracy and computational requirements. Our proposed architectures have been thoroughly evaluated to demonstrate their benefits. *MiniNet-v2* achieves similar or better results than state-of-the-art models on known public benchmarks for semantic segmentation, while it provides lower memory and computation requirements. *MiniNet-v2-cpu* can be used for

Fig. 4. Key-frame selected by a standard VSLAM algorithm (ORB-SLAM2) and the proposed selection using MiniNet [1]. Note how frames with better illumination (a) and more semantic information (b) are selected by our approach. MiniNet is trained to segment text regions and helps avoiding frames with only partially visible scene text (red bounding boxes).

real-time CPU-only applications, achieving similar results than top-performing CNN architectures when the segmentation task is not very complex, e.g., binary segmentation. We have shown a proof of concept demonstration of the benefits of these models in robotic applications, namely, to allow a robot to quickly analyze the content of a video to decide which frames to share with the rest of the robotic team. The availability of all our code leaves the door open for the development of additional real-time applications using our models.

Acknowledgements

References

[1] Alonso, I., Riazuelo, L., Murillo, A.C.: Enhancing v-slam keyframe selection with an efficient convnet for semantic analysis. In: 2019 International Conference on Robotics and Automation (ICRA), pp. 4717–4723. IEEE (2019) 1, 4, 6, 7, 8

[2] Alvarez, J., Petersson, L.: DecomposeMe: Simplifying convnets for end-to-end learning. arXiv preprint arXiv:1606.05426 (2016) 2

[3] Badrinarayanan, V., Kendall, A., Cipolla, R.: Segnet: A deep convolutional encoder-decoder architecture for image segmentation. IEEE transactions on pattern analysis and machine intelligence 39(12), 2481–2495 (2017) 1, 2, 6, 7

[4] Briot, A., Viswanath, P., Yogamani, S.: Analysis of efficient cnn design techniques for semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 663–672 (2018) 1

[5] Brostow, G.J., Fauqueur, J., Cipolla, R.: Semantic object classes in video: A high-definition ground truth database. Pattern Recognition Letters 30(2), 88–97 (2009) 1, 4

[6] Chen, L.C., Zhu, Y., Papandreou, G., Schroff, F., Adam, H.: Encoder-decoder with atrous separable convolution for semantic image segmentation. In: Proceedings of the European conference on computer vision (ECCV), pp. 801–818 (2018) 1, 2, 3, 7

[7] Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., Schiele, B.: The cityscapes dataset for semantic urban scene understanding. In: computer vision and pattern recognition (2016) 1, 4

[8] Courbariaux, M., Bengio, Y., David, J.P.: Binaryconnect: Training deep neural networks with binary weights during propagations. In: Advances in neural information processing systems, pp. 3123–3131 (2015) 2

[9] Fisher, Y., Vladlen, K.: Multi-scale context aggregation by dilated convolutions. In: International Conference on learning representations (2016) 2, 3

[10] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778 (2016) 2

[11] Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015) 1

[12] Huang, G., Liu, Z., Weinberger, K.Q., van der Maaten, L.: Densely connected convolutional networks. In: Computer Vision and Pattern Recognition (2017) 2

[13] Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y.: Quantized neural networks: Training neural networks with low precision weights and activations. The Journal of Machine Learning Research 18(1), 6869–6898 (2017) 2

[14] Jégou, S., Drozdzal, M., Vazquez, D., Romero, A., Bengio, Y.: The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition workshops, pp. 11–19 (2017) 2, 3, 7

[15] Li, X., Zhou, Y., Pan, Z., Feng, J.: Partial order pruning: for best speed/accuracy trade-off in neural architecture search. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 9145–9153 (2019) 6

[16] Liu, C., Chen, L.C., Schroff, F., Adam, H., Hua, W., Yuille, A.L., Fei-Fei, L.: Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 82–92 (2019) 2

[17] Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 3431–3440 (2015) 2

[18] Mazzini, D., Buzzelli, M., Pauy, D.P., Schettini, R.: A cnn architecture for efficient semantic segmentation of street scenes. In: 2018 IEEE 8th International Conference on Consumer Electronics-Berlin (ICCE-Berlin), pp. 1–6. IEEE (2018) 2, 6

[19] Mazzini, D., Schettini, R.: Spatial sampling network for fast scene understanding. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 0–0 (2019) 2, 6, 7

[20] Mehta, S., Rastegari, M., Caspi, A., Shapiro, L., Hajishirzi, H.: ESPNet: Efficient spatial pyramid of dilated convolutions for semantic segmentation. In: Proceedings of the european conference on computer vision (ECCV), pp. 552–568 (2018) 6, 7

[21] Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J.: Pruning convolutional neural networks for resource efficient transfer learning. arXiv preprint arXiv:1611.06440 (2016) 2

[22] Paszke, A., Chaurasia, A., Kim, S., Culurciello, E.: Enet: A deep neural network architecture for real-time semantic segmentation. arXiv preprint arXiv:1606.02147 (2016) 1, 2, 3, 6, 7

[23] Ravi, S.: Projectionnet: Learning efficient on-device deep networks using neural projections. arXiv preprint arXiv:1708.00630 (2017) 1

[24] Romera, E., Alvarez, J.M., Bergasa, L.M., Arroyo, R.: Erfnet: Efficient residual factorized convnet for real-time semantic segmentation. IEEE Transactions on Intelligent Transportation Systems 19(1), 263–272 (2017) 1, 2, 3, 5, 6, 7

[25] Romera, E., Bergasa, L.M., Alvarez, J.M., Trivedi, M.: Train here, deploy there: Robust segmentation in unseen domains. In: 2018 IEEE Intelligent Vehicles Symposium (IV), pp. 1828–1833. IEEE (2018) 7

[26] Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: International Medical image computing and computer-assisted intervention (2015) 3

[27] Sifre, L., Mallat, S.: Rigid-motion scattering for image classification. Ph.D. thesis, Citeseer (2014) 2

[28] Veit, A., Matera, T., Neumann, L., Matas, J., Belongie, S.: Coco-text: Dataset and benchmark for text detection and recognition in natural images. arXiv preprint arXiv:1601.07140 (2016) 4, 7

[29] Wu, T., Tang, S., Zhang, R., Zhang, Y.: Cgnet: A light-weight context guided network for semantic segmentation. arXiv preprint arXiv:1811.08201 (2018) 6

[30] Xiang, W., Mao, H., Athitsos, V.: Thundernet: A turbo unified network for real-time semantic segmentation. In: 2019 IEEE Winter Applications of Computer Vision (WACV), pp. 1789–1796. IEEE (2019) 2, 6

[31] Yu, C., Wang, J., Peng, C., Gao, C., Yu, G., Sang, N.: Bisenet: Bilateral segmentation network for real-time semantic segmentation. In: Proceedings of the European conference on computer vision (ECCV), pp. 325–341 (2018) 3

[32] Zhao, H., Qi, X., Shen, X., Shi, J., Jia, J.: ICNet for real-time semantic segmentation on high-resolution images. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 405–420 (2018) 1, 2, 3, 6, 7

[33] Zhao, H., Shi, J., Qi, X., Wang, X., Jia, J.: Pyramid scene parsing network. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2881–2890 (2017) 1, 2