Spatial Sampling Network for Fast Scene Understanding

Davide Mazzini Raimondo Schettini
University of Milano-Bicocca
{davide.mazzini, raimondo.schettini}@unimib.it

Abstract

We propose a network architecture to perform efficient scene understanding. This work presents three main novelties: the first is an Improved Guided Upsampling Module that can replace in toto *the decoder part in common semantic segmentation networks. Our second contribution is the introduction of a new module based on spatial sampling to perform Instance Segmentation. It provides a very fast instance segmentation, needing only a thresholding as post-processing step at inference time. Finally, we propose a novel efficient network design that includes the new modules and test it against different datasets for outdoor scene understanding. To our knowledge, our network is one of the most efficient architectures for scene understanding published to date, furthermore being 8.6% more accurate than the fastest competitor on semantic segmentation and almost five times faster than the most efficient network for instance segmentation.*

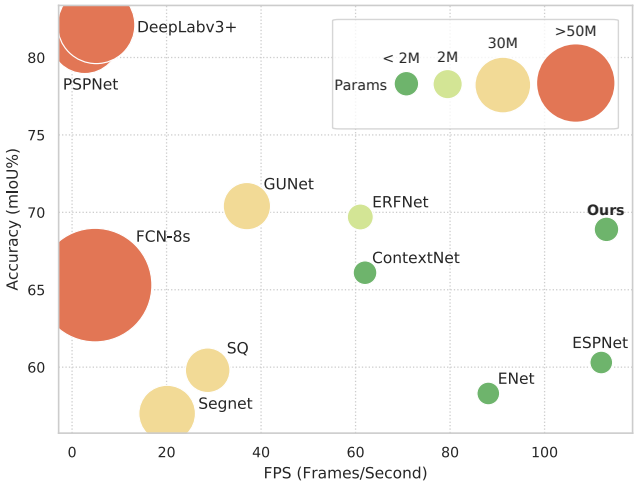

Figure 1. Speed vs mIoU computed on Cityscapes test set for semantic segmentation architectures. Points are proportionally sized with respect to the number of parameters of the model.

1. Introduction

Most of the current architectures that perform semantic segmentation rely on an encoder-decoder architecture. This is the simplest and most effective way to design the model in order to increase the receptive field of the network at the same time keeping the computational cost feasible. To mitigate the effect of information loss caused by downsampling, state-of-the-art architectures make use of additional ways to increase the receptive field, e.g. dilated convolutions [63], ASPP [11], PSP [65]. However the use of dilated convolutions results in computational heavy architectures and, in most cases, the best trade-off consists of a mix of downsampling and dilation. As a consequence, common network architectures, employ upsampling operators to output a semantic map with the same resolution as the input.

Our **first contribution** consists of a new module named Improved Guided Upsampling Module i.e. iGUM. It replaces traditional upsampling operators like bilinear and nearest neighbor and can be plugged into any existing architecture and trained end-to-end within the network. With a low additional computational cost, it helps to improve the prediction along object boundaries when upsampling output probability maps of semantic segmentation networks. We discovered that it can replace the entire decoder part of efficient semantic segmentation networks, obtaining a consistent speed-up and even an improvement of performance in most cases.

As our **second contribution** we designed a novel module that model Semantic Instance Segmentation as a diffusion process through a differentiable sampling operator. The resulting layer has three main advantages: It is computationally lightweight allowing for very fast inference at test time. It can be trained end-to-end with the whole network and requires only a thresholding as post-processing step.

Our **third contribution** consists of a novel lightweight neural architecture to perform scene understanding in applications where speed is a mandatory requirement. We run experiments to assess different aspects of our modules and our complete architecture on three different datasets: Camvid [7], PASCAL VOC 2012 [20] and Cityscapes [15]. Our network is able to achieve 68.9% of mIoU on the popular Cityscapes dataset at 113 FPS on a Titan Xp GPU being

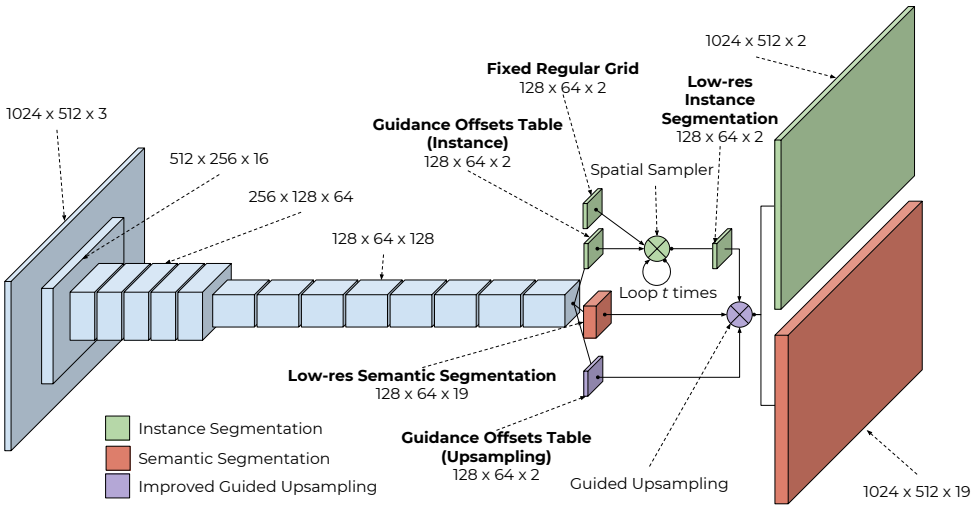

Figure 2. Our network design. The decoder is composed of three parts: the semantic segmentation part, the Instance Segmentation module and the Improved Guided Upsampling Module

8.6% more accurate than the fastest published network up to date.

2. Related Works

Efficiency-oriented architectures SegNet[4], one of the first efficiency-oriented architectures together with [58], introduced an efficient way to exploit high-resolution information by saving max-pooling indices from the encoder and using them during upsampling. ENet authors [43] designed the first highly-efficient architecture by making use of clever design patterns and state-of-the-art building blocks: residual blocks [28], early downsampling, and 1D factorized convolutions [56]. We think that the use of factorized convolutions is the main reason for the success of this architecture. ERFNet [48] authors implemented an architecture with a very similar overall structure to ENet but with a residual module named Non-Bottleneck-1D module. The overall architecture raises the performance of ENet of a great margin. We will build our fast architecture on top of ERFNet. ESPNet [40] is a very recent, efficient architecture that makes use of a novel module named ESP module. It achieves a very good tradeoff in terms of accuracy and network speed.

Instance Segmentation architectures Following [19] we categorize instance segmentation architectures in four different classes: Proposal-based, Recurrent methods, Energy-based and Clustering. [24, 13] are examples of proposal-based approaches. They employ MCG [2] as a class-agnostic predictor of object proposals and they subsequently make use of a classification step to produce instances. A good number of recent works rely on the joint use of an object detector and a semantic segmentation network [17, 18, 3, 25]. As a matter of fact, recent state-of-the-art instance segmentation algorithms [36, 27] are based on enhanced versions of the Faster R-CNN object detector [22] but, from the point-of-view of this work, architectures like PANet [36] or Mask-RCNN [27], are still computationally too heavy to be employed on edge devices for real-time applications.

Other works like [54] fall into the category of recurrent methods, i.e. they adopt recurrent networks to generate instances in a sequential way. [49] uses LSTMs to output binary segmentation maps for each instance. [47] enhance the approach of [49] by adding an object detector network to refine the output. [5, 32] uses alternative ways to detect instances. [5] exploit the watershed transform. [32] uses an approach based on CRF together with a customized MultiCut algorithm. The last category of methods involves the transformation of the input image into a representation that is afterward clustered into a set of discrete instances [50, 64]. [19] introduces a novel loss function that induces an embedding space representing separate instances. As a second step, it employs a clustering algorithm to extract the segmented instances. Instead [59, 30] trains a network to predict a dense vector field where each vector points towards the instance center. The fastest architectures for instance segmentation up to date belong to this category, see also Table 7. However, accurate clustering algorithms are quite heavy to run on a large set of points (pixels) with high-resolution images. This makes impractical to use this category of algorithms in real settings with real-time processing pipelines.

Our network architecture performs instance segmentation in a similar way to the clustering methods. However, it makes the use of a clustering algorithm unnecessary by exploiting a very simple and efficient iterative sampling algorithm. We will explain in details our method for instance

segmentation and outline the differences with [59, 30] in Section 4.

3. Guided Upsampling Module

Most state-of-the-art semantic segmentation networks are based on encoder-decoder architectures [12, 65, 11, 8, 62, 38, 43, 48, 40]. Since the task involves a dense per-pixel prediction, usually a probability vector over the classes distribution is predicted for each pixel. This is arguably inefficient both computationally and from a memory occupation point-of-view. A more efficient output representation would involve a non-uniform density grid which follows the objects density distribution in the scene. A step towards this direction has been presented in [39] with the introduction of the Guided Upsampling Module (GUM). GUM is an upsampling operator built to efficiently handle segmentation maps and improve them along objects' boundaries. Classical upsampling operators (e.g. nearest neighbor or bilinear) make use of a regular grid to sample from the low-resolution image. GUM introduces a warping grid named *Guidance Offset Table* to correct the prediction map along object boundaries. The Guidance Offset Table is predicted by a neural network branch named *Guidance Module*. The whole module can be plugged-in in any existing architecture and trained end-to-end. Given V_i the output feature map and U_{nm} the input feature map, GUM over the nearest neighbor operator can be defined as follows:

$$V_i = \sum_{n}^{H} \sum_{m}^{W} U_{nm} (\delta(\lfloor x_i^s + p_i + 0.5 \rfloor - m) \\ \delta(\lfloor y_i^s + q_i + 0.5 \rfloor - n)) \quad (1)$$

where x_i^s and y_i^s are the spatial sampling coordinates. $\lfloor x_i^s + 0.5 \rfloor$ rounds coordinates to the nearest integer location and δ is a Kronecker function. p_i and q_i are what makes GUM different from nearest-neighbor: two offsets that shifts the sampling coordinates of each grid element in x and y dimensions respectively. They are the output of a function ϕ_i of i, the Guidance Module, defined as: $\phi_i = (p_i, q_i)$. For the definition of GUM over bilinear sampling refer to [39]. The resulting operator is differentiable with respect to U, p_i and q_i. In [39], the GUM module is applied to the network output probability map, even though it could be employed anywhere within the architecture.

3.1. Improved Guided Upsampling Module

Let $U_{cnm} \in \mathbb{R}^{C \times N \times M}$ be an output probability map to be upsampled, where C represents the number of classes, N and M are the spatial dimensions. The GUM module needs a *Guidance Offsets Table* of the same spatial dimensionality as the output features map. Let the upsampling factor be f. To produce a feature map $V_{cnm} \in \mathbb{R}^{C \times fN \times fM}$ the GUM needs to predict a guidance offsets table of cardinality $2 \times fN \times fM$. This is certainly more efficient than predicting the full resolution probability map, especially if C is large because, regardless of the cardinality of classes, the network has to predict only a 2-dimensional vector (offset) for each pixel in the output map. However, a lightweight decoder structure is still needed. Our desiderata are to reduce even more the computational burden and to completely remove the decoder part of the network.

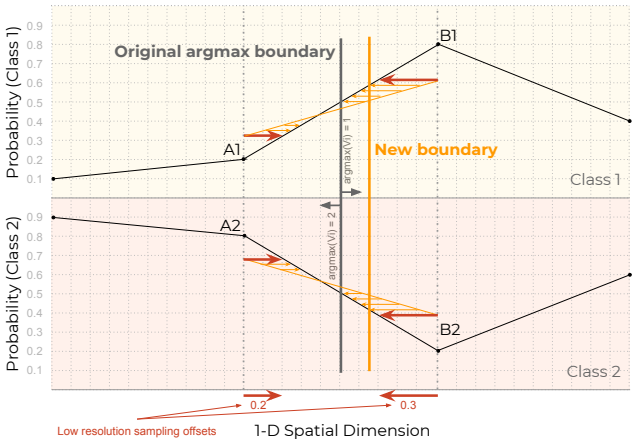

Figure 3. A toy example: upsampling a 1D probability map with two pixels (A and B on the horizontal axis) and two classes (1 and 2 on the vertical axis). The argmax boundary can be moved in two ways: with the orange arrows i.e. GUM [39], or with the red arrows i.e. Improved GUM. The latter formulation requires only two sampling offsets.

A geometric view on GUM Figure 3 depicts a toy example on the use of GUM to upsample an output probability map. The spatial dimension is visualized on the x-axis. In this example, only one dimension is represented although in the real problem there are two spatial dimensions. On the y-axis is represented the probability over the two classes. The two dashed vertical lines indicate two points in the low-resolution probability map. Pairs of black points lying on these lines, i.e ($A1$, $A2$) and ($B1$, $B2$) represent the predicted probability vector over the two classes (1 and 2). The continuous black lines represent the sub-pixel values of the probability distribution obtained by linear interpolation. Notice that, this visualization in a continuous space, allow us to abstract from the upsampling factor. A gray vertical line is depicted on the *argmax* boundary. i.e. the point where the probability distribution is equal over the two classes and where the *argmax* value changes. By looking at Figure 3 it is clear that the position of the original boundary is dependent on the values of the four points $A1$, $A2$, $B1$, $B2$. Thus, the boundary subpixel position is tied to the two adjacent probability vectors. The idea of GUM is represented in Figure 3 by the multiple thin parallel orange arrows. They are spatial sampling vectors predicted by the network at high resolution. The probability value is sampled, i.e. copied, from the head to the tail of each ar-

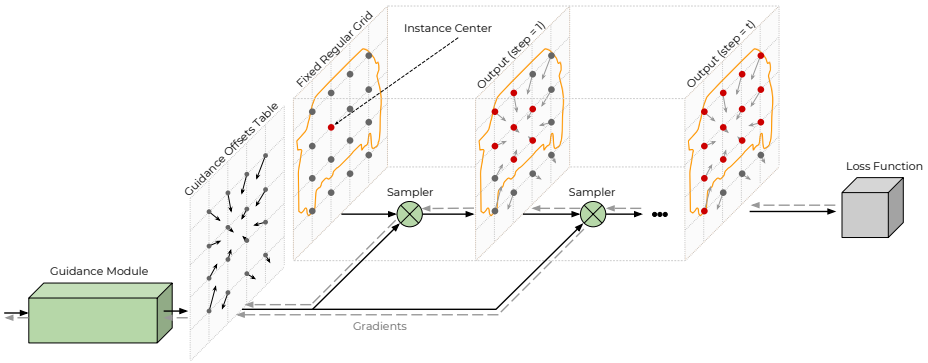

Figure 4. Instance Segmentation Module: a Guidance Offsets Table is used to iteratively sample values from the instance center and spread it to all the neighboring pixels.

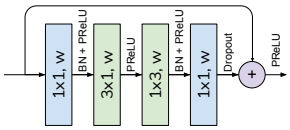

Figure 5. Our proposed Lightweight Non-Bottleneck Module. w represents the number of input channels. $d_1 \times d_2$, f represents the kernel sizes (d_1, d_2) and the numbere of output channels f.

row. With this sampling operation, it is possible to move the *argmax* boundary as shown in Figure 3. Notice that the orange parallel arrows are the same for the two classes: i.e. the boundary can be moved by predicting the same arrows for every class.

Improved GUM intuition Our intuition is that the argmax boundary can be moved by predicting only the low-resolution sampling offsets, i.e. the red arrows at the bottom of Figure 3. (The other bold red arrows are the same arrows moved along the vertical axis). By looking at the parallel thin arrows in between, we observe that they can be obtained by interpolating the bold arrows. We can move the *argmax* boundary by predicting only one additional value low-resolution pixel. Intuitively we can extend this concept to two dimensions by predicting a 2D spatial offsets vector for each spatial location and obtain the other sampling offsets by bilinear interpolation. To summarize, a network with the Improved Guided Upsampling Module, results in a very simple structure. It is composed by an encoder with two branches: the first predicts the output probability map and the second predicts a low-resolution *Guidance Offsets Table*. Both have spatial dimension $N \times M$. The Guidance Offsets Table is then upsampled to the target resolution $fN \times fM$ depending on the upsampling factor f. Finally the high resolution *Guidance Offsets Table* is given as input to the GUM module as in [39]. The resulting module can be plugged into any common CNN architecture and trained end-to-end with the whole network.

Figure 6. Top: visualization of the Guidance Offsets Table of the Instance Module. Bottom: visualization obtained by coloring with a different color each unique value in the output map.

4. Instance Segmentation Module

The most efficient architectures for instance segmentation e.g. [30, 41] (see Section 2) are trained to produce a dense output map with a particular form of embedding for each pixel. To extract every single instance, the embedding needs to be post-processed by a clustering algorithm. Usually, the time needed to run the clustering algorithm on the raw output is not considered when evaluating the speed of state-of-the-art methods. As a matter of fact, running a clustering algorithm on a high-resolution output is computationally intensive, making these approaches inefficient in real-world scenarios. Consider for example the mean-shift algorithm [14] employed by [19]. It has a complexity of $O(Tn^2)$ where T is the number of iterations and n is the number of points in the data set. In case of Cityscapes dataset where the resolution of the images is 1024×2048 the number of points is 2 Million and the number of operations can easily scale up to 20 G-FLOPs, which is roughly the amount of operations needed to perform a forward pass on a VGG-19 architecture [51] (one of the heaviest architectures to date [6]).

Instance Segmentation by Sampling We propose an instance segmentation module that associates to each pixel the instance centroid, similarly to [30, 41]. We train our

network with an L^2 regression loss applied directly to the network output:

$$\mathcal{L}_{\text{Instance}} = \frac{1}{N} \sum_{n \in N} \|i_n - \hat{i}_n\|_2 \quad (2)$$

where $i \in \mathbb{R}^2$ is the instance center in pixel coordinates. $\hat{i}_n \in \mathbb{R}^2$ is the vector predicted by the network. The loss is averaged over every labeled pixel $n \in N$ in a minibatch. The peculiar trait of our method lies in the architecture. Figure 4 shows how the Instance Segmentation module works: a *Guidance Module* predicts a *Guidance Offsets Table* of vectors (bearing the terminology from [39]). The usual way to train instance segmentation networks [30] is to directly apply the loss function in Eq.2 to the Guidance Offsets Table. In our architecture instead, a differentiable sampler, like the one used in [39] or [29], samples 2D points from a fixed regular grid using the Guidance Offsets Table. The fixed regular grid codifies the 2D coordinates of the exact location of each pixel. This sampling process is applied t times *sharing the same* Guidance Offsets Table (see Figure 4). Thus, by consecutive sampling steps, the 2D coordinate values of instance centers are spread all over the area covered by the instance. Finally, the loss function in Eq. 2 is computed on the sampled output. Figure 4 shows the gradient flow: notice that, it flows backward through every sampler step but not to the fixed regular grid. The only way to decrease the loss function is to produce a Guidance Offsets Table with vectors that point towards the instance center.

There is a main advantage of this approach over the classical method of training directly the vector field: in an ideal setting, if the network predicts a perfect output, all the vectors associated with a particular instance will points precisely to the instance center. This doesn't happen in practical cases, thus, the need of a clustering algorithm. Here emerges the major advantage of our approach: by sampling multiple times, a diffusion process is generated. As a consequence, the value of the instance center propagates through all the pixels associated with that particular instance. If a far vector points to an imprecise location towards the instance center, by successive sampling steps, the center value propagates to increasing areas eventually covering all the instance area. Top visualization in Figure 6 shows the Guidance Offsets Table overimposed on the input image. Note that the vectors' magnitude decrease drastically near the instance center. The visualization at the bottom of Figure 6 is obtained by summing for each pixel location the 2D centroids' predictions coordinates. A single value per location is obtained (it can be interpreted as a unique instance identifier). In Figure 6 a different color is assigned to *each* unique value. The only post-processing step applied to the final output is a thresholding over the instance area. Very small false positives are eliminated by this step, they can be noticed by zooming Figure 6 (bottom) along object boundaries.

Two Practical Tips First, to simplify the learning process, the values of the Guidance Offsets Tables both for the Improved GUM and for the Instance Module are limited to the interval $[-1, 1]$ by mean of a $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ function. Second, the GUM operator used to upsample the instance output is based on Bilinear sampling at training time, and Nearest-neighbor at test time. Nearest-neighbor is the right choice to upsample unique id values but it is not directly differentiable w.r.t the Guidance Offsets Table.

5. Network Architecture

Our architecture for semantic segmentation consists of a lightweight encoder and an iGUM layer as decoder. For the task of instance segmentation we added the Instance Segmentation Module before the guided upsampling (refer to Figure 2 for details). The iGUM is described in details in Section 3 whereas the encoder has a very simple structure inspired by [48]. However, the main building block is a novel Lightweight Non-bottleneck Module described in this section.

Encoder-only architecture In encoder-decoder architectures, the decoder plays a refinement role where features are subsequently upsampled to match the input size and to finally output the dense per-pixel prediction. By employing our iGUM module we are able to completely remove the decoder part: as shown in Table 1 this a major factor for a highly-efficient semantic segmentation network.

Lightweight Non-Bottleneck-1D Module The Non-Bottleneck-1D module has been proposed by Romera et al. as the main building block of ERFNet [48]. It is a residual block composed of 1D convolutions also known as *Asymmetric convolutions* [37]. Numerous works investigated decomposed filters from a theoretical point of view, e.g. [52, 1]) and in practical settings , e.g. [57, 55, 43, 37]. The idea is that each convolutional layer can be decomposed in 1D filters that can additionally include a non-linearity in between. The decomposed layers have intrinsically lower computational cost than the corresponding full rank counterparts. Our Lightweight Non-Bottleneck Module consists in a very simple design: it is a non-bottleneck residual block with two asymmetric kernels (3×1) and (1×3) preceded and followed by 1×1 channelwise convolutions. The module design is depicted in Figure 5. We discovered this module to be particularly effective as part of our architecture. We motivate every design choice experimentally within the ablation study in Section 6.1.

Early downsampling Following the last works on efficient models [43, 63, 48, 46] our architecture employs an early downsampling strategy to speed-up inference time. In our network, the first layers act as early feature extractors whereas the most intense operations are carried out by the inner network modules to favor a more complex representa-

tion in late activations. Our downsampling block, inspired by [48, 43], performs downsampling by concatenating the parallel outputs of a single 3×3 convolution with stride 2 and a Max-Pooling module.

6. Experiments

We built our architecture based on different types of experiments to verify each design hypothesis. After presenting the implementation details for the sake of reproducibility, we introduce different experiments concerning the Improved Guided Upsampling Module and in-depth ablation studies on our architecture. Finally, we compare our network with state-of-the-art efficient architectures on different datasets for semantic and instance segmentation.

Evaluation metrics *mean of class-wise Intersection over Union (mIoU)* is used to measure semantic segmentation quality on all datasets. It is computed as the classwise mean of the intersection over union measure. On Camvid dataset also the class average and the global average are computed being the mean of the accuracy on all classes and the global pixel accuracy respectively. *Frame Per Second (FPS)* is used as speed measure, defined as the inverse of time needed for our network to perform a single forward pass. FPS have been computed on a single Titan Xp GPU whether not differently specified. Following [43], we removed all Batch-Normalizations at test time merging them with close convolutions.

Training Recipes All experiments have been conducted within the Pytorch framework [44] v1.0. For training, following [48] we use the Adam optimizer [31] in an initial learning rate of $5e^{-4}$ and weight decay of $1e^{-4}$. The learning rate is scheduled by multiplying the initial learning rate by $(1 - \frac{epoc}{maxEpochs})^{0.9}$. All models are trained for 150 epochs with a mini-batch size of 8. We also include Dropout [53] in all our Lightweight Non-Bottleneck modules as regularizers. Following [48] we set the dropout rate of the first five modules to 0.03 and all the others to 0.3.

6.1. Results on Cityscapes

Cityscapes [15] is a large scale dataset for semantic urban scene understanding. It consists of 5000 finely annotated high-resolution images with pixel-level fine annotations. Images have been collected in 50 different cities around Europe, with high variability of weather conditions and in different seasons. We used the standard split suggested by the authors which consist in 2975, 500, and 1525 images for train, validation, and test sets respectively. Annotations include 19 classes used to train and evaluate models. Following a common practice for efficient oriented architectures [48, 43, 40], images have been subsampled by a factor 2 for every experiment reported on Cityscapes dataset.

Model	Original Decoder		iGUM Module	
	mIoU	FPS	mIoU	FPS
GUNet [39]	**64.8**	40	64.4	**48**
ERFNet [48]	60.7	59	**63.7**	**81**
ENet [43]	47.3	87	**55.7**	**137**
ESPNet [40]	48.2	172	**52.9**	**206**

Table 1. mIoU and FPS on Cityscapes val set by replacing the original decoder with our Improved Guidance Upsampling Module in GUNet and three other efficient architectures for semantic segmentation.

Encoder	Pretraining	Decoder	mIoU%	FPS
ERFNet (baseline)	✓	ERFNet	**72.3**	60.3
ERFNet	✓	improved GUM	71.6	84.4
Ours	✓	improved GUM	69.3	**113.1**

Table 2. Ablation study on Cityscapes val dataset for our encoder and decoder. We adopted ERFNet [48] as baseline and replaced encoder and decoder in two steps. mIoU slighly decreases w.r.t. the baseline but the inference speed almost doubles. Encoders have been pre-trained on Imagenet.

Improved GUM on efficient architectures We investigated the use of iGUM module on GUNet [39] as a replacement for the original GUM module. Furthermore, we tested our iGUM within three state-of-the-art efficient architectures for semantic segmentation. We trained the networks with their original decoder and compared them with modified versions where the decoder is replaced by the improved GUM module. Table 1 shows the results of these experiments: the first line shows the performance of GUM architecture with the original GUM and with our iGUM module. The results support the theoretical study exposed in Section 3.1: by replacing GUM with iGUM in GUNet architecture [39], the loss in accuracy is negligible. On the other hand, the model exhibits a visible benefit in speed. Both speed and mIoU improve over the baselines on all the other architectures by a large margin. The baseline does not correspond exactly with results reported by the papers because we trained these three networks with the same settings exposed in Section 6 which may differ from those used by the authors to train their own architecture. Moreover, they have not been pretrained and the speed has been evaluated by removing the Batch-Normalization layer. For the sake of these experiments, we are only interested in the relative performances.

Ablation Studies We designed our network starting from ERFNet [48] as baseline. First, we replaced the decoder part, which in the original ERFNet is composed of three deconvolutions and four Non-Bottleneck-1D Modules. The introduction of the Improved GUM cause a negligible performance decrease, i.e. from 72.3 to 71.6 but improves speed by 24 frames per seconds. As a second step, we replaced also the encoder obtaining a completely new architecture. Again, a small decrease in performance but a

Module Name	Conv 1x1	No Bias	PReLU	Moved 1x1	Pretrain	mIoU%	FPS
Non-bt-1D (baseline)						63.2	84.4
	✓					62.4	95.3
		✓				63.6	96.0
			✓			65.6	82.3
	✓	✓	✓			63.1	**113.1**
Lightweight Non-bt-1D	✓	✓	✓	✓		64.1	**113.1**
Lightweight Non-bt-1D	✓	✓	✓	✓	✓	**69.3**	**113.1**

Table 3. Ablation study for our Lightweight Non-Bottleneck 1D Module on Cityscapes val set.

Method	IoU class	iIoU class	IoU category	iIoU category	FPS
Mapillary [8]	82.0	65.9	91.2	81.7	n/a
PSPNet [65]	81.2	59.6	91.2	79.2	2.7
FCN-8s [38]	65.3	41.7	85.7	70.1	4.9
DeepLabv3+ [12]	**82.1**	**62.4**	**92.0**	**81.9**	5.1
SegNet [4]	57.0	32.0	79.1	61.9	20.1
SQ [58]	59.8	32.3	84.3	66.0	28.7
GUNet [39]	70.4	40.8	86.8	69.1	37.0
ERFNet [48]	69.7	44.1	87.3	72.7	61.0
ContextNet [45]	66.1	36.8	82.8	64.3	62.0
ENet [43]	58.3	34.4	80.4	64.0	88.1
ESPNet [40]	60.3	31.8	82.2	63.1	112.0
Ours	68.9	39.0	85.9	66.5	**113.1**

Table 4. Comparison with representative architectures for semantic segmentation on Cityscapes test set. mIoU evaluated by Cityscapes evaluation server. FPS reported from original paper if authors used a TitanX (Pascal) GPU, otherwise FPS computed on our GPU.

large speedup compared to the baseline. Table 2 shows the results of the two ablation experiments.

In Table 3 we show the results of the ablation experiments for every choice made to obtain the Lightweight Non-Bottleneck-1D module. The baseline is an architecture composed by Non-bt-1D modules from [48]. A Non-bt-1D module is composed by four factorized 1D kernels. The first two have dilation term 1 whereas the last two have different dilation terms, i.e from 1 to 16, depending on the position within the architecture. We replaced the first two factorized convolutions with 1x1 convolutions. This speeded up the architecture by 15 FPS with a negligible mIoU decrease. Then we removed the biases from every convolution obtaining an mIoU of 63.6% and a slight increase in speed. Inspired by [43] we replaced ReLUs with PReLUs. This increased mIoU by 2% without any loss in speed. By applying these modifications together we got a mIoU of 63% and a very fast encoder, i.e. 113.1 FPS. Finally we moved one 1x1 convolution to the end of the residual module like shown in Figure 5 obtaining an mIoU of 64.1%. Last row shows the effect of pretraining on Imagenet.

Instance Segmentation With the first cluster of experiments we want to determine the best loss function to train our architecture for the instance segmentation task. Table 5 shows results on the Cityscapes validation set in terms of

Loss Function	mIoU	AP	AP50%
L2	65.7	**10.7**	**20.2**
L1	65.3	9.8	18.0
Smooth L1	**66.7**	10.2	18.3

Table 5. Different loss functions to train our instance segmentation module. Results on Cityscapes val set.

Iterations	AP	AP50%	ms	FPS
40	10.7	20.2	10.0	105.1
30	**10.7**	**20.2**	9.0	106.4
20	10.5	20.0	8.9	107.4
15	10.3	19.3	8.8	109.3
10	9.7	17.7	8.8	109.7
7	8.5	16.0	8.7	109.7
5	7.2	14.3	8.7	110.1
3	4.0	8.6	**8.7**	**110.2**

Table 6. Experiments to assess the impact on performance and speed w.r.t the number of iterations of sampling in the Instance Segmentation Module. Tested on Cityscapes val set.

name	AP	AP 50%	AP 100m	AP 50m	FPS
Deep Contours [61]	2.3	3.7	3.9	4.9	5.0
R-CNN + MCG convex hull [15]	4.6	12.9	7.7	10.3	0.1
FCN+Depth [60]	8.9	21.1	15.3	16.7	n/a
Joint Graph Decomposition [33]	9.8	23.2	16.8	20.3	n/a
Boundary-aware [26]	17.4	36.7	29.3	34	n/a
Discriminative Loss Function [19]	17.5	35.9	27.8	31	n/a
Dist. Watershed Transform [5]	19.4	35.3	31.4	36.8	n/a
Fast Scene Understanding [41]	21.0	38.6	34.8	38.7	21.3
Multitask Learning [30]	21.6	39	35	37	n/a
Mask R-CNN [27]	26.2	49.9	37.6	40.1	n/a
PANet [36]	**31.8**	**57.1**	**44.2**	**46**	n/a
Ours	9.2	16.8	16.4	21.4	**106.4**

Table 7. Comparison with State of the art methods on Cityscapes test set from the cityscapes leaderboard.

mIoU on semantic segmentation and AP on instance segmentation. Surprisingly the variance is low between different loss functions. We decided to keep L^2 loss for the next experiments. With our second cluster of experiments we want to assess how many sampling iterations t (see Section 4) are needed for convergence and how much they affect performance and speed. Table 6 shows the results for these experiments. The FPS are not very affected, due to the efficiency of the sampling module, the performances start to degrade significantly from 15 iterations. We decided to keep 30 iterations for the next experiment. We tested the network against the Cityscapes test set: results are shown in Table 7. Our method achieve the 9.2% of AP which is far from the state-of-the-art accuracy-oriented methods like PANet [36] but it exhibit a remarkably fast inference time.

6.2. Results on Camvid

We tested our architecture on the Camvid[7] dataset. It is composed of 367 training and 233 testing images of urban outdoor environments. It has been tagged in eleven semantic classes of which one is not evaluated and thus not used for training. The original frame resolution is 970×720. Fol-

Method	Pretraining	Class avg.	mIoU	Global avg.
Segnet [4]	✓	65.2	55.6	88.5
ENet [43]		68.3	51.3	n/a
ESPNet [40]		68.3	55.6	n/a
ERFNet [48]		65.8	53.1	86.3
ERFNet [48]	✓	72.5	62.7	89.4
FCN-8s [38]	✓	n/a	57.0	88.0
Dilation8 [63]	✓	n/a	65.3	79.0
DeepLab [9]	✓	n/a	61.6	n/a
Ours	✓	**76.9**	**68.7**	**91.9**

Table 8. Results on Camvid test set ordered by increasing mIoU. Our model outperform every other efficient architecture by a large margin. It even yelds better results with respect to some accuracy-oriented architectures.

Method	mIoU%
SegNet [4]	59.10
LRR [21]	79.30
Dilation-8 [63]	75.30
FCN-8s [38]	67.20
ESPNet [40]	63.01
DeepLab [9]	79.70
RefineNet [34]	82.40
PSPNet [65]	85.40
DeepLabv3+ [12]	**87.80**
Ours	63.54

Table 9. Results on PASCAL VOC 2012 test set. We reported some popular methods and the state-of-the-art on Pascal VOC.

lowing [43, 4, 40], for fair comparison, we downsampled the images to 480×360 pixels before training. This dataset represents an interesting benchmark to test the behaviour of our method with low-cardinality datasets. In Table 8 we compare the performance of our architecture with existing state-of-the-art efficient architectures. We also include three computationally-heavy architectures. Our architecture yields very good results with respect to other efficiency-oriented architectures and even outperforms some accuracy-oriented methods.

6.3. Results on PASCAL VOC 2012

We tested our network architecture against the popular PASCAL VOC 2012 segmentation dataset [20] which contains 20 object categories and a background class. It is composed of 1464, 1448 and 1456 images for the training, validation and test sets respectively. Following [38, 10, 42, 16] we used additional images to train our network with data annotation of [23] resulting in 10582 1449 and 1456 images for training, validation and testing. Table 9 shows a comparison of our method with state-of-the-art popular architectures. Only ESPNet and our architecture are speed-oriented while all the others focus on accuracy. Notice that, besides being computationally heavy PSPNet and DeepLabV3+ have been pretrained on COCO [35] dataset.

6.4. Speed

We report in Table 10 the speed of our network on a high-end Titan Xp GPU with Pytorch 1.0 and CUDA 10.0. Then we tested our architecture on an edge device: Nvidia Tegra TX 1 (Jetson) with Pytorch 0.4.

NVIDIA TEGRA TX1 (Jetson)											
ms	fps	ms	fps	ms	fps	ms	fps	ms	fps	ms	fps
640×360		1280×720		1920×1080		512×256		1024×512		2048×1024	
109	9.9	471	2.1	1.43	0.7	49	20.5	207	5.1	1228	0.9
NVIDIA TITAN Xp											
640×360		1280×720		1920×1080		512×256		1024×512		2048×1024	
4	235.9	13	79.1	29	34.4	4	242.1	8	113.1	29	34.7

Table 10. Network speed on a high-end GPU and an embedded device.

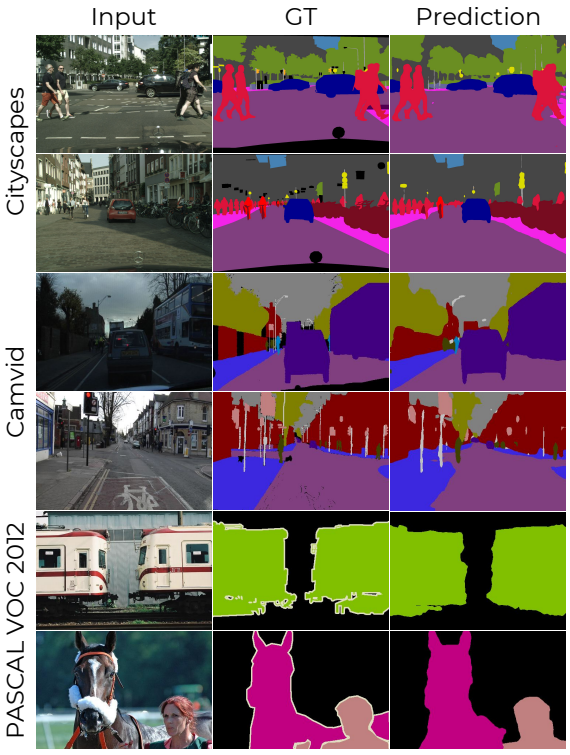

Figure 7. Visual results on images from three datasets: Cityscapes, Camvid and PASCAL VOC 2012

7. Concluding Remarks

We presented a novel architecture for efficient scene understanding which includes a novel module to speed up the decoder part of encoder-decoder architectures and a module for Instance Segmentation based on iterative sampling. We tested our architecture on three different datasets showing that our network is fast and accurate compared to state-of-the-art efficient architectures.

Acknowledgments This work was supported by TEINVEIN, CUP: E96D17000110009 - Call "Accordi per la Ricerca e l'Innovazione", cofunded by POR FESR 2014-2020. We gratefully acknowledge the support of NVIDIA Corporation with the donation of a Titan Xp GPU used for this research.

References

[1] J. Alvarez and L. Petersson. Decomposeme: Simplifying convnets for end-to-end learning. *arXiv preprint arXiv:1606.05426*, 2016. 5

[2] P. Arbeláez, J. Pont-Tuset, J. T. Barron, F. Marques, and J. Malik. Multiscale combinatorial grouping. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 328–335, 2014. 2

[3] A. Arnab and P. H. Torr. Bottom-up instance segmentation using deep higher-order crfs. *arXiv preprint arXiv:1609.02583*, 2016. 2

[4] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017. 2, 7, 8

[5] M. Bai and R. Urtasun. Deep watershed transform for instance segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2858–2866. IEEE, 2017. 2, 7

[6] S. Bianco, R. Cadene, L. Celona, and P. Napoletano. Benchmark analysis of representative deep neural network architectures. *IEEE Access*, 2018. 4

[7] G. J. Brostow, J. Fauqueur, and R. Cipolla. Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*, xx(x):xx–xx, 2008. 1, 7

[8] S. R. Bulò, L. Porzi, and P. Kontschieder. In-place activated batchnorm for memory-optimized training of dnns. *CoRR, abs/1712.02616, December*, 5, 2017. 3, 7

[9] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *CoRR*, abs/1412.7062, 2014. 8

[10] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*, 2014. 8

[11] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. *arXiv preprint arXiv:1802.02611*, 2018. 1, 3

[12] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. *arXiv preprint arXiv:1802.02611*, 2018. 3, 7, 8

[13] Y.-T. Chen, X. Liu, and M.-H. Yang. Multi-instance object segmentation with occlusion handling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3470–3478, 2015. 2

[14] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 24(5):603–619, 2002. 4

[15] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 1, 6, 7

[16] J. Dai, K. He, and J. Sun. Boxsup: Exploiting bounding boxes to supervise convolutional networks for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1635–1643, 2015. 8

[17] J. Dai, K. He, and J. Sun. Convolutional feature masking for joint object and stuff segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3992–4000, 2015. 2

[18] J. Dai, K. He, and J. Sun. Instance-aware semantic segmentation via multi-task network cascades. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3150–3158, 2016. 2

[19] B. De Brabandere, D. Neven, and L. Van Gool. Semantic instance segmentation with a discriminative loss function. *arXiv preprint arXiv:1708.02551*, 2017. 2, 4, 7

[20] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010. 1, 8

[21] G. Ghiasi and C. C. Fowlkes. Laplacian pyramid reconstruction and refinement for semantic segmentation. In *European Conference on Computer Vision*, pages 519–534. Springer, 2016. 8

[22] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015. 2

[23] B. Hariharan, P. Arbeláez, L. Bourdev, S. Maji, and J. Malik. Semantic contours from inverse detectors. 2011. 8

[24] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik. Simultaneous detection and segmentation. In *European Conference on Computer Vision*, pages 297–312. Springer, 2014. 2

[25] Z. Hayder, X. He, and M. Salzmann. Shape-aware instance segmentation. *CoRR*, 1(2):5, 2016. 2

[26] Z. Hayder, X. He, and M. Salzmann. Boundary-aware instance segmentation. In *30Th Ieee Conference On Computer Vision And Pattern Recognition (Cvpr 2017)*, number CONF. Ieee, 2017. 7

[27] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2980–2988. IEEE, 2017. 2, 7

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

[29] M. Jaderberg, K. Simonyan, A. Zisserman, et al. Spatial transformer networks. In *Advances in neural information processing systems*, pages 2017–2025, 2015. 5

[30] A. Kendall, Y. Gal, and R. Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. *arXiv preprint arXiv:1705.07115*, 3, 2017. 2, 3, 4, 5, 7

[31] D. Kinga and J. B. Adam. A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, volume 5, 2015. 6

[32] A. Kirillov, E. Levinkov, B. Andres, B. Savchynskyy, and C. Rother. Instancecut: from edges to instances with multicut. In *CVPR*, volume 3, page 9, 2017. 2

[33] E. Levinkov, J. Uhrig, S. Tang, M. Omran, E. Insafutdinov, A. Kirillov, C. Rother, T. Brox, B. Schiele, and B. Andres. Joint graph decomposition & node labeling: Problem, algorithms, applications. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 7. IEEE, 2017. 7

[34] G. Lin, A. Milan, C. Shen, and I. Reid. Refinenet: Multi-path refinement networks for high-resolution semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1925–1934, 2017. 8

[35] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 8

[36] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8759–8768, 2018. 2, 7

[37] S.-Y. Lo, H.-M. Hang, S.-W. Chan, and J.-J. Lin. Efficient dense modules of asymmetric convolution for real-time semantic segmentation. *arXiv preprint arXiv:1809.06323*, 2018. 5

[38] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015. 3, 7, 8

[39] D. Mazzini. Guided upsampling network for real-time semantic segmentation. In *British Machine Vision Conference (BMVC)*, 2018. 3, 4, 5, 6, 7

[40] S. Mehta, M. Rastegari, A. Caspi, L. Shapiro, and H. Hajishirzi. Espnet: Efficient spatial pyramid of dilated convolutions for semantic segmentation. *arXiv preprint arXiv:1803.06815*, 2018. 2, 3, 6, 7, 8

[41] D. Neven, B. De Brabandere, S. Georgoulis, M. Proesmans, and L. Van Gool. Fast scene understanding for autonomous driving. *arXiv preprint arXiv:1708.02550*, 2017. 4, 7

[42] G. Papandreou, L.-C. Chen, K. P. Murphy, and A. L. Yuille. Weakly-and semi-supervised learning of a deep convolutional network for semantic image segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 1742–1750, 2015. 8

[43] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147*, 2016. 2, 3, 5, 6, 7, 8

[44] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017. 6

[45] R. P. Poudel, U. Bonde, S. Liwicki, and C. Zach. Contextnet: Exploring context and detail for semantic segmentation in real-time. 2018. 7

[46] Z. Qin, Z. Zhang, X. Chen, C. Wang, and Y. Peng. Fd-mobilenet: Improved mobilenet with a fast downsampling strategy. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 1363–1367. IEEE, 2018. 5

[47] M. Ren and R. Zemel. End-to-end instance segmentation and counting with recurrent attention. arxiv 2016. *arXiv preprint arXiv:1605.09410*. 2

[48] E. Romera, J. M. Alvarez, L. M. Bergasa, and R. Arroyo. Erfnet: Efficient residual factorized convnet for real-time semantic segmentation. *IEEE Transactions on Intelligent Transportation Systems*, 19(1):263–272, 2018. 2, 3, 5, 6, 7, 8

[49] B. Romera-Paredes and P. H. S. Torr. Recurrent instance segmentation. In *European Conference on Computer Vision*, pages 312–329. Springer, 2016. 2

[50] N. Silberman, D. Sontag, and R. Fergus. Instance segmentation of indoor scenes using a coverage loss. In *European Conference on Computer Vision*, pages 616–631. Springer, 2014. 2

[51] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 4

[52] A. Sironi, B. Tekin, R. Rigamonti, V. Lepetit, and P. Fua. Learning separable filters. *IEEE transactions on pattern analysis and machine intelligence*, 37(1):94–106, 2015. 5

[53] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014. 6

[54] R. Stewart, M. Andriluka, and A. Y. Ng. End-to-end people detection in crowded scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2325–2333, 2016. 2

[55] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017. 5

[56] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015. 2

[57] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016. 5

[58] M. Treml, J. Arjona-Medina, T. Unterthiner, R. Durgesh, F. Friedmann, P. Schuberth, A. Mayr, M. Heusel, M. Hofmarcher, M. Widrich, et al. Speeding up semantic segmentation for autonomous driving. In *MLITS, NIPS Workshop*, 2016. 2, 7

[59] J. Uhrig, M. Cordts, U. Franke, and T. Brox. Pixel-level encoding and depth layering for instance-level semantic labeling. In *German Conference on Pattern Recognition*, pages 14–25. Springer, 2016. 2, 3

[60] J. Uhrig, M. Cordts, U. Franke, and T. Brox. Pixel-level encoding and depth layering for instance-level semantic labeling. In *German Conference on Pattern Recognition*, pages 14–25. Springer, 2016. 7

[61] J. van den Brand, M. Ochs, and R. Mester. Instance-level segmentation of vehicles by deep contours. In *Asian Conference on Computer Vision*, pages 477–492. Springer, 2016. 7

[62] Z. Wu, C. Shen, and A. v. d. Hengel. Wider or deeper: Revisiting the resnet model for visual recognition. *arXiv preprint arXiv:1611.10080*, 2016. 3

[63] F. Yu, V. Koltun, and T. A. Funkhouser. Dilated residual networks. In *CVPR*, volume 2, page 3, 2017. 1, 5, 8

[64] Z. Zhang, A. G. Schwing, S. Fidler, and R. Urtasun. Monocular object instance segmentation and depth ordering with cnns. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2614–2622, 2015. 2

[65] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 2881–2890, 2017. 1, 3, 7, 8