ESPNet: Efficient Spatial Pyramid of Dilated Convolutions for Semantic Segmentation

Sachin Mehta[1], Mohammad Rastegari[2], Anat Caspi[1], Linda Shapiro[1], and Hannaneh Hajishirzi[1]

[1] University of Washington, Seattle, WA, USA
{sacmehta, hannaneh}@uw.edu, {caspian, shapiro}@cs.washington.edu
[2] Allen Institute for AI and XNOR.AI
mohammadr@allenai.org

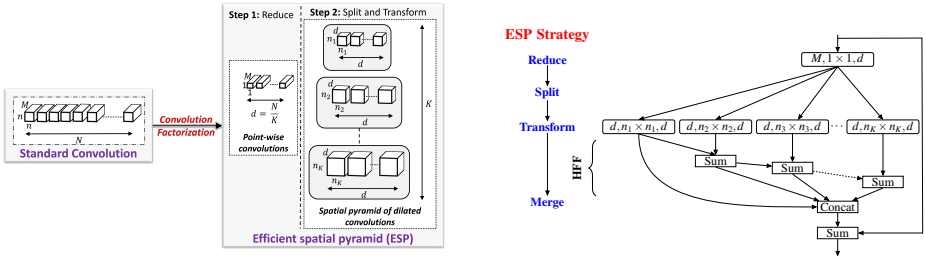
Source code: https://github.com/sacmehta/ESPNet

Abstract. We introduce a fast and efficient convolutional neural network, ESPNet, for semantic segmentation of high resolution images under resource constraints. ESPNet is based on a new convolutional module, efficient spatial pyramid (ESP), which is efficient in terms of computation, memory, and power. ESPNet is 22 times faster (on a standard GPU) and 180 times smaller than the state-of-the-art semantic segmentation network PSPNet [1], while its category-wise accuracy is only 8% less. We evaluated EPSNet on a variety of semantic segmentation datasets including Cityscapes, PASCAL VOC, and a breast biopsy whole slide image dataset. Under the same constraints on memory and computation, ESPNet outperforms all the current efficient CNN networks such as MobileNet, ShuffleNet, and ENet on both standard metrics and our newly introduced performance metrics that measure efficiency on edge devices. Our network can process high resolution images at a rate of 112 and 9 frames per second on a standard GPU and edge device, respectively.

1 Introduction

Deep convolutional neural networks (CNNs) have achieved high accuracy in visual scene understanding tasks [1,2,3]. While the accuracy of these networks has improved with their increase in depth and width, large networks are slow and power hungry. This is especially problematic on the computationally heavy task of semantic segmentation [4,5,6,7,8,9,10]. For example, PSPNet [1] has 65.7 million parameters and runs at about 1 FPS while discharging the battery of a standard laptop at a rate of 77 Watts. Many advanced real-world applications, such as self-driving cars, robots, and augmented reality, are sensitive and demand on-line processing of data locally on edge devices. These accurate networks require enormous resources and are not suitable for edge devices, which have limited energy overhead, restrictive memory constraints, and reduced computational capabilities.

Convolution factorization has demonstrated its success in reducing the computational complexity of deep CNNs (e.g. Inception[11,12,13], ResNext [14], and Xception [15]). We introduce an efficient convolutional module, ESP (efficient spatial pyramid), which is based on the convolutional factorization principle (Fig. 1). Based on these ESP modules, we introduce an efficient network structure, ESPNet, that can be easily deployed on resource-constrained edge devices. ESPNet is *fast*, *small*, *low power*, and *low latency*, yet still preserves segmentation accuracy.



(a) (b)

Fig. 1: (a) The standard convolution layer is decomposed into point-wise convolution and spatial pyramid of dilated convolutions to build an efficient spatial pyramid (ESP) module. (b) Block diagram of ESP module. The large effective receptive field of the ESP module introduces gridding artifacts, which are removed using hierarchical feature fusion (HFF). A skip-connection between input and output is added to improve the information flow. See Section 3 for more details. Dilated convolutional layers are denoted as (# input channels, effective kernel size, # output channels). The effective spatial dimensions of a dilated convolutional kernel are $n_k \times n_k$, where $n_k = (n-1)2^{k-1}+1,\ k = 1,\cdots,K$. Note that only $n \times n$ pixels participate in the dilated convolutional kernel. In our experiments $n = 3$ and $d = \frac{M}{K}$.

ESP is based on a convolution factorization principle that decomposes a standard convolution into two steps: (1) ***point-wise convolutions*** and (2) ***spatial pyramid of dilated convolutions***, as shown in Fig. 1. The point-wise convolutions help in reducing the computation, while the spatial pyramid of dilated convolutions re-samples the feature maps to learn the representations from large effective receptive field. We show that our ESP module is more efficient than other factorized forms of convolutions, such as Inception [11,12,13] and ResNext [14]. Under the same constraints on memory and computation, ESPNet outperforms MobileNet [16] and ShuffleNet [17] (two other efficient networks that are built upon the factorization principle). We note that existing spatial pyramid methods (e.g. the atrous spatial pyramid module in [3]) are computationally expensive and cannot be used at different spatial levels for learning the representations. In contrast to these methods, ESP is computationally efficient and can be used at different spatial levels of a CNN network. Existing networks based on dilated convolutions [1,3,18,19] are large and inefficient, but our ESP module generalizes the use of dilated convolutions in a novel and efficient way.

To analyze the performance of a CNN network on edge devices, we introduce several new performance metrics, such as sensitivity to GPU frequency and warp execution efficiency. To showcase the power of ESPNet, we evaluate our network on one of the most expensive tasks in AI and computer vision: semantic segmentation. ESPNet is empirically demonstrated to be more accurate, efficient, and fast than ENet [20], one of the most power-efficient semantic segmentation networks, while learning a similar number of parameters. Our results also show that ESPNet learns generalizable representations and outperforms ENet [20] and another efficient network ERFNet [21] on the unseen dataset. ESPNet can process a *high resolution RGB image* at a rate of 112 frames per second (FPS) on a high-end GPU, 21 FPS on a laptop, and 9 FPS on an edge device[3].

[3] We used a desktop with NVIDIA TitanX GPU, a laptop with GTX-960M GPU, and NVIDIA Jetson TX2 as an edge device. See **Appendix A** for more details.

2 Related Work

Multiple different techniques, such as convolution factorization, network compression, and low-bit networks, have been proposed to speed up convolutional neural networks. We, first, briefly describe these approaches and then provide a brief overview of CNN-based semantic segmentation.

Convolution factorization: Convolutional factorization decomposes the convolutional operation into multiple steps to reduce the computational complexity. This factorization has successfully shown its potential in reducing the computational complexity of deep CNN networks (e.g. Inception [11,12,13], factorized network [22], ResNext [14], Xception [15], and MobileNets [16]). ESP modules are also built on this factorization principle. The ESP module decomposes a convolutional layer into a point-wise convolution and spatial pyramid of dilated convolutions. This factorization helps in reducing the computational complexity, while simultaneously allowing the network to learn the representations from a large effective receptive field.

Network Compression: Another approach for building efficient networks is compression. These methods use techniques such as hashing [23], pruning [24], vector quantization [25], and shrinking [26,27] to reduce the size of the pre-trained network.

Low-bit networks: Another approach towards efficient networks is low-bit networks, which quantize the weights to reduce the network size and complexity (e.g. [28,29,30,31]).

Sparse CNN: To remove the redundancy in CNNs, sparse CNN methods, such as sparse decomposition [32], structural sparsity learning [33], and dictionary-based method [34], have been proposed.

We note that compression-based methods, low-bit networks, and sparse CNN methods are equally applicable to ESPNets and are complementary to our work.

Dilated convolution: Dilated convolutions [35] are a special form of standard convolutions in which the effective receptive field of kernels is increased by inserting zeros (or holes) between each pixel in the convolutional kernel. For a $n \times n$ dilated convolutional kernel with a dilation rate of r, the effective size of the kernel is $[(n-1)r+1]^2$. The dilation rate specifies the number of zeros (or holes) between pixels. However, due to dilation, only $n \times n$ pixels participate in the convolutional operation, reducing the computational cost while increasing the effective kernel size.

Yu and Koltun [18] stacked dilated convolution layers with increasing dilation rate to learn contextual representations from a large effective receptive field. A similar strategy was adopted in [19,36,37]. Chen *et al.* [3] introduced an atrous spatial pyramid (ASP) module. This module can be viewed as a parallelized version of [3]. These modules are computationally inefficient (e.g. ASPs have high memory requirements and learn many more parameters; see Section 3.2). Our ESP module also learns multi-scale representations using dilated convolutions in parallel; however, it is computationally efficient and can be used at any spatial level of a CNN network.

CNN for semantic segmentation: Different CNN-based segmentation networks have been proposed, such as multi-dimensional recurrent neural networks [38], encoder-decoders [20,21,39,40], hypercolumns [41], region-based representations [42,43], and cascaded networks [44]. Several supporting techniques along with these networks have been used for achieving high accuracy, including ensembling features [3], multi-stage

training [45], additional training data from other datasets [1,3], object proposals [46], CRF-based post processing [3], and pyramid-based feature re-sampling [1,2,3].

Encoder-decoder networks: Our work is related to this line of work. The encoder-decoder networks first learn the representations by performing convolutional and down-sampling operations. These representations are then decoded by performing up-sampling and convolutional operations. ESPNet first learns the encoder and then attaches a *light-weight decoder* to produce the segmentation mask. This is in contrast to existing networks where the decoder is either an exact replica of the encoder (e.g. [39]) or is relatively small (but not light weight) in comparison to the encoder (e.g. [20,21]).

Feature re-sampling methods: The feature re-sampling methods re-sample the convolutional feature maps at the same scale using different pooling rates [1,2] and kernel sizes [3] for efficient classification. Feature re-sampling is computationally expensive and is performed just before the classification layer to learn scale-invariant representations. We introduce a computationally efficient convolutional module that allows feature re-sampling at different spatial levels of a CNN network.

3 ESPNet

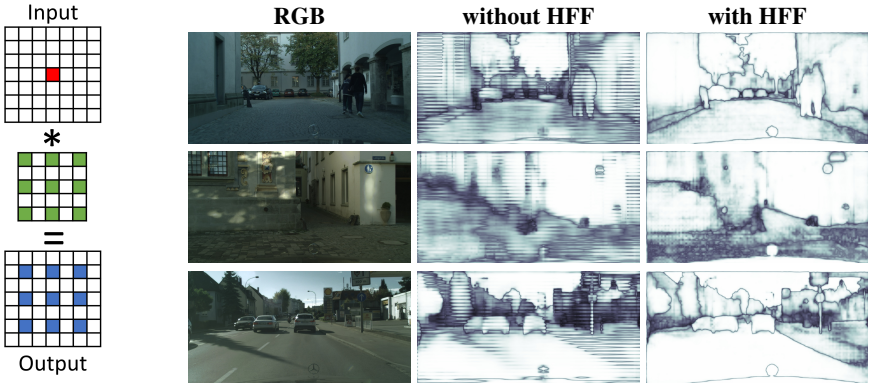
This section elaborates on the details of ESPNET and describes the core ESP module on which it is built. We compare ESP modules with similar CNN modules, such as Inception [11,12,13], ResNext [14], MobileNet[16], and ShuffleNet[17] modules.

3.1 ESP module

ESPNet is based on efficient spatial pyramid (ESP) modules, which are a factorized form of convolutions that decompose a standard convolution into a point-wise convolution and a spatial pyramid of dilated convolutions (see Fig. 1a). The point-wise convolution in the ESP module applies a 1×1 convolution to project high-dimensional feature maps onto a low-dimensional space. The spatial pyramid of dilated convolutions then re-samples these low-dimensional feature maps using K, $n \times n$ dilated convolutional kernels simultaneously, each with a dilation rate of 2^{k-1}, $k = \{1, \cdots, K\}$. This factorization drastically reduces the number of parameters and the memory required by the ESP module, while preserving a large effective receptive field $\left[(n-1)2^{K-1}+1\right]^2$. This pyramidal convolutional operation is called a spatial pyramid of dilated convolutions, because each dilated convolutional kernel learns weights with different receptive fields and so resembles a spatial pyramid.

A standard convolutional layer takes an input feature map $\mathbf{F}_i \in \mathbb{R}^{W \times H \times M}$ and applies N kernels $\mathbf{K} \in \mathbb{R}^{m \times n \times M}$ to produce an output feature map $\mathbf{F}_o \in \mathbb{R}^{W \times H \times N}$, where W and H represent the width and height of the feature map, m and n represent the width and height of the kernel, and M and N represent the number of input and output feature channels. For the sake of simplicity, we will assume that $m = n$. A standard convolutional kernel thus learns n^2MN parameters. These parameters are *multiplicatively* dependent on the spatial dimensions of the $n \times n$ kernel and the number of input M and output N channels.

Width divider K: To reduce the computational cost, we introduce a simple hyper-parameter K. The role of K is to shrink the dimensionality of the feature maps uniformly across each ESP module in the network. *Reduce:* For a given K, the ESP module first



(a) (b)

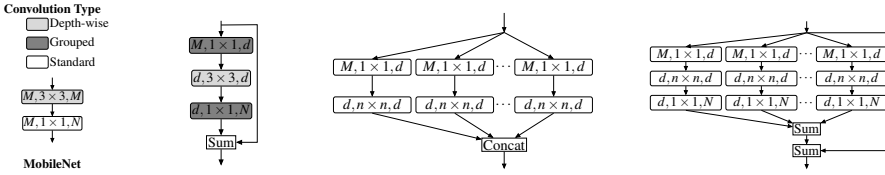
Fig. 2: (a) An example illustrating a gridding artifact with a single active pixel (red) convolved with a 3×3 dilated convolutional kernel with dilation rate $r = 2$. (b) Visualization of feature maps of ESP modules with and without hierarchical feature fusion (HFF). HFF in ESP eliminates the gridding artifact. Best viewed in color.

reduces the feature maps from M-dimensional space to $\frac{N}{K}$-dimensional space using a point-wise convolution (Step 1 in Fig. 1a). *Split:* The low-dimensional feature maps are then split across K parallel branches. *Transform:* Each branch then processes these feature maps simultaneously using $n \times n$ dilated convolutional kernels with different dilation rates given by 2^{k-1}, $k = \{1, \cdots, K-1\}$ (Step 2 in Fig. 1a). *Merge:* The output of these K parallel dilated convolutional kernels is then concatenated to produce an N-dimensional output feature map[4]. Fig. 1b visualizes the *reduce-split-transform-merge* strategy used in ESP modules.

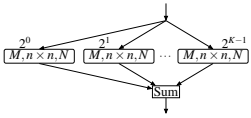
The ESP module has $\frac{MN}{K} + \frac{(nN)^2}{K}$ parameters and its effective receptive field is $\left[(n-1)2^{K-1}+1\right]^2$. Compared to the n^2NM parameters of the standard convolution, factorizing it using the two steps reduces the total number of parameters in the ESP module by a factor of $\frac{n^2MK}{M+n^2N}$, while increasing the effective receptive field by $\sim [2^{K-1}]^2$. For example, an ESP module learns ~ 3.6 times fewer parameters with an effective receptive field of 17×17 than a standard convolutional kernel with an effective receptive field of 3×3 for $n = 3$, $N = M = 128$, and $K = 4$.

Hierarchical feature fusion (HFF) for de-gridding: While concatenating the outputs of dilated convolutions give the ESP module a large effective receptive field, it introduces unwanted checkerboard or gridding artifacts, as shown in Fig. 2. To address the gridding artifact in ESP, the feature maps obtained using kernels of different dilation rates are hierarchically added before concatenating them (HFF in Fig. 1b). This solution is simple and effective and does not increase the complexity of the ESP module, in contrast to existing methods that remove the gridding artifact by learning more parame-

[4] In general, $\frac{N}{K}$ may not be a perfect divisor, and therefore concatenating K, $\frac{N}{K}$-dimensional feature maps would not result in an N-dimensional output. To handle this, we use $\left(N-(K-1)\lfloor\frac{N}{K}\rfloor\right)$ kernels with a dilation rate of 2^0 and $\lfloor\frac{N}{K}\rfloor$ kernels for each dilation rate 2^{k-1} for $k = \{2, \cdots, K\}$.



(a) MobileNet (b) ShuffleNet (c) Inception (d) ResNext



(e) ASP

Module	# Parameters	Memory (in MB)	Effective Receptive Field
MobileNet	$M(n^2+N) = 11,009$	$(M+N)WH = 2.39$	$[n]^2 = 3\times 3$
ShuffleNet	$\frac{d}{g}(M+N)+n^2d = \mathbf{2,180}$	$WH(2*d+N) = 1.67$	$[n]^2 = 3\times 3$
Inception	$K(Md+n^2d^2) = 28,000$	$2KWHd = 2.39$	$[n]^2 = 3\times 3$
ResNext	$K(Md+d^2n^2+dN) = 38,000$	$KWH(2d+N) = 8.37$	$[n]^2 = 3\times 3$
ASP	$KMNn^2 = 450,000$	$KWHN = 5.98$	$\left[(n-1)2^{K-1}+1\right]^2 = \mathbf{33\times 33}$
ESP (Fig. 1b)	$Md+Kn^2d^2 = 20,000$	$WHd(K+1) = \mathbf{1.43}$	$\left[(n-1)2^{K-1}+1\right]^2 = \mathbf{33\times 33}$

Here, $M = N = 100$, $n = 3$, $K = 5$, $d = \frac{N}{K} = 20$, $g = 2$, and $W = H = 56$.

(f) Comparison between different modules

Fig. 3: Different types of convolutional modules for comparison. We denote the layer as (# input channels, kernel size, # output channels). Dilation rate in (e) is indicated on top of each layer. Here, g represents the number of convolutional groups in grouped convolution [48]. For simplicity, we only report the memory of convolutional layers in (d). For converting the required memory to bytes, we multiply it by 4 (1 float requires 4 bytes for storage).

ters using dilated convolutional kernels with small dilation rates [19,37]. To improve the gradient flow inside the network, the input and output feature maps of the ESP module are combined using an element-wise sum [47].

3.2 Relationship with other CNN modules

The ESP module shares similarities with the following CNN modules.

MobileNet module: The MobileNet module [16], visualized in Fig. 3a, uses a depth-wise separable convolution [15] that factorizes a standard convolutions into depth-wise convolutions (*transform*) and point-wise convolutions (*expand*). It learns less parameters, has high memory requirement, and low receptive field than the ESP module. An extreme version of the ESP module (with $K = N$) is almost identical to the MobileNet module, differing only in the order of convolutional operations. In the MobileNet module, the spatial convolutions are followed by point-wise convolutions; however, in the ESP module, point-wise convolutions are followed by spatial convolutions. Note that the effective receptive field of an ESP module ($\left[(n-1)2^{K-1}+1\right]^2$) is higher than a MobileNet module ($[n]^2$).

ShuffleNet module: The ShuffleNet module [17], shown in Fig. 3b, is based on the principle of *reduce-transform-expand*. It is an optimized version of the bottleneck block in ResNet [47]. To reduce computation, Shufflenet makes use of grouped convolutions [48] and depth-wise convolutions [15]. It replaces 1×1 and 3×3 convolutions in the bottleneck block in ResNet with 1×1 grouped convolutions and 3×3 depth-wise separable convolutions, respectively. The Shufflenet module learns many less parameters than the ESP module, but has higher memory requirements and a smaller receptive field.

Inception module: Inception modules [11,12,13] are built on the principle of *split-reduce-transform-merge*. These modules are usually heterogeneous in number of channels and kernel size (e.g. some of the modules are composed of standard and factored

convolutions). In contrast to the Inception modules, ESP modules are straightforward and simple to design. For the sake of comparison, the homogeneous version of an Inception module is shown in Fig. 3c. Fig. 3f compares the Inception module with the ESP module. ESP (1) learns fewer parameters, (2) has a low memory requirement, and (3) has a larger effective receptive field.

ResNext module: A ResNext module [14], shown in Fig. 3d, is a parallel version of the bottleneck module in ResNet [47] and is based on the principle of *split-reduce-transform-expand-merge*. The ESP module is similar to ResNext in the sense that it involves branching and residual summation. However, the ESP module is more efficient in memory and parameters and has a larger effective receptive field.

Atrous spatial pyramid (ASP) module: An ASP module [3], shown in Fig. 3e, is built on the principle of *split-transform-merge*. The ASP module involves branching with each branch learning kernel at a different receptive field (using dilated convolutions). Though ASP modules tend to perform well in segmentation tasks due to their high effective receptive fields, ASP modules have high memory requirements and learn many more parameters. Unlike the ASP module, the ESP module is computationally efficient.

4 Experiments

Semantic segmentation is one of the most expensive task in AI and computer vision. To showcase the power of ESPNet, ESPNet's performance is evaluated on several datasets for semantic segmentation and compared to the state-of-the-art networks.

4.1 Experimental set-up

Network structure: ESPNet uses ESP modules for learning convolutional kernels as well as down-sampling operations, except for the first layer which is a standard strided convolution. All layers (convolution and ESP modules) are followed by a batch normalization [49] and a PReLU [50] non-linearity except for the last point-wise convolution, which has neither batch normalization nor non-linearity. The last layer feeds into a softmax for pixel-wise classification.

Different variants of ESPNet are shown in Fig. 4. The first variant, ESPNet-A (Fig. 4a), is a standard network that takes an RGB image as an input and learns representations at different spatial levels[5] using the ESP module to produce a segmentation mask. The second variant, ESPNet-B (Fig. 4b), improves the flow of information inside ESPNet-A by sharing the feature maps between the previous strided ESP module and the previous ESP module. The third variant, ESPNet-C (Fig. 4c), reinforces the input image inside ESPNet-B to further improve the flow of information. These three variants produce outputs whose spatial dimensions are $\frac{1}{8}th$ of the input image. The fourth variant, ESPNet (Fig. 4d), adds a light weight decoder (built using a principle of *reduce-upsample-merge*) to ESPNet-C that outputs the segmentation mask of the same spatial resolution as the input image.

To build deeper computationally efficient networks for edge devices without changing the network topology, a hyper-parameter α controls the depth of the network; the ESP module is repeated α_l times at spatial level l. CNNs require more memory at higher

[5] At each spatial level l, the spatial dimensions of the feature maps are the same. To learn representations at different spatial levels, a down-sampling operation is performed (see Fig. 4a).

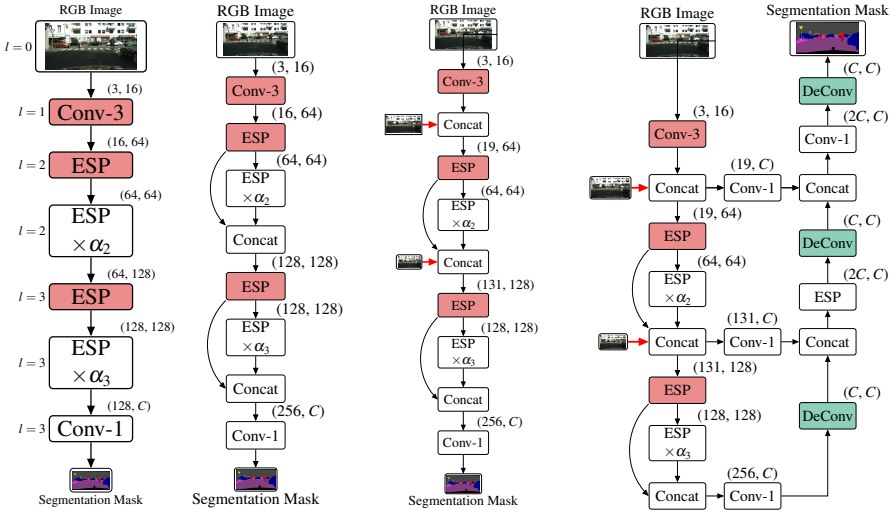


Fig. 4: The path from ESPNet-A to ESPNet. Red and green color boxes represent the modules responsible for down-sampling and up-sampling operations, respectively. Spatial-level l is indicated on the left of every module in (a). We denote each module as (# input channels, # output channels). Here, Conv-n represents $n \times n$ convolution.

spatial levels (at $l = 0$ and $l = 1$) because of the high spatial dimensions of feature maps at these levels. To be memory efficient, neither the ESP nor the convolutional modules are repeated at these spatial levels. The building block functions used to build the ESPNet (from ESPNet-A to ESPNet) are discussed in **Appendix B**.

Dataset: We evaluated the ESPNet on the Cityscapes dataset [6], an urban visual scene understanding dataset that consists of 2,975 training, 500 validation, and 1,525 test high-resolution images. The dataset was captured across 50 cities and in different seasons. The task is to segment an image into 19 classes belonging to 7 categories (e.g. person and rider classes belong to the same category *human*). We evaluated our networks on the test set using the Cityscapes *online* server.

To study the generalizability, we tested the ESPNet on an unseen dataset. We used the Mapillary dataset [51] for this task because of its diversity. We mapped the annotations (65 classes) in the validation set (# 2,000 images) to seven categories in the Cityscape dataset. To further study the segmentation power of our network, we trained and tested the ESPNet on two other popular datasets from different domains. First, we used the widely known PASCAL VOC dataset [52] that has 1,464 training images, 1,448 validation images, and 1,456 test images. The task is to segment an image into 20 foreground classes. We evaluate our networks on the test set (comp6 category) using the PASCAL VOC *online* server. Following the convention, we used additional images from [53,54]. Secondly, we used a breast biopsy whole slide image dataset [36], chosen because tissue structures in biomedical images vary in size and shape and because this dataset allowed us to check the potential of learning representations from a large receptive field. The dataset consists of 30 training images and 28 validation images, whose

average size is $10,000 \times 12,000$, much larger than natural scene images. The task is to segment the images into 8 biological tissue labels; details are in [36].

Performance evaluation metrics: Most traditional CNNs measure network performance in terms of accuracy, latency, number of network parameters, and network size (e.g. [16,17,20,21,55]). These metrics provide high-level insight about the network, but fail to demonstrate the efficient usage of limited available hardware resources. In addition to these metrics, we introduce several *system-level metrics* to characterize the performance of a CNN on resource-constrained devices [56,57].

Segmentation accuracy is measured as a mean Intersection over Union (mIOU) score between the ground truth and the predicted segmentation mask.

Latency represents the amount of time a CNN network takes to process an image. This is usually measured in terms of frames per second (FPS).

Network parameters represents the number of parameters learned by the network.

Network size represents the amount of storage space required to store the network parameters. An efficient network should have a smaller network size.

Sensitivity to GPU frequency measures the computational capability of an application and is defined as a ratio of percentage change in execution time to the percentage change in GPU frequency. A higher value indicates that the application tends to utilize the GPU more efficiently.

Utilization rates measures the utilization of compute resources (CPU, GPU, and memory) while running on an edge device. In particular, computing units in edge devices (e.g. Jetson TX2) share memory between CPU and GPU.

Warp execution efficiency is defined as the average percentage of active threads in each executed warp. GPUs schedule threads in the form of warps, and each thread inside the warp is executed in *single instruction multiple data* fashion. A high value of warp execution efficiency represents efficient usage of GPU.

Memory efficiency is the ratio of number of bytes requested/stored to the number of bytes transfered from/to device (or shared) memory to satisfy load/store requests. Since memory transactions are in blocks, this metric allows us to determine how efficiently we are using the memory bandwidth.

Power consumption is the amount of average power consumed by the application during inference.

Training details: ESPNet networks were trained using PyTorch [58] with CUDA 9.0 and cuDNN back-ends. ADAM [59] was used with an initial learning rate of 0.0005, and decayed by two after every 100 epochs and with a weight decay of 0.0005. An inverse class probability weighting scheme was used in the cross-entropy loss function to address the class imbalance [20,21]. Following [20,21], the weights were initialized randomly. Standard strategies, such as scaling, cropping and flipping, were used to augment the data. The image resolution in the Cityscape dataset is 2048×1024, and all the accuracy results were reported at this resolution. For training the networks, we sub-sampled the RGB images by two. When the output resolution was smaller than 2048×1024, the output was up-sampled using bi-linear interpolation. For training on the PASCAL dataset, we used a fixed image size of 512×512. For the WSI dataset, the patch-wise training approach was followed [36]. ESPNet was trained in two stages.

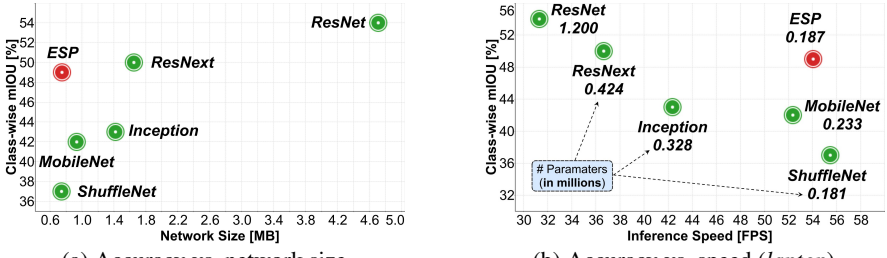
First, ESPNet-C was trained with down-sampled annotations. Second, a light-weight decoder was attached to ESPNet-C and then, the entire ESPNet network was trained.

Three different GPU devices were used for our experiments: (1) a desktop with a NVIDIA TitanX GPU (3,584 CUDA cores), (2) a laptop with a NVIDIA GTX-960M GPU (640 CUDA cores), and (3) an edge device with NVIDIA Jetson TX2 (256 CUDA cores). See **Appendix A** for more details about the hardware. Unless and otherwise stated explicitly, statistics, such as power consumption and inference speed, are reported for an RGB image of size 1024×512 averaged over 200 trials. For collecting the hardware-level statistics, NVIDIA's and Intel's hardware profiling and tracing tools, such as NVPROF [60], Tegrastats [61], and PowerTop [62], were used. In our experiments, we will refer to ESPNet with $\alpha_2 = 2$ and $\alpha_3 = 8$ as ESPNet until and otherwise stated explicitly.

4.2 Results on the Cityscape dataset

Comparison with state-of-the-art efficient convolutional modules: In order to understand the ESP module, we replaced the ESP modules in ESPNet-C with state-of-the-art efficient convolutional modules, sketched in Fig. 3 (MobileNet [16], ShuffleNet [17], Inception [11,12,13], ResNext [14], and ResNet [47]) and evaluate their performance on the Cityscape validation dataset. We did not compare with ASP [3], because it is computationally expensive and not suitable for edge devices. Fig. 5 compares the performance of ESPNet-C with different convolutional modules. Our ESP module outperformed MobileNet and ShuffleNet modules by 7% and 12%, respectively, while learning a similar number of parameters and having comparable network size and inference speed. Furthermore, the ESP module delivered comparable accuracy to ResNext and Inception more efficiently. A basic ResNet module (stack of two 3×3 convolutions with a skip-connection) delivered the best performance, but had to learn $6.5\times$ more parameters.

Comparison with state-of-the-art segmentation methods: We compared the performance of ESPNet with state-of-the-art semantic segmentation networks. These networks either use a pre-trained network (*VGG* [63]: FCN-8s [45] and SegNet [39], *ResNet* [47]: DeepLab-v2 [3] and PSPNet [1], and *SqueezeNet* [55]: SQNet [64]) or were trained from scratch (ENet [20] and ERFNet [21]). Fig. 6 compares ESPNet with state-of-the-art methods. ESPNet is 2% more accurate than ENet [20], while running $1.27\times$ and $1.16\times$ faster on a desktop and a laptop, respectively. ESPNet makes some

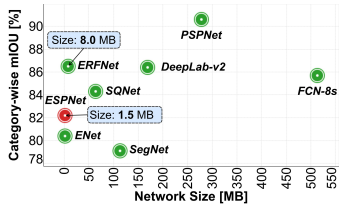


(a) Accuracy vs. network size (b) Accuracy vs. speed (*laptop*)

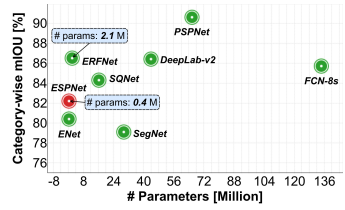
Fig. 5: Comparison between state-of-the-art efficient convolutional modules. For a fair comparison between different modules, we used $K = 5$, $d = \frac{N}{K}$, $\alpha_2 = 2$, and $\alpha_3 = 3$. We used standard strided convolution for down-sampling. For ShuffleNet, we used $g = 4$ and $K = 4$ so that the resultant ESPNet-C network has the same complexity as with the ESP block.

Network	mIOU Class	mIOU Category
ENet [20]	58.3	80.4
ERFNet [21]	68.0	86.5
SQNet [27]	59.8	84.3
SegNet [39]	57.0	79.1
ESPNet (Ours)	60.3	82.2
FCN-8s [39]	65.3	85.7
DeepLab-v2 [3]	70.4	86.4
PSPNet [1]	**78.4**	**90.6**

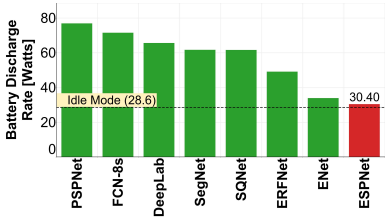
(a) Test set



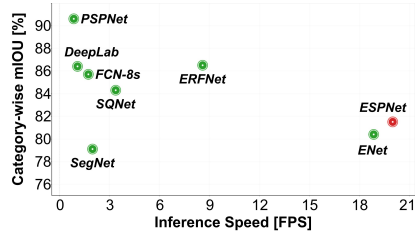
(b) Accuracy vs. network size



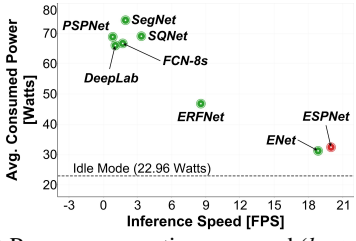
(c) Accuracy vs. # parameters



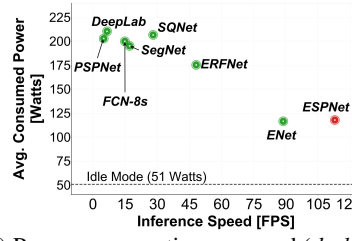
(d) Battery discharge rate vs. network (*laptop*)



(e) Accuracy vs. speed (*laptop*)



(f) Power consumption vs. speed (*laptop*)



(g) Power consumption vs. speed (*desktop*)

Fig. 6: Comparison between state-of-the-art segmentation methods on the Cityscape test set on two different devices. All networks (FCN-8s [45], SegNet [39], SQNet [64], ENet [20], DeepLab-v2 [3], PSPNet [1], and ERFNet [21]) were without conditional random field and converted to PyTorch for a fair comparison. Best viewed in color.

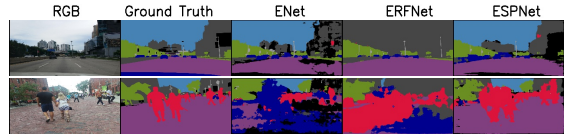
mistakes between classes that belong to the same category, and hence has a lower class-wise accuracy (see **Appendix F** for the confusion matrix). For example, a rider can be confused with a person. However, ESPNet delivers a good category-wise accuracy. ESPNet had 8% lower category-wise mIOU than PSPNet [1], while learning 180× fewer parameters. ESPNet had lower power consumption, had lower battery discharge rate, and was significantly faster than state-of-the-art methods, while still achieving a competitive category-wise accuracy; this makes ESPNet suitable for segmentation on edge devices. ERFNet, an another efficient segmentation network, delivered good segmentation accuracy, but has 5.5× more parameters, is 5.44× larger, consumes more power, and has a higher battery discharge rate than ESPNet. Also, ERFNet does not utilize limited available hardware resources efficiently on edge devices (Section 4.4).

4.3 Segmentation results on other datasets

Unseen dataset: Table 1a compares the performance of ESPNet to that of ENet [20] and ERFNet [21] on an unseen dataset. These networks were trained on the Cityscapes dataset [6] and tested on the Mapillary (unseen) dataset [51]. ENet and ERFNet were chosen, because ENet was one of the most power efficient segmentation networks, while

	mIOU	# Params
ENet [20]	0.33	0.364
ERFNet [21]	0.25	2.06
ESPNet	**0.40**	0.364

(a) Mapillary validation set [51]



(b) Mapillary validation set [51] (unseen)

Network	ESPNet (Ours)	SegNet [39]	RefineNet [44]	DeepLab [3]	PSPNet [1]	LRR [65]	Dilation-8 [18]	FCN-8s [45]
# Params	**0.364**	29.5	42.6	44.04	65.7	48	141.13	134.5
mIOU	63.01	59.10	82.40	79.70	**85.40**	79.30	75.30	67.20

(c) PASCAL VOC test set [52]

Network	Module	mIOU	# Params
ESPNet (Ours)*	ESP	44.03	**2.75**
SegNet [39]	VGG	37.6	12.80
Mehta *et al.* [36]	ResNet	**44.20**	26.03

(d) Breast biopsy validation set [36]

Table 1: Results on different datasets. Here, the number of parameters are in million and * indicates that a wider version of ESPNet was used. At $l = \{1,2,3\}$, we used (16, 128, 256) as the number of output channels and $K = 4$. See **Appendix F** for more sample images.

ERFNet has high accuracy and moderate efficiency. Our experiments show that ESPNet learns good generalizable representations of objects and outperforms ENet and ERFNet both qualitatively and quantitatively on the unseen dataset.

PASCAL VOC 2012 dataset: (Table 1c) On the PASCAL dataset, ESPNet is 4% more accurate than SegNet, one of the smallest network on the PASCAL VOC, while learning 81× fewer parameters. ESPNet is 22% less accurate than PSPNet (one of the most accurate network on the PASCAL VOC) while learning 180× fewer parameters.

Breast biopsy dataset: (Table 1d) On the breast biopsy dataset, ESPNet achieved the same accuracy as [36] while learning 9.5× less parameters.

4.4 Performance analysis on an edge device

We measure the performance on the NVIDIA Jetson TX2, a computing platform for edge devices. Performance analysis results are given in Fig. 7.

Network size: Fig. 7a compares the uncompressed 32-bit network size of ESPNet with ENet and ERFNet. ESPNet had a 1.12× and 5.45× smaller network than ENet and ERFNet, respectively, which reflects well on the architectural design of ESPNet.

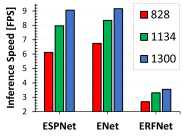
Inference speed and sensitivity to GPU frequency: Fig. 7b compares the inference speed of ESPNet with ENet and ERFNet. ESPNet had almost the same frame rate as ENet, but it was more sensitive to GPU frequency (Fig. 7c). As a consequence, ESPNet achieved a higher frame rate than ENet on high-end graphic cards, such as the GTX-960M and TitanX (see Fig. 6). For example, ESPNet is 1.27× faster than ENet on an NVIDIA TitanX. ESPNet is about 3× faster than ERFNet on an NVIDIA Jetson TX2.

Utilization rates: Fig. 7d compares the CPU, GPU, and memory utilization rates of different networks. These networks are throughput intensive, and therefore, GPU utilization rates are high, while CPU utilization rates are low for these networks. Memory utilization rates are significantly different for these networks. The memory footprint of ESPNet is low in comparison to ENet and ERFNet, suggesting that ESPNet is suitable for memory-constrained devices.

Warp execution efficiency: Fig. 7e compares the warp execution efficiency of ESPNet with ENet and ERFNet. The warp execution of ESPNet was about 9% higher than

Network	Size
ENet	1.64 MB
ERFNet	7.95 MB
ESPNet	**1.46 MB**

(a)



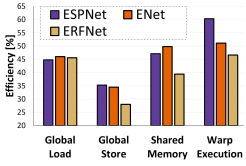
(b)

Network	Sensitivity to GPU freq.	
	828 to 1134	1134 to 1300
ENet	71%	70%
ERFNet	69%	53%
ESPNet	86%	95%

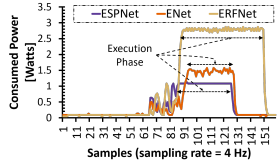
(c)

Network	Utilization (%)		
	CPU	GPU	Memory
ENet	20.5	99.00	50.6
ERFNet	**19.7**	99.00	61.3
ESPNet	20.3	99.00	**44.0**

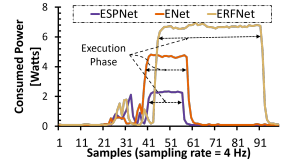
(d)



(e)



(f) GPU freq. @ 828 MHz



(g) GPU freq. @ 1,134 MHz

Fig. 7: Performance analysis of ESPNet with ENet and ERFNet on a NVIDIA Jetson TX2: (a) network size, (b) inference speed vs. GPU frequency (in MHz), (c) sensitivity analysis, (d) utilization rates, (e) efficiency rates, and (f, g) power consumption at two different GPU frequencies. In (d), the statistics for the network's initialization phase were not considered, because they were the same across all networks. See **Appendix E** for time vs. utilization plots. Best viewed in color.

ENet and about 14% higher than ERFNet. This indicates that ESPNet has less warp divergence and promotes the efficient usage of limited GPU resources available on edge devices. We note that warp execution efficiency gives a better insight into the utilization of GPU resources than the GPU utilization rate. GPU frequency will be busy even if few warps are active, resulting in a high GPU utilization rate.

Memory efficiency: (Fig. 7e) All networks have similar global load efficiency, but ERFNet has a poor store and shared memory efficiency. This is likely due to the fact that ERFNet spends 20% of the compute power performing memory alignment operations, while ESPNet and ENet spend 4.2% and 6.6% time for this operation, respectively. See **Appendix C** for the compute-wise break down of different kernels.

Power consumption: Fig. 7f and 7g compares the power consumption of ESPNet with ENet and ERFNet at two different GPU frequencies. The average power consumption (during network execution phase) of ESPNet, ENet, and ERFNet were 1 W, 1.5 W, and 2.9 W at a GPU frequency of 824 MHz and 2.2 W, 4.6 W, and 6.7 W at a GPU frequency of 1,134 MHz, respectively; suggesting ESPNet is a power-efficient network.

4.5 Ablation studies: The path from ESPNet-A to ESPNet

Larger networks or ensembling the output of multiple networks delivers better performance [1,3,19], but with ESPNet (sketched in Fig. 4), the goal is an efficient network for edge devices. To improve the performance of ESPNet while maintaining efficiency, a systematic study of design choices was performed. Table 2 summarizes the results.

ReLU vs PReLU: (Table 2a) Replacing ReLU [66] with PReLU [50] in ESPNet-A improved the accuracy by 2%, while having a minimal impact on the network complexity.

Residual learning in ESP: (Table 2b) The accuracy of ESPNet-A dropped by about 2% when skip-connections in ESP (Fig. 1b) modules were removed. This verifies the effectiveness of the residual learning.

Down-sampling: (Table 2c) Replacing the standard strided convolution with the strided ESP in ESPNet-A improved accuracy by 1% with 33% parameter reduction.

Activation	mIOU	# Params°
ReLU	0.36	0.183
PReLU	**0.38**	0.183

(a)

Module	mIOU	# Params°
ESP w/o RL	0.37	0.183
ESP w/ RL	**0.39**	0.183

where RL represents residual learning

(b)

Downsample	mIOU	# Params°
Strided Convolution	0.38	0.274
Strided ESP	**0.39**	**0.183**

(c)

	Width divider K					
	2	4	5	6	7	8
mIOU	**0.415**	0.378	0.381	0.359	0.321	0.303
# Params°	0.358	0.215	0.183	0.165	0.152	**0.143**
ERF ($n^2 = n \times n$)	5^2	17^2	33^2	65^2	129^2	$\mathbf{257}^2$

ERF represents effective receptive field

(d)

Network	mIOU	# Params°
ESPNet-A*	0.39	**0.183**
ESPNet-B	0.40	0.186
ESPNet-C	**0.42**	0.187
ESPNet-C†	**0.42**	0.206

(e)

α_3	ESPNet-C (Fig. 4c)			ESPNet (Fig. 4d)		
	mIOU	# Params (in million)	Network size	mIOU	# Params (in million)	Network size
3	49.0	**0.187**	**0.75 MB**	56.3	**0.202**	**0.82 MB**
5	51.2	0.252	1.01 MB	57.9	0.267	1.07 MB
8	**53.3**	0.349	1.40 MB	**61.4**	0.364	1.46 MB

(f)

Table 2: The path from ESPNet-A to ESPNet. Here, * denotes that strided ESP was used for down-sampling, † indicates that the input reinforcement method was replaced with input-aware fusion method [36], and $^\circ$ denotes the values are in million. All networks in (a-e) are trained for 100 epochs with $\alpha_3 = 3$ while networks in (f) are trained for 300 epochs with variable α_3.

Width divider (K): (Table 2d) Increasing K enlarges the effective receptive field of the ESP module, while simultaneously decreasing the number of network parameters. Importantly, ESPNet-A's accuracy decreased with increasing K. For example, raising K from 2 to 8 caused ESPNet-A's accuracy to drop by 11%. This drop in accuracy is explained in part by the ESP module's effective receptive field growing beyond the size of its input feature maps. For an image with size 1024×512, the spatial dimensions of the input feature maps at spatial level $l = 2$ and $l = 3$ are 256×128 and 128×64, respectively. However, some of the kernels have larger receptive fields (257×257 for $K = 8$). The weights of such kernels do not contribute to learning, thus resulting in lower accuracy. At $K = 5$, we found a good trade-off between number of parameters and accuracy, and therefore, we used $K = 5$ in our experiments.

ESPNet-A $\rightarrow$ ESPNet-C: (Table 2e) Replacing the convolution-based network width expansion operation in ESPNet-A with the concatenation operation in ESPNet-B improved the accuracy by about 1% and did not increase the number of network parameters noticeably. With input reinforcement (ESPNet-C), the accuracy of ESPNet-B further improved by about 2%, while not increasing the network parameters drastically. This is likely due to the fact that the input reinforcement method establishes a direct link between the input image and encoding stage, improving the flow of information.

The closest work to our input reinforcement method is the input-aware fusion method of [36], which learns representations on the down-sampled input image and additively combines them with the convolutional unit. When the proposed input reinforcement method was replaced with the input-aware fusion in [36], no improvement in accuracy was observed, but the number of network parameters increased by about 10%.

ESPNet-C $\rightarrow$ ESPNet: (Table 2f) Adding a light-weight decoder to ESPNet-C improved the accuracy by about 6%, while increasing the number of parameters and network size by merely 20,000 and 0.06 MB from ESPNet-C to ESPNet, respectively.

5 Conclusion

We introduced a semantic segmentation network, ESPNet, based on an efficient spatial pyramid module. In addition to legacy metrics, we introduced several new system-level metrics that help to analyze the performance of a CNN network. Our empirical analysis suggests that ESPNets are fast and efficient. We also demonstrated that ESPNet learns good generalizable representations of the objects and perform well in the wild.

Acknowledgement

This research was supported by Washington State Department of Transportation research grant T1461-47. We would also like to acknowledge NVIDIA Corporation for donating the Jetson TX2 board and the Titan X Pascal GPU used for this research.

References

1. Zhao, H., Shi, J., Qi, X., Wang, X., Jia, J.: Pyramid scene parsing network. In: CVPR. (2017)
2. He, K., Zhang, X., Ren, S., Sun, J.: Spatial pyramid pooling in deep convolutional networks for visual recognition. In: ECCV. (2014)
3. Chen, L.C., Papandreou, G., Kokkinos, I., Murphy, K., Yuille, A.L.: Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. arXiv preprint arXiv:1606.00915 (2016)
4. Ess, A., Müller, T., Grabner, H., Van Gool, L.J.: Segmentation-based urban traffic scene understanding. In: BMVC. (2009)
5. Geiger, A., Lenz, P., Stiller, C., Urtasun, R.: Vision meets robotics: The kitti dataset. The International Journal of Robotics Research (2013)
6. Cordts et al.: The cityscapes dataset for semantic urban scene understanding. In: CVPR. (2016)
7. Menze, M., Geiger, A.: Object scene flow for autonomous vehicles. In: CVPR. (2015)
8. Franke, U., Pfeiffer, D., Rabe, C., Knoeppel, C., Enzweiler, M., Stein, F., Herrtwich, R.G.: Making bertha see. In: ICCV Workshops, IEEE (2013)
9. Xiang, Y., Fox, D.: Da-rnn: Semantic mapping with data associated recurrent neural networks. Robotics: Science and Systems (RSS) (2017)
10. Kundu, A., Li, Y., Dellaert, F., Li, F., Rehg, J.M.: Joint semantic segmentation and 3d reconstruction from monocular video. In: ECCV. (2014)
11. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., et al.: Going deeper with convolutions. In: CVPR. (2015)
12. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: CVPR. (2016)
13. Szegedy, C., Ioffe, S., Vanhoucke, V.: Inception-v4, inception-resnet and the impact of residual connections on learning. CoRR (2016)
14. Xie, S., Girshick, R., Dollár, P., Tu, Z., He, K.: Aggregated residual transformations for deep neural networks. In: CVPR. (2017)
15. Chollet, F.: Xception: Deep learning with depthwise separable convolutions. CVPR (2017)
16. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017)
17. Zhang, X., Zhou, X., Lin, M., Sun, J.: Shufflenet: An extremely efficient convolutional neural network for mobile devices. arXiv preprint arXiv:1707.01083 (2017)

18. Yu, F., Koltun, V.: Multi-scale context aggregation by dilated convolutions. ICLR (2016)
19. Yu, F., Koltun, V., Funkhouser, T.: Dilated residual networks. CVPR (2017)
20. 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)
21. 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 (2018)
22. Jin, J., Dundar, A., Culurciello, E.: Flattened convolutional neural networks for feedforward acceleration. arXiv preprint arXiv:1412.5474 (2014)
23. Chen, W., Wilson, J., Tyree, S., Weinberger, K., Chen, Y.: Compressing neural networks with the hashing trick. In: ICML. (2015)
24. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. ICLR (2016)
25. Wu, J., Leng, C., Wang, Y., Hu, Q., Cheng, J.: Quantized convolutional neural networks for mobile devices. In: CVPR. (2016)
26. Zhao, H., Qi, X., Shen, X., Shi, J., Jia, J.: Icnet for real-time semantic segmentation on high-resolution images. arXiv preprint arXiv:1704.08545 (2017)
27. Jaderberg, M., Vedaldi, A., Zisserman, A.: Speeding up convolutional neural networks with low rank expansions. BMVC (2014)
28. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: Xnor-net: Imagenet classification using binary convolutional neural networks. In: ECCV. (2016)
29. Hwang, K., Sung, W.: Fixed-point feedforward deep neural network design using weights 1, 0, and -1. In: 2014 IEEE Workshop on Signal Processing Systems (SiPS). (2014)
30. Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks: Training neural networks with weights and activations constrained to+ 1 or- 1. arXiv preprint arXiv:1602.02830 (2016)
31. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y.: Quantized neural networks: Training neural networks with low precision weights and activations. arXiv preprint arXiv:1609.07061 (2016)
32. Liu, B., Wang, M., Foroosh, H., Tappen, M., Pensky, M.: Sparse convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2015) 806–814
33. Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H.: Learning structured sparsity in deep neural networks. In: Advances in Neural Information Processing Systems. (2016) 2074–2082
34. Bagherinezhad, H., Rastegari, M., Farhadi, A.: Lcnn: Lookup-based convolutional neural network. In: Proc. IEEE CVPR. (2017)
35. Holschneider, M., Kronland-Martinet, R., Morlet, J., Tchamitchian, P.: A real-time algorithm for signal analysis with the help of the wavelet transform. In: Wavelets. (1990)
36. Mehta, S., Mercan, E., Bartlett, J., Weaver, D.L., Elmore, J.G., Shapiro, L.G.: Learning to segment breast biopsy whole slide images. WACV (2018)
37. Wang, P., Chen, P., Yuan, Y., Liu, D., Huang, Z., Hou, X., Cottrell, G.: Understanding convolution for semantic segmentation. arXiv preprint arXiv:1702.08502 (2017)
38. Graves, A., Fernández, S., Schmidhuber, J.: Multi-dimensional recurrent neural networks. In: "17th International Conference on Artificial Neural Networks – ICANN 2007. ("2007")
39. Badrinarayanan, V., Kendall, A., Cipolla, R.: Segnet: A deep convolutional encoder-decoder architecture for image segmentation. TPAMI (2017)
40. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: MICCAI. (2015)
41. Hariharan, B., Arbeláez, P., Girshick, R., Malik, J.: Hypercolumns for object segmentation and fine-grained localization. In: CVPR. (2015)

42. Dai, J., He, K., Sun, J.: Convolutional feature masking for joint object and stuff segmentation. In: CVPR. (2015)
43. Caesar, H., Uijlings, J., Ferrari, V.: Region-based semantic segmentation with end-to-end training. In: ECCV. (2016)
44. Lin, G., Milan, A., Shen, C., Reid, I.: Refinenet: Multi-path refinement networks for high-resolution semantic segmentation. In: CVPR. (2017)
45. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: CVPR. (2015)
46. Noh, H., Hong, S., Han, B.: Learning deconvolution network for semantic segmentation. In: ICCV. (2015)
47. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR. (2016)
48. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: NIPS. (2012)
49. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: ICML. (2015)
50. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: ICCV. (2015)
51. Neuhold, G., Ollmann, T., Rota Bulò, S., Kontschieder, P.: The mapillary vistas dataset for semantic understanding of street scenes. In: ICCV. (2017)
52. Everingham, M., Van Gool, L., Williams, C.K., Winn, J., Zisserman, A.: The pascal visual object classes (voc) challenge. International journal of computer vision (2010)
53. Hariharan, B., Arbeláez, P., Bourdev, L., Maji, S., Malik, J.: Semantic contours from inverse detectors. In: ICCV. (2011)
54. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In: ECCV. (2014)
55. Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K.: SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and $<$ 0.5 MB model size. arXiv preprint arXiv:1602.07360 (2016)
56. Yasin, A., Ben-Asher, Y., Mendelson, A.: Deep-dive analysis of the data analytics workload in cloudsuite. In: Workload Characterization (IISWC), 2014 IEEE International Symposium on. (2014)
57. Wu, Y., Wang, Y., Pan, Y., Yang, C., Owens, J.D.: Performance characterization of high-level programming models for gpu graph analytics. In: Workload Characterization (IISWC), 2015 IEEE International Symposium on, IEEE (2015) 66–75
58. PyTorch: Tensors and Dynamic neural networks in Python with strong GPU acceleration. http://pytorch.org/ Accessed: 2018-02-08.
59. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. ICLR (2015)
60. NVPROF: CUDA Toolkit Documentation. http://docs.nvidia.com/cuda/profiler-users-guide/index.html Accessed: 2018-02-08.
61. TegraTools: NVIDIA Embedded Computing. https://developer.nvidia.com/embedded/develop/tools Accessed: 2018-02-08.
62. PowerTop: For PowerTOP saving power on IA isn't everything. It is the only thing! https://01.org/powertop/ Accessed: 2018-02-08.
63. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. ICLR (2015)
64. Treml et al.: Speeding up semantic segmentation for autonomous driving. In: MLITS, NIPS Workshop. (2016)
65. Ghiasi, G., Fowlkes, C.C.: Laplacian pyramid reconstruction and refinement for semantic segmentation. In: ECCV. (2016)

66. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: ICML. (2010)
67. Springenberg, J.T., Dosovitskiy, A., Brox, T., Riedmiller, M.: Striving for simplicity: The all convolutional net. arXiv preprint arXiv:1412.6806 (2014)
68. Huang, G., Liu, Z., Weinberger, K.Q., van der Maaten, L.: Densely connected convolutional networks. In: CVPR. (2017)

A Hardware Details

Three machines were used in our experiments. Table 3 summarizes the details about these machines. A computing platform (e.g. Jetson TX2) on an edge device shares the global memory or RAM between CPU and GPU, while laptop and desktop devices have dedicated CPU and GPU memory.

NVIDIA Jetson TX2 can run in different modes. In performance mode (Max-P), all CPU cores are enabled in TX2, while in normal mode (Max-Q mode) only 4 out of 6 CPU cores are active. CPU and GPU clock frequencies are different in these modes and therefore, applications will have different power requirements in different modes.

B The path from ESPNet-A to ESPNet

Different variants of ESPNet are shown in Fig. 8. The first variant, ESPNet-A (Fig. 8a), is a standard network that takes an RGB image as an input and learns representations at different spatial levels using the ESP module to produce a segmentation mask. The second variant, ESPNet-B (Fig. 8b), improves the flow of information inside ESPNet-A by sharing the feature maps between the previous strided ESP module and the previous ESP module. The third variant, ESPNet-C (Fig. 8c), reinforces the input image inside ESPNet-B to further improve the flow of information. These three variants produce outputs whose spatial dimensions are $\frac{1}{8}th$ of the input image. The fourth variant, ESPNet (Fig. 8d), adds a light weight decoder (built using a principle of *reduce-upsample-merge*) to ESPNet-C that outputs the segmentation mask of the same spatial resolution as the input image. The building block functions used to build the ESPNet (from ESPNet-A to ESPNet) are discussed next.

Efficient down-sampling: Recent CNN architectures have used strided convolution (e.g. [67,47,14]) instead of pooling operations (e.g. [63,48]) for down-sampling operations, because it allows the non-linear down-sampling operations to be learned while simultaneously enabling expansion of the network width. Standard strided convolutional operations are expensive; therefore, they are replaced by strided ESP modules for down-sampling. Point-wise convolutions are replaced by $n \times n$ strided convolutions in

		Desktop	Laptop	Edge Device
CPU	**CPU Architecture**	x86_64	x86_64	aarch64
	CPU Cores	8	8	6 (Max-P mode) 4 (Max-Q mode)
	CPU Model Name	Intel(R) Core(TM) i7-6700k @ 4 GHz	Intel(R) Core(TM) i7-6700HQ CPU @ 2.60 GHz	ARMv8 Processor rev 3(v81)
	L1 Cache	32 KB	32 KB	32 KB
	L2 Cache	256 KB	256 KB	2 MB
	L3 Cache	8 MB	6 MB	–
	RAM	16 GB	16 GB	8 GB (shared)
GPU	**GPU Model Name**	TitanX Pascal	GeForce GTX 960M	Tegra X2
	CUDA Driver Version	9.1	9.1	8.0
	Global Memory	12 GB	4 GB	8 GB (shared)
	Max. GPU frequency	1.53 GHz	1.18 GHz	1.3 GHz (Max-P mode)) 824 MHz (Max-Q mode)
	CUDA Cores	3584	640	256
	Streaming multiprocessors (SM)	28	5	2
	CUDA Cores per SM	128	128	128

Table 3: This table summarizes the hardware that we used in our experiments.

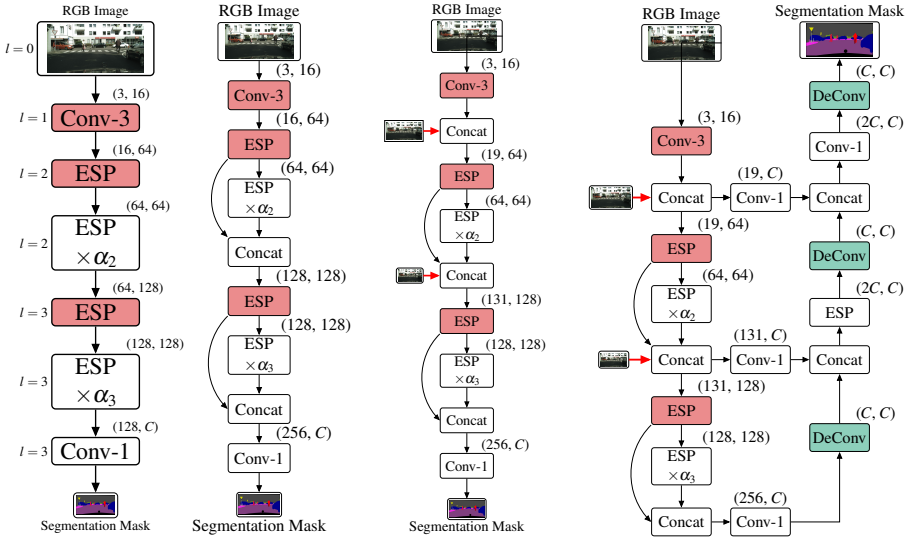


Fig. 8: The path from ESPNet-A to ESPNet. Red and green color boxes represent the modules responsible for down-sampling and up-sampling operations, respectively. Spatial-level l is indicated on the left of every module in (a). We denote each module as (# input channels, # output channels). Here, Conv-n represents $n \times n$ convolution. This figure is the same as Fig. 4.

the ESP module for learning non-linear down-sampling operations. The spatial dimensions of the feature maps are changed by down-sampling operations. Following [47,68], we do not combine the input and output feature maps using the skip-connection during down-sampling operations. The number of parameters learned by strided convolution and strided ESP are n^2MN and $\frac{n^2MN}{K} + \left(\frac{n^2N^2}{K^2} \cdot K\right)$, respectively. By expressing strided convolution as strided ESP for down-sampling, the number of parameters required is reduced by a factor of $\frac{KM}{M+N}$ and the effective receptive field is increased by $\sim [2^{K-1}]^2$ times. We will refer to this network as ESPNet-A (Fig. 8a).

Network width expansion: To maintain the computational complexity at each spatial level, traditional CNNs (e.g. [63,47,14]) double the width of the network after every down-sampling operation, usually using a convolution operation. Following [68], we concatenate the feature maps received from the previous strided ESP module and the previous ESP module to increase the width of the network, as shown in Fig. 8b with a curved arrow. The concatenation operation establishes a long-range connection between the input and output at the same spatial level and, therefore, improves the flow of information inside the network. We will refer to this network as ESPNet-B (Fig. 8b).

Input reinforcement: Spatial information is lost due to down-sampling and convolutional operations. To compensate, we reinforce the input image inside the network. We down-sample the input-image and concatenate it with the feature maps from the previous strided ESP module and the previous ESP module. We will refer to ESPNet-B with input reinforcement as ESPNet-C (Fig. 8c). Since the input RGB image has only 3 channels, the increase in network complexity due to input reinforcement is minimal.

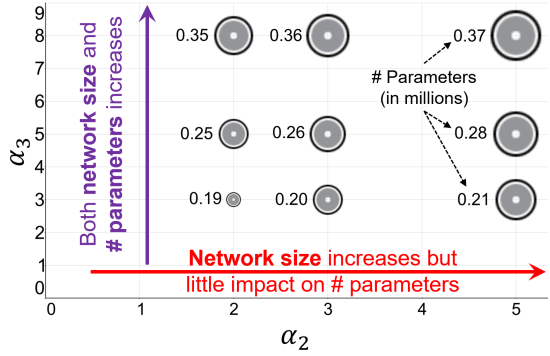


Fig. 9: Relationship between depth multipliers α_2 and α_3 for creating efficient networks. Here, circle size $\propto$ network size.

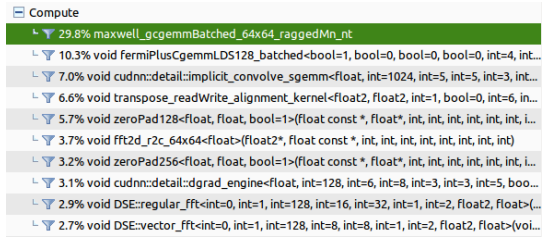
Depth multiplier α: To build deeper computationally efficient networks for edge devices without changing the network topology, we introduce a hyper-parameter α to control the depth of the network. This parameter, α, repeats the ESP module α_l times at spatial level l. CNNs require more memory at higher spatial levels i.e. at $l = 0$ and $l = 1$ because of the high spatial dimensions of feature maps at these levels. To be memory efficient, we do not repeat ESP or convolutional modules at these spatial levels.

As we change the values of these parameters, the amount of computational resources required by a network will change. Fig. 9 shows the impact of $\alpha_l, l = \{2,3\}$ on the network parameters and its size. As we increase α_2, the network size increases with little impact on the number of parameters. When we increase α_3, both the network size and number of parameters increase. Both the number of parameters and network size should increase with depth [47,14,13]. Therefore, for creating deep and efficient ESPNet networks, we fix the value of α_2 and vary the value of α_3.

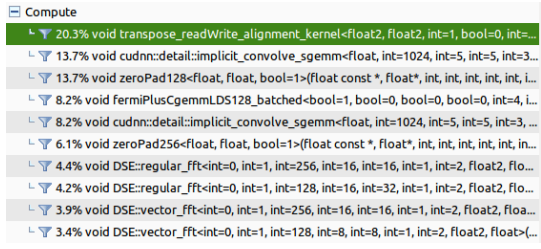
RUM for efficient decoding: The spatial resolution of the output produced by ESPNet-C is $\frac{1}{8}th$ of the input image size. Up-sampling the feature maps directly, say using bilinear interpolation, may give good accuracy on a standard metric, but the output is usually coarse [45]. We adopt a bottom-up approach (e.g. [39,40]) to aggregate the multi-level information learned by ESPNet-C using a simple rule: *Reduce-Upsample-Merge* (RUM). *Reduce:* The feature map from spatial levels l and $l-1$ are projected to a C-dimensional space, where C represents the number of classes in the dataset. *Upsample:* The reduced feature map from spatial level l is upsampled by a factor of 2 using a 2×2 deconvolutional kernel so that it has the same spatial dimensions as that of the feature map at level $l-1$. *Merge:* The up-sampled feature map from level l is then combined with the C-dimensional feature map from level $l-1$ using a concatenation operation. This process is repeated until the spatial dimensions of the feature map are the same as the input image. We refer to this network as ESPNet (Fig. 8d).

C Top-10 Kernels in ESPNet, ENet, and ERFNet

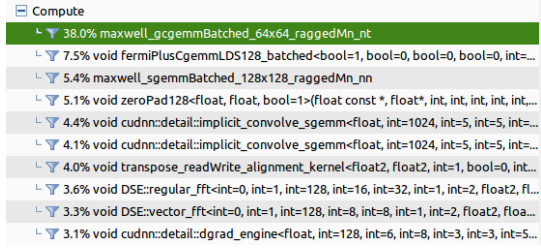
Convolutional operations are implemented using a highly optimized general matrix multiplication (GEMM) operations and memory re-ordering operations such as im2col. For fast and efficient networks, the kernel corresponding to GEMM operations should have high contribution towards compute resource utilization. Figure 10 visualizes the



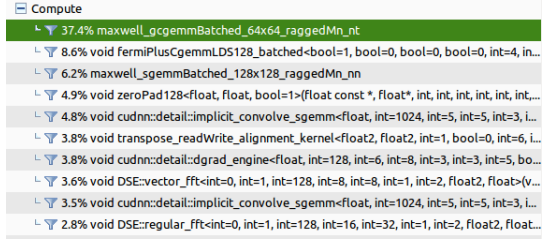
(a) ENet



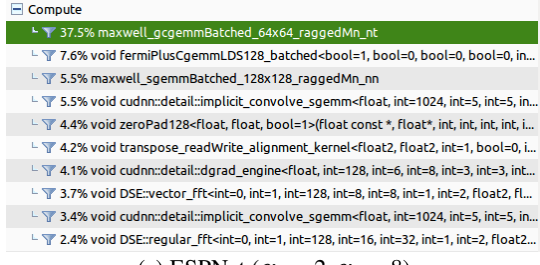
(b) ERFNet



(c) ESPNet ($\alpha_2 = 2, \alpha_3 = 3$)



(d) ESPNet ($\alpha_2 = 2, \alpha_3 = 5$)



(e) ESPNet ($\alpha_2 = 2, \alpha_3 = 8$)

Fig. 10: This figure visualizes the top-10 kernels along with their contribution towards compute resource utilization. The top-1 kernel is highlighted in green color.

top-10 kernels executed by ENet, ERFNet, and ESPNet. We can see that the top-1 kernel in ESPNet is GEMM, and it is responsible for about 38% of the total computational time. Since convolution operations are implemented using the GEMM kernel, this suggest that ESPNet utilizes the limited computational resources available in TX2 efficiently. Similarly, the top-1 kernel in ENet is also GEMM; however, the contribution of this kernel towards computing is not as high as ESPNet. This is why the sensitivity of ENet towards GPU frequency is low and runs 1.27× slower on NVIDIA TitanX than ESPNet while running at almost the same rate on the NVIDIA TX2. On the other hand, the top-1 kernel in ERFNet is the memory alignment kernel. This suggests that ERFNet gets bottlenecked by the memory operations.

D Image Size vs. Inference Speed

Figure 11 summarizes the impact of image size on the inference speed. At smaller image resolutions (224x224 and 640x360), ESPNet is faster than ENet and ERFNet. However, ESPNet delivers a similar inference speed to ENet for-high resolution images. We presume that ESPNet is bottlenecked by the limited and shared resources on the TX2 device. We note that ESPNet processes high resolution images faster than ENet on high-end devices, such as laptop and desktop.

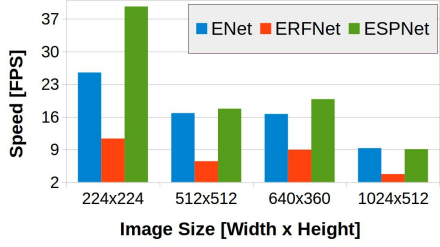


Fig. 11: The impact of image size on the inference speed on an edge device

E Resource Utilization Plots for ENet, ERFNet, and ESPNet

Figures 12, 13, and 14 show the utilization of TX2 resources (CPU, GPU, and memory) over time for ENet, ERFNet and ESPNet. The data were collected using Tegrastats in *Max-Q* mode. These networks are throughput intensive, and therefore, GPU utilization rates are high while CPU utilization rates are low for these networks. Note that the average CPU utilization rate is below 25%; suggesting that these networks are using only one CPU core out of the available four CPU cores and can be bound to a single CPU core for better utilization of CPU resources, if running additional applications on TX2. Memory utilization rates are significantly different for these networks. The memory footprint of ESPNet is low in comparison to ENet and ERFNet, suggesting ESPNet is suitable for memory constrained devices.

Recall that ESPNet with $\alpha_2 = 2$ and $\alpha_3 = 8$ learns the same number of parameters as ENet. However, ESPNet has a low memory footprint than ENet (Fig. 14); suggesting ESPNet is more memory efficient and utilizes the shared memory efficiently.

F Results on the Cityscape and the Mapillary Dataset

A summary of class-wise and category-wise results on the Cityscape [6] dataset was given in Table 4, while category-wise results on the Mapillary [51] dataset were given in Table 5. Though ERFNet outperformed ENet and ESPNet on every class, it performed badly on the Mapillary dataset. In particular, ERFNet struggled classifying simple classes, such as sky, on the Mapillary dataset, while on such classes, ENet and ESPNet performed relatively well. We note that ESPNet learns good generalization representations about the objects and performs well, even in the wild. Qualitative results on the Cityscape and Mapillary dataset were given in Figure 16 and Figure 17, respectively.

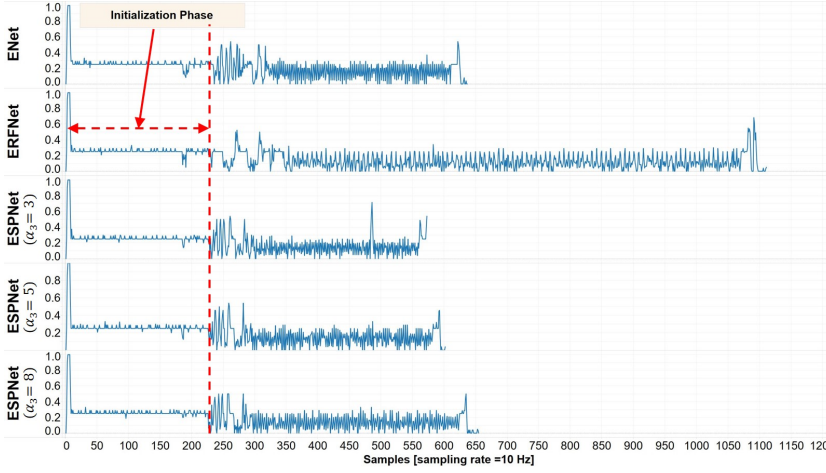


Fig. 12: This figure compares the **CPU utilization** rates on NVIDIA Jetson TX2. For ESPNet, we used $\alpha_2 = 2$. Here, 1.0 represents 100% CPU utilization.

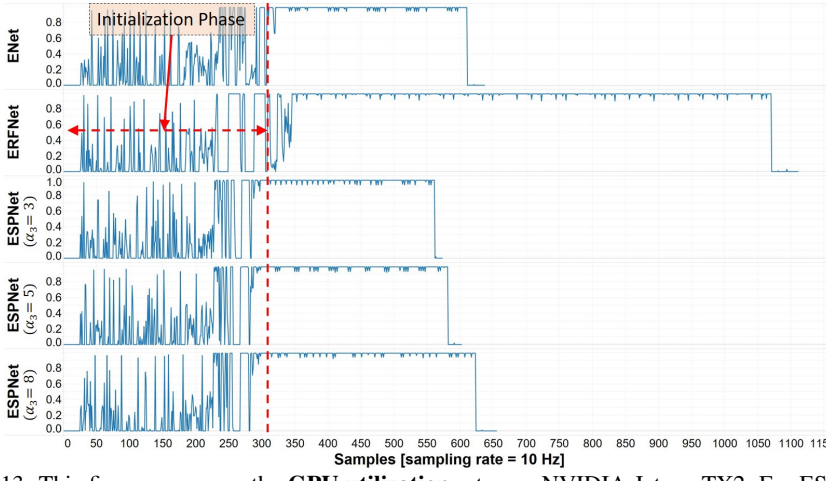


Fig. 13: This figure compares the **GPU utilization** rates on NVIDIA Jetson TX2. For ESPNet, we used $\alpha_2 = 2$. Here, 1.0 represents 100% GPU utilization.

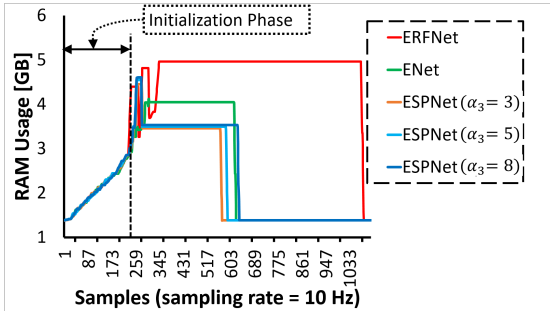


Fig. 14: This figure compares the **memory utilization** on NVIDIA Jetson TX2. For ESPNet, we used $\alpha_2 = 2$. Maximum available memory on TX2 is 8 GB and is *shared* between CPU and GPU.

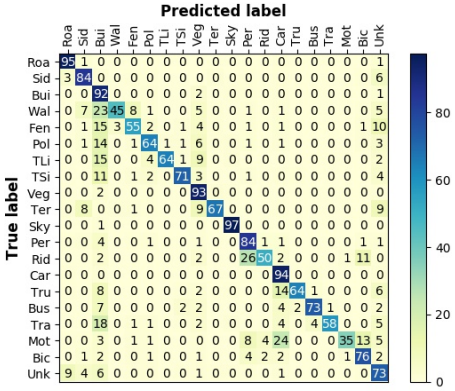


Fig. 15: ESPNet's (with $\alpha_2 = 2$ and $\alpha_3 = 8$) confusion matrix on the Cityscape *validation* set. ESPNet makes some mistakes between classes that belong to the same category, and hence has lower class-wise accuracy. However, ESPNet delivers a good category-wise accuracy. Here, the class names were represented by the first three characters of a word. For class names with two words, the first character from the first word and the first two characters from the second word were used to represent the class name. Here, Unk denotes the unknown class.

Network	mIOU	Roa	Sid	Bui	Wal	Fen	Pol	TLi	TSi	Veg	Ter	Sky	Per	Rid	Car	Tru	Bus	Tra	Mot	Bic
ENet [20]	58.29	96.33	74.24	85.05	32.16	33.23	43.45	34.10	44.02	88.61	61.39	90.64	65.51	38.43	90.60	36.90	50.51	48.08	38.80	55.41
ERFNet [21]	**68.02**	97.74	80.99	89.83	42.46	47.99	56.25	59.84	65.28	91.38	68.20	94.19	76.75	57.08	92.76	50.77	60.09	51.80	47.27	61.65
ESPNet (Ours)	60.34	95.68	73.29	86.60	32.79	36.43	47.06	46.92	55.41	89.83	65.96	92.47	68.48	45.84	89.90	40.00	47.73	40.70	36.40	54.89

(a) Class-wise comparison on the *test* set

Network	mIOU	Flat	Nature	Object	Sky	Construction	Human	Vehicle
ENet [20]	80.40	97.34	88.28	46.75	90.64	85.40	65.50	88.87
ERFNet [21]	**86.46**	98.18	91.12	62.42	94.19	90.06	77.43	91.87
ESPNet (Ours)	82.18	95.49	89.46	52.94	92.47	86.67	69.76	88.45

(b) Category-wise comparison on the *test* set

Table 4: Comparison on the Cityscape dataset. For comparison with other networks, please see the **Cityscape leader-board**: https://www.cityscapes-dataset.com/benchmarks/.

Network	mIOU	Flat	Nature	Object	Sky	Construction	Human	Vehicle
ENet [20]	0.33	0.61	0.57	0.16	0.37	0.35	0.08	0.20
ERFNet [21]	0.25	0.73	0.29	0.16	0.03	0.23	0.06	0.24
ESPNet (Ours)	**0.40**	0.66	0.69	0.20	0.52	0.32	0.16	0.21

Table 5: Category-wise comparison on the Mapillary *validation* set. ESPNet learned generalizable representations of objects and outperformed both ENet and ERFNet in the wild.

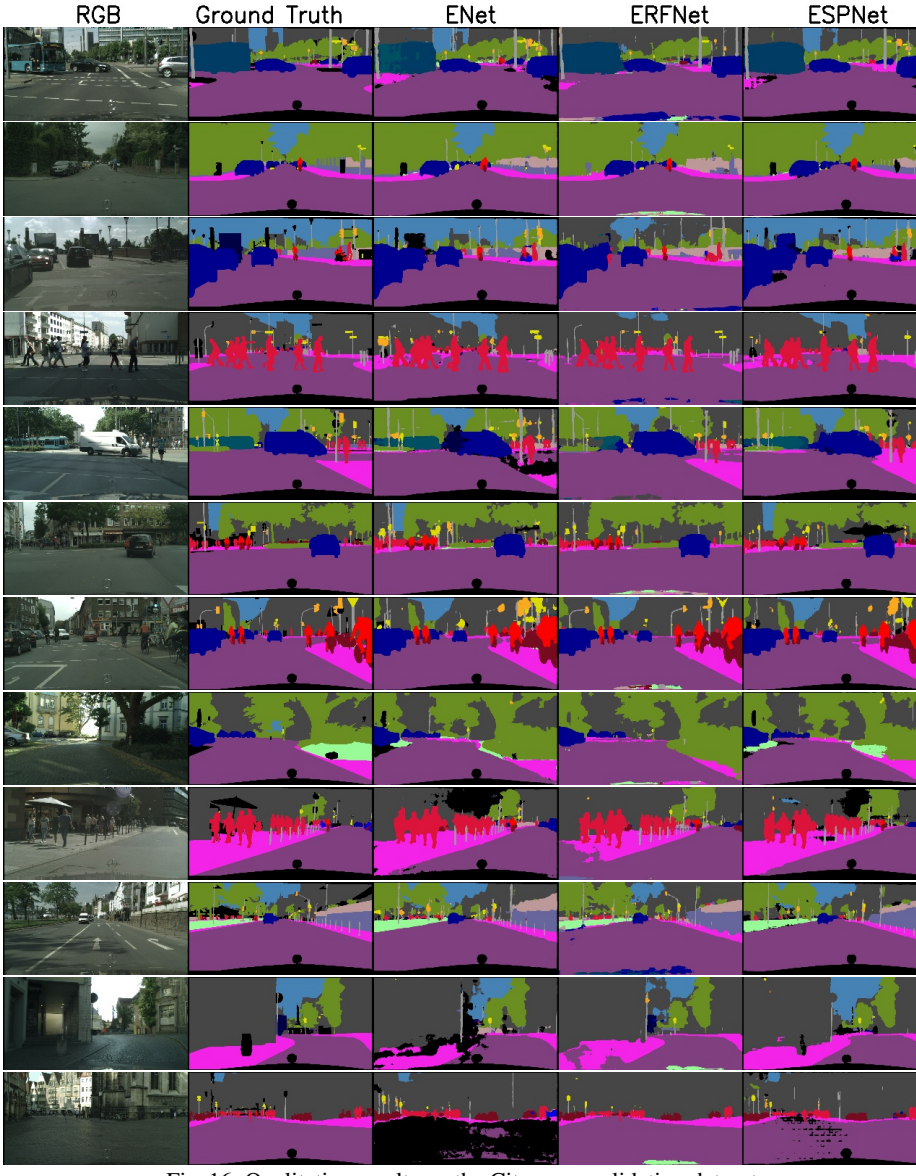


Fig. 16: Qualitative results on the Cityscape validation dataset.

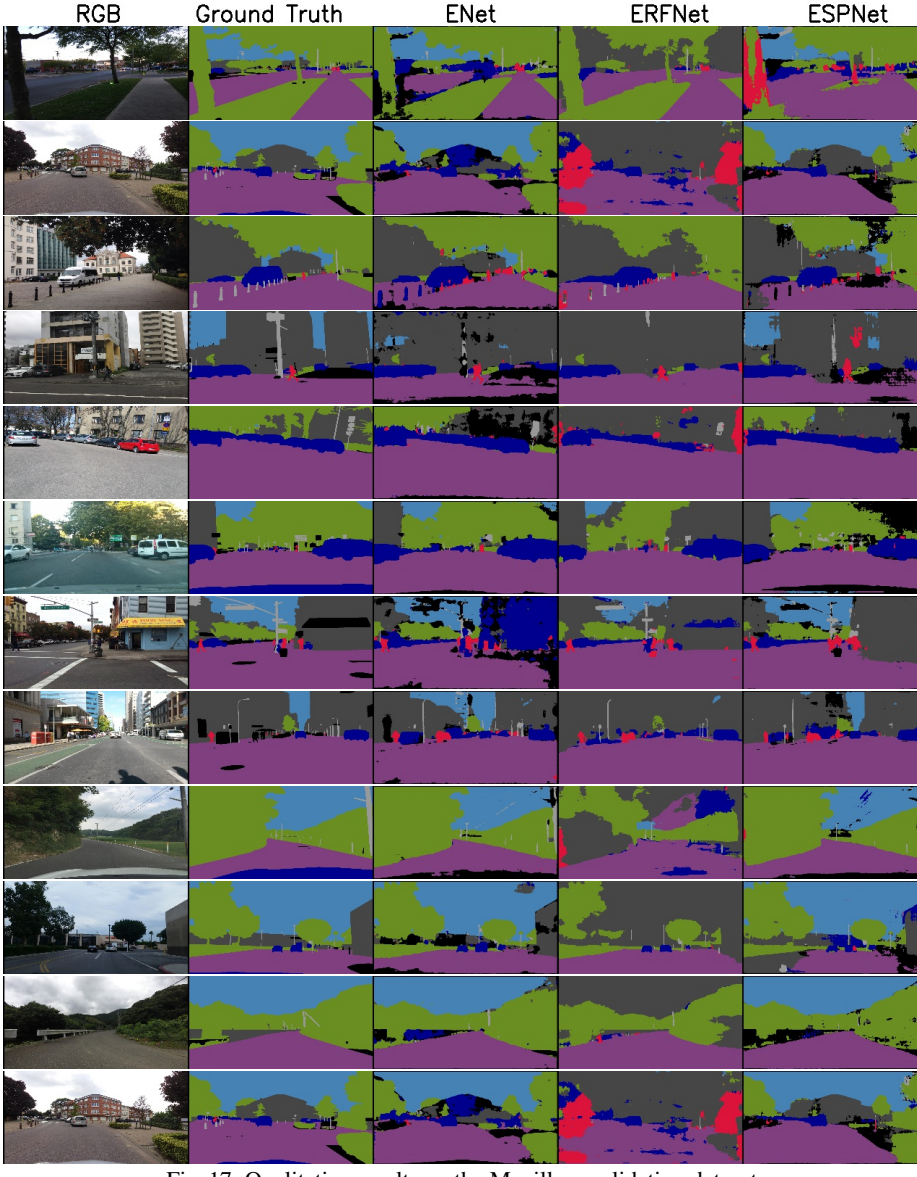


Fig. 17: Qualitative results on the Mapillary validation dataset.