International Journal of Computer Vision (2023) 131:2153–2171
https://doi.org/10.1007/s11263-023-01801-1

Lightweight and Progressively-Scalable Networks for Semantic Segmentation

Yiheng Zhang[1] · **Ting Yao**[1] · **Zhaofan Qiu**[1] · **Tao Mei**[1]

Received: 27 July 2022 / Accepted: 17 April 2023 / Published online: 18 May 2023

Abstract
Multi-scale learning frameworks have been regarded as a capable class of models to boost semantic segmentation. The problem nevertheless is not trivial especially for the real-world deployments, which often demand high efficiency in inference latency. In this paper, we thoroughly analyze the design of convolutional blocks (the type of convolutions and the number of channels in convolutions), and the ways of interactions across multiple scales, all from lightweight standpoint for semantic segmentation. With such in-depth comparisons, we conclude three principles, and accordingly devise Lightweight and Progressively-Scalable Networks (LPS-Net) that novelly expands the network complexity in a greedy manner. Technically, LPS-Net first capitalizes on the principles to build a tiny network. Then, LPS-Net progressively scales the tiny network to larger ones by expanding a single dimension (the number of convolutional blocks, the number of channels, or the input resolution) at one time to meet the best speed/accuracy tradeoff. Extensive experiments conducted on three datasets consistently demonstrate the superiority of LPS-Net over several efficient semantic segmentation methods. More remarkably, our LPS-Net achieves 73.4% mIoU on Cityscapes test set, with the speed of 413.5FPS on an NVIDIA GTX 1080Ti, leading to a performance improvement by 1.5% and a 65% speed-up against the state-of-the-art STDC. Code is available at https://github.com/YihengZhang-CV/LPS-Net.

Keywords Convolutional neural networks · Semantic segmentation · Lightweight · Scalable

1 Introduction

Semantic segmentation is to assign semantic labels to every pixel of an image or a video frame. With the development of deep neural networks, the state-of-the-art networks have successfully pushed the limits of semantic segmentation with remarkable performance improvements. For example, DeepLabV3+ (Chen et al., 2018c) and Hierarchical Multi-Scale Attention (Tao et al., 2020) achieve 82.1% and 85.4% mIoU on Cityscapes test set, which are almost saturated on that dataset. The recipe behind these successes originates from multi-scale learning. In the literature, the recent advances involve utilization of multi-scale learning for semantic segmentation along three different dimensions: U-shape (Chen et al., 2020; Peng et al., 2017), pyramid pooling (Chen et al., 2018c; Zhao et al., 2017), and multi-path framework (Chen et al., 2016; Tao et al., 2020). The U-shape structure hierarchically fuses the features to gradually increase the spatial resolution and naturally produce multi-scale features. The pyramid pooling methods delve into multi-scale information through executing spatial or atrous spatial pyramid pooling at multiple scales. Unlike the former two research schemes, the multi-path frameworks resize the input image to multiple resolutions or scales and feed each scale into an individual path of a deep network. By doing so, the multi-path design places the input resolutions from high to low in parallel and explicitly maintains the high-resolution information rather than recovering from low-scale feature maps. As a result, the learnt features are potentially more capable of classifying and localizing each pixel.

We employ this elegant recipe of multi-path framework and further evolve such type of architectures with good

Communicated by Jifeng Dai.

✉ Ting Yao
tingyao.ustc@gmail.com

Yiheng Zhang
yihengzhang.chn@gmail.com

Zhaofan Qiu
zhaofanqiu@gmail.com

Tao Mei
tmei@live.com

[1] JD Explore Academy, Beijing, China

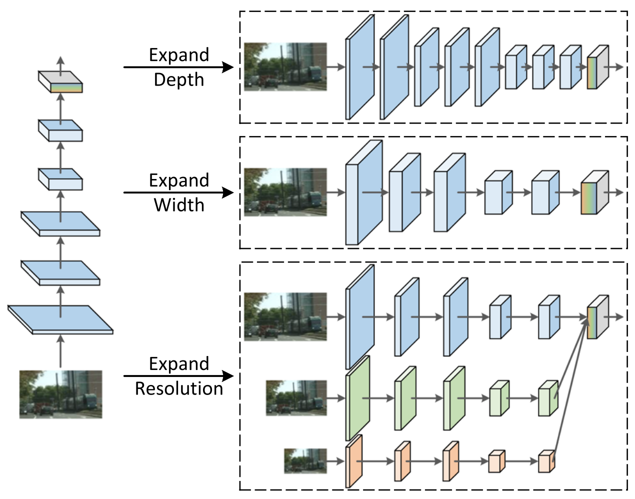


Fig. 1 LPS-Net executes the growth of network complexity through expanding a single dimension of the number of convolutional blocks (depth), the number of channels (width), or the input resolution (resolution) at one time

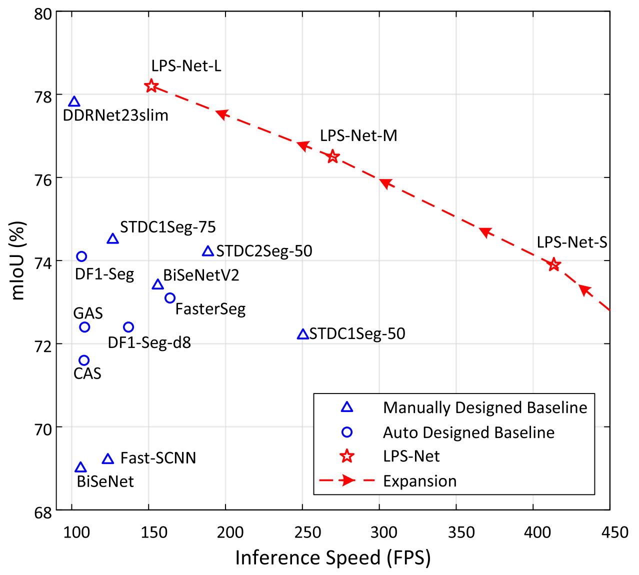


Fig. 2 Comparisons of inference speed/accuracy tradeoff on Cityscapes validation set. LPS-Net (-S, -M, and -L) which are progressively expanded along multiple dimensions demonstrate a good balance between accuracy and inference speed compared to other manually-/auto-deigned models

accuracy/speed tradeoff for semantic segmentation. Our philosophies are from two perspectives: (1) lightweighting computational units for semantic segmentation, and (2) progressively scaling up the network while balancing accuracy and inference latency. We propose to explore the first problem by probing the basic unit of convolutional blocks, including the type of convolutions and the number of channels in convolutions, on the basis of several uniqueness (e.g., large feature maps, thin channel widths) in efficient semantic segmentation. Moreover, we further elaborate different ways of interaction across multiple paths with respect to the accuracy/speed tradeoff. Based on these lightweight practice, we build a tiny model, and then progressively expand the tiny model along multiple possible dimensions and select a single dimension that achieves the best tradeoff in each step to alleviate the second issue of accuracy/speed balance.

To materialize the idea, we present Lightweight and Progressively-Scalable Networks (LPS-Net) for efficient semantic segmentation. Specifically, LPS-Net bases the multi-path design upon the low-latency regime. Each path takes the resized image as the input to an individual network, which consists of stacked convolutional blocks. The networks across paths share the same structure but have independent learnable weights. The outputs from all the paths are aggregated to produce the score maps, which are upsampled via bilinear upsampling for pixel-level predictions. In an effort to achieve a lightweight and efficient architecture, we look into the basic unit of convolutional blocks and empirically suggest to purely use 3×3 convolutions with 2^n-divisible channel widths. Furthermore, we capitalize on a simple yet effective way of bilinear interpolation to encourage mutual exchange and interactions between paths. With these practical guidelines, LPS-Net first builds a tiny network and then scales the tiny network to a family of larger ones in a progressive manner. Technically, LPS-Net executes the growth of network complexity through expanding a single dimension of the number of convolutional blocks, the number of channels, or the input resolution at one time, as depicted in Fig. 1. In our case, LPS-Net ensures a good balance between accuracy and inference speed during expansion, and shows the superiority over other manually-/auto-deigned models, as shown in Fig. 2.

In summary, we have made the following contributions: (1) The lightweight design of convolutional blocks and the way of path interactions in multi-path framework are shown capable of regarding as the practical principles for efficient semantic segmentation; (2) The exquisitely devised LPS-Net is shown able to progressively expand the network complexity while striking the right accuracy-efficiency tradeoff; (3) LPS-Net has been properly verified through extensive experiments over three datasets, and superior capability is observed on both NVIDIA GPUs and embedded devices in our experiments.

2 Related Works

2.1 Semantic Segmentation

With the success of convolutional neural networks (CNNs), FCN (Long et al., 2015) which deploys CNN in a fully convolutional manner enables dense semantic prediction and end-to-end training for semantic segmentation. Following FCN, researchers propose various techniques, which achieve remarkable performances. For example, in order to extract multi-scale context information which is known to be critical for pixel labeling tasks (Ladický et al., 2009), PSPNet (Zhao et al., 2017) applies spatial pyramid pooling and DeepLab (Chen et al., 2018b, c) utilizes parallel dilated convolutions with different rates. Yuan et al. (2021) propose to use object context represented as a dense relation matrix to emphasize the contribution of object information. Multi-layer feature aggregation (Ghiasi & Fowlkes, 2016; Peng et al., 2017; Qiu et al., 2017; Fu et al., 2019; Lin et al., 2019; Nirkin et al., 2021) is performed to recover the spatial information loss caused by spatial reduction in the network. Chen et al. (2016) and Tao et al. (2020) capitalize on the multi-path framework to improve the predictions. Benefiting from neural architecture search (NAS) (Zoph & Le, 2017; Zoph et al., 2018; Liu et al., 2019b; Qiu et al., 2019) which is introduced to automatically generate optimal neural networks, research works can search dense prediction cell (Chen et al., 2018a), hierarchical network architectures (Liu et al., 2019a), and densely connected neural architectures (Zhang et al., 2021) for semantic segmentation. Such efforts are made to achieve high-quality segmentation but without taking inference latency into account.

2.2 Lightweight Networks

The real-world deployments often demand accurate and efficient networks. To accelerate the inference, there have been several techniques, e.g., pruning (Han et al., 2016), factorization (Szegedy et al., 2016), depthwise separable convolution (Chollet, 2017; Howard et al., 2017), group convolution (Krizhevsky et al., 2012), and quantization (Zhang et al., 2019a) being proposed in the literature. NAS (Ding et al., 2021a; Wu et al., 2021a) is also employed to optimize the tradeoff between performance and FLOPs for multiple vision tasks. To facilitate model deployments and speed up the inference for semantic segmentation, recent works present to manually (Fan et al., 2021; Li et al., 2019a, 2020b) or automatically (Chen et al., 2020; Li et al., 2020a, 2019b; Lin et al., 2020; Zhang et al., 2019b) design lightweight networks. ICNet (Zhao et al., 2018) employs a cascade network structure to achieve real-time segmentation. BiSeNet (Yu et al., 2021a, 2018a) treats the spatial details and category semantics of the images separately to obtain a lightweight network. DFANet (Li et al., 2019) designs a feature reuse method to incorporate multi-level context into encoded features. SFNet (Li et al., 2020b, 2022) boosts the segmentation by broadcasting high-level features to high-resolution features of backbone networks with cheap operations. Fan et al. (2021) propose a short-term dense concatenate module to enrich features with scalable receptive field and multi-scale information in a lightweight network. Lite-HRNet (Yu et al., 2021b) utilizes the high-resolution design pattern of HRNet (Wang et al., 2021) and introduces conditional channel weighting to replace costly pointwise (1×1) convolutions in shuffle blocks (Zhang et al., 2018). BiAlignNet (Wu et al., 2021b) augments the BiSeNet by employing a gated flow alignment module to align features in a bidirectional way. To automate lightweight network design, CAS (Zhang et al., 2019b) presents resource constraints to achieve an accuracy/computation tradeoff and GAS (Lin et al., 2020) further integrates a graph convolution network as a communication mechanism between different blocks. Li et al. (2019b) prunes the search space with a partial order assumption to search a balanced network. Chen et al. (2020) forms a search space integrating multi-resolution branches and calibrates the balance between accuracy and latency by an additional regularization. AutoRTNet (Sun et al., 2021) jointly optimizes network depth, downsampling strategies, and the way of feature aggregation to obtain real-time segmentation networks.

Despite having such lightweight networks, the way to balance speed and accuracy for efficient semantic segmentation is not fully explored. Unlike the works like (Zhao et al., 2018; Yu et al., 2021b) which model different resolutions with standard Residual Conv blocks, our work aims at probing the basic design of Conv blocks, including the type of convolutions and the number of channels, particularly for efficient semantic segmentation. Instead of pre-fixing two-path design in networks like BiSeNet (Yu et al., 2021a, 2018a) and BiAlignNet (Wu et al., 2021b), we exquisitely devise a progressive expansion paradigm to scale up the network. Such expansion dynamically balances the depth, width, resolution, and the number of paths of the networks to strike the right accuracy-efficiency tradeoff.

2.3 Summary

Our work focuses on developing light-weight and scalable networks for efficient semantic segmentation upon the low-latency regime. The proposal of LPS-Net contributes by studying not only the practical design of convolutional blocks and the way of path interactions in multi-path framework for semantic segmentation, but also how a scalable multi-path network can be nicely expanded to meet the right accuracy/speed balance.

3 LPS-Net

We proceed to present our core proposal, i.e., the Lightweight and Progressively-Scalable Networks (LPS-Net). Specifically, we first introduce the macro architecture in LPS-Net that employs the multi-path recipe. Then, three design principles are presented to upgrade this architecture from

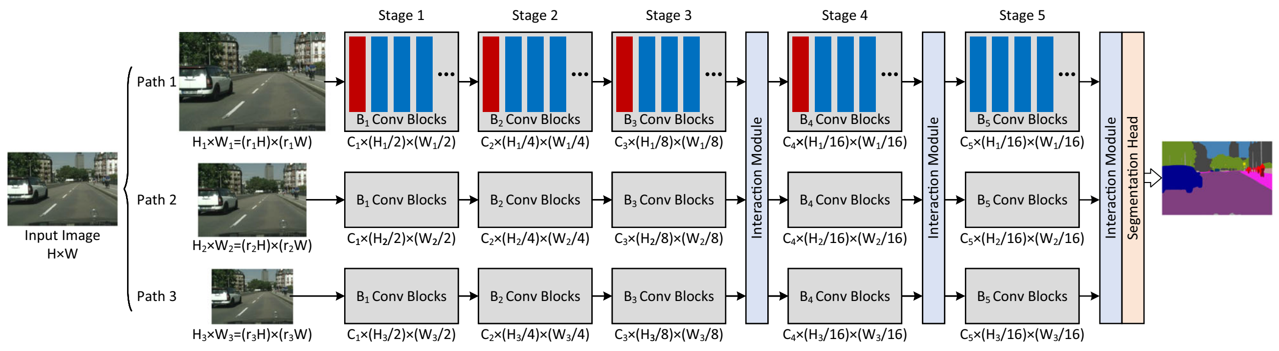


Fig. 3 An overview of the design space in our proposed LPS-Net. The red block and blue block represent the convolutional block with stride=2 and stride=1, respectively. The size of output feature map is given for each stage. Please note that the 3-path network here is an example to depict the macro architecture, and the number of paths in LPS-Net is dynamically determined by the parameter $\mathcal{R} = \{r_i|_{i=1}^N\}$ which is optimized for the right tradeoff along the network expansion

lightweight standpoint, including the type of convolutions, the number of channels in convolution, and the way of interaction across multiple paths. Based on these practical guidelines, a family of scalable LPS-Net is devised in a progressive manner by expanding a single dimension at one time to seek the best speed/accuracy tradeoff.

3.1 Macro Architecture

Figure 3 depicts an overview of the macro architecture in our LPS-Net. The macro architecture is basically constructed in the multi-path design, which resizes the input image to multiple scales and feeds each scale into an individual path. Formally, given the input image with resolution $H \times W$, LPS-Net with N paths (e.g., $N = 3$ in Fig. 3) resizes the image to $r_i H \times r_i W$, which acts as the input of the i-th path. Here r_i denotes the scaling ratio. Each path is implemented as a stack of convolutional blocks in five stages, and the j-th stage contains B_j blocks. The number of channels (i.e., channel width) are maintained within each stage. Similar to (Chen et al., 2018c), the first convolutional block in Stage 1∼4 reduces the spatial dimension by a factor of two. By doing so, the resolution of output feature map from the last stage is $\frac{r_i H}{16} \times \frac{r_i W}{16}$. Moreover, we place interaction modules at the end of Stage 3∼5, aiming to encourage the mutual exchange and interactions between paths. The outputs of all paths are aggregated and fed into a segmentation head to produce the score maps with *num_classes* channels. Finally, we perform bilinear upsampling over the score maps, yielding the outputs with resolution $H \times W$ that exactly matches the input resolution. Please note that the scaling ratios of paths $\mathcal{R} = \{r_i|_{i=1}^N\}$, the repeated number of convolutional blocks $\mathcal{B} = \{B_j|_{j=1}^5\}$, and the number of channels $\mathcal{C} = \{C_j|_{j=1}^5\}$ in this macro architecture can be flexibly set to make the network structure tailored to the target inference time and adjust the scalability of LPS-Net.

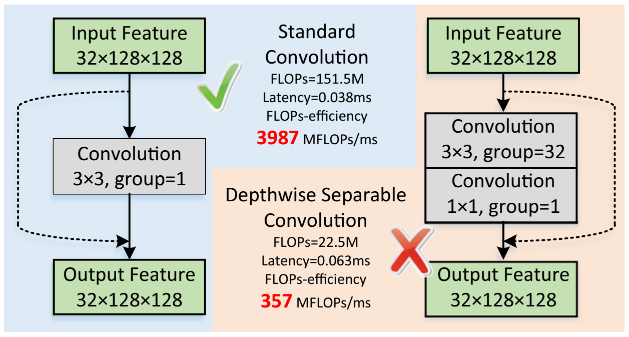


Fig. 4 A comparison between standard convolution and depthwise separable convolution. The dash line is a optional skip connection and the directed acyclic graph of depthwise separable convolution is a standard version. The shape of input/output feature map is $32 \times 128 \times 128$, and the activation layers are omitted in the figure

3.2 Convolutional Block

Convolutional block is the basic computing unit in Convolutional Neural Networks. The stack of convolutional blocks naturally consumes a high percentage of inference time throughout the entire network. Therefore, lightweighting convolutional block is essential for an efficient network. In this section, we delve into the design of basic computational block in LPS-Net along two dimensions: (1) the type of convolutional block, and (2) the number of channels.

3.2.1 The Type of Convolutional Block

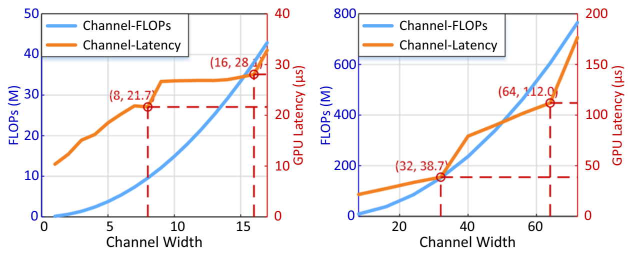
In pursuit of less computational complexity, a series of innovations have been proposed to remould convolutional blocks (Han et al., 2020; He et al., 2016; Howard et al., 2017; Zhang et al., 2018). In these works, the amount of floating-point operations (FLOPs) is often utilized as the measurement of computational complexity, which guides the design of lightweight network for image recognition.

The FLOPs metric measures the computational complexity of networks in a device-irrelevant manner, and is generally exploited to theoretically analyze the network complexity, with the assumption that the network has been well implemented. However, FLOPs ignores the memory access cost (MAC) and the degree of parallelism that substantially impact the inference latency of networks, thereby resulting in the discrepancy between FLOPs and actual latency. The latency metric is directly measured on devices and is more applicable to model deployment in real-world applications, which expect the networks to be actually (not theoretically) efficient. That motivates us to utilize the latency metric for searching lightweight networks. More importantly, the extension of lightweight designs from image recognition to semantic segmentation is not trivial due to several uniqueness in efficient semantic segmentation (e.g., larger feature maps and thin channel widths).

To this end, we perform a comparison between the standard convolution and the widely-adopted depthwise separable convolution (SepConv) (Chollet, 2017; Howard et al., 2017; Tan & Le, 2019) in Fig. 4. Here we set the shape of input feature map as 32 × 128 × 128, which is the most common shape in lightweight networks for semantic segmentation. The latency is evaluated by executing the block on a PC (i7-8700K/16GB RAM) with an 1070Ti GPU. The FLOPs-efficiency, which denotes the number of floating-point operations that is processed in per unit of run time, is utilized as the efficiency metric. This metric is computed by dividing the FLOPs of a network by its inference latency to estimate if the network is device-friendly or well-implemented on the given devices. A high FLOPs-efficiency indicates that the numerical computation of the network is efficiently processed by the device, i.e., device-friendly. As shown in Fig. 4, the FLOPs-efficiency of standard convolution (3987 MFLOPs/ms) is about 10× more than that of SepConv (357 MFLOPs/ms). Accordingly, we utilize the standard convolution as the building block in LPS-Net.

3.2.2 The Number of Channels

The channel width is another important factor that influences the FLOPs-efficiency of convolutions. Theoretically, the computational complexity necessitates quadratic progression with the increase of channel width. However, due to the highly-optimized software (e.g., BLAS, cuDNN) and hardware (e.g., CUDA Core, Tensor Core) in modern libraries/devices, the relation between defacto inference time and channel width in practice becomes fuzzy. To analyze this relation, we compare FLOPs and latency of 3 × 3 convolutions with different channel widths in Fig. 5. As expected, FLOPs grows quadratically when increasing the channel width from 1 to 17 (see the blue curve in Fig. 5a). Meanwhile, the latency only linearly grows and reaches plateaus at around 8 and 16 channel widths (see the red curve in Fig. 5a). Such setups of channel width with high FLOPs-efficiency are regarded as the "sweet spots" for convolutions. Similarly, high FLOPs-efficiencies are attained at 32 and 64 channel widths in Fig. 5b. These sweet points are mainly attributed to the highly parallelizable implementation of convolutions with 2^n-divisible channel width. As a result, we make the channel width of the convolution in LPS-Net to be 2^n-divisible with n as large as possible.



(a) Channel width from 1 to 17, interval 1. (b) Channel width from 8 to 72, interval 8.

Fig. 5 Efficiency comparisons of 3 × 3 convolutions with channel width from 1 to 17, interval 1 (**a**), and 8 to 72, interval 8 (**b**). The spatial resolution of input feature map is fixed as 128 × 128 and the latencies are measured on an NVIDIA GTX 1070Ti GPU with TensorRT

3.3 Multi-Path Interaction

In an effort to elegantly diffuse the complementary information across multiple scales, we design the multi-path interaction module that encourages mutual exchange and interactions between pathes in our LPS-Net. Concretely, we define the interaction module $\mathcal{I}$ as $\{x_h^{out}, x_l^{out}\} = \mathcal{I}(\{x_h^{in}, x_l^{in}\})$, where $\{x_h^{in}, x_l^{in}\}$ and $\{x_h^{out}, x_l^{out}\}$ denote the input pair and output pair, respectively. The input pair consists of a high-resolution feature map $x_h^{in} \in \mathbb{R}^{C\times H_h\times W_h}$ and a low-resolution feature map $x_l^{in} \in \mathbb{R}^{C\times H_l\times W_l}$, where $H_h \geq H_l$ and $W_h \geq W_l$. The resolution of each path remains unchanged during the interaction.

Here, we study six variants of interaction modules which can be grouped into three categories as shown in Fig. 6). For the first category (*Direct-A/B*), the transformed low-resolution features are directly integrated into the high-resolution ones through element-wise summation or channel-wise concatenation. Rather than simply combining the features from two paths, the second category employs attention mechanism to boost the multi-path interaction, namely *Attention-A/B*. Specifically, in *Attention-A*, an attention map $Att(x_l^{in})$ is first calculated from x_l^{in}, and we apply it over the transformed x_h^{in} and x_l^{in} before aggregation:

$$\begin{cases} x_h^{out} = \mathcal{U}(\mathcal{F}(x_l^{in}) * Att(x_l^{in})) + \mathcal{F}(x_h^{in}) * \mathcal{U}(1 - Att(x_l^{in})) \\ x_l^{out} = \mathcal{F}(x_l^{in}) * Att(x_l^{in}) \end{cases} \quad (1)$$

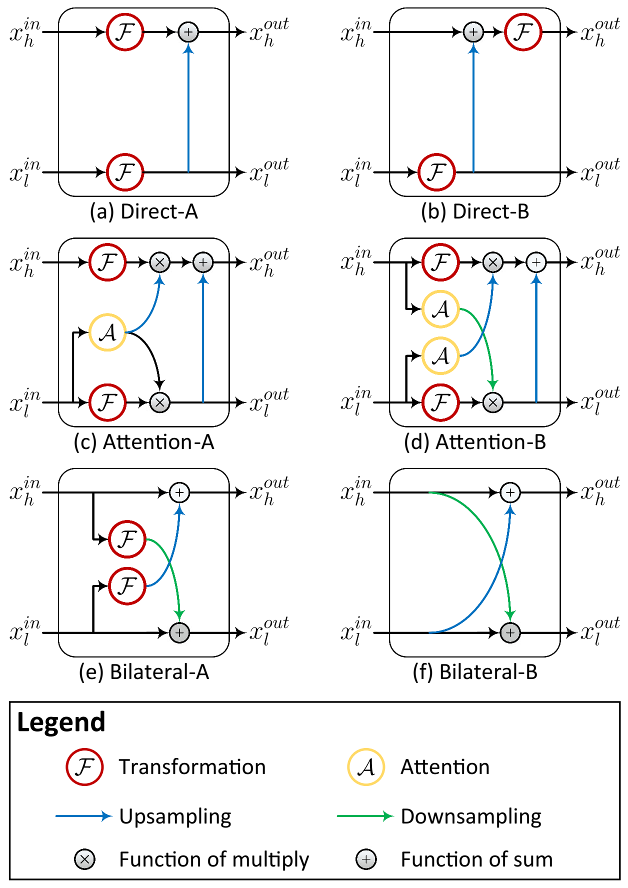


Fig. 6 Diagrams of six multi-path interaction modules. These variants of interaction modules can be grouped into three categories: direct feature aggregation (**a**, **b**), attention-augmented feature aggregation (**c**, **d**), and bilateral feature interaction (**e**, **f**)

where $\mathcal{F}$ and $\mathcal{U}$ denote the convolutional transformation and spatial upsampling, respectively. *Attention-B* further upgrades the simple attention in a bi-direction manner by integrating each path with the attention derived from another path.

In practice, the above four types of interaction modules are computationally expensive due to the heavy transformations and attentions. Thus, techniques (Chen et al., 2016; Tao et al., 2020; Yu et al., 2021a; Zhao et al., 2018) usually place them at the end of the multi-path networks, while leaving the path interactions at early stages unexploited. To alleviate this issue, two efficient interaction modules are devised to enable the interactions at early stages, namely *Bilateral-A/B*. *Bilateral-A* transforms low-resolution feature and injects it into the high-resolution one, and vice versa. Such symmetrical design treats each network path equally and the simple transformation triggers the interaction at early stages in an efficient fashion. *Bilateral-B* further eliminates the convolutional transformation in *Bilateral-A*, leading to the simplest interaction by solely applying spatial resizing and element-wise summation on input feature maps:

Table 1 Expanding operations along three dimensions

Depth $\Delta_{\mathcal{B}}$	Width $\Delta_{\mathcal{C}}$	Resolution $\Delta_{\mathcal{R}}$
$\{0, 1, 1, 1, 1\}$	$\{4, 8, 16, 32, 32\}$	$\{1/8, 0, 0\}$
$\{0, 0, 1, 1, 1\}$	$\{0, 8, 16, 32, 32\}$	$\{0, 1/8, 0\}$
$\{0, 0, 0, 1, 1\}$	$\{0, 0, 16, 32, 32\}$	$\{0, 0, 1/8\}$
	$\{0, 0, 0, 32, 32\}$	

$$\{x_h^{out}, x_l^{out}\} = \{x_h^{in} + \mathcal{U}(x_l^{in}),\ x_l^{in} + \mathcal{D}(x_h^{in})\}, \tag{2}$$

where $\mathcal{U}$ and $\mathcal{D}$ denote upsampling and downsampling of the feature maps. The comparisons between all the six variants of interactions will be elaborated in the experiments, and we empirically verify that *Bilateral-B* achieves the best speed/accuracy tradeoff. Therefore, we adopt *Bilateral-B* as the default interaction module in our LPS-Net.

3.4 Scalable Architectures

Next, we elaborate our progressive expansion process to automatically develop a series of scalable LPS-Net. Given the selected convolutional block and interaction module, a particular LPS-Net architecture can be defined as $\mathcal{N} = \{\mathcal{B}, \mathcal{C}, \mathcal{R}\}$, where $\mathcal{B}$, $\mathcal{C}$, $\mathcal{R}$ are adjustable parameters described in Sect. 3.1. The network complexity is thus determined by these parameters from three dimensions. **Depth** dimension ($\mathcal{B}$) is the number of blocks which determines the ability of the network to capture the high-level semantic information. **Width** dimension ($\mathcal{C}$), i.e., the number of channels at each stage, impacts the learning capacity of each convolution. **Resolution** dimension ($\mathcal{R}$) represents the spatial granularity of each path. To balance the three dimensions when designing LPS-Net architecture, we start by building a tiny network, and then expand a single dimension at one time in a progressive manner.

Technically, we set the parameters of tiny network $\mathcal{N}^0$ as $\mathcal{B}^0 = \{1, 1, 1, 1, 1\}$, $\mathcal{C}^0 = \{4, 8, 16, 32, 32\}$, and $\mathcal{R}^0 = \{1/2, 0, 0\}$. Please note that the path with $r = 0$ is not included, and thus the tiny network only contains a single path. In this way, the tiny network is very efficient with only 0.38 ms latency on a single GPU, and we further scale it to the heavier ones by multi-step expansion.

Following the key principle that the searched networks should have fewer layers/channels at the early stages, and more layers/channels at the later stages, we define the ten candidates of expanding operations ($\Delta = \Delta_{\mathcal{B}} \cup \Delta_{\mathcal{C}} \cup \Delta_{\mathcal{R}}$) along the three dimensions in Table 1. The spirit behind this principle is that the extraction/representation of high-level semantic information of images demands more layers/channels than that of low-level features. Such an empirical principle has been widely adopted for manually (e.g., BiSeNetV2 (Yu et al., 2021a)) and automatically (e.g., CAS (Zhang et al., 2019b)) designing networks. To this end, when expanding the depth/width of shallow layers, we enforce the deeper layers to be expanded at the same time, and the width of deep layers is expanded faster than shallow ones.

When $\mathcal{N} = \{\mathcal{B}, \mathcal{C}, \mathcal{R}\}$ is expanded by δ, δ will be added to $\mathcal{N}$ along its corresponding dimension and produce the expanded network (e.g. $\{\mathcal{B} + \delta, \mathcal{C}, \mathcal{R}\}$ when $\delta \in \Delta_{\mathcal{B}}$). For each step, we greedily determine the executed expanding operation by comparing the tradeoff between speed and accuracy of each operation $\delta \in \Delta$. Specifically, for the i-th step, the target latency after expansion is first determined by the maximum latency across different expanding operations:

$$L_T = \max_{\delta \in \Delta} Lat(\mathcal{N}^{i-1} + \delta), \tag{3}$$

where $\mathcal{N}^{i-1}$ denotes the architecture from the last step and $Lat(\cdot)$ measures the latency of the network. Then, a stepsize for each operation is calculated as:

$$k_\delta = \arg\min_{k \geq 1} |Lat(\mathcal{N}^{i-1} + k \cdot \delta) - L_T|, \tag{4}$$

targeting for better aligning the resultant latency to L_T after expanding different operations. Nevertheless, since the expansion only produces networks in discrete steps, the latencies of the expanded networks may exceed L_T. If the selection of operation only depends on the performances of expanded networks, it may be biased to the operations which leads to a larger latency increase. To tackle this issue, the optimal expanding operation δ^* is finally selected by maximizing the ratio of performance increase and latency increase after expanding at each step:

$$\delta^* = \arg\max_{\delta \in \Delta} \frac{Perf(\mathcal{N}^i_\delta) - Perf(\mathcal{N}^{i-1})}{Lat(\mathcal{N}^i_\delta) - Lat(\mathcal{N}^{i-1})}, \qquad \mathcal{N}^i_\delta = \mathcal{N}^{i-1} + k_\delta \cdot \delta \tag{5}$$

where $Perf(\cdot)$ evaluates the performances of semantic segmentation. As such, starting from the tiny network $\mathcal{N}^0$, we iteratively perform the network expansion I times and finally derive a family of LPS-Net with different complexities. Algorithm 1 summarizes the expansion algorithm of LPS-Net.

4 Experiments

We empirically verify the merit of our LPS-Net by conducting a thorough evaluation of efficient semantic segmentation on Cityscapes (Cordts et al., 2016). Expressly, we first undertake the experiments for semantic segmentation to validate the lightweight designs of the convolutional blocks and the way of interactions across paths in LPS-Net with a pre-fixed structure. The second experiment examines the accuracy and latency tradeoffs along the progressive growth of network complexity of LPS-Net with regard to the number of convolutional blocks, the number of channels, and the input resolution. The third experiment compare LPS-Net with two progressive expansion schemes to verify the merit of our greedy algorithm on progressive expansion. The fourth experiment compares three scalable levels of LPS-Net with several state-of-the-art fast semantic segmentation methods and demonstrates the better tradeoffs. Furthermore, we evaluate the transferability of the three versions of LPS-Net, which are learnt on Cityscapes, on BDD100K (Yu et al., 2018b) and CamVid (Brostow et al., 2008) datasets for semantic segmentation. We also migrate our LPS-Net to embedded devices and prove the efficacy on two different devices. We further scale up LPS-Net and verify that it has the ability to achieve better segmentation results in a more efficient way. Finally, we validate the high flexibility of LPS-Net on two object detection tasks.

Algorithm 1 The progressive expansion in LPS-Net.

```
1:  Input:
2:     Expanding operations $\Delta$
3:     number of expansion steps $I$
4:  Output:
5:     A family of LPS-Net with different latencies:
6:     $\mathbf{N} = \{\mathcal{N}^i | i = 0, 1, 2, 3, ..., I\}$
7:  Initialization:
8:     $\mathcal{N}^0 = \{\mathcal{B}^0, \mathcal{C}^0, \mathcal{R}^0\}$
9:        $= \{\{1, 1, 1, 1, 1\}, \{4, 8, 16, 32, 32\}, \{1/2, 0, 0\}\}$
10:    $P = Perf(\mathcal{N}^0)$, $L = Lat(\mathcal{N}^0)$, $i = 1$
11: Do:
12: while $i \leq I$ do
13:    for all $\delta \in \Delta$ do
14:        $L'_\delta = Lat(\mathcal{N}^{i-1} + \delta)$
15:    end for
16:    $L_T = max(\{L'_\delta | \delta \in \Delta\})$
17:    for all $\delta \in \Delta$ do
18:        $k_\delta = \arg\min_{k \geq 1} |Lat(\mathcal{N}^{i-1} + k \cdot \delta) - L_T|$
19:        $\mathcal{N}^i_\delta = \mathcal{N}^{i-1} + k_\delta \cdot \delta$
20:                          ▷ Expand $\mathcal{N}^{i-1}$ using $\delta$ with stepsize $k_\delta$.
21:        $P_\delta = Perf(\mathcal{N}^i_\delta)$
22:                      ▷ Train and evaluate $\mathcal{N}^i_\delta$ for semantic segmentation.
23:        $L_\delta = Lat(\mathcal{N}^i_\delta)$
24:    end for
25:    $\delta^* = \arg\max_{\delta \in \Delta} (P_\delta - P)/(L_\delta - L)$
26:                          ▷ Select operation by comparing tradeoffs.
27:    Update: $\mathcal{N}^i = \mathcal{N}^{i-1} + k_{\delta^*} \cdot \delta^*$, $P = P_{\delta^*}$, $L = L_{\delta^*}$
28: end while
```

4.1 Datasets

The Cityscapes dataset contains 5,000 urban street images with high-quality pixel-level annotations of 19 classes. The image resolution is $1,024 \times 2,048$. All the images are split into three sets of 2,975, 500 and 1,525 for training, validation, and testing, respectively. For fair comparison, the additional set of 23,473 coarsely annotated images of this dataset is not utilized in our experiments. We train LPS-Net on the training set and evaluate the networks on the validation set. Following the standard protocol in (Fan et al., 2021; Lin et al., 2020; Zhang et al., 2019b), LPS-Net is learnt on the training plus validation sets when submitting the model to online Cityscapes server and reporting the performance on the testing set.

To further examine the generalization of LPS-Net, we re-evaluate the networks on BDD100K (Yu et al., 2018b) and CamVid (Brostow et al., 2008) datasets. BDD100K is a recently released urban street dataset. For semantic segmentation, BDD100K includes 8,000 images being annotated with the consistent labels in Cityscapes. In between, 7,000 and 1,000 images are exploited for training and validation, respectively. The image resolution is $720 \times 1,280$. CamVid consists of 5 video sequences with resolution 720×960, and is labeled at one frame per second with 11 semantic categories. 468 and 233 labeled frames in CamVid is utilized for training and testing, respectively.

4.2 Implementation Details

We implement our proposal on PyTorch (Paszke et al., 2019) platform and employ mini-batch Stochastic Gradient Descent (SGD) algorithm for model optimization. In the **training stage**, we utilize the cross-entropy loss and train LPS-Net from scratch for 90K iterations, with a batch size of 16, momentum 0.9 and weight decay 0.0005. "Poly" policy with power 0.9 is adopted with initial learning rate 0.01. We use a crop size of 768×768 and exploit color-jittering, random horizontal flip and random scaling ($0.5\times \sim 2.0\times$) to augment training data. Model re-parameterization (Ding et al., 2021b) is also adopted for training. To further improve the capability of multi-path networks, we involve more powerful training strategies for the experiments in Sect. 4.6, including ImageNet (Russakovsky et al., 2015) pre-training, large crop size ($768 \times 1,536$), more training iterations (180K) and online hard element mining (Chen et al., 2020; Fan et al., 2021). Note that these strategies only enhance the network training and have no influence on model inference. Such training strategies employed for experiments on Cityscapes dataset are adjusted to train LPS-Net on BDD100K and CamVid datasets. For the experiments on BDD100K, we use a crop size of 720×1280 and the other strategies remain unchanged. For CamVid, we train the LPS-Net for 40K iterations, with a batch size of 12, an initial learning rate of 0.03, and a crop size 720×960. ImageNet pre-training and online hard element mining are also involved in experiments on BDD100K and CamVid.

In the **inference stage**, we input the image into our LPS-Net and perform semantic segmentation without any evaluation tricks (e.g., flipping, sliding-window testing and multi-scale inference), which in general improve the performance but at the expenses of extra latency. The Intersection over Union (IoU) per category and mean IoU over all the semantic categories are utilized as the performance metric. The inference latency of the network is measured by running networks with batch size of 1 on one NVIDIA GTX 1070Ti (Sects. 4.3, 4.4, 4.5 and 4.9), 1080Ti (Sects. 4.6 and 4.8) or embedded devices (Sect. 4.7) with TensorRT in FP32 mode, unless otherwise stated.

4.3 Evaluation on Lightweight Designs

We first examine the lightweight designs of the convolutional blocks and the ways of interactions across paths in LPS-Net. We start the validation on the impact of the type of convolutional blocks. Specifically, we capitalize on seven convolutional blocks to build the corresponding multi-path networks. Following (Chen et al., 2016; Ding et al., 2021b), $\mathcal{B}$ and $\mathcal{R}$ of all the networks are respectively set to $\{1, 2, 4, 7, 7\}$ and $\{1, 1/4, 0\}$. The channel widths $\mathcal{C}$ are set to $\{1c, 2c, 4c, 8c, 8c\}$ for all the networks, where c is to adjust the inference latency of each network to approximate 5 ms for fair comparisons. In order to purely verify the convolutional blocks irrespective of the interaction influence, we do not include any path interactions here and exploit channel-wise concatenation to aggregate the outputs from all paths of the multi-path network.

Table 2 summarizes the mIoU performances, latencies, FLOPs, and FLOPs-efficiencies of the multi-path networks built on seven types of convolutional blocks. Similar to the observations in Sect. 3.2, 3×3 SepConv shows a very low FLOPs-efficiency. Conditioning on almost the same latency with 3×3 Conv, the utilization of 3×3 SepConv results in much fewer FLOPs, making the computational capability of the network weak. 3×3 SepConv is also inferior to 3×3 Conv in terms of mIoU performance. Furthermore, ShuffleNet Unit, Inverted Residual and Ghost Module are proven to be three very effective and lightweight convolutional blocks and all three blocks involve 3×3 SepConv. Inverted Residual and Ghost Module have benefited from the inverted residual structure with linear bottleneck and a series of linear transformations with cheap cost, respectively. They both greatly improve 3×3 SepConv, but the mIoU performances are still lower than that of 3×3 Conv. Residual contains a stack of two 3×3 convolutions with a shortcut connection and is extended to Bottleneck by stacking three convolutions (1×1, 3×3, 1×1) instead of two. The use of 1×1 convolution in Bottleneck leads to block fragmentation and affects its FLOPs-efficiency. Compared to 3×3 Conv, an extra short cut in Residual slightly lowers FLOPs-efficiency

Table 2 Performance comparisons on Cityscapes validation set between multi-path networks which are built on seven types of convolutional block, including ShuffleNet Unit (Ma et al., 2018), Inverted Residual (Sandler et al., 2018), Ghost Module (Han et al., 2020), and Residual/Bottleneck (He et al., 2016)

Conv Blocks	$B = \{1, 2, 4, 7, 7\}$ $Latency \approx 5.0ms@1070Ti$				$B = \{1, 2, 2, 4, 4\}$ $Latency \approx 3.5ms@1070Ti$				$B = \{1, 2, 2, 4, 4\}$ $Latency \approx 100ms@Nano$			
	mIoU (%)	Latency (ms)	FLOPs (G)	FLOPs-eff. (G/ms)	mIoU (%)	Latency (ms)	FLOPs (G)	FLOPs-eff. (G/ms)	mIoU (%)	Latency (ms)	FLOPs (G)	FLOPs-eff. (G/ms)
ShuffleNet Unit	50.8	5.11	0.8	0.15	41.8	3.36	0.4	0.11	50.5	101	0.8	0.01
Inverted Residual	59.2	5.12	2.0	0.39	50.4	3.47	0.8	0.24	54.5	99	1.6	0.02
Ghost Module	58.3	5.25	2.5	0.47	55.3	3.52	1.4	0.41	59.9	99	3.4	0.03
Residual	66.1	5.24	16.1	3.08	62.6	3.53	7.5	2.12	62.6	95	7.5	0.08
Bottleneck	55.7	5.65	2.2	0.39	51.2	3.72	1.5	0.40	56.0	86	2.5	0.03
3x3 SepConv	53.9	5.10	1.1	0.21	48.1	3.91	0.6	0.15	55.4	102	1.4	0.01
3x3 Conv	66.2	5.08	18.5	3.63	63.8	3.31	10.6	3.20	62.6	84	7.8	0.09

The latencies here are measured with 1070Ti GPU and the embedded device NVIDIA Jetson Nano, and performances are evaluated by training networks without powerful strategies described in Sect. 4.2

and increases the latency by ∼3%. Benefiting from the high FLOPs-efficiency and better tradeoff, 3×3 Conv exhibits the highest mIoU (66.2%) with the lowest latency (5.08 ms). We further analyze the high FLOPs-efficiency of 3×3 Conv from the perspective of devices. FLOPs-lightweight convolutions (e.g. Ghost Module and 3×3 SepConv) usually suffer from a low ratio of computing to memory operations (Gholami et al., 2018) and introduce extra overheads like kernel launching and synchronization. In contrast, 3×3 Conv is more friendly to devices and could be accelerated by the Winograd supported in modern computing libraries like cuDNN. As such, 3×3 Conv has high FLOPs-efficiency. To further verify the effectiveness of 3×3 Conv in other latency ranges and hardware platforms, we change $\mathcal{B}$ from $\{1, 2, 4, 7, 7\}$ to $\{1, 2, 2, 4, 4\}$ for faster inference, and utilize both the 1070Ti (GPU) and the NVIDIA Jetson Nano (embedded device) for latency measurement. For the two hardware platforms 1070Ti and Nano, we adjust the parameter c to make the inference latency of each network approximate 3.5 ms and 100 ms, respectively. As detailed in Table 2, 3×3 Conv achieves the best tradeoff with high FLOPs-efficiency across different latency ranges and hardware platforms. Figure 7a visualizes the mIoU-latency curves of the multi-path networks built up with seven types of convolutional blocks. As expected, the curve of 3×3 Conv is over all the other curves, showing its better tradeoff. Thus, we exploit the 3×3 Conv as the convolutional block in LPS-Net.

Then, we study how the interactions across paths influence the accuracy-latency tradeoff of the multi-path networks. In this comparison, the network parameters $\mathcal{B}$ and $\mathcal{R}$ are kept unchanged. We fix $c = 16$ and exploit 3×3 Conv as the convolutional block to construct the multi-path networks. Table 3 details the mIoU performance and latency when integrating different path interactions in multi-path networks. As expected, the results across six ways of interactions consistently indicate that using interaction exhibits better mIoU performance and higher latency against Baseline which does not involve path interaction. More specifically, the larger mIoU improvements are attained when employing the Bilateral-A/B interactions than the Direct-A/B and Attention-A/B interactions. Bilateral-B purely basing on bilinear interpolation leads to less increase of latency than Bilateral-A which uses two additional convolutional transformations. Among all the six interactions, Bilateral-B achieves the largest mIoU gain of 3.0% with the least extra latency of 0.29 ms, and hence we utilize Bilateral-B as the path interaction in our LPS-Net. To consolidate such a choice, we additionally validate the interactions by integrating them into the multi-path networks with $c = 12$ and $c = 8$, respectively. The inference latencies are measured on both 1070Ti and Nano. Table 3 summarizes the comparisons of interactions from the perspective of mIoU-latency tradeoff, which are also demonstrated in Fig. 7b, c. The experimental results

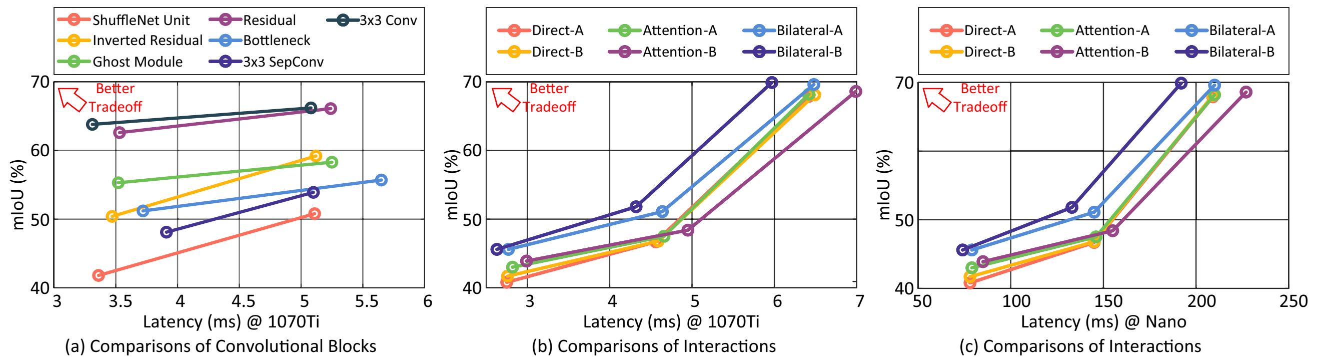


Fig. 7 The mIoU-latency curve comparisons between **a** convolutional blocks and **b**, **c** interactions in multi-path networks

Table 3 Performance comparisons on Cityscapes validation set between multi-path networks with different interactions

Interaction	$c = 16$						$c = 12$						$c = 8$					
	mIoU (%)		Latency (ms) 1070Ti		Nano		mIoU (%)		Latency (ms) 1070Ti		Nano		mIoU (%)		Latency (ms) 1070Ti		Nano	
Baseline	66.9		5.68		183		45.5		3.92		127		39.6		2.38		70	
Direct-A	67.9	+1.0	6.42	+0.74	209	+25	46.7	+1.2	4.56	+0.64	145	+18	40.8	+1.2	2.75	+0.37	78	+8
Direct-B	68.1	+1.2	6.49	+0.81	209	+26	46.8	+1.3	4.59	+0.66	145	+18	41.7	+2.1	2.76	+0.38	78	+9
Attention-A	68.2	+1.3	6.42	+0.74	210	+27	47.5	+1.9	4.66	+0.73	146	+19	43.0	+3.4	2.82	+0.44	79	+9
Attention-B	68.6	+1.7	6.99	+1.31	227	+44	48.4	+2.9	4.95	+1.03	155	+28	43.9	+4.3	2.99	+0.61	85	+15
Bilateral-A	69.6	+2.7	6.48	+0.80	210	+27	51.1	+5.5	4.64	+0.71	145	+18	45.6	+6.0	2.77	+0.39	79	+9
Bilateral-B	69.9	+3.0	5.97	+0.29	192	+9	51.8	+6.3	4.32	+0.39	133	+6	45.6	+6.0	2.63	+0.25	74	+4

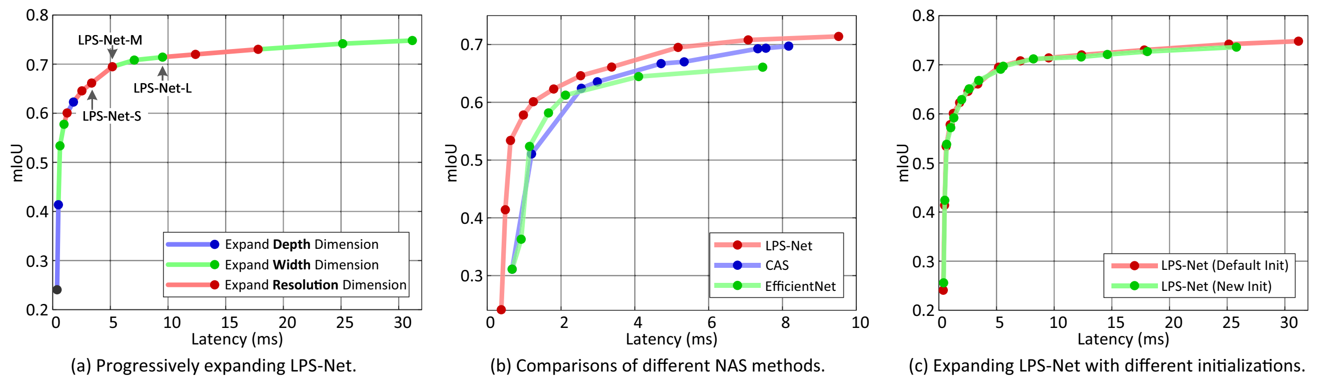


Fig. 8 **a** Progressively expanding LPS-Net on Cityscapes. Each dot denotes an expanded networks and each colored segment of the curve represents an expansion along a specific dimension. **b** Comparisons of different neural architecture search (NAS) algorithms performed in our search space for efficient semantic segmentation. **c** Progressively expanding LPS-Net on Cityscapes with different initialization. Note that the inference latencies here are measured with 1070Ti GPU, and performances are evaluated by training networks without powerful strategies described in Sect. 4.2

verify that Bilateral-B achieves the best tradeoff across different latency ranges and devices.

4.4 Evaluation on Progressively-Scalable Scheme

Next, we analyze the accuracy-latency balance as the progressive expansion of LPS-Net proceeds. Figure 8a depicts the first 14 steps of the process on Cityscapes. Table 4 further details the parameters, latency, mIoU performance and expanding dimension of the network at each expansion step of LPS-Net in Fig. 8a. We initialize LPS-Net with a tiny structure and the network parameters are $\mathcal{B}^0 = \{1, 1, 1, 1, 1\}$, $\mathcal{C}^0 = \{4, 8, 16, 32, 32\}$, $\mathcal{R}^0 = \{1/2, 0, 0\}$. The mIoU and latency of such start point is 24.1% and 0.38 ms, respectively.

Table 4 Network Parameters (Depth-$\mathcal{B}$, Width-$\mathcal{C}$ and Resolution-$\mathcal{R}$), latency, mIoU performance and expanding dimension of the network at each expansion step of LPS-Net

Step	$\mathcal{B}$	$\mathcal{C}$	$\mathcal{R}$	Latency (ms)	mIoU (%)	Dimension	Name
0	{1, 1, 1, 1, 1}	{4, 8, 16, 32, 32}	{1/2, 0, 0}	0.38	24.1	–	–
1	{1, 1, 1, 8, 8}	{4, 8, 16, 32, 32}	{1/2, 0, 0}	0.49	41.4	Depth	–
2	{1, 1, 1, 8, 8}	{4, 16, 32, 64, 64}	{1/2, 0, 0}	0.63	53.4	Width	–
3	{1, 1, 1, 8, 8}	{8, 24, 48, 96, 96}	{1/2, 0, 0}	0.98	57.8	Width	–
4	{1, 1, 1, 8, 8}	{8, 24, 48, 96, 96}	{5/8, 0, 0}	1.25	60.1	Resolution	–
5	{1, 3, 3, 10, 10}	{8, 24, 48, 96, 96}	{5/8, 0, 0}	1.80	62.3	Depth	–
6	{1, 3, 3, 10, 10}	{8, 24, 48, 96, 96}	{5/8, 1/4, 0}	2.53	64.6	Resolution	–
7	{1, 3, 3, 10, 10}	{8, 24, 48, 96, 96}	{3/4, 1/4, 0}	3.37	66.1	Resolution	LPS-Net-S
8	{1, 3, 3, 10, 10}	{8, 24, 48, 96, 96}	{1, 1/4, 0}	5.17	69.5	Resolution	LPS-Net-M
9	{1, 3, 3, 10, 10}	{8, 24, 64, 128, 128}	{1, 1/4, 0}	7.07	70.8	Width	–
10	{1, 3, 3, 10, 10}	{8, 24, 64, 160, 160}	{1, 1/4, 0}	9.52	71.4	Width	LPS-Net-L
11	{1, 3, 3, 10, 10}	{8, 24, 64, 160, 160}	{9/8, 1/4, 0}	12.38	72.0	Resolution	–
12	{1, 3, 3, 10, 10}	{8, 24, 64, 160, 160}	{11/8, 1/4, 0}	17.81	73.0	Resolution	–
13	{1, 3, 3, 10, 10}	{8, 32, 80, 192, 192}	{11/8, 1/4, 0}	25.14	74.2	Width	–
14	{1, 3, 3, 10, 10}	{8, 32, 96, 224, 224}	{11/8, 1/4, 0}	31.18	74.8	Width	–

Note that the inference latencies here are measured with 1070Ti GPU, and performances are evaluated by training networks without powerful strategies described in Sect. 4.2

As shown in the figure, expanding one certain dimension of Depth (the number of convolutional blocks), Width (the number of channels), or the input Resolution in LPS-Net always improves the mIoU performance. The fact that the selected dimension is various to strike the best mIoU performance-inference latency tradeoff at different steps necessitates the use of multiple dimensions for expansion.

The first step is along the network Depth direction and significantly boosts up the mIoU to 41.4% with only 0.11 ms additional latency. We speculate that this may be the result of feature enrichment by stacking more blocks particularly in the context of small channel width and low image resolution. In this case, deepening networks could readily enhance the network capability. The second and third steps further expand Width dimension, and the network after three steps achieves 57.8% mIoU with less than 1 ms latency. The results indicate that the network tends to expand the Depth and Width dimensions especially at the very early steps. At the sixth step, the expansion proceeds along the Resolution direction and starts to develop a two-path networks. The next two consecutive steps also work on the factor of Resolution and increase $\mathcal{R}$ to $\{1, 1/4, 0\}$. The mIoU performance of the two-path networks at the eighth step reaches 69.5% and the latency is 5.17 ms. That basically verifies the effectiveness of multi-path design to manage the accuracy-latency balance for semantic segmentation. The following steps continue the expansion along either the Width or Resolution dimension. We observe that the performance gain decreases in these steps and speculate that this may be caused by two reasons: (1) the performance of the heavyweight model becomes saturated and is hard to be improved by a large margin, and (2) we utilize the same training strategy for the models with different scales. The heavier network usually requires more training epochs, stronger data augmentation and more network regularization. The advance of heavier network is not fully exploited under our training strategy. Such an expansion process indicates some practical insights for efficient semantic segmentation. For example, stacking more blocks for tiny networks could effectively improve the segmentation with negligible extra latency. The multi-path fusion is not always the best choice especially when the networks' inference latency is extremely constrained (e.g., $\leq$ 2 ms). Compared to increasing channel width and enlarging input resolutions, fusing three or more paths is not optimal for efficient semantic segmentation. To cover different levels of LPS-Net in complexity, we choose the networks at the seventh, eighth, and tenth steps as the small, medium, and large version of LPS-Net. We name these networks as LPS-Net-S, LPS-Net-M, and LPS-Net-L, respectively.

To better clarify the effectiveness of our network search algorithm, we have additionally experimented by remoulding existing network search algorithms in CAS (Zhang et al., 2019b) and EfficientNet (Tan & Le, 2019) in our search space for efficient semantic segmentation. The performance comparisons of different network search algorithms are provided in Fig. 8b. Particularly, our LPS-Net achieves the highest mIoU across different latencies. This somewhat reveals the difficulty to dynamically optimize the input resolutions and the number of paths for a multi-path network by the NAS-like auto-searching schemes. Moreover, our progressive expan-

Table 5 The intuitively designed expanding operations. Such operations allow the depth and width of each stage in network to be expanded independently

Depth $\Delta_{\mathcal{B}}$	Width $\Delta_{\mathcal{C}}$	Resolution $\Delta_{\mathcal{R}}$
{1, 0, 0, 0, 0}	{4, 0, 0, 0, 0}	
{0, 1, 0, 0, 0}	{0, 8, 0, 0, 0}	{1/8, 0, 0}
{0, 0, 1, 0, 0}	{0, 0, 16, 0, 0}	{0, 1/8, 0}
{0, 0, 0, 1, 0}	{0, 0, 0, 32, 0}	{0, 0, 1/8}
{0, 0, 0, 0, 1}	{0, 0, 0, 0, 32}	

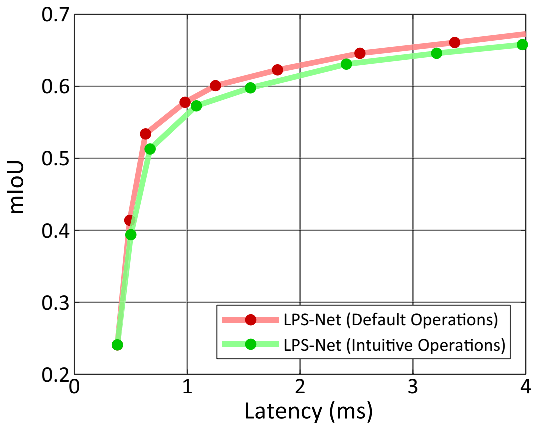


Fig. 9 Comparisons of different candidates of expanding operations

sion also takes the advantage of deriving a series of networks with various accuracy-speed tradeoffs in one search process, while CAS requires much more search time to seek the architecture for each target latency independently.

To verify how the initial tiny structure influences the progressive expansion of LPS-Net, we compare the expansion process by starting from the default point ($\mathcal{B}^0 = \{1, 1, 1, 1, 1\}$, $\mathcal{C}^0 = \{4, 8, 16, 32, 32\}$, $\mathcal{R}^0 = \{1/2, 0, 0\}$) or a new point ($\mathcal{B}^0 = \{1, 1, 1, 2, 2\}$, $\mathcal{C}^0 = \{4, 8, 16, 32, 32\}$, $\mathcal{R}^0 = \{1/2, 0, 0\}$). Figure 8c depicts the first 14 steps of expansion starting from the two points and shows that the searching process is less influenced by different start points.

To validate the effectiveness of the design principle of expanding operations utilized in LPS-Net, we experiment with a group of modified expanding operations detailed in Table 5. This intuitive design allows the depth and width of different stages to be expanded independently. The performance comparisons of different groups of expanding operations are shown in Fig. 9. Particularly, the default expanding operations utilized in LPS-Net constantly lead to higher mIoU across different latencies.

4.5 Comparisons of Progressive Expansion Schemes

To better verify the merit of our greedy algorithm on progressive expansion, we additionally compared with two

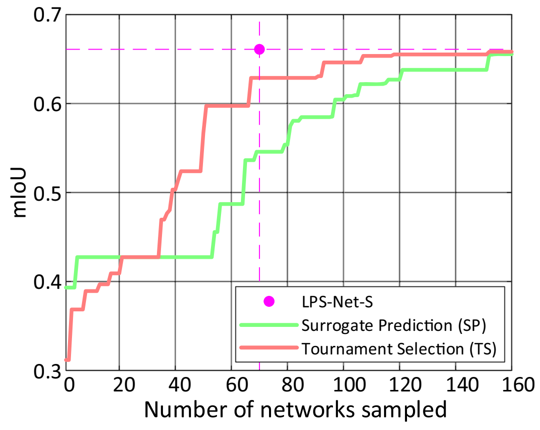


Fig. 10 Comparisons of different progressive expansion schemes

expansion schemes: surrogate prediction (SP) (Liu et al., 2017), and tournament selection (TS) (Goldberg & Deb, 1990; Real et al., 2017), for searching efficient semantic segmentation networks. In our implementations, we run the two approaches and enforce the searched networks with inference time less than 3.37ms@1070Ti (comparable with LPS-Net-S). All the searched networks are trained and evaluated on the Cityscapes dataset without powerful training strategies like ImageNet pre-training, large crop size, more training iterations, and online hard element mining. The latencies are measured with a 1070Ti GPU.

The surrogate prediction (SP) utilizes a sequential model-based optimization strategy to perform the progressive expansion in an evolutionary manner. Specifically, we first expand the tiny network to generate 32 networks as the initial population by using randomly selected operations. Each network is regarded as an individual of the population, and the mIoU represents the fitness of the individual. A surrogate model learnt on the population is then employed to predict the mIoU of a network without actually training it. For each expansion step, we transverse all the possible expansions of each individual and select the 16 most promising expanded network architectures with respect to the prediction by the surrogate model. Note that all the expanded networks with latencies larger than 3.37 ms are filtered out before the selection. The selected networks are trained to obtain their final mIoU and exploited to upgrade the surrogate model.

The tournament selection (TS) based progressive expansion uses repeated pairwise competitions of random individuals for population updating. Specifically, we set the population size as 32 and initialize the population via random expansion, e.g., surrogate prediction based expansion. For each expansion step, we randomly choose two individuals from the population and compare their fitness. The individual with lower fitness is removed and the other one is further expanded by using a randomly selected operation. The expanded individual is trained and evaluated for seman-

Table 6 Inference latencies/FPS of LPS-Net measured on NVIDIA GTX 1070Ti/1080Ti with the resolution 1024 × 2048

	LPS-Net-S	LPS-Net-M	LPS-Net-L	GPU
Latency	3.37	5.17	9.52	1070Ti
(ms)	2.42	3.71	6.59	1080Ti
FPS	296.7	193.4	105.0	1070Ti
	413.5	269.7	151.8	1080Ti

Table 7 Performance comparisons on Cityscapes validation set between different training strategies of LPS-Net

mIoU (%)	LPS-Net-S	LPS-Net-M	LPS-Net-L
baseline	66.1	69.5	71.4
+ImageNet	69.3	71.5	73.3
+LargeCrop	71.3	73.5	75.1
+MoreIters	72.5	75.4	76.7
+OHEM	73.9	76.5	78.2

tic segmentation on the Cityscapes, and included into the population. We employ 16 workers to asynchronously and repeatedly update the population until the number of sampled networks reaches 160.

To compare the efficiency of SP, TS, and LPS-Net, we plot the mIoU on the Cityscapes validation set of the best network sought by each method versus the number of networks that are sampled (fully trained and validated). As shown in Fig. 10, the mIoU of the best networks consistently increases when searching for larger segmentation models. More remarkably, our LPS-Net-S achieves 66.1% mIoU with 3.37 ms latency at the cost of 70 sampled networks while the other two competitors fail to find a network with a better latency-mIoU tradeoff. That demonstrates the search efficiency of our LPS-Net.

4.6 Comparisons with State-of-the-Art Methods

We compare our LPS-Net with several state-of-the-art fast semantic segmentation methods. To ensure the inference speed is comparable with the most recent works (Chen et al., 2020; Fan et al., 2021; Yu et al., 2021a), we report the inference FPS of LPS-Net on an NVIDIA GTX 1080Ti GPU card here. Table 6 shows the comparisons of inference latencies/FPS measured on NVIDIA GTX 1070Ti and 1080Ti GPUs. We also exploit a stronger training setting to fairly compare with the baselines. Table 7 details the performance contribution of each training strategy, including ImageNet pre-training (ImageNet), large crop size (LargeCrop), more training iterations (MoreIters), and online hard element mining (OHEM). The mIoU performances of the three versions of LPS-Net are consistently improved by using each additional training strategy and finally reach 73.9%, 76.5%, and 78.2% mIoU on the Cityscapes validation set.

Table 8 summarizes the comparisons of mIoU performance and inference FPS on Cityscapes dataset. Note that only the models with FPS≥100 are included in the table. Overall, the results across validation and testing sets consistently indicate that LPS-Net exhibits better tradeoffs than other techniques including hand-crafted networks (e.g., (Fan et al., 2021; Li et al., 2019)) and auto-designed models (e.g., (Chen et al., 2020; Zhang et al., 2019b)). In particular, the mIoU score of LPS-Net-S achieves 73.4% on test set, making the absolute improvement over the best competitor STDC1Seg-50 by 1.5%. Moreover, LPS-Net-S is faster than STDC1Seg-50 and the inference speed reaches 413.5 FPS, which is impressive. The large version of LPS-Net, i.e., LPS-Net-L, attains 78.2%/77.3% mIoU on validation/testing set at 151.8 FPS. It runs 49% faster than DDRNet23slim with comparable mIoU. Compared with the auto-designed efficient models, LPS-Net-S obtains 0.4% higher test mIoU and 3.8× speed than the best competitor DF1-Seg. In addition, LPS-Net-L (151.8 FPS) also surpasses the auto-designed HR-NAS-A (Ding et al., 2021a) (64 FPS) by 3.0% test mIoU.

We showcase the qualitative results of LPS-Net-S, LPS-Net-M, and LPS-Net-L in Fig. 11. Similar to the observation in Table 8, the quality of prediction is consistently improved when the LPS-Net is incrementally expanded. For example, the prediction in the regions of *pole* (first row), *car* (second row), and *wall* (third row) is amended by expanding the complexity of LPS-Net.

Table 9 further summarizes the comparison between LPS-Net and several existing neural architecture search approaches with regard to the performances and search time. CAS, GAS, and FasterSeg, as the gradient-based methods (differentiable NAS), do not necessitate to repeatedly train/evaluate the searched networks and thus spend less time for searching. Nevertheless, their performances are apparently lower than that of our LPS-Net due to the gap between training objective during searching and the performance of the searched network. Compared to SP and TS with progressive expansion, LPS-Net consumes about 700 GPU hours (70 networks × 10 h/network) for LPS-Net-S, which is less than half of the search time of SP and TS. Moreover, LPS-Net takes the advantage of deriving a series of networks with various accuracy-speed tradeoffs in one search process. As a result, we can obtain the LPS-Net-S/M/L in one searching process at the cost of 1,000 GPU hours.

To examine the generalization of the network structures of LPS-Net, we further conduct the experiments on CamVid and BDD100K datasets with the three versions of LPS-Net which are learnt on Cityscapes. Table 10 details the comparisons of both performance and inference FPS on CamVid. The first group contains manually designed lightweight models, which are BiSeNet (Yu et al., 2018a), BiSeNetV2 (Yu et

Table 8 Comparisons of mIoU and inference FPS with the state-of-the-art methods on Cityscapes validation & testing sets. Note that only the models with FPS≥100 are included in the table and performances of LPS-Net are evaluated by training networks with powerful strategies described in Sect. 4.2

Method	Resolution	mIoU (%)		FPS	GPU	Accelerators
		Val	Test			
BiSeNet (Yu et al., 2018a)	768 × 1536	69.0	68.4	105.8	TitanXp	
BiSeNetV2 (Yu et al., 2021a)	512 × 1024	73.4	72.6	156.0	1080Ti	TensorRT
DABNet (Li et al., 2019)	512 × 1024	-	70.1	104.2	1080Ti	
DDRNet23slim (Hong et al., 2021)	1024 × 2048	77.8	77.4	101.6	2080Ti	
DFANet A (Li et al., 2019a)	1024 × 1024	-	71.3	100.0	TitanX	
DFANet B (Li et al., 2019a)	1024 × 1024	-	67.1	120.0	TitanX	
DFANet A' (Li et al., 2019a)	512 × 1024	-	70.3	160.0	TitanX	
Fast-SCNN (Poudel et al., 2019)	1024 × 1024	69.2	68.0	123.5	TitanXp	
MSFNet (Si et al., 2020)	512 × 1024	-	71.3	117.0	2080Ti	
SFNet(DF1) (Li et al., 2020b)	1024 × 2048	-	74.5	121.0	1080Ti	TensorRT
STDC1Seg-50 (Fan et al., 2021)	512 × 1024	72.2	71.9	250.4	1080Ti	TensorRT
STDC2Seg-50 (Fan et al., 2021)	512 × 1024	74.2	73.4	188.6	1080Ti	TensorRT
STDC1Seg-75 (Fan et al., 2021)	768 × 1536	74.5	75.3	126.7	1080Ti	TensorRT
CAS (Zhang et al., 2019b)	768 × 1536	71.6	70.5	108.0	TitanXp	
GAS (Lin et al., 2020)	769 × 1537	72.4	71.8	108.4	TitanXp	
AutoRTNet-A (Sun et al., 2021)	768 × 1536	72.9	72.2	110.0	TitanXp	
DF1-Seg-d8 (Li et al., 2019b)	1024 × 2048	72.4	71.4	136.9	1080Ti	TensorRT
DF1-Seg (Li et al., 2019b)	1024 × 2048	74.1	73.0	106.4	1080Ti	TensorRT
FasterSeg (Chen et al., 2020)	1024 × 2048	73.1	71.5	163.9	1080Ti	TensorRT
TinyHMSeg (Li et al., 2020a)	768 × 1536	-	71.4	172.4	1080Ti	TVM
LPS-Net-S	1024 × 2048	73.9	73.4	413.5	1080Ti	TensorRT
LPS-Net-M	1024 × 2048	76.5	74.9	269.7	1080Ti	TensorRT
LPS-Net-L	1024 × 2048	78.2	77.3	151.8	1080Ti	TensorRT

Table 9 Comparisons of performance and search time with different NAS approaches

Method	Performance			Search Time
	mIoU (%)	FPS	GPU	(GPU Hours)
gradient-based search				
CAS	71.6	108.0	TitanXp	144
GAS	72.4	108.4	TitanXp	160
FasterSeg†	73.1	163.9	1080Ti	48
progressive expansion				
SP†	72.5	414.5	1080Ti	1,600
TS†	72.6	413.9	1080Ti	1,600
LPS-Net-S†	73.9	413.5	1080Ti	700
LPS-Net-M†	76.5	269.7	1080Ti	800
LPS-Net-L†	78.2	151.8	1080Ti	1,000

The performances of the searched networks are evaluated on the Cityscapes validation set. † indicate that the FPS is measured with TensorRT

al., 2021a), DDRNet (Hong et al., 2021), DFANet (Li et al., 2019a), ICNet (Zhao et al., 2018), MSFNet (Si et al., 2020), SFNet (Li et al., 2020b), and STDC (Fan et al., 2021). The auto-designed lightweight architectures are listed in the second group, including CAS (Zhang et al., 2019b), GAS (Lin et al., 2020), AutoRTNet (Sun et al., 2021), FasterSeg (Chen et al., 2020), and HMSeg (Li et al., 2020a). LPS-Net-S suppresses the fastest competitor FasterSeg by 2.5% and the inference speed achieves 432.4 FPS. LPS-Net-L yields a better tradeoff (76.5% mIoU @ 169.3 FPS) than DDRNet23 (76.3% mIoU@94.0 FPS). Table 11 shows the comparisons between DRN (Yu et al., 2017), FasterSeg (Chen et al., 2020), and LPS-Net on the BDD100K dataset. Our LPS-Net-S again leads to an improvement of mIoU by 0.8% and a 99% speed-up against FasterSeg. The results basically validate LPS-Net from the perspective of network generalization.

4.7 LPS-Net-S on Embedded Devices

The real-world scenarios often demand the deployments on embedded devices. Here, we also test the small version of our LPS-Net, i.e., LPS-Net-S, on two devices of a single Nano and a single TX2, both of which are based on NVIDIA Jetson platform. Table 12 compares LPS-Net-S with DF1-Seg-d8 (Li et al., 2019b), FasterSeg (Chen et al., 2020), BiSeNetV2 (Yu et al., 2018a), and STDC1Seg-50 (Fan et al., 2021), regarding the mIoU performance on Cityscapes and inference speed (FPS) on the two devices. Note that the inference speed is measured by using FP32 data preci-

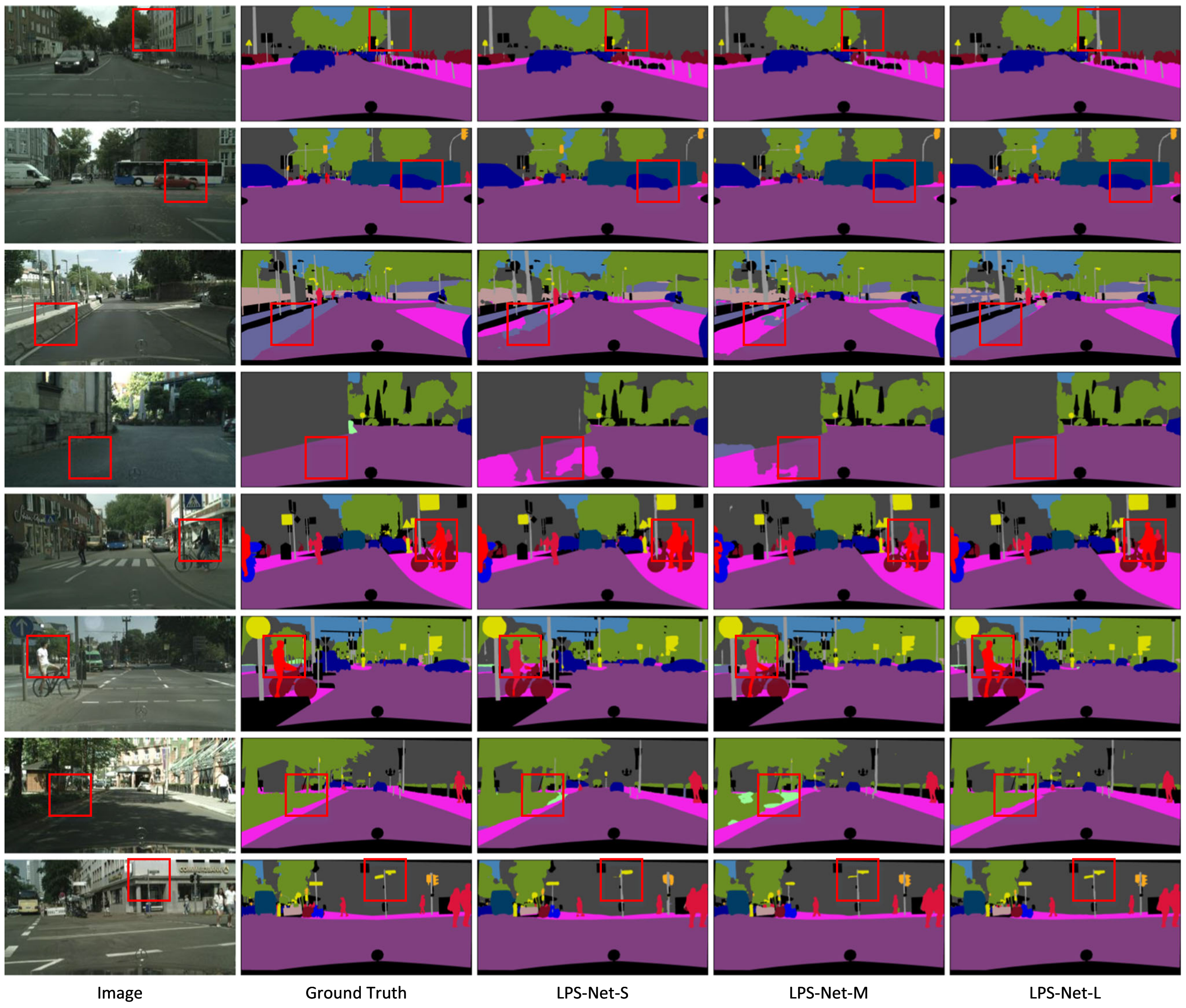


Fig. 11 Example of semantic segmentation results in Cityscapes. The improvements could be observed in the red bounding boxes

sion to avoid performance degradation. Clearly, LPS-Net-S strikes the superior tradeoff and achieves 10.0/25.6 FPS on Nano/TX2 at resolution 1024×2048, which is 41%/50% faster than the best competitor STDC1Seg-50 (Fan et al., 2021). The results demonstrate the advantage of LPS-Net on embedded devices.

4.8 LPS-Net for Better Segmentation Results

To further scale up LPS-Net for better segmentation results, we choose the network at the fourteenth expansion step as the huge version of LPS-Net, namely LPS-Net-H. With the stronger training setting (including ImageNet pre-training, large crop size, more training, and online hard element mining), LPS-Net-H achieves 80.7% mIoU on Cityscapes validation set at 46 FPS on NVIDIA GTX 1080Ti GPU, against DeepLabV3+ (Chen et al., 2018c) with 79.6% and 2 FPS. Note that the input images are not augmented (e.g. resizing and flipping) during inference. The results indicate that our LPS-Net has the ability to surpass models like DeepLab in a more efficient way.

4.9 LPS-Net for Object Detection

To validate the high flexibility of LPS-Net, we conduct the experiments on COCO (Lin et al., 2014) and DUTS (Wang et al., 2017) datasets for efficient object detection and salient object detection by directly applying the large version of LPS-Net (i.e., LPS-Net-L). Note that LPS-Net-L is learnt on Cityscapes for semantic segmentation. Table 13 summarizes the comparisons of performance and inference speed (FPS).

Table 10 Comparisons of mIoU and inference FPS with the state-of-the-art methods on CamVid testing set

Method	mIoU (%)	FPS	GPU
BiSeNet	65.6	175.0	TitanXp
BiSeNet(Res18)	68.7	116.3	TitanXp
BiSeNetV2†	72.4	124.5	1080Ti
BiSeNetV2-L†	73.2	32.7	1080Ti
DDRNet23slim	74.7	230.0	2080Ti
DDRNet23	76.3	94.0	2080Ti
DFANet A	64.7	120.0	TitanX
DFANet B	59.3	160.0	TitanX
ICNet	67.1	27.8	TitanX
MSFNet	72.7	160.0	2080Ti
MSFNet	75.4	91.0	2080Ti
SFNet(DF2)†	70.4	134.1	1080Ti
SFNet(Res18)†	73.8	35.5	1080Ti
STDC1Seg-50†	73.0	197.6	1080Ti
STDC2Seg-50†	73.9	152.2	1080Ti
CAS	71.2	169.0	TitanXp
GAS	72.8	153.1	TitanXp
AutoRTNet-A	73.5	140.0	TitanXp
FasterSeg†	71.1	398.1	1080Ti
HMSeg‡	75.1	130.8	1080Ti
TinyHMSeg‡	71.8	278.5	1080Ti
LPS-Net-S†	73.6	432.4	1080Ti
LPS-Net-M†	75.1	317.3	1080Ti
LPS-Net-L†	76.5	169.3	1080Ti

† and ‡ indicate that the FPS is measured with deep learning accelerators of TensorRT or TVM

Table 11 mIoU on BDD100K validation set and inference speed (FPS) on an 1080Ti GPU with TensorRT

Method	mIoU (%)	FPS
DRN-D-38	55.2	12.9
DRN-D-22	53.2	21.0
FasterSeg	55.1	318.0
LPS-Net-S	55.9	634.7
LPS-Net-M	58.1	495.0
LPS-Net-L	59.3	283.3

For efficient object detection, we build the network following CenterNet (Zhou et al., 2019) by using LPS-Net-L as the backbone. To align the number of output channels of LPS-Net-L with that of the original CenterNet backbone, we additionally append two 3×3 Conv at the end of LPS-Net-L. The performance is evaluated on the COCO dataset, which contains 118K, 5K, and 20K images for training, validation, and testing, respectively. Our remoulded LPS-Net-L achieves 28.9% average precision over all IoU thresholds on the validation set, which is higher than 28.1% reported by (Zhou et al., 2019) using ResNet-18 as the backbone network. Note that the input images are not augmented during inference. Using the input resolution 512×512, the remoulded LPS-Net-L attains 442 FPS on one 1070Ti GPU with TensorRT, which is faster than ResNet-18 (401 FPS).

Table 12 mIoU performance on Cityscapes validation/testing set and inference speed (FPS) on two embedded devices

Method	Resolution	mIoU (%)		FPS	
		val	test	Nano	TX2
DF1-Seg-d8	1024x2048	72.4	71.4	3.2	7.7
FasterSeg	1024x2048	73.1	71.5	4.0	10.1
BiSeNetV2	512x1024	73.4	72.6	4.3	10.3
STDC1Seg-50	512x1024	72.2	71.9	7.1	17.1
LPS-Net-S	1024x2048	73.9	73.4	10.0	25.6

Table 13 Comparisons of performance and inference speed (FPS) for LPS-Net on two object detection tasks

Object Detection (CenterNet)			Salient Object Detection		
Backbone	AP (%)	FPS	Model	MAE	FPS
ResNet-18	28.1	401	U^2-Net†	0.054	107
LPS-Net-L	28.9	442	LPS-Net-L	0.048	264

For the salient object detection task which aims at segmenting the most visually attractive objects in an image, we follow the idea of U^2-Net (Qin et al., 2020) and extend our LPS-Net-L to a U-Net like encoder-decoder architecture with several additional 3×3 convolutions. Following Qin et al. (2020), we train our network on DUTS-TR (10,553 images) and evaluate it on DUTS-TE (5,019 images), both of which are subsets of the DUTS dataset (Wang et al., 2017). We report the Mean Absolute Error (MAE) as the performance metric and measure the inference speed (FPS) on one 1070Ti GPU with TensorRT. Comparing to the small version U^2-Net† that achieves 0.054 MAE at 107 FPS, our extended LPS-Net-L yields better tradeoff (0.048 MAE at 264 FPS).

Such experimental results on the tasks of object detection and salient object detection basically verify the applicability of LPS-Net when being generalized to other vision tasks with the large-resolution inputs.

5 Conclusion and Discussion

We have presented Lightweight and Progressively-Scalable Networks (LPS-Net) which explores the economic design and progressively scales up the network for efficient semantic segmentation. Particularly, we analyze the basic convolutional block and the way of path interaction in multi-path framework which could affect the accuracy-latency tradeoff for semantic segmentation. Empirically, we suggest to use 3 × 3 Conv in convolutional blocks and bilinear interpolation to implement interactions across paths. Based on these guidelines, we first build a tiny network and then extends the tiny network to a series of larger ones through expanding a certain dimension of Width, Depth or Resolution at one time. Experiments conducted on three datasets, i.e., Cityscapes, CamVid and BDD100K, validate our proposal and analysis on both GPUs and embedded devices. More remarkably, our LPS-Net shows impressive accuracy-latency tradeoff on Cityscapes: 413.5 FPS on an NVIDIA GTX 1080Ti GPU or 25.6 FPS on an NVIDIA TX2, with mIoU of 73.4% on testing set.

Data Availability The image data that support the findings of this study are available in Cityscapes (Cordts et al., 2016) (https://www.cityscapes-dataset.com/), BDD100K (Yu et al., 2018b) (https://bdd-data.berkeley.edu/), CamVid (Brostow et al., 2008) (http://mi.eng.cam.ac.uk/research/projects/VideoRec/CamVid/), COCO (Lin et al., 2014) (https://cocodataset.org/), and DUTS (Wang et al., 2017) (http://saliencydetection.net/duts/).

References

Brostow, G. J., Shotton, J., Fauqueur, J., & Cipolla, R. (2008). Segmentation and recognition using structure from motion point clouds. *European Conference on Computer Vision (ECCV), 5302*, 44–57. https://doi.org/10.1007/978-3-540-88682-2_5

Chen, L.C., Yang, Y., Wang, J., Xu, W., & Yuille, A.L. (2016). Attention to scale: Scale-aware semantic image segmentation. In *2016 IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 3640–3649).https://doi.org/10.1109/CVPR.2016.396.

Chen, L. C., Collins, M., Zhu, Y., Papandreou, G., Zoph, B., Schroff, F., Adam, H., & Shlens, J. (2018). Searching for efficient multi-scale architectures for dense image prediction. *Advances in Neural Information Processing Systems (NeurIPS), 31*, 8713–8724.

Chen, L. C., Papandreou, G., Kokkinos, I., Murphy, K., & Yuille, A. L. (2018). Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 40*(4), 834–848. https://doi.org/10.1109/TPAMI.2017.2699184

Chen, L. C., Zhu, Y., Papandreou, G., Schroff, F., & Adam, H. (2018). Encoder-decoder with atrous separable convolution for semantic image segmentation. *European Conference on Computer Vision (ECCV), 11211*, 801–818. https://doi.org/10.1007/978-3-030-01234-2_49

Chen, W., Gong, X., Liu, X., Zhang, Q., Li, Y., & Wang, Z. (2020). Fasterseg: Searching for faster real-time semantic segmentation. In *International conference on learning representations (ICLR)*

Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In *2017 IEEE conference on computer vision and pattern recognition (CVPR)* (pp 1800–1807). https://doi.org/10.1109/CVPR.2017.195.

Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., & Schiele, B. (2016). The cityscapes dataset for semantic urban scene understanding. In *2016 IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 3213–3223). https://doi.org/10.1109/CVPR.2016.350.

Ding, M., Lian, X., Yang, L., Wang, P., Jin, X., Lu, Z., & Luo, P. (2021a). Hr-nas: Searching efficient high-resolution neural architectures with lightweight transformers. In *2021 IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 2982–2992). https://doi.org/10.1109/CVPR46437.2021.00300.

Ding, X., Zhang, X., Ma, N., Han, J., Ding, G., & Sun, J. (2021b). Repvgg: Making vgg-style convnets great again. In *2021 IEEE/CVF conference on computer vision and pattern recognition (CVPR)* (pp. 13728–13737). https://doi.org/10.1109/CVPR46437.2021.01352.

Fan, M., Lai, S., Huang, J., Wei, X., Chai, Z., Luo, J., & Wei, X. (2021) Rethinking bisenet for real-time semantic segmentation. In *2021 IEEE/CVF conference on computer vision and pattern recognition (CVPR)* (pp. 9711–9720). https://doi.org/10.1109/CVPR46437.2021.00959

Fu, J., Liu, J., Wang, Y., Li, Y., Bao, Y., Tang, J., & Lu, H. (2019) Adaptive context network for scene parsing. In 2019 IEEE/CVF international conference on computer vision (ICCV) (pp. 6747–6756). https://doi.org/10.1109/ICCV.2019.00685.

Ghiasi, G., & Fowlkesl, C. C. (2016). Laplacian pyramid reconstruction and refinement for semantic segmentation. *European Conference on Computer Vision (ECCV), 9907*, 519–534. https://doi.org/10.1007/978-3-319-46487-9_32

Gholami, A., Kwon, K., Wu, B., Tai, Z., Yue, X., Jin, P., Zhao, S., & Keutzer, K. (2018). Squeezenext: Hardware-aware neural network design. In *2018 IEEE/CVF conference on computer vision and pattern recognition workshops (CVPRW)* (pp. 1719–171909). https://doi.org/10.1109/CVPRW.2018.00215.

Goldberg, D.E., & Deb, K. (1990) A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of genetic algorithms* (pp 69–93). https://doi.org/10.1016/b978-0-08-050684-5.50008-2.

Han, K., Wang, Y., Tian, Q., Guo, J., Xu, C., & Xu, C. (2020) Ghostnet: More features from cheap operations. In *2020 IEEE/CVF conference on computer vision and pattern recognition (CVPR)* (pp. 1577–1586). https://doi.org/10.1109/CVPR42600.2020.00165.

Han, S., Mao, H., & Dally, W.J. (2016). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International conference on learning representations (ICLR)*.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 770–778). https://doi.org/10.1109/CVPR.2016.90.

Hong, Y., Pan, H., Sun, W., & Jia, Y. (2021) *Deep dual-resolution networks for real-time and accurate semantic segmentation of road scenes*. ArXiv:2101.06085.

Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). *Mobilenets: Efficient convolutional neural networks for mobile vision applications*. ArXiv:1704.04861

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems (NIPS), 25*, 1106–1114.

Ladický, L., Russell, C., Kohli, P., & Torr, P.H. (2009). Associative hierarchical crfs for object class image segmentation. In *2009 IEEE 12th international conference on computer vision* (pp. 739–746). https://doi.org/10.1109/ICCV.2009.5459248

Li, G., Yun, I., Kim, J., & Kim, J. (2019). Dabnet: Depth-wise asymmetric bottleneck for real-time semantic segmentation. In *British machine vision conference (BMVC)*

Li, H., Xiong, P., Fan, H., & Sun, J. (2019a). Dfanet: Deep feature aggregation for real-time semantic segmentation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 9514–9523). https://doi.org/10.1109/CVPR.2019.00975.

Li, P., Dong, X., Yu, X., & Yang, Y. (2020a) When humans meet machines: Towards efficient segmentation networks. In *British machine vision conference (BMVC)*

Li, X., Zhou, Y., Pan, Z., & Feng, J. (2019b) Partial order pruning: For best speed/accuracy trade-off in neural architecture search. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 9137–9145). https://doi.org/10.1109/CVPR.2019.00936.

Li, X., You, A., Zhu, Z., Zhao, H., Yang, M., Yang, K., Tan, S., & Tong, Y. (2020). Semantic flow for fast and accurate scene parsing. *European Conference on Computer Vision (ECCV), 12346*, 775–793. https://doi.org/10.1007/978-3-030-58452-8_45

Li, X., Zhang, J., Yang, Y., Cheng, G., Yang, K., Tong, Y., & Tao, D. (2022) *Sfnet: Faster, accurate, and domain agnostic semantic segmentation via semantic flow*. ArXiv:2207.04415

Lin, D., Shen, D., Shen, S., Ji, Y., Lischinski, D., Cohen-Or, D., & Huang, H. (2019) Zigzagnet: Fusing top-down and bottom-up context for object segmentation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 7482–7491). https://doi.org/10.1109/CVPR.2019.00767.

Lin, P., Sun, P., Cheng, G., Xie, S., Li, X., & Shi, J. (2020) Graph-guided architecture search for real-time semantic segmentation. In *2020 IEEE/CVF conference on computer vision and pattern recognition (CVPR)* (pp. 4202–4211). https://doi.org/10.1109/CVPR42600.2020.00426.

Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., & Zitnick, C. L. (2014). Microsoft coco: Common objects in context. *European Conference on Computer Vision (ECCV), 8693*, 740–755. https://doi.org/10.1007/978-3-319-10602-1_48

Liu, C., Zoph, B., Shlens, J., Hua, W., Li, L. J., Fei-Fei, L., Yuille, A. L., Huang, J., & Murphy, K. P. (2017). Progressive neural architecture search. *European Conference on Computer Vision (ECCV), 11205*, 19–35. https://doi.org/10.1007/978-3-030-01246-5_2

Liu, C., Chen, L.C., Schroff, F., Adam, H., Hua, W., Yuille, A.L., & Fei-Fei, L. (2019a) Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *2019 IEEE/CVF conference on computer vision and pattern recognition (CVPR)* (pp. 82–92). https://doi.org/10.1109/CVPR.2019.00017.

Liu, H., Simonyan, K., & Yang, Y. (2019b). DARTS: Differentiable architecture search. In *International conference on learning representations (ICLR)*

Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *2015 IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 3431–3440). https://doi.org/10.1109/CVPR.2015.7298965.

Ma, N., Zhang, X., Zheng, H. T., & Sun, J. (2018). Shufflenet v2: Practical guidelines for efficient CNN architecture design. *European Conference on Computer Vision (ECCV), 11218*, 116–131. https://doi.org/10.1007/978-3-030-01264-9_8

Nirkin, Y., Wolf, L., & Hassner, T. (2021) Hyperseg: Patchwise hypernetwork for real-time semantic segmentation. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 4060–4069). https://doi.org/10.1109/CVPR46437.2021.00405.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., & Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)* (Vol. 32, pp. 8024–8035).

Peng, C., Zhang, X., Yu, G., Luo, G., & Sun, J. (2017) Large kernel matters—Improve semantic segmentation by global convolutional network. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 1743–1751). https://doi.org/10.1109/CVPR.2017.189.

Poudel, R.P., Liwicki, S., & Cipolla, R. (2019) Fast-scnn: Fast semantic segmentation network. In *British Machine Vision Conference (BMVC)*

Qin, X., Zhang, Z., Huang, C., Dehghan, M., Zaiane, O. R., & Jagersand, M. (2020). U2-net: Going deeper with nested u-structure for salient object detection. *Pattern Recognition, 106*, 107404. https://doi.org/10.1016/j.patcog.2020.107404

Qiu, Z., Yao, T., & Mei, T. (2017). Learning deep spatio-temporal dependence for semantic video segmentation. *IEEE Transactions on Multimedia, 20*(4), 939–949. https://doi.org/10.1109/TMM.2017.2759504

Qiu, Z., Yao, T., Zhang, Y., Zhang, Y., & Mei, T. (2019) *Scheduled differentiable architecture search for visual recognition*. ArXiv:1909.10236

Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y.L., Tan, J., Le, Q.V., & Kurakin, A. (2017). Large-scale evolution of image classifiers. In *34th International Conference on Machine Learning (ICML)* (Vol. 70, pp. 2902–2911).

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. S., Berg, A. C., & Fei-Fei, L. (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV), 115*(3), 211–252. https://doi.org/10.1007/s11263-015-0816-y

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.C. (2018) Mobilenetv2: Inverted residuals and linear bottlenecks. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 4510–4520). https://doi.org/10.1109/CVPR.2018.00474

Si, H., Zhang, Z., & Lu, F. (2020). Real-time semantic segmentation via multiply spatial fusion network. In *British Machine Vision Conference (BMVC)*

Sun, P., Wu, J., Li, S., Lin, P., Huang, J., & Li, X. (2021). Real-time semantic segmentation via auto depth, downsampling joint decision and feature aggregation. *International Journal of Computer Vision (IJCV), 129*(5), 1506–1525. https://doi.org/10.1007/s11263-021-01433-3

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016) Rethinking the inception architecture for computer vision. In *2016 IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 2818–2826). https://doi.org/10.1109/CVPR.2016.308.

Tan, M., & Le, Q.V. (2019) Efficientnet: Rethinking model scaling for convolutional neural networks. In *36th International Conference on Machine Learning (ICML)* (Vol 97, pp. 6105–6114)

Tao, A., Sapra, K., & Catanzaro, B. (2020) *Hierarchical multi-scale attention for semantic segmentation*. ArXiv:2005.10821

Wang, J., Sun, K., Cheng, T., Jiang, B., Deng, C., Zhao, Y., Liu, D., Mu, Y., Tan, M., Wang, X., Liu, W., & Xiao, B. (2021). Deep high-resolution representation learning for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 43*(10), 3349–3364. https://doi.org/10.1109/TPAMI.2020.2983686

Wang, L., Lu, H., Wang, Y., Feng, M., Wang, D., Yin, B., & Ruan, X. (2017). Learning to detect salient objects with image-level supervision. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 136–145). https://doi.org/10.1109/CVPR.2017.404.

Wu, B., Li, C., Zhang, H., Dai, X., Zhang, P., Yu, M., Wang, J., Lin, Y., & Vajda, P. (2021a). *Fbnetv5: Neural architecture search for multiple tasks in one run*. ArXiv:2111.10007

Wu, Y., Li, X., Shi, C., Tong, Y., Hua, Y., Song, T., Ma, R., & Guan, H. (2021b). Fast and accurate scene parsing via bi-direction alignment networks. In *2021 IEEE International Conference on Image Processing (ICIP)* (pp. 2508–2512). https://doi.org/10.1109/ICIP42928.2021.9506720

Yu, C., Wang, J., Peng, C., Gao, C., Yu, G., & Sang, N. (2018). Bisenet: Bilateral segmentation network for real-time semantic segmentation. *European Conference on Computer Vision (ECCV), 11217*, 325–341. https://doi.org/10.1007/978-3-030-01261-8_20

Yu, C., Gao, C., Wang, J., Yu, G., Shen, C., & Sang, N. (2021). Bisenet v2: Bilateral network with guided aggregation for real-time semantic segmentation. *International Journal of Computer Vision (IJCV), 129*(11), 3051–3068. https://doi.org/10.1007/s11263-021-01515-2

Yu, C., Xiao, B., Gao, C., Yuan, L., Zhang, L., Sang, N., & Wang, J. (2021b) Lite-hrnet: A lightweight high-resolution network. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 10435–10445). https://doi.org/10.1109/CVPR46437.2021.01030.

Yu, F., Koltun, V., & Funkhouser, T. (2017) Dilated residual networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 636–644). https://doi.org/10.1109/CVPR.2017.75.

Yu, F., Xian, W., Chen, Y., Liu, F., Liao, M., Madhavan, V., & Darrell, T. (2018b) *Bdd100k: A diverse driving video database with scalable annotation tooling*. ArXiv:1805.04687

Yuan, Y., Huang, L., Guo, J., Zhang, C., Chen, X., & Wang, J. (2021). Ocnet: Object context for semantic segmentation. *International Journal of Computer Vision (IJCV), 129*(8), 2375–2398. https://doi.org/10.1007/s11263-021-01465-9

Zhang, J., Pan, Y., Yao, T., Zhao, H., & Mei, T. (2019a) dabnn: A super fast inference framework for binary neural networks on arm devices. In *Proceedings of the 27th ACM international conference on multimedia* (pp. 2272–2275). https://doi.org/10.1145/3343031.3350534.

Zhang, X., Zhou, X., Lin, M., & Sun, J. (2018) Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 6848–6856). https://doi.org/10.1109/CVPR.2018.00716

Zhang, X., Xu, H., Mo, H., Tan, J., Yang, C., Wang, L., & Ren, W. (2021) Dcnas: Densely connected neural architecture search for semantic image segmentation. In *2021 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 13956–13967). https://doi.org/10.1109/CVPR46437.2021.01374.

Zhang, Y., Qiu, Z., Liu, J., Yao, T., Liu, D., & Mei, T. (2019b) Customizable architecture search for semantic segmentation. In *2019 IEEE/CVF conference on computer vision and pattern recognition (CVPR)* (pp 11633–11642). https://doi.org/10.1109/CVPR.2019.01191.

Zhao, H., Shi, J., Qi, X., Wang, X., & Jia, J. (2017). Pyramid scene parsing network. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 6230–6239). https://doi.org/10.1109/CVPR.2017.660.

Zhao, H., Qi, X., Shen, X., Shi, J., & Jia, J. (2018). Icnet for real-time semantic segmentation on high-resolution images. *European Conference on Computer Vision (ECCV), 11207*, 405–420. https://doi.org/10.1007/978-3-030-01219-9_25

Zhou, X., Wang, D., & Krähenbühl, P. (2019) *Objects as points*. ArXiv:1904.07850

Zoph, B., & Le, Q.V. (2017) Neural architecture search with reinforcement learning. In *International Conference on Learning Representations (ICLR)*

Zoph, B., Vasudevan, V., Shlens, J., & Le, Q.V. (2018) Learning transferable architectures for scalable image recognition. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 8697–8710). https://doi.org/10.1109/CVPR.2018.00907.