International Journal of Computer Vision (2021) 129:1506–1525
https://doi.org/10.1007/s11263-021-01433-3

Real-Time Semantic Segmentation via Auto Depth, Downsampling Joint Decision and Feature Aggregation

Peng Sun[1] · Jiaxiang Wu[2] · Songyuan Li[1] · Peiwen Lin[3] · Junzhou Huang[4] · Xi Li[1,5]

Received: 20 February 2020 / Accepted: 6 January 2021 / Published online: 19 February 2021

Abstract
To satisfy the stringent requirements for computational resources in the field of real-time semantic segmentation, most approaches focus on the hand-crafted design of light-weight segmentation networks. To enjoy the ability of model auto-design, Neural Architecture Search (NAS) has been introduced to search for the optimal building blocks of networks automatically. However, the network depth, downsampling strategy, and feature aggregation method are still set in advance and nonadjustable during searching. Moreover, these key properties are highly correlated and essential for a remarkable real-time segmentation model. In this paper, we propose a joint search framework, called AutoRTNet, to automate all the aforementioned key properties in semantic segmentation. Specifically, we propose hyper-cells to jointly decide the network depth and the downsampling strategy via a novel cell-level pruning process. Furthermore, we propose an aggregation cell to achieve automatic multi-scale feature aggregation. Extensive experimental results on Cityscapes and CamVid datasets demonstrate that the proposed AutoRTNet achieves the new state-of-the-art trade-off between accuracy and speed. Notably, our AutoRTNet achieves 73.9% mIoU on Cityscapes and 110.0 FPS on an NVIDIA TitanXP GPU card with input images at a resolution of 768 $\times$ 1536.

Keywords Real-time semantic segmentation · Neural architecture search · Computer vision · Deep learning

Communicated by Antonio Torralba.

✉ Xi Li
xilizju@zju.edu.cn

Peng Sun
sunpeng1996@zju.edu.cn

Jiaxiang Wu
jonathanwu@tencent.com

Songyuan Li
leizungjyun@zju.edu.cn

Peiwen Lin
linpeiwen@sensetime.com

Junzhou Huang
jzhuang@uta.edu

1 College of Computer Science and Technology, Zhejiang University, Hangzhou, China

2 Tencent AI Lab, Shenzhen, China

3 Sensetime Research, Beijing, China

4 University of Texas at Arlington, Arlington, TX, USA

5 Shanghai Institute for Advanced Study, Zhejiang University, Shanghai, China

1 Introduction

Semantic segmentation, a fundamental topic in computer vision, aims at assigning per-pixel semantic labels for images. Recent approaches (Zhao et al. 2017; Chen et al. 2017, 2018b; Zhao et al. 2018b) based on fully convolutional networks (Long et al. 2015) have achieved remarkable accuracy on public benchmarks (Brostow et al. 2008; Cordts et al. 2016; Everingham et al. 2015). Such improvements, however, come at the cost of deeper and less efficient networks, which may not be applicable to many real-time systems, e.g., autonomous driving and video surveillance.

To perform fast semantic segmentation with satisfactory accuracy, the design philosophy of real-time segmentation network architectures mainly concentrates on three aspects: (1) building block design (Li and Kim 2019; Paszke et al. 2016), which considers the block-level feature representation capacity, computational complexity, and receptive field size; (2) network depth and downsampling strategy (Li and Kim 2019; Li et al. 2019a), which directly affect the accuracy and speed of a network, hence real-time networks favor shallow layers and fast downsampling; and (3) feature aggregation (Yu et al. 2018; Zhao et al. 2018a), which fuses multi-scale features to compensate the loss of spatial details caused by fast downsampling.

The above hand-crafted networks make huge progress, while they require expertise in architecture design based on laborious trial and error. To relieve this burden, some researchers introduce neural architecture search (NAS) methods (Baker et al. 2016; Zoph and Le 2016; Liu et al. 2019b; Xie et al. 2019) into this field, and obtain excellent results (Chen et al. 2018a; Liu et al. 2019a; Zhang et al. 2019b; Nekrasov et al. 2019). Liu et al. (2019a) and Chen et al. (2018a) focus on high-quality segmentation instead of real-time applications. To meet the real-time demand, Zhang et al. (2019b) search a customized architecture by introducing a latency loss function. Although its building blocks are searched, the network depth, downsampling strategy, and feature aggregation method are still set by hand in advance and nonadjustable during searching. Since these three aspects are highly correlated and indispensable for a remarkable real-time segmentation network, the fact that they are nonadjustable increases the difficulty of finding an optimal real-time architecture (i.e. the best trade-off between accuracy and speed). These motivate us to explore all the aspects automatically during the searching process.

In this paper, we propose a joint search framework to search for the optimal building blocks, network depth, downsampling strategy, and feature aggregation method simultaneously. Specifically, we propose hyper-cells to decide the network depth and the downsampling strategy jointly and automatically via a cell-level pruning process. Moreover, we propose an aggregation cell to fuse features from multiple spatial scales automatically. As for the hyper-cell, we introduce a novel learnable architecture parameter. Thus, the network depth and downsampling strategy are fully determined concurrently according to the optimized architecture parameters. As for the aggregation cell, we aggregate multi-level features in the network automatically to fuse the low-level spatial details and high-level semantic context effectively.

We denote the resulting network as **Auto** searched **R**eal-**T**ime semantic segmentation network or **AutoRTNet**. We evaluate AutoRTNet on both Cityscapes (Cordts et al. 2016) and CamVid (Brostow et al. 2008) datasets. The experiments demonstrate the superiority of AutoRTNet, as shown in Fig. 1, where our AutoRTNet achieves the best accuracy-efficiency trade-off.

The main contributions can be summarized as follows:

– We propose a joint search framework for real-time semantic segmentation that automatically searches for the building blocks, network depth, downsampling strategy, and feature aggregation method simultaneously.
– We propose the hyper-cell to learn the network depth and downsampling strategy jointly and automatically via the cell-level pruning process, and the aggregation cell to achieve automatic multi-scale feature aggregation.
– Notably, AutoRTNet has achieved 73.9% mIoU on the Cityscapes test set and 110.0 FPS on an NVIDIA TitanXP GPU card with 768 × 1536 input images.

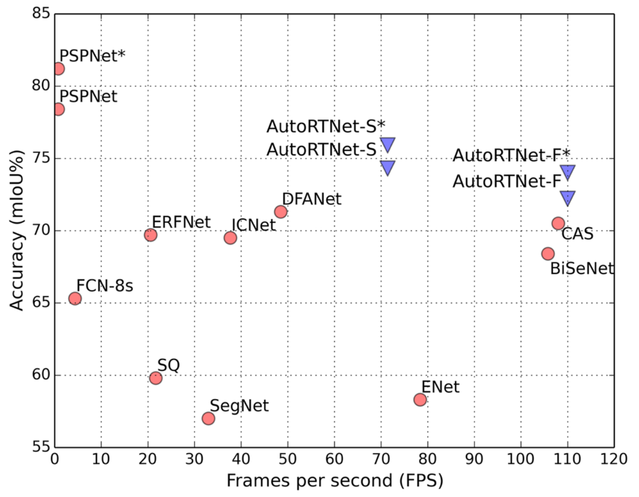

Fig. 1 The inference speed and accuracy for different networks on the Cityscapes test set. Compared with other methods, our AutoRTNet locates in the right-top since it features lower latency with comparable accuracy. Methods trained using both fine and coarse data are marked with ∗

2 Related Work

2.1 Semantic Segmentation

High-quality segmentation FCN (Long et al. 2015) is the pioneer work which has greatly promoted the development of semantic segmentation. Extensions to FCN follow many directions. Encoder–decoder structures (Badrinarayanan et al. 2017; Lin et al. 2017a; Noh et al. 2015) combine low-level and high-level features to improve the accuracy of semantic segmentation. DRN (Yu et al. 2017) and DeepLab (Chen et al. 2017, 2018b) use dilated convolution operations to effectively enlarge the receptive field size. To capture multi-scale context information, DeepLabV3 (Chen et al. 2017) and PSPNet (Zhao et al. 2017) propose the pyramid modules. Recently, attention mechanism (Vaswani et al. 2017) has been used in segmentation methods (Fu et al. 2019; Zhang et al. 2019a; Zhao et al. 2018b; Li et al. 2018). These outstanding works are designed for high-quality segmentation, which is inapplicable to real-time applications.
Real-time methods Various algorithms have been proposed for real-time semantic segmentation. Some works (Wu et al. 2017) reduce the computation overheads via restricting the size of input images. Channel-pruning algorithms (Paszke

et al. 2016; Badrinarayanan et al. 2017) are introduced to boost the inference speed, and most real-time methods focus on designing the light-weight and effective network architectures. The design philosophy of real-time network architectures mainly can be summarized in the following three aspects. And in our work, we fully explore all three aspects simultaneously.

Building block design The building block design (Paszke et al. 2016; Romera et al. 2017; Mehta et al. 2018; Li and Kim 2019) requires researchers to give sufficient consideration to the computational complexity, feature representation capacity, and receptive field size, which is essential for real-time semantic segmentation. For example, ENet (Paszke et al. 2016) and DABNet (Li and Kim 2019) propose light-weight blocks and stack them with different dilation rates to form a whole network. MobileNet and its variants (Howard et al. 2017; Sandler et al. 2018) use blocks with depth-wise separable convolution in pursuit of light-weight models.

Network depth and downsampling strategy High-quality segmentation networks always use the pre-defined backbones, e.g. ResNet (He et al. 2016), Xception (Chollet 2017), as encoders. However, for real-time segmentation networks, [e.g. DABNet (Li and Kim 2019), DFANet (Li et al. 2019a), ERFNet (Romera et al. 2017)], the network depth and downsampling strategy (i.e. how many layers in each stage) are determined mostly by hand as they directly affect the accuracy and speed of the networks. For pursuing the fast inference speed, real-time networks always enjoy shallow layers and perform fast downsampling with factor 16 or 32.

Feature aggregation The fast downsampling in real-time networks easily results in the loss of spatial details. Thus, multi-scale feature aggregation (Yu et al. 2018; Zhao et al. 2018a; Li et al. 2019a) has been proposed to remedy the loss of spatial details. Zhao et al. (2018a) propose an image cascade network with multi-scale inputs. Yu et al. (2018) decouple the network into context and spatial paths to make the right balance between accuracy and speed. Li et al. (2019a) aggregate multi-scale features from different layers to remedy the loss of spatial details.

2.2 Neural Architecture Search

Overview Neural architecture search (NAS) focuses on automating the network architecture design process. Early NAS methods are time-consuming (e.g. thousands of GPU days) and computationally expensive via reinforcement learning (Zoph and Le 2016; Baker et al. 2016; Zoph et al. 2018; Tan et al. 2019) or evolutionary algorithms (Miikkulainen et al. 2019; Real et al. 2019). Recently, the emergence of differentiable NAS methods (Liu et al. 2019b; Xie et al. 2019; Cai et al. 2018) has greatly relieved the time-consuming problem while achieving excellent performance. DARTs (Liu et al. 2019b) is the pioneer work for gradient-based NAS, and Liu et al. (2019b) propose an iterative optimization framework which is based on the continuous relaxation of architecture representation. Xie et al. (2019) constrain the architecture parameters to approximate one-hot, resolving the inconsistency in optimizing between the performance of derived child networks and converged parent networks. In addition, FBNet (Wu et al. 2019), Proxyless-NAS (Cai et al. 2018), and MnasNet (Tan et al. 2019) propose multi-objective optimization with the consideration of real-world latency.

In this paper, we propose hyper-cells to jointly decide the key properties (i.e. the downsampling strategy and the depth of a network) automatically in semantic segmentation. Searching at this network architecture level gives rise to the suitable downsampling strategy and depth for a semantic segmentation network. In contrast, DARTs (Liu et al. 2019b) and SNAS (Xie et al. 2019) only search at the cell level under a fixed network architecture without considering the intrinsic properties of semantic segmentation. Thus, the search spaces of ours and other NAS methods (Liu et al. 2019b; Xie et al. 2019) are essentially different.

NAS for segmentation DPC (Chen et al. 2018a) is the first work for dense image prediction using NAS methods and searches for a multi-scale representation module. A similar work to us is AutoDeepLab (Liu et al. 2019a), they propose a hierarchical search space and search for the downsampling path. Although they also search for the downsampling strategy, the mechanism is fundamentally different from ours. They design the network level continuous relaxation to learn the downsampling path, while we search for the downsampling strategy via the cell-level pruning progress. Moreover, they cannot search for the network depth and feature aggregation method, and focus on high-quality segmentation. For real-time requirements, CAS (Zhang et al. 2019b) searches for the architecture with customized resource constraints and achieves excellent real-time performance. However, our approach can search for the network depth, downsampling strategy, and feature aggregation method, which is significantly different from CAS (Zhang et al. 2019b).

NAS for object detection The combination of multi-scale features is also essential for object detection (Lin et al. 2017b; Liu et al. 2016). In the field of NAS, NAS-FPN (Ghiasi et al. 2019) and Auto-FPN (Xu et al. 2019) search for an architecture that merges features of varying dimensions and are successful at searching for the appropriate combination method. Unlike us, Ghiasi et al. (2019) propose the merging cell and use an RNN controller to select candidate feature layers and a binary operation in each merging cell. Their search space only consists of two binary operations, i.e. sum and global pooling for simplicity. Xu et al. (2019) search for an efficient feature fusion module, and their search space is specially designed for detection and flexible enough to cover

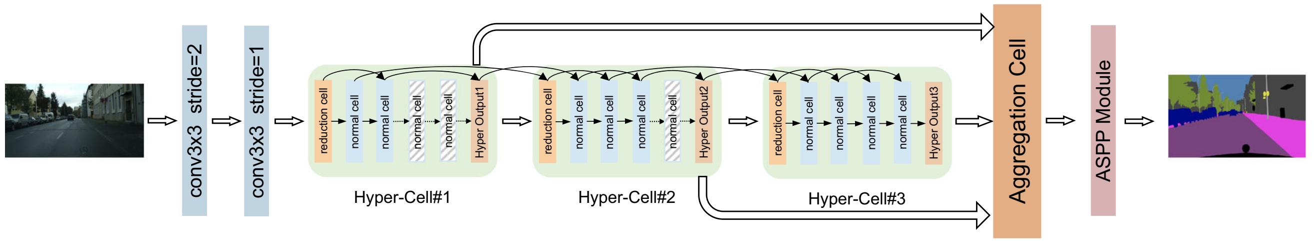

Fig. 2 Illustration of our joint network architecture search framework. The network begins with two convolution layers and contains three hyper-cells which search for the optimal network depth and downsampling strategy via the cell-level pruning process. Each hyper-cell contains a reduction cell and n normal cells. The cells marked with the dotted white line are pruned after optimization. The aggregation cell is designed to perform automatic multi-scale feature aggregation effectively, and it seamlessly integrates the outputs of hyper-cells

many popular designs of detectors. Thus, the search space design, motivation, and implementation of the above methods are significantly different from ours.

3 Methods

In this section, we illustrate the proposed real-time semantic segmentation network search framework in detail. First, we briefly introduce an overview of the proposed framework. Second, we describe the differentiable architecture search. Next, we elaborate on the proposed hyper-cell for joint network depth and downsampling search. Finally, we illustrate the proposed aggregation cell for automatic multi-scale feature aggregation.

3.1 Overview

The joint search framework is shown in Fig. 2. We propose the hyper-cell to search for the optimal network depth and downsampling strategy as they directly affect the accuracy and speed of a network. For remedying the loss of spatial details caused by fast downsampling, a novel aggregation cell is proposed for automatic multi-scale feature aggregation. The whole framework contains two pre-defined convolution layers, three hyper-cells, and an aggregation cell. The multi-scale module (Chen et al. 2017) is subsequently used to extract the global and local context for final prediction. For real-time demands, we take real-world latency into consideration during the searching process.

3.2 Differentiable Architecture Search

Intra-cell search space

The hyper-cell is the building block of the network, and the cell is the basic component unit of the hyper-cell, as shown in Fig. 2. There are two types of cells, i.e., normal cells and reduction cells (Liu et al. 2019b; Xie et al. 2019). The reduction cells reduce the feature map size by a factor of 2 for downsampling, and the factor is 1 in normal cells.

A cell is a directed acyclic graph (DAG) consisting of an ordered sequence of N nodes, denoted by $\mathcal{N} = \{x^{(1)}, \ldots, x^{(N)}\}$. Each node $x^{(i)}$ is a latent representation (i.e. feature map), and each directed edge (i, j) is associated with some candidate operations (e.g. conv, pooling) in operation set $\mathcal{O}^{(i,j)}$, representing all possible transformations from $x^{(i)}$ to $x^{(j)}$. Each cell has two inputs (the outputs of the previous two cells) and one output (the concatenation of all the intermediate nodes in the cell). The structure of cell is shown on the right in Fig. 3. Each intermediate node $x^{(j)}$ is computed based on all of its predecessors:

$$x^{(j)} = \sum_{i<j} \widetilde{o}^{(i,j)}\left(x^{(i)}\right), \tag{1}$$

where $\widetilde{o}^{(i,j)} \in \mathcal{O}^{(i,j)}$ is the optimal operation at edge (i, j).

In order to determine the optimal operation $\widetilde{o}^{(i,j)}$ at edge (i, j), we represent the intra-cell search space with a set of one-hot random variables from a fully factorizable joint distribution $p(M)$ (Xie et al. 2019). Specifically, each edge (i, j) is associated with a one-hot random variable $M^{(i,j)}$. We use $M^{(i,j)}$ as a mask to multiply all the candidate operations $\mathcal{O}^{(i,j)}$ at edge (i, j), and thus the intermediate node $x^{(j)}$ is given by:

$$x^{(j)} = \sum_{i<j} \sum_{o \in \mathcal{O}} m_o^{(i,j)} \cdot o^{(i,j)}\left(x^{(i)}\right), \tag{2}$$

where $m_o^{(i,j)} \in M^{(i,j)}$ and $m_o^{(i,j)}$ is a random variable in $\{0, 1\}$, it is evaluated to 1 if operation $o^{(i,j)}$ is selected.

To make $p(M)$ differentiable, we use Gumbel Softmax technique (Jang et al. 2016; Maddison et al. 2016) to relax the discrete sampling distribution to be continuous and differentiable:

$$M^{(i,j)} = f_{\alpha^{(i,j)}}(G^{(i,j)}) = \mathrm{softmax}((\log \alpha^{(i,j)} + G^{(i,j)})/\lambda), \tag{3}$$

where $M^{(i,j)}$ is the softened one-hot random variable for operation selection at edge (i, j), $\alpha^{(i,j)}$ is the intra-cell architecture parameter at edge (i, j), $G^{(i,j)} = -\log(-\log(U^{(i,j)}))$ is a vector of Gumbel random variables, $U^{(i,j)}$ is a uniform random variable in the range $(0, 1)$. λ is the temperature of softmax, and as λ approaches 0, $M^{(i,j)}$ approximately becomes one-hot. The technique of Gumbel Softmax makes the entire intra-cell search differentiable (Wu et al. 2018, 2019; Xie et al. 2019) to both network parameter w and architecture parameter α.

For the candidate operation set $\mathcal{O}$, we collect the operations as follows:

- zero operation
- skip connection
- 3 × 3 max pooling
- 3 × 3 conv
- 3 × 3 conv, repeat 2
- 3 × 3 separable conv
- 3 × 3 separable conv, repeat 2
- 3 × 3 dilated separable conv, dilation=2
- 3 × 3 dilated separable conv, dilation=4
- 3 × 3 dilated separable conv, dilation=2, repeat 2

Intra-cell latency cost

For the operation selection of cells of a real-time network, we take real-world latency into consideration. Specifically, we build a GPU-latency lookup table (Cai et al. 2018; Tan et al. 2019; Wu et al. 2019; Zhang et al. 2019b) that records the inference time cost of each candidate operation. The latency of each operation is measured in micro-second on a TitanXP GPU. During the searching process, we associate a cost $lat_o^{(i,j)}$ with each candidate operation $o^{(i,j)}$ at edge (i, j), thus the latency cost of cell p is formulated as:

$$lat_p = \sum_{(i,j)} \sum_{o \in \mathcal{O}} m_o^{(i,j)} \cdot lat_o^{(i,j)}, \tag{4}$$

where $m_o^{(i,j)} \in M^{(i,j)}$ and $M^{(i,j)}$ denotes the softened one-hot random variable at edge (i, j). By using the pre-built lookup table and above sampling process, the latency loss is also differentiable with respect to $m_o^{(i,j)}$.

3.3 Joint Network Depth and Downsampling Search

Hyper-cell search space

The network depth and downsampling strategy affect the accuracy and speed of a network directly in real-time semantic segmentation. To adjust them jointly and automatically, we formulate the two design-making processes as a single cell-level pruning process. Specifically, we propose a hyper-cell, as shown in Fig. 3, which consists of a reduction cell and n normal cells. We introduce $n + 1$ edges to connect each cell with the hyper-cell's output and associate the edges with

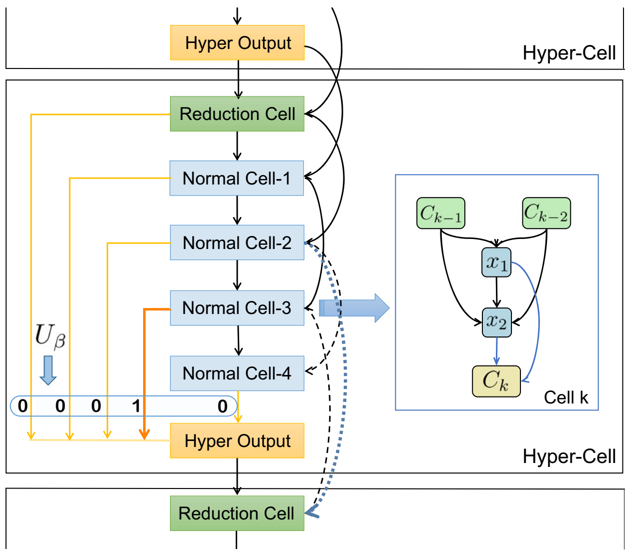

Fig. 3 Illustration of our hyper-cell. The hyper-cell consists of a reduction cell and n normal cells and $n + 1$ edges with architecture parameter which encodes the depth of the hyper-cell. The structure of cell is shown on the right in this figure

the learnable architecture parameter β. The intra-cell architecture parameters α of n normal cells are shared in the same hyper-cell.

We determine the depth of each hyper-cell by limiting that only one edge can be activated for each hyper-cell, and all cells behind this activated edge can be pruned safely. Each specific edge in hyper-cell s is associated with a one-hot random variable $U^s = (u_1^s, u_2^s, \ldots, u_{n+1}^s)$ from a fully factorizable joint distribution $P(U)$. The U^s works as a mask during the training process, and the output of the hyper-cell s is designed as:

$$HyperOut^{(s)} = \sum_{p=1}^{n+1} u_p{}^s \cdot (C_p{}^s), \tag{5}$$

where C_p^s is the output of p-th cell in hyper-cell s, u_p^s represents the random variable in $\{0, 1\}$ of p-th edge of hyper-cell s. We adopt the Gumbel Softmax based sampling process to make the training process differentiable:

$$U^s = f_{\beta^s}(G^s) = \mathrm{softmax}((\log \beta^s + G^s)/\lambda), \tag{6}$$

where U^s is the softened one-hot random variable for edge selection of hyper-cell s, β^s is the architecture parameter of hyper-cell s. G^s and λ are similar to the ones in Eq. (3). The hyper-cell architecture parameter β we introduced can be effectively optimized together with the network parameter w, intra-cell architecture parameter α in the same round of back-propagation. After stacking hyper-cells to form a whole network, the network depth and downsampling strategy can be fully explored concurrently according to the architecture parameter β.

To better explain the cell-level pruning process, we give an example as follows. In the initial phase, let's say we have five cells (one reduction cell and four normal cells) and each cell in hyper-cell keeps its original inputs and outputs. As shown in Fig. 3, if the fourth edge is activated currently (i.e. U is $\{0, 0, 0, 1, 0\}$), the Normal Cell-4 will be pruned in this iteration, and the output of this hyper-cell is the output of Normal Cell-3. At the same time, the reduction cell in the next hyper-cell $s + 1$ takes the outputs of hyper-cell s and Normal Cell-2 in hyper-cell s as its inputs, to stick to the "two-input" principle of the cell. The learning and adjusting like this go through the entire searching phase.

By introducing the architecture parameter β in the proposed hyper-cell, we can dynamically adjust and search for the network depth as well as the downsampling strategy for real-time semantic segmentation.

Network Latency Cost We define the set of cells in all hyper-cells in the initial phase as P, after optimization, the number of the set is reduced and the new set is marked as $\bar{P}$. For the current architecture (α, β) containing several hyper-cells, the total latency excludes the pruned cells and can be calculated as:

$$Lat(\alpha, \beta) = \sum_{p \in \bar{P}} lat_p, \tag{7}$$

where $\bar{P}$ is the set of remaining cells in all hyper-cells of architecture (α, β). The lat_p is the latency of cell p. We construct the latency loss function L_{Lat} as:

$$L_{lat} = \log(Lat(\alpha, \beta)). \tag{8}$$

Thus, the total loss function can be formulated as:

$$L_{total} = L_{CE} + \gamma \; L_{lat}, \tag{9}$$

where L_{CE} is the cross-entropy loss between the predictions of the architecture (α, β) with network weights w and the ground truth. L_{Lat} denotes the total latency loss of architecture (α, β). Moreover, γ controls the magnitude of latency term (i.e. balance the trade-off between accuracy and speed).

3.4 Network-Level Auto Feature Aggregation

For remedying the loss of spatial details in real-time segmentation networks due to fast downsampling, we propose the aggregation cell to automatically aggregate features by optimal operations from different levels in the network. The aggregation cell seamlessly integrates the outputs of the above hyper-cells, and the outputs of the early hyper-cells can compensate for the loss of spatial details.

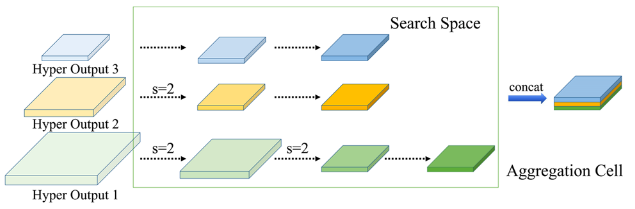

Fig. 4 Overview of the aggregation cell for automatic multi-scale feature aggregation. The aggregation cell contains E edges (dotted arrows), and each edge is equiped with some candidate operations. The "s = 2" means stride = 2

The structure of the proposed aggregation cell is shown in Fig. 4. The aggregation cell takes three hyper-cells' outputs with different resolutions as its inputs, and thus the aggregation cell is designed to combine multi-scale features (i.e. low-level spatial details and high-level semantic context). The aggregation cell is designed as a directed acyclic graph consisting of M nodes and E edges. Each node is a latent representation (i.e. feature map) and each directed edge is associated with some candidate operations. As shown in Fig. 4, each edge's stride is set to 1, unless explicitly specified by "s = 2" (stride 2), which acts as the downsampling connection. The output of the aggregation cell is designed as the concatenation of the final feature maps from three hyper-cells. We use the same sampling and optimization process as intra-cell search in Sect. 3.2 to optimize the aggregation cell's architecture parameter.

Given the candidate operation set, the aggregation cell also efficiently enlarges the receptive field of the network. For the operation set of the aggregation cell, we collect following 5 kinds of operations:

- 1×1 conv, repeat 2
- 3×3 conv, repeat 2
- 3×3 dilated separable conv, dilation=2, repeat 2
- 3×3 dilated separable conv, dilation=4, repeat 2
- 3×3 dilated separable conv, dilation=8, repeat 2

4 Experiments

To verify the effectiveness and superiority of our joint search framework, we compare our AutoRTNet with other state-of-the-art methods on two challenging benchmarks: Cityscapes (Cordts et al. 2016) and CamVid (Brostow et al. 2008). Moreover, we conduct a series of ablation studies to verify the effectiveness of the proposed hyper-cell and aggregation cell. Furthermore, we provide an in-depth analysis of the architecture searched by our framework. Finally, we give detailed quantitative results, visualization results, and adequate comparisons with other state-of-the-art methods.

4.1 Implementation Details

Searching For the searching process, the whole network contains three hyper-cells and the initial cell numbers in these hyper-cells are {5, 10, 10}, respectively. The intermediate node number of the cell is set to 2. The initial channel number is 8, and the channels are $\times 3$ when downsampling in reduction cells. The searching process, which is conducted on the Cityscapes dataset, runs 150 epochs with mini-batch size 16, which takes approximately 16 hours with 16 TitanXP GPU cards. Similar to FBNet (Wu et al. 2019), we postpone the training of the hyper-cell architecture parameters β by 50 epochs to warm-up network weights w and intra-cell architecture parameters α. The α and β are optimized by Adam, with an initial learning rate of 0.001, a momentum (0.5, 0.999) and a weight decay 1e-4. The w is optimized using SGD with a momentum 0.9, a weight decay 1e-3, and the cosine learning scheduler that decays learning rate from 0.025 to 0.001. For Gumbel Softmax, we set the initial temperature λ in equation (3) and (6) as 3.0 empirically, and gradually decrease to the minimum value of 0.03. We set the node number M and edge number E as 7 in the aggregation cell.

Retraining When the searching process is over, the searched network is firstly pretrained on the ImageNet dataset from scratch. We then finetune the network on the specific segmentation dataset (i.e. Cityscapes or CamVid) for 200 epochs with mini-batch size 16. The base learning rate is 0.01 and the 'poly' learning rate policy is adopted with a power 0.9, together with a momentum 0.9 and a weight decay 0.0005. Following (Wu et al. 2016; Yu et al. 2018), we compute the loss function with the online bootstrapping strategy. Data augmentation contains random horizontal flip, random resizing with scale ranges in [0.5, 2.0], and random cropping into fix size for training.

4.2 Benchmarks and Evaluation Metrics

Cityscapes (Cordts et al. 2016), a public street scene dataset, contains high quality pixel-level annotations of 5000 images with size 1024 $\times$ 2048 and 19,998 images with coarse annotations. 19 semantic classes are used for training and evaluation. CamVid (Brostow et al. 2008) is another public dataset, and it contains 701 images in total. We follow the training/testing set split in (Zhang et al. 2019b; Brostow et al. 2008), with 468 training and 233 testing labeled images. These images are densely labeled with 11 semantic class labels. We use three evaluation metrics, including the mean of class-wise intersection over union (mIoU), network forward time (Latency), and Frames Per Second (FPS).

4.3 Real-Time Semantic Segmentation Results

In this section, we compare our AutoRTNet with other real-time segmentation methods. We run all experiments based on Pytorch 0.4 (Paszke et al. 2017) and measure the latency on an NVIDIA TitanXP GPU card under CUDA 9.0. For a fair comparison, we directly quote the reported remeasured or estimated speed results on TitanXP of other algorithms mentioned in (Zhang et al. 2019b; Orsic et al. 2019). For the AutoRTNet, we report the average inference time through 500 times. In this process, we don't employ any test augmentation.

Results on Cityscapes.

We conduct the searching process with latency term weight γ 0.01 and 0.001, and obtain the relatively fast and slow networks named AutoRTNet-F and AutoRTNet-S, respectively. We evaluate them on the Cityscapes test set. The validation set is added for training before submitting to the online Cityscapes server. Following (Zhang et al. 2019b; Yu et al. 2018), we scale the resolution of the images from 1024 $\times$ 2048 to 768 $\times$ 1536 as inputs to measure the speed and accuracy. As shown in Table 1, our AutoRTNet achieves the best trade-off between accuracy and speed. AutoRTNet-F yields 72.2% mIoU while maintaining 110.0 FPS on the Cityscapes test set with only fine data and without any test augmentation. When the coarse data is added to the training set, the mIoU achieves 73.9%, which is the state-of-the-art trade-off for real-time semantic segmentation. Compared with BiseNet (Yu et al. 2018) and CAS (Zhang et al. 2019b) which have a comparable speed to us, AutoRTNet-F surpasses them by 3.8% and 1.7% in mIoU on the Cityscapes test set, respectively. Compared with other real-time segmentation methods (e.g. ENet (Paszke et al. 2016), ICNet (Zhao et al. 2018a)), our AutoRTNet-F surpasses them in both speed and accuracy by a large margin. Moreover, our AutoRTNet-S achieves 74.3% and 75.8% mIoU (+ coarse data) on the Cityscapes test set with 71.4 FPS, which is also the state-of-the-art real-time performance.

Results on CamVid To validate the transferability of the networks searched by our framework, we directly transfer AutoRTNet-F and AutoRTNet-S, which are obtained on Cityscapes, to the CamVid dataset, as reported in Table 2. We only transfer the network architectures and train them on CamVid from scratch. With 720 $\times$ 960 input images, AutoRTNet-F achieves 73.5% mIoU on the CamVid test set with 140.0 FPS, which is the state-of-the-art trade-off between accuracy and speed. AutoRTNet-S achieves 74.2% mIoU with 82.5 FPS. We also conduct the architecture search on CamVid ($\gamma = 0.1$) and name the resulting ultra-fast network AutoRTNet-U. Notably, AutoRTNet-U achieves appealing 250.0 FPS while maintaining 68.6% mIoU on the CamVid test set, which surpasses ICNet (Zhao et al. 2018a)

Table 1 Accuracy and speed comparison of our method against other state-of-the-art methods on the Cityscapes test set

Method	Input size	mIoU (%)	Latency (ms)	FPS
FCN-8S 2015	512 × 1024	65.3	227.23	4.4
PSPNet 2017	713 × 713	78.4	1288.0	0.78
PSPNet* 2017	713 × 713	81.2	1288.0	0.78
DeepLabV3* 2017	769 × 769	81.3	769.23	1.3
AutoDeepLab* †2019a	769 × 769	81.2	303.0	3.3
SegNet 2017	360 × 640	57.0	30.3	33
ENet 2016	360 × 640	58.3	12.7	78.4
SQ 2016	1024 × 2048	59.8	46.0	21.7
ERFNet 2017	512 × 1024	69.7	48.5	20.6
ICNet 2018a	1024 × 2048	69.5	26.5	37.7
DF1-Seg 2019b	768 × 1536	73.0	29.1	34.4
SwiftNet 2019	1024 × 2048	75.1	26.2	38.1
ESPNet 2018	512 × 1024	60.3	8.2	121.7
DFANet 2019a	1024 × 1024	71.3	10.0	100.0
DFANet † 2019a	1024 × 1024	71.3	20.6 †	48.5 †
BiSeNet 2018	768 × 1536	68.4	9.52	105.8
CAS 2019b	768 × 1536	70.5	9.25	108.0
CAS* 2019b	768 × 1536	72.3	9.25	108.0
AutoRTNet-F	768 × 1536	72.2	9.09	110.0
AutoRTNet-F*	768 × 1536	73.9	9.09	110.0
AutoRTNet-S	768 × 1536	74.3	14.0	71.4
AutoRTNet-S*	768 × 1536	75.8	14.0	71.4

Methods trained using both fine and coarse data are marked with $*$. The mark † represents the speed is remeasured by us on Titan XP

Table 2 Results on the CamVid test set with resolution 720 × 960

Method	mIoU (%)	Latency(ms)	FPS	Parameters (M)
SegNet 2017	55.6	34.01	29.4	29.5
ENet 2016	51.3	16.33	61.2	0.36
ICNet 2018a	67.1	28.98	34.5	26.5
BiSeNet 2018	65.6	–	–	5.8
DFANet 2019a	64.7	8.33	120	7.8
CAS 2019b	71.2	5.92	169	–
AutoRTNet-F	73.5	7.14	140.0	2.5
AutoRTNet-S	74.2	12.1	82.5	3.9
AutoRTNet-U	68.6	4.0	250.0	1.4

(67.1% mIoU with 34.5 FPS) and DFANet (Li et al. 2019a) (64.7% mIoU with 120 FPS) significantly.

Parameter results Many computationally limited mobile platforms have restrictive memory constraints for real-time applications, and thus the parameter size is also an important consideration. Table 2 shows the results of our AutoRTNet and other methods on the CamVid test set. With only 2.5 million parameters, our AutoRTNet-F achieves impressive accuracy (i.e. 73.5% mIoU) on the CamVid test set, which significantly outperforms existing real-time segmentation networks. The parameter sizes of AutoRTNet-S and AutoRTNet-U are 3.9M and 1.4M, respectively.

4.4 Ablation Study

The contribution of each component is investigated in the following ablation studies on the Cityscapes validation set. The latency term weight γ in Eq. (7) is set to 0.01 and all networks are firstly pretrained on ImageNet in following experiments for a fair comparison, if not specially noted.

Table 3 Comparison with random search on the Cityscapes validation set

Method	mIoU (%)	Latency (ms)	FPS
AutoRTNet	72.9	9.09	110.0
Random search	66.7 ± 2.5	11.4 ∼ 16.2	87.5 ∼ 61.2

Table 4 The optimization results of hyper-cells with different initial states and different random seeds

Random seed setting	Initial phase {a, b, c}	After optimization $\{\overline{a}, \overline{b}, \overline{c}\}$	mIoU (%)	Frames Per Second (FPS)
2	{5, 10, 10}	{2, 4, 6}	72.9	110.0
2	{5, 10, 15}	{2, 4, 6}	73.0	106.5
2	{5, 15, 15}	{1, 4, 6}	72.5	102.8
2	{10, 10, 10}	{2, 4, 6}	72.8	112.3
1	{5, 10, 10}	{2, 4, 6}	72.3	113.9
3	{5, 10, 10}	{1, 4, 7}	72.8	107.9

Table 5 Comparison to random downsampling strategy

Downsampling positions	mIoU (%)	FPS	Downsampling design rule
(1, 3, 7)	72.9	110.0	Hyper cell
(1, 5, 10)	71.6	90.2	Random
(3, 5, 8)	72.5	75.7	Random
(1, 2, 4)	68.4	125	Random

4.4.1 Comparison with Random Search

As discussed in (Li and Talwalkar 2019; Yu et al. 2019), NAS is a specialized hyper-parameter optimization problem, and random search is a competitive baseline for the problem. We apply random search to semantic segmentation by randomly sampling ten architectures from our previously-defined search space. The whole search space contains intra-cell operation selection and hyper-cell depth decision, which is significantly challenging for random search to find a satisfactory network. As shown in Table 3, random search achieves the average 66.7% mIoU ± 2.5% on the Cityscapes validation set with ImageNet pretrained, which is substantially lower than our AutoRTNet. The results also demonstrate the effectiveness of our search algorithm.

4.4.2 Hyper-Cell

Robustness Firstly, to verify the robustness of hyper-cell, we set different initial numbers of cells and different random seeds in the initialization phase. The network contains three hyper-cells and the initial cell numbers in hyper-cells are set to {a, b, c}, after optimization, the numbers of cells remaining in each hyper-cell are $\{\overline{a}, \overline{b}, \overline{c}\}$. As shown in Table 4, the experiments demonstrate that the hyper-cells are insensitive to both initial numbers of cells and random seeds, which verify the robustness and stability of the hyper-cell.

Downsampling strategy To demonstrate the effectiveness of the downsampling strategy searched by hyper-cells, we compare the random downsampling position settings with the searched one. The total cell number is 12 (i.e., $\overline{a}+\overline{b}+\overline{c}$ = 12) searched by our framework, we fix the searched cell structures and only change the downsampling positions (x, y, z) randomly for a fair comparison. The (x, y, z) represents the index positions of reduction cells in the 12 cells. After pre-training and retraining, the results in Table 5 demonstrate the superiority of the searched downsampling strategy through hyper-cells. Compared with the random ones, our hyper-cell achieves the best trade-off between accuracy and speed.

Hyper-cell number In our framework, we set the number of hyper-cells as 3 empirically. Thus, the downsampling factor is 16 with a stem convolution layer. In fact, the number of hyper-cells also can be learned by a learnable architecture parameter δ, and the optimization process of δ is similar to that of the architecture parameter β. Specifically, the number of hyper-cells can be learned by a hyper-cell-level pruning process, i.e., from the initial hyper-cell numbers to the reduced hyper-cell numbers automatically as follows. First, we introduce edges to connect with each output of hyper-cells, and associate them with the learnable parameters δ. Then, we determine the number of hyper-cells by limiting that only one edge can be activated. We set the initial hyper-cell number as 5 and thus the initial max downsampling factor is 64, which covers the common practices. Finally, the parameter δ and network parameters w are optimized to

Table 6 The optimization results of the number of hyper-cells

Initial hyper-cell number	δ after optimization	Optimized hyper-cell number
5	{0, 0, 1, 0, 0}	3
5	{0, 0, 0, 1, 0}	4
5	{0, 0, 1, 0, 0}	3
5	{0, 0, 1, 0, 0}	3
5	{0, 0, 0, 1, 0}	4

Table 7 The optimization results with different hyper-cell numbers

Hyper-cell number	Initial phase	After optimization	mIoU (%)	FPS
3	{5, 10, 10}	{2, 4, 6}	72.9	110.0
4	{5, 10, 10, 10}	{1, 2, 5, 5}	73.1	106.7
5	{5, 10, 10, 10, 10}	{1, 2, 3, 4, 2}	71.7	118.3

Table 8 Ablation study for the aggregation cell

Methods	mIoU (%)	FPS
(a) Without aggregation cell	69.9	116.2
(b) Random aggregation cell	71.4	108.5
(c) AutoRTNet-$\bar{F}$	72.2	112.8
(d) AutoRTNet-F	72.9	110.0

determine the number of hyper-cells automatically. We perform five repeated experiments as shown in Table 6. With latency weight $\gamma = 0.01$, the hyper-cell number determined by the parameter δ is 3 or 4.

Then, we conduct the experiments with different numbers of hyper-cells, including the numbers 3 and 4, which are determined by δ, and hand-designed number 5. The results are shown in Table 7, we observe that AutoRTNet obtains similar performance when the hyper-cell number is 3 or 4. When the hyper-cell number is 5, there has a degradation in performance, which demonstrates the effectiveness of the hyper-parameter δ. Hence, the hyper-cell number can be set as 3 or 4 in our framework.

4.4.3 Aggregation Cell

To demonstrate the effectiveness of the proposed aggregation cell, we conduct a series of experiments with different strategies: (a) without multi-scale feature aggregation; (b) with the aggregation cell using selected operations randomly from the aggregation cell's search space; (c) with the searched aggregation cell under $\gamma = 0.01$, the corresponding network is named AutoRTNet-$\bar{F}$; d) with the searched unconstrained aggregation cell (i.e. our AutoRTNet-F), i.e., we have not introduced the latency constraint for the aggregation cell. Among them, the result of the random aggregation cell is the average result over ten repeated random experiments and the results are shown in Table 8. Overall, the searched aggregation cell successfully boosts up the mIoU from 69.9 to 72.9% on the Cityscapes validation set. Particularly, the searched aggregation cell surpasses the random one 1.5% performance gains. Moreover, we observe that the accuracy degrades from 72.9 to 72.2% mIoU while the speed only gains 0.22 ms (+2.8 FPS) improvement when adding the latency constraint to the aggregation cell. Thus, for a better overall trade-off between accuracy and speed, we do not introduce the latency constraint to the aggregation cell.

4.4.4 Hyper-Cell Searching Process

To better analyze how hyper-cell works throughout the whole searching process, we visualize the number of cells of each hyper-cell after the warm-up phase, as depicted in Fig. 5. The initial cell numbers are {5, 10, 10} and eventually converge to {2, 4, 6} in three hyper-cells. The blue lines from top to bottom denote the actual cell numbers according to the current architecture parameter β of each hyper-cell, and red curves represent the mathematical expectation values of current cell numbers. We observe that our framework explores different cell numbers (i.e. different depths) in each hyper-cell actively at the early stage of searching, and the expectation values of cell numbers also change gradually. The cell numbers progressively become stable towards the final architecture in the late stage of searching, and the actual cell number lines gradually get close to the expectation curves. Another interesting observation is that hyper-cell #1 finds its optimal

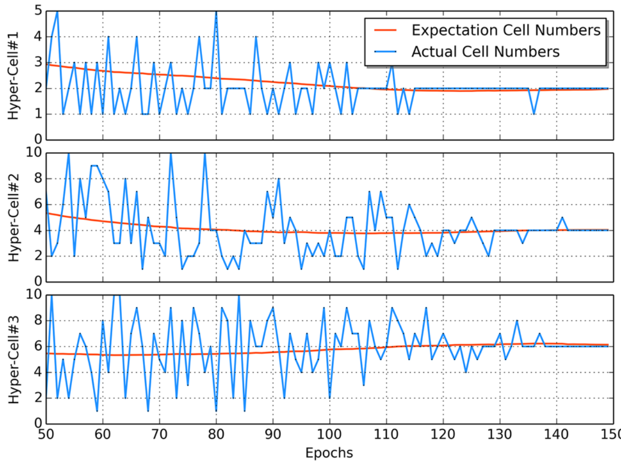

Fig. 5 Illustration of cell numbers in hyper-cells during the searching process. Blue lines from top to bottom denote the actual cell number changing in each hyper-cell with the increase of epochs, and red curves represent the mathematical expectation values of the current cell numbers in hyper-cells (Color figure online)

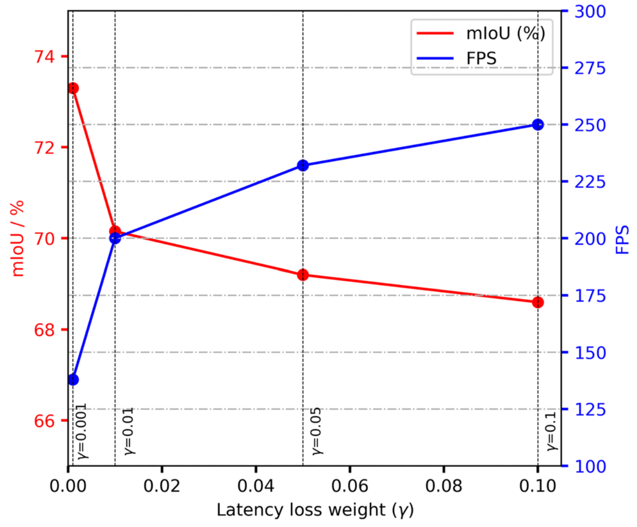

Fig. 6 The results of different latency settings on CamVid

depth much earlier than the other ones, indicating that the searching process follows a shallow-to-deep manner as we expected.

4.4.5 Different Latency Settings

Our joint search framework searches for the optimal network architectures under different latency settings (i.e. loss weight γ). In Sect. 4.3, AutoRTNet-F and AutoRTNet-S are searched with $\gamma = 0.01$ and 0.001 on the Cityscapes dataset, respectively, which demonstrates the flexibility of our framework. We also conduct the architecture search on the CamVid dataset with different latency settings, and the results are as depicted in Fig. 6. The networks searched with $\gamma = 0.001, 0.01, 0.05, 0.1$ achieve 73.3%, 70.2%, 69.2%, 68.6% mIoU and 138.0, 200.2, 232.1, 250.0 FPS on the CamVid test set, respectively. Notably, our AutoRTNet-U achieves 250.0 FPS while maintaining 68.6% mIoU, which surpasses ICNet (67.1% mIoU with 34.5 FPS) and DFANet (64.7% mIoU with 120 FPS) significantly.

4.5 Insights from Searched AutoRTNet-F

We provide an in-depth analysis of the AutoRTNet-F searched by our framework. We use the NAS methods to search the suitable architectures for specific tasks, likewise, we should understand why the searched network works well and it will guide the hand-designed process in turn. We have the following three important observations.

Early downsampling We notice that the searched downsampling strategy is stable and reasonable. As shown in Table 4, in the first hyper-cell, whether the initialized cell number is 5 or 10, the final number is at most 2 after the optimization. The reason is that the visual information is highly spatially redundant, thus can be compressed into a more efficient representation. Under the latency constraints, the searched downsampling strategy is as we expected and follows the early downsampling prior knowledge (Paszke et al. 2016).

Suitable receptive field The suitable receptive field size is crucial for semantic segmentation (Luo et al. 2016). A too-large receptive field may introduce some extra noises or negative interference, and the network cannot capture enough context information if it is too small. During the searching process, our AutoRTNet continuously adjusts the operation selection to determine the suitable receptive field. For example, in the optimized aggregation cell, as shown in Fig. 7, the

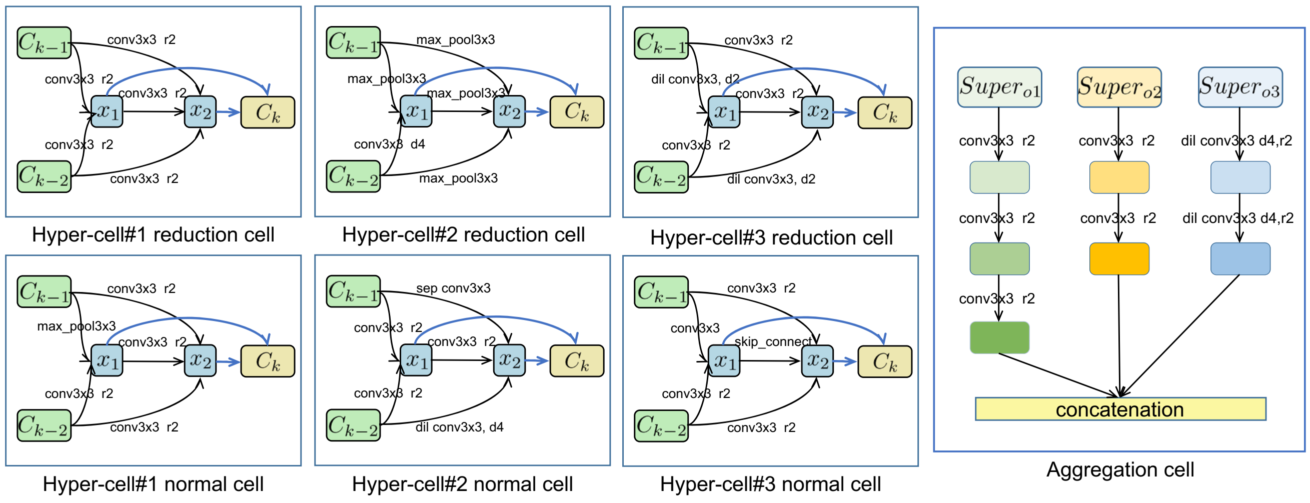

Fig. 7 Illustration of the detailed AutoRTNet-F architecture. The structures of the reduction cells and normal cells in three hyper-cells are shown in the figure respectively. The structure of searched aggregation cell in shown on the right. Best viewed in color (Color figure online)

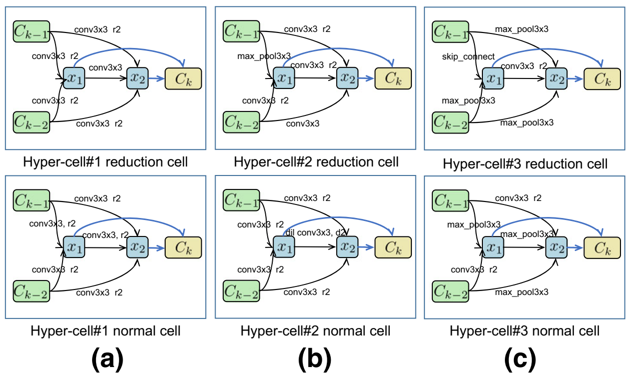

Fig. 8 Illustration of the detailed AutoRTNet-F' architecture

Table 9 The hyper-parameters used for the search and the corresponding results of AutoRTNet-F and AutoRTNet-F'

	AutoRTNet-F	AutoRTNet-F'
Hyper-parameters when training		
(a) Latency loss weight (γ)	0.01	0.001
(b) Input image size (H × W)	768 × 1536	720 × 960
(c) Mini-batch size	16	16
(d) Searched dataset	Cityscapes	CamVid
Results after retraining		
Speed on CamVid (FPS)	140.0 FPS	138.0 FPS
Performance on CamVid (mIoU)	73.5%	73.3%

operations from the outputs of the third hyper-cell always choose the operation with a dilation rate is 4 rather than 2 or 8 also in the search space. So we should choose the corresponding operations for the suitable receptive field in hand-designed semantic segmentation networks.

Operation selection The early operations act as good feature extractors. As shown in Fig. 7, the selection of operations in early stages always tends to be conv 3×3. The middle and deep layers have a diversity of operation selection. When performing multi-scale feature aggregation in the aggregation cell, we clearly found that the deeper layers enjoy dilated convolution operations, while the lower layers only prefer the common convolution operations.

4.6 Comparison of Networks Searched on Different Datasets

In this part, we compare the similarities and differences of AutoRTNet-F searched on Cityscapes and its counterpart, which is searched on CamVid. We depict the architecture searched on CamVid with $\gamma = 0.001$ in Fig. 8, which is named as AutoRTNet-F'. Compared with AutoRTNet-F, AutoRTNet-F' has a similar Hyper-cell #1 architecture. However, Hyper-cell #2 and Hyper-cell #3 of two networks are rather different. Specifically, Hyper-cell #2 of AutoRTNet-F selects more max-pooling operations than AutoRTNet-F', and Hyper-cell #3 of AutoRTNet-F prefers more convolution operations than AutoRTNet-F'. Moreover, AutoRTNet-F, which is searched at higher image resolution, has more dilated convolution operations than AutoRTNet-F'. We also list the used hyper-parameters for the network search and the corresponding network performance as shown in Table 9. Other searching hyper-parameters remain the same when searching for AutoRTNet-F and AutoRTNet-F'.

4.7 Detailed Time and GPU Information

The inference time or FPS is influenced by the GPU device and the input image size of the model. Here we list detailed information of previous approaches in Table 10 for readers as reference. Our GPU device is Nvidia TitanXP GPU. For a fair comparison, we directly quote the reported remeasured or estimated results on TitanXP of other algorithms in CAS (Zhang et al. 2019b) and SwiftNet (Orsic et al. 2019) paper. And we remeasure the speed of the methods based on our implementation if the original speed was reported on different GPUs and not mentioned in CAS (Zhang et al. 2019b) and SwiftNet (Orsic et al. 2019). Note that our implementations and speed measurements do not use TensorRT optimizations.

The speed of DFANet is reported on TitanX GPU, and also not mentioned in CAS (Zhang et al. 2019b) and SwiftNet (Orsic et al. 2019). Thus, we remeasure the inference time on TitanXP carefully for a fair comparison. There still has a speed gap between the original speed and the one we measured, we suspect that this is caused by the inconsistency of the implementation platform. We reimplement DFANet using official PyTorch (Paszke et al. 2017), and they measure it on their framework in which the depth-wise separable convolution is more fully optimized.

4.8 Detailed Quantitative Results and Visualization Results

Here we provide detailed quantitative results of per-class mIoU on the Cityscapes and CamVid datasets. Moreover, we provide the performance of the AutoRTNet on the full-resolution Cityscapes validation set. Finally, we provide some visual segmentation results on Cityscapes and CamVid.

4.8.1 Cityscapes Dataset

Compared with other methods, our AutoRTNet-F achieves an overall 72.2% mIoU with 110.0 FPS, which is the state-of-the-art trade-off between accuracy and speed. The per-class accuracy values are shown in Table 11. In comparison with other methods with public per-class accuracy on the Cityscapes test set, our predictions are more accurate in 13 out of 19 classes. AutoRTNet-F achieves slight improve-

Table 10 The detailed information of our AutoRTNet and other state-of-the-art methods on the Cityscapes test set

Method	Input size	mIoU (%)		Latency (ms) on TitanXP	FPS on TitanXP	Original results	
		val	test			FPS	GPU
FCN-8S (Long et al. 2015)	512 × 1024	–	65.3	227.23	4.4	–	–
PSPNet* (Zhao et al. 2017)	713 × 713	–	81.2	1288.0	0.78	–	–
DeepLabV3* (Chen et al. 2017)	769 × 769	–	81.3	769.23	1.3	–	–
AutoDeepLab* (Liu et al. 2019a)	769 × 769	–	81.2	303.0	3.3	–	–
SegNet (Badrinarayanan et al. 2017)	640 × 320	–	57.0	30.3	33	–	–
SQ (Treml et al. 2016)	1024 × 2048	–	59.8	46.0	21.7	–	Titan X M
ENet (Paszke et al. 2016)	640 × 320	–	58.3	12.7	78.4	135.4	Titan X
ERFNet (Romera et al. 2017)	1024 × 512	–	69.7	48.5	20.6	11.2	TitanX M
ICNet (Zhao et al. 2018a)	1024 × 2048	67.7	69.5	26.5	37.7	30.3	TITAN X(M)
SwiftNet (Orsic et al. 2019)	1024 × 2048	74.4	75.1	26.2	38.1	34.0	GTX 1080Ti
DF1-Seg (Li et al. 2019b)	768 × 1536	74.1	73.0	29.1	34.4	30.7	GTX 1080Ti
ESPNet (Mehta et al. 2018)	1024 × 512	–	60.3	8.2	121.7	112	TitanX
BiSeNet (Yu et al. 2018)	768 × 1536	69.0	68.4	9.52	105.8	105.8	TitanXP
DFANet (Li et al. 2019a)	1024 × 1024	–	71.3	20.6†	48.5†	100.0	TitanX
CAS (Zhang et al. 2019b)	768 × 1536	71.6	70.5	9.25	108.0	108.0	TitanXP
CAS* (Zhang et al. 2019b)	768 × 1536	72.5	72.3	9.25	108.0	108.0	TitanXP
AutoRTNet-F	768 × 1536	72.9	72.2	9.09	110.0	110.0	TitanXP
AutoRTNet-F*	768 × 1536	74.5	73.9	9.09	110.0	110.0	TitanXP
AutoRTNet-S	768 × 1536	74.7	74.3	14.0	71.4	71.4	TitanXP
AutoRTNet-S*	768 × 1536	76.0	75.8	14.0	71.4	71.4	TitanXP

Methods trained using both fine and coarse data are marked with $*$. The mark † represents the speed is remeasured by us

Table 11 Detailed per-class accuracy comparison of our AutoRTNet with other methods on the Cityscapes test set

Method	Road	Sidewalk	Building	Wall	Fence	Pole	Traffic light	Traffic sign	Vegetation	Terrain	Sky	Person	Rider	Car	Truck	Bus	Train	Motorcycle	Bicycle	Mean IOU(%)	FPS
SegNet 2017	96.4	73.2	84.0	28.4	29.0	35.7	39.8	45.1	87.0	63.8	91.8	62.8	42.8	89.3	38.1	43.1	44.1	35.8	51.9	57.0	33
ENet 2016	96.3	74.2	75.0	32.2	33.2	43.4	34.1	44.0	88.6	61.4	90.6	65.5	38.4	90.6	36.9	50.5	48.1	38.8	55.4	58.3	78.4
ICNet 2018a	97.1	79.2	89.7	43.2	48.9	61.5	60.4	63.4	91.5	68.3	93.5	74.6	56.1	92.6	51.3	72.7	51.3	53.6	70.5	69.5	37.7
ESPNet 2018	97.0	77.5	76.2	35.0	36.1	45.0	35.6	46.3	90.8	63.2	92.6	67.0	40.9	92.3	38.1	52.5	50.1	41.8	57.2	60.3	121.7
ERFNet 2017	97.9	82.1	90.7	45.2	50.4	59.0	62.6	68.3	91.9	69.4	94.2	78.5	59.8	93.4	52.3	60.8	53.7	49.9	64.2	69.7	20.6
BiSeNet 2018	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	68.4	105.8
DFANet 2019a	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	71.3	83.1
CAS 2019b	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	70.5	108.0
AutoRTNet-F	98.5	84.9	91.4	45.9	53.0	52.2	60.3	67.3	91.5	70.6	93.9	78.2	62.5	95.3	63.7	74.5	63.9	56.8	67.0	72.2	110.0
AutoRTNet-S	98.4	86.6	91.2	52.2	54.9	58.5	63.7	68.4	91.5	71.5	94.9	79.3	61.4	95.4	65.9	78.0	69.8	59.6	69.8	74.3	71.4

Table 12 Accuracy and speed comparison on the Cityscapes validation set with image resolution 1024 × 2048

Method	Input size	mIoU (%)	FPS on Titan XP	Original Results	
				FPS	GPU
SQ (Treml et al. 2016)	1024 × 2048	59.8	–	–	Titan X(M)
ICNet (Zhao et al. 2018a)	1024 × 2048	69.5	55.6	30.3	Titan X(M)
SwiftNet (Orsic et al. 2019)	1024 × 2048	74.4	38.1	34.0	GTX 1080Ti
CAS (Zhang et al. 2019b)	1024 × 2048	74.0	45.2	34.2	GTX 1070
AutoRTNet-F	1024 × 2048	75.0	62.7	62.7	Titan XP
AutoRTNet-S	1024 × 2048	76.8	45.6	45.6	Titan XP

Table 13 Detailed per-class accuracy comparison of our AutoRTNet with other methods on the CamVid test set

Method	Building	Tree	Sky	Car	Sign	Road	Pedestrian	Fence	Pole	Sidewalk	Bicyclist	Mean IOU(%)	FPS
SegNet 2017	88.8	87.3	92.4	82.1	20.5	97.2	57.1	49.3	27.5	84.4	30.7	55.6	29.4
ENet 2016	74.7	77.8	95.1	82.4	51.0	95.1	67.2	51.7	35.4	86.7	34.1	51.3	61.2
ICNet 2018a	–	–	–	–	–	–	–	–	–	–	–	67.1	34.5
BiseNet-Xception39 2018	82.2	74.4	91.9	80.8	42.8	93.3	53.8	49.7	25.4	77.3	50.0	65.6	–
BiseNet-Res18 2018	83.0	75.8	92.0	83.7	46.5	94.6	58.8	53.6	31.9	81.4	54.0	68.7	–
DFANet 2019a	–	–	–	–	–	–	–	–	–	–	–	64.7	120.0
CAS 2019b	–	–	–	–	–	–	–	–	–	–	–	71.2	169.0
AutoRTNet-F	88.1	78.4	91.7	93.0	44.0	96.2	66.8	60.8	33.0	88.6	67.4	73.5	140.0
AutoRTNet-S	89.1	79.3	91.7	92.5	41.0	96.8	64.7	66.5	32.2	90.0	72.1	74.2	82.5
AutoRTNet-U	87.5	77.9	91.7	90.6	30.4	95.6	62.2	47.1	25.2	87.5	58.9	68.6	250.0

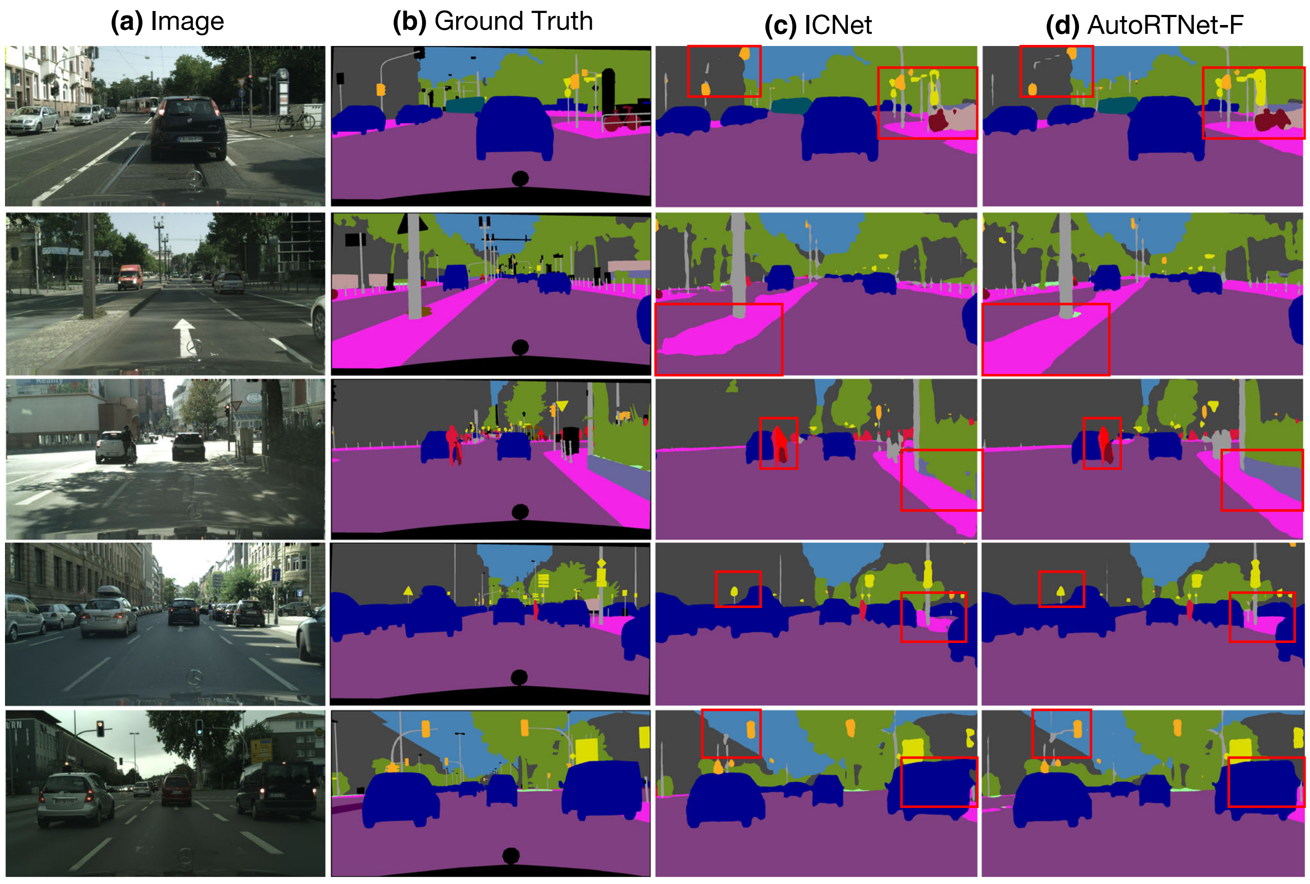

Fig. 9 Visual segmentation results on the Cityscapes validation set. **a** Image. **b** Ground Truth. **c** ICNet. **d** AutoRTNet-F

ments on the general classes (Road, Sidewalk, Building, Terrain, Car, etc.) while obtaining a significant accuracy improvement on the challenging classes (Truck, Motorbike, Train, Fence, Rider, etc.). Moreover, AutoRTNet-S achieves 74.3% mIoU on the Cityscapes test set with 71.4 FPS.

The cityscapes dataset contains high-resolution 1024×2048 images, which makes it a big challenge for real-time semantic segmentation. With high-resolution image inputs, Zhao et al. (2018a) focus on building a practically fast semantic segmentation system while accomplishing high-quality results. SwiftNet (Orsic et al. 2019) and CAS (Zhang et al. 2019b) also perform the experiments on Cityscapes with full-resolution image inputs. In this part, we compare with these methods on the Cityscapes validation set and the results are

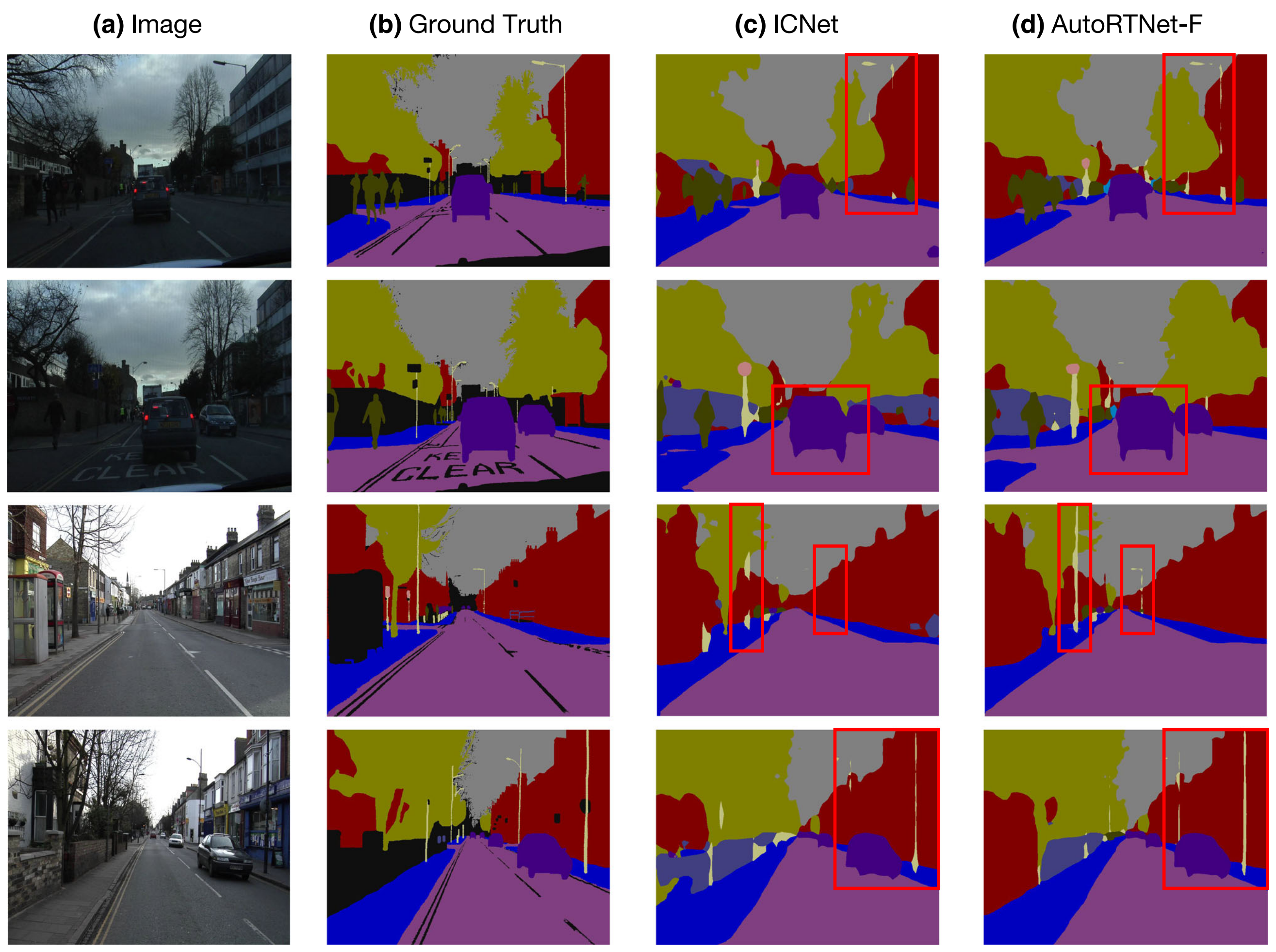

Fig. 10 Visual segmentation results on CamVid test set. **a** Image. **b** Ground Truth. **c** ICNet. **d** AutoRTNet-F

shown in Table 12. We refer to the speed scaling factors on different GPUs in SwiftNet (Orsic et al. 2019) paper and estimate the speed values of ICNet, SwiftNet, CAS on Titan XP GPU.

AutoRTNet-F Our AutoRTNet-F achieves 75.0% mIoU and 62.7 FPS on the full-resolution Cityscapes validation set (i.e. 1024 × 2048). To the best of our knowledge, the real-time performance of AutoRTNet-F outperforms all existing real-time methods. Compared with ICNet, AutoRTNet surpasses it by 5.5% in mIoU with a faster inference speed. Moreover, AutoRTNet outperforms SwiftNet and CAS by 0.6% and 1.0% in mIoU, and has a great advantage in inference speed (i.e. 62.7 FPS vs 38.1 FPS, 62.7 FPS vs 45.2 FPS).

AutoRTNet-S Our AutoRTNet-S achieves 76.8% mIoU with 45.6 FPS on the full-resolution Cityscapes validation set, which is the state-of-the-art real-time performance. Compared with SwiftNet and CAS that both have a little bit slower speed than us, our AutoRTNet-S surpasses them by 2.4% and 2.8% in mIoU, respectively.

4.8.2 CamVid Dataset

As shown in Table 13, with 720 × 960 input images, the searched AutoRTNet-F achieves 73.5% mIoU with 140.0 FPS, which is the state-of-the-art trade-off between accuracy and speed on the CamVid test set. In comparison with other methods, the predictions of our AutoRTNet-F are more accurate in 7 out of the 11 classes. More importantly, the inference speed of AutoRTNet-F achieves 140 FPS, which is impressive compared with other methods. (e.g. SegNet 29.4 FPS, ENet 61.2 FPS, ICNet 34.5 FPS). The per-class accuracy of AutoRTNet-S and AutoRTNet-U are also shown in Table 13.

4.8.3 Visual Segmentation Results

We provide some visual prediction results on both Cityscapes and CamVid datasets here. As shown in Figs. 9 and 10, the columns correspond to the input image, ground truth, the prediction of ICNet, and the prediction of our AutoRTNet-

Table 14 Results on the Cityscapes validation set of our networks and PSPNet

Method	mIoU (%)	Latency (ms)	FPS	Parameters (M)
PSPNet-Res50	77.3	636.9	1.57	49.08
PSPNet-Res101	78.6	1288.0	0.78	68.07
AutoRTNet-S	74.7	12.1	82.5	3.88
AutoRTNet-Ŝ	78.5	114.9	8.7	58.96

Table 15 Results on the Cityscapes test set of our AutoRTNet-Ŝ and other high-accuracy methods

Method	mIoU (%)	Latency(ms)	FPS
PSPNet-Res101*2017	81.2	1288.0	0.78
DeepLabV3*2017	81.3	769.23	1.3
AutoDeepLabV3*2019a	81.2	303.0	3.3
AutoRTNet-Ŝ*	81.0	114.9	8.7

F. Compared with ICNet, AutoRTNet-F produces more accurate and detailed results with faster inference speed. For example, AutoRTNet-F captures small objects in more details (e.g. traffic light in Fig. 9, poles in Fig. 10) and generates "smoother" results on object boundaries (e.g. rider, fence in Fig. 9, car in Fig. 10).

4.9 Comparison with High-Accuracy Models

It can be found that our AutoRTNet has a performance gap to high-accuracy methods, e.g. PSPNet, even with small γ values. In fact, in our framework, the performance of the searched network is mainly affected by two key factors: (1) the search space size and (2) the latency loss weight γ. We aim to search real-time networks and thus use a relatively small search space, which is responsible for the performance gap. Specifically, the maximal number of channels in our search space is only 144, and the parameter size of the search space is only 12.14 M. The searched AutoRTNet-S has only 3.88 M parameters, while PSPNet-ResNet101 has 68.07 M.

For a comparison with PSPNet, we directly expand our AutoRTNet-S by widening the network widths (i.e. 4 $\times$ channels) to obtain a larger network named AutoRTNet-Ŝ. Then, we evaluate it on the Cityscapes validation set, and the corresponding results are shown in Table 14. We observe that AutoRTNet-Ŝ achieves 78.5% mIoU on the validation set, which has a very close accuracy (only 0.1% mIoU difference) with PSPNet-ResNet101 (78.6% mIoU). Note that our AutoRTNet-Ŝ is about 10 times faster in inference than PSPNet-ResNet101 (8.7 FPS VS 0.78 FPS).

Moreover, we also train AutoRTNet-Ŝ with the coarse annotations in comparison with other high-accuracy methods, and the results on the Cityscapes test set are shown in Table 15. From the experimental results, AutoRTNet-Ŝ sacrifices the real-time speed and obtains the accuracy improvement. Even so, the high-performance network still has a 10 $\times$ inference speed compared to PSPNet. In conclusion, the experiment proves the great potential of our AutoRTNet. Although our framework focuses on the real-time segmentation network search under a relatively small search space, it also can be generalized and adaptive to the high-accuracy scenario flexibly by enlarging the search space.

5 Conclusion

In this paper, we propose a novel joint search framework that covers all three main aspects of the design philosophy for real-time semantic segmentation networks. The framework searches for building blocks, network depth, downsampling strategy, and feature aggregation method simultaneously. The hyper-cell is proposed for searching the network depth and downsampling strategy jointly and automatically, and the aggregation cell is introduced for automatic multi-scale feature aggregation. Extensive experiments on both Cityscapes and CamVid datasets demonstrate the superiority and effectiveness of our approach.

Acknowledgements This work is supported in part by National Natural Science Foundation of China under Grant U20A20222, Zhejiang Provincial Natural Science Foundation of China under Grant LR19F020004, National Key Research and Development Program of

China under Grant 2020AAA0107400, and key scientific technological innovation research project by Ministry of Education.

References

Badrinarayanan, V., Kendall, A., & Cipolla, R. (2017). Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(12), 2481–2495.

Baker, B., Gupta, O., Naik, N., & Raskar, R. (2016). Designing neural network architectures using reinforcement learning. arXiv preprint arXiv:1611.02167.

Brostow, G. J., Shotton, J., Fauqueur, J., & Cipolla, R. (2008). Segmentation and recognition using structure from motion point clouds. In *European conference on computer vision* (pp. 44–57). Springer.

Cai, H., Zhu, L., & Han, S. (2018). Proxylessnas: Direct neural architecture search on target task and hardware. arXiv:1812.00332.

Chen, L. C., Papandreou, G., Schroff, F., & Adam, H. (2017). Rethinking atrous convolution for semantic image segmentation. arXiv preprint arXiv:1706.05587.

Chen, L. C., Collins, M., Zhu, Y., Papandreou, G., Zoph, B., Schroff, F., Adam, H., Shlens, J. (2018a). Searching for efficient multi-scale architectures for dense image prediction. In *Advances in Neural Information Processing Systems* (pp. 8699–8710).

Chen, L. C., Zhu, Y., Papandreou, G., Schroff, F., & Adam, H. (2018b). Encoder–decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)* (pp. 801–818).

Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1251–1258).

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 *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 3213–3223).

Everingham, M., Eslami, S. A., Van Gool, L., Williams, C. K., Winn, J., & Zisserman, A. (2015). The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, *111*(1), 98–136.

Fu, J., Liu, J., Tian, H., Li, Y., Bao, Y., Fang, Z., & Lu, H. (2019). Dual attention network for scene segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 3146–3154).

Ghiasi, G., Lin, T. Y., & Le, Q. V. (2019). Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *The IEEE conference on computer vision and pattern recognition (CVPR)*

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

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 preprint arXiv:1704.04861.

Jang, E., Gu, S., & Poole, B. (2016). Categorical reparameterization with gumbel-softmax. arXiv preprint arXiv:1611.01144.

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

Li, H., Xiong, P., An, J., & Wang, L. (2018). Pyramid attention network for semantic segmentation. arXiv preprint arXiv:1805.10180.

Li, H., Xiong, P., Fan, H., & Sun, J. (2019a). Dfanet: Deep feature aggregation for real-time semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 9522–9531).

Li, L., & Talwalkar, A. (2019). Random search and reproducibility for neural architecture search. CoRR abs/1902.07638. http://arxiv.org/abs/1902.07638.

Li, X., Zhou, Y., Pan, Z., Feng, J. (2019b). Partial order pruning: For best speed/accuracy trade-off in neural architecture search. In *Proceedings of IEEE conference on computer vision and pattern recognition (CVPR)*.

Lin, G., Milan, A., Shen, C., & Reid, I. (2017a). Refinenet: Multi-path refinement networks for high-resolution semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1925–1934).

Lin, T. Y., Dollár, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017b). Feature pyramid networks for object detection. In *CVPR*.

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 *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 82–92).

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

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. E., Fu, C. Y., & Berg, A. C. (2016). Ssd: Single shot multibox detector. In *ECCV*.

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

Luo, W., Li, Y., Urtasun, R., & Zemel, R. (2016). Understanding the effective receptive field in deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 4898–4906).

Maddison, C. J., Mnih, A., & Teh, Y. W. (2016). The concrete distribution: A continuous relaxation of discrete random variables. arXiv preprint arXiv:1611.00712.

Mehta, S., Rastegari, M., Caspi, A., Shapiro, L., & Hajishirzi, H. (2018). Espnet: Efficient spatial pyramid of dilated convolutions for semantic segmentation. In *Proceedings of the European conference on computer vision (ECCV)* (pp. 552–568).

Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., et al. (2019). Evolving deep neural networks. *Artificial intelligence in the age of neural networks and brain computing* (pp. 293–312). Amsterdam: Elsevier.

Nekrasov, V., Chen, H., Shen, C., & Reid, I. (2019). Fast neural architecture search of compact semantic segmentation models via auxiliary cells. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 9126–9135).

Noh, H., Hong, S., & Han, B. (2015). Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE international conference on computer vision* (pp. 1520–1528).

Orsic, M., Kreso, I., Bevandic, P., & Segvic, S. (2019). In defense of pre-trained imagenet architectures for real-time semantic segmentation of road-driving images. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 12607–12616).

Paszke, A., Chaurasia, A., Kim, S., & Culurciello, E. (2016). Enet: A deep neural network architecture for real-time semantic segmentation. arXiv preprint arXiv:1606.02147.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., & Lerer, A. (2017). Automatic differentiation in PyTorch. In *NIPS autodiff workshop*.

Real, E., Aggarwal, A., Huang, Y., & Le, Q. V. (2019). Regularized evolution for image classifier architecture search. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*, 4780–4789.

Romera, E., Alvarez, J. M., Bergasa, L. M., & Arroyo, R. (2017). Erfnet: Efficient residual factorized convnet for real-time semantic segmentation. *IEEE Transactions on Intelligent Transportation Systems*, *19*(1), 263–272.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4510–4520).

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., & Le, Q. V. (2019). Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2820–2828).

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

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998–6008).

Wu, B., Wang, Y., Zhang, P., Tian, Y., Vajda, P., & Keutzer, K. (2018). Mixed precision quantization of convnets via differentiable neural architecture search. arXiv preprint arXiv:1812.00090.

Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., & Keutzer, K. (2019). Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR* (pp. 10734–10742).

Wu, Z., Shen, C., & Hengel, A. V. D. (2016). High-performance semantic segmentation using very deep fully convolutional networks. arXiv preprint arXiv:1604.04339.

Wu, Z., Shen, C., Hengel, A. V. D. (2017). Real-time semantic image segmentation via spatial sparsity. arXiv preprint arXiv:1712.00213.

Xie, S., Zheng, H., Liu, C., & Lin, L. (2019). SNAS: Stochastic neural architecture search. In *ICLR*.

Xu, H., Yao, L., Zhang, W., Liang, X., & Li, Z. (2019). Auto-fpn: Automatic network architecture adaptation for object detection beyond classification. In *The IEEE international conference on computer vision (ICCV)*.

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

Yu, F., Koltun, V., & Funkhouser, T. (2017). Dilated residual networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 472–480).

Yu, K., Sciuto, C., Jaggi, M., Musat, C., & Salzmann, M. (2019). Evaluating the search phase of neural architecture search. arXiv preprint arXiv:1902.08142.

Zhang, H., Zhang, H., Wang, C., & Xie, J. (2019a). Co-occurrent features in semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 548–557).

Zhang, Y., Qiu, Z., Liu, J., Yao, T., Liu, D., & Mei, T. (2019b). Customizable architecture search for semantic segmentation. In *CVPR* (pp. 11641–11650).

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

Zhao, H., Qi, X., Shen, X., Shi, J., & Jia, J. (2018a). Icnet for real-time semantic segmentation on high-resolution images. In *Proceedings of the European conference on computer vision (ECCV)* (pp. 405–420).

Zhao, H., Zhang, Y., Liu, S., Shi, J., Loy, C. C., Lin, D., & Jia, J. (2018b). PSANet: Point-wise spatial attention network for scene parsing. In *ECCV*.

Zoph, B., & Le, Q. V. (2016). Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578.

Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 8697–8710).