# Customizable Architecture Search for Semantic Segmentation*

Yiheng Zhang [†], Zhaofan Qiu [†], Jingen Liu[§], Ting Yao [‡], Dong Liu [†], and Tao Mei [‡]
[†] University of Science and Technology of China, Hefei, China
[‡] JD AI Research, Beijing, China     [§] JD AI Research, Mountain View, USA
{yihengzhang.chn, zhaofanqiu, jingenliu, tingyao.ustc}@gmail.com
dongeliu@ustc.edu.cn, tmei@live.com

## Abstract

*In this paper, we propose a Customizable Architecture Search (CAS) approach to automatically generate a network architecture for semantic image segmentation. The generated network consists of a sequence of stacked computation cells. A computation cell is represented as a directed acyclic graph, in which each node is a hidden representation (i.e., feature map) and each edge is associated with an operation (e.g., convolution and pooling), which transforms data to a new layer. During the training, the CAS algorithm explores the search space for an optimized computation cell to build a network. The cells of the same type share one architecture but with different weights. In real applications, however, an optimization may need to be conducted under some constraints such as GPU time and model size. To this end, a cost corresponding to the constraint will be assigned to each operation. When an operation is selected during the search, its associated cost will be added to the objective. As a result, our CAS is able to search an optimized architecture with customized constraints. The approach has been thoroughly evaluated on Cityscapes and CamVid datasets, and demonstrates superior performance over several state-of-the-art techniques. More remarkably, our CAS achieves 72.3% mIoU on the Cityscapes dataset with speed of 108 FPS on an Nvidia TitanXp GPU.*

## 1. Introduction

Semantic segmentation, which aims at assigning semantic labels to every pixel of an image, is a fundamental topic in computer vision. Leveraging the strong capability of CNNs, which have been widely and successfully applied to image classification [12, 13, 28, 29, 30], most state-of-the-art works have made significant progress on semantic segmentation [4, 6, 19, 21]. To tackle the challenges (e.g., reduced feature resolution and objects at multiple scales) in CNN based semantic segmentation, researchers have proposed various network architectures, such as the application of dilated convolutions [4, 34] to capture larger contextual information without losing the spatial resolution, and multi-scale prediction ensemble [32]. Although these methods achieve promising high accuracy, they generally require long inference time due to the complicated networks, which carry huge numbers of operations and parameters.

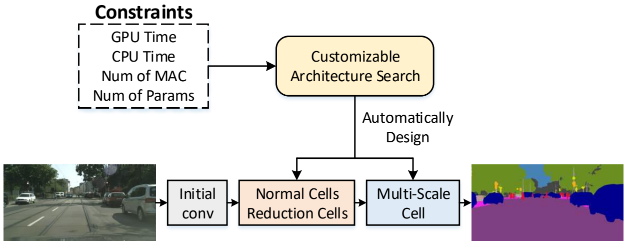


Figure 1. Our proposed Customizable Architecture Search (CAS) for semantic image segmentation. Given some constraints such as GPU/CPU time and number of parameters, our CAS is able to automatically generate an optimized network which consists of a sequence of stacked computation cells.

With the increasing need of semantic segmentation on some real-time applications like augmented reality wearables and autonomous driving, there is a high demand for fast semantic segmentation without sacrificing much accuracy, even on a low-power mobile device. Accordingly, some researchers attempt to make a real-time inference by various manually designed strategies including resizing or cropping the input [36], pruning the network channels [1], dropping some stages of the model [20], multiple scales feature integration [36] and spatial-context decoupling [33]. These designs usually require significant engineering effort of human experts. In addition, they have less flexibility to adjust the inference speed according to the actual dataset and hardware configurations. In other words, it is difficult to find a tradeoff between speed and performance for a specific task. To deal with these issues, we propose a Customizable Architecture Search approach to automatically generate a lightweight network with customized con-

*This work was performed at JD AI Research.

straints on the availability of computational resource and speed requirements. Our work is inspired by recently proposed solutions to automate the manual process of network design [17, 38]. The successes of these approaches have been demonstrated on some image classification tasks by surpassing the performances of human manually designed architectures [17]. Rather than solely pursuing the best performance like [3, 17, 38], we aim at searching an appropriate network under the constraints on the computational resource of an application. We call this procedure as Customizable Architecture Search (CAS). To the best of our knowledge, our CAS is the first effort to automatically generate network architectures for semantic segmentation given some constraints in real applications.

Figure 1 illustrates an overview of the proposed CAS for semantic segmentation. The proposed lightweight network consists of a couple of initial convolutions followed by sequentially stacked computation cells including both reduction cell and normal cell in the backbone network. A computation cell is a directed acyclic graph, which forms the building block of the learned network. The CAS aims at searching for an optimized architecture to achieve high-quality feature maps. To further recover the loss of spatial information during feature map learning, a multi-scale cell is attached to the backbone network to fuse multiple scales information. CAS jointly learns the architecture of the cells as well as the associated weights. The same type of cells share an identical architecture but with different weights. By relaxing the search space to be continuous, we employ the differential architecture search [17] to solve our CAS objective. As a result, the network search can be optimized with respect to a validation set by gradient descent.

The proposed CAS has been thoroughly evaluated on Cityscapes [8] dataset and promising results have been obtained. To compare with state-of-the-art approaches, we generate architectures constrained by GPU time and evaluate them on Cityscapes [8] and CamVid [2] datasets. The results exceed the state-of-the-art approaches in term of both performance and inference speed.

## 2. Related Work

**CNN based Semantic Image Segmentation.** Inspired by the success of CNN on visual recognition [12, 13, 25, 26, 28, 29, 30], recently researchers have proposed various CNN based approaches for semantic segmentation. The typical way of applying CNNs to segmentation is through patch-by-patch scanning [9, 23]. The fully convolutional network (FCN) [19] is proposed for semantic segmentation to exploit the high learning capacity of CNNs. It enables spatial dense prediction and efficient end-to-end training. Following FCN, researchers propose several advanced techniques ranging from cross-layer feature ensemble [10, 15, 24, 32] to context information exploitation [4, 5, 6, 18, 21, 27, 35, 37]. The FCN formulation could be further improved by employing post-processing techniques, such as the conditional random field [4], to consider label spatial consistency.

A lot of recent efforts have been made to achieve high-quality segmentation without considering the cost such as inference time. For example, PSPNet [37] and DeepLabv3 [5] have achieved over 81% of mIoU on Cityscape dataset running with less than 2 FPS, which is far away from real-time. Some works attempt to improve the inference speed by restricting the input resolution [1], pruning the channels of the network [36], dropping stages of the model [20] and utilizing the lightweight networks [31], while the loss of spatial information and network capacity corrupt the dense prediction of semantic segmentation. To remedy the information loss, experienced experts have designed network architectures to balance speed and segmentation quality. ICNet [36] is proposed to achieve real-time segmentation with a decent performance by employing a cascade network structure and incorporating multi-resolution branches. BiSeNet [33] decouples the network into a spatial path and a context path, in order to obtain a faster network with a competitive performance of semantic segmentation. Differing from the aforementioned efforts, in this paper we propose the solution of CAS, which automatically generates a lightweight architecture with the best tradeoff between speed and accuracy under some application constraints.

**Network Architecture Search**. The target of architecture search is to automatically design network architectures tailored for a specific task. The sequential model-based optimization [16] is proposed to guide the searching by learning a surrogate model. The reinforcement learning based methods [22, 38], which train a controller network to generate neural architectures, are proposed to obtain state-of-the-art performances on the tasks of image classification and natural language processing. Instead of treating the architecture search as a black-box optimization problem over a discrete domain, differentiable architecture search (DARTS) [17], which searches architectures in a continuous space, is presented to make the architecture be optimized by gradient descent and achieve competitive performance using fewer computational resources.

Our work is inspired by [17, 38]. Unlike these methods, however, our work attempts to achieve a good tradeoff between system performance and the availability of the computational resource. In other words, our algorithm is optimized with some constraints from real applications. We notice that the recent DPC work [3] is very related to ours. It addresses the dense image prediction problem via searching an efficient multi-scale architecture on the use of performance driven random search [11]. Nevertheless, our work is different from [3]. First of all, we have different objectives. Instead of targeting high-quality segmentation in [3], our solution is customizable to search for an optimized architecture which is constrained by the requirements of real applications. The generated architecture tries to keep a balance between the quality and limited computational resource. Secondly, our solution optimizes the architecture of the whole network including both backbone and multi-scale module, while [3] focuses on multi-scale optimization. Finally, our method employs a lightweight network, which costs much less training time as compared to that of [3].

## 3. Customizable Architecture Search

As shown in Figure 1, given the customized constraints in semantic segmentation task, the proposed CAS searches for a computation cell (e.g., normal/reduction cell, and multi-scale cell, which are represented as directed acyclic graphs as depicted in Figure 2) as the building block for an optimized network. Unlike the previous work [17], CAS not only searches for effective operations for a cell, but also considers the cost of choosing these operations. Namely, each operation has an associated cost being selected. As a result, the objective of architecture search is to generate a network that minimizes the following function:

$$\mathcal{L}_{val} + \lambda \mathcal{L}_{cost} \ , \tag{1}$$

where $\mathcal{L}_{val}$ is the loss on validation dataset, $\mathcal{L}_{cost}$ is the cost associated with the network, and $\lambda$ is the tradeoff controller. To solve this objective, following [17], we optimize the architecture of the computation cell by using gradient descent. Figure 2 illustrates an illustration of generating an architecture with and without constraints. To make this section self-contained, we first discuss the differentiable architecture search of [17] in a general form in subsection 3.1. We then describe how to perform the customizable optimization for semantic segmentation in subsection 3.2 , and detail the search space for network backbone and multi-scale cell in subsection 3.3 and 3.4, respectively.

### 3.1. Differentiable Architecture Search

A computation cell is a directed acyclic graph (DAG) as shown in Figure 2. The graph has an ordered sequence of $N$ nodes, represented as $\mathcal{N} = \{x^{(i)} | i = 1, \dots, N\}$, where $x^{(i)}$ denotes the feature map in a convolutional network. The transformation from $x^{(i)}$ to $x^{(j)}$ is represented as an operation $o^{(i,j)}(\cdot)$, which corresponds to a directed edge in the graph. Each computation cell has two input nodes (i.e., outputs of the previous two layers) and one output node (i.e., the concatenation of the intermediate nodes in the cell). Specifically, an intermediate node is calculated as:

$$x^{(j)} = \sum_{i<j} o^{(i,j)}(x^{(i)}) \ , \tag{2}$$

where $x^{(i)}$ is a node coming before $x^{(j)}$ in the cell. Hence, the problem of architecture search is equivalent to learning the operation on each edge in DAG.

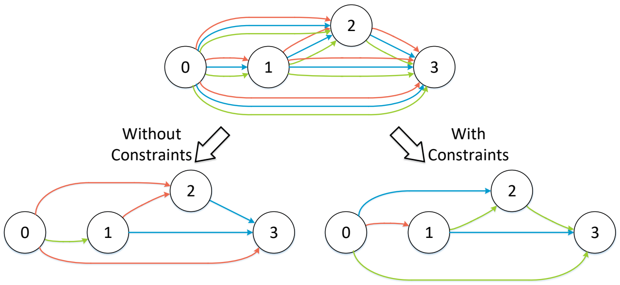


Figure 2. An illustration of generating a computing cell with/without constraints. Each edge represents one operation between two nodes. The top graph shows many candidate operations existing between nodes, and each candidate operation has its own cost. The red edge denotes a heavy cost, and the green one has a light cost. Without considering constraints, the search may generate a costly architecture (bottom left) for better performance, while our CAS outputs an architecture with light cost (bottom right).

To make the search space continuous, a weighted combination of all candidate operations is utilized as the transformation on the directed edge as follows:

$$\bar{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} Softmax(\alpha_o^{(i,j)}) o(x) \ , \tag{3}$$

where $o(\cdot)$ is an operation in the operation candidate set $\mathcal{O}$ of size $N_o$, and $\alpha_o^{(i,j)}$ is a learnable score of the operation $o(\cdot)$. The vector $\alpha^{(i,j)} \in \mathbb{R}^{N_o}$ represents the scores of all candidate operations on the edge from $x^{(i)}$ to $x^{(j)}$. Then the cell architecture is denoted as $\alpha = \{\alpha^{(i,j)}\}$, which is a set of vectors for all edges. Now the architecture search could be formulated as finding $\alpha$ to minimize the validation loss $\mathcal{L}_{val}(w'(\alpha), \alpha)$, where $w'(\alpha)$ is the parameters of the operations. The parameters are obtained by minimizing the training loss, formulated as $w'(\alpha) = argmin_w \mathcal{L}_{train}(w, \alpha)$. Accordingly, a cell could be optimized by adjusting $\alpha$ via gradient descent.

Since the variation of $\alpha$ leads to the recomputation of $w'(\alpha)$ by minimizing $\mathcal{L}_{train}(w, \alpha)$, the optimization procedure could be approximately performed by alternately optimizing weight parameters $w$ and cell architecture $\alpha$ with gradient descent steps. In particular, for the parameter update step $k$, $w_{k-1}$ is moved to $w_k$ according to the gradient $\nabla_w \mathcal{L}_{train}(w_{k-1}, \alpha_{k-1})$, and the architecture is updated to minimize the validation loss:

$$\mathcal{L}_{val}(w_k - \xi \nabla_w \mathcal{L}_{train}(w_k, \alpha_{k-1}), \alpha_{k-1}) \ , \tag{4}$$

where $\nabla_w \mathcal{L}_{train}(w_k, \alpha_{k-1})$ is a virtual gradient step of $w_k$ and $\xi$ is the step's learning rate. After optimizing the architecture of the computation cell encoded as $\alpha$ via gradient descent, each operation combination $\bar{o}^{(i,j)}$, which locates on the directed edge from $x^{(i)}$ to $x^{(j)}$ of the DAG, is replaced with the most likely operation candidate according to $\alpha^{(i,j)}$. Then $k$ strongest predecessors of each intermediate node are retained, where the strength of an edge is

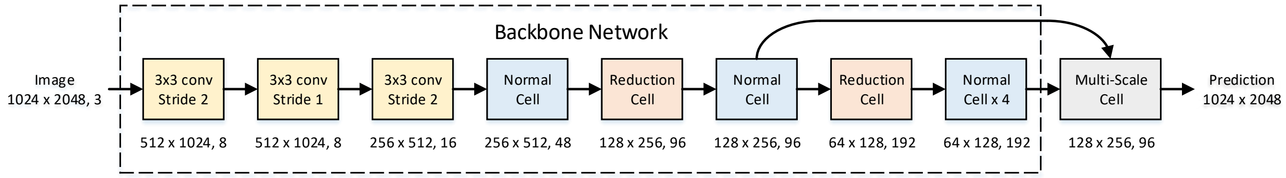


Figure 3. An overview of our network structure for semantic segmentation. We take $1024 \times 2048$ input as an example. It consists of two main components: the backbone network on the left followed by the multi-scale cell on the right. The backbone designed for efficient feature extraction begins with three convolutional layers followed by 6 normal cells and 2 reduction cells. The multi-scale cell learns to refine the feature map by integrating accurate spatial information from the second normal cell into the final feature map. Each cell employs the previous two cells' outputs as its inputs.

defined as $max(Softmax(\alpha^{(i,j)}))$. The $k$ is set as 2 in the following sections.

### 3.2. Customizable Optimization

As aforementioned, the differentiable architecture search enables an efficient search of high-performance architecture. Nevertheless, considering some practical constraints in real applications, a high-performance architecture is not the only pursuit given limited computational resource. In this section, we propose a constrained architecture search method, which takes a further step forward to discover an appropriate design of the network satisfying customizable constraints. To address the constraints in the architecture search procedure, we associate a cost with each operation, such that whenever an operation is selected, there is a cost for the selection. Hence, the cost of a cell is formulated as:

$$\mathcal{L}_{cost} = \sum_{j} \sum_{i<j} \sum_{o \in \mathcal{O}} c_o Softmax(\alpha_o^{(i,j)}) \ , \tag{5}$$

where $c_o$ is the cost associated with operation $o(\cdot)$ (Please refer to the implementation details in section 4.1 for how to convert constraints to costs). Hence, the architecture is optimized by updating $\alpha$ according to the following gradient:

$$\nabla_\alpha \mathcal{L}_{val} + \lambda \nabla_\alpha \mathcal{L}_{cost} \ , \tag{6}$$

where $\lambda$ is the tradeoff parameter and maintains the balance between the performance and network cost.

When applying CAS to semantic image segmentation, we employ a network structure as shown in Figure 3. It mainly contains two components: backbone and multi-scale cell, which are built and optimized by CAS separately. Given the input images, the backbone is first utilized to learn feature representations with rich semantics, while the accuracy of pixel-level localization will accordingly drop due to consecutive down-sampling operations. On the other hand, the multi-scale cell learns a refinement structure to recover spatial information from the feature on different stages of the backbone and leads to better predictions for semantic segmentation. The following two sections describe the details of the search for both components, respectively.

### 3.3. Backbone Architecture Search

As shown in Figure 3, the backbone network starts with three convolutional layers, followed by eight cells, each of which consists of $N = 6$ nodes including the input and output node. The first two nodes of the $i$-th cell are the outputs of the $(i-1)$-th and $(i-2)$-th cells or layers with $1 \times 1$ convolutions if dimension projection needed. In general, a backbone for image classification contains 5 spatial reduction which results in a feature map of 1/32 size of the original image [12, 29, 30]. Different from image classification which focuses on semantic aggregation, the loss of spatial information caused by spatial reduction is more important for semantic segmentation. As such, following [5], the spatial resolution of the final feature map is set only 16 times smaller than the input image resolution to balance the spatial density, semantics and expensive computation. In our case, in addition to the first and third convolutional layers of the backbone network with strides of 2, the two reduction cells also serve for down-sampling the feature map. Except for the reduction cells, the other cells are normal cells without reduction. Hence, the searchable architectures of the backbone are represented as $\alpha_{normal}$ and $\alpha_{reduce}$ shared by all normal cells and reduction cells, respectively, but with different weights.

We draw inspiration from the recent advances in the CNN literatures and collect the operation set $\mathcal{O}_b$:

- identity
- 3x3 max pooling
- 3x3 ave pooling
- 3x3 conv
- 3x3 dilated conv
- 3x3 separable conv, repeat 2
- 3x3 separable conv, repeat 4
- 3x3 conv, repeat 2
- 3x3 dilated conv, repeat 2
- 2x2 ave pooling stride 2 + 3x3 conv + upsampling
- 2x2 ave pooling stride 2 + 3x3 conv repeat 2 + upsampling

The $\mathcal{O}_b$ consists of four types of operations, i.e., non-learned operations, standard convolutions, separable convolutions and pooled convolutions. The identity shortcut [12], max pooling and average pooling are non-learned operations. The standard $3 \times 3$ convolutional layers with optional dilation are widely utilized in the convolutional networks designed for semantic segmentation. The separable convolution proposed in [7] is an operation that efficiently balances cost and performance by factorizing the standard convolution into a depthwise convolution and a pointwise convolution. It is worth noting that the separable convolution is often applied at least twice in an operation [17, 38]. In addition to existing operations, we propose the spatial bottleneck operation, namely pooled convolution. This operation applies average pooling with stride 2 on the feature map, followed by $3 \times 3$ convolutions and finally recovers the resolution of the feature map via bilinear upsampling. Our experiments demonstrate that such operation could effectively enlarge the receptive field and reduce computational cost. Note that we also repeat each weighted operation twice to enlarge the potential capacity of backbone network.

### 3.4. Multi-Scale Cell Search

With an optimized backbone network, the high-quality feature maps learned from images could be obtained and fed into the classifier to generate dense predictions for the images. To further refine feature maps by recovering the spatial information, multi-scale fusion, which aggregates different level features, has been proved to be effective for semantic segmentation [6, 10, 15, 19, 21, 33, 36]. In this paper, we aim at searching a multi-scale cell rather than directly utilizing manually designed architectures. The cell $\alpha_{ms}$ consisting of $N = 9$ nodes is heavier than $\alpha_{normal}$ and $\alpha_{reduce}$ in terms of cost. Nevertheless, the cell $\alpha_{ms}$ is only applied once at the end of the network and thus the cost is negligible compared to other cells. In $\alpha_{ms}$, the spatial resolutions of the inputs are firstly aligned by upsampling the smaller one via bilinear interpolation and then independent $1 \times 1$ convolutions are applied on each directed edge from spatially aligned inputs to intermediate nodes for channel projection. Inspired by the recent works on semantic segmentation, an operation set $\mathcal{O}_{ms}$ is collected specifically as:

- 3x3 conv, dilation=1
- 3x3 conv, dilation=2
- 3x3 conv, dilation=4
- 3x3 conv, dilation=8
- 15x1 then 1x15 conv
- 25x1 then 1x25 conv
- 8x8 residual SPP
- 16x16 residual SPP
- 24x24 residual SPP
- identity

Three types of operations, i.e., standard convolutions, spatial decomposed convolutions and residual spatial pyramid pooling, are included in $\mathcal{O}_{ms}$. Convolutional layers with multiple dilations could effectively capture multi-scale information [6]. The spatial decomposed convolution with large kernel size enables densely connections within a large region in the feature map and embeds rich context information in each location with less computational cost than general convolution with large kernel [21]. To provide contextual scenery prior to the feature map, the residual spatial pyramid pooling (SPP) with different window sizes is explored. Inside of each window, an average pooling is performed followed by an $1 \times 1$ convolution to encode the contextual information. The spatial resolution of the encoded context, which is combined with input feature map as residual value, is recovered by bilinear upsampling.

# 4. Implementation

## 4.1. Customizable Architecture Search

We utilize the gradient in Eq.(6) to update the $\alpha$ in CAS, The $\nabla_{\alpha}\mathcal{L}_{val}$ could be derived from Eq.(4) as:

$$\nabla_{\alpha}\mathcal{L}_{val}(w', \alpha) - \xi \nabla^{2}_{\alpha,w}\mathcal{L}_{train}(w, \alpha)\nabla_{w'}\mathcal{L}_{val}(w', \alpha)\ , \quad (7)$$

where $w' = w - \xi \nabla_{w}\mathcal{L}_{train}(w, \alpha)$. The weight parameters $w$ are updated by the virtual gradient step. For ease of optimization, an approximation of Eq.(7) is applied and the gradient of architecture could be represented as $\nabla_{\alpha}\mathcal{L}_{val}(w, \alpha)$ with respect to the case of $\xi = 0$ on the assumption that $\alpha$ and $w$ are independent.

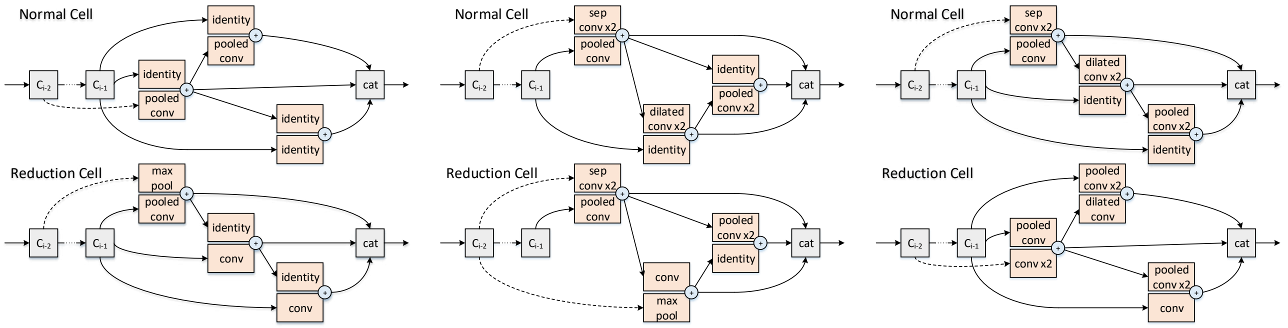
Given the candidate operation set $\mathcal{O}$, to evaluate the cost $c_o$, we firstly measure the cost $c'_o$ of the whole network whose cells only consist of $o(\cdot)$, and $c_o$ is computed as $c_o = c'_o - c'_{id}$, where $c_{id}$ denotes the cost of the network whose operations of cells are replaced by "identity". The cost could be defined according to the constraints, e.g., GPU / CPU inference time, number of parameters and number of multiply-accumulate operations (MAC). In order to characterize the lack of concatenation between two nodes in the computation cell, a special "None" operation is appended to $\mathcal{O}$ during the optimization but this operation is excluded in the decision of the final architecture.

## 4.2. Semantic Segmentation

In our implementations, we search the architectures of the backbone network and multi-scale cell separately. The backbone network architecture is firstly determined according to $\alpha_{normal}$ and $\alpha_{reduce}$ which are both optimized by CAS on the task of semantic segmentation. Then we utilize ImageNet ILSVRC12 dataset [28] to pre-train the backbone network from the scratch. With the ImageNet pre-trained weights, the multi-scale cell is appended at the top of the backbone. The architecture is fixed by $\alpha_{ms}$ after the procedure of CAS. The whole network initialized with the ImageNet pre-trained weights in backbone, is finally optimized on semantic segmentation.

## 4.3. Training Strategy

Our proposal is implemented on Caffe [14] framework with CUDNN, and mini-batch stochastic gradient descent algorithm is exploited to optimize the model. In the search procedure of CAS, the initial learning rate is 0.005. We exploit the "poly" learning rate policy with power fixed to 0.9. Momentum and weight decay are set to 0.9 and 0.0005, respectively. The batch size is 16. The maximum iteration number is $15k$. To evaluate the architecture generated by CAS, we train the whole network for $90k$ iterations. The rest hyper-parameters are the same as those in the search procedure of CAS.



(a) $1k$ iters, mIoU=62.4%, time=14.1ms (b) $5k$ iters, mIoU=64.6%, time=22.4ms (c) $15k$ iters, mIoU=68.1%, time=23.8ms

Figure 4. Examples of the normal cell and reduction cell during CAS procedure with the GPU Time constraint. The performance of the network is consistently increased from 62.4% to 68.1% with the increase of iterations, and the inference time converges to 23.8ms.

# 5. Experiments

In all experiments, the Intersection over Union (IoU) per category and mean IoU over all the categories are used as the performance metric. The resolution of the input image is $1024 \times 2048$, and the GPU/CPU inference time is reported on one Nvidia GTX 1070 GPU card and Intel i7 8700 CPU, respectively, unless otherwise stated.

## 5.1. Datasets

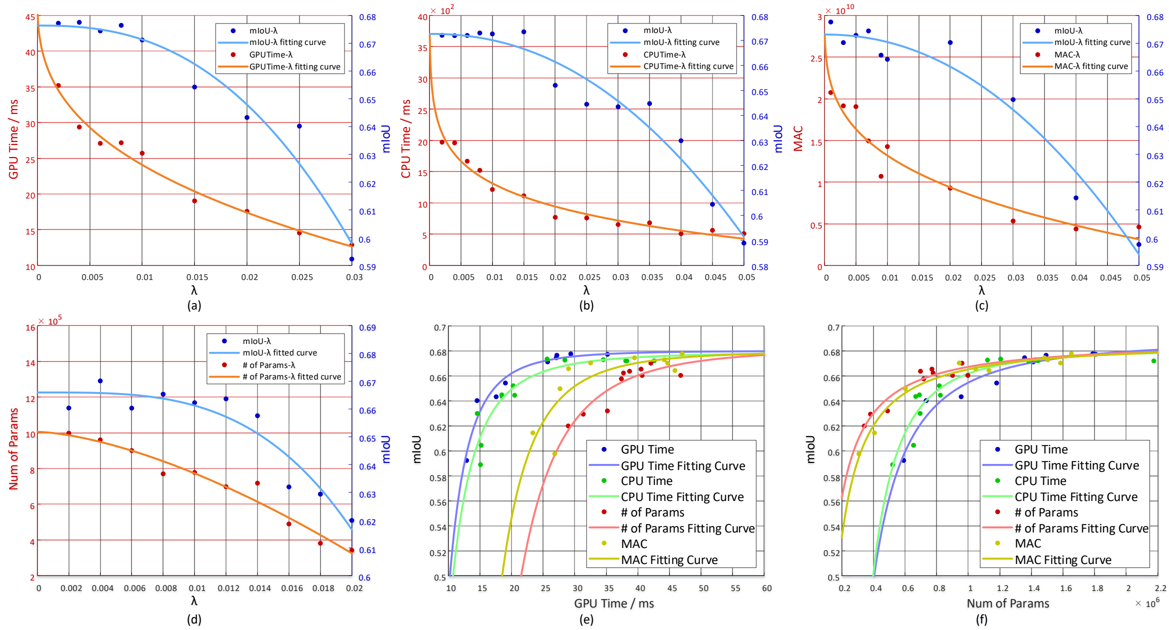
We conduct a thorough evaluation of CAS on Cityscapes [8], one popular benchmark for semantic understanding of urban street scenes. It contains high-quality pixel-level annotations of 5,000 images collected in street scenes from 50 different cities. The image resolution is $1024 \times 2048$. Following the standard protocol in segmentation task [8], 19 semantic labels are used for evaluation. The training, validation, and test sets contain 2975, 500, and 1525 images, respectively. An additional set of 23,473 coarsely annotated images are also available in this dataset. In our evaluation, the training set is further split into two groups, which play the roles of "training set" (1599 images from 9 cities) and "validation set" (1376 images from another 9 cities) in architecture search, respectively. Note that the original validation set or test set is never used for architecture search.

Moreover, we also evaluate the merit of CAS on the CamVid dataset, which is a standard scene parsing dataset. There are five video sequences in total with resolution up to $720 \times 960$. The sequences are densely labeled at one frame per second with 11 class labels. We follow the training/testing split in [2], with 468/233 labeled frames in the dataset for training/testing.

## 5.2. Evaluation of CAS

**Architecture search by CAS.** First, we conduct experiments to explore the evolution procedure of the architecture optimization given some constraints. The architecture search is performed on Cityscapes training set from the scratch and the searched architectures are evaluated on Cityscapes validation set. Figure 4 illustrates the architecture evolution of a normal cell and a reduction cell during the CAS optimization given the constraint on GPU time. Let us look at how a normal cell architecture changes during CAS optimization, which attempts to reach a tradeoff between network performance and GPU time. As shown in Figure 4(a), at the beginning of optimization, the cell selects the most lightweight operation "identity" and "pooled conv", which is able to immediately decrease the network computation by reducing the spatial resolution. As a result, the inference of the network is fast with relatively low mIoU, i.e., 62.4%@14.1ms. When iterating the search process $5k$ times, heavy operations (e.g., separable convolution and dilated convolution) are selected in pursuit of better performance by sacrificing some inference time (from 14.1ms (a) to 22.4ms (b)) as shown in Figure 4(b). The search converges after $15k$ iterations to reach a cell in Figure 4(c) and no extra heavy operations are employed after $5k$ iterations in our observations. The results indicate that CAS could optimize cells well with the constraints during architecture search. The whole search procedure of cells verifies our design that the performance and customized constraints of the network could be automatically balanced by CAS.

**CAS with different constraints.** Then, we conduct another group of experiments to demonstrate the effectiveness of CAS. More specifically, we examine the impact of the tradeoff parameter $\lambda$ towards a balance between segmentation performance and constraint costs. All the experiments are evaluated on Cityscapes validation set with networks trained on the training set from the scratch. The experiment on each setting is repeated five times, and the average values are reported. Figure 5(a)$\sim$5(d) depicts results under constraints of GPU time, CPU time, MAC, and Number of Parameters, respectively. The blue and red points/curves in the figure illustrate the mIoU and cost of networks given different $\lambda$ values, and the curves are fit to the points utilizing 2-terms power function. All the experiments con-



(a) (b) (c) (d) (e) (f)

Figure 5. (a) GPU Time-$\lambda$ and mIoU-$\lambda$ curves under the constraint of GPU time. (b) CPU Time-$\lambda$ and mIoU-$\lambda$ curves under the constraint of CPU time. (c) MAC-$\lambda$ and mIoU-$\lambda$ curves under the constraint of MAC. (d) Num of Params-$\lambda$ and mIoU-$\lambda$ curves under the constraint of number of parameters. (e) mIoU-GPUTime curves under four constraints. (f) mIoU-Num of Params curves under four constraints. Better viewed in original color pdf.

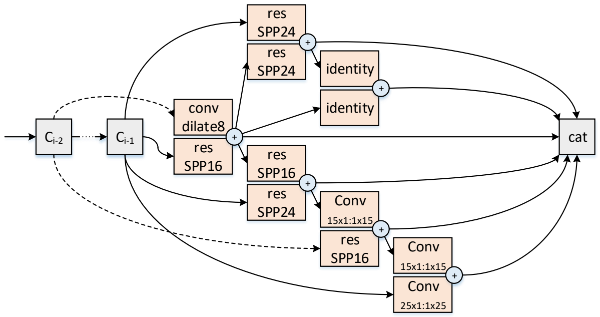


Figure 6. The architecture of the multi-scale cell.

sistently show that the network cost decreases rapidly with the increase of $\lambda$, resulting in the drop of the performance. Please also note that a small increment of $\lambda$ could lead to a significantly reduced cost but without notably sacrificing the performance, especially when $\lambda$ is relatively small. In other words, we could expect an affordable network whose performance is not much worse than that of the costly ones.

Next, we turn to compare the network design of CAS with respect to different constraints. Figure 5(e) and 5(f) shows the mIoU performances when utilizing GPU time and number of parameters as the measure of cost under each constraint, respectively. In the two figures, each curve depicts the performances of networks which are generated by CAS with the corresponding constraint. For instance, the blue and green curve in Figure 5(e) represents the performances of the networks optimized with constraints of GPU time and CPU time, respectively. As expected, optimizing networks when setting the alignment of constraint and the computation on cost will lead to better performance. Specifically, capitalizing on the constraint of GPU time constantly exhibits an mIoU boost over other constraints when computing cost on GPU time. Similarly, when the cost is calculated on number of parameters, the networks with the constraint of number of parameters achieve the best mIoU. The results indicate the flexibility of our CAS for architecture search with customizable constraints.

Table 1. Evaluation of pre-training and multi-scale cell.

| **Method** | | **mIoU (%)** | **Time (ms)** |
|---|---|---|---|
| CAS-GT | | 68.1 | 23.8 |
| +ImageNet Pre-train | | 70.4 | 23.8 |
| +MSC | PSP[37] | 71.5 | 26.5 |
| | ASPP[5] | 72.9 | 33.2 |
| | ASPP+[6] | 73.9 | 56.9 |
| | MSCell | **74.0** | **29.2** |

**Evaluation of the multi-scale cell.** The multi-scale cell is employed to recover the spatial information loss caused by the downsampling operations in the backbone network. Here, we study how the multi-scale cell influences the overall performance. Let CAS-GT be the best backbone network searched by CAS under the constraint of GPU time and $\lambda = 0.01$. The multi-scale cell is placed at the top of CAS-GT. The architecture of the multi-scale cell searched by CAS, which is denoted as MSCell, is illustrated in Figure 6. As the most frequently selected operation, the residu-

Table 2. mIoU and inference FPS on Ciytscapes validation (*val*) and test (*test*) sets. The mIoU and inference FPS of our method are given on the downsampled images with resolution $768 \times 1536$.

| Method | mIoU (%) | | FPS |
|---|---|---|---|
| | *val* | *test* | |
| FCN-8s [19] | - | 65.3 | 4.4 |
| Dilation10 [34] | 68.7 | 67.1 | 0.7 |
| PSPNet [37] | - | 81.2 | 1.3 |
| DeepLabv3 [5] | - | 81.3 | 1.3 |
| SegNet [1] | - | 57.0 | 33.0 |
| ENet [20] | - | 58.3 | 78.4 |
| SQ [31] | - | 59.8 | 21.7 |
| ICNet [36] | 67.7 | 69.5 | 37.7 |
| ICNet [36] (+coarse) | - | 70.6 | 37.7 |
| BiSeNet-Xception39 [33] | 69.0 | 68.4 | 105.8 |
| BiSeNet-Res18 [33] | 74.8 | 74.7 | 65.5 |
| CAS-GT+MSCell | 71.6 | 70.5 | 108.0 |
| CAS-GT+MSCell (+coarse) | 72.5 | 72.3 | 108.0 |

al pyramid pooling benefits from its capability of gathering the context information from large regions and preserving fine spatial information. Table 1 details the mIoU and GPU time of CAS-GT with and without the multi-scale cell. In our case, ImageNet pre-training successfully boosts up the mIoU performance from 68.1% to 70.4% without additional inference time. Utilizing multi-scale cells (MSC) at the top of ImageNet pre-trained CAS-GT could further increase the mIoU of the network. Particularly, PSP[37], ASPP[5] and ASPP+[6], which are manually designed multi-scale cells, obtain 1.1%, 2.5% and 3.5% performance gains with extra 2.7ms, 9.4ms and 33.1ms inference time, respectively. Compared to the manually designed ones, our MSCell leads to an mIoU increase of 3.6% and the mIoU performance reaches 74.0% with only 5.4ms additional inference time.

### 5.3. Real-time Semantic Segmentation

In this section, we validate CAS with the configuration of CAS-GT plus MSCell on the scenario of real-time semantic segmentation. The architecture search is optimized with the constraint of GPU time. We run all the inferences on an Nvidia TitanXp GPU card and calculate the frame per second (FPS) for all the methods. For fair comparisons, we measure the speed of the methods based on our implementations if the original speed was reported on different GPUs.

**Results on Cityscapes.** We evaluate CAS-GT+MSCell on Cityscapes validation and test sets. The validation set is included for training when submitting our network to online Cityscapes server and evaluating the performance on official test set. Following [33], we scale the resolution of the image from $1024 \times 2048$ to $768 \times 1536$, and measure the speed and mIoU without other evaluation tricks. Both the performance and FPS comparisons are summarized in Table 2. Overall, our CAS-GT+MSCell is the fastest among all the methods. Compared to BiSeNet-Xception39 [33] which is as fast as ours, CAS-GT+MSCell leads to an mIoU performance boost of 2.1% on the test set. Compared to the methods designed for high-speed semantic segmentation such as ENet [20], SQ [31] and ICNet [36], CAS-GT+MSCell achieves faster inference and makes performance improvement over them by 12.2%, 10.7% and 1.0%, respectively. The results demonstrate the effectiveness of our CAS for balancing performance and constraints. When additionally leveraging coarse annotations of Cityscapes, CAS-GT+MSCell yields the mIoU of 72.3% on test set.

Table 3. mIoU and inference FPS on CamVid test set. The mIoU and inference FPS of our method are given on the original images with resolution $720 \times 960$.

| Method | mIoU (%) | FPS |
|---|---|---|
| Dilation8 [34] | 65.3 | 6.5 |
| PSPNet50 [37] | 69.1 | 6.8 |
| SegNet [1] | 55.6 | 29.4 |
| ENet [20] | 51.3 | 61.2 |
| ICNet [36] | 67.1 | 34.5 |
| BiSeNet-Xception39[33] | 65.6 | - |
| BiSeNet-Res18[33] | 68.7 | - |
| CAS-GT+MSCell | **71.2** | **169.0** |

**Results on CamVid.** To validate the transferability of learnt architectures, we perform the experiments on CamVid with the cells searched on Cityscapes for real-time semantic segmentation. Note that we merely transfer the architectures of CAS-GT+MSCell but train the weights on CamVid. Table 3 details the comparisons of both performance and inference time on CamVid test set. The input resolution is $720 \times 960$. In particular, our CAS-GT+MSCell surpasses the best competitor BiSeNet-Res18 by 2.5% in mIoU. More importantly, the inference speed of CAS-GT+MSCell achieves 169 FPS, which is very impressive. The results basically verify the merit of CAS from the aspect of network generalization.

## 6. Conclusion

In this paper, we propose an approach to automatically generate a network architecture for semantic image segmentation. Unlike some previous approaches, which require huge efforts from human experts to manually design a network, our approach utilizes a lightweight framework, and automatically searches for optimized computation cells which are the building blocks of the network. In addition, our CAS takes the constraints of real applications into account when optimizing the architecture. As a result, ours is able to seek a good balance between segmentation performance and available computational resource. Experiments on both Cityscapes and CamVid datasets demonstrate the advantages over other state-of-the-art approaches.

**Acknowledgments.** This work was supported in part by the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant XDB06040900.

## References

[1] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Trans. on PAMI*, 39(12):2481–2495, 2017.

[2] Gabriel J. Brostow, Jamie Shotton, Julien Fauqueur, and Roberto Cipolla. Segmentation and recognition using structure from motion point clouds. In *ECCV*, 2008.

[3] Liang-Chieh Chen, Maxwell D. Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jonathon Shlens. Searching for efficient multi-scale architectures for dense image prediction. In *NIPS*, 2018.

[4] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Trans. on PAMI*, 40(4):834–848, 2018.

[5] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[6] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *ECCV*, 2018.

[7] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. In *CVPR*, 2017.

[8] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016.

[9] Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *IEEE Trans. on PAMI*, 35(8):1915–1929, 2013.

[10] Golnaz Ghiasi and Charless C. Fowlkes. Laplacian pyramid reconstruction and refinement for semantic segmentation. In *ECCV*, 2016.

[11] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D Sculley. Google vizier: A service for black-box optimization. In *SIGKDD*, 2017.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[13] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *CVPR*, 2018.

[14] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM MM*, 2014.

[15] Guosheng Lin, Anton Milan, Chunhua Shen, and Ian Reid. Refinenet: Multi-path refinement networks for high-resolution semantic segmentation. In *CVPR*, 2017.

[16] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *ECCV*, 2018.

[17] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *ICLR*, 2019.

[18] Wei Liu, Andrew Rabinovich, and Alexander C. Berg. Parsenet: Looking wider to see better. In *ICLR Workshop*, 2016.

[19] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

[20] Adam Paszke, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147*, 2016.

[21] Chao Peng, Xiangyu Zhang, Gang Yu, Guiming Luo, and Jian Sun. Large kernel matters – improve semantic segmentation by global convolutional network. In *CVPR*, 2017.

[22] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *ICML*, 2018.

[23] Pedro HO Pinheiro and Ronan Collobert. Recurrent convolutional neural networks for scene labeling. In *ICML*, 2014.

[24] Tobias Pohlen, Alexander Hermans, Markus Mathias, and Bastian Leibe. Full-resolution residual networks for semantic segmentation in street scenes. In *CVPR*, 2017.

[25] Zhaofan Qiu, Ting Yao, and Tao Mei. Deep quantization: Encoding convolutional activations with deep generative model. In *CVPR*, 2017.

[26] Zhaofan Qiu, Ting Yao, and Tao Mei. Learning spatio-temporal representation with pseudo-3d residual networks. In *ICCV*, 2017.

[27] Zhaofan Qiu, Ting Yao, and Tao Mei. Learning deep spatio-temporal dependence for semantic video segmentation. *IEEE Trans. on MM*, 20(4):939–949, 2018.

[28] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015.

[29] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[30] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[31] Michael Treml, José Arjona-Medina, Thomas Unterthiner, Rupesh Durgesh, Felix Friedmann, Peter Schuberth, Andreas Mayr, Martin Heusel, Markus Hofmarcher, Michael Widrich, et al. Speeding up semantic segmentation for autonomous driving. In *NIPS Workshop*, 2016.

[32] Fangting Xia, Peng Wang, Liang-Chieh Chen, and Alan L. Yuille. Zoom better to see clearer: Human and object parsing with hierarchical auto-zoom net. In *ECCV*, 2016.

[33] Changqian Yu, Jingbo Wang, Chao Peng, Changxin Gao, Gang Yu, and Nong Sang. Bisenet: Bilateral segmentation network for real-time semantic segmentation. In *ECCV*, 2018.

[34] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *ICLR*, 2016.

[35] Yiheng Zhang, Zhaofan Qiu, Ting Yao, Dong Liu, and Tao Mei. Fully convolutional adaptation networks for semantic segmentation. In *CVPR*, 2018.

[36] Hengshuang Zhao, Xiaojuan Qi, Xiaoyong Shen, Jianping Shi, and Jiaya Jia. Icnet for real-time semantic segmentation on high-resolution images. In *ECCV*, 2018.

[37] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *CVPR*, 2017.

[38] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018.