Context and Spatial Feature Calibration for Real-Time Semantic Segmentation

Kaige Li, Qichuan Geng, Maoxian Wan, Xiaochun Cao, *Senior Member, IEEE*, and Zhong Zhou

***Abstract*— Context modeling or multi-level feature fusion methods have been proved to be effective in improving semantic segmentation performance. However, they are not specialized to deal with the problems of pixel-context mismatch and spatial feature misalignment, and the high computational complexity hinders their widespread application in real-time scenarios. In this work, we propose a lightweight Context and Spatial Feature Calibration Network (CSFCN) to address the above issues with pooling-based and sampling-based attention mechanisms. CSFCN contains two core modules: Context Feature Calibration (CFC) module and Spatial Feature Calibration (SFC) module. CFC adopts a cascaded pyramid pooling module to efficiently capture nested contexts, and then aggregates private contexts for each pixel based on pixel-context similarity to realize context feature calibration. SFC splits features into multiple groups of sub-features along the channel dimension and propagates sub-features therein by the learnable sampling to achieve spatial feature calibration. Extensive experiments on the Cityscapes and CamVid datasets illustrate that our method achieves a state-of-the-art trade-off between speed and accuracy. Concretely, our method achieves 78.7% mIoU with 70.0 FPS and 77.8% mIoU with 179.2 FPS on the Cityscapes and CamVid test sets, respectively. The code is available at http://nave.vr3i.com/ and https://github.com/kaigelee/CSFCN.**

***Index Terms*— Real-time semantic segmentation, context feature calibration, spatial feature calibration.**

I. Introduction

SEMANTIC segmentation is a fundamental computer vision task that aims to assign a predefined category label to each pixel in the image. It is extensively applied in various practical applications, such as video surveillance, remote sensing, and automatic driving [1], [2].

Manuscript received 1 November 2022; revised 9 June 2023 and 24 August 2023; accepted 9 September 2023. Date of publication 29 September 2023; date of current version 5 October 2023. This work was supported in part by the National Natural Science Foundation of China under Grant 62272018 and in part by the National Key Research and Development Program of China under Grant 2018YFB2100603. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Nikolaos Mitianoudis. *(Corresponding author: Zhong Zhou.)*

Kaige Li and Maoxian Wan are with the State Key Laboratory of Virtual Reality Technology and Systems, Beihang University, Beijing 100191, China (e-mail: lkg@buaa.edu.cn; wanmaoxian@buaa.edu.cn).

Qichuan Geng is with the Information Engineering College, Capital Normal University, Beijing 100048, China (e-mail: gengqichuan1989@cnu.edu.cn).

Xiaochun Cao is with the School of Cyber Science and Technology, Shenzhen Campus, Sun Yat-sen University, Shenzhen 518107, China (e-mail: caoxiaochun@mail.sysu.edu.cn).

Zhong Zhou is with the State Key Laboratory of Virtual Reality Technology and Systems, Beihang University, Beijing 100191, China, and also with the Zhongguancun Laboratory, Beijing 100191, China (e-mail: zz@buaa.edu.cn).

This article has supplementary downloadable material available at https://doi.org/10.1109/TIP.2023.3318967, provided by the authors.

Digital Object Identifier 10.1109/TIP.2023.3318967

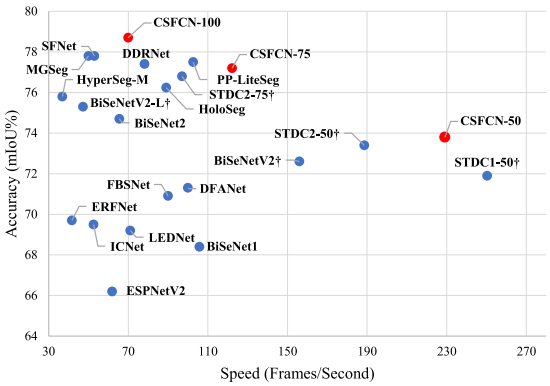
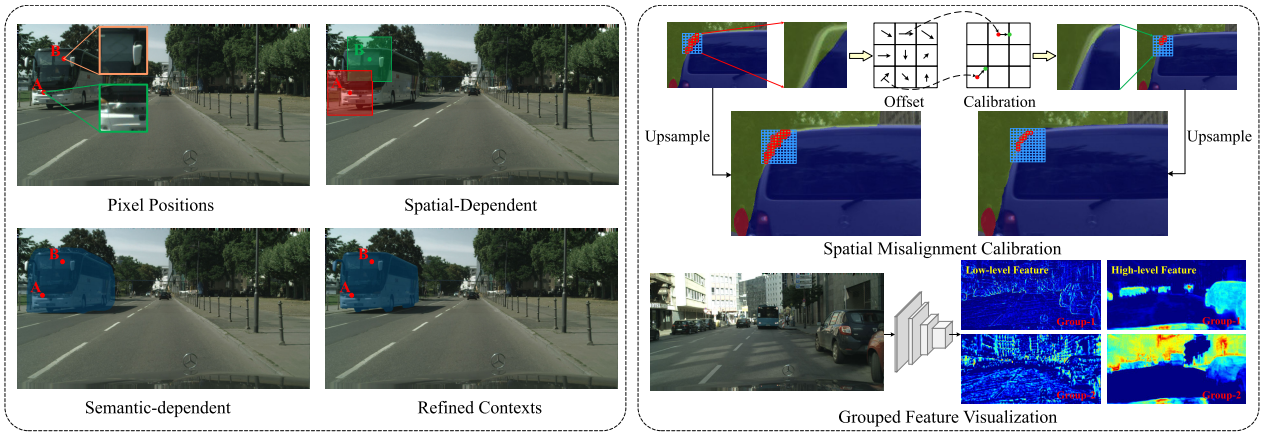


Fig. 1. Speed-accuracy performance comparisons on the Cityscapes test dataset. The speed of all methods is measured on a single GTX 1080Ti.

Recently, deep learning has brought a huge leap-forward to semantic segmentation with Fully Convolutional Networks (FCNs) [3]. Numerous accuracy-oriented methods have been proposed, which adopt deep networks to explore scene contexts [4], [5], [6] and spatial details [7], [8], [9] to ensure accuracy. However, they inevitably suffer from high computation, resulting in slow inference speeds. To this end, many efforts try to design highly lightweight networks [10], [11] or efficient decoders [12], [13] to realize real-time inference. Despite reducing computation, the information loss in lightweight networks leads to the misalignment of input and output. Besides, context modeling methods [14], [15] in existing decoders lack adaptability to different inputs, which leads to the pixel-context mismatch issue. These two issues hinder the performance of real-time methods, making it still challenging to achieve a satisfactory speed-accuracy trade-off.

We observe that the context mismatch mainly comes from the indiscriminate treatment in context modeling. Specifically, common methods [4], [5], [6] for aggregating contexts introduce non-adaptive contexts for each pixel, overlooking their inherent differences in context demands. As in Fig. 2(a), for pixels A and B, previous methods model spatial-dependent contexts for them within a predefined region. However, the activated context regions may be too large or too small, and these mismatched contexts will bring unexpected irrelevant information or cannot provide enough semantic clues. On the other hand, feature misalignment mainly arises from repeated downsampling, which causes spatial misalignment between the output (e.g., features or predictions) and the input image. This issue will be aggravated in parameter-free upsampling and introduce more prediction errors (especially at the boundaries), as in Fig. 2(b). Given the spatial attention (especially



(a) Context Mismatch (b) Spatial Misalignment

Fig. 2. Illustration of two challenges in semantic segmentation. (a) The spatial scope of different context modeling methods. Most existing methods (e.g., PPM [4] and ASPP [5]) model spatial-dependent contexts for all pixels in a fixed rectangle region. By contrast, our method aggregates semantic-dependent contexts and further refines them. (b) Visualization of feature misalignment and grouping. The blue and red grids indicate the region correctly and incorrectly predicted, respectively. For calibration, we replace the features at the red dot with the features at the green dot to get the output. Besides, we group the features for calibration since the features exhibit grouping properties.

self-attention) [16], [17] can capture beneficial features and suppress irrelevant information for each pixel, while their expensive computation overhead is unaffordable for real-time methods. To tackle the above issues, we propose a lightweight Context and Spatial Feature Calibration Network (CSFCN) with two simplified spatial attention ways to match the pixels and contexts and align features with the input. As shown in Fig. 1, our CSFCN achieves higher accuracy than other methods while satisfying real-time requirements.

Specifically, to mitigate the context mismatch issue, we propose a Context Feature Calibration (CFC) module. Unlike previous methods that aggregate contexts for each pixel in a fixed way, CFC models different contexts for each pixel, i.e., contextual features are functions of the input and also vary from pixel to pixel. Here we first design a cascade pyramid pooling module to efficiently capture the multi-scale contexts by reusing pooling results. Then, similar to self-attention [16], [18], but instead of computing pixel-to-pixel similarity, we compute pixel-to-context similarity to aggregate semantic-dependent contexts for each pixel, thus enabling context feature calibration. However, since the pooling contexts are easily biased towards features with large patterns, distributing them evenly to each location will overwhelm the representation of small patterns and cause over-smoothing results. Therefore, we further propose a Context Recalibration Block (CRB) to learn local contexts conditionally by sharpening the large objects and preserving spatial details, as shown in Fig. 2(a).

Furthermore, we introduce a Spatial Feature Calibration (SFC) module to tackle the feature misalignment issue. Stemming from self-attention [16], [18], but instead of aggregating information from all locations, SFC only collects information from a learned sampling location, which depend on the input. Intuitively, we simply sample the most favorable features for the prediction of the current location and place them at the current location, which greatly reduces the computation burden while enabling feature calibration. Moreover, we observe that feature maps carve various semantic information (e.g., an object or stuff, as in Fig. 2(b)), and there is often no single optimal calibration way for different feature maps. In other words, uniformly calibrating all feature maps will weaken the discrimination of the overall features. Consequently, for finer feature calibration, we first group the channel dimension into multiple sub-features and then perform calibration separately, which further improves the performance.

With these two modules, our CSFCN can efficiently mine the context and spatial information with little computation, resulting in leading performance, as shown in Fig. 1. To summarize, our contributions are three-fold:

1) We introduce two simplified self-attention ways to address the issues of context mismatch and feature misalignment while realizing real-time segmentation.
2) We design a pooling-based Context Feature Calibration module, which tailors the contexts for each pixel by matching pixels with pooling-based contexts.
3) We design a sampling-based Spatial Feature Calibration module, which samples the most representative and informative features for each location.

Comprehensive experiments on the Cityscapes [19] and CamVid [20] datasets show that our CSFCN achieves the state-of-the-art speed-accuracy trade-off. Specifically, our CSFCN achieves 78.7% mIoU with 70.0 FPS and 77.8% mIoU with 179.2 FPS on the Cityscapes and CamVid test set, respectively.

II. Related Work

A. Generic Semantic Segmentation

Most existing methods for semantic segmentation are based on FCNs [3], while they have an inherent weakness, i.e., the local receptive fields. Recent progresses mainly employ two strategies to improve performance, i.e., context modeling [16], [21] and feature fusion [17], [22].

1) Context Modeling: PSPNet [4] uses a pyramid pooling module (PPM), DeepLabV3+ [5], DenseASPP [6] and DUC [23] use an atrous spatial pyramid pooling (ASPP) module to integrate multi-scale contexts to improve accuracy.

However, they model homogeneous contextual information for all pixels, inevitably leading to pixel-context mismatch. Thus, some works [16], [24] exploit self-attention to encode the global contexts for each pixel adaptively, but their expensive computation is problematic in real-time scenarios. ANNN [21] propose to leverage a pyramid sampling module to decrease computation of self-attention while keeping the ability to harvest global contexts. However, ANNN still lacks the multiscale context extraction capability. In this regard, we embed a cascaded pyramid pooling block into self-attention and introduce a context refinement strategy to customize the multi-scale contexts for each pixel while reducing computation.

2) Feature Fusion: Previous feature fusion methods [7], [8] can effectively improve performance but often ignore feature misalignment problem. Therefore, SFNet [17] and AlignSeg [22] propose to align multi-level features by learning 2D transformation offsets for better feature fusion. However, they overlook the representation differences between different level features and sub-features of the same level, which weakens the effectiveness of information propagation. In contrast, we treat different feature maps and different level features in different ways, leading to more accurate alignment and fusion.

3) Efficient Transformer: The multi-head self-attention (MHSA) layer in Transformer enables it to model the global context naturally, which is essential for semantic segmentation. To accommodate various dense prediction tasks, PVT [25] proposes to use a linear projection to decrease the length of the input sequence, thus reducing the computation of MHSA. P2T [26] proposes to adapt pyramid pooling to the MHSA to simultaneously reduce the sequence length and learn rich contextual representations. However, both PVT and P2T only focus on building an efficient transformer backbone, while ignoring the design of the decoder. Meanwhile, their proposed module is also unable to model multi-scale information.

B. Real-Time Semantic Segmentation

Real-time semantic segmentation aims to produce high-quality predictions while maintaining high efficiency. In this case, three main strategies are available: (1) *Input restricting.* Such methods directly reduce computation by restricting the input size. For instance, ENet [27] and ERFNet [28] downsample the input for faster inference. Though this way is simple and effective, much information loss leads to dramatic degradation of accuracy. (2) *Network compression.* Such methods focus on network compression, especially channel pruning and convolution factorization. For example, ENet [27] and FRNet [10] adopt factorized convolution, ESPNetV2 [29], NDNet [30], and FBSNet [31] adopt depth-wise separable convolution to reduce computation. This is also an efficient method, but it sacrifices the spatial capacity and representational ability of the low-level features. (3) *Multi-branch structure.* As the above two strategies are difficult to achieve satisfactory performance, most recent methods adopt multi-branch structures to capture multi-scale information to improve accuracy. For example, BiSeNet series [32], [33] utilize a two-branch structure to encode spatial and semantic information separately. DFANet [34] uses a multi-branch framework to effectively combine multi-level features. DMA-Net [35] recursively fuses the features from the high- and the low-level branch to produce more robust features. SwiftNet [9] leverages lateral connections to fuse the features of different resolution branches to handle multi-scale problems. STDC-Seg [36] proposes a backbone network specifically designed for semantic segmentation, and replaces the spatial path in BiSeNet [32] with a detail aggregation module, thereby improving efficiency without compromising performance.

Despite achieving impressive performance, the above methods generally overlook the issues of context mismatch and feature misalignment. In this paper, we design a lightweight CSFCN with two simplified attention mechanisms to efficiently address these issues while performing real-time segmentation with improved accuracy.

III. Methodology

In this section, we first give an overview of CSFCN, and then expatiate the details of its two components.

A. Network Architecture

The overall architecture of the proposed CSFCN is illustrated in Fig. 3, which adopts an asymmetric encoder-decoder architecture. Specifically, CSFCN employs a lightweight backbone network to extract multi-level features. After that, we devise the Context Feature Calibration module to build private contexts for each pixel to enhance its discrimination. Further, we use the Spatial Feature Calibration module for spatial feature calibration to produce strong semantic features with precise boundaries. Without loss of generality, we adopt pre-trained ResNet-18 [37] from the ImageNet [38] as the backbone network, and other CNNs can also be used as the backbone in our method.

B. Context Feature Calibration Module

Context information can provide rich scene category priors to correct unexpected misclassification. However, previous methods [4], [5], [6] aggregate contexts within predefined regions and overlook that not all contexts contribute equally to classify a given pixel, which inevitably leads to the problem of context mismatch. Besides, the captured contexts are heavily biased towards large objects as they contain more pixels, and lead to over-smoothing results for small objects. Based on the above insights, we propose a Context Feature Calibration module to tailor and refine semantic contexts for each pixel.

Typically, large objects or stuff dominate the image, and the global contexts are often similar to them while different from spatial details. Inspired by this intuition, we match the contexts with each pixel to catch the most instructive context for its classification. That is, we aggregate the contexts from the semantic-closer region instead of the spatial-closer one. Concretely, given feature $\mathbf{X} \in \mathbb{R}^{C\times H\times W}$, we first process $\mathbf{X}$ to capture highly-abstracted multi-scale contexts $\mathbf{Z} \in \mathbb{R}^{C\times M}$. Then, we compute the pixel-context similarity $\Theta \in \mathbb{R}^{N\times M}$, a.k.a., spatial attention map, where $N = H \times W$ and M denote the total number of pixels and contexts, respectively. We employ Θ as guidance to aggregate contexts for each

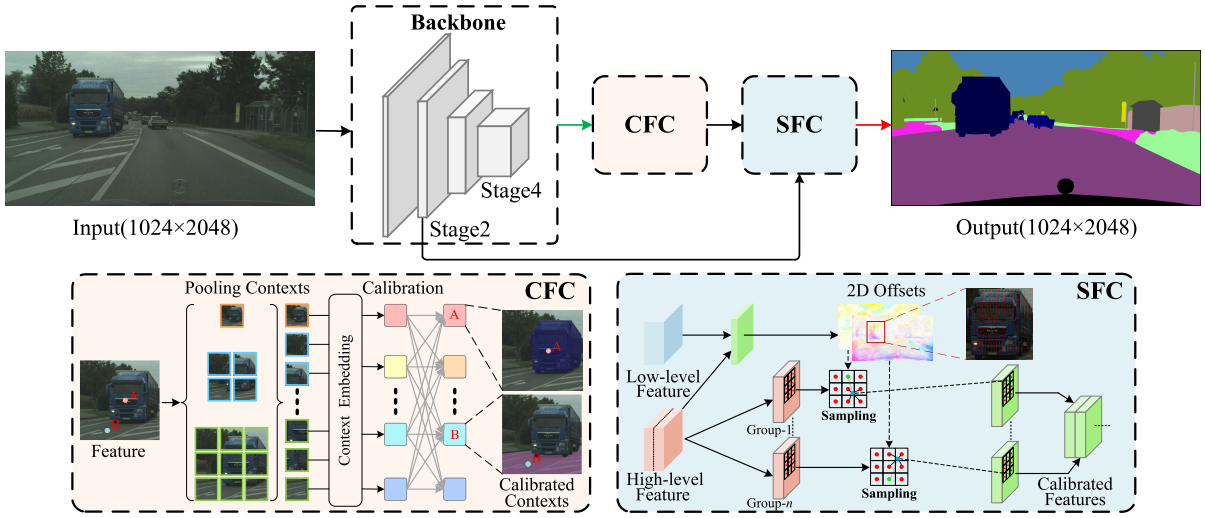


Fig. 3. Overview of CSFCN. Given an input image, we first use the backbone network to extract features of different stages. Then we adopt a 3×3 convolution layer (green arrow) with a reduction ratio r ($r = 2$ by default) to squeeze the features to reduce the computation cost. Afterward, we use CFC and SFC to calibrate features to produce more robust and discriminative features. Finally, we employ an output layer (red arrow) containing a convolutional layer and an upsampling layer to generate the final output. For 2D offsets, we visualize them by color coding. Best viewed in color and zoom in.

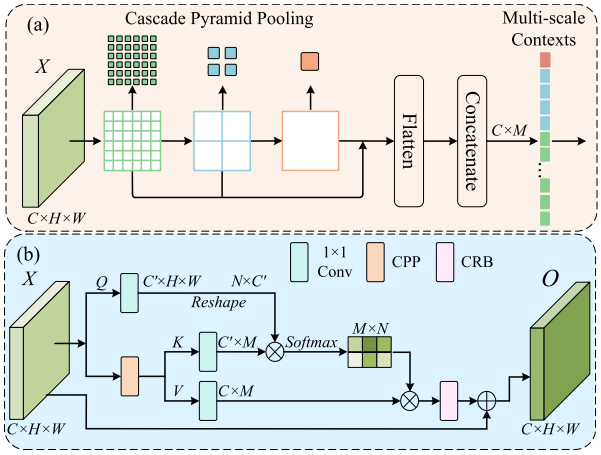


Fig. 4. (a) Demonstration of our cascade pyramid pooling layer. We first employ several cascade adaptive pooling layers to generate pooling results, then flatten and concatenate them to obtain the pyramid multi-scale contexts. (b) The implementation procedure of the Context Feature Calibration module.

pixel to realize context feature calibration. Finally, we further adjust the response values of each semantic context to generate fine-grained contexts, thereby enabling context recalibration. Mathematically, our CFC can be defined as:

$$\mathbf{y}_i = \alpha_i \cdot \sum_{j=1}^{M} f\left(\mathbf{x}_i, \mathbf{z}_j\right) \cdot \mathbf{z}_j + \mathbf{x}_i \tag{1}$$

where $\{\mathbf{x}_i, \mathbf{y}_i, \alpha_i, \mathbf{z}_j\} \in \mathbb{R}^{C\times1}$ represent the input, output, recalibration factor and context, respectively. i ranges in $[1, \cdots, H \times W]$, $f(\cdot)$ denotes a pairwise function to calculate the affinity between features.

1) Module Details: We illustrate our CFC module in Fig. 4. In pursuit of efficiency, we design Cascaded Pyramid Pooling (CPP) block to reuse the pooling results of previous layers, which reduces unnecessary redundant computations. As in Fig. 4, given a feature $\mathbf{X} \in \mathbb{R}^{C\times H\times W}$, we first employs a 1×1 convolutional layer to generate a reduced-dimensional feature $\mathbf{Q} \in \mathbb{R}^{C'\times H\times W}$, where C' is much less than C (By default, $C' = 32$, $C = 256$). Then, we use the CPP block to harvest the multi-scale contexts $\mathbf{Z} \in \mathbb{R}^{C\times M}$. In particular, when the pooling layer's output height $n \in [1, 2, 3]$ (generally, the output width equals the height), $M = \sum_{n\in[1,2,3,6]} n^2 = 50$. By default, we set $n = [1, 2, 3, 6]$. Since the pooling layer performs pooling over a homogeneous spatial grid (e.g., 3×3), this may lead to information redundancy of the pooled features when the aspect ratio of the input is not 1. To this end, we keep the aspect ratio of the pooling output size to be equal to the input. For example, our smallest output size is 1×1 when training on square crops, while its size will be 1×2 for a 1024×2048 input. Finally, we feed $\mathbf{Z}$ into two convolution layers (with BN and ReLU) to produce two forms of context representation, i.e., $\mathbf{K} \in \mathbb{R}^{C'\times M}$ and $\mathbf{V} \in \mathbb{R}^{C\times M}$.

Afterward, we reshape and transpose $\mathbf{Q}$ to $\mathbb{R}^{N\times C'}$ and take a matrix multiplication between $\mathbf{Q}$ and $\mathbf{K}$, and add a softmax layer to produce pixel-context affinity $\Theta \in \mathbb{R}^{N\times M}$:

$$\Theta = \frac{exp\left(\mathbf{Q}_i \cdot \mathbf{K}_j\right)}{\sum_{j=1}^{M} exp\left(\mathbf{Q}_i \cdot \mathbf{K}_j\right)} \tag{2}$$

where $\Theta_{i,j}$ represents the affinity between i^{th} pixel $\mathbf{Q}_i$ and j^{th} context $\mathbf{K}_j$. Finally, we perform a matrix multiplication between $\mathbf{V}$ and Θ^T to attain the calibrated semantic context $\mathbf{E} \in \mathbb{R}^{C\times N}$ and reshape $\mathbf{E}$ to $\mathbb{R}^{C\times H\times W}$. After obtaining the calibrated context $\mathbf{E}$, we send it to the Context Recalibration Block (CRB) to generate the refined context $\mathbf{E}'$. CRB adopts residual-like design [37], which can be formulated as:

$$\mathbf{E}' = \overbrace{\tanh\left(\mathbf{W}_2\left(\mathbf{W}_1\left(\mathbf{X} + \mathbf{E}\right)\right)\right)}^{\alpha} \cdot \mathbf{E} + \mathbf{E} \tag{3}$$

where $\alpha \in \mathbb{R}^{C\times H\times W}$ denotes recalibration factor, $\mathbf{W}_1 \in \mathbb{R}^{\frac{C}{4}\times C\times1\times1}$ and $\mathbf{W}_2 \in \mathbb{R}^{C\times\frac{C}{4}\times3\times3}$ denote convolution layers. Here, we choose the tanh function to remove redundant information and highlight beneficial information (e.g., boundaries and small objects) in the contexts. Finally, we conduct an element-wise summation among $\mathbf{X}$ and $\mathbf{E}'$ to generate the final

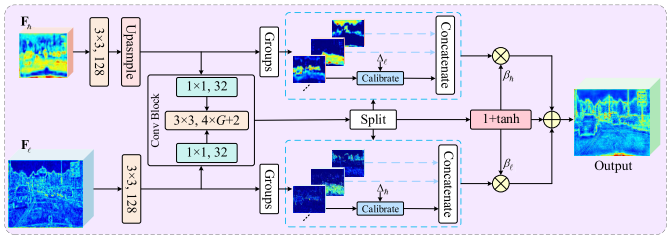


Fig. 5. The details of the Spatial Feature Calibration module.

output $\mathbf{Y} \in \mathbb{R}^{C\times H\times W}$ as follows:

$$\mathbf{Y} = \mathbf{X} + \mathbf{E}' \tag{4}$$

Here we adopt summation instead of concatenation operation to reduce the computational costs.

C. Spatial Feature Calibration Module

To remedy the loss of spatial details caused by the stepwise downsampling, previous methods [5], [8], [9] adopt cross-level feature fusion to enhance the high-level semantic feature with low-level details. Given low-resolution feature $\mathbf{F}_\ell \in \mathbb{R}^{C_\ell\times H_\ell\times W_\ell}$ and high-resolution feature $\mathbf{F}_\hbar \in \mathbb{R}^{C_\hbar\times H_\hbar\times W_\hbar}$, they first upsample $\mathbf{F}_\ell$ via the standard bilinear interpolation, and then add or concatenate $\mathbf{F}_\hbar$ and the upsampled $\mathbf{F}_\ell$ to obtain the output. However, due to the spatial misalignment and huge representation gap, directly fusing them still cannot achieve satisfactory performance. Besides, as feature map encode various semantic information [39], performing unified feature alignment along the channel dimension [17], [22] would also harm performance. To mitigate these issues, we group the dimension of channels into multiple sub-features to perform calibration operations separately, and seamlessly integrate the gating mechanism to fuse cross-level features adaptively.

Regarding feature calibration, we propose to adopt feature resampling to reconstruct features. Concretely, suppose the spatial coordinates of each location on the feature map are $\{(1,1),(1,2),\ldots,(H,W)\}$ and the learned 2D offset map is $\Delta \in \mathbb{R}^{2\times H\times W}$. Our calibration function $\mathcal{T}(\cdot)$ can be formulated as:

$$\begin{aligned}\mathbf{U}_{h,w} = \sum_{h'}^{H}\sum_{w'}^{W} \mathbf{F}_{h',w'} \cdot max(0, 1 - \left|h + \Delta^1_{h,w} - h'\right|) \\ \cdot max(0, 1 - \left|w + \Delta^2_{h,w} - w'\right|)\end{aligned} \tag{5}$$

which sample feature at $p = (h+\Delta^1_{h,w}, w+\Delta^2_{h,w})$ to derive the output $\mathbf{U}_{h,w}$. Since p denotes an arbitrary (fractional) location, Eq. 5 enumerates all integral locations and uses bilinear interpolation kernel to obtain the features of the sampling location. Intuitively, as shown in the **Sampling** process in SFC in Fig. 3, we simply sample the most favorable feature for the current location (blue point) to replace the current location (green point) feature for calibration. Besides, for finer calibration, we split feature $\mathbf{F}$ into G groups along the channel dimension, and then align the features within each group separately, as depicted in Fig. 5.

As $\mathbf{F}'_\hbar$ generally contain rich spatial details, while $\mathbf{F}'_\ell$ contain more semantics, merely calibrating and then fusing them cannot yield satisfactory performance. This is because feature calibration alone cannot handle the huge representational differences between features. Thus, we further propose to fuse calibrated semantic features $\mathbf{F}'_\ell$ and fine-grained features $\mathbf{F}'_\hbar$ adaptively through a gating strategy, bridging the representation gap between them:

$$\mathbf{O} = \beta_\ell \otimes \mathbf{F}'_\ell + \beta_\hbar \otimes \mathbf{F}'_\hbar \tag{6}$$

where β_ℓ and $\beta_\hbar$ denote the gate masks.

1) Module Details: For efficiency, we integrate the calibration and fusion procedure in a single Spatial Feature Calibration module. As depicted in Fig. 5, given feature $\mathbf{F}_\ell \in \mathbb{R}^{C_\ell\times H_\ell\times W_\ell}$ and $\mathbf{F}_\hbar \in \mathbb{R}^{C_\hbar\times H_\hbar\times W_\hbar}$, we first unify their channels to the same number C (= 128 by default) by two convolution layers. Secondly, we adopt bilinear interpolation to upsample $\mathbf{F}_\ell$. Then, the upsampled $\mathbf{F}_\ell$ and $\mathbf{F}_\hbar$ are concatenated. After that, we feed them into a convolution block to predict two groups of offset maps $\Delta_\ell \in \mathbb{R}^{(2\times G)\times H\times W}$ and $\Delta_\hbar \in \mathbb{R}^{(2\times G)\times H\times W}$ for aligning two levels of features and two gate masks $\beta_\ell, \beta_\hbar \in \mathbb{R}^{1\times H\times W}$ for controlling the flow of feature information at two levels. Finally, the calibrated cross-level features perform element-wise summation to obtain the output. In summary, our SFC module can be formally written as:

$$\mathbf{O} = \beta_\ell \otimes \mathcal{T}\left(\mathcal{U}\left(\mathbf{W}_\ell\mathbf{F}_\ell\right), \Delta_\ell\right) + \beta_\hbar \otimes \mathcal{T}\left(\mathbf{W}_\hbar\mathbf{F}_\hbar, \Delta_\hbar\right) \tag{7}$$

where $\mathcal{U}(\cdot)$ denotes bilinear upsample function, $\mathbf{W}_\ell \in \mathbb{R}^{C_\ell\times C'\times 3\times 3}$ and $\mathbf{W}_\hbar \in \mathbb{R}^{C_\hbar\times C\times 3\times 3}$ denote convolutional layer with BN and ReLU.

Note that we adopt the residual idea to alleviate the negative effects of large offset and mask prediction errors in the initial phase, which allows us to insert SFC into the network without compromising its original performance (if the convolution block is constructed as zero mapping). That is, we initialize the weight of the last convolutional layer of the convolution block to **zero** to gradually learn more accurate offsets and masks. Besides, we opt to adopt a $1+\tanh$ activation for the gate masks. Thus, initially, $\mathcal{T}(\mathbf{F}, 0)$ becomes an identity mapping and $\beta = 1 + \tanh(0) = 1$. Now, SFC can be formulated as:

$$\mathbf{O} = \mathcal{U}\left(\mathbf{W}_\ell\mathbf{F}_\ell\right) + \mathbf{W}_\hbar\mathbf{F}_\hbar \tag{8}$$

At this time, our SFC module is equivalent to the simple feature fusion strategy in [3], [8], and [9].

IV. Experiments

To verify the superiority of our method, we conduct thorough experiments on Cityscapes [19] and CamVid [20] datasets, whose details are shown in Table I. In the following, we first introduce the implementation details of our method. Then, we conduct ablation studies on the Cityscapes to discuss the effect of our proposed modules. Finally, we make detailed comparisons with the state-of-the-art (SOTA) works on two benchmarks to illustrate the effectiveness of our method.

TABLE I
SEMANTIC SEGMENTATION DATASET STATISTICS

Dataset	Purpose	Classes	Resolution	Samples (training)	Samples (validation)	Samples (test)	Samples (total)
Cityscapes [19]	Urban	30 (19)	1024×2048	2975	500	1525	5000
CamVid [20]	Urban (Driving)	32 (11)	960×720	367	100	233	701

A. Implementation Details

1) Baseline: To validate the proposed modules, we first build a Baseline model, which is an FCN32-like [3] ResNet-18 network with an auxiliary supervision branch [4]. More details about the Baseline are in Supplementary Material.

2) Training: We train our network using stochastic gradient descent (SGD) with momentum 0.9, weight decay 5e-4. The batch size is set to 12 and 16 for the Cityscapes and CamVid, respectively. Following the previous work [36], [40], the initial learning rate is set to $1e-2$ with a "ploy" learning rate policy in which the learning rate is attenuated by $\left(1-\frac{iter}{total_{iter}}\right)^{0.9}$. For data augmentation, we adopt random color jittering, random horizontal flipping, random cropping, and random scaling. The scale ranges in $[0.125, 2.0]$, and the crop size is 1024×1024 for training the Cityscapes. For CamVid, the scale ranges in $[0.5, 2.0]$ and the crop size is 720×960. Finally, we train our networks for $120K$ and $40K$ iterations for the Cityscapes and CamVid datasets, respectively.

3) Testing: We use mean Intersection-over-Union (mIoU) and Frames Per Second (FPS) to measure the accuracy and latency, respectively, and use the float-point operations (FLOPs) and model parameters (Params) to evaluate the model complexity. Note, similar to [9] and [15], we exclude the BN layers during testing since they can be merged with the convolution layer, which can further improve inference speed.

B. Results on the Cityscapes Dataset

We perform ablation studies to understand each module. In the follow-up experiments, we train the network on the Cityscapes training set and evaluate it on the validation set.

1) Context Feature Calibration: In our method, the granularity of the contexts significantly affects performance, largely depending on the value of M, which is determined by the pooling layer output size. To this end, we perform a set of comparative experiments by changing the output size. As in Table II, keeping the aspect ratio (KAR) of the pooling layer input and output consistent can improve performance (76.96% vs. 77.59%) visibly without adding excessive computation, which comes from the more fine-grained contexts. Besides, we find that the performance increases as the number of contexts (M) increases, and the performance plateaus when the output size exceeds $(1, 2, 3, 6)$. Note that overly detailed divisions (e.g., $(1, 3, 6, 8)$) will cause performance degradation (77.27% vs. 77.59%), possibly because the output contains too little contextual information to provide high-quality semantic cues. Ultimately, considering the compromise between efficiency and accuracy, we adopt $(1, 2, 3, 6)$ as the default setting.

Table III shows the results from various context modeling methods. For fair comparisons, we reproduce all compared modules under the same settings and uniformly append them to the dimensionality reduction layer (DRL, the green arrow in Fig. 3). As in Table III, our CFC is superior to other counterparts under similar or lower computing costs. For example, CFC attains 78.09% mIoU, which is 0.83 points higher than that of AlignCM [22] (77.26%) when FLOPs are lower. In particular, CFC achieves the best performance with faster speed, surpassing PAM [16] (77.56% vs. 78.09%), while our FLOPs are 81.4% fewer.

2) Visual Analysis: We first select some representative images from the Cityscapes to further discuss the effect of the CFC module. As in Fig. 6, since the contexts obtained by pooling are often dominated by salient objects or stuff, the prediction of inconspicuous or incomplete objects by PPM [4] is somewhat weakened or even disregarded (e.g., the "motorcycle" in the first row, the "traffic sign" in the second row). By contrast, our CFC calibrates context features to tailor the contexts for each pixel, which avoids the adverse effects of salient objects and achieves better segmentation results.

TABLE II
ABLATION STUDY FOR CFC WITHOUT CRB. "n" COLUMN INDICATES THE OUTPUT HEIGHT/WIDTH OF THE ADAPTIVE POOLING LAYER. WE COMPUTE FLOPS OF DIFFERENT METHODS EXCLUDING THE BACKBONE, DRL, AND ALL OUTPUT LAYERS, WHERE THE INPUT SIZE IS $3 \times 1024 \times 2048$

n	KAR	M	FLOPs (G)	Speed (FPS)	mIoU(%)
$(1, 2, 3, 6)$	w/o	50	0.08	77.27	76.96
$(1, 2, 3, 6)$	w/	100	0.14	76.57	**77.59**
$(1,)$	w/	2	**0.02**	**77.36**	75.38
$(6,)$		72	0.11	77.07	76.95
$(1, 2, 3)$		28	0.05	77.31	76.51
$(1, 2, 6)$		82	0.12	76.99	77.19
$(1, 3, 6)$		92	0.14	76.85	77.43
$(1, 2, 3, 6)$		100	0.14	76.57	**77.59**
$(1, 3, 6, 8)$		220	0.29	76.26	77.27

TABLE III
COMPARISONS OF DIFFERENT CONTEXT MODELING METHODS. WE COMPUTE FLOPS OF DIFFERENT METHODS EXCLUDING THE BACKBONE, DRL, AND ALL OUTPUT LAYERS. FPS IS MEASURED ON GTX 1080 TI. THE INPUT SIZE IS $3 \times 1024 \times 2048$

Method	Params (M)	FLOPs (G)	Speed (FPS)	mIoU(%)
Baseline	-	-	**77.93**	72.69
PPM [4]	**0.07**	**0.02**	75.99	76.45(3.76 ↑)
ASPP [5]	2.23	4.44	66.82	76.57(3.88 ↑)
AlignCM [22]	0.76	0.83	73.09	77.26(4.57 ↑)
DAPPM [41]	0.87	1.56	74.94	77.36(4.67 ↑)
APNB [21]	0.27	0.71	73.28	77.37(4.68 ↑)
CAM [16]	0.00	0.27	74.59	76.99(4.30 ↑)
PAM [16]	0.08	2.58	71.82	77.56(4.87 ↑)
CFC (w/o CRB)	0.08	0.14	76.57	77.59(4.90 ↑)
CFC (w/ CRB)	0.25	0.48	76.55	**78.09(5.40 ↑)**

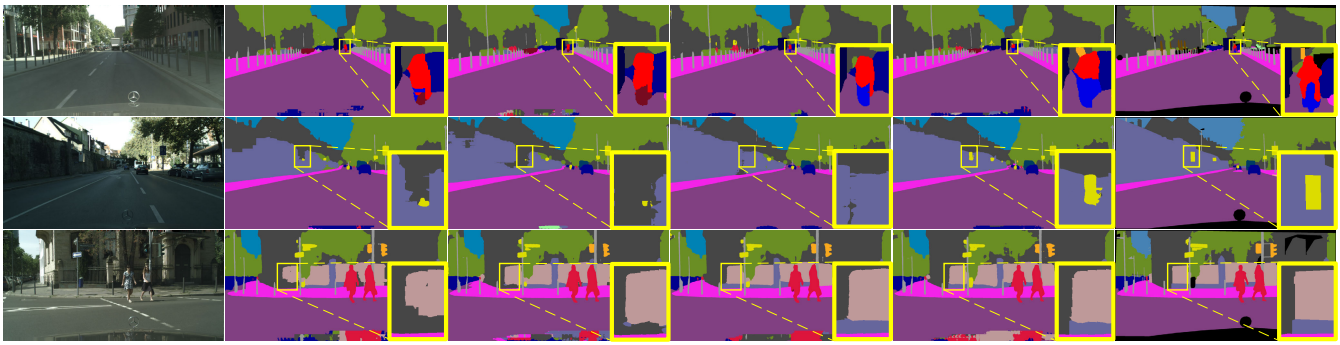


Fig. 6. Qualitative comparison against different context modeling modules. Best viewed in color.

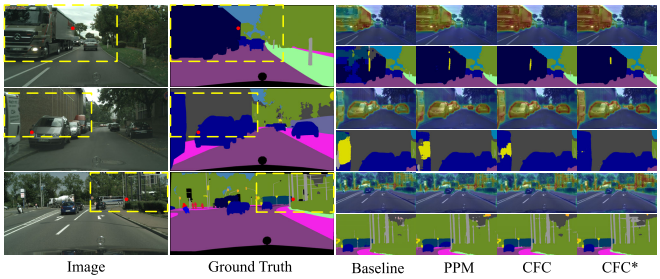


Fig. 7. Visualization results of various methods. We denote CFC and CFC* as not performing and performing context recalibration, respectively. We randomly sample a point and compute its similarity against the whole feature map. We visualize the semantic segmentation output for several challenging regions in the second row. Hot colors represent larger values and vice versa. Best viewed in color and zoom in.

In parallel, we visualize the cosine similarity between the selected point and the whole feature (generated from CFC) to further analyze the mechanism of our CFC module. As in Fig. 7, CFC can produce purer features than other competitors, which indicates the features of a certain class in CFC are hardly confused with others. For example, the similarity for class "truck" and "car" in CFC is more complete and clearer, which shows that CFC can reduce the misclassification or inconsistent predictions inside large objects. In contrast, PPM [4] that ignores pixel-context relationships tends to mislabel these regions (e.g., the "bus" in the third image). In addition, context recalibration can further improve the performance (e.g., the "car" in the second image) and make our method more friendly to small objects (e.g., the "pole" in the third image).

3) Spatial Feature Calibration: We append the SFC module to the Stage4 of the backbone without the DRL to investigate its capability. Note that if we place SFC after DRL (like CFC), its computation and performance will decrease. Experimental results of different settings are presented in Table IV. We can see that performing grouping calibration can obviously improve performance. For example, when using $1 + \tanh$ activation, setting $G = 2$ outperforms $G = 1$ (i.e., w/o grouping) by 0.97% (77.15% vs. 78.12%) with only a minor extra computation burden (6.73G vs. 6.80G). Meanwhile, $1 + \tanh$ also achieves better performance than sigmoid (77.37% vs. 78.12%), probably because of its superior initial performance. Besides, different grouping numbers also bring different performance improvements. Finally, considering the trade-off between performance and efficiency, we choose $G = 2$ and $1 + \tanh$ activation as the default settings.

TABLE IV

ABLATION STUDY OF DIFFERENT SETTINGS FOR SFC. WE COMPUTE FLOPS OF DIFFERENT METHODS EXCLUDING THE BACKBONE AND ALL OUTPUT LAYERS, WHERE THE INPUT SIZE IS $3 \times 1024 \times 2048$

Groups	Activation	Params (M)	FLOPs (G)	Speed (FPS)	mIoU(%)
$G = 1$	$1 + \tanh$	0.754	6.73	70.58	77.15
$G = 2$	sigmoid	0.756	6.80	70.34	77.37
	1+tanh (w/ DRL)	**0.461**	**6.20**	**70.81**	77.66
	1+tanh	0.756	6.80	70.02	78.12
$G = 4$	$1 + \tanh$	0.761	6.96	69.96	77.95
$G = 8$	$1 + \tanh$	0.770	7.26	69.43	**78.19**

TABLE V

PERFORMANCE COMPARISONS OF DIFFERENT METHODS. WE COMPUTE FLOPS OF DIFFERENT METHODS EXCLUDING THE BACKBONE AND ALL OUTPUT LAYERS, WHERE THE INPUT SIZE IS $3 \times 1024 \times 2048$

Method	Params (M)	FLOPs (G)	Speed (FPS)	mIoU(%)
Baseline	-	-	**77.93**	72.69
AFNB [21]	0.856	4.16	66.22	73.62(0.93 ↑)
iGUM [42]	0.661	**1.35**	39.45	74.07(1.38 ↑)
AlignFA [22]	0.825	22.74	59.71	75.25(2.56 ↑)
FAM [17]	**0.599**	1.39	72.87	76.21(3.52 ↑)
iAFF [43]	1.382	13.46	61.55	77.22(4.53 ↑)
Fusion	0.751	6.49	70.57	75.09(2.40 ↑)
Calibration	0.753	6.69	70.23	77.45(4.76 ↑)
SFC	0.756	6.80	70.02	**78.12(5.43 ↑)**

In addition, we also compare SFC with its similar methods. Table V presents the results, where "Calibration" and "Fusion" denotes that SFC only performs feature calibration or gated fusion. For a fair comparison, we also reproduce all compared modules under the same settings. It is observed that iGUM [42] which directly corrects the predicted output is difficult to obtain high accuracy. Besides, FAM [17] and AlignFA [22], which align different level features and directly blend the multi-level features, ignoring the differences between different level features and sub-features, resulting in poor performance (76.21% and 75.25% mIoU). Meanwhile, iAFF [43], which adopts attentional feature fusion, achieves relatively superior performance (77.22%). In comparison, our SFC not only calibrates the sub-features in each group of multi-level features

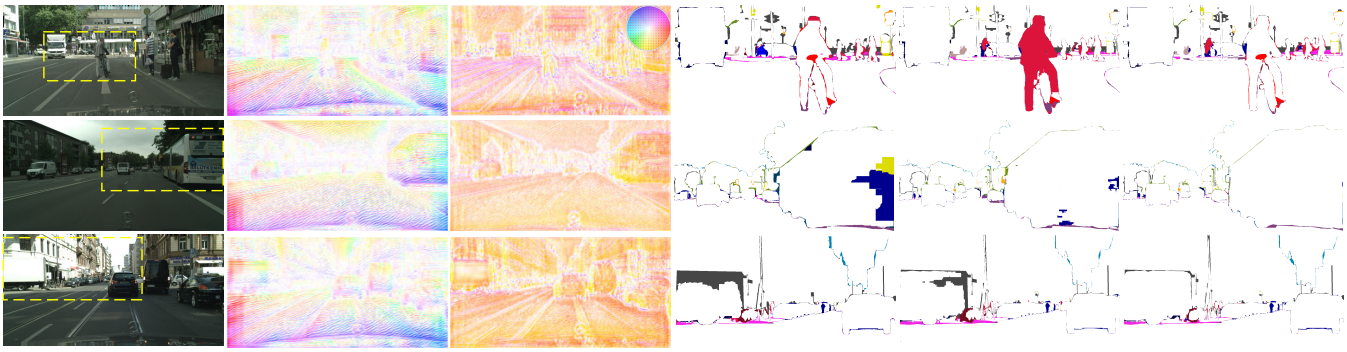


Fig. 8. Visual results of the SFC module. For offset visualization, we visualize it using the color coding in the third column. For a clear comparison, we crop representative patches (yellow dashed boxes) from the original image and visualize their prediction errors, where the correctly predicted pixels are displayed with the white background, and the incorrectly predicted pixels are colored with their ground truth color. Best viewed in color and zoom in.

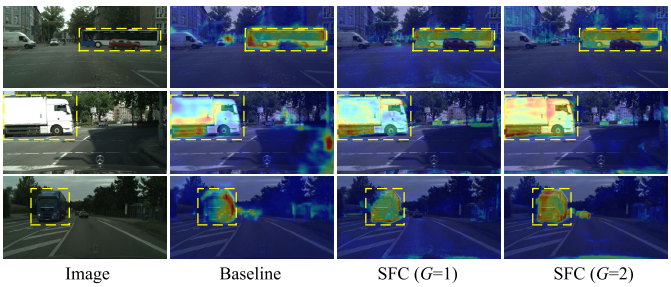


Fig. 9. Visualization results of output feature maps from different methods.

but also adopts a weighted fusion strategy to bridge the representation gap among them, thus deriving more leading performance (78.12% mIoU and 70.02 FPS).

4) Visual Analysis: We first observe the mechanism of SFC through visualization. As in Fig. 8, the offsets typically point from the boundaries to some locations inside the objects, which are usually located near the center of the object and thus have a large receptive field to grab strong semantics. That is, by using the offsets to calibrate features, the prediction of object boundary pixels can be corrected by interior pixels. Therefore, SFC can generate more precise boundaries (e.g., the "truck" in the first row). However, since the differences between sub-features are ignored, the unified calibration ($G = 1$) even brings unexpected errors. For example, the "rider" in the first row is incorrectly classified as the "person". Meanwhile, SFC ($G = 1$) also has misclassification in the region with similar textures, such as the upper part of the "truck" in the third row. In contrast, grouping calibration ($G = 2$) can produce more accurate results. The visualization results manifest the effectiveness of feature grouping calibration, which is consistent with the results in Table IV.

We further compare the feature maps produced by different methods to verify the effectiveness of our SFC, as shown in Fig. 9. We can see that although SFC ($G = 1$) calibrates the features to some extent, it does perform poorly on some details, e.g., the rear wheel of the "bus" in the first row and the top left corner of the "truck" in the third row, probably because performing uniform calibration on all feature maps weakens the information of the inconspicuous objects and stuff. In comparison, performing group calibration ($G = 2$) obtains more structural and complete feature maps, which shows the importance of feature grouping calibration.

5) In-Depth Comparison: To further analyze the effects of the two modules, we present the class-wise quantitative comparison. As in Table VI, importing CFC (w/o CRB) to the Baseline brings the mIoU improvement of 21.5% and 22.0% on "truck" and "train", respectively. Meanwhile, compared to the PPM [4] which builds homogeneous contexts for each pixel, CFC (w/o CRB) also improves performance on inconspicuous and incomplete objects, e.g., by 2.2% and 3.3% on "rider" and "motorcycle", respectively. The performance improvement illustrates that calibrating the context features of each pixel is indeed effective. Besides, recalibrating the contexts can further improve performance, especially for small objects. For example, CFC (w/ CRB) outperforms CFC (w/o CRB) on "pole", "traffic light" and "traffic sign" by 0.7%, 0.8% and 1.2% mIoU, respectively.

Meanwhile, SFC also raises the accuracy of small objects. Quantitatively, by appending SFC ($G = 1$) to the Baseline, we obtain mIoU improvement of 4.7%, 4.6%, and 4.7% on "pole", "traffic light" and "traffic sign", respectively. Meanwhile, by integrating the gating mechanism into SFC seamlessly, we solve the semantic confusion caused by simple fusion strategies to some extent. For example, compared to FAM [17] and AlignFA [22], SFC ($G = 1$) improves the accuracy of the large object "truck" by 6.9% and 1.6%, and the accuracy of the small object "pole" by 0.8% and 7.8%, respectively. Additionally, after group calibration, SFC ($G = 2$) still obtains a significant performance improvement (77.15% vs. 78.12%), which proves its necessity and effectiveness.

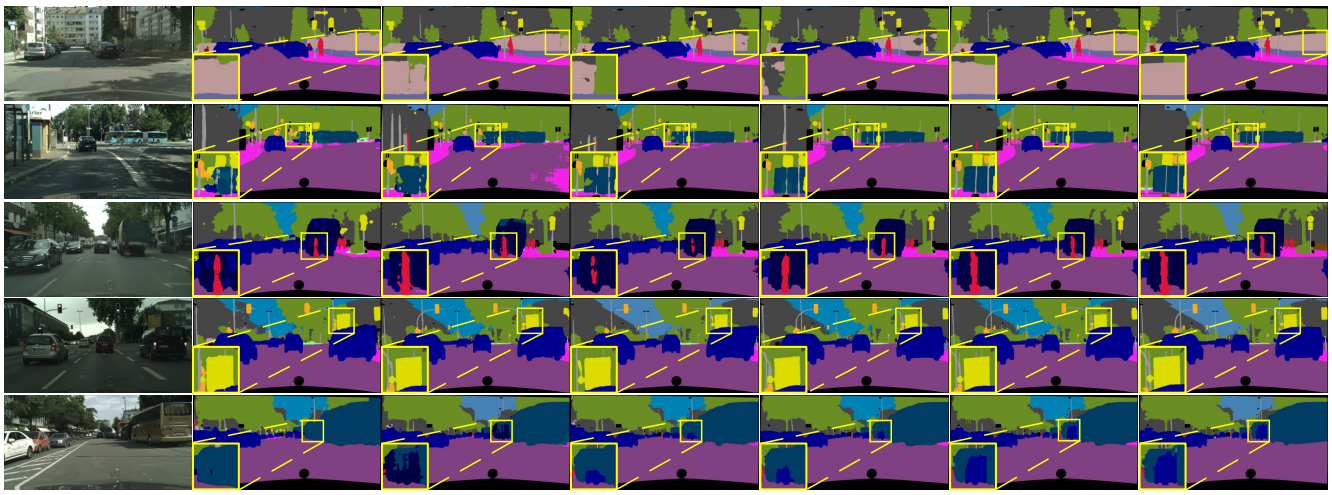
Finally, CSFCN, built with these two modules, attains 79.00% mIoU, where 12 out of the 19 classes yield the best accuracy, which verifies the robust adaptability of our method to various objects and stuff.

6) Visual Analysis: To reveal the superiority of our CSFCN intuitively, we compare the segmentation results of CSFCN and several fashionable real-time methods,i.e., ENet [27], CGNet [44], BiSeNetV2 [33] and STDC2-Seg75 [36], on the Cityscapes validation set. As shown in Fig. 10, our CSFCN generates higher-quality segmentation results. Concretely, CSFCN produces more consistent predictions inside large objects (e.g., the "fence" and "bus") with a large

TABLE VI

CLASS-WISE RESULTS ON THE CITYSCAPES VALIDATION SET. LIST OF CLASSES (FROM LEFT TO RIGHT): ROAD, SIDE-WALK, BUILDING, WALL, FENCE, POLE, TRAFFIC LIGHT, TRAFFIC SIGN, VEGETATION, TERRAIN, SKY, PERSON, RIDER, CAR, TRUCK, BUS, TRAIN, MOTORCYCLE, AND BICYCLE. **RED**/**BLUE** INDICATE SOTA/THE SECOND BEST

Method	Roa	Sid	Bui	Wal	Fen	Pol	TLi	TSi	Veg	Ter	Sky	Per	Rid	Car	Tru	Bus	Tra	Mot	Bic	mIoU
Baseline	97.8	82.9	91.6	45.9	57.0	61.6	65.9	74.2	91.8	61.1	93.9	78.7	56.7	93.9	61.5	77.7	57.9	57.3	73.7	72.69
PPM [4]	98.0	84.4	92.3	57.3	58.1	63.1	67.2	75.7	92.1	63.9	94.3	79.5	59.5	94.7	81.0	86.2	72.3	59.1	73.8	76.45
ASPP [5]	**98.3**	**85.6**	92.2	51.5	59.2	64.7	67.9	77.3	92.2	62.6	94.4	80.1	59.5	94.9	79.3	85.5	74.9	59.8	74.9	76.57
AlignCM [22]	98.1	84.7	92.4	54.7	**60.9**	64.3	67.2	76.6	92.1	62.4	94.4	80.0	59.4	94.8	81.6	**88.4**	79.5	61.4	75.0	77.26
DAPPM [41]	98.0	84.1	92.3	54.2	59.3	64.2	66.8	76.3	92.0	63.9	94.4	80.6	60.9	94.8	80.9	88.0	80.5	**63.3**	75.1	77.36
APNB [21]	98.2	84.9	92.3	53.3	60.0	64.1	67.8	77.1	92.2	62.4	94.4	80.1	59.9	94.9	**84.5**	87.3	79.2	62.4	75.1	77.37
PAM [16]	**98.3**	85.5	92.4	**57.5**	59.0	64.3	68.0	76.5	92.3	63.1	94.5	80.0	60.5	94.8	80.1	88.3	**81.9**	61.2	75.4	77.56
CFC (w/o CRB)	**98.2**	85.0	92.3	55.6	59.0	64.3	67.9	75.9	92.2	63.9	94.5	80.5	61.7	94.9	83.0	87.7	79.9	62.4	75.2	77.59
CFC (w/ CRB)	**98.2**	85.3	92.4	**59.1**	**61.0**	65.0	68.7	77.1	92.2	**64.4**	94.4	80.9	60.3	94.9	83.5	87.9	80.5	62.6	75.2	78.09
AFNB [21]	97.9	83.7	91.7	38.5	57.2	63.3	67.6	75.8	91.9	61.8	94.2	79.7	57.3	94.0	61.1	81.1	68.1	58.9	74.8	73.62
iGUM [42]	97.8	82.8	91.7	50.3	57.3	61.4	66.2	74.1	91.8	60.7	94.2	78.2	55.7	93.6	66.0	83.9	72.6	55.7	73.3	74.07
AlignFA [22]	97.5	81.7	91.9	50.2	59.2	58.5	66.9	76.0	91.8	59.8	94.5	79.6	60.3	94.3	76.1	85.0	73.0	58.9	74.6	75.25
FAM [17]	98.1	84.1	92.2	45.8	59.5	65.5	69.6	78.2	92.3	60.2	**95.1**	81.6	61.1	94.8	70.8	86.2	78.0	58.6	76.4	76.21
iAFF [43]	**98.3**	**85.6**	92.2	43.8	58.6	66.2	71.2	**80.0**	92.5	60.9	**95.1**	82.2	62.6	95.1	81.7	86.1	76.6	62.4	76.2	77.22
SFC ($G = 1$)	98.1	84.3	92.3	48.6	59.2	**66.3**	70.5	78.9	92.5	**64.2**	**95.1**	82.2	63.1	95.1	77.7	85.4	74.2	61.3	76.9	77.15
SFC ($G = 2$)	98.1	84.4	**92.8**	55.8	59.7	**66.3**	**71.5**	**80.0**	**92.6**	62.5	**95.2**	**82.4**	**64.0**	**95.3**	79.4	85.8	79.5	61.7	**77.0**	**78.12**
CSFCN	**98.3**	**86.0**	**92.6**	52.9	60.4	**67.5**	**71.3**	**80.1**	**92.8**	62.7	**95.2**	**82.7**	**63.2**	**95.6**	**85.8**	**88.9**	**85.0**	**62.7**	**77.3**	**79.00**



(a) Input (b) ENet (c) CGNet (d) BiSeNetV2 (e) STDC2-Seg75 (f) CSFCN (g) Ground Truth

Fig. 10. Visual comparison of various methods on the Cityscapes validation set. Best viewed in color and zoom in.

internal variation. By contrast, BiSeNetV2 [33] and STDC2-Seg75 [36] generally cannot handle this situation due to their limited receptive fields. Besides, CSFCN also performs better on small objects (e.g., the "person" and "pole"), while other methods (e.g., ENet [27] and CGNet [44]) have difficulty handling them because they lose considerable spatial information. Finally, our method also works well with occluded objects, such as the "car" in the fifth image. In summary, our CSFCN performs favorably against the other four methods.

7) Comparisons With State-of-the-Arts: We first present the detailed per-class results of various methods on the Cityscapes test dataset to study the superiority of CSFCN. As in Table VII, our CSFCN has reached the best mIoU on most classes (e.g., "road", "truck" and "bicycle"), which significantly outperforms existing real-time methods (e.g., ENet [27], CGNet [44] and PCNet [11]). Nonetheless, CSFCN is still difficult to handle spatial-detail related classes (e.g., "traffic light" and " traffic sign") compared with accuracy-oriented methods (e.g., DUC [23]), which generally adopt a large dilated backbone (e.g., ResNet-101 [37]) to extract high-resolution features. However, our method can also achieve comparable performance and outperforms it in overall accuracy (78.7% vs. 77.6%). Moreover, CSFCN that adopts MobileNetV3-L as the Backbone also achieves better accuracy (75.1%) with less computation (21.7G FLOPs).

We further compare the CSFCN with SOTA real-time networks on the Cityscapes test set. In particular, we append 50, 75 and 100 after the network name to represent the input sizes of 512 × 1024, 768 × 1536 and 1024 × 2048, respectively. As in Table VIII, our CSFCN achieves the best speed-accuracy trade-off and outperforms other competitors by a relatively large margin. Concretely, with faster speed, CSFCN-100 achieves the best accuracy of 78.7% mIoU, outperforming SFANet [12], MGSeg [1], HyperSeg [48], and BiSeNetV2-L† [33] by 0.6%, 0.9%, 2.9% and 3.4%, respectively. Meanwhile, our CSFCN-50 outperforms

TABLE VII

CLASS-WISE EVALUATION RESULTS ON THE CITYSCAPES TEST SET. LIST OF CLASSES (FROM LEFT TO RIGHT): ROAD, SIDE-WALK, BUILDING, WALL, FENCE, POLE, TRAFFIC LIGHT, TRAFFIC SIGN, VEGETATION, TERRAIN, SKY, PERSON, RIDER, CAR, TRUCK, BUS, TRAIN, MOTORCYCLE, AND BICYCLE. METHODS WITH THE ‡ SYMBOL ARE REPRESENTED AS THE ACCURACY-ORIENTED METHODS

Method	Roa	Sid	Bui	Wal	Fen	Pol	TLi	TSi	Veg	Ter	Sky	Per	Rid	Car	Tru	Bus	Tra	Mot	Bic	**mIoU**
ENet [27]	96.3	74.2	85.0	32.2	33.2	43.5	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
CGNet [44]	95.5	78.7	88.1	40.0	43.0	54.1	59.8	63.9	89.6	67.6	92.9	74.9	54.9	90.2	44.1	59.5	25.2	47.3	60.2	64.8
ERFNet [28]	97.9	82.1	90.7	45.2	50.4	59.0	62.6	68.4	91.9	69.4	94.2	78.5	59.8	93.4	52.3	60.8	53.7	49.9	64.2	69.7
ESPNetV2 [29]	97.3	78.6	88.8	43.5	42.1	49.3	52.6	60.0	90.5	66.8	93.3	72.9	53.1	91.8	53.0	65.9	53.2	44.2	59.9	66.2
FBSNet [31]	98.0	83.2	91.5	50.9	53.5	62.5	67.6	71.5	92.7	70.5	94.4	82.5	63.8	93.9	50.5	56.0	37.6	56.2	70.1	70.9
PCNet [11]	98.3	84.4	91.4	48.4	52.6	57.1	63.8	69.7	92.3	70.0	94.6	80.6	61.5	94.5	61.2	73.9	63.2	57.3	69.3	72.9
Reference [14]	98.2	84.0	91.6	50.7	49.5	60.9	69.0	73.6	92.6	70.3	94.4	83.0	65.7	94.9	62.0	70.9	53.3	62.5	71.8	73.6
DMA-Net [35]	**98.5**	85.5	**92.2**	53.3	55.3	62.5	70.9	74.9	93.0	71.2	95.0	84.0	66.6	**95.6**	68.2	82.8	76.6	64.5	73.2	77.0
RefineNet‡ [8]	98.2	83.3	91.3	47.8	50.4	56.1	66.9	71.3	92.3	70.3	94.8	80.9	63.3	94.5	64.6	76.1	64.3	62.2	70.0	73.6
DUC‡ [23]	**98.5**	85.5	**92.8**	**58.6**	55.5	65.0	**73.5**	**77.9**	**93.3**	**72.0**	95.2	84.8	68.5	95.4	**70.9**	78.8	68.7	**65.9**	73.8	77.6
CSFCN (M)	98.4	84.7	92.2	50.3	54.8	61.2	67.8	73.5	92.8	70.3	**95.3**	82.5	64.4	95.1	56.3	76.8	77.7	60.9	71.5	75.1
CSFCN-50	98.2	83.3	90.9	48.8	50.1	56.9	61.8	67.3	91.8	68.8	94.5	79.0	62.3	94.1	67.2	83.6	78.5	57.8	66.5	73.8
CSFCN-75	**98.6**	**86.0**	**92.8**	51.9	**59.4**	**66.3**	72.8	76.9	**93.5**	71.3	95.2	**85.2**	**69.2**	**95.8**	**71.6**	**84.3**	**82.3**	**65.5**	**74.2**	77.2
CSFCN-100	**98.6**	**86.2**	**92.8**	**53.6**	**59.0**	**66.4**	**73.1**	**77.1**	**93.5**	**71.9**	**95.4**	**85.3**	**69.0**	**95.8**	70.0	**83.9**	**84.5**	64.8	**74.6**	**78.7**

TABLE VIII

ACCURACY AND SPEED COMPARISONS WITH SOTA WORKS ON THE CITYSCAPES TEST DATASET. METHOD DENOTED WITH † USES TENSORRT ACCELERATION STRATEGY TO MEASURE THE INFERENCE SPEED, WHICH WILL SUBSTANTIALLY IMPROVE THE EFFICIENCY

Method	Backbone	Input Size	FLOPs (G)	GPU	FPS	mIoU(%)
NDNet [30]	Custom	1024 × 2048	8.4	TitanX	52.6	65.7
ESPNetV2 [29]	ESPNetV2	512 × 1024	**2.7**	GTX 1080Ti	61.9	66.2
ERFNet [28]	Custom	512 × 1024	27.7	TitanX M	41.7	69.7
FRNet [10]	Custom	512 × 1024	-	TitanXp	127.0	70.4
FBSNet [31]	Custom	512 × 1024	9.7	RTX 2080Ti	90.0	70.9
DFANet A [34]	Xception A	1024 × 1024	3.4	TitanX	100.0	71.3
DFANet B [34]	Xception B	1024 × 1024	**2.1**	TitanX	120.0	67.1
BiSeNet1 [32]	Xception-39	768 × 1536	14.8	GTX 1080Ti	105.8	68.4
BiSeNet2 [32]	ResNet-18	768 × 1536	55.3	GTX 1080Ti	65.5	74.7
BiSeNetV2† [33]	Custom	512 × 1024	21.1	GTX 1080Ti	156.0	72.6
BiSeNetV2-L† [33]	Custom	512 × 1024	118.5	GTX 1080Ti	47.3	75.3
PCNet [11]	Custom	1024 × 2048	11.5	GTX 2080Ti	79.1	72.9
Reference [14]	MobileNetV2	448 × 896	49.5	TitanX	51.0	73.6
DMA-Net [35]	ResNet-18	1024 × 2048	94.2	GTX 1080Ti	46.7	77.0
SwiftNetRN-18 [9]	ResNet-18	1024 × 2048	104.0	GTX 1080Ti	41.0	75.5
RGPNet† [45]	ResNet-18	1024 × 2048	-	RTX 2080Ti	153.4	74.1
SegFormer [46]	MiT-B0	1024 × 2048	125.5	Tesla V100	15.2	76.2
HoloSeg [47]	Custom	512 × 1024	17.3	RTX 2080Ti	118.0	76.2
STDC1-Seg50† [36]	STDC1	512 × 1024	-	GTX 1080Ti	**250.4**	71.9
STDC2-Seg75† [36]	STDC2	768 × 1536	-	GTX 1080Ti	97.0	76.8
HyperSeg [48]	EfficientNet-B1	512 × 1024	7.5	GTX 1080Ti	36.9	75.8
MSFNet [49]	ResNet-18	1024 × 2048	96.8	RTX 2080Ti	41.0	77.1
DDRNet-23-Slim [41]	DDRNet-23-Slim	1024 × 2048	36.3	RTX 2080Ti	108.8	77.4
PP-LiteSeg-B2† [13]	STDC2	768 × 1536	-	GTX 1080Ti	102.6	**77.5**
SFNet [17]	DF2	1024 × 2048	-	GTX 1080Ti	53.0	77.8
MGSeg [1]	ResNet-18	768 × 1536	54.3	GTX 1080Ti	84.0	76.4
MGSeg [1]	ResNet-18	1024 × 2048	96.5	GTX 1080Ti	50.0	77.8
SFANet [12]	ResNet-18	1024 × 2048	99.6	GTX 1080Ti	37.0	**78.1**
CSFCN-100	MobileNetV3-L	1024 × 2048	21.7	GTX 1080Ti	63.4	75.1
CSFCN-50	ResNet-18	512 × 1024	24.9	GTX 1080Ti	**229.1**	73.8
CSFCN-75	ResNet-18	768 × 1536	56.2	GTX 1080Ti	122.2	77.2
CSFCN-100	ResNet-18	1024 × 2048	99.8	GTX 1080Ti	70.0	**78.7**

STDC1-Seg50† [36] by 1.9% mIoU with a similar inference speed. In particular, our CSFCN-75 also outperforms BiSeNet2 [32] by a large margin (2.5%) with the same backbone (ResNet-18) and similar FLOPs (55.3G vs. 56.2G). Particularly, although depth-wise convolutions can significantly reduce computation (means fewer FLOPs), its actual inference speed is often lower than standard convolution due to memory access cost or other reasons [50]. Hence, unlike methods such as FBSNet [31] or SFANet [12], we only use standard convolution for faster speed (70.0 FPS).

8) Efficiency Evaluation: Efficiency is crucial for real-time semantic segmentation. Previous studies reported efficiency on various hardware setups. To ensure fairness, we re-test previous methods with their open-source codes on our environment.

TABLE IX
CLASS-WISE EVALUATION RESULTS ON THE CAMVID TEST SET. LIST OF CLASSES (FROM LEFT TO RIGHT): BUILDING, TREE, SKY, CAR, SIGN, ROAD, PEDESTRIAN, FENCE, POLE, SIDEWALK AND BICYCLIST

Method	Pretrain	Bui	Tre	Sky	Car	Sig	Roa	Ped	Fen	Pol	Sid	Bic	**mIoU**
ENet [27]	Scratch	74.7	77.8	**95.1**	82.4	51.0	95.1	67.2	51.7	35.4	86.7	34.1	51.3
BiSeNet1 [32]	ImageNet	82.2	74.4	91.9	80.8	42.8	93.3	53.8	49.7	25.4	77.3	50.0	65.6
BiSeNet2 [32]	ImageNet	83.0	75.8	92.0	83.7	46.5	94.6	58.8	53.6	31.9	81.4	54.0	68.7
PCNet [11]	Scratch	82.3	74.5	91.4	80.5	44.8	95.1	56.8	40.2	34.0	81.7	55.3	67.0
Reference [14]	ImageNet	83.2	70.5	92.5	81.7	**51.6**	93.0	55.6	53.2	36.3	82.1	47.9	68.0
BiSeNetV2/BiSeNetV2-L [33]	ImageNet	-	-	-	-	-	-	-	-	-	–	-	72.4/73.2
CSFCN	ImageNet	**90.6**	81.1	92.9	**94.5**	50.6	96.9	**74.0**	**68.6**	**45.3**	**90.7**	**70.6**	77.8
BiSeNetV2/BiSeNetV2-L [33]	ImageNet	-	-	-	-	-	-	-	-	-	–	-	76.7/78.5
PIDNet-S [15]	Cityscapes	-	-	-	-	-	-	-	-	-	–	-	**80.1**
CSFCN	Cityscapes	**91.7**	**82.1**	**93.0**	**95.4**	**55.6**	**97.5**	**76.4**	**75.4**	**52.0**	**92.2**	**79.2**	**81.0**

TABLE X
EFFICIENCY AND ACCURACY COMPARISONS ON THE CITYSCAPES VAL SET. THE FLOPS AND MAC ARE ESTIMATED BY USING THE INPUT OF $3 \times 1024 \times 2048$

Method	FLOPs (G)	MAC (GB)	FPS	mIoU (%)
BiSeNet [32]	118.8	4.07	51.1	75.37
SwiftNet [9]	103.7	3.67	56.3	75.40
MGSeg† [1]	96.6	2.82	59.7	77.80
SFANet† [12]	99.4	4.85	45.4	78.40
CSFCN (M)	**21.7**	4.98	63.4	75.31
CSFCN	99.8	**2.72**	**70.0**	**79.00**

If no open-source code is available, we have reproduced them to report results, denoted by † in Table X. Regarding accuracy (mIoU), we directly copy the value from the original paper. In particular, inspired by [1] and [51], we additionally use Memory Access Cost (MAC) to evaluate the efficiency of different methods. MAC indicates the number of memory access operations for features and model weights:

$$\text{MAC} = (h_{in} \cdot w_{in} \cdot c_{in} + h_{out} \cdot w_{out} \cdot c_{out}) + c_{in} \cdot c_{out} \cdot k \cdot k \tag{9}$$

where the two terms denote the memory access for input/output features and kernel weights, respectively. For light-weight networks, MAC also has a significant impact on the final inference speed [1], [51].

As in Table X, our CSFCN outperforms previous real-time state-of-the-art methods. Compared with MGSeg [1], CSFCN achieves higher accuracy while speeding up by around 17%. Meanwhile, with similar FLOPs, CSFCN is 24% faster than SFANet [12]. This is because the more complex structure of SFANet leads to its higher MAC, which slows down network inference. Besides, compared to SFANet, our CSFCN (M) with MobileNetV3-L as the backbone has a 39.6% increase in speed (63.4 FPS vs. 45.4 FPS), while FLOPs (21.7 G vs. 99.4 G) have significantly decreased. We speculate that the difference in metrics is due to the greater impact of MAC on speed.

We further evaluate the effect of input sizes on performance by testing four resolutions (ranging from 384×768 to full resolution of 1024×2048) on the Cityscapes validation set. Two open-source methods (i.e., SwiftNet [9] and BiSeNet [32]) that share the same backbone as ours while achieving promising performance are adopted for comparison. As shown in Fig. 11,

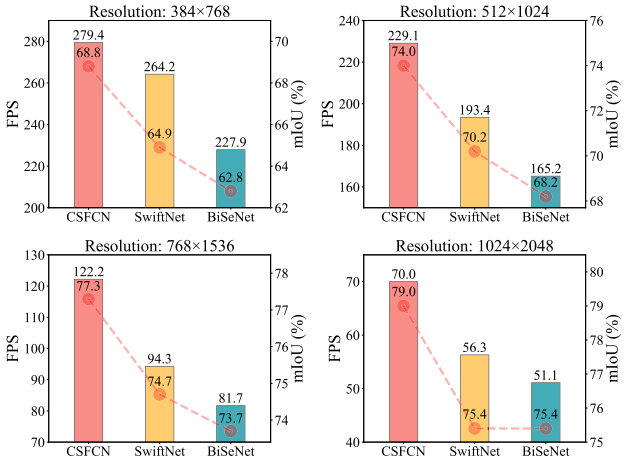


Fig. 11. Comparisons of the speed (FPS) and accuracy (mIoU) obtained by different methods at different resolution inputs. We use bars and points to denote the speed and accuracy of different methods, respectively.

our method achieves the best accuracy and speed at different resolutions. In particular, our CSFCN achieves higher accuracy (77.3%) than the best accuracy of its counterparts (75.4%) with a smaller resolution of 768×1536, which denotes a downscale of more than one times the full resolution. Besides, with a minimum resolution of 384×768, our CSFCN achieves better results (68.8%) than the minimum acceptable mIoU (defined as 65.0% [52]) at a faster speed. We attribute the speed advantage of our method to its less FLOPs and MAC, and higher parallelism (i.e., fewer branches). In summary, the above experimental results further prove that our method achieves a SOTA speed-accuracy trade-off.

C. Results on CamVid Dataset

We also conduct experiments on the CamVid dataset to demonstrate the generalization of CSFCN. First, we present the per-class results of various methods to discuss the effectiveness of our method thoroughly. As presented in Table IX, our CSFCN can obtain the best accuracy on most classes. For example, compared with ENet [27] and BiSeNet2 [32], our CSFCN (ImageNet) improves the accuracy of the large object "car" by 12.1% and 10.8% and the accuracy of the small object "pole" by 9.9% and 13.4%, respectively. In particular, our CSFCN pre-trained on Cityscapes achieves leading accuracy while maintaining

TABLE XI
ACCURACY AND SPEED COMPARISONS WITH SOTA WORKS ON THE CAMVID TEST DATASET. METHOD DENOTED WITH † USES TENSORRT ACCELERATION STRATEGY TO MEASURE THE SPEED

Method	Input Size	FPS	mIoU(%)
DFANet [34]	720 × 960	160.0	59.3
CGNet [44]	720 × 960	-	65.6
RGPNet† [45]	352 × 480	190.0	66.9
PCNet [11]	720 × 960	62.1	67.0
BiSeNet [32]	720 × 960	116.3	68.7
Reference [14]	720 × 960	39.3	68.0
BiSeNetV2† [33]	720 × 960	124.5	72.4
BiSeNetV2-L† [33]	720 × 960	32.7	73.2
MGSeg [1]	736 × 960	127.0	72.7
DMA-Net [35]	720 × 960	119.8	73.6
SwiftNetRN-18 [9]	720 × 960	-	72.6
STDC1-Seg† [36]	720 × 960	**197.6**	73.0
STDC2-Seg† [36]	720 × 960	152.2	73.9
HoloSeg [47]	720 × 960	105.3	74.1
DDRNet-23-Slim [41]	720 × 960	**230.0**	74.4
SFANet [12]	720 × 960	96.0	74.7
PP-LiteSeg-B† [13]	720 × 960	154.8	75.0
MSFNet [49]	768 × 1024	91.0	**75.4**
CSFCN	720 × 960	179.2	**77.8**

real-time performance. Concretely, Cityscapes pre-trained CSFCN obtains 81.0% mIoU at 179.2 FPS, stronger and faster than MGSeg [1] and HoloSeg [47]. Second, we compare CSFCN with SOTA works on the CamVid test dataset. As reported in Table XI, our CSFCN yields 77.8% mIoU with 179.2 FPS for a 720×960 input, which realizes the encouraging speed/accuracy compromise.

V. CONCLUSION

In this paper, we focus on the issues of context mismatch and feature misalignment, and propose a novel segmentation network, CSFCN, to resolve such issues while maintaining high efficiency. Specifically, CSFCN has two core components: the Context Feature Calibration (CFC) module and the Spatial Feature Calibration (SFC) module. CFC uses an efficient pooling module to capture multi-scale contexts, then matches pixels and contexts to tackle the issue of pixel-context mismatch. SFC groups channel dimensions into multiple sub-features and then calibrates them separately to ease the problem of spatial feature misalignment. With these two modules, our CSFCN achieves remarkable performance improvements without entailing excessive computation overhead. Comprehensive experiments have illustrated that our CSFCN achieves SOTA speed/accuracy trade-off on two challenging datasets, Cityscapes and CamVid.

REFERENCES

[1] J.-Y. He, S.-H. Liang, X. Wu, B. Zhao, and L. Zhang, "MGSeg: Multiple granularity-based real-time semantic segmentation network," *IEEE Trans. Image Process.*, vol. 30, pp. 7200–7214, 2021.

[2] B. Xie et al., "Multi-scale fusion with matching attention model: A novel decoding network cooperated with NAS for real-time semantic segmentation," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 8, pp. 12622–12632, Aug. 2022.

[3] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 3431–3440.

[4] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 6230–6239.

[5] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder–decoder with atrous separable convolution for semantic image segmentation," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Sep. 2018, pp. 801–818.

[6] M. Yang, K. Yu, C. Zhang, Z. Li, and K. Yang, "DenseASPP for semantic segmentation in street scenes," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 3684–3692.

[7] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. Int. Conf. Med. Image Comput. Comput.-Assist. Intervent*. Cham, Switzerland: Springer, 2015, pp. 234–241.

[8] G. Lin, A. Milan, C. Shen, and I. Reid, "RefineNet: Multi-path refinement networks for high-resolution semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 5168–5177.

[9] M. Oršić and S. Šegvić, "Efficient semantic segmentation with pyramidal fusion," *Pattern Recognit.*, vol. 110, Feb. 2021, Art. no. 107611.

[10] M. Lu, Z. Chen, Q. M. J. Wu, N. Wang, X. Rong, and X. Yan, "FRNet: Factorized and regular blocks network for semantic segmentation in road scene," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 4, pp. 3522–3530, Apr. 2022.

[11] Q. Lv, X. Sun, C. Chen, J. Dong, and H. Zhou, "Parallel complement network for real-time semantic segmentation of road scenes," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 5, pp. 4432–4444, May 2022.

[12] X. Weng, Y. Yan, S. Chen, J.-H. Xue, and H. Wang, "Stage-aware feature alignment network for real-time semantic segmentation of street scenes," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 32, no. 7, pp. 4444–4459, Jul. 2022.

[13] J. Peng et al., "PP-LiteSeg: A superior real-time semantic segmentation model," 2022, *arXiv:2204.02681*.

[14] G. Dong, Y. Yan, C. Shen, and H. Wang, "Real-time high-performance semantic image segmentation of urban street scenes," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 6, pp. 3258–3274, Jun. 2021.

[15] J. Xu, Z. Xiong, and S. P. Bhattacharyya, "PIDNet: A real-time semantic segmentation network inspired by PID controllers," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2023, pp. 19529–19539.

[16] J. Fu et al., "Dual attention network for scene segmentation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 3141–3149.

[17] X. Li et al., "Semantic flow for fast and accurate scene parsing," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2020, pp. 775–793.

[18] X. Wang, R. Girshick, A. Gupta, and K. He, "Non-local neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 7794–7803.

[19] M. Cordts et al., "The cityscapes dataset for semantic urban scene understanding," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 3213–3223.

[20] G. J. Brostow, J. Shotton, J. Fauqueur, and R. Cipolla, "Segmentation and recognition using structure from motion point clouds," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2008, pp. 44–57.

[21] Z. Zhu, M. Xu, S. Bai, T. Huang, and X. Bai, "Asymmetric non-local neural networks for semantic segmentation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 593–602.

[22] Z. Huang, Y. Wei, X. Wang, W. Liu, T. S. Huang, and H. Shi, "AlignSeg: Feature-aligned segmentation networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 1, pp. 550–557, Jan. 2022.

[23] P. Wang et al., "Understanding convolution for semantic segmentation," in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Mar. 2018, pp. 1451–1460.

[24] Y. Yuan, L. Huang, J. Guo, C. Zhang, X. Chen, and J. Wang, "OCNet: Object context for semantic segmentation," *Int. J. Comput. Vis.*, vol. 129, no. 8, pp. 2375–2398, Aug. 2021.

[25] W. Wang et al., "Pyramid vision transformer: A versatile backbone for dense prediction without convolutions," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 548–558.

[26] Y.-H. Wu, Y. Liu, X. Zhan, and M.-M. Cheng, "P2T: Pyramid pooling transformer for scene understanding," *IEEE Trans. Pattern Anal. Mach. Intell.*, early access, Aug. 30, 2022, doi: 10.1109/TPAMI.2022.3202765.

[27] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "ENet: A deep neural network architecture for real-time semantic segmentation," 2016, *arXiv:1606.02147*.

[28] E. Romera, J. M. Álvarez, L. M. Bergasa, and R. Arroyo, "ERFNet: Efficient residual factorized ConvNet for real-time semantic segmentation," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 1, pp. 263–272, Jan. 2018.

[29] S. Mehta, M. Rastegari, L. Shapiro, and H. Hajishirzi, "ESPNetv2: A light-weight, power efficient, and general purpose convolutional neural network," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 9182–9192.

[30] Z. Yang et al., "Small object augmentation of urban scenes for real-time semantic segmentation," *IEEE Trans. Image Process.*, vol. 29, pp. 5175–5190, 2020.

[31] G. Gao, G. Xu, J. Li, Y. Yu, H. Lu, and J. Yang, "FBSNet: A fast bilateral symmetrical network for real-time semantic segmentation," *IEEE Trans. Multimedia*, vol. 25, pp. 3273–3283, 2022.

[32] C. Yu, J. Wang, and C. Peng, "BiseNet: Bilateral segmentation network for real-time semantic segmentation," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Sep. 2018, pp. 325–341.

[33] C. Yu, C. Gao, J. Wang, G. Yu, C. Shen, and N. Sang, "BiSeNet v2: Bilateral network with guided aggregation for real-time semantic segmentation," *Int. J. Comput. Vis.*, vol. 129, no. 11, pp. 3051–3068, Nov. 2021.

[34] H. Li, P. Xiong, H. Fan, and J. Sun, "DFANet: Deep feature aggregation for real-time semantic segmentation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 9514–9523.

[35] X. Weng et al., "Deep multi-branch aggregation network for real-time semantic segmentation in street scenes," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 10, pp. 17224–17240, Oct. 2022.

[36] M. Fan et al., "Rethinking BiSeNet for real-time semantic segmentation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 9711–9720.

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[38] O. Russakovsky et al., "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[39] X. Li, X. Hu, and J. Yang, "Spatial group-wise enhance: Improving semantic feature learning in convolutional networks," 2019, *arXiv:1905.09646*.

[40] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," 2017, *arXiv:1706.05587*.

[41] H. Pan, Y. Hong, W. Sun, and Y. Jia, "Deep dual-resolution networks for real-time and accurate semantic segmentation of traffic scenes," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 3, pp. 3448–3460, Mar. 2023.

[42] D. Mazzini and R. Schettini, "Spatial sampling network for fast scene understanding," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2019, pp. 1286–1296.

[43] Y. Dai, F. Gieseke, S. Oehmcke, Y. Wu, and K. Barnard, "Attentional feature fusion," in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Jan. 2021, pp. 3559–3568.

[44] T. Wu, S. Tang, R. Zhang, J. Cao, and Y. Zhang, "CGNet: A light-weight context guided network for semantic segmentation," *IEEE Trans. Image Process.*, vol. 30, pp. 1169–1179, 2021.

[45] E. Arani, S. Marzban, A. Pata, and B. Zonooz, "RGPNet: A real-time general purpose semantic segmentation," in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Jan. 2021, pp. 3008–3017.

[46] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo, "SegFormer: Simple and efficient design for semantic segmentation with transformers," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 12077–12090.

[47] S. Li, Q. Yan, C. Liu, M. Liu, and Q. Chen, "HoloSeg: An efficient holographic segmentation network for real-time scene parsing," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2022, pp. 2395–2402.

[48] Y. Nirkin, L. Wolf, and T. Hassner, "HyperSeg: Patch-wise hypernetwork for real-time semantic segmentation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 4060–4069.

[49] H. Si, Z. Zhang, and F. Lu, "Real-time semantic segmentation via multiple spatial fusion network," in *Proc. Brit. Mach. Vis. Conf.*, 2020, pp. 1–12.

[50] X. Wang, S. Lai, Z. Chai, X. Zhang, and X. Qian, "SPGNet: Serial and parallel group network," *IEEE Trans. Multimedia*, vol. 24, pp. 2804–2814, 2022.

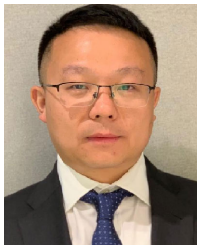
[51] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient CNN architecture design," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, pp. 116–131, 2018.

[52] W. Xiang, H. Mao, and V. Athitsos, "ThunderNet: A turbo unified network for real-time semantic segmentation," in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Jan. 2019, pp. 1789–1796.

Kaige Li received M.S. degree from the Ocean University of China, Qingdao, China, in 2020. He is currently pursuing the Ph.D. degree with the State Key Laboratory of Virtual Reality Technology and Systems, Computer Science and Technology, Beihang University, Beijing, China. His current research interests include deep learning, image semantic segmentation, computer vision, and smart city.

Qichuan Geng received the B.S. degree in automation science and the Ph.D. degree in the technology of computer application from Beihang University, Beijing, China, in 2012 and 2021, respectively. He is currently a Lecturer and the Master's Instructor with the Information Engineering College, Capital Normal University. His main research interests include computer vision, semantic segmentation, and scene geometry recovery.

Maoxian Wan received the B.S. degree from Beijing University of Posts and Telecommunications in 2022. He is currently pursuing the M.S. degree with the State Key Laboratory of Virtual Reality Technology and Systems, Beihang University, China. His research interests include computer vision and semantic segmentation.



Xiaochun Cao (Senior Member, IEEE) received the B.E. and M.E. degrees in computer science from Beihang University (BUAA), China, and the Ph.D. degree in computer science from the University of Central Florida, USA. After graduation, he spent about three years with ObjectVideo Inc., as a Research Scientist. From 2008 to 2012, he was a Professor with Tianjin University. Before joining SYSU, he was a Professor with the Institute of Information Engineering, Chinese Academy of Sciences. He is currently a Professor with the School of Cyber Science and Technology, Sun Yat-sen University. He has authored and coauthored over 200 journals and conference papers. In 2004 and 2010, he was a recipient of the Piero Zamperoni Best Student Paper Award at the International Conference on Pattern Recognition. He is on the editorial boards of IEEE TRANSACTIONS ON IMAGE PROCESSING and IEEE TRANSACTIONS ON MULTIMEDIA. He was on the editorial board of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY. His Ph.D. dissertation nominated for the university-level Outstanding Dissertation Award.

Zhong Zhou received the B.S. degree in material physics from Nanjing University in 1999 and the Ph.D. degree in computer science and engineering from Beihang University, Beijing, China, in 2005. He is currently a Professor and a Ph.D. Advisor with the State Key Laboratory of Virtual Reality Technology and Systems, Beihang University. His main research interests include virtual reality, augmented reality, computer vision, and artificial intelligence.