FastFCN: Rethinking Dilated Convolution in the Backbone for Semantic Segmentation

Huikai Wu, Junge Zhang, Kaiqi Huang
Institute of Automation, Chinese Academy of Sciences
{huikai.wu, jgzhang, kaiqi.huang}@nlpr.ia.ac.cn

Kongming Liang, Yizhou Yu
Deepwise AI Lab
liangkongming@deepwise.com, yizhouy@acm.org

Abstract

Modern approaches for semantic segmentation usually employ dilated convolutions in the backbone to extract high-resolution feature maps, which brings heavy computation complexity and memory footprint. To replace the time and memory consuming dilated convolutions, we propose a novel joint upsampling module named Joint Pyramid Upsampling (JPU) by formulating the task of extracting high-resolution feature maps into a joint upsampling problem. With the proposed JPU, our method reduces the computation complexity by more than three times without performance loss. Experiments show that JPU is superior to other upsampling modules, which can be plugged into many existing approaches to reduce computation complexity and improve performance. By replacing dilated convolutions with the proposed JPU module, our method achieves the state-of-the-art performance in Pascal Context dataset (mIoU of 53.13%) and ADE20K dataset (final score of 0.5584) while running 3 times faster. Code is available in https://github.com/wuhuikai/FastFCN.

1. Introduction

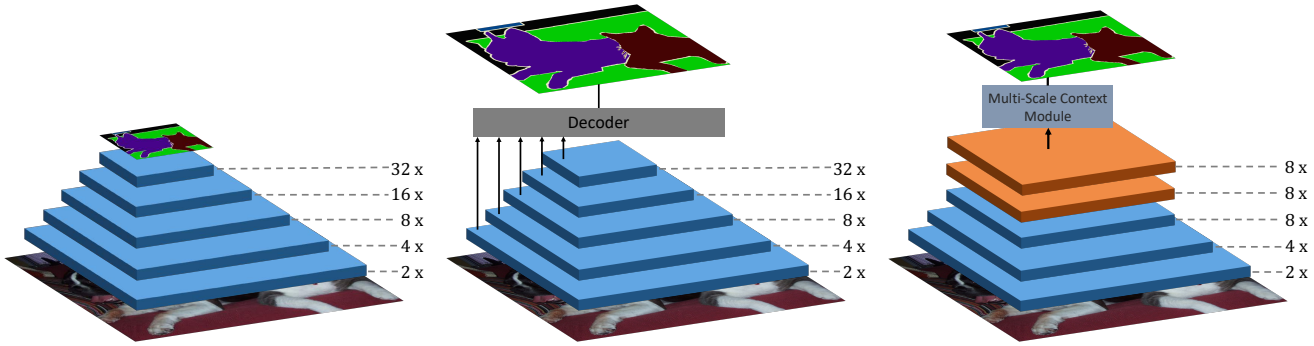
Semantic segmentation [23, 40, 4] is one of the fundamental tasks in computer vision, with the goal of assigning a semantic label to each pixel of an image. Modern approaches usually employ a Fully Convolution Network (FCN) [22] to address this task, achieving tremendous success among several segmentation benchmarks.

The original FCN is proposed by Long *et al*. [22], which is transformed from a Convolutional Neural Network (CNN) [16, 15] designed for image classification. Inheriting from the design for image classification, the original FCN downsamples the input image progressively by stride convolutions and/or spatial pooling layers, resulting in a final feature map in low resolution. Although the final feature map encodes rich semantic information, the fine image structure information is lost, leading to inaccurate predictions around the object boundaries. As shown in Figure 1a, the original FCN typically downsamples the input image 5 times, reducing the spatial resolution of the final feature map by a factor of 32.

To obtain a high-resolution final feature map, [3, 28, 18, 30, 27] employ the original FCN as the encoder to capture high-level semantic information, and a decoder is designed to gradually recover the spatial information by combining multi-level feature maps from the encoder. As shown in Figure 1b, we term such methods EncoderDecoder, of which the final prediction generated by the decoder is in high resolution. Alternatively, DeepLab [5] removes the last two downsampling operations from the original FCN and introduces dilated (atrous) convolutions to maintain the receptive field of view unchanged.[1] Following DeepLab, [38, 6, 36] employ a multi-scale context module on top of the final feature map, outperforming most EncoderDecoder methods significantly on several segmentation benchmarks. As shown in Figure 1c, the spatial resolution of the last feature map in DilatedFCN is 4 times larger than that in the original FCN, thus maintaining more structure and location information.

The dilated convolutions play an important role in maintaining the spatial resolution of the final feature map, leading to superior performance compared to most methods in EncoderDecoder. However, the introduced dilated convolutions bring heavy computation complexity and memory footprint, which limit the usage in many real-time applications. Taking ResNet-101 [13] as an example, compared to

[1] In most cases, dilated convolutions in this paper refer to (1) removing downsampling operations and (2) replacing regular convolutions with dilated convolutions.



(a) FCN [22] (b) EncoderDecoder (c) DilatedFCN

Figure 1: **Different types of networks for semantic segmentation.** (a) is the original FCN, (b) follows the encoder-decoder style, and (c) employs dilated convolutions to obtain high-resolution final feature maps. Best viewed in color.

the original FCN, 23 residual blocks (69 convolution layers) in DilatedFCN require to take 4 times more computation resources and memory usages, and 3 residual blocks (9 convolution layers) need to take 16 times more resources.

We aim at tackling the aforementioned issue caused by dilated convolutions in this paper. To achieve this, we propose a novel joint upsampling module to replace the time and memory consuming dilated convolutions, namely Joint Pyramid Upsampling (JPU). As a result, our method employs the original FCN as the backbone while applying JPU to upsample the low-resolution final feature map with output stride (OS) 32, resulting in a high-resolution feature map (OS=8). Accordingly, the computation time and memory footprint of the whole segmentation framework is dramatically reduced. Meanwhile, there's no performance loss when replacing the dilated convolutions with the proposed JPU. We attribute this to the ability of JPU to exploit multi-scale context across multi-level feature maps.

To validate the effectiveness of our method, we first conduct a systematical experiment, showing that the proposed JPU can replace dilated convolutions in several popular approaches without performance loss. We then test the proposed method on several segmentation benchmarks. Results show that our method achieves the state-of-the-art performance while running more than 3 times faster. Concretely, we outperform all the baselines on Pascal Context dataset [23] by a large margin, which achieves the state-of-the-art performance with mIoU of 53.13%. On ADE20K dataset [40], we obtain the mIoU of 42.75% with ResNet-50 as the backbone, which sets a new record on the *val* set. Moreover, our method with ResNet-101 achieves the state-of-the-art performance in the *test* set of ADE20K dataset.

In summary, our contributions are three folds, which are: (1) We propose a computationally efficient joint upsampling module named JPU to replace the time and memory consuming dilated convolutions in the backbone. (2) Based on the proposed JPU, the computation time and memory footprint of the whole segmentation framework can be reduced by a factor of more than 3 and meanwhile achieves better performance. (3) Our method achieves the new state-of-the-art performance in both Pascal Context dataset (mIoU of 53.13%) and ADE20K dataset (mIoU of 42.75% with ResNet-50 as the backbone on the *val* set and final score of 0.5584 with ResNet-101 on the *test* set).

2. Related Work

In this section, we first give an overview on methods for semantic segmentation, which can be categorized into two directions. We then introduce some related works on upsampling.

2.1. Semantic Segmentation

FCNs [22] have achieved huge success in semantic segmentation. Following FCN, there're two prominent directions, namely DilatedFCN and EncoderDecoder. DilatedFCNs [11, 34, 7, 6, 38, 36, 5] utilize dilated convolutions to keep the receptive field of view and employ a multi-scale context module to process high-level feature maps. Alternatively, EncoderDecoders [24, 28, 18, 1, 26, 12, 33, 37] propose to utilize an encoder to extract multi-level feature maps, which are then combined into the final prediction by a decoder.

DilatedFCN In order to capture multi-scale context information on the high-resolution final feature map, PSPNet [38] performs pooling operations at multiple grid scales while DeepLabV3 [6] employs parallel atrous convolutions with different rates named ASPP. Alternatively, EncNet [36] utilizes the Context Encoding Module to capture global contextual information. Differently, our method proposes a joint upsampling module named JPU to replace the dilated convolutions in the backbone of DilatedFCNs, which can

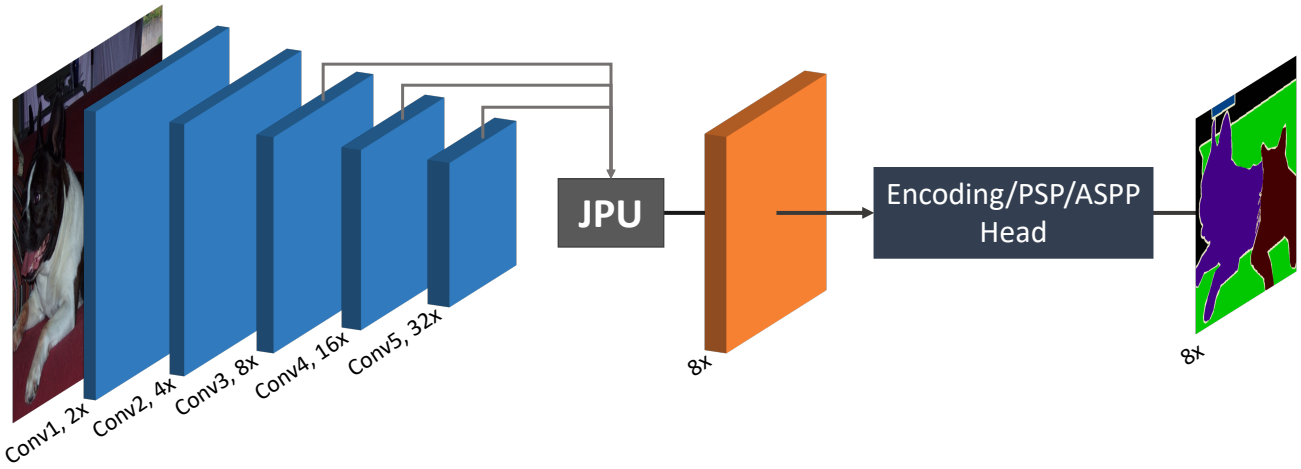


Figure 2: **Framework Overview of Our Method.** Our method employs the same backbone as the original FCN. After the backbone, a novel upsampling module named Joint Pyramid Upsampling (JPU) is proposed, which takes the last three feature maps as the inputs and generates a high-resolution feature map. A multi-scale/global context module is then employed to produce the final label map. Best viewed in color.

reduce computation complexity dramatically without performance loss.

EncoderDecoder To gradually recover the spatial information, [28] introduces skip connections to construct U-Net, which combines the encoder features and the corresponding decoder activations. [18] proposes a multi-path refinement network, which explicitly exploits all the information available along the down-sampling process. DeepLabV3+ [8] combines the advantages of DilatedFCN and EncoderDecoder, which employs DeepLabV3 as the encoder. Our method is complementary to DeepLabV3+, which can reduce the computation overload of DeepLabV3 without performance loss.

2.2. Upsampling

In our method, we propose a module to upsample a low-resolution feature map given high-resolution feature maps as guidance, which is closely related to joint upsampling as well as data-dependent upsampling.

Joint Upsampling In the literature of image processing, joint upsampling aims at leveraging the guidance image as a prior and transferring the structural details from the guidance image to the target image. [17] constructs a joint filter based on CNNs, which learns to recover the structure details in the guidance image. [31] proposes an end-to-end trainable guided filtering module, which upsamples a low-resolution image conditionally. Our method is related to the aforementioned approaches. However, the proposed JPU is designed for processing feature maps with a large number of channels while [17, 31] are specially designed for processing 3-channel images, which fail to capture the complex relations in high dimensional feature maps. Besides, the motivation and target of our method is completely different.

Data-Dependent Upsampling DUpsampling [29] is also related to our method, which takes advantages of the redundancy in the segmentation label space and is able to recover the pixel-wise prediction from low-resolution outputs of CNNs. Compared to our method, DUpsampling has a strong dependency on the label space, which generalizes poorly to a larger or more complex label space.

3. Method

In this section, we first introduce the most popular methods for semantic segmentation, named DilatedFCNs. We then reform the architecture of DilatedFCNs with a novel joint upsampling module, Joint Pyramid Upsampling (JPU). Finally, we discuss the proposed JPU in details, before which joint upsampling, dilated convolution and stride convolution are briefly introduced.

3.1. DilatedFCN

To exploit Deep CNNs in semantic segmentation, Long *et al*. [22] transform the CNN designed for image classification into FCN. Taking ResNet-101 as an example, the original CNN contains 5 convolution stages, a global average pooling layer and a linear layer. To construct an FCN, the global average pooling layer and the linear layer are replaced by a convolution layer, which is used to generate the final label map, as shown in Figure 1a. Between each two

consecutive convolution stages, stride convolutions and/or spatial pooling layers are employed, resulting in 5 feature maps with gradually reduced spatial resolutions.

The spatial resolution of the last feature map in FCN is reduced by a factor of 32, leading to inaccurate predictions about the locations and details. To obtain a final feature map with high resolution, DeepLab [5] removes the downsampling operations before the last two feature maps, as shown in Figure 1c. Besides, the convolution layers inside the last two convolution stages are replaced by dilated convolutions to maintain the receptive field of view, thus named DilatedFCN. As a result, the resolution of the last feature map is reduced by a factor of 8, which reserves more location and detail information. Following DeepLab, [38, 6] propose a multi-scale context module to capture context information from the last feature map, achieving tremendous success in several segmentation benchmarks.

3.2. The Framework of Our Method

To obtain a high-resolution final feature map, methods in DilatedFCN remove the last two downsampling operations from the original FCN, which bring in heavy computation complexity and memory footprint due to the enlarged feature maps. In this paper, we aim at seeking an alternative way to approximate the final feature map of DilatedFCN without computation and memory overload. Meanwhile, we expect the performance of our method to be as good as that of the original DilatedFCNs.

To achieve this, we first put back all the stride convolutions removed by DilatedFCN, while replacing all the dilated convolutions with regular convolution layers. As shown in Figure 2, the backbone of our method is the same as that of the original FCN, where the spatial resolutions of the five feature maps (Conv1−Conv5) are gradually reduced by a factor of 2. To obtain a feature map similar to the final feature map of DilatedFCN, we propose a novel module named Joint Pyramid Upsampling (JPU), which takes the last three feature maps (Conv3−Conv5) as inputs. Then a multi-scale context module (PSP [38]/ASPP [6]) or a global context module (Encoding [36]) is employed to produce the final predictions.

Compared to DilatedFCN, our method takes 4 times fewer computation and memory resources in 23 residual blocks (69 layers) and 16 times fewer in 3 blocks (9 layers) when the backbone is ResNet-101. Thus, our method runs much faster than DilatedFCN while consuming less memory.

3.3. Joint Pyramid Upsampling

The proposed JPU is designed for generating a feature map that approximates the activations of the final feature map from the backbone of DilatedFCN. Such a problem can be reformulated into joint upsampling, which is then resolved by a CNN designed for this task.

3.3.1 Background

Joint Upsampling Given a low-resolution target image and a high-resolution guidance image, joint upsampling aims at generating a high-resolution target image by transferring details and structures from the guidance image. Generally, the low-resolution target image y_l is generated by employing a transformation $f(\cdot)$ on the low-resolution guidance image x_l, *i.e.* $y_l = f(x_l)$. Given x_l and y_l, we are required to obtain a transformation $\hat{f}(\cdot)$ to approximate $f(\cdot)$, where the computation complexity of $\hat{f}(\cdot)$ is much lower than $f(\cdot)$. For example, if $f(\cdot)$ is a multi-layer perceptron (MLP), then $\hat{f}(\cdot)$ can be simplified as a linear transformation. The high-resolution target image y_h is then obtained by applying $\hat{f}(\cdot)$ on the high-resolution guidance image x_h, *i.e.* $y_h = \hat{f}(x_h)$. Formally, given x_l, y_l and x_h, joint upsampling is defined as follows:

$$y_h = \hat{f}(x_h), \text{ where } \hat{f}(\cdot) = \underset{h(\cdot)\in\mathcal{H}}{\operatorname{argmin}} ||y_l - h(x_l)||, \quad (1)$$

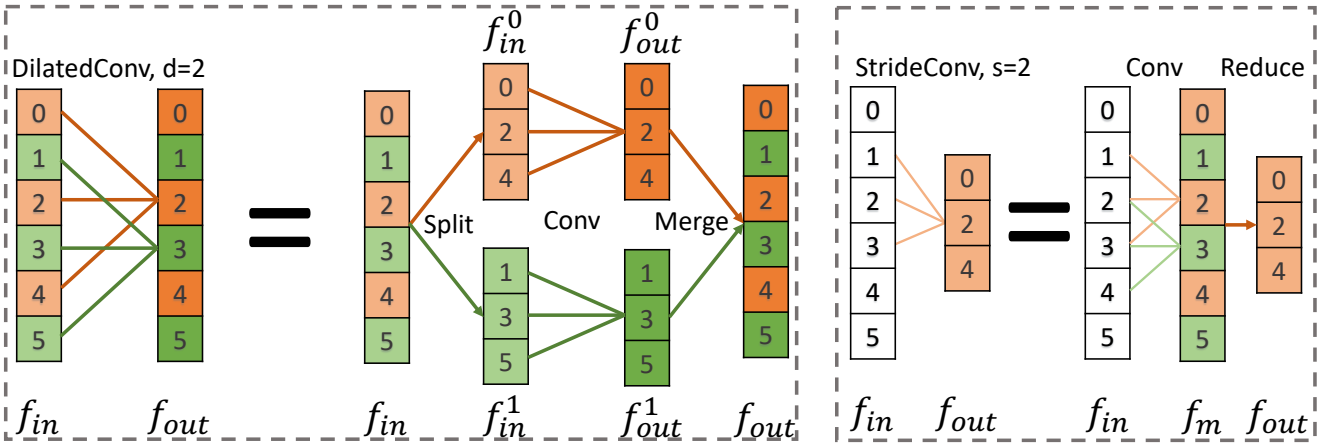
where $\mathcal{H}$ is a set of all possible transformation functions, and $||\cdot||$ is a pre-defined distance metric.

Dilated Convolution Dilated convolution is introduced in DeepLab [5] for obtaining high-resolution feature maps while maintaining the receptive field of view. Figure 3a gives an illustration of the dilated convolution in 1D (dilation rate = 2), which can be divided into the following three steps: (1) split the input feature f_{in} into two groups f_{in}^0 and f_{in}^1 according to the parity of the index, (2) process each feature with the same convolution layer, resulting in f_{out}^0 and f_{out}^1, and (3) merge the two generated features interlaced to obtain the output feature f_{out}.

Stride Convolution Stride convolution is proposed to transform the input feature into an output feature with reduced spatial resolution, which is equivalent to the following two steps as shown in Figure 3b: (1) process the input feature f_{in} with a regular convolution to obtain the intermediate feature f_m, and (2) remove the elements with an odd index, resulting in f_{out}.

3.3.2 Reformulating into Joint Upsampling

The differences between the backbone of our method and DilatedFCN lie on the last two convolution stages. Taking the 4th convolution stage (Conv4) as an example, in DilatedFCN, the input feature map is first processed by a regular convolution layer, followed by a series of dilated convolutions (d=2). Differently, our method first processes the input feature map with a stride convolution (s=2), and then employs several regular convolutions to generate the output.



(a) Dilated Convolution in 1D (d=2)

(b) Stride Convolution in 1D (s=2)

Figure 3: Dilated Convolution (dilation rate=2) and Stride Convolution (stride=2) in 1D. Best viewed in color.

Formally, given the input feature map x, the output feature map y_d in DilatedFCN is obtained as follows:

$$\begin{aligned} y_d &= x \to C_r \to \underbrace{C_d \to ..... \to C_d}_{n} \\ &= x \to C_r \to \underbrace{SC_rM \to ..... \to SC_rM}_{n} \text{ (Fig 3a)} \\ &= x \to C_r \to S \to \underbrace{C_r \to ..... \to C_r}_{n} \to M \\ &= y_m \to S \to C_r^n \to M \\ &= \{y_m^0, y_m^1\} \to C_r^n \to M \text{ (Fig 3a)}, \end{aligned} \tag{2}$$

while in our method, the output feature map y_s is generated as follows:

$$\begin{aligned} y_s &= x \to C_s \to \underbrace{C_r \to ..... \to C_r}_{n} \\ &= x \to C_r \to R \to \underbrace{C_r \to ..... \to C_r}_{n} \text{ (Fig 3b)} \\ &= y_m \to R \to C_r^n = y_m^0 \to C_r^n \text{ (Fig 3b)}. \end{aligned} \tag{3}$$

C_r, C_d, and C_s represent a regular/dilated/stride convolution respectively, and C_r^n is n layers of regular convolutions. S, M and R are split, merge, and reduce operations in Figure 3, where adjacent S and M operations can be canceled out. Notably, the convolutions in Equations 2 and 3 are in 1D, which is for simplicity. Similar results can be obtained for 2D convolutions.

The aforementioned equations show that y_s and y_d can be obtained with the same function C_r^n with different inputs: y_m^0 and y_m, where the former is downsampled from the latter. Thus, given x and y_s, the feature map y that approximates y_d can be obtained as follows:

$$\begin{aligned} y = &\{y_m^0, y_m^1\} \to \hat{h} \to M \\ &\text{where } \hat{h} = \underset{h \in \mathcal{H}}{\operatorname{argmin}} ||y_s - h(y_m^0)||, \\ &y_m = x \to C_r, \end{aligned} \tag{4}$$

which is the same as the joint upsampling problem defined in Equation 1. Similar conclusions can be easily obtained for the 5th convolution stage (Conv5).

3.3.3 Solving with CNNs

Equation 4 is an optimization problem, which takes lots of time to converge through the iterative gradient descent. Alternatively, we propose to approximate the optimization process with a CNN module. To achieve this, we first require to generate y_m given x, as shown in Equation 4. Then, features from y_m^0 and y_s need to be gathered for learning the mapping $\hat{h}$. Finally, a convolution block is required to transform the gathered features into the final prediction y.

Following the aforementioned analysis, we design the JPU module as in Figure 4. Concretely, each input feature map is firstly processed by a regular convolution block (Fig. 4a), which is designed for (1) generating y_m given x, and (2) transforming f_m into an embedding space with reduced dimensions. As a result, all the input features are mapped into the same space, which enables a better fusion and reduces the computation complexity.

Then, the generated feature maps are upsampled and concatenated, resulting in y_c (Fig. 4b). Four separable convolutions [14, 9] with different dilation rates (1, 2, 4, and 8) are employed in parallel to extract features from y_c, where different dilation rates take different functions. Concretely,

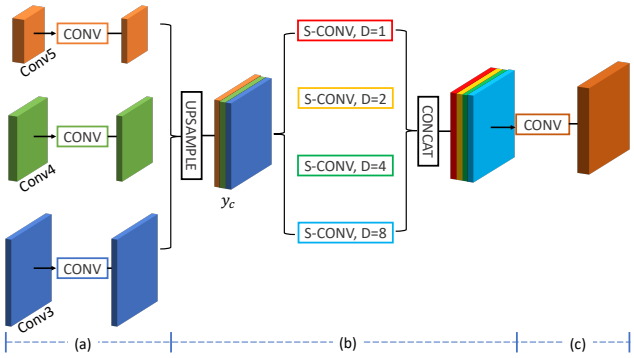


Figure 4: **The Proposed Joint Pyramid Upsampling (JPU).** Best viewed in color.

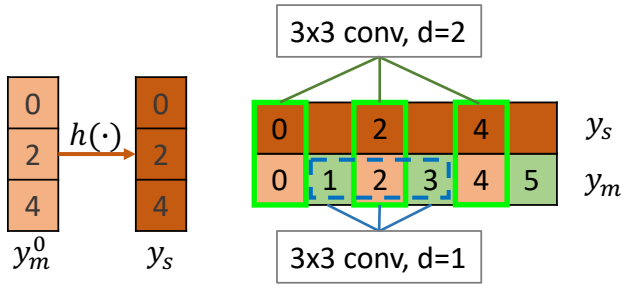


Figure 5: The convolution with dilation rate 1 focuses on y_m^0 and the rest part of y_m, and the convolution with dilation rate 2 aims at y_m^0 and y_s. Best viewed in color.

the convolution with dilation rate 1 is employed to capture the relation between y_m^0 and the rest part of y_m, as shown by the blue box in Figure 5. Alternatively, the convolutions with dilation rate 2, 4 and 8 are designed for learning the mapping $\hat{h}$ to transform y_m^0 into y_s, as shown by the green boxes in Figure 5. Thus, JPU can extract multi-scale context information from multi-level feature maps, which leads to a better performance. This is significantly different from ASPP [6], which only exploit the information in the last feature map.

The extracted features encode the mapping between y_m^0 and y_s as well as the relation between y_m^0 and the rest part of y_m. Thus, another regular convolution block is employed, which transforms the features into the final predictions (Fig. 4c).

Notably, the proposed JPU module solves two closely related joint upsampling problems jointly, which are (1) upsampling Conv4 based on Conv3 (the 4th convolution stage), and (2) upscaling Conv5 with the guidance of the enlarged Conv4 (the 5th convolution stage).

4. Experiment

In this section, we first introduce the datasets used in our experiments as well as the implementation details. We then conduct a systematic ablation study to show the effectiveness of the proposed JPU from the view of both performance and efficiency. Finally, to compare with the state-of-the-art methods, we report the performance on two segmentation datasets, Pascal Context [23] and ADE20K [40], which are widely used as the segmentation benchmarks. Moreover, we also show some visual results to demonstrate the superiority of our method.

4.1. Experimental Settings

Dataset Pascal Context dataset [23] is based on the PASCAL VOC 2010 detection challenge, which provides additional pixel-wise semantic annotations. There're 4,998 images for training (*train*) and 5,105 images for testing (*val*). Following the prior works [18, 5, 36], we use the most frequent 59 object categories plus background (60 classes in total) as the semantic labels.

Implementation Details Our method is implemented in PyTorch [25]. For training on Pascal Context, we follow the protocol presented in [36]. Concretely, we set the learning rate to 0.001 initially, which gradually decreases to 0 by following the "poly" strategy (power = 0.9). For data augmentation, we randomly scale (from 0.5 to 2.0) and left-right flip the input images. The images are then cropped to 480×480 and grouped with batch size 16. The network is trained for 80 epochs with SGD, of which the momentum is set to 0.9 and weight decay is set to 1e-4. All the experiments are conducted in a workstation with 4 Titan-Xp GPUs (12G per GPU). We employ pixel-wise cross-entropy as the loss function. ResNet-50 and ResNet-101 are used as the backbone, which are widely used in most existing segmentation methods as the standard backbones.

4.2. Ablation Study

To show the effectiveness of the proposed method, we conduct a systematical ablation study on Pascal Context dataset with ResNet-50 as the backbone, as shown in Table 1. We report the standard evaluation metrics of pixel accuracy (pixAcc) and mean Intersection of Union (mIoU). Notably, no multi-scale testing and left-right flipping are applied to the *val* images.

Dilated Convolutions For methods in DilatedFCN, the downsampling operations in the last two convolution stages are removed, resulting in the output stride (OS) to be 8. Encoding-8-None in Table 1 represents the original EncNet [36]. To show the effect of dilated convolutions, we replace the backbone of EncNet with that of the original FCN (the same as our method), resulting in the OS to be 32. We then upsample the last feature map by 4 times

Head	OS	Upsampling	pixAcc%	mIoU%
Encoding [36]	8	None	78.39	49.91
	32	Bilinear	76.10	46.47
		FPN [20]	78.16	49.59
		JPU (ours)	**78.98**	**51.05**
ASPP [6]	8	None	78.27	49.19
	32	JPU (ours)	78.79	50.07
PSP [38]	8	None	78.60	50.58
	32	JPU (ours)	78.91	50.89

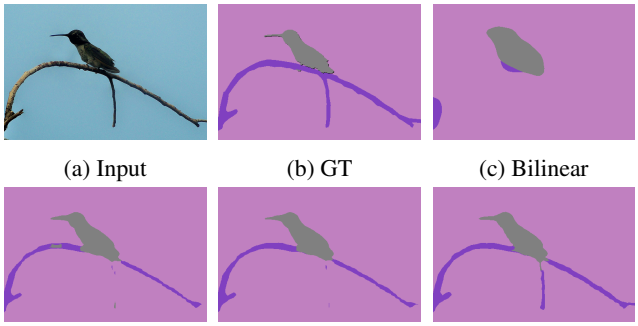
Table 1: Performance on the *val* set of Pascal Context dataset with the ResNet-50 as the backbone.

with bilinear interpolation before feeding it into the Encoding Head, noted as Encoding-32-Bilinear. As shown in Table 1, Encoding-32-Bilinear performs significantly worse than Encoding-8-None, which shows that it's not trivial to replace the dilated convolutions in the backbone of Dilated-FCNs.

Upsampling Module To show the effectiveness of the proposed JPU, we compare it with other classic upsampling methods, bilinear upsampling and feature pyramid network (FPN) [20]. As shown in Table 1, FPN outperforms bilinear interpolation by a large margin. Even compared with EncNet, FPN achieves comparable performance in both pixAcc and mIoU. By replacing FPN with our JPU, our method outperforms both FPN and EncNet by more than 1% in mIoU, which achieves the state-of-the-art performance.

The visual results are shown in Figure 6. Encoding-32-Bilinear (Fig. 6c) captures the global semantic information successfully, which gives a rough segmentation of the bird and sky. However, the boundary of the bird is inaccurate, and most parts of the branch are failed to be labeled out. When replacing bilinear interpolation with FPN (Fig. 6d), the bird and branch are labeled out successfully with accurate boundaries, which shows the effect of combing low-level and high-level feature maps. A slightly better result can be obtained with dilated convolutions (Fig. 6e). As for our method (Fig. 6f), it labels out both the main branch and the side shoot accurately, which shows the effectiveness of the proposed joint upsampling module. Particularly, the side shoot demonstrates the ability of JPU to extract multi-scale context from multi-level feature maps. Thus, our method can achieve a better performance.

Generalization to Other Methods To show the generalization ability of the proposed JPU, we replace EncNet with two popular methods in DilatedFCN, namely DeepLabV3 (ASPP Head) [6] and PSPNet [38]. As shown in Table 1, our methods transformed from DeepLabV3 and PSP outperforms the corresponding original methods consistently.



(a) Input (b) GT (c) Bilinear

(d) FPN [20] (e) EncNet [36] (f) Ours

Figure 6: Visual comparison of different upsampling modules with Encoding Head and ResNet-50 as the backbone.

Backbone	Head	Upsampling	FPS
ResNet-50	Encoding [36]	None	18.77
		Bilinear	45.67
		FPN [20]	37.87
		JPU (ours)	37.56
	ASPP [6]	None	15.99
		JPU (ours)	20.67
	PSP [38]	None	18.08
		JPU (ours)	28.48
ResNet-101	Encoding [36]	None	10.51
		Bilinear	35.20
		FPN [20]	32.40
		JPU (ours)	32.02
	ASPP [6]	None	10.46
		JPU (ours)	18.08
	PSP [38]	None	11.36
		JPU (ours)	23.87

Table 2: **Comparison of Computation Complexity.** The FPS is measured on a Titan-Xp GPU with a 512×512 image as input, which is averaged among 100 runs.

FPS To compare the computation complexity, we employ frame per second (FPS) as the evaluation metric, which is measured on a Titan-Xp GPU with a 512×512 image as input. As shown in Table 2, the reported FPS is averaged among 100 runs. For ResNet-50, our method (Encoding-JPU) runs about two times faster than EncNet (Encoding-None). When changing the backbone to ResNet-101, our method runs more than three times faster than EncNet. The speed of our method is also comparable to FPN, but our method achieves much better performance. As for DeepLabV3 (ASPP) and PSP, our method can accelerate them to a certain degree while having a better performance.

Method	Backbone	mIoU%
FCN-8s [22]		37.8
CRF-RNN [39]		39.3
ParseNet [21]		40.4
BoxSup [10]		40.5
HO_CRF [2]		41.3
Piecewise [19]		43.3
VeryDeep [32]		44.5
DeepLabV2 [5]	ResNet-101 + COCO	45.7
RefineNet [18]	ResNet-152	47.3
EncNet [36]	ResNet-101	51.7
DUpsampling [29]	Xception-71	52.5
EncNet+JPU (ours)	ResNet-50	51.2[2]
EncNet+JPU (ours)	ResNet-101	**53.1**

Table 3: The state-of-the-art methods on the *val* set of the Pascal Context dataset.

Method	Backbone	pixAcc%	mIoU%
FCN [22]		71.32	29.39
SegNet [3]		71.00	21.64
DilatedNet [35]		73.55	32.31
CascadeNet [40]		74.52	34.90
RefineNet [18]	ResNet-152	-	40.7
PSPNet [38]	ResNet-101	81.39	43.29
	ResNet-269	81.69	44.94
EncNet [36]	ResNet-50	79.73	41.11
	ResNet-101	81.69	44.65
Ours	ResNet-50	80.39	42.75
	ResNet-101	80.99	44.34

Table 4: Results on the *val* set of ADE20K dataset.

Rank	Team	Single Model	Final Score
1	CASIA_IVA_JD	✗	0.5547
2	WinterIsComing	✗	0.5544
-	PSPNet [38]	ResNet-269	0.5538
-	EncNet [36]	ResNet-101	0.5567
-	Ours	ResNet-101	**0.5584**

Table 5: Results on ADE20K *test* set. The first two entries ranked 1st and 2nd place in COCO-Place challenge 2017.

4.3. Comparison with Other Methods

Pascal Context In Table 1, our method employs ResNet-50 as the backbone without multi-scale evaluation, and the metrics are calculated on 59 classes excluding background by following [36]. To compare fairly with the state-of-the-art methods, we average the prediction in multiple scales and calculate the metrics among 60 classes including background, which are then reported in Table 3. With ResNet-50 as the backbone, our method outperforms DeepLabV2 (with COCO pretraining) and RefineNet by a large margin, which employ ResNet-101 and ResNet-152 as the backbone, respectively. Moreover, our method (ResNet-50) achieves competitive performance compared to EncNet with ResNet-101 as the backbone. By replacing ResNet-50 with a deeper network ResNet-101, our method gets an additional 1.9% improvement in mIoU, which outperforms EncNet (ResNet-101) and DUpsampling (Xception-71) significantly and achieves the state-of-the-art performance. Notably, Xception-71 is a much stronger backbone than ResNet-101. For completeness, we also report the mIoU on 59 classes (w/o background), which is 52.10%(ResNet-50) and 54.03% (ResNet-101).

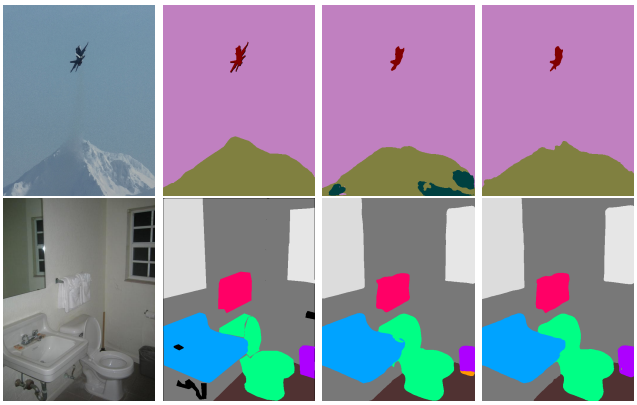
ADE20K ADE20K dataset [40] is a scene parsing benchmark, which contains 150 stuff/object categories. The dataset includes 20K/2K/3K images for training (*train*), validation (*val*), and testing (*test*).

We train our network on the *train* set for 120 epochs with learning rate 0.01. We then evaluate the model on the *val* set and report pixAcc and mIoU in Table 4. When employing ResNet-50 as the backbone, our method outperforms EncNet (ResNet-50) by 1.64% in mIoU, while achieving a much better performance compared to RefineNet (ResNet-152). By replacing ResNet-50 with ResNet-101, our method obtains competitive performance compared to EncNet (ResNet-101) and PSPNet (ResNet-269). Our method (ResNet-101) performs a little worse than EncNet, and we attribute this to the spatial resolution of the training images. Concretely, in our method, the training images are cropped to 480×480 for processing 4 images in a GPU with 12G memory. However, EncNet is trained with 576×576 images on GPUs with memory larger than 12G.

We then fine-tune our network on the *train* set and *val* set for another 20 epochs with learning rate 0.001. The predictions on the *test* set are submitted to the evaluation server. As shown in Table 5, our method outperforms two winning entries from the COCO-Place challenge 2017. Moreover, our method also achieves better performance compared to PSPNet and EncNet, although it performs worse on the *val* set. Notably, Final Score is the metric used in the evaluation server, which is the average of pixAcc and mIoU.

The visual results from both the Pascal Context dataset and the ADE20K dataset are shown in Figure 7. More results are shown in the supplementary material.

[2]Following [36], the mIoU reported in Table 1 is on 59 classes w/o background. In this table, the mIoU is measured on 60 classes w/ background for a fair comparison with other methods. Besides, we average the network prediction in multiple scales for evaluation in this table.



(a) Input (b) GT (c) EncNet [36] (d) Ours

Figure 7: Visual results of our method (ResNet-101). The first row is from Pascal Context *val* set, while the second row is from ADE20K *val* set. Best viewed in color.

5. Conclusion

In this paper, we have analyzed the differences and connections between dilated convolution and stride convolution. Based on the analysis, we formulated the task of extracting high-resolution feature maps into a joint upsampling problem and proposed a novel CNN module JPU to solve the problem. By replacing the time and memory consuming dilated convolutions with our JPU, the computation complexity is reduced by more than three times without performance loss. The ablation study shows that the proposed JPU is superior to other upsampling modules. By plugging JPU, several modern approaches for semantic segmentation achieve a better performance while runs much faster than before. Results on two segmentation datasets show that our method achieves the state-of-the-art performance while reducing the computation complexity dramatically.

References

[1] M. Amirul Islam, M. Rochan, N. D. Bruce, and Y. Wang. Gated feedback refinement network for dense image labeling. In *CVPR*, 2017. 2

[2] A. Arnab, S. Jayasumana, S. Zheng, and P. H. Torr. Higher order conditional random fields in deep neural networks. In *ECCV*, 2016. 8

[3] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv*, 2015. 1, 8

[4] H. Caesar, J. Uijlings, and V. Ferrari. Coco-stuff: Thing and stuff classes in context. In *CVPR*, 2018. 1

[5] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *TPAMI*, 2018. 1, 2, 4, 6, 8

[6] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv*, 2017. 1, 2, 4, 6, 7

[7] L.-C. Chen, Y. Yang, J. Wang, W. Xu, and A. L. Yuille. Attention to scale: Scale-aware semantic image segmentation. In *CVPR*, 2016. 2

[8] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *ECCV*, 2018. 3

[9] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *CVPR*, 2017. 5

[10] J. Dai, K. He, and J. Sun. Boxsup: Exploiting bounding boxes to supervise convolutional networks for semantic segmentation. In *ICCV*, 2015. 8

[11] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *TPAMI*, 2013. 2

[12] J. Fu, J. Liu, Y. Wang, and H. Lu. Stacked deconvolutional network for semantic segmentation. *arXiv*, 2017. 2

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1

[14] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv*, 2017. 5

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 1

[16] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. Gradient-based learning applied to document recognition. *IEEE*, 1998. 1

[17] Y. Li, J.-B. Huang, N. Ahuja, and M.-H. Yang. Deep joint image filtering. In *ECCV*, 2016. 3

[18] G. Lin, A. Milan, C. Shen, and I. D. Reid. Refinenet: Multi-path refinement networks for high-resolution semantic segmentation. In *CVPR*, 2017. 1, 2, 3, 6, 8

[19] G. Lin, C. Shen, A. Van Den Hengel, and I. Reid. Efficient piecewise training of deep structured models for semantic segmentation. In *CVPR*, 2016. 8

[20] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *CVPR*, 2017. 7

[21] W. Liu, A. Rabinovich, and A. C. Berg. Parsenet: Looking wider to see better. *arXiv*, 2015. 8

[22] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015. 1, 2, 3, 8

[23] R. Mottaghi, X. Chen, X. Liu, N.-G. Cho, S.-W. Lee, S. Fidler, R. Urtasun, and A. Yuille. The role of context for object detection and semantic segmentation in the wild. In *CVPR*, 2014. 1, 2, 6

[24] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In *ICCV*, 2015. 2

[25] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS Workshop*, 2017. 6

[26] C. Peng, X. Zhang, G. Yu, G. Luo, and J. Sun. Large kernel mattersimprove semantic segmentation by global convolutional network. In *CVPR*, 2017. 2

[27] T. Pohlen, A. Hermans, M. Mathias, and B. Leibe. Full-resolution residual networks for semantic segmentation in street scenes. In *CVPR*, 2017. 1

[28] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015. 1, 2, 3

[29] Z. Tian, C. Shen, T. He, and Y. Yan. Decoders matter for semantic segmentation: Data-dependent decoding enables flexible feature aggregation. *arXiv*, 2019. 3, 8

[30] Z. Wojna, V. Ferrari, S. Guadarrama, N. Silberman, L.-C. Chen, A. Fathi, and J. Uijlings. The devil is in the decoder. *arXiv*, 2017. 1

[31] H. Wu, S. Zheng, J. Zhang, and K. Huang. Fast end-to-end trainable guided filter. In *CVPR*, 2018. 3

[32] Z. Wu, C. Shen, and A. v. d. Hengel. Bridging category-level and instance-level semantic image segmentation. *arXiv*, 2016. 8

[33] C. Yu, J. Wang, C. Peng, C. Gao, G. Yu, and N. Sang. Learning a discriminative feature network for semantic segmentation. In *CVPR*, 2018. 2

[34] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. In *ICLR*, 2016. 2

[35] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv*, 2016. 8

[36] H. Zhang, K. Dana, J. Shi, Z. Zhang, X. Wang, A. Tyagi, and A. Agrawal. Context encoding for semantic segmentation. In *CVPR*, 2018. 1, 2, 4, 6, 7, 8, 9

[37] Z. Zhang, X. Zhang, C. Peng, D. Cheng, and J. Sun. Exfuse: Enhancing feature fusion for semantic segmentation. In *ECCV*, 2018. 2

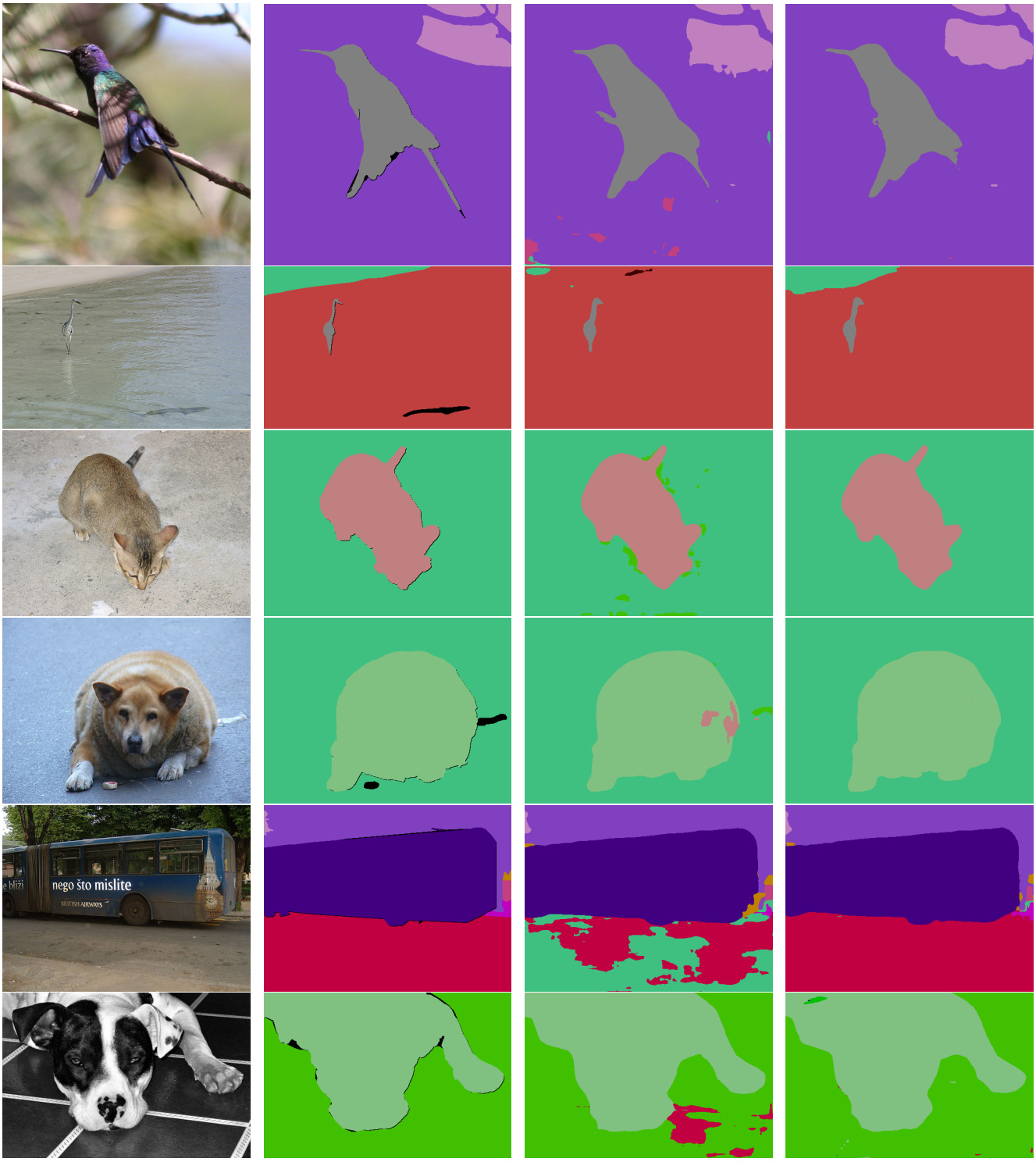
[38] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network. In *CVPR*, 2017. 1, 2, 4, 7, 8

[39] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. Torr. Conditional random fields as recurrent neural networks. In *ICCV*, 2015. 8

[40] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba. Scene parsing through ade20k dataset. In *CVPR*, 2017. 1, 2, 6, 8

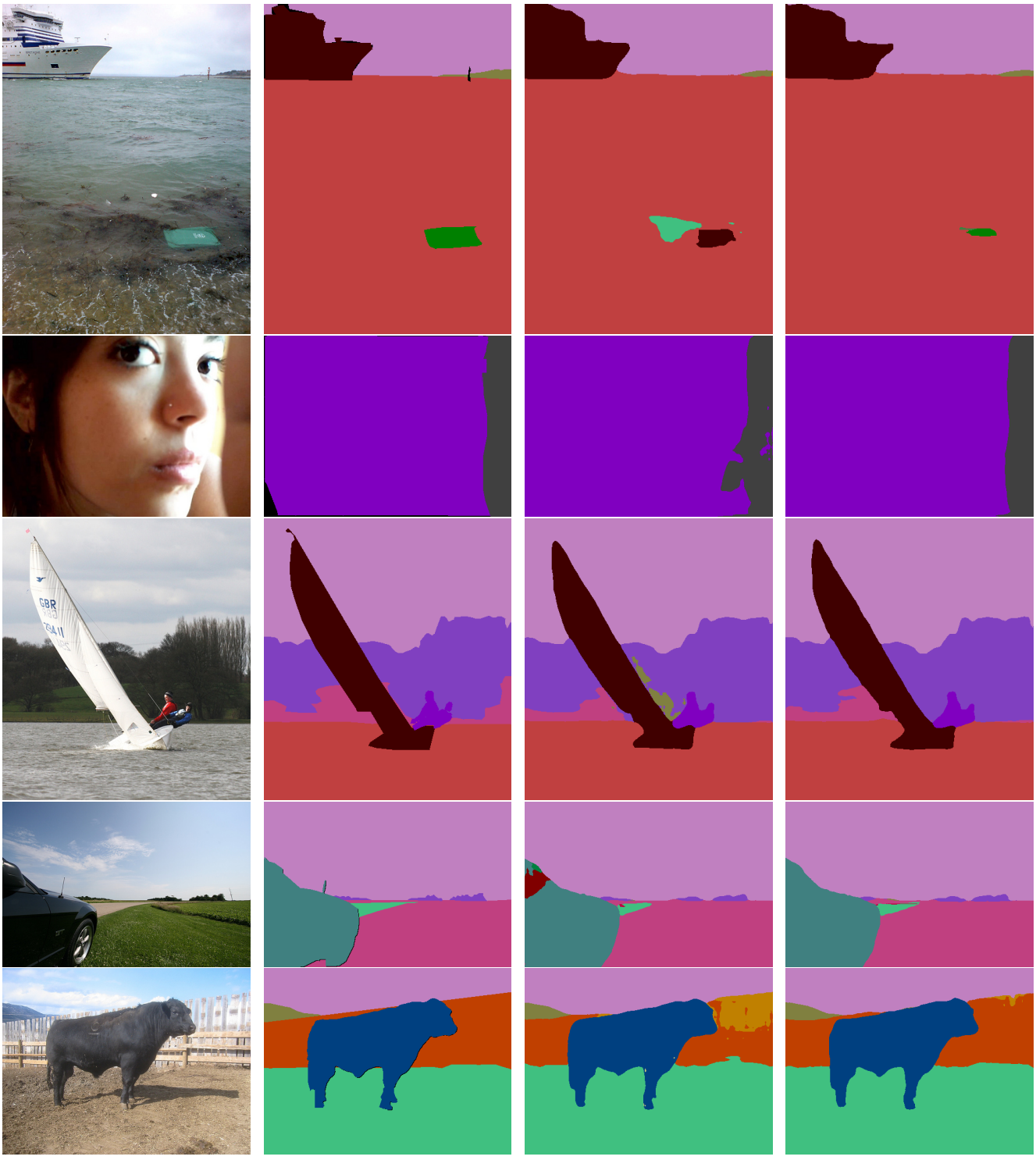
More Visual Results

See Figure 8−12 in the following pages for more visual results.



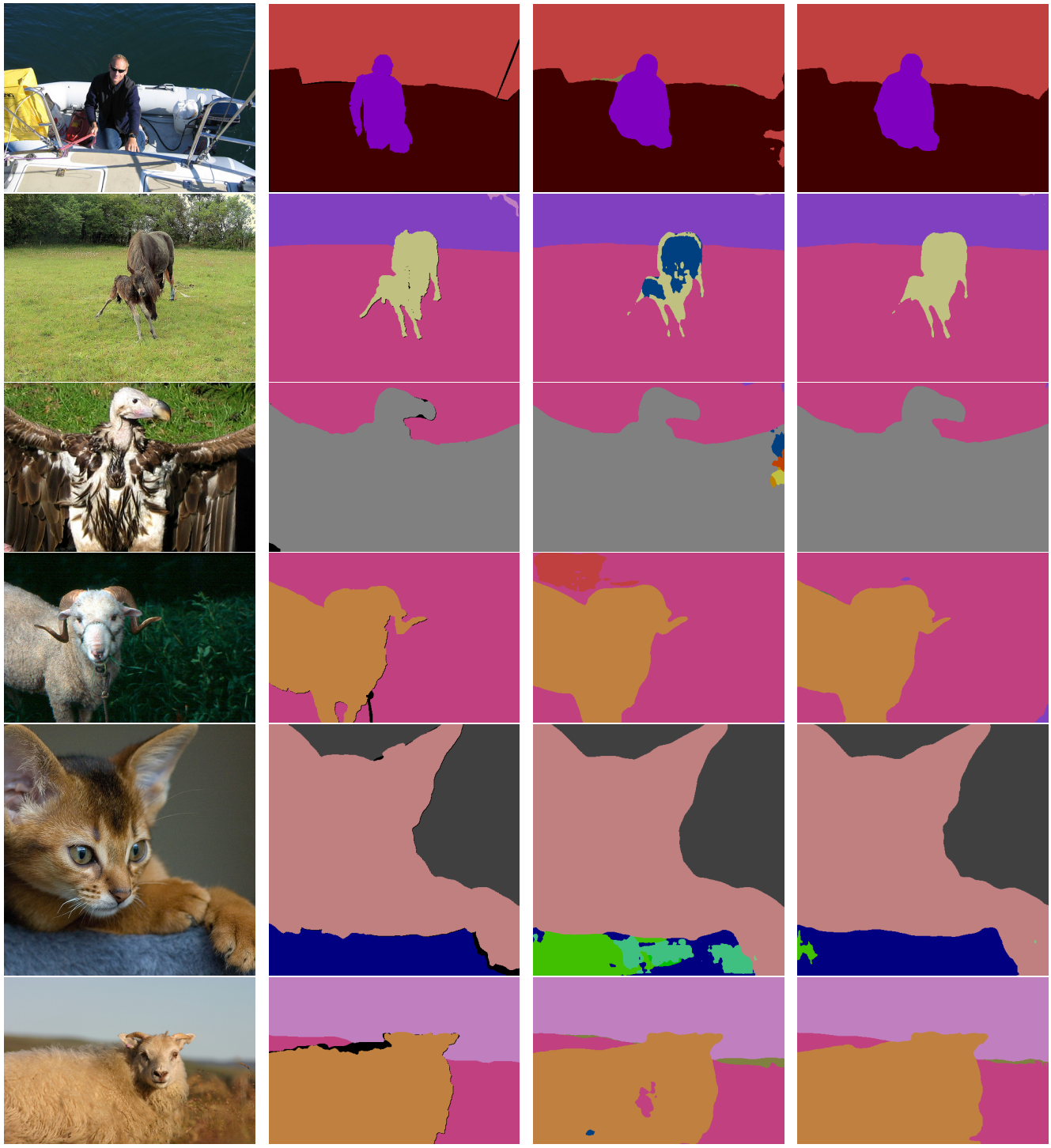
(a) Input (b) GT (c) EncNet (d) Ours

Figure 8: Visual results of our method (ResNet-101) on the Pascal Context *val* set. Best viewed in color.



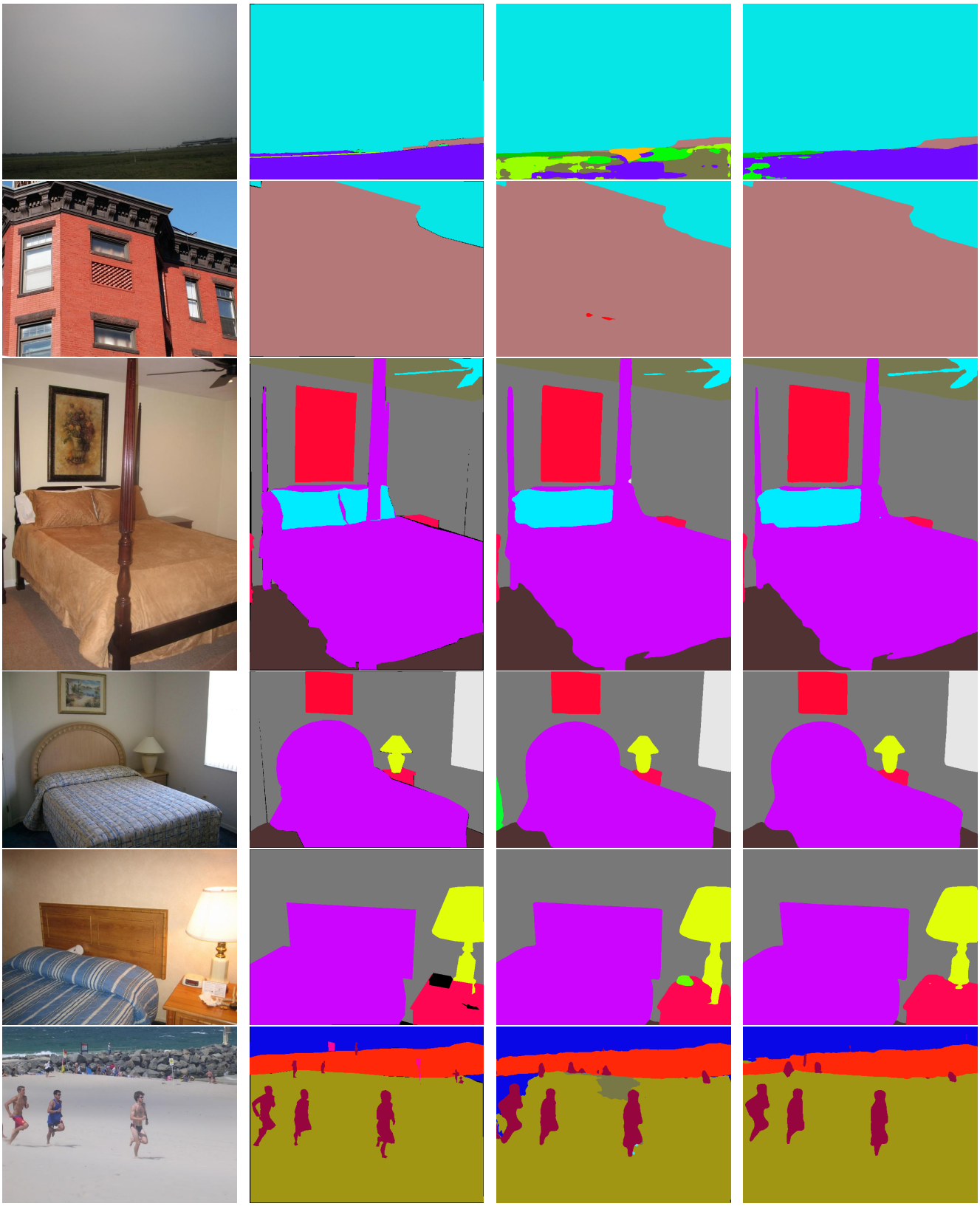
(a) Input (b) GT (c) EncNet (d) Ours

Figure 9: Visual results of our method (ResNet-101) on the Pascal Context *val* set. Best viewed in color.



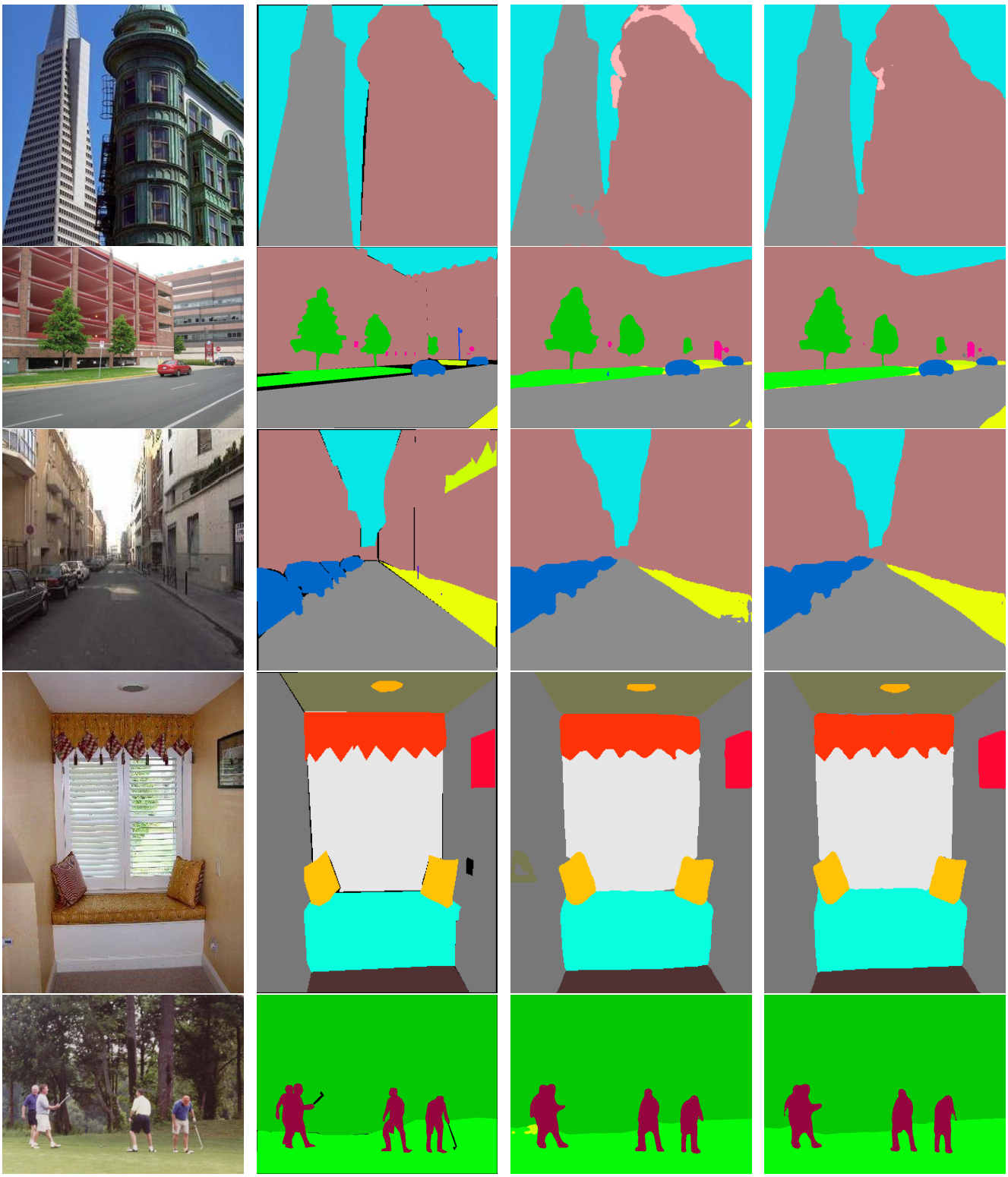
(a) Input (b) GT (c) EncNet (d) Ours

Figure 10: Visual results of our method (ResNet-101) on the Pascal Context *val* set. Best viewed in color.



(a) Input (b) GT (c) EncNet (d) Ours

Figure 11: Visual results of our method (ResNet-101) on the ADE20K *val* set. Best viewed in color.



(a) Input (b) GT (c) EncNet (d) Ours

Figure 12: Visual results of our method (ResNet-101) on the ADE20K *val* set. Best viewed in color.