

# Learning to Upsample by Learning to Sample

Wenze Liu

Hao Lu\*

Hongtao Fu

Zhiguo Cao

School of Artificial Intelligence and Automation,  
 Huazhong University of Science and Technology, China  
 {wzliu, hlu}@hust.edu.cn

## Abstract

We present *DySample*, an ultra-lightweight and effective dynamic upsampler. While impressive performance gains have been witnessed from recent kernel-based dynamic upsamplers such as CARAFE, FADE, and SAPA, they introduce much workload, mostly due to the time-consuming dynamic convolution and the additional sub-network used to generate dynamic kernels. Further, the need for high-res feature guidance of FADE and SAPA somehow limits their application scenarios. To address these concerns, we bypass dynamic convolution and formulate upsampling from the perspective of point sampling, which is more resource-efficient and can be easily implemented with the standard built-in function in PyTorch. We first showcase a naive design, and then demonstrate how to strengthen its upsampling behavior step by step towards our new upsampler, *DySample*. Compared with former kernel-based dynamic upsamplers, *DySample* requires no customized CUDA package and has much fewer parameters, FLOPs, GPU memory, and latency. Besides the light-weight characteristics, *DySample* outperforms other upsamplers across five dense prediction tasks, including semantic segmentation, object detection, instance segmentation, panoptic segmentation, and monocular depth estimation. Code is available at <https://github.com/tiny-smart/dysample>.

## 1. Introduction

Feature upsampling is a crucial ingredient in dense prediction models for gradually recovering the feature resolution. The most commonly used upsamplers are nearest neighbor (NN) and bilinear interpolation, which follows fixed rules to interpolate upsampled values. To increase flexibility, learnable upsamplers are introduced in some specific tasks, *e.g.*, deconvolution in instance segmentation [13] and pixel shuffle [34] in image super-resolution [31, 12, 22].

\*Corresponding author

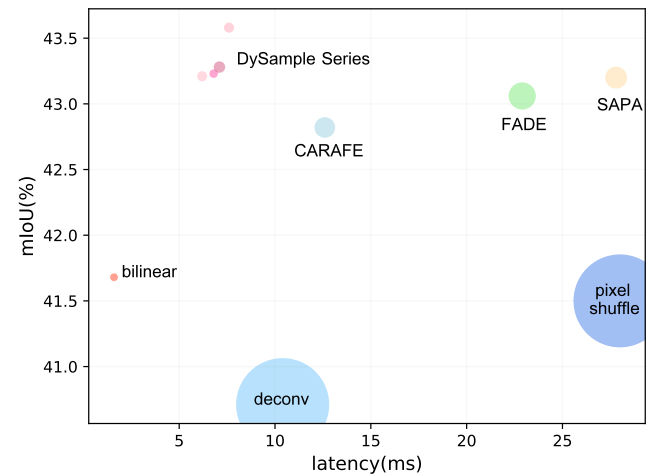


Figure 1. **Comparison of performance, inference speed, and GFLOPs of different upsamplers.** The circle size indicates the GFLOPs cost. The inference time is tested by  $\times 2$  upsampling a feature map of size  $256 \times 120 \times 120$ . The mIoU performance and additional GFLOPs are tested with SegFormer-B1 [40] on the ADE20K data set [42].

However, they either suffer from checkerboard artifacts [32] or seem not friendly to high-level tasks. With the popularity of dynamic networks [14], some dynamic upsamplers have shown great potential on several tasks. CARAFE [37] generates content-aware upsampling kernels to upsample the feature by dynamic convolution. The following work FADE [29] and SAPA [30] propose to combine both the high-res guiding feature and the low-res input feature to generate dynamic kernels, such that the upsampling process could be guided by the higher-res structure. These dynamic upsamplers are often of complicated structures, require customized CUDA implementation, and cost much more inference time than bilinear interpolation. Particularly for FADE and SAPA, the higher-res guiding feature introduces even more computational workload and narrows their application

scenarios (higher-res features must be available). Different from the early plain network [27], multi-scale features are often used in modern architectures; therefore the higher-res feature as an input into upsamplers may not be necessary. For example in Feature Pyramid Network (FPN) [23], the higher-res feature would add into the low-res feature after upsampling. As a result, we believe that a well-designed single-input dynamic upsampler would be sufficient.

Considering the heavy workload introduced by dynamic convolution, we bypass the kernel-based paradigm and return to the essence of upsampling, *i.e.*, point sampling, to reformulate the upsampling process. Specifically, we hypothesize that the input feature is interpolated to a continuous one with bilinear interpolation, and content-aware sampling points are generated to re-sample the continuous map. From this perspective, we first present a simple design, where point-wise offsets are generated by linear projection and used to re-sample point values with the `grid_sample` function in PyTorch. Then we showcase how to improve it with step-by-step tweaks by i) controlling the initial sampling position, ii) adjusting the moving scope of the offsets, and iii) dividing the upsampling process into several independent groups, and obtain our new upsampler, DySample. At each step, we explain why the tweak is required and conduct experiments to verify the performance gain.

Compared with other dynamic upsamplers, DySample i) does not need high-res guiding features as input nor ii) any extra CUDA packages other than PyTorch, and particularly, iii) has much less inference latency, memory footprint, FLOPs, and number of parameters, as shown in Fig. 1 and Fig. 8. For example, on semantic segmentation with MaskFormer-SwinB [8] as the baseline, DySample invites 46% more performance improvement than CARAFE, but requires only 3% number of parameters and 20% FLOPs of CARAFE. Thanks to the highly optimized PyTorch built-in function, the inference time of DySample also approaches to that of bilinear interpolation (6.2 ms vs. 1.6 ms when upsampling a  $256 \times 120 \times 120$  feature map). Besides these appealing light-weight characteristics, DySample reports better performance compared with other upsamplers across five dense prediction tasks, including semantic segmentation, object detection, instance segmentation, panoptic segmentation, and monocular depth estimation.

In a nutshell, we think DySample can safely replace NN/bilinear interpolation in existing dense prediction models, in light of not only effectiveness but also efficiency.

## 2. Related Work

We review dense prediction tasks, feature upsampling operators and dynamic sampling in deep learning.

**Dense Prediction Tasks.** Dense prediction refers to a branch of tasks that require point-wise label prediction,

such as semantic/instance/panoptic segmentation [2, 39, 40, 8, 7, 13, 11, 16, 19], object detection [33, 4, 24, 36], and monocular depth estimation [38, 18, 3, 21]. Different tasks often exhibit distinct characteristics and difficulties. For example, it is hard to predict both smooth interior regions and sharp edges in semantic segmentation and also difficult to distinguish different objects in instance-aware tasks. In depth estimation, pixels with the same semantic meaning may have rather different depths, and vice versa. One often has to customize different architectures for different tasks.

Though model structure varies, upsampling operators are essential ingredients in dense prediction models. Since a backbone typically outputs multi-scale features, the low-res ones need to be upsampled to higher resolution. Therefore a light-weight, effective upsampler would benefit many dense prediction models. We will show our new upsampler design brings a consistent performance boost on SegFormer [40] and MaskFormer [8] for semantic segmentation, on Faster R-CNN [33] for object detection, on Mask R-CNN [13] for instance segmentation, on Panoptic FPN [16] for panoptic segmentation, and on DepthFormer [21] for monocular depth estimation, while introducing negligible workload.

**Feature Upsampling.** The commonly used feature upsamplers are NN and bilinear interpolation. They apply fixed rules to interpolate the low-res feature, ignoring the semantic meaning in the feature map. Max unpooling has been adopted in semantic segmentation by SegNet [2] to preserve the edge information, but the introduction of noise and zero filling destroy the semantic consistency in smooth areas. Similar to convolution, some learnable upsamplers introduce learnable parameters in upsampling. For example, deconvolution upsamples features in a reverse fashion of convolution. Pixel Shuffle [34] uses convolution to increase the channel number ahead and then reshapes the feature map to increase the resolution.

Recently, some dynamic upsampling operators conduct content-aware upsampling. CARAFE [37] uses a sub-network to generate content-aware dynamic convolution kernels to reassemble the input feature. FADE [29] proposes to combine the high-res and low-res feature to generate dynamic kernels, for the sake of using the high-res structure. SAPA [30] further introduces the concept of point affiliation and computes similarity-aware kernels between high-res and low-res features. Being model plugins, these dynamic upsamplers increase more complexity than expected, especially for FADE and SAPA that require high-res feature input. Hence, our goal is to contribute a simple, fast, low-cost, and universal upsampler, while reserving the effectiveness of dynamic upsampling.

**Dynamic Sampling.** Upsampling is about modeling geometric information. A stream of work also models geometric information by dynamically sampling an image or a

feature map, as a substitution of the standard grid sampling. Dai *et al.* [9] and Zhu *et al.* [43] propose deformable convolutional networks, where the rectangular window sampling in standard convolution is replaced with shifted point sampling. Deformable DETR [44] follows this manner and samples key points relative to a certain query to conduct deformable attention. Similar practices also take place when images are downsampled to low-res ones for content-aware image resizing, *a.k.a.* seam carving [1]. *E.g.*, Zhang *et al.* [41] propose to learn to downsample an image with saliency guidance, in order to preserve more information of the original image, and Jin *et al.* [15] also set a learnable deformation module to downsample the images.

Different from recent kernel-based upsamplers, we interpret the essence of upsampling as point re-sampling. Therefore in feature upsampling, we tend to follow the same spirit as the work above and use simple designs to achieve a strong and efficient dynamic upsampler.

### 3. Learning to Sample and Upsample

In this section we elaborate the designs of DySample and its variants. We first present a naive implementation and then show how to improve it step by step.

#### 3.1. Preliminary

We return to the essence of upsampling, *i.e.*, point sampling, in the light of modeling geometric information. With the built-in function in PyTorch, we first provide a naive implementation to demonstrate the feasibility of sampling based dynamic upsampling (Fig. 2(a)).

**Grid Sampling.** Given a feature map  $\mathcal{X}$  of size  $C \times H_1 \times W_1$ , and a sampling set  $\mathcal{S}$  of size  $2 \times H_2 \times W_2$ , where 2 of the first dimension denotes the  $x$  and  $y$  coordinates, the `grid_sample` function uses the positions in  $\mathcal{S}$  to re-sample the hypothetical bilinear-interpolated  $\mathcal{X}$  into  $\mathcal{X}'$  of size  $C \times H_2 \times W_2$ . This process is defined by

$$\mathcal{X}' = \text{grid\_sample}(\mathcal{X}, \mathcal{S}). \quad (1)$$

**Naive Implementation.** Given an upsampling scale factor of  $s$  and a feature map  $\mathcal{X}$  of size  $C \times H \times W$ , a linear layer, whose input and output channel numbers are  $C$  and  $2s^2$ , is used to generate the offset  $\mathcal{O}$  of size  $2s^2 \times H \times W$ , which is then reshaped to  $2 \times sH \times sW$  by Pixel Shuffling [34]. Then the sampling set  $\mathcal{S}$  is the sum of the offset  $\mathcal{O}$  and the original sampling grid  $\mathcal{G}$ , *i.e.*,

$$\mathcal{O} = \text{linear}(\mathcal{X}), \quad (2)$$

$$\mathcal{S} = \mathcal{G} + \mathcal{O}, \quad (3)$$

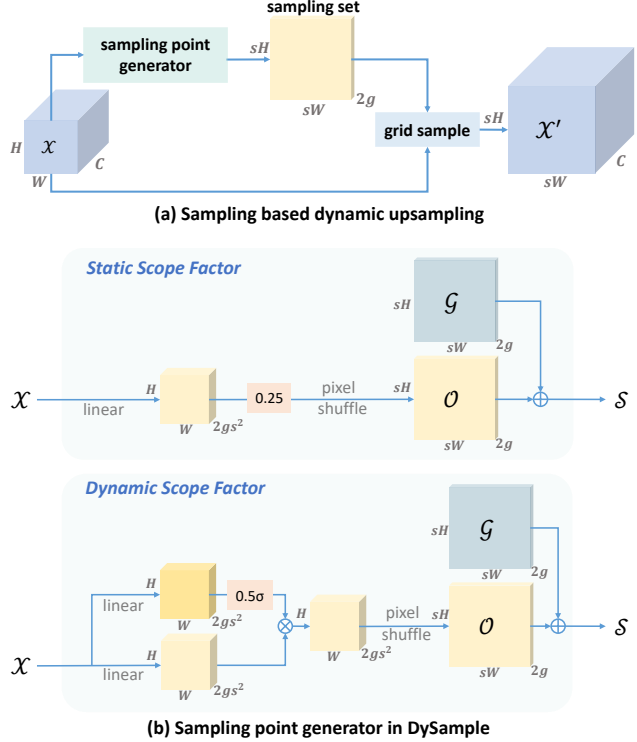


Figure 2. **Sampling based dynamic upsampling and module designs in DySample.** The input feature, upsampled feature, generated offset, and original grid are denoted by  $\mathcal{X}$ ,  $\mathcal{X}'$ ,  $\mathcal{O}$ , and  $\mathcal{G}$ , respectively. (a) The sampling set is generated by the sampling point generator, with which the input feature is re-sampled by the `grid_sample` function. In the generator (b), the sampling set is the sum of the generated offset and the original grid position. The upper box shows the version with the ‘static scope factor’, where the offset is generated with a linear layer. The bottom one describes the version with ‘dynamic scope factor’, where the a scope factor is first generated and then is used to modulate the offset. ‘ $\sigma$ ’ denotes the sigmoid function.

where the reshaping operation is omitted. Finally the upsampled feature map  $\mathcal{X}'$  of size  $C \times sH \times sW$  can be generated with the sampling set by `grid_sample` as Eq. (1).

This preliminary design obtains 37.9 AP with Faster R-CNN [33] on object detection [25] and 41.9 mIoU with SegFormer-B1 [40] on semantic segmentation [42] (cf. CARAFE: 38.6 AP and 42.8 mIoU). Next we present DySample upon this naive implementation.

#### 3.2. DySample: Upsampling by Dynamic Sampling

By studying the naive implementation, we observe that shared initial offset position among the  $s^2$  upsampled points neglects the position relation, and that the unconstrained walking scope of offsets can cause disordered point sam-

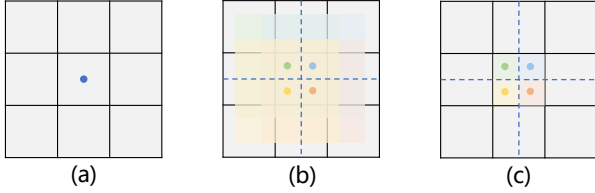


Figure 3. **Initial sampling positions and offset scopes.** The points and the colored masks represent the initial sampling positions and the offset scopes, respectively. Considering sampling four points ( $s = 2$ ), (a) in the case of nearest initialization, the four offsets share the same initial position but ignore position relation; in bilinear initialization (b), we separate the initial positions such that they distribute evenly. Without offset modulation (b), the offset scope would typically overlap, so in (c) we locally constrain the offset scope to reduce the overlap.

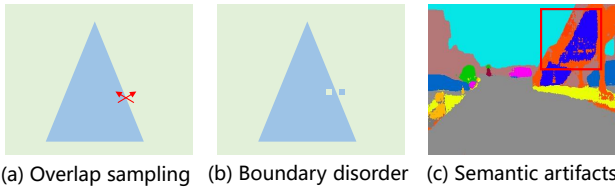


Figure 4. **Prediction artifacts due to offset overlap.** If the offsets overlap (a), the point value near boundaries may be in disorder (b), and the error would propagate layer by layer and finally cause prediction artifacts (c).

pling. We first discuss the two issues. We will also study implementation details such as feature groups and dynamic offset scope.

**Initial Sampling Position.** In the preliminary version, the  $s^2$  sampling positions w.r.t. one point in  $\mathcal{X}$  are all made fixed at the same initial position (the standard grid points in  $\mathcal{X}$ ), as shown in Fig. 3 (a). This practice ignores the position relation among the  $s^2$  neighboring points such that the initial sampling positions distribute unevenly. If the generated offsets are all zeros, the upsampled feature is equivalent to the NN interpolated one. Hence, this preliminary initialization can be called ‘nearest initialization’. Targeting this problem, we alter the initial position to ‘bilinear initialization’ as in Fig. 3(b), where zero offsets would bring the bilinearly interpolated feature map.

After changing the initial sampling position, the performance improves to 38.1 (+0.2) AP and 42.1 (+0.2) mIoU, as shown in Table 1.

**Offset Scope.** Due to the existence of normalization layers, the values of one certain output feature are typically in the range of  $[-1, 1]$ , centered at 0. Therefore, the walking scope of the local  $s^2$  sampling positions could overlap

Sampling Initialization	mIoU	AP
Nearest Initialization	41.9	37.9
Bilinear Initialization	<b>42.1</b>	<b>38.1</b>

Table 1. Ablation study on initial sampling position.

Factor	mIoU	AP	Groups	Dynamic	mIoU	AP
0.1	42.2	38.1	1		42.4	38.3
0.25	<b>42.4</b>	<b>38.3</b>	1	✓	42.6	38.4
0.5	42.2	38.1	4		43.2	38.6
1	42.1	38.1	4	✓	<b>43.3</b>	<b>38.7</b>

Table 2. Ablation study on the effect of static scope factor.

Table 3. Ablation study on the effect of dynamic scope factor.

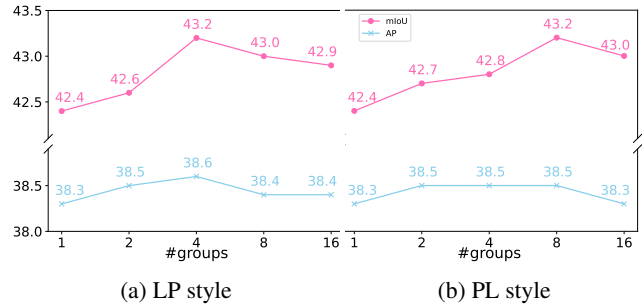


Figure 5. Ablation study on the number of feature groups.

significantly, as shown in Fig. 4(a). The overlap would easily influence the prediction near boundaries (Fig. 4(b)), and such errors would propagate stage by stage and cause output artifacts (Fig. 4(c)). To alleviate this, we multiply the offset by a factor of 0.25, which just meets the theoretical marginal condition between overlap and non-overlap. This factor is called the ‘static scope factor’, such that the walking scope of the sampling positions is locally constrained, as shown in Fig. 3(c). Here we rewrite Eq. (2) as

$$\mathcal{O} = 0.25 \text{ linear}(\mathcal{X}). \quad (4)$$

By setting the scope factor to 0.25, performance improves to 38.3 (+0.2) AP and 42.4 (+0.3) mIoU. We also test other possible factors, as shown in Table 2.

*Remark.* Multiplying the factor is a soft solution of the problem; it cannot completely solve it. We have also tried to strictly constrain the offset scope in  $[-0.25, 0.25]$  with  $\tanh$  function, but it works worse. Perhaps the explicit constraint limits the representation power, e.g., the explicit constraint version cannot handle the situation where some certain position expects a shift larger than 0.25.

**Grouping.** Here we study group-wise upsampling, where features share the same sampling set in each group. Specif-

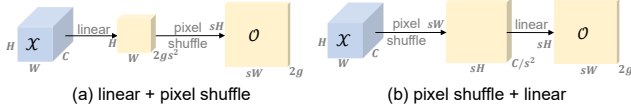


Figure 6. **Offset generation styles.** While the (a) ‘linear+pixel shuffle’ (LP) version requires more parameters than the (b) ‘pixel shuffle + linear’ (PL) version, the former is more flexible, consumes smaller memory footprint, and has faster inference speed.

ically, one can divide the feature map into  $g$  groups along the channel dimension and generate  $g$  groups of offsets.

According to Fig. 5, grouping works. When  $g = 4$ , performance reaches to 38.6 (+0.3) AP and 43.2 (+0.8) mIoU.

**Dynamic Scope Factor.** To increase the flexibility of the offset, we further generate point-wise ‘dynamic scope factors’ by linear projecting the input feature. By using the sigmoid function and a 0.5 static factor, the dynamic scope takes the value in the range of  $[0, 0.5]$ , centered at 0.25 as the static ones. The dynamic scope operation can refer to Fig. 2(b). Here we rewrite Eq. (4) as

$$\mathcal{O} = 0.5 \text{ sigmoid}(\text{linear}_1(\mathcal{X})) \cdot \text{linear}_2(\mathcal{X}). \quad (5)$$

Per Table 3, the dynamic scope factor further boosts the performance to 38.7 (+0.1) AP and 43.3 (+0.1) mIoU.

**Offset Generation Styles.** In the design above, linear projection is first used to produce  $s^2$  offset sets. The sets are then reshaped to satisfy the spatial size. We call this process as ‘linear+pixel shuffle’ (LP). To save parameters and GFLOPs, we can execute the reshaping operation ahead, *i.e.*, first reshaping the feature  $\mathcal{X}$  to the size of  $\frac{C}{s^2} \times sH \times sW$  and then linearly projecting it to  $2g \times sH \times sW$ . Similarly, we call this procedure ‘pixel shuffle+linear’ (PL). With other hyper parameters fixed, the number of parameters can be reduced to  $1/s^4$  under the PL setting. Through experiments, we empirically set the group number as 4 and 8 for the LP and PL version respectively according to Fig. 5. Further, we find that the PL version works better than the LP version on SegFormer (Table 4) and MaskFormer (Table 5), but slightly worse on other tested models.

**DySample Series.** According to the form of scope factor (static/dynamic) and offset generation styles (LP/PL), we investigate four variants:

- i) DySample: LP-style with the static scope factor;
- ii) DySample+: LP-style with the dynamic scope factor;
- iii) DySample-S: PL-style with the static scope factor;
- iv) DySample-S+: PL-style with dynamic scope factor.

### 3.3. How DySample works

The sampling process of DySample is visualized in Fig. 9. We highlight a (red boxed) local region to show how DySample divides one point on the edge to four to make the edge clearer. For the yellow boxed point, it generates four offsets pointing to the four upsampled points in sense of bilinear interpolation. In this example, the top left point is divided to the ‘sky’ (lighter), while the other three are divided to the ‘house’ (darker). The rightmost subplot indicates how the bottom right upsampled point is formed.

### 3.4. Complexity Analysis

We use a random feature map of size  $256 \times 120 \times 120$  (and a guidance map of size  $256 \times 240 \times 240$  if required) as the input to test the inference latency. We use SegFormer-B1 to compare the performance, training memory, training time, GFLOPs, and number of parameters when bilinear interpolation (default) is replaced by other upsamplers.

The quantitative results are shown in Fig. 8. Besides the best performances, DySample series cost the least inference latency, training memory, training time, GFLOPs, and number of parameters than all previous strong dynamic upsamplers. For the inference time, DySample series cost 6.2 ~ 7.6 ms to upsample a  $256 \times 120 \times 120$  feature map, which approaches to that of bilinear interpolation (1.6ms). Particularly, due to the use of highly optimized PyTorch built-in function, the backward propagation of DySample is rather fast; the increased training time is negligible.

Among DySample series, the ‘-S’ versions cost less parameters and GFLOPs, but more memory footprint and latency, because PL needs an extra storage of  $\mathcal{X}$ . The ‘+’ versions also introduce a bit more computational amount.

### 3.5. Discussion on Related Work

Here we compare DySample with CARAFE [37], SAPA [30] and deformable attention [44].

**Relation to CARAFE.** CARAFE generates content-aware upsampling kernels to reassemble the input feature. In DySample, we generate upsampling positions instead of kernels. Under the kernel-based view, DySample uses  $2 \times 2$  bilinear kernels, while CARAFE uses  $5 \times 5$  ones. In CARAFE if placing a kernel centered at a point, the kernel size must at least be  $3 \times 3$ , so the GFLOPs is at least 2.25 times larger than DySample. Besides, the upsampling kernel weights in CARAFE are learned, but in DySample they are conditioned on the  $x$  and  $y$  position. Therefore, to maintain a single kernel DySample only needs a 2-channel feature map (given that the group number  $g = 1$ ), but CARAFE requires a  $K \times K$ -channel one, which explains why DySample is more efficient.

**Relation to SAPA.** SAPA introduces the concept of semantic cluster into feature upsampling and views the up-

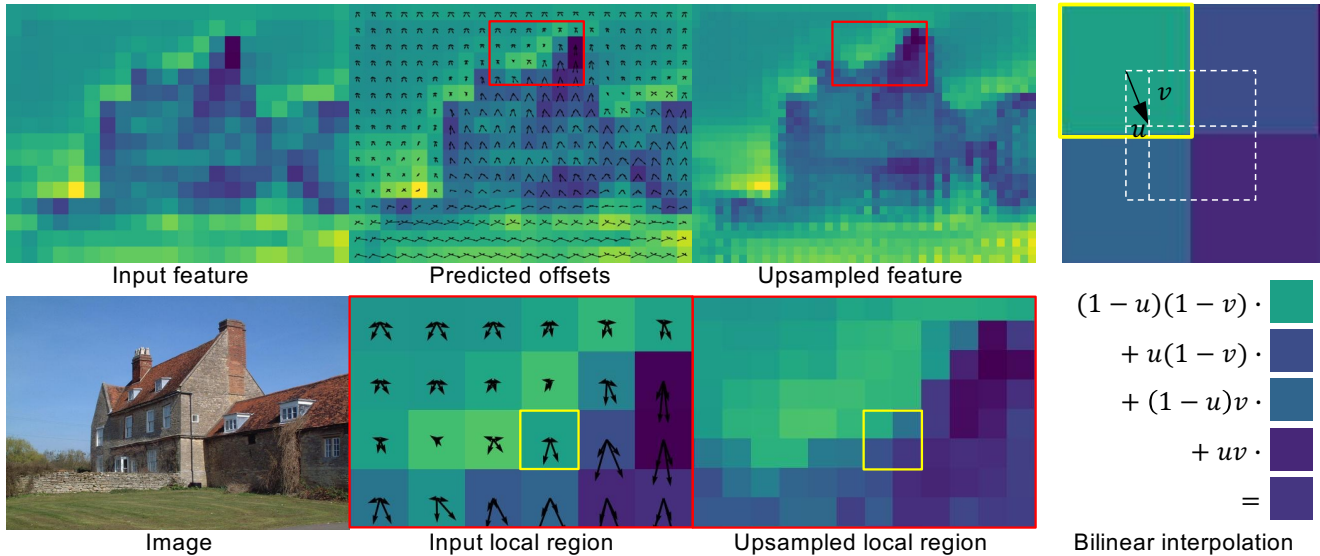


Figure 7. **Visualization of the upsampling process in DySample.** A part of the boundary in red box is highlighted for a close view. We generate content-aware offsets to construct new sampling points to resample the input feature map with bilinear interpolation. The new sampling positions are indicated by the arrowheads. The yellow boxed point in the low-res feature is selected to illustrate the bilinear interpolation process.

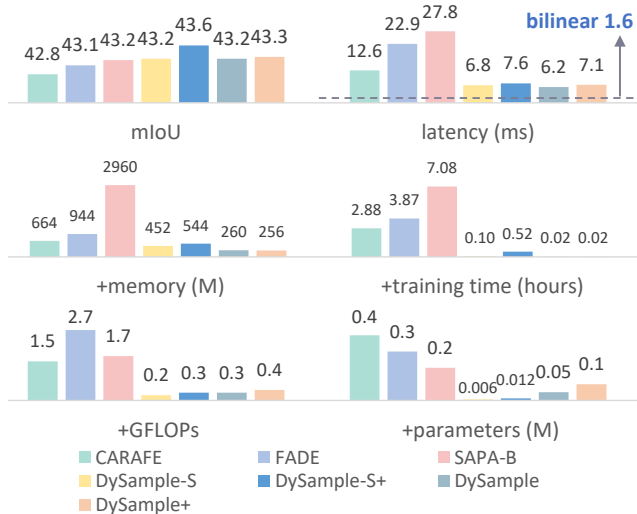


Figure 8. **Complexity analysis.** DySample series achieve the overall best performances on SegFormer-B1 [40], and cost the least latency, memory footprint, training time, GFLOPs, and number of parameters among the recent strong dynamic upsamplers. The inference time is tested by upsampling a  $256 \times 120 \times 120$  feature map (and a  $256 \times 240 \times 240$  guidance feature if needed) with a single Nvidia GTX 3090 GPU on a server. ‘+’ means the additional amount compared with bilinear interpolation.

sampling process as finding a correct semantic cluster for each upsampling point. In DySample, offset generation can

also be seen as seeking for a semantically similar region for each point. However, DySample does not need the guidance map and thus is more efficient and easy-to-use.

**Relation to Deformable Attention.** Deformable attention [44] mainly enhances features; it samples many points at each position to *aggregate* them to form a new point. But DySample is tailored for upsampling; it samples a single point for each upsampled position to *divide* one point to  $s^2$  upsampled points. DySample reveals that sampling a single point for each upsampled position is enough as long as the upsampled  $s^2$  points can be dynamically divided.

## 4. Applications

Here we apply DySample on five dense prediction tasks, including semantic segmentation, object detection, instance segmentation, panoptic segmentation, and depth estimation. Among the upsampler competitors, in bilinear interpolation, we set the scale factor as 2 and ‘align corners’ as False. For deconvolution, we set the kernel size as 3, the stride as 2, the padding as 1 and the output padding as 1. For pixel shuffle [34], we first use a 3-kernel size convolution to increase the channel number to 4 times of the original one, and then apply the ‘pixel shuffle’ function. For CARAFE [37], we adopt its default setting. The ‘HIN’ version of IndexNet [28] and the ‘dynamic-cs-d†’ version of A2U [10] are used. FADE [29] without gating mechanism and SAPA-B [30] are used because they are more stable across all the dense prediction tasks.

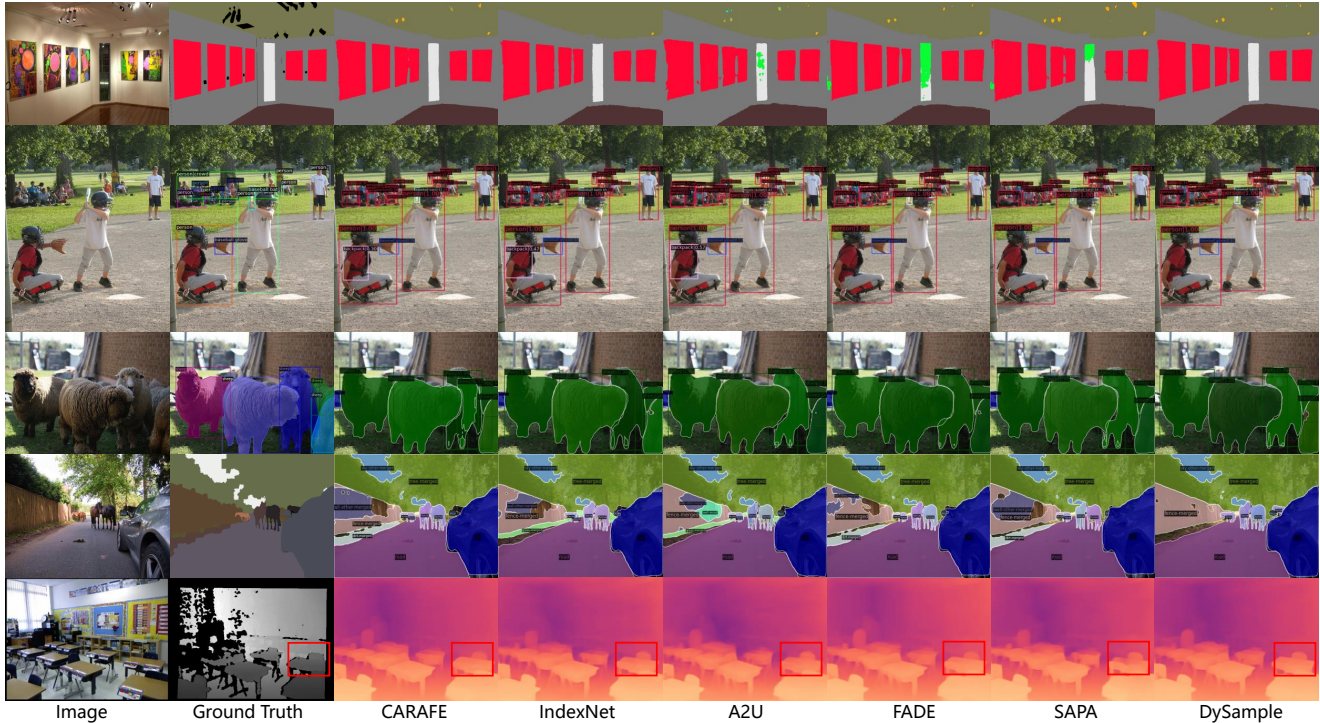


Figure 9. **Qualitative visualizations.** From top to bottom: semantic segmentation, object detection, instance segmentation, panoptic segmentation, and monocular depth estimation.

#### 4.1. Semantic Segmentation

Semantic segmentation infers per-pixel class labels. Upsamplers are often adopted several times to obtain the high-resolution output in typical models. The precise per-pixel prediction is largely dependent on the upsampling quality.

**Experimental Protocols.** We use the ADE20K [42] dataset. Besides the commonly used mIoU metric, we also report the bIoU [6] metric to evaluate the boundary quality. We first use a light-weight baseline SegFormer-B1 [40], where  $3 + 2 + 1 = 6$  upsampling stages are involved, and then test DySample on a stronger baseline MaskFormer [8], with Swin-B [26] and Swin-L as the backbone, where 3 upsampling stages are involved in FPN. We use the official codebase provided by the authors and follow all the training settings except for only modifying the upsampling stages.

**Semantic Segmentation Results.** Quantitative results are shown in Tables 4 and 5. We can see DySample achieves the best mIoU metric of 43.58 on SegFormer-B1, but the bIoU metric is lower than those guided upsamplers such as FADE and SAPA. Therefore we can infer that DySample improves the performance mainly from the interior regions, and the guided upsamplers mainly improve boundary quality. As shown in Fig. 9 row 1, the output of DySample is similar to that of CARAFE, but more distinctive near boundaries; the guided upsamplers predict sharper boundaries, but

SegFormer-B1	FLOPs	Params	mIoU	bIoU
Bilinear	15.9	13.7M	41.68	27.80
Deconv	+34.4	+3.5M	40.71	25.94
PixelShuffle [34]	+34.4	+14.2M	41.50	26.58
CARAFE [37]	+1.5	+0.4M	42.82	29.84
IndexNet [28]	+30.7	+12.6M	41.50	28.27
A2U [10]	+0.4	+0.1M	41.45	27.31
FADE [29]	+2.7	+0.3M	43.06	<b>31.68</b>
SAPA-B [30]	+1.0	+0.1M	43.20	<u>30.96</u>
DySample-S	+0.2	+6.1K	43.23	29.53
DySample-S+	+0.3	+12.3K	<b>43.58</b>	29.93
DySample	+0.3	+49.2K	43.21	29.12
DySample+	+0.4	+0.1M	<u>43.28</u>	29.23

Table 4. Semantic segmentation results with SegFormer-B1 on ADE20K. Best performance is in **boldface** and second best is underlined.

have wrong predictions on interior regions. For the stronger baseline MaskFormer, DySample also improves the mIoU metric from 52.70 to 53.91 (+1.21) with Swin-B and from 54.10 to 54.90 (+0.80) with Swin-L.

#### 4.2. Object Detection and Instance Segmentation

Being instance-level tasks, object detection aims to localize and classify objects, while instance segmentation need to further segment the objects. The quality of the up-

Backbone	Upsampler	mIoU
Swin-B	Nearest	52.70
	CARAFE	<u>53.53</u>
	DySample-S+	<b>53.91</b>
Swin-L	Nearest	54.10
	CARAFE	<u>54.61</u>
	DySample-S+	<b>54.90</b>

Table 5. Semantic segmentation results with MaskFormer on ADE20K. Best performance is in **boldface** and second best is underlined.

Faster R-CNN	Backbone	Params	AP	AP <sub>50</sub>	AP <sub>75</sub>	AP <sub>S</sub>	AP <sub>M</sub>	AP <sub>L</sub>
Nearest	R50	46.8M	37.5	58.2	40.8	21.3	41.1	48.9
Deconv	R50	+2.4M	37.3	57.8	40.3	21.3	41.1	48.0
PixelShuffle [34]	R50	+9.4M	37.5	58.5	40.4	21.5	41.5	48.3
CARAFE [37]	R50	+0.3M	<u>38.6</u>	<u>59.9</u>	<b>42.2</b>	<b>23.3</b>	<u>42.2</u>	49.7
IndexNet [28]	R50	+8.4M	37.6	58.4	40.9	21.5	41.3	49.2
A2U [10]	R50	+38.9K	37.3	58.7	40.0	21.7	41.1	48.5
FADE [29]	R50	+0.2M	38.5	59.6	41.8	<u>23.1</u>	<u>42.2</u>	49.3
SAPA-B [30]	R50	+0.1M	37.8	59.2	40.6	22.4	41.4	49.1
DySample-S	R50	+4.1K	38.5	59.5	<u>42.1</u>	22.7	41.9	<b>50.2</b>
DySample-S+	R50	+8.2K	<u>38.6</u>	<u>59.8</u>	<u>42.1</u>	22.5	42.1	<u>50.0</u>
DySample	R50	+32.7K	<u>38.6</u>	<u>59.9</u>	42.0	22.9	42.1	<b>50.2</b>
DySample+	R50	+65.5K	<b>38.7</b>	<b>60.0</b>	<b>42.2</b>	<b>22.5</b>	<b>42.4</b>	<b>50.2</b>
Nearest	R101	65.8M	39.4	60.1	43.1	22.4	43.7	51.1
DySample+	R101	+65.5K	<b>40.5</b>	<b>61.6</b>	<b>43.8</b>	<b>24.2</b>	<b>44.5</b>	<b>52.3</b>

Table 6. Object detection results with Faster R-CNN on MS COCO. Best performance is in **boldface** and second best is underlined.

sampled features can have large effect on the classification, localization, and segmentation accuracy.

**Experimental Protocols.** We use the MS COCO [25] data set. The AP series metrics are reported. Faster R-CNN [33] and Mask R-CNN [13] are chosen as the baselines. We modify the upsamplers in the FPN architecture for performance comparison. There are four and three upsampling stages in the FPN of Faster R-CNN and of Mask R-CNN, respectively. We use the code provided by mmdetection [5] and follow the 1× training settings.

### Object Detection and Instance Segmentation Results.

Quantitative results are shown in Tables 6 and 7. Results show that DySample outperforms all compared upsamplers. With R50, DySample achieves the best performance among all tested upsamplers. When a stronger backbone is used, notable improvements can also be witnessed (R50 +1.2 vs. R101 +1.1 box AP on Faster R-CNN, and R50 +1.0 vs. R101 +0.8 mask AP on Mask R-CNN).

Mask R-CNN	Task	Backbone	AP	AP <sub>50</sub>	AP <sub>75</sub>	AP <sub>S</sub>	AP <sub>M</sub>	AP <sub>L</sub>	
Nearest	Bbox	R50	38.3	58.7	42.0	21.9	41.8	50.2	
Deconv		R50	37.9	58.5	41.0	22.0	41.6	49.0	
PixelShuffle [34]		R50	38.5	59.4	41.9	22.0	42.3	49.8	
CARAFE [37]		R50	39.2	60.0	<u>43.0</u>	23.0	<u>42.8</u>	50.8	
IndexNet [28]		R50	38.4	59.2	41.7	22.1	41.7	50.3	
A2U [10]		R50	38.2	59.2	41.4	22.3	41.7	49.6	
FADE [29]		R50	39.1	<u>60.3</u>	42.4	<b>23.6</b>	42.3	51.0	
SAPA-B [30]		R50	38.7	59.7	42.2	23.1	41.8	49.9	
DySample-S		R50	<u>39.3</u>	<b>60.4</b>	<u>43.0</u>	23.2	42.7	<u>51.1</u>	
DySample-S+		R50	<u>39.3</u>	<u>60.3</u>	42.8	23.2	42.7	50.8	
DySample		R50	39.2	<u>60.3</u>	<u>43.0</u>	<u>23.5</u>	42.5	51.0	
DySample+		R50	<b>39.6</b>	<b>60.4</b>	<b>43.5</b>	23.4	<b>42.9</b>	<b>51.7</b>	
Nearest		R101	40.0	60.4	43.7	22.8	43.7	52.0	
DySample+		R101	<b>41.0</b>	<b>61.9</b>	<b>44.9</b>	<b>24.3</b>	<b>45.0</b>	<b>53.5</b>	
Nearest		Segm	R50	34.7	55.8	37.2	16.1	37.3	50.8
Deconv			R50	34.5	55.5	36.8	16.4	37.0	49.5
PixelShuffle [34]	R50		34.8	56.0	37.3	16.3	37.5	50.4	
CARAFE [37]	R50		35.4	56.7	37.6	16.9	<u>38.1</u>	51.3	
IndexNet [28]	R50		34.7	55.9	37.1	16.0	37.0	51.1	
A2U [10]	R50		34.6	56.0	36.8	16.1	37.4	50.3	
FADE [29]	R50		35.1	56.7	37.2	16.7	37.5	51.4	
SAPA-B [30]	R50		35.1	56.5	37.4	16.7	37.6	50.6	
DySample-S	R50		35.4	56.8	<u>37.8</u>	16.7	38.0	51.4	
DySample-S+	R50		<u>35.5</u>	56.8	<u>37.8</u>	17.0	37.9	<b>51.9</b>	
DySample	R50		35.4	<u>56.9</u>	<u>37.8</u>	<u>17.1</u>	37.7	51.1	
DySample+	R50		<b>35.7</b>	<b>57.3</b>	<b>38.2</b>	<b>17.3</b>	<b>38.2</b>	<u>51.8</u>	
Nearest	R101		36.0	57.6	38.5	16.5	39.3	52.2	
DySample+	R101		<b>36.8</b>	<b>58.7</b>	<b>39.5</b>	<b>17.5</b>	<b>40.0</b>	<b>53.8</b>	

Table 7. Instance segmentation results with Mask R-CNN on MS COCO. The parameter increment is identical as in Faster R-CNN. Best performance is in **boldface** and second best is underlined.

Panoptic FPN	Backbone	Params	PQ	PQ <sup>th</sup>	PQ <sup>st</sup>	SQ	RQ
Nearest	R50	46.0M	40.2	47.8	28.9	77.8	49.3
Deconv	R50	+1.8M	39.6	47.0	28.4	77.1	48.5
PixelShuffle [34]	R50	+7.1M	40.0	47.4	28.8	77.1	49.1
CARAFE [37]	R50	+0.2M	40.8	47.7	30.4	78.2	50.0
IndexNet [28]	R50	+6.3M	40.2	47.6	28.9	77.1	49.3
A2U [10]	R50	+29.2K	40.1	47.6	28.7	77.3	48.0
FADE [29]	R50	+0.1M	40.9	48.0	30.3	78.1	50.1
SAPA-B [30]	R50	+0.1M	40.6	47.7	29.8	78.0	49.6
DySample-S	R50	+3.1K	40.6	48.0	29.6	78.0	49.8
DySample-S+	R50	+6.2K	41.1	<u>48.1</u>	30.5	78.2	<u>50.2</u>
DySample	R50	+24.6K	<u>41.4</u>	<b>48.5</b>	<u>30.7</u>	<b>78.6</b>	<b>50.7</b>
DySample+	R50	+49.2K	<b>41.5</b>	<b>48.5</b>	<b>30.8</b>	<u>78.3</u>	<b>50.7</b>
Nearest	R101	65.0M	42.2	50.1	30.3	78.3	51.4
DySample+	R101	+49.2K	<b>43.0</b>	<b>50.2</b>	<b>32.1</b>	<b>78.6</b>	<b>52.4</b>

Table 8. Panoptic segmentation results with Panoptic FPN on MS COCO. Best performance is in **boldface** and second best is underlined.



DepthFormer	Params	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$	Abs Rel	RMS	log10	RMS(log)	Sq Rel
Bilinear	47.6M	0.873	0.978	<u>0.994</u>	0.120	0.402	<u>0.050</u>	0.148	0.071
Deconv	+7.1M	0.872	<b>0.980</b>	<b>0.995</b>	<u>0.117</u>	0.401	<u>0.050</u>	0.147	<b>0.067</b>
PixelShuffle	+28.2M	0.874	<u>0.979</u>	<b>0.995</b>	<u>0.117</u>	<u>0.395</u>	<b>0.049</b>	<u>0.146</u>	<u>0.068</u>
CARAFE [37]	+0.3M	<u>0.877</u>	0.978	<b>0.995</b>	<b>0.116</b>	0.397	<b>0.049</b>	<u>0.146</u>	0.069
IndexNet [28]	+6.3M	0.873	<b>0.980</b>	<b>0.995</b>	<u>0.117</u>	0.401	<b>0.049</b>	0.147	<b>0.067</b>
A2U [10]	+30.0K	0.874	<u>0.979</u>	<b>0.995</b>	0.118	0.397	<b>0.049</b>	0.147	<u>0.068</u>
FADE [29]	+0.2M	0.874	0.978	<u>0.994</u>	0.118	0.399	<b>0.049</b>	0.147	0.071
SAPA-B	+0.1M	0.870	0.978	<b>0.995</b>	<u>0.117</u>	0.406	<u>0.050</u>	0.149	0.069
DySample-S	+5.8K	0.871	<u>0.979</u>	<b>0.995</b>	0.118	0.402	<u>0.050</u>	0.148	0.069
DySample-S+	+11.5K	0.872	0.978	<u>0.994</u>	0.119	0.398	<u>0.050</u>	0.148	0.070
DySample	+46.1K	0.872	<u>0.979</u>	<b>0.995</b>	<u>0.117</u>	0.400	<u>0.050</u>	0.147	<u>0.068</u>
DySample+	+92.2K	<b>0.878</b>	<b>0.980</b>	<b>0.995</b>	<b>0.116</b>	<b>0.393</b>	<b>0.049</b>	<b>0.145</b>	<u>0.068</u>

Table 9. Monocular depth estimation results with DepthFormer (Swin-T) on NYU Depth V2. Best performance is in **boldface** and second best is underlined.

### 4.3. Panoptic Segmentation

Panoptic segmentation is the joint task of semantic segmentation and instance segmentation. In this context, the upsamplers face the difficulty to discriminate instance boundaries, which places high demands on good semantic perception and discriminative ability of the upsamplers.

**Experimental Protocols.** We also conduct experiments on the MS COCO [25] data set and report the PQ, SQ, and RQ metrics [17]. We adopt Panoptic FPN [16] as the baseline and `mmdetection` as our codebase. The default training setting is used to ensure a fair comparison. We only modify the total three upsampling stages in FPN.

**Panoptic Segmentation Results.** The quantitative results shown in Table 8 demonstrate that DySample invites consistent performance gains, *i.e.*, 1.2 and 0.8 PQ improvement for R50 and R101 backbone respectively.

### 4.4. Monocular Depth Estimation

Monocular depth estimation requires a model to estimate a per-pixel depth map from a single image. A high-quality upsampler for depth estimation should simultaneously recover the details, maintain the consistency of the depth value in a plain region, and also tackles gradually changed depth values.

**Experimental Protocols.** We conduct the experiments on the NYU Depth V2 data set [35] and report the  $\delta < 1.25$ ,  $\delta < 1.25^2$  and  $\delta < 1.25^3$  accuracy, absolute relative error (Abs Rel), root mean squared error (RMS) and its log version (RMS(log)), average log10 error (log10), and squared relative error (Sq Rel). We adopt DepthFormer-SwinT [21] as the baseline including four upsampling stages in the fusion module. For reproducibility, we use the codebase

provided by monocular depth estimation toolbox [20] and follow its recommended training settings, while only modifying the upsamplers.

**Monocular Depth Estimation Results.** Quantitative results are shown in Table 9. Among all upsamplers, DySample+ achieves the best performance, with an increase of 0.05 in  $\delta < 1.25$  accuracy, a decrease of 0.04 in Abs Rel, and a decrease of 0.09 in RMS compared with bilinear upsampling. Further, the qualitative comparison in Fig. 9 row 5 also verifies the superiority of DySample, *e.g.*, the accurate, consistent depth map of the chair.

## 5. Conclusion

We propose DySample, a fast, effective, and universal dynamic upsampler. Different from common kernel based dynamic upsampling, DySample is designed from the perspective of point sampling. We start from a naive design and show how to gradually improve its performance from our deep insight of upsampling. Compared with other dynamic upsamplers, DySample not only reports the best performance but also gets rid of customised CUDA packages and consumes the least computational resources, showing superiority across latency, training memory, training time, GFLOPs, and number of parameters. For future work, we plan to apply DySample to low-level tasks and study joint modeling of upsampling and downsampling.

**Acknowledgement.** This work is supported by the National Natural Science Foundation of China under Grant No. 62106080.

## References

- [1] Shai Avidan and Ariel Shamir. Seam carving for content-aware image resizing. In *ACM SIGGRAPH 2007 papers*, pages 10–es. 2007.
- [2] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. SegNet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495, 2017.
- [3] Shariq Farooq Bhat, Ibraheem Alhashim, and Peter Wonka. Adabins: Depth estimation using adaptive bins. In *Proceedings of IEEE Conference on Computer Vision Pattern Recognition (CVPR)*, pages 4009–4018, 2021.
- [4] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *Proceedings of IEEE Conference on Computer Vision Pattern Recognition (CVPR)*, pages 6154–6162, 2018.
- [5] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, et al. Mmdetection: Open mmlab detection toolbox and benchmark. *arXiv Computer Research Repository*, 2019.
- [6] Bowen Cheng, Ross Girshick, Piotr Dollár, Alexander C Berg, and Alexander Kirillov. Boundary iou: Improving object-centric image segmentation evaluation. In *Proceedings of IEEE Conference on Computer Vision Pattern Recognition (CVPR)*, pages 15334–15342, 2021.
- [7] Bowen Cheng, Ishan Misra, Alexander G. Schwing, Alexander Kirillov, and Rohit Girdhar. Masked-attention mask transformer for universal image segmentation. In *Proceedings of IEEE Conference on Computer Vision Pattern Recognition (CVPR)*, 2022.
- [8] Bowen Cheng, Alex Schwing, and Alexander Kirillov. Per-pixel classification is not all you need for semantic segmentation. In *Proceedings of Annual Conference on Neural Information Processing Systems (NeurIPS)*, volume 34, 2021.
- [9] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, pages 764–773, 2017.
- [10] Yutong Dai, Hao Lu, and Chunhua Shen. Learning affinity-aware upsampling for deep image matting. In *Proceedings of IEEE Conference on Computer Vision Pattern Recognition (CVPR)*, pages 6841–6850, 2021.
- [11] Yuxin Fang, Shusheng Yang, Xinggang Wang, Yu Li, Chen Fang, Ying Shan, Bin Feng, and Wenyu Liu. Instances as queries. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, pages 6910–6919, 2021.
- [12] Muhammad Haris, Gregory Shakhnarovich, and Norimichi Ukita. Deep back-projection networks for super-resolution. In *Proceedings of IEEE Conference on Computer Vision Pattern Recognition (CVPR)*, pages 1664–1673, 2018.
- [13] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, pages 2961–2969, 2017.
- [14] Xu Jia, Bert De Brabandere, Tinne Tuytelaars, and Luc V Gool. Dynamic filter networks. In *Proceedings of Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 667–675, 2016.
- [15] Chen Jin, Ryutaro Tanno, Thomy Mertzanidou, Eleftheria Panagiotaki, and Daniel C Alexander. Learning to down-sample for segmentation of ultra-high resolution images. In *Proceedings of International Conference on Learning Representations*, 2022.
- [16] Alexander Kirillov, Ross Girshick, Kaiming He, and Piotr Dollár. Panoptic feature pyramid networks. In *Proceedings of IEEE Conference on Computer Vision Pattern Recognition (CVPR)*, pages 6399–6408, 2019.
- [17] Alexander Kirillov, Kaiming He, Ross Girshick, Carsten Rother, and Piotr Dollár. Panoptic segmentation. In *Proceedings of IEEE Conference on Computer Vision Pattern Recognition (CVPR)*, pages 9404–9413, 2019.
- [18] Jin Han Lee, Myung-Kyu Han, Dong Wook Ko, and Il Hong Suh. From big to small: Multi-scale local planar guidance for monocular depth estimation. *arXiv Computer Research Repository*, 2019.
- [19] Yanwei Li, Hengshuang Zhao, Xiaojuan Qi, Liwei Wang, Zeming Li, Jian Sun, and Jiaya Jia. Fully convolutional networks for panoptic segmentation. In *Proceedings of IEEE Conference on Computer Vision Pattern Recognition (CVPR)*, pages 214–223, 2021.
- [20] Zhenyu Li. Monocular depth estimation toolbox. <https://github.com/zhyever/Monocular-Depth-Estimation-Toolbox>, 2022.
- [21] Zhenyu Li, Zehui Chen, Xianming Liu, and Junjun Jiang. Depthformer: Exploiting long-range correlation and local information for accurate monocular depth estimation. *arXiv Computer Research Repository*, 2022.
- [22] Jingyun Liang, Jiezhong Cao, Guolei Sun, Kai Zhang, Luc Van Gool, and Radu Timofte. Swinir: Image restoration using swin transformer. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, pages 1833–1844, 2021.
- [23] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of IEEE Conference on Computer Vision Pattern Recognition (CVPR)*, pages 2117–2125, 2017.
- [24] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017.
- [25] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 740–755, 2014.
- [26] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, pages 10012–10022, 2021.
- [27] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *IEEE Transactions on*

- Pattern Analysis and Machine Intelligence*, 39(4):640–651, 2015.
- [28] Hao Lu, Yutong Dai, Chunhua Shen, and Songcen Xu. Indices matter: Learning to index for deep image matting. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, pages 3266–3275, 2019.
- [29] Hao Lu, Wenze Liu, Hongtao Fu, and Zhiguo Cao. FADE: Fusing the assets of decoder and encoder for task-agnostic upsampling. In *Proceedings of European Conference on Computer Vision (ECCV)*, 2022.
- [30] Hao Lu, Wenze Liu, Zixuan Ye, Hongtao Fu, Yuliang Liu, and Zhiguo Cao. SAPA: Similarity-aware point affiliation for feature upsampling. In *Proceedings of Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2022.
- [31] Yiqun Mei, Yuchen Fan, and Yuqian Zhou. Image super-resolution with non-local sparse attention. In *Proceedings of IEEE Conference on Computer Vision Pattern Recognition (CVPR)*, pages 3517–3526, 2021.
- [32] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 1(10), 2016.
- [33] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. *Proceedings of Annual Conference on Neural Information Processing Systems (NeurIPS)*, 28, 2015.
- [34] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of IEEE Conference on Computer Vision Pattern Recognition (CVPR)*, pages 1874–1883, 2016.
- [35] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from RGBD images. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 746–760, 2012.
- [36] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: Fully convolutional one-stage object detection. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, pages 9627–9636, 2019.
- [37] Jiaqi Wang, Kai Chen, Rui Xu, Ziwei Liu, Chen Change Loy, and Dahua Lin. CARAFE: Context-aware reassembly of features. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, pages 3007–3016, 2019.
- [38] Diana Wofk, Fangchang Ma, Tien-Ju Yang, Sertac Karaman, and Vivienne Sze. FastDepth: Fast monocular depth estimation on embedded systems. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2019.
- [39] Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun. Unified perceptual parsing for scene understanding. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 418–434, 2018.
- [40] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. In *Proceedings of Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2021.
- [41] Xuaner Zhang, Qifeng Chen, Ren Ng, and Vladlen Koltun. Zoom to learn, learn to zoom. In *Proceedings of IEEE Conference on Computer Vision Pattern Recognition (CVPR)*, pages 3762–3770, 2019.
- [42] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proceedings of IEEE Conference on Computer Vision Pattern Recognition (CVPR)*, 2017.
- [43] Xizhou Zhu, Han Hu, Stephen Lin, and Jifeng Dai. Deformable convnets v2: More deformable, better results. In *Proceedings of IEEE Conference on Computer Vision Pattern Recognition (CVPR)*, pages 9308–9316, 2019.
- [44] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable DETR: Deformable transformers for end-to-end object detection. In *Proceedings of International Conference on Learning Representations*, 2021.