CARAFE: Content-Aware ReAssembly of FEatures

Jiaqi Wang¹  Kai Chen¹  Rui Xu¹  Ziwei Liu¹  Chen Change Loy²  Dahua Lin¹
¹CUHK - SenseTime Joint Lab, The Chinese University of Hong Kong
²Nanyang Technological University

{wj017,ck015,xr018,dhlin}@ie.cuhk.edu.hk  zwliu.hust@gmail.com  ccloy@ntu.edu.sg

Abstract

Feature upsampling is a key operation in a number of modern convolutional network architectures, e.g. feature pyramids. Its design is critical for dense prediction tasks such as object detection and semantic/instance segmentation. In this work, we propose Content-Aware ReAssembly of FEatures (CARAFE), a universal, lightweight and highly effective operator to fulfill this goal. CARAFE has several appealing properties: (1) Large field of view. Unlike previous works (e.g. bilinear interpolation) that only exploit sub-pixel neighborhood, CARAFE can aggregate contextual information within a large receptive field. (2) Content-aware handling. Instead of using a fixed kernel for all samples (e.g. deconvolution), CARAFE enables instance-specific content-aware handling, which generates adaptive kernels on-the-fly. (3) Lightweight and fast to compute. CARAFE introduces little computational overhead and can be readily integrated into modern network architectures. We conduct comprehensive evaluations on standard benchmarks in object detection, instance/semantic segmentation and inpainting. CARAFE shows consistent and substantial gains across all the tasks (1.2% AP, 1.3% AP, 1.8% mIoU, 1.1dB respectively) with negligible computational overhead. It has great potential to serve as a strong building block for future research. Code and models are available at https://github.com/open-mmlab/mmdetection.

1. Introduction

Feature upsampling is one of the most fundamental operations in deep neural networks. On the one hand, for the decoders in dense prediction tasks (e.g. super resolution [6, 17], inpainting [11, 29] and semantic segmentation [39, 41]), the high-level/low-res feature map is upsampled to match the high-resolution supervision. On the other hand, feature upsampling is also involved in fusing a high-level/low-res feature map with a low-level/high-res feature map, which is widely adopted in many state-of-the-art architectures, e.g., Feature Pyramid Network [18], U-Net [31]...
number of parameters and thus heavy computational workload when a large kernel size is used. This makes it difficult to cover a larger region that goes beyond a small neighborhood, thus limiting its expressive power and performance.

In this work, we move beyond these limitations, and seek a feature upsampling operator that is capable of 1) aggregating information within large receptive field, 2) adapting to instance-specific contents on-the-fly, and 3) maintaining computation efficiency. To this end, we propose a lightweight yet highly effective operator, called Content-Aware ReAssembly of Features (CARAFE). Specifically, CARAFE reassembles the features inside a predefined region centered at each location via a weighted combination, where the weights are generated in a content-aware manner. Furthermore, there are multiple groups of such upsampling weights for each location. Feature upsampling is then accomplished by rearranging the generated features as a spatial block.

Note that these spatially adaptive weights are not learned as network parameters. Instead, they are predicted on-the-fly, using a lightweight fully-convolutional module with softmax activation. Figure 1 reveals the working mechanism of CARAFE. After upsampled by CARAFE, a feature map can represent the shape of an object more accurately, so that the model can predict better instance segmentation results. Our CARAFE not only upsamples the feature map spatially, but also learns to enhance its discrimination.

To demonstrate the universal effectiveness of CARAFE, we conduct comprehensive evaluations across a wide range of dense prediction tasks, i.e., object detection, instance segmentation, semantic segmentation, image inpainting, with mainstream architectures. CARAFE can boost the performance of Faster RCNN [30] by 1.2% AP in object detection and Mask RCNN [8] by 1.3% AP in instance segmentation on MS COCO [19] test-dev 2018. CARAFE further improves UperNet [35] by 1.8% mIoU on ADE20k [43, 44] in semantic segmentation, and improves Global&Local [11] by 1.1 dB of PSNR on Places [42] in image inpainting. When upsampling an $H \times W$ feature map with 256 channels by a factor of two, the introduced computational overhead by CARAFE is only $H \times W \times 199k$ FLOPs, vs., $H \times W \times 1180k$ FLOPs of deconvolution. The substantial gains on all the tasks demonstrate that CARAFE is an effective and efficient feature upsampling operator that has great potential to serve as a strong building block for future research.

2. Related Work

Upsampling Operators. The most commonly used upsampling methods are nearest neighbor and bilinear interpolations. These interpolations leverage distances to measure the correlations between pixels, and hand-crafted upsampling kernels are used in them. In deep learning era, several methods are proposed to upsample a feature map using learnable operators. For example, deconvolution [27], which is an inverse operator of a convolution, is the most famous among those learnable upsamplers. Pixel Shuffle [32] proposes a different upsampler which reshapes depth on the channel space into width and height on the spatial space. Recently, [23] proposed guided upsampling (GUM), which performs interpolation by sampling pixels with learnable offsets. However, these methods either exploit contextual information in a small neighborhood, or require expensive computation to perform adaptive interpolation. Within the realms of super-resolution and denoising, some other works [24, 14, 9] also explore using learnable kernels spatially in low-level vision. With a similar design spirit, here we demonstrate the effectiveness and working mechanism of content-aware feature reassembly for upsampling in several visual perception tasks, and provide a lightweight solution.

Dense Prediction Tasks. Object detection is the task of localizing objects with bounding-boxes, instance segmentation further requires the prediction of instance-wise masks. Faster-RCNN [30] introduces Region Proposal Network (RPN) for end-to-end training, which is further improved by the guided anchoring scheme [34]. [18, 21, 15, 41, 28] exploits multi-scale feature pyramids to deal with objects at different scales. By adding extra mask prediction branches, Mask-RCNN [8] and its variants [1, 10] yield promising pixel-level results. Semantic segmentation [22, 16] requires pixel-wise semantic prediction for given images. PSPNet [39] introduces spatial pooling at multiple grid scales, and UperNet [35] designs a more generalized framework based on PSPNet. Image or Video inpainting [38, 37, 36] is a classical problem to fill in the missing regions of the input pictures. U-net [31] is popular among recent works [11, 33], and adopts multiple upsampling operators. Liu et al. [20] introduce partial convolution layer to alleviate the influence of missing regions on the convolution layers. Our CARAFE demonstrates universal effectiveness across a wide range of dense prediction tasks.

3. Content-Aware ReAssembly of Features

Feature upsampling is a key operator in many modern convolutional network architectures developed for tasks including object detection, instance segmentation, and scene parsing. In this work, we propose the content-aware reassembly of features (CARAFE) to upsample a feature map. On each location, CARAFE can leverage the underlying content information to predict reassembly kernels and reassemble the features inside a predefined nearby region. Thanks to the content information, CARAFE can use an adaptive and optimized reassembly kernel in different locations and achieve better performance than mainstream upsampling operators, e.g. interpolations or deconvolution.
Kernel Prediction Module

3.1. Formulation

CARAFE works as a reassembly operator with content-aware kernels. It consists of two steps. The first step is to predict a reassembly kernel for each target location according to its content, and the second step is to reassemble the features with predicted kernels. Given a feature map \( X \) of size \( C \times H \times W \) and an upsample ratio \( \sigma \) (supposing \( \sigma \) is an integer), CARAFE will produce a new feature map \( X' \) of size \( C \times \sigma H \times \sigma W \). For any target location \( l' = (i', j') \) of the output \( X' \), there is a corresponding source location \( l = (i, j) \) at the input \( X \), where \( i = \lfloor i'/\sigma \rfloor \) and \( j = \lfloor j'/\sigma \rfloor \).

Here we denote \( N(X_l, k) \) as the \( k \times k \) sub-region of \( X \) centered at the location \( l \), i.e., the neighbor of \( X_l \).

In the first step, the kernel prediction module \( \psi \) predicts a location-wise kernel \( W_{l'} \) for each location \( l' \), based on the neighbor of \( X_l \), as shown in Eqn. (1). The reassembly step is formulated as Eqn. (2), where \( \phi \) is the content-aware reassembly module that reassembles the neighbor of \( X_l \) with the kernel \( W_{l'} \):

\[
W_{l'} = \psi(N(X_l, k_{\text{encoder}})). \quad (1)
\]

\[
X_{l'} = \phi(N(X_l, k_{\text{up}}), W_{l'}). \quad (2)
\]

We specify the details of \( \psi \) and \( \phi \) in the following parts.

3.2. Kernel Prediction Module

The kernel prediction module is responsible for generating the reassembly kernels in a content-aware manner. Each source location on \( X \) corresponds to \( \sigma^2 \) target locations on \( X' \). Each target location requires a \( k_{\text{up}} \times k_{\text{up}} \) reassembly kernel, where \( k_{\text{up}} \) is the reassembly kernel size. Therefore, this module will output the reassembly kernels of size \( C_{\text{up}} \times H \times W \), where \( C_{\text{up}} = \sigma^2 k_{\text{up}}^2 \).

The kernel prediction module is composed of three sub-modules, i.e., channel compressor, content encoder and kernel normalizer, as shown in Figure 2. The channel compressor reduces the channel of the input feature map. The content encoder then takes the compressed feature map as input and encodes the content to generate reassembly kernels. Lastly, the kernel normalizer applies a softmax function to each reassembly kernel. The three submodules are explained in detail as follows.

Channel Compressor. We adopt a \( 1 \times 1 \) convolution layer to compress the input feature channel from \( C \) to \( C_{m} \). Reducing the channel of input feature map leads to less parameters and computational cost in the following steps, making CARAFE more efficient. It is also possible to use larger kernel sizes for the content encoder under the same budget. Experimental results show that reducing the feature channel in an acceptable range will not harm the performance.

Content Encoder. We use a convolution layer of kernel size \( k_{\text{encoder}} \) to generate reassembly kernels based on the content of input features. The parameters of the encoder is \( k_{\text{encoder}} \times k_{\text{encoder}} \times C_{m} \times C_{\text{up}} \). Intuitively, increasing \( k_{\text{encoder}} \) can enlarge the receptive field of the encoder, and exploits the contextual information within a larger region, which is important for predicting the reassembly kernels. However, the computational complexity grows with the square of the kernel size, while the benefits from a larger kernel size do not. An empirical formula \( k_{\text{encoder}} = k_{\text{up}} - 2 \) is a good trade-off between performance and efficiency through our study in Section 5.3.

Kernel Normalizer. Before being applied to the input fea-
ture map, each $k_{up} \times k_{up}$ reassembly kernel is normalized with a softmax function spatially. The normalization step forces the sum of kernel values to 1, which is a soft selection across a local region. Due to the kernel normalizer, CARAFE does not perform any rescaling and change the mean values of the feature map, that is why our proposed operator is named the reassembly of features.

3.3. Content-aware Reassembly Module

With each reassembly kernel $W_l$, the content-aware reassembly module will reassemble the features within a local region via the function $\phi$. We adopt a simple form of $\phi$ which is just a weighted sum operator. For a target location $l'$ and the corresponding square region $N(X_l, k_{up})$ centered at $l = (i, j)$, the reassembly is shown in Eqn. (3), where $r = \lfloor k_{up}/2 \rfloor$:

$$X_l' = \sum_{n=-r}^{r} \sum_{m=-r}^{r} W_l(n, m) \cdot X_{l+i+n, j+m}. \quad (3)$$

With the reassembly kernel, each pixel in the region of $N(X_l, k_{up})$ contributes to the upsampled pixel $l'$ differently, based on the content of features instead of distance of locations. The semantics of the reassembled feature map can be stronger than the original one, since the information from relevant points in a local region can be more attended.

3.4. Relation to Previous Operators

Here we discuss the relations between CARAFE and dynamic filter [13], spatial attention [3], spatial transformer [12] and deformable convolution [5], which share similar design philosophy but with different focuses.

**Dynamic Filter.** Dynamic filter generates instance-specific convolutional filters conditioned on the input of the network, and then applies the predicted filter on the input. Both dynamic filter and CARAFE are content-aware operators, but a fundamental difference between them lies at their kernel generation process. Specifically, dynamic filter works as a two-step convolution, where the additional filter prediction layer and filtering layer require heavy computation. On the contrary, CARAFE is simply a reassembly of features in local regions, without learning the feature transformation across channels. Supposing the channels of input feature map is $C$ and kernel size of the filter is $K$, then the predicted kernel parameters for each location is $C \times C \times K \times K$ in dynamic filter. For CARAFE, the kernel parameter is only $K \times K$. Thus, it is more efficient in memory and speed.

**Spatial Attention.** Spatial attention predicts an attention map with the same size as the input feature, and then rescales the feature map on each location. Our CARAFE reassembles the features in a local region by weighted sum. In summary, spatial attention is a rescaling operator with point-wise guidance while CARAFE is a reassembly operator with region-wise local guidance. Spatial attention can be seen as a special case of CARAFE where the reassembly kernel size is 1, regardless of the kernel normalizer.

**Spatial Transformer Networks (STN).** STN predicts a global parametric transformation conditioned on the input feature map and warps the feature via the transformation. However, this global parametric transformation assumption is too strong to represent complex spatial variance; and STN is known to be hard to train. Here, CARAFE uses the location-specific reassembly to handle the spatial relations, which enables more flexible local geometry modeling.

**Deformable Convolutional Networks (DCN).** DCN also adopts the idea of learning geometric transformation and combines it with the regular convolution layers. It predicts kernel offsets other than using grid convolution kernels. Similar to dynamic filter, it is also a heavy parametric operator with 24 times more computational cost than CARAFE. It is also known to be sensitive to parameter initialization.

4. Applications of CARAFE

CARAFE can be seamlessly integrated into existing frameworks where upsampling operators are needed. Here we present some applications in mainstream dense prediction tasks. With negligible additional parameters, CARAFE benefits state-of-the-art methods in both high-level and low-level tasks, such as object detection, instance segmentation, semantic segmentation and image inpainting.

4.1. Object Detection and Instance Segmentation

Feature Pyramid Network (FPN) is an important and effective architecture in the field of object detection and instance segmentation. It significantly improves the performance of popular frameworks like Faster R-CNN and Mask R-CNN. FPN constructs feature pyramids of strong semantics with the top-down pathway and lateral connections. In the top-down pathway, a low-resolution feature map is firstly upsampled by 2x with the nearest neighbor interpolation and then fused with a high-resolution one, as shown in Figure 3.

We propose to substitute the nearest neighbor interpolation in all the feature levels with CARAFE. This modification is smooth and no extra change is required. In addition to the FPN structure, Mask R-CNN adopts a deconvolution layer at the end of mask head. It is used to upsample the predicted digits from $14 \times 14$ to $28 \times 28$, to obtain finer mask predictions. We can also use CARAFE to replace the deconvolution layer, resulting in even less computational cost.

4.2. Semantic Segmentation

Semantic segmentation requires the model to output per-pixel level predictions on the whole image, so that high-resolution feature maps are usually preferred. Upsampling is widely adopted to enlarge feature maps and fuse the se-
semantic information of different levels in this task. UperNet is a strong baseline for semantic segmentation. It uses upsampling in the following three components, *i.e.*, PPM, FPN, FUSE. We adopt CARAFE instead of their original upsamplers.

**Pyramid Pooling Module (PPM).** PPM is the key component in PSPNet that hierarchically down-samples an input feature map into multiple scales \( \{1 \times 1, 2 \times 2, 3 \times 3, 6 \times 6\} \), and then upsamples them back to the original sizes with bilinear interpolation. The features are finally fused with the original feature by concatenation. Since the upsampling ratio is very large, we adopt a two-step strategy with CARAFE as a trade-off between performance and efficiency. Firstly we upsample the \( \{1 \times 1, 2 \times 2, 3 \times 3, 6 \times 6\} \) features to half the size of the original feature map with bilinear interpolation, and then use CARAFE to further upsample them by 2x.

**Feature Pyramid Network (FPN).** Similar to detection models, UperNet also adopts FPN to enrich the feature semantics. It only has four different feature levels \( \{P2, P3, P4, P5\} \) with strides \( \{4, 8, 16, 32\} \). We replace the upsampling operators in the same way as Section 4.1.

**Multi-level Feature Fusion (FUSE).** UperNet proposes a multi-level feature fusion module after the FPN. It upsamples P3, P4, P5 to the same size as P2 by bilinear interpolation and then fuses these features from different levels by concatenation. The process is equivalent to a sequential upsampling-concatenation that first upsamples P5 to P4 and concatenates them, and then upsamples the concatenated feature map to P3 and so on. We replace the sequential bilinear upsampling here with CARAFE.

### 4.3. Image Inpainting

The U-net architecture is popular among recent proposed image inpainting methods, such as Global&Local [11] and Partial Conv [20]. There are two upsampling operators in the second half of the network. We simply replace the two upsampling layers with CARAFE and evaluate the performance. As for Partial Conv, we can conveniently keep the mask propagation in CARAFE by updating the mask with our content-aware reassembly kernels.

### 5. Experiments

#### 5.1. Experimental Settings

**Datasets & Evaluation Metrics.** We evaluate CARAFE on several important dense prediction benchmarks. We use the train split for training and evaluate the performance on the val split for all these datasets by default.

**Object Detection and Instance Segmentation.** We perform experiments on the challenging MS COCO 2017 dataset. Results are evaluated with the standard COCO metric, *i.e.* mAP of IoUs from 0.5 to 0.95.

**Semantic Segmentation.** We adopt the ADE20k benchmark to evaluate our method in the semantic segmentation task. Results are measured with mean IoU (mIoU) and Pixel Accuracy (P.A.), which respectively indicates the average IoU between predictions and ground truth masks and per-pixel classification accuracy.

**Image Inpainting.** Places dataset is adopted for image inpainting. We use L1 error (lower is better) and PSNR (higher is better) as evaluation metrics.

**Implementation Details.** If not otherwise specified, CARAFE adopts a fixed set of hyper-parameters in experiments, where \( C_m = 64 \) for the channel compressor and \( k_{encoder} = 3, k_{up} = 5 \) for the content encoder. See more implementation details in supplementary materials.

**Object Detection and Instance Segmentation.** We evaluate CARAFE on Faster RCNN and Mask RCNN with the ResNet-50 w/ FPN backbone, and follow the 1x training schedule settings as Detectron [7] and MMDetection [2].

**Semantic Segmentation.** We use the official implementation of UperNet and adopt the same experiment settings.

**Image Inpainting** We adopt Global&Local [11] and Partial Conv as baseline methods to evaluate CARAFE.

#### 5.2. Benchmarking Results

**Object Detection & Instance Segmentation.** We first evaluate our method by substituting the nearest neighbor interpolation in FPN with CARAFE for both Faster RCNN and Mask RCNN, and the deconvolution layer in the mask head for Mask RCNN. As shown in Table 1, CARAFE improves Faster RCNN by 1.2% on bbox AP, and Mask RCNN by 1.3% on mask AP. The improvements of \( \text{AP}_S, \text{AP}_M, \text{AP}_L \) are all above 1% AP, which suggests that it is beneficial for various object scales.

\[ \text{https://github.com/CSAILVision/semantic-segmentation-pytorch} \]
Table 1: Detection and Instance Segmentation results on MS COCO 2018 test-dev.

<table>
<thead>
<tr>
<th>Method</th>
<th>Backbone</th>
<th>Task</th>
<th>AP</th>
<th>AP50</th>
<th>AP75</th>
<th>APs</th>
<th>APm</th>
<th>AL</th>
<th>FLOPs</th>
<th>Params</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faster R-CNN</td>
<td>ResNet-50</td>
<td>BBox</td>
<td>36.9</td>
<td>59.1</td>
<td>39.7</td>
<td>21.5</td>
<td>40.0</td>
<td>45.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Faster R-CNN w/ CARAFE</td>
<td>ResNet-50</td>
<td>BBox</td>
<td><strong>38.1</strong></td>
<td><strong>60.7</strong></td>
<td><strong>41.0</strong></td>
<td><strong>22.8</strong></td>
<td><strong>41.2</strong></td>
<td><strong>46.9</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mask R-CNN</td>
<td>ResNet-50</td>
<td>BBox</td>
<td>37.8</td>
<td>59.7</td>
<td>40.8</td>
<td>22.2</td>
<td>40.7</td>
<td>46.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mask R-CNN w/ CARAFE</td>
<td>ResNet-50</td>
<td>BBox</td>
<td><strong>38.8</strong></td>
<td><strong>61.2</strong></td>
<td><strong>42.1</strong></td>
<td><strong>23.2</strong></td>
<td><strong>41.7</strong></td>
<td><strong>47.9</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Detection results with Faster R-CNN. Various upsampling methods are used in FPN. N.C., B.C., P.S., and S.A. indicate Nearest + Conv, Bilinear + Conv, Pixel Shuffle and Spatial Attention, respectively.

<table>
<thead>
<tr>
<th>Method</th>
<th>AP</th>
<th>AP50</th>
<th>AP75</th>
<th>APs</th>
<th>APm</th>
<th>AP_L</th>
<th>FLOPs</th>
<th>Params</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nearest</td>
<td>36.5</td>
<td>58.4</td>
<td>39.3</td>
<td>21.3</td>
<td>40.3</td>
<td>47.2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bilinear</td>
<td>36.7</td>
<td>58.7</td>
<td>39.7</td>
<td>21.0</td>
<td>40.5</td>
<td>47.5</td>
<td>8k</td>
<td>0</td>
</tr>
<tr>
<td>N.C.</td>
<td>36.6</td>
<td>58.6</td>
<td>39.5</td>
<td>21.4</td>
<td>40.3</td>
<td>46.4</td>
<td>4.7M</td>
<td>590k</td>
</tr>
<tr>
<td>B.C.</td>
<td>36.6</td>
<td>58.7</td>
<td>39.4</td>
<td>21.6</td>
<td>40.6</td>
<td>46.8</td>
<td>4.7M</td>
<td>590k</td>
</tr>
<tr>
<td>Deconv [27]</td>
<td>36.4</td>
<td>58.2</td>
<td>39.2</td>
<td>21.3</td>
<td>39.9</td>
<td>46.5</td>
<td>1.2M</td>
<td>590k</td>
</tr>
<tr>
<td>P.S.[32]</td>
<td>36.5</td>
<td>58.8</td>
<td>39.1</td>
<td>20.9</td>
<td>40.4</td>
<td>46.7</td>
<td>4.7M</td>
<td>2.4M</td>
</tr>
<tr>
<td>GUM[23]</td>
<td>36.9</td>
<td>58.9</td>
<td>39.7</td>
<td>21.5</td>
<td>40.6</td>
<td>48.1</td>
<td>1.1M</td>
<td>132k</td>
</tr>
<tr>
<td>S.A.[3]</td>
<td>36.9</td>
<td>58.8</td>
<td>39.8</td>
<td>21.7</td>
<td>40.8</td>
<td>47.0</td>
<td>28k</td>
<td>2.3k</td>
</tr>
<tr>
<td>CARAFE</td>
<td><strong>37.8</strong></td>
<td><strong>60.1</strong></td>
<td><strong>40.8</strong></td>
<td><strong>23.1</strong></td>
<td><strong>41.7</strong></td>
<td><strong>48.5</strong></td>
<td>199k</td>
<td>74k</td>
</tr>
</tbody>
</table>

Our encouraging performance is supported by the qualitative results as shown in Figure 1. We visualize the feature maps in the top-down pathway of FPN and compare CARAFE with the baseline, i.e., nearest neighbor interpolation. It is obvious that with the content-aware reassembly, the feature maps are more discriminative and a more accurate mask for the object is predicted. In Figure 4, we show some examples of instance segmentation results comparing the baseline and CARAFE.

To investigate the effectiveness of different upsampling operators, we perform extensive experiments on Faster R-CNN by using different operators to perform upsampling in FPN. Results are illustrated in Table 2. For ‘N.C.’ and ‘B.C.’, which respectively indicate ‘Nearest + Conv’ and ‘Bilinear + Conv’, we add an extra 3 × 3 convolution layer after the corresponding upsampling. ‘Deconv’, ‘Pixel Shuffle’ (indicated as ‘P.S.’), ‘GUM’ are three representative learning based upsampling methods. We also compare ‘Spatial Attention’ here, indicated as ‘S.A.’. CARAFE achieves the best AP among all these upsampling operators, the FLOPs and parameters are relatively small, which proves it is both effective and efficient. The results of ‘Nearest + Conv’ and ‘Bilinear + Conv’ show that extra parameters do not lead to a significant gain. ‘Deconv’, ‘Pixel Shuffle’, ‘GUM’ and ‘Spatial Attention’ obtain inferior performance to CARAFE, indicating that the design of effective upsampling operators is critical.

Besides FPN which is a pyramid feature fusion structure, we also explore different upsampling operators in the mask head. In typical Mask R-CNN, a deconvolution layer is adopted to upsample the RoI features by 2x. For a fair comparison, we do not make any changes to FPN, and only replace the deconvolution layer with various operators. Since we only modify the mask prediction branch, performance is reported in terms of mask AP, as shown in Table 3. CARAFE achieves the best performance in instance segmentation among these methods.

In Table 4, we report the object detection and instance segmentation results of adopting CARAFE in FPN and mask head on Mask R-CNN respectively. Consistent improvements are achieved in these experiments.

**Semantic Segmentation.** We replace the upsamplers in UperNet with CARAFE and evaluate the results on ADE20k benchmark. As shown in Table 5, CARAFE improves the mIoU by a large margin from 40.44% to 42.23% with single scale testing. Note that UperNet with
CARAFE also achieves better performance than recent strong baselines such as PSPNet[39] and PSANet[40].

We perform a step-by-step study to inspect the effectiveness of modifying different components in UperNet, as described in Section 4.2. Results in Table 6 show that CARAFE is helpful for all the three components and the combination of them results in further gains.

Image Inpainting. We show that CARAFE is also effective in low-level tasks such as image inpainting. By replacing the upsampling operators with CARAFE in two strong baselines Global&Local [11] and Partial Conv [20], we observe significant improvements for both methods. As shown in Table 7, our method improves two baselines by 1.1 dB and 0.2 dB on the PSNR metric.

5.3. Ablation Study & Further Analysis

Model Design & Hyper-parameters. We investigate the influence of hyper-parameters in the model design, i.e., the compressed channels $C_m$, encoder kernel size $k_{encoder}$ and reassembly kernel size $k_{up}$. We also test different normalization methods in the kernel normalizer. We perform the ablation study of the designs and settings on Faster R-CNN with a ResNet-50 backbone, and evaluate the results on COCO 2017 val.

Towards an efficient design, we first analyze the computational complexity measured by FLOPs. When upsampling the feature map with input channel $C_{in}$ by a factor of $\sigma$, the per pixel FLOPs of CARAFE is computed as

$2(C_{in} + 1)C_m + 2(C_m k_{encoder}^2 + 1)\sigma^2k_{up}^2 + 2\sigma^2k_{up}^2C_{in}$

referring to [25].

We experiment with different values of $C_m$ in the channel compressor. In addition, we also try removing the channel compressor module, which means the content encoder directly uses input features to predict reassembly kernels. Experimental results in Table 8 show that compress $C_m$ down to 64 leads to no performance decline, while being more efficient. A further smaller $C_m$ will result in a slight drop of the performance. With no channel compressor, it can achieve the same performance, which proves that the channel compressor can speed up the kernel prediction without harming the performance. Based on the above results, we set $C_m$ to 64 by default as a trade-off between performance and efficiency.

We then investigate the influence of $k_{encoder}$ and $k_{up}$. Intuitively, increasing $k_{up}$ also requires a larger $k_{encoder}$, since the content encoder needs a large receptive field to predict a large reassembly kernel. As illustrated in Table 9, increasing $k_{encoder}$ and $k_{up}$ at the same time can boost the performance, while just enlarging one of them will not. We summarize an empirical formula that $k_{encoder} = k_{up} - 2$. We report the performance in model zoo of the official implementation.
Table 8: Ablation study of various compressed channels $C_m$. N/A means channel compressor is removed.

<table>
<thead>
<tr>
<th>$C_m$</th>
<th>AP</th>
<th>AP$_{50}$</th>
<th>AP$_{75}$</th>
<th>AP$_S$</th>
<th>AP$_M$</th>
<th>AP$_L$</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>37.6</td>
<td>60.1</td>
<td>40.6</td>
<td>22.7</td>
<td>41.6</td>
<td>48.4</td>
</tr>
<tr>
<td>32</td>
<td>37.7</td>
<td>60.3</td>
<td>40.7</td>
<td>22.8</td>
<td>41.2</td>
<td>49.0</td>
</tr>
<tr>
<td>64</td>
<td>37.8</td>
<td>60.1</td>
<td>40.8</td>
<td>23.1</td>
<td>41.7</td>
<td>48.5</td>
</tr>
<tr>
<td>128</td>
<td>37.8</td>
<td>60.1</td>
<td>40.8</td>
<td>22.4</td>
<td>41.7</td>
<td>48.7</td>
</tr>
<tr>
<td>256</td>
<td>37.8</td>
<td>60.4</td>
<td>40.8</td>
<td>22.7</td>
<td>41.3</td>
<td>48.8</td>
</tr>
<tr>
<td>N/A</td>
<td>37.8</td>
<td>60.3</td>
<td>40.8</td>
<td>22.9</td>
<td>41.5</td>
<td>48.7</td>
</tr>
</tbody>
</table>

Table 9: Detection results with various encoder kernel size $k_{encoder}$ and reassembly kernel size $k_{up}$.

<table>
<thead>
<tr>
<th>$k_{encoder}$</th>
<th>$k_{up}$</th>
<th>AP</th>
<th>AP$_{50}$</th>
<th>AP$_{75}$</th>
<th>AP$_S$</th>
<th>AP$_M$</th>
<th>AP$_L$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>37.3</td>
<td>59.6</td>
<td>40.5</td>
<td>22.0</td>
<td>40.7</td>
<td>48.1</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>37.3</td>
<td>59.9</td>
<td>40.0</td>
<td>22.3</td>
<td>41.1</td>
<td>47.3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>37.3</td>
<td>59.7</td>
<td>40.4</td>
<td>22.1</td>
<td>40.8</td>
<td>48.3</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>37.8</td>
<td>60.1</td>
<td>40.8</td>
<td>23.1</td>
<td>41.7</td>
<td>48.5</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>37.7</td>
<td>60.0</td>
<td>40.9</td>
<td>23.0</td>
<td>41.5</td>
<td>48.4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>37.8</td>
<td>60.2</td>
<td>40.7</td>
<td>22.5</td>
<td>41.4</td>
<td>48.6</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>38.1</td>
<td>60.4</td>
<td>41.3</td>
<td>23.0</td>
<td>41.6</td>
<td>48.8</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>38.0</td>
<td>60.2</td>
<td>41.1</td>
<td>23.0</td>
<td>41.8</td>
<td>48.8</td>
</tr>
</tbody>
</table>

which is a good choice in all the settings. Though adopting larger kernel size is shown helpful, we set $k_{up} = 5$ and $k_{encoder} = 3$ by default as a trade-off between performance and efficiency.

Other than the softmax function, we also test other alternatives in the kernel normalizer, such as sigmoid or sigmoid with normalization. As shown in Table 10, ‘Softmax’ and ‘Sigmoid Normalized’ have the same performance and better than ‘Sigmoid’, which shows that it is crucial to normalize the reassembly kernel to be summed to 1.

**How CARAFE Works.** We conduct further qualitative study to figure out how CARAFE works. With a trained Mask RCNN model adopting CARAFE as the upsampling operator, we visualize the reassembling process in Figure 5. In the FPN structure, the low-resolution feature map will be consecutively upsampled for several times to a higher resolution, so a pixel in the upsampled feature map reassembles information from a more larger region. We sample some pixels in the high-resolution feature map, and see which neighbors it is reassembled from. The green circle denotes example locations and red dots indicates highly weighted sources during the reassembly. From the figure, we can clearly learn that CARAFE is content-aware. It tends to reassemble points with similar semantic information. A location at human body prefers other points from the same human, rather than other objects or nearby background. For locations in the background regions which has weaker semantics, the reassembly is more uniform or just biased on points with similar low-level texture features.

**6. Conclusion**

We have presented Content-Aware ReAssembly of FEatures (CARAFE), a universal, lightweight and highly effective upsampling operator. It consistently boosts the performances on standard benchmarks in object detection, instance/semantic segmentation and inpainting by 1.2% AP, 1.3% AP, 1.8% mIoU, 1.1dB, respectively. More importantly, CARAFE introduces little computational overhead and can be readily integrated into modern network architectures. Future directions include exploring the applicability of CARAFE in low-level vision tasks such as image restoration and super-resolution.

**Acknowledgements.** This work is partially supported by the Collaborative Research Grant from SenseTime Group (CUHK Agreement No. TS1610626 & No. TS1712093), the General Research Fund (GRF) of Hong Kong (No. 14236516 & No. 14203518), Singapore MOE AcRF Tier 1 (M4012082.020), NTU SUG, and NTU NAP.
References


Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through...