FADE: Fusing the Assets of Decoder and Encoder for Task-Agnostic Upsampling Supplementary Material

Hao Lu, Wenze Liu, Hongtao Fu, and Zhiguo Cao*

School of Artificial Intelligence and Automation,
Huazhong University of Science and Technology, Wuhan 430074, China
{hlu,wzliu,htfu,zgcao}@hust.edu.cn

We provide the following contents in this supplementary:

- Visualization of upsampled features between FADE and CARAFE;
- Pseudo-code of semi-shift convolution;
- Implementation details of segmentation on the Weizmann Horse data set;
- Illustration on how FADE is incorporated into SegFormer;
- Additional visualizations of semantic segmentation on ADE20K and image matting on Adobe Composition-1K.

S1 Visualization of Upsampled Features

We visualize the upsampled feature maps w.r.t. CARAFE and FADE in SegFormer. We select one checkpoint for every 100 iterations in the range from the 100-th to 3000-th iteration. We also highlight the 320-th, 330-th, 340-th, 350-th, and 360-th iteration, because we observe fast variances of the feature maps during this period. We compute the average response along the channel dimension and normalize it to $[0, 255]$. From Figures S2 and S1, we can see that the two upsampling operators have different behaviors: FADE first learns to delineate the outlines of objects and then gradually fills the interior regions, while CARAFE focuses on the interior initially and then spreads outside slowly. We think the reason is that the gating mechanism is relatively simple and learns fast. By the way, one can see that there is 'checkerboard artifacts' in the visualizations of CARAFE due to the adoption of Pixel Shuffle.

S2 Semi-shift Convolution

The PyTorch-style implementation of semi-shift convolution is illustrated in Algorithm 1. The definitions of used parameters are kept same to that in the main text.

* Corresponding author

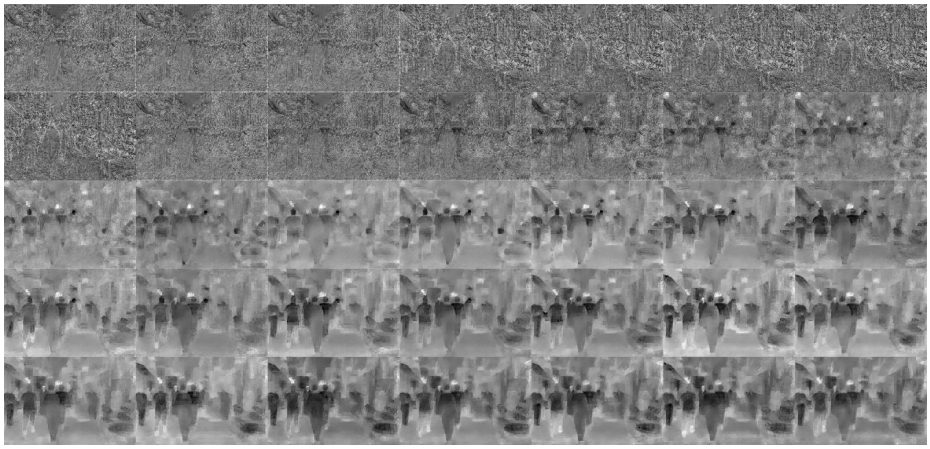

Figure S1: Feature maps upsampled by FADE with increased training iterations. From left to right, from top to bottom, FADE first learns to delineate the outlines of objects and then gradually fills the interior regions.

Figure S2: Feature maps upsampled by CARAFE with increased training iterations. From left to right, from top to bottom, CARAFE starts from the interior and then spreads outside.

S3 Implementation Details for Segmentation on the Weizmann Horse Data Set

There are 328 images with ground truth masks in the Weizimann Horse data set, in which we randomly choose 85% images as the training set, and the rest as the testing set. SegNet pretrained on ImageNet is used as the basic architecture and we only modify its upsampling operator during experiments. Images are resized

Algorithm 1: Semi-shift Convolution

```
def SpaceReassemble(x, B, C, H, W, scale = 2)
    # x (input): a tensor formed by 4 channelwise concatenated sub-tensors
    # y (output): spatially reassembled output
    # B: batch size
    # C: output number of channels
    # H, W: height and width of decoder feature map
    # scale: scaling factor of upsampling
    y = x.permute(0, 2, 3, 1).contiguous().view(
              0, 1, 3, 2, 4, 5).contiguous().view(
                B, scale * H, scale * W, C).permute(0, 3, 1, 2).contiguous()
    return y

def SemiShiftConv(en, de, alpha_en, alpha_de, a_de, beta, b, d = 64, K = 5, scale = 2)
    # en, de (input): encoder and decoderfeature map
    # kernels (output): upsampling kernels
    # alpha_en, {alpha_de,a_de}: params of channel compressor
    # beta,b: params of content encoder
    # d: compressed dimension
    # K: upsampling kernel size
    # scale: scaling factor of upsampling
    B, C, H, W = x.shape
    # F implies torch.nn.functional
    # en_c means compressed encoder feature.
    en_c = F.conv2d(en, weight = alpha_en, bias = None, stride = 1)
    de_c = F.conv2d(de, weight = alpha_de, bias = a_de, stride = 1)
    pad_en = []
    pad_en.append(F.pad(en_c, pad = [1, 0, 1, 0])) # padding top-left borders
    pad_en.append(F.pad(en_c, pad = [0, 1, 1, 0])) # padding top-right borders
    pad_en.append(F.pad(en_c, pad = [1, 0, 0, 1])) # padding bottom-left borders
    pad_en.append(F.pad(en_c, pad = [0, 1, 0, 1])) # padding bottom-right borders
    # To apply the same convolution to the 4 sub-tensors, we first concatenate the
    # 4 tensors at the channel dimension and then move it to the batch dimension.
    pad_en = torch.cat(pad_en, dim = 1).view(scale * *2 * B, d, scale * H + 1, scale * W + 1)
    kernels = F.conv2d(pad_en, weight = beta, bias = b, stride = 2).view(
                B, scale * *2, K * *2, H, W) + \
                  F.conv2d(de_c, weight = beta, bias = b, stride = 1, padding = 1).unsqueeze(1)
    # Reassemble the 4 sub-outputs
    kernels = kernels.view(B, scale * *2 * K * *2, H, W)
    kernels = SpaceReassemble(kernels, B, K * *2, H, W)
    kernels = F.softmax(kernels, dim = 1)
    return kernels
```

to 224×224. We use the cross entropy loss. The batch size is set to 4. We use SGD with a momentum of 0.9 as the optimizer. We set the initial learning rate as 0.01 and decay the rate at the 35-th and 45-th epoch to 0.001 and 0.0001, respectively.

For visualization, we output the gradient and the feature maps from an intermediate decoding layer (NOT the last layer). For the gradient maps, considering that there exist positive or negative values among different regions, we pass them through a ReLU function first and then sum all the channels. For gated feature maps, we select some representative ones from the channels. Both maps are normalized to $[0, 255]$ for visualization.

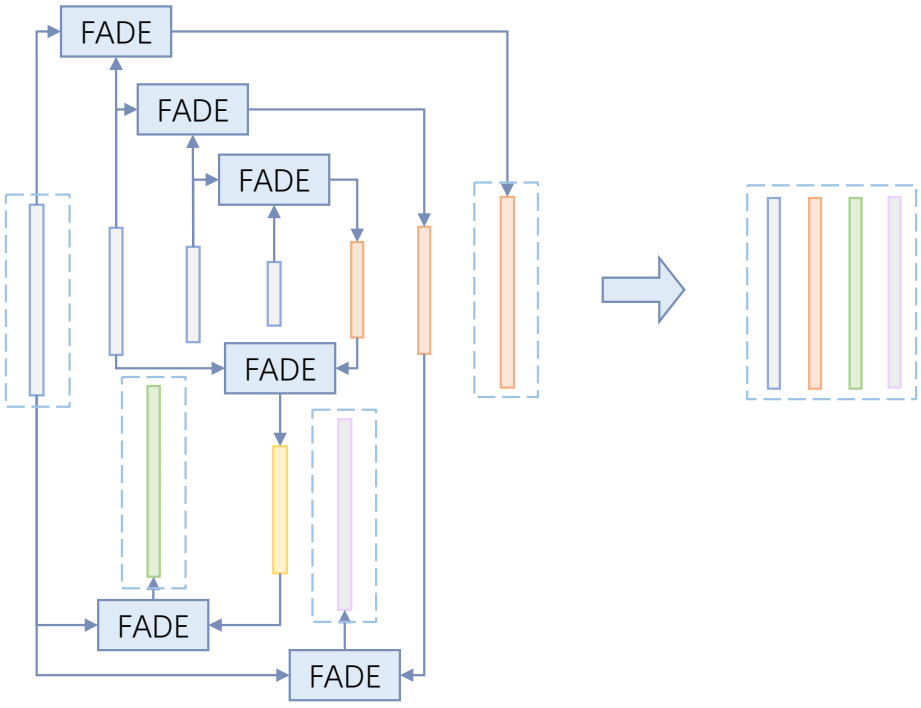

Figure S3: Illustration on how FADE is incorporated into SegFormer.

S4 How FADE Is Incorporated into SegFormer

In image matting, the input and output are the same as A2U. In semantic segmentation, as shown in Fig. S3, feature maps of each scale need to be upsampled to 1/4 of the original image. Therefore, there are $3 + 2 + 1 = 6$ upsampling operators involved in all.

S5 Additional Visualizations

Here we give additional visualizations on the ADE20K (Fig. S4) and the Adobe Composition 1K (Fig. S5) data sets. In segmentation, 'SegFormer+FADE' exhibits not only improved regional integrity but also sharp and consistent edges. In matting, FADE also contributes significantly to detail recovery, *e.g.*, the water drop below the bulb.

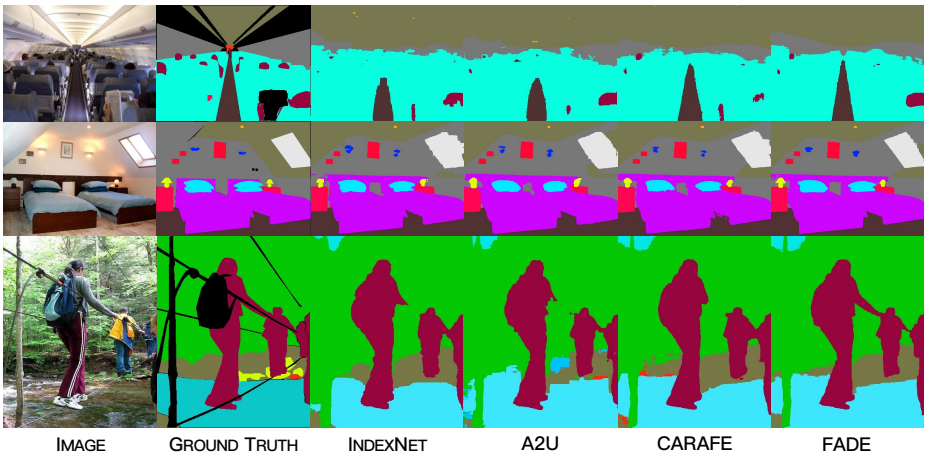

Figure S4: Additional visualizations of different upsampling operators on the ADE20K data set. Compared with other upsampling operators, FADE maintains both regional continuity and boundary accuracy.

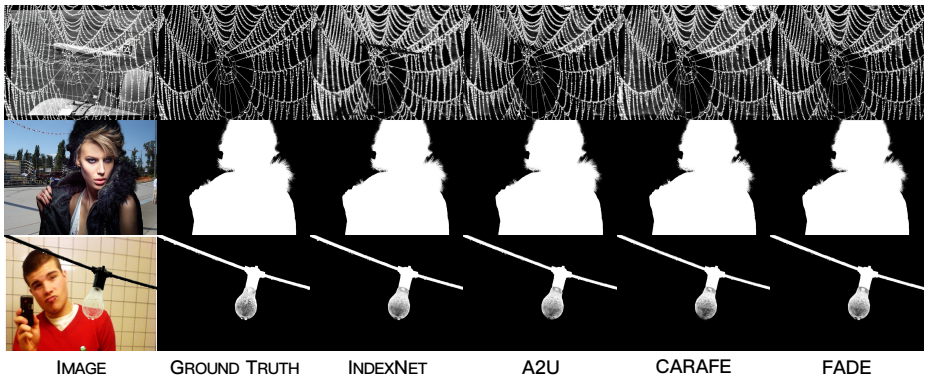

Figure S5: Additional visualizations on the Adobe Composition-1K testing set. Compared with other upsampling operators, FADE invites better detail delineation, *e.g.*, the water drop below the bulb.