CUF: Continuous Upsampling Filters

Anonymous Author(s)
Affiliation

Abstract

Neural fields have rapidly been adopted for representing 3D signals, but their application to more classical 2D image processing has been relatively limited. In this paper, we consider one of the most important operations in image processing: upsampling. In deep learning, learnable upsampling layers have extensively been used for single image super-resolution. We propose to parameterize upsampling kernels as neural fields. This parameterization leads to a compact architecture that obtains a 40-fold reduction in the number of parameters when compared with competing arbitrary-scale super-resolution architectures. When upsampling images of size 256x256 we show that our architecture is 2x-10x more efficient than competing arbitrary-scale super-resolution architectures, and more efficient than sub-pixel convolutions when instantiated to a single-scale model. In the general setting, these gains grow polynomially with the square of the target scale. We validate our method on standard benchmarks showing such efficiency gains can be achieved without sacrifices in super-resolution performance. https://cuf-paper.github.io

1. Introduction

Neural fields represent signals with coordinate-based neural-networks. They have found application in a multitude of areas including 3D reconstruction [24], novel-view synthesis [32], convolutions [14], and many others [37].

Recent research has investigated the use of neural fields in the context of single image super-resolution [6, 19]. These models are based on multi-layer perceptrons conditioned on latent representation produced by encoders. While such architectures allow for continuous-scale super-resolution, they require the execution of a conditional neural field for every pixel at the target resolution, making them unsuitable in applications with limited computational resources. Further, such a large use of resources is not justified by a increase in performance compared to classical convolutional architectures such as sub-pixel convolutions [31]. In summary, neural fields have not yet found widespread adoption as classical solutions are trivial to implement and more efficient. As they generally perform compa-

---

1Thereon we assume single image when talking about super-resolution.
2These encoders were originally proposed for classical super-resolution applications, and include both convolutional [8, 21, 40] as well as attentional [7, 20, 39] architectures.
rably, their usage is not justified in light of these points; see Figure 1. In this paper, we focus on overcoming these limitations, while noting that (regressive) super-resolution performance is in the saturation regime (i.e. further improvements in image quality seem unlikely without relying on generative modeling [30], and small improvements in PSNR do not necessarily correlate with image quality).

Our driving hypothesis is that super-resolution convolutional filters are highly correlated – both spatially, as well as across scales. Hence, representing such filters in the latent space of a conditional neural field can effectively capture and compress such correlations. Given that neural fields encode continuous functions within neural networks, we call our filters Continuous Upsampling Filters (CUFs). While neural fields have proven effective in parameterizing 3D convolutions [5, 14, 36], we demonstrate that there are very significant savings to be had in the parameterization of 2D convolutions.

In implementing continuous upsampling filters, we draw inspiration from sub-pixel convolutions [31], and realize them via depth-wise convolutions. This not only makes CUFs significantly more efficient than competing continuous super-resolution architectures, but when instantiated to single-image super-resolution they are, surprisingly, even more efficient than “ad-hoc” sub-pixel convolutions with same number of input and output channels; see Figure 1.

**Contributions.** We investigate the use of neural fields as a parameterization of convolutional upsampling layers in super-resolution architectures, and show how:

- The continuity of neural-fields leads to training of compact convolutional architecture for continuous super-resolution (similar super-resolution PSNR but with flop reduction that grows polynomially with the square of the target scale.).
- Instantiating our continuous kernels into their discrete counterpart leads to an efficient inference pipeline, even more efficient in number of operations per target pixel than ad-hoc sub-pixel convolutions with same number of input and output channels.
- Discrete cosine transforms can be used as an efficient replacement for Fourier bases in the implementation of positional encoding for neural fields.
- These gains do not hinder reconstruction performance (PSNR), by carefully validating our method on standard benchmarks, and thoroughly ablating our design choices.

**2. Related work**

A thorough coverage of single-image super-resolution can be found in the following surveys [2, 3, 35]; in what follows, we provide an overview of classical techniques for super-resolution based on deep learning. As our model is based on neural-fields, we also point the reader to a survey in this topic [37], and below we discuss the existing works for super-resolution based on neural fields.

**Single scale.** There are two main frameworks for super-res, which mainly differ in the placement of upsampling operators within the architecture.

- In the pre-upsampling framework [8], an initial upscaled image is obtained using a non-trainable upsampler (e.g. bicubic), which is then post-processed by a neural network. This enables arbitrary size-scaling, but can introduce side effects such as noise amplification and blurring.
- In the post-upsampling framework [31], an encoder that preserves the input spatial resolution is followed by a shallow upsampling component.

Note the computational complexity of pre-upsampling is significant, as these model operate directly at the target resolution. Consequently, post-upsampling is the de-facto mainstream approach [9, 20, 21, 34, 39, 40]. For this reason, we investigate the impact of an implicit upsampler in post-upsampling frameworks, and cover a diverse set of architectures ranging from large [22], to extremely lightweight models [10].

**Neural Fields.** The application of neural fields to super-resolution [16] has enabled the design of architectures that support multiple target scales, as well as non-integer super-resolution scales.

- In *meta super-resolution* MetaSR [16], a hyper-network is conditioned on the target scale factor and on the output pixel relative coordinates (i.e. the difference between source and target grids covering the same normalized global space). Their model consists of: (1) transforming the input image using an encoder into deep features at the source resolution; (2) upsampling the deep features into the target space (using nearest-neighbors); (3) using the hyper-network to produce a filter set per target relative position and scale; (4) processing the enlarged target
resolution feature map into the final output using the corresponding weights.

- In **local implicit image function** LIIF [6], the author eliminates the use of a hyper-network (MetaSR, step 3) by extending the sampled target resolution features (step 2) with extra channels representing the relative coordinates and target scale. Their combined features are further processed through a stack of five dense layers inspired by the multi-layer perceptron (MLP) architecture typically adopted as the end-to-end model in neural-fields formulations.

- In **local texture estimator** LTE [19] the authors enhance the features fed as input to the MLP layers. In their model, the output of the encoder is processed by three extra trainable layers associated with the amplitude, frequency and phase of sin/cosine waves. The resulting projection in the target resolution layer is used as the input to an MLP that stacks 4 fully-connected layers. In order to further improve the results they also adopt a global skip connection with a bilinear up-scaled version of the input around the full model, such that the deep model focus on the computation of the residual between the closed form approximation and the final result.

A visualization of these architectures can be found in Figure 2. In practice, LIIF and LTE also increase the number of feature channels as their MLP layers have more neurons (256) than the number of channels produced as the encoder output (64 for encoders such as EDSR [21] and RDN [40]). In contrast to these previous works, our upsampler:

- Moves the computational burden of arbitrary upsampling into the continuous upsampler operator while reducing the operations performed in the target resolution space, preserving the number of channels produced by the deep encoder (Table 1).

- Requires fewer operations than the single-scale sub-pixel convolution in the most typical application case of up-sampling an image by an integer scale factor, and producing results of comparable quality.

- Adopts a neural-fields formulation that focuses the sensitivity of our hyper-network to fine-grained changes in the representation of scale and relative position (reducing spectral bias).

### 3. Method

Given a target scale $s$, we aim to produce an up-scaled image of size $sH \times sW \times 3$ given an input image of size $H \times W \times 3$. Within the context of super-resolution, our core contribution is the introduction of a novel learnable upsampling layer. The upsampling layer can be interpreted as a decoder in a classical encoder-decoder architecture. We briefly review our encoder architecture in Section 3.1, and detail our decoder in Section 3.2. Note that our analysis focuses on the decoder, as a variety of encoders can be used. In Section 3.4, we then perform a conceptual comparison between our architecture, shown in Figure 3, and others based on neural fields, which are visually summarized in Figure 2.

**Training.** Our network with parameters $\theta$ is trained to **regress** a high resolution image $\hat{I}^s$ matching the ground truth $I^s_{gt}$ at random positions $x$ given the low resolution input image $I$:

$$\arg\min_{\theta} \mathbb{E}_I \mathbb{E}_{\hat{I}_s} \mathbb{E}_x \|\hat{I}^s(x; I, \theta) - \hat{I}^s_{gt}(x)\|_2$$ (1)
Differently from these architectures, we achieve this objective by instantiating upsampling kernels $\mathcal{K}$ via hyper-networks [13]. Our super-resolution network is algebraically expressed as:

$$\tilde{I}^t(x; \theta) = D_θ(\hat{K}_θ(δ^s(x); s) \cdot F(\lfloor x/s \rfloor)) \tag{5}$$

where $x$ indexes pixel locations in the target resolution, and $δ^s(x) = \text{mod}(x, s)/s$ renders our convolutional filters translation invariant, and the dot product implements a spatial convolution. The network $D$ is a point-wise layer that maps (super-resolved) features back to RGB values:

$$D_θ: \mathbb{R}^C \rightarrow \mathbb{R}^3 \tag{6}$$

This is applied to a grid of coordinates of size $\mathbb{R}^{sH \times sW \times C}$ to get an image of size $\mathbb{R}^{sH \times sW \times 3}$. The coordinate $x$ is continuous, hence $\hat{K}_θ$ is a neural field parameterization of a convolutional kernel mapping (continuous) spatial offsets and (continuous) scales to convolutional weights:

$$K_θ: [0, 1]^2 \times \mathbb{R}^+ \rightarrow \mathbb{R}^{C \times K^2} \tag{7}$$

Continuous kernel indexing. We can take our continuous formulation a step further by introducing a continuous parameterization of its kernel indexes $k_i, k_j$ indexing our $K \times K$ convolution weight entries. Thus, the hyper-network representing the convolution field becomes:

$$\hat{K}_θ: [0, 1]^2 \times \{0, \ldots, K - 1\}^2 \times \mathbb{R}^+ \rightarrow \mathbb{R}^C \tag{8}$$

and $K_θ$ can then be constructed by 9 invocations to $\hat{K}_θ$ in the case of a $K=3$, and $K^2$ invocations in the general setting. We use $K=3$ throughout the paper in order to facilitate a direct comparison to pre-existing baseline models.

Equation (7) and Equation (8) are both valid ways of generating spatial kernels. The first can be seen as a multi-headed hypernetwork, while the second uses input conditioning to generate the kernel values. We use the latter in our experiments, as the layers of nonlinearities provide additional expressiveness compared to the linear transformation used in the multi-headed version. A comparison between the two is presented in Appendix D.

3.3. Positional encoding

Naively representing signals with MLPs leads to “spectral bias” [26]: the overall difficulty these networks have in representing high-frequency signals. Hence we implement $K_θ$ as:

$$K_θ(δ^s(x), s, k) = \text{MLP}(\Pi(δ^s(x)), \Pi(s), \Pi(k)) \tag{9}$$

where we apply positional encoding $\Pi(\cdot)$ to the MLP inputs. Several variants of positional encoding exist, including Fourier [24] and random Fourier [32] variants. However, as we strive for efficiency, we take inspiration from classical signal processing (e.g. JPEG) and instead employ

Figure 4. Continuous super-resolution – Our architecture directly allows for an arbitrary and continuous choice of the super-resolution factor. Conversely, traditional methods perform this indirectly: they need to upsample to an integer factor, and then consequently downsample to the target resolution. More examples are available in the appendix. Note that $\times 4.5$ and $\times 5$ are upsampling factors not available in the training data.

Figure 5. Continuous filters – CUF filters are continuous functions stored in neural fields. We visualize our filters by densely sampling each on the $[0, 1]^2$ domain to highlight their continuous nature (left). Close-up of an exemplar filter (middle). Filters are discretized by sampling at pixel centers (right), allowing them to be used in classical fixed-scale super-res architectures.

3.1. Encoder

Towards this objective, we first process our input image with an encoder producing $C$-dimensional features:

$$E_θ: \mathbb{R}^{H \times W \times 3} \rightarrow \mathbb{R}^{H \times W \times C} \tag{2}$$

and then unfold a $k \times k$ spatial neighborhood of $C$-dimensional features into a tensor of $C \times k^2$ channels. Finally, we upsample the encoded image using nearest-neighbour interpolation. We define this step as:

$$U: \mathbb{R}^{sH \times sW \times C} \rightarrow \mathbb{R}^{sH \times sW \times C \times k^2} \tag{3}$$

The unfolding part of this procedure leads to feature maps that will allow us to implement depth-wise, spatial convolutions as dot products, as is typically done for low-level neural network implementations. We define this whole feature extraction procedure as:

$$F = U(E_θ(I)) \tag{4}$$

3.2. Decoder – “Continuous Upsampling Filters”

We draw our inspiration from classical sub-pixel convolutions [31], and combine it with recent research that apply neural fields for continuous super-resolution [6, 19]. Differently from these architectures, we achieve this objective by instantiating upsampling kernels $\mathcal{K}$ via hyper-networks [13]. Our super-resolution network is algebraically expressed as:

$$\tilde{I}^t(x; \theta) = D_θ(\hat{K}_θ(δ^s(x); s) \cdot F(\lfloor x/s \rfloor)) \tag{5}$$

where $x$ indexes pixel locations in the target resolution, and $δ^s(x) = \text{mod}(x, s)/s$ renders our convolutional filters translation invariant, and the dot product implements a spatial convolution. The network $D$ is a point-wise layer that maps (super-resolved) features back to RGB values:

$$D_θ: \mathbb{R}^C \rightarrow \mathbb{R}^3 \tag{6}$$

This is applied to a grid of coordinates of size $\mathbb{R}^{sH \times sW \times C}$ to get an image of size $\mathbb{R}^{sH \times sW \times 3}$. The coordinate $x$ is continuous, hence $\hat{K}_θ$ is a neural field parameterization of a convolutional kernel mapping (continuous) spatial offsets and (continuous) scales to convolutional weights:

$$K_θ: [0, 1]^2 \times \mathbb{R}^+ \rightarrow \mathbb{R}^{C \times K^2} \tag{7}$$

Continuous kernel indexing. We can take our continuous formulation a step further by introducing a continuous parameterization of its kernel indexes $k_i, k_j$ indexing our $K \times K$ convolution weight entries. Thus, the hyper-network representing the convolution field becomes:

$$\hat{K}_θ: [0, 1]^2 \times \{0, \ldots, K - 1\}^2 \times \mathbb{R}^+ \rightarrow \mathbb{R}^C \tag{8}$$

and $K_θ$ can then be constructed by 9 invocations to $\hat{K}_θ$ in the case of a $K=3$, and $K^2$ invocations in the general setting. We use $K=3$ throughout the paper in order to facilitate a direct comparison to pre-existing baseline models.

Equation (7) and Equation (8) are both valid ways of generating spatial kernels. The first can be seen as a multi-headed hypernetwork, while the second uses input conditioning to generate the kernel values. We use the latter in our experiments, as the layers of nonlinearities provide additional expressiveness compared to the linear transformation used in the multi-headed version. A comparison between the two is presented in Appendix D.

3.3. Positional encoding

Naively representing signals with MLPs leads to “spectral bias” [26]: the overall difficulty these networks have in representing high-frequency signals. Hence we implement $K_θ$ as:

$$K_θ(δ^s(x), s, k) = \text{MLP}(\Pi(δ^s(x)), \Pi(s), \Pi(k)) \tag{9}$$

where we apply positional encoding $\Pi(\cdot)$ to the MLP inputs. Several variants of positional encoding exist, including Fourier [24] and random Fourier [32] variants. However, as we strive for efficiency, we take inspiration from classical signal processing (e.g. JPEG) and instead employ
Table 2. Out-of-domain evaluation – performance on datasets not seen during training. Metrics computed on the luminance channel in the YCbCr color space, following previous work. Models using self-ensemble are marked with +geo and models using local self-ensemble are marked with +loc. **Legend:** best 2nd best

<table>
<thead>
<tr>
<th>Encoder</th>
<th>Upsampler</th>
<th>Ens.</th>
<th>Set5</th>
<th>Set14</th>
<th>BSD100</th>
<th>Urban100</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDN</td>
<td>fixed-scale [60]</td>
<td>x2</td>
<td>34.71</td>
<td>32.47</td>
<td>30.01</td>
<td>28.81</td>
</tr>
<tr>
<td>MetaSR [16]</td>
<td>x2</td>
<td>34.83</td>
<td>32.38</td>
<td>29.04</td>
<td>31.98</td>
<td>30.54</td>
</tr>
<tr>
<td>CUF (ours)</td>
<td>x2</td>
<td>34.72</td>
<td>32.54</td>
<td>29.25</td>
<td>31.99</td>
<td>30.58</td>
</tr>
</tbody>
</table>

| SwinIR | fixed-scale [31] | x2 | 34.89 | 32.72 | 34.14 | 30.75 | 29.84 | 32.44 | 29.37 | 27.83 | 33.40 | 29.29 | 27.03 |
| MetaSR [16] | x2 | 34.77 | 32.47 | 29.09 | 34.14 | 30.66 | 28.85 | 26.58 | 32.39 | 29.31 | 27.75 | 25.94 |
| CUF (ours) | x2 | 34.88 | 32.89 | 29.53 | 34.25 | 30.80 | 29.00 | 26.85 | 32.45 | 29.38 | 27.85 | 26.09 |

| RDN | fixed-scale [31] | x2 | 34.95 | 32.81 | 34.24 | 30.83 | 29.92 | 32.47 | 29.41 | 27.87 | 33.51 | 29.42 | 27.24 |
| MetaSR [16] | x2 | 34.77 | 32.47 | 29.09 | 34.14 | 30.66 | 28.85 | 26.58 | 32.39 | 29.31 | 27.75 | 25.94 |
| CUF (ours) | x2 | 34.88 | 32.89 | 29.53 | 34.25 | 30.80 | 29.00 | 26.85 | 32.45 | 29.38 | 27.85 | 26.09 |


<table>
<thead>
<tr>
<th>Encoder</th>
<th>Upsampler</th>
<th>Ens.</th>
<th>seen scales</th>
<th>unseen scales</th>
</tr>
</thead>
<tbody>
<tr>
<td>BTNR</td>
<td>x4</td>
<td></td>
<td>30.11</td>
<td>28.42</td>
</tr>
<tr>
<td>EDSR-B</td>
<td>Sub-pixel conv.</td>
<td>x4</td>
<td>34.69</td>
<td>30.94</td>
</tr>
<tr>
<td>MetaSR</td>
<td>x4</td>
<td></td>
<td>34.70</td>
<td>30.99</td>
</tr>
<tr>
<td>CUF (ours)</td>
<td>x4</td>
<td></td>
<td>34.72</td>
<td>31.02</td>
</tr>
</tbody>
</table>

| SwinIR | x4        |     | 35.10 | 31.33 | 29.31 | 26.88 | 23.73 | 22.18 |
| MetaSR  | x4        |     | 35.00 | 31.27 | 29.25 | 27.03 | 23.94 | 22.38 |

| CUF (ours) | x4 |     | 35.03 | 31.31 | 29.32 | 27.03 | 23.94 | 22.38 |

| SwinIR | x4        |     | 35.28 | 31.47 | 29.40 | 26.94 | 23.80 | 22.26 |
| MetaSR  | x4        |     | 35.15 | 31.40 | 29.33 | 27.03 | 23.94 | 22.40 |
| CUF (ours) | x4 |     | 35.32 | 31.52 | 29.44 | 27.19 | 24.07 | 22.49 |

| CUF (ours) | x4 |     | 35.33 | 31.52 | 29.44 | 27.19 | 24.07 | 22.49 |
| SwinIR | x4        |     | 35.17 | 31.46 | 29.46 | 27.15 | 24.02 | 22.43 |
| MetaSR  | x4        |     | 35.24 | 31.50 | 29.51 | 27.20 | 24.09 | 22.50 |
| CUF (ours) | x4 |     | 35.31 | 31.56 | 29.56 | 27.23 | 24.10 | 22.52 |

a cosine-only transformation. Given a scalar $f_n$ sampled uniformly in the range $[0, f_{\text{max}}]$, a scalar quantity $z \in [0, 1]$ is encoded as the (sorted) vector:

$$
\Pi(z) = \left\{ \cos \left( \frac{(2z + 1)f_n\pi}{2} \right) \right\}_{n=1}^{N}
$$

By eliminating the imaginary component from the Fourier basis, we show in Section 4.3 how this reduces the number of trainable parameters of the first hyper-network layer by half, without affecting super-resolution quality.

**3.4. Analysis**

We now perform a conceptual comparison of our network architecture to the commonly used Sub-Pixel Convolution, as well as several super-resolution approaches based on neural-fields. We further note how continuous upsampling filters can be instantiated to generate filters compatible with Sub-Pixel Convolution.

**Comparison to Sub-Pixel Convolution** [31]. Sub-Pixel Convolution is the most commonly used super-resolution operator, especially when efficiency is critical. This is achieved by initially employing an expansion convolution that generates a feature map with $s^2 N_{\text{out}}$ channels, where $N_{\text{out}}$ is a hyper parameter defining the target number of channels of the Sub-Pixel Convolution operation. Next, the periodic shuffling operator $\pi^3$ re-arranges the channels to produce a higher-resolution feature map (see Figure 6):

$$
P : \mathbb{R}^{H \times W \times C} \rightarrow \mathbb{R}^{sH \times sW \times C / s^2}
$$

Similar to (6), whenever $N_{\text{out}}$ is chosen as different from the number of output colors, the resulting feature map is further projected by point-wise convolutions. Often, $N_{\text{out}}$ is taken as the same as the number of channels as produced by the encoder $\mathcal{E}$ for architectures adopting Sub-Pixel Convolution and targeting high quality results [20,21,40]. We note that the Sub-Pixel Convolution design based on the expansion convolution does not enforce spatial correlations between neighboring pixels (here we refer to neighborhood in the output, high resolution, domain) and need to be learned from training data. Conversely, these correlations are inherently captured by our continuous convolutional filters.

Further, in the integer scale setup where hyper-networks can be pre-instantiated, CUFs are computationally more efficient than sub-pixel convolutions whenever $K^2 + N_{\text{in}} + \frac{3}{4}$ also called pixel-shuffle (PyTorch) or depth-to-space (TensorFlow),
\( N_{out} < N_{out} \times K^2 \) (in the worst case scenario) as the cost associated with pre-computing the weights for a given integer up-sampling scale can be neglected when compared to the operations performed on the image grid (see Appendix E).

**Comparisons to LIIF [6] and LTE [19].** In comparison to these works, we make the main network shallower (1 depth-wise continuous convolution and 2 dense layers) in order to reduce the number of layers that operate at the target spatial resolution and at the encoder’s channel resolution. At the same time, our upsampler head is considerably lighter than previous arbitrary scale methods, not only by the use of a depth-wise convolution but also by keeping the same number of channels produced by the features encoder in the main network \((C_e = 64\), which results in dense layers \(16 \times \) cheaper than its LIIF and LTE counterparts using 256 neurons). When performing non-integer upsampling, the costs with processing the hyper-network layers are proportional to the target image resolution, but still smaller than a single dense layer of LIFF and LTE heads, due to the adoption of a reduced number of channels (as \(C_h = 32\), each of CUF’s hypernetwork dense layers is \(64 \times \) cheaper than LIIF and LTE layers using 256 neurons).

**Instantiating CUFs.** At inference time, when targeting an integer upscaling factor \(s\), the hyper-network representing \(K\) can be queried to retrieve the weights corresponding to \(s^2\) relative subpixel positions as an initialization step during pre-processing. The retrieved weights are re-used across all pixels taking advantage of the existent periodicity. Thus, in the CUF-instantiated architecture the continuous kernel is replaced at test time with a discrete depth-wise convolution, followed by a pixel shuffling operation, in contrast with the unfolding operator used in the regular, fully continuous CUF described in this paper. Otherwise, the architecture remains the same. The costs associated with the hyper network at initialization can be neglected as \(s^2 < sH \times sW\). In this setting, our model becomes as efficient as a sub-pixel-convolution architecture, while retaining the aforementioned continuous modeling properties at training time.

**4. Results**

In this section we describe our experimental setup in terms of backbone encoders and training/validation dataset, perform careful comparisons to the state-of-the-art (Section 4.1), and additional evaluation towards the implementation of lightweight super-res architectures (Section 4.2). We conclude by performing a thorough ablation (Section 4.3).

**Encoders.** We apply CUFs to a variety of encoders, to both show its generality, as well as its performance gains in a variety of settings.

- Well-known convolution-based encoders, namely EDSR-baseline [21] and Residual Dense Networks (RDN) [40].
- Architectures for lightweight and extremely lightweight inference, respectively SwinT-lightweight [20] and ABPN [10].

<table>
<thead>
<tr>
<th>Method</th>
<th>Composition</th>
<th>Parameters (in M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDN</td>
<td>Convolutions</td>
<td>22.00 0.30</td>
</tr>
<tr>
<td>SwinT</td>
<td>Conv. and Self-Attention</td>
<td>11.60 0.30</td>
</tr>
<tr>
<td>EDSR-baseline</td>
<td>Convolutions</td>
<td>1.20 0.30</td>
</tr>
<tr>
<td>SwinT-lightweight</td>
<td>Conv. and Self-Attention</td>
<td>0.90 0.03</td>
</tr>
<tr>
<td>ABPN</td>
<td>Convolutions</td>
<td>0.03 0.03</td>
</tr>
</tbody>
</table>

Table 4. **Encoders** – Our experiments use CUF layers in combination with various encoders differing in composition and size. Parameter counts reflect the encoder size and the upsampling head using a Sub-Pixel Convolution (spc) targeting \(4 \times\) upsampling.

**Datasets.** All models are trained using the DIV2K dataset training subset (800k images), introduced at the NTIRE 2017 Super Resolution Challenge [25]. Similarly to previous works, each image is randomly cropped 20 times per epoch and augmented with random flips, and 90 degrees rotations. We report peak signal-to-noise ratio (PSNR) results on the DIV2K validation set (100 images)\(^4\), in the RGB space. To understand the generalizability of our experimental results, we test our models on additional well known datasets (Set5 [4] – 5 images; Set14 [38] – 14 images; B100 [23] – 100 images; and Urban100 [17] – 100 images), in which, following previous works, PSNR is measured in the YCbCr luminance channel.

**Ensembling.** Whenever demarked by ‘\(+_{\text{geo}}\)’, results incorporate the (commonly adopted) Geometric Self-ensemble [21], where the results of rotated versions of the input images are averaged at test time. LIIF and LTE results adopt Local Self-ensemble (‘\(+_{\text{loc}}\)’), where the result of applying the upsampler at 4 shifted grid points is averaged, resulting in a \(4 \times\) increase in computational complexity at training/test time. It may also introduce confounding factors to the optimization process that invalidate direct comparison to the fixed-scale upsamplers, as during backpropagation the encoder gradients are averaged. In Appendix B we combine these models.

**4.1. State-of-the-art comparisons**

Table 2 and Table 3 show the quantitative comparisons, while qualitative results can be observed in Figure 4, Figure 7 and Appendix F as well as on our supplementary website https://cuf-paper.github.io/. Table 3 presents the ablated encoders’ composition and size. Hyperparameter settings can be found in the appendix.

\[^4\]The models we compare to also do not report results on the test-set, which is no longer publicly available.
contains in-domain results: models trained on DIV2k and tested on DIV2k (train/test split), while Table 2 presents out-of-domain results: models are trained on DIV2k, but tested on different datasets. Note that upsampling ratios larger than $\times 4$ are not provided at training time. The models adopting sub-pixel convolutions are fixed-scale models, that is, one model is trained per target scale and thus do not generalize to unseen or fractional scales. On the remaining rows we present results of arbitrary-scale upsamplers (MetaSR [16], LIIF [6], LTE [19]), where a single model is trained and tested across different upsampling ratios. Our method is the first arbitrary-scale method to match (or surpass) fixed-scale performance across all encoders in both single-pass inference and self-ensemble scenarios.

### 4.2. Lightweight super-resolution

The encoders and upsamplers considered so far are not designed to be run on devices with limited memory and power. Running super-resolution algorithms on mobile devices comes with unique challenges due to limited RAM and non-efficient support for certain common operations [18]. The proposed CUF-based upsampler, in combination with a mobile-compatible encoder, can be effectively used for upsampling on mobile devices.

To test the effectiveness of the CUFs, we adopt a lightweight version of SwinIR (0.9M parameters) [20] as well as one of the best performing super-lightweight architectures for mobile devices (30K parameters) [10] as the encoder. For the latter, the original architecture (Figure 8, left) consists of a lightweight encoder and an upsampling module consisting of a series of convolutions, ReLU’s, and a standard periodic shuffle layer. We replace this upsampling module with a CUF-based one (Figure 8, right). With instantiated CUFs, this means simply introducing a depthwise convolution followed by pointwise layers, where the weights are output by the learned CUFs.

Table 5 illustrates that CUFs provide the same level of accuracy while enabling continuous scale and efficient inference. To our knowledge, this is the first mobile-friendly continuous scale super-resolution architecture.

### 4.3. Ablations

We run several ablation experiments in order to analyze the effects of various model choices. Here, we detail our

![Figure 7. Qualitative evaluation – CUF combined with different encoders and upsampling scales.](image)

![Figure 8. Architectures (lightweight) – The original ABPN [11] single scale architecture (left) and our arbitrary-scale version using the implicit upsampler (right). Note how the two consecutive $3 \times 3$ native convolutions are exchanged for a $3 \times 3$ depth-wise implicit convolution followed by point-wise layers.](image)
choice of using DCT for positional encoding vs Fourier features, as well as an investigation of the redundancy of CUF filters versus subpixel convolution. An analysis on the various conditioning factors of our neural fields (positional encoding, kernel indices, scale, and target subpixel) can be found in the appendix.

**Positional encoding.** A large body of work uses a Fourier basis for the positional encodings, e.g., [24, 33]. However, we opt to use a DCT basis as this requires half of the sinusoidal projections and neurons at the hyper-network first layer. This trick has been used in other domains, such as random approximations of stationary kernel functions [27]. In Table 6 we show the PSNR, as obtained using a DFT versus a DCT basis, and verify that we do not observe loss in performance, while at the same time saving half of the computation associated with positional encoding operations.

**Redundancy of filters.** As noted in Section 3.4, sub-pixel convolution does not enforce spatial correlation between neighboring weights in its convolutional filters, but has to learn them from data. CUF on the other hand generates the filters with a hypernetwork, which intrinsically builds on smoothness due to its functional form. Here, we test this hypothesis on the filters of CUF and sub-pixel convolutions. For this analysis, we use an EDSR encoder with \( C_e = 64, K = 3 \times 3 \), and \( s = 3 \). Subpixel convolution maps \( C_e \) features to \( s^2C_e \) features, and each group of \( s^2 \) output features are rearranged to form the upsampled feature space. We therefore consider each group of \( s^2 \) filters of size \( K \times K \times C_e \) and do an eigenvalue analysis to determine the compressibility of these filters. This is done by forming \( C_e \) separate matrices of size \( s^2 \times C_eK^2 \) and performing PCA on each of them separately, forming \( 64 \) separate sets of eigenvalues. We plot this distribution for trained and untrained sub-pixel layers in Figure 9a. A faster decay means more redundancy, as more variance is explained by fewer components. In this case, training adds significant structure to the filters.

Conversely, each output channel in CUF is computed by \( s^2 \) filters of size \( K \times K \). These are not directly comparable to subpixel convolution, but we perform an analogous eigenvalue analysis on trained and untrained CUFs. As shown in Figure 9b, unlike subpixel convolution, CUFs actually become less redundant with training, suggesting that the initial hypernetwork already imposes a very smooth prior on the filters. In other words, CUF filters incorporate spatial correlations from the very beginning due to their functional form.

### 5. Conclusions

We propose CUF, a computationally efficient modeling of continuous upsample filters as neural fields. Different from previous arbitrary-scale architectures, the performance gains obtained by our upsampler are not due to increased capacity but rather to a more efficient use of model parameters. A single hyper-network supports filter adaptation across scales while at the same time CUF’s upsampling head has fewer parameters than previous arbitrary-scale heads, as well as fewer parameters than a single Sub-Pixel Convolution head at \( 4 \times 4 \) upsampling. CUF is the first arbitrary scale model that can be effectively used for upscaling on mobile devices, an area previously dominated by single scale Sub-Pixel Convolution architectures. CUF’s architecture is quite general. A natural direction of future work is to investigate its use for architectures with multiple upsampling layers (and the reuse of filters across scales), for different image generation settings such as GANs [12] and diffusion models [15]. Neural fields have done particularly well on higher-dimensional signals. Another line of future work is the use of CUF for efficient super-resolution of video data. Overall, the presented results aligned with low computational resources suggest CUF offers a practical modeling of continuous filters. Continuous convolution filters are an exciting and increasingly studied area of research [28, 29] with many applications as an alternative to traditional convolutions.
References


CUF: Continuous Upsampling Filters

Supplementary Material

A. Experiment details

Training Hyper-parameters. Table 7 contains the hyper-parameters used for training the ablated models, taken from their single-scale (Sub-Pixel Conv.) training settings. All models are trained for 1K epochs with $L_1$ loss and ADAM optimizer by setting $\beta_1 = 0.9, \beta_2 = 0.999$, and $\epsilon = 1e^{-8}$. We use a step-wise learning rate schedule that is halved at epochs $[500, 800, 900, 950]$. Unless referred as single-scale, models were trained with random scales by sampling the scale factor uniformly within the continuous interval $[1, 4]$. In order to ensure that the dimensions within the training mini-batch match despite heterogeneous scale factors, we first scale each image as required and then apply the same crop size to both LR and HR images, such that the HR random crop contains $1/s^2$ of the content of the LR crop, and fix the relative grid coordinates to point to the random sub-region.

CUF’s hyper parameters. Table 8 describes the number of neurons used in the presented ablations across different encoders. $C_e$ was chosen to replicate the original setting. CUF’s hyper-parameters were obtained by grid search on the EDSR-baseline and Set5 dataset and replicated on the remaining encoders. CUF’s positional encoding hyper-parameters enforce a small number of basis per input parameter. For each input (represented in $2D$ space), the number of basis $N^2$ was searched within the set $\{0, 1^2, 2^2, ..., 5^2\}$ while $f_{\text{max}}$ within the set $\{0, 0.5, ..., 4.0\}$ for $\Pi(\delta(x))$ and $\Pi(s)$,

and within the set $\{0, 0.5, ..., 3.0\}$ for $\Pi(k)$. The final positional encoding hyper-parameters adopted are $\Pi(\delta(x);s) : \{N^2 = 25; f_{\text{max}} = 2.0\}$, $\Pi(s) : \{N^2 = 25; f_{\text{max}} = 2.0\}$ and $\Pi(k) : \{N^2 = 9; f_{\text{max}} = 1.0\}$.

B. On the use of ensemble

Table 9. Disentangling the effect of ensembling on optimization: Models trained under same supervision (no ensemble), and tested with (marked with $+_{geo}$) and without geometric self-ensemble. Results on DIV2K’s validation subset [25].

The baseline settings from LIIF [6] and LTE [19] include locally ensembling pixels around the target sub-pixel, by shifting by its position by half pixel in the low resolution grid, and averaging their results. This procedure introduces a training and inference overhead, as the sampled points is increased by a factor of four. As a direct consequence, during training models adopting local self-ensemble evaluate four times more gradients per optimization step. In order to disentangle possible optimization side effects, in this section we ablate the models LIIF and LTE under same optimization conditions as other models, that is, no ensemble is adopted during training, but on inference time only. LTE presents a strong result on scales 3 and larger, but a reduction in performance on scale 2, in which LIIF matches or surpass its performance. Overall, this ablation confirms the benefits from CUF as the lighter arbitrary scale upsampler with strong performance under single pass and self-ensemble settings, across both smaller and larger scales.

C. On the use of positional encoding

Figure 10 contains a comparison of CUF models trained using the neural-fields parameters as raw values versus the projection using positional encoding. The stronger impact of using positional encoding is observed on ABPN encoder (Figure 11) and Urban-100 dataset [17]. We note that the content of this dataset is characterized by sharp straight lines and geometric structures, thus the quantitative gain is aligned with the expected behaviour.
Figure 10. Impact of CUF’s positional encoding on different datasets and encoders. Bar plots represent PSNR differences relative to baseline models adopting sub-pixel convolutions and corresponding encoder.

Figure 11. Qualitative evaluation on mobile-compatible encoder – ABPN-CUF with and without positional encoding. Butterfly image from Set5 dataset.

D. Conditioning on the kernel indexes

Figure 12 contains a comparison between representing the kernel indexes $k_i, k_j$ as input parameters to the neural-fields versus representing their discrete set as individual neurons at the output of the hyper-network. On the efficiency side, setting them as the hyper-network output neurons reduces memory and computation used, as hidden layers are shared. On the other hand, the layers of the hyper-network and its nonlinearities provide additional expressiveness compared to the linear transformation used in the multi-headed version. This additional expressiveness results in in performance improvement in stronger encoders (RDN, SWINIR), but not on smaller ones (ABPN, EDSR). Thus, we recommend conditioning on kernel indexes only for those encoders that take advantage of it.

E. Sub-Pixel Convolution vs. CUF-instantiated

In this section we compare the costs associated with Sub-Pixel Convolution and CUF-instantiated upsampling heads. The presented comparison contrasts their designs choices based on full convolution (Sub-Pixel Convolution) versus depthwise-pointwise decomposition (CUF). As unitary element of comparison we evaluate the number of multiplications performed to produce a single output pixel. We assume an input feature map with $C_{in}$ channels, the resulting image with $C_{out}$ channels and that both Sub-Pixel Convolution and CUF-instantiated adopt filters of same size $K$.

CUF-instantiated architecture is composed with a depthwise convolution and pointwise projections. Its three layers perform respectively $C_{in} \times k^2$, $C_{in} \times C_{in}$ and $C_{in} \times C_{out}$ multiplications per output pixel.

Next, we cover two common compositions with Sub-Pixel Convolution. The most common design for upsampler heads targeting high quality results is to combine a Sub-Pixel Convolution layer with a pointwise layer projecting from $C_{in}$ into RGB channels ($C_{out}$) ([20, 21, 40]). In this setting, both Sub-Pixel Conv. and CUF-instantiated have identical output layers, that is removed from our analysis. The number of multiplications performed by the Sub-Pixel Convolution layer alone per target subpixel is: $C_{in} \times k \times k \times C_{in}$. Thus, the fraction of multiplications performed by CUF-instantiated in relation to Sub-Pixel Convolution can be expressed as: $\frac{k^2 + C_{in}}{k^2 + C_{in}} = \frac{1}{C_{in}} + \frac{1}{k}$. That is, the decomposition has the desired effect of saving computation whenever $C_{in}, k > 1$.

An alternative use of a Sub-Pixel Convolution layer is its direct use as output layer. In this case, the three layers that compose CUF’s upsampling head are compared to the full
expansion convolution alone. Thus, the total operations performed by CUF-instantiated is smaller that those performed by Sub-Pixel Convolution whenever \( k^2 + C_{in} + C_{out} < k^2 * C_{out} \).

In practice, the depthwise-pointwise decomposition adopted by CUF-instantiated faces a memory drawback of storing an extra feature map created in-between the decomposition layers \( C_{in} \) (Figure 1). The reduction of such drawback with fused-convolutions is left as future work [1].

F. Qualitative comparisons

The difference between existent arbitrary-scale upsamplers can only be observed at textured regions of the image. In this section we disentangle the rule of the encoder and upsampler in the perceived quality of the results. Figure 15 contain additional results, with non-integer scale factors.
Figure 13. Qualitative comparisons of arbitrary-scale super resolution methods using different encoders. Scale factor $4 \times$
Figure 14. Hard cases: Aliasing artifacts observed on hard cases. Our upsampler produces the sharper results. Scale factor $8 \times$

Figure 15. Qualitative results using non-integer scales. Images from Set14 dataset.