Pixel-Adaptive Convolutional Neural Networks

Hang Su[1], Varun Jampani[2], Deqing Sun[2], Orazio Gallo[2], Erik Learned-Miller[1], and Jan Kautz[2]

[1]UMass Amherst [2]NVIDIA

Abstract

Convolutions are the fundamental building blocks of CNNs. The fact that their weights are spatially shared is one of the main reasons for their widespread use, but it is also a major limitation, as it makes convolutions content-agnostic. We propose a pixel-adaptive convolution (PAC) operation, a simple yet effective modification of standard convolutions, in which the filter weights are multiplied with a spatially varying kernel that depends on learnable, local pixel features. PAC is a generalization of several popular filtering techniques and thus can be used for a wide range of use cases. Specifically, we demonstrate state-of-the-art performance when PAC is used for deep joint image upsampling. PAC also offers an effective alternative to fully-connected CRF (Full-CRF), called PAC-CRF, which performs competitively compared to Full-CRF, while being considerably faster. In addition, we also demonstrate that PAC can be used as a drop-in replacement for convolution layers in pre-trained networks, resulting in consistent performance improvements.

1. Introduction

Convolution is a basic operation in many image processing and computer vision applications and the major building block of Convolutional Neural Network (CNN) architectures. It forms one of the most prominent ways of propagating and integrating features across image pixels due to its simplicity and highly optimized CPU/GPU implementations. In this work, we concentrate on two important characteristics of standard spatial convolution and aim to alleviate some of its drawbacks: *Spatial Sharing* and its *Content-Agnostic* nature.

Spatial Sharing: A typical CNN shares filters' parameters across the whole input. In addition to affording translation invariance to the CNN, spatially invariant convolutions significantly reduce the number of parameters compared with fully connected layers. However, spatial sharing is not without drawbacks. For dense pixel prediction tasks, such as semantic segmentation, the loss is spatially varying because of varying scene elements on a pixel grid. Thus the optimal gradient direction for parameters differs at each pixel. However, due to the spatial sharing nature of convolution, the loss gradients from all image locations are globally pooled to train each filter. This forces the CNN to learn filters that minimize the error across all pixel locations at once, but may be sub-optimal at any specific location.

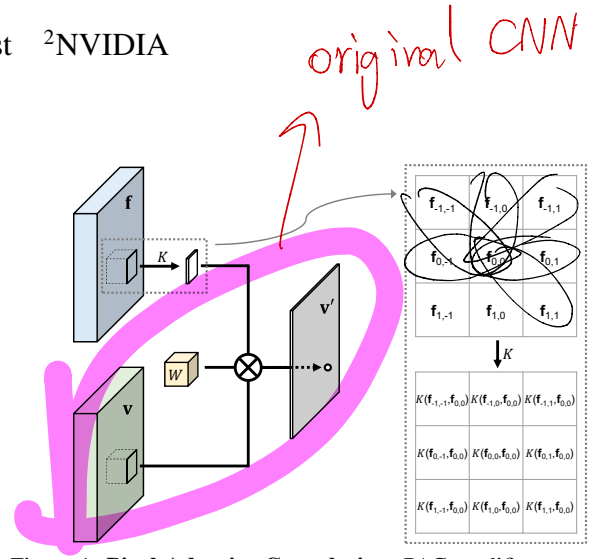

Figure 1: **Pixel-Adaptive Convolution.** PAC modifies a standard convolution on an input $\mathbf{v}$ by modifying the spatially invariant filter $\mathbf{W}$ with an adapting kernel K. The adapting kernel is constructed using either pre-defined or learned features $\mathbf{f}$. $\otimes$ denotes element-wise multiplication of matrices followed by a summation. Only one output channel is shown for the illustration.

Content-Agnostic: Once a CNN is trained, the same convolutional filter banks are applied to all the images and all the pixels irrespective of their content. The image content varies substantially across images and pixels. Thus a single trained CNN may not be optimal for all image types (e.g., images taken in daylight and at night) as well as different pixels in an image (e.g., sky vs. pedestrian pixels). Ideally, we would like CNN filters to be adaptive to the type of image content, which is not the case with standard CNNs. These drawbacks can be tackled by learning a large number of filters in an attempt to capture both image and pixel variations. This, however, increases the number of parameters, requiring a larger memory footprint and an extensive amount of labeled data. A different approach is to use *content-adaptive* filters inside the networks.

Existing content-adaptive convolutional networks can be

broadly categorized into two types. One class of techniques make traditional image-adaptive filters, such as bilateral filters [2, 42] and guided image filters [18] differentiable, and use them as layers inside a CNN [25, 29, 52, 11, 9, 21, 30, 8, 13, 31, 44, 46]. These content-adaptive layers are usually designed for enhancing CNN results but not as a replacement for standard convolutions. Another class of content-adaptive networks involve learning position-specific kernels using a separate sub-network that predicts convolutional filter weights at each pixel. These are called "Dynamic Filter Networks" (DFN) [48, 22, 12, 47] (also referred to as cross-convolution [48] or kernel prediction networks [4]) and have been shown to be useful in several computer vision tasks. Although DFNs are generic and can be used as a replacement to standard convolution layers, such a kernel prediction strategy is difficult to scale to an entire network with a large number of filter banks.

In this work, we propose a new content-adaptive convolution layer that addresses some of the limitations of the existing content-adaptive layers while retaining several favorable properties of spatially invariant convolution. Fig. 1 illustrates our content-adaptive convolution operation, which we call "Pixel-Adaptive Convolution" (PAC). Unlike a typical DFN, where different kernels are predicted at different pixel locations, we *adapt* a standard spatially invariant convolution filter $\mathbf{W}$ at each pixel by multiplying it with a spatially varying filter K, which we refer to as an "adapting kernel". This adapting kernel has a pre-defined form (e.g., Gaussian or Laplacian) and depends on the pixel features. For instance, the adapting kernel that we mainly use in this work is Gaussian: $e^{-\frac{1}{2}||\mathbf{f}_i - \mathbf{f}_j||^2}$, where $\mathbf{f}_i \in \mathbb{R}^d$ is a d-dimensional feature at the i^{th} pixel. We refer to these pixel features $\mathbf{f}$ as "adapting features", and they can be either pre-defined, such as pixel position and color features, or can be learned using a CNN.

We observe that PAC, despite being a simple modification to standard convolution, is highly flexible and can be seen as a generalization of several widely-used filters. Specifically, we show that PAC is a generalization of spatial convolution, bilateral filtering [2, 42], and pooling operations such as average pooling and detail-preserving pooling [36]. We also implement a variant of PAC that does pixel-adaptive transposed convolution (also called deconvolution) which can be used for learnable guided upsampling of intermediate CNN representations. We discuss more about these generalizations and variants in Sec. 3.

As a result of its simplicity and being a generalization of several widely used filtering techniques, PAC can be useful in a wide range of computer vision problems. In this work, we demonstrate its applicability in three different vision problems. In Sec. 4, we use PAC in joint image upsampling networks and obtain state-of-the-art results on both depth and optical flow upsampling tasks. In Sec. 5, we use PAC in a learnable conditional random field (CRF) framework and observe consistent improvements with respect to the widely used fully-connected CRF [25]. In Sec. 6, we demonstrate how to use PAC as a drop-in replacement of trained convolution layers in a CNN and obtain performance improvements after fine-tuning. In summary, we observe that PAC is highly versatile and has wide applicability in a range of computer vision tasks.

2. Related Work

Image-adaptive filtering. Some important image-adaptive filtering techniques include bilateral filtering [2, 42], guided image filtering [18], non-local means [6, 3], and propagated image filtering [35], to name a few. A common line of research is to make these filters differentiable and use them as content-adaptive CNN layers. Early work [52, 11] in this direction back-propagates through bilateral filtering and can thus leverage fully-connected CRF inference [25] on the output of CNNs. The work of [21] and [13] proposes to use bilateral filtering layers inside CNN architectures. Chandra *et al*. [8] propose a layer that performs closed-form Gaussian CRF inference in a CNN. Chen *et al*. [9] and Liu *et al*. [31] propose differentiable local propagation modules that have roots in domain transform filtering [14]. Wu *et al*. [46] and Wang *et al*. [44] propose neural network layers to perform guided filtering [18] and non-local means [44] respectively inside CNNs. Since these techniques are tailored towards a particular CRF or adaptive filtering technique, they are used for specific tasks and cannot be directly used as a replacement of general convolution. Closest to our work are the sparse, high-dimensional neural networks [21] which generalize standard 2D convolutions to high-dimensional convolutions, enabling them to be content-adaptive. Although conceptually more generic than PAC, such high-dimensional networks can not learn the adapting features and have a larger computational overhead due to the use of specialized lattices and hash tables.

Dynamic filter networks. Introduced by Jia *et al*. [22], dynamic filter networks (DFN) are an example of another class of content-adaptive filtering techniques. Filter weights are themselves directly predicted by a separate network branch, and provide custom filters specific to different input data. The work is later extended by Wu *et al*. [47] with an additional attention mechanism and a dynamic sampling strategy to allow the position-specific kernels to also learn from multiple neighboring regions. Similar ideas have been applied to several task-specific use cases, e.g., motion prediction [48], semantic segmentation [17], and Monte Carlo rendering denoising [4]. Explicitly predicting all position-specific filter weights requires a large number of parameters, so DFNs typically require a sensible architecture design and are difficult to scale to multiple dynamic-filter layers. Our approach differs in that PAC reuses spatial filters

just as standard convolution, and only modifies the filters in a position-specific fashion. Dai *et al*. propose deformable convolution [12], which can also produce position-specific modifications to the filters. Different from PAC, the modifications there are represented as *offsets* with an emphasis on learning geometric-invariant features.

Self-attention mechanism. Our work is also related to the self-attention mechanism originally proposed by Vaswani *et al*. for machine translation [43]. Self-attention modules compute the responses at each position while attending to the global context. Thanks to the use of global information, self-attention has been successfully used in several applications, including image generation [51, 34] and video activity recognition [44]. Attending to the whole image can be computationally expensive, and, as a result, can only be afforded on low-dimensional feature maps, e.g., as in [44]. Our layer produces responses that are sensitive to a more local context (which can be alleviated through *dilation*), and is therefore much more efficient.

3. Pixel-Adaptive Convolution

In this section, we start with a formal definition of standard spatial convolution and then explain our generalization of it to arrive at our pixel-adaptive convolution (PAC). Later, we will discuss several variants of PAC and how they are connected to different image filtering techniques. Formally, a spatial convolution of image features $\mathbf{v} = (\mathbf{v}_1, \dots, \mathbf{v}_n), \mathbf{v}_i \in \mathbb{R}^c$ over n pixels and c channels with filter weights $\mathbf{W} \in \mathbb{R}^{c' \times c \times s \times s}$ can be written as

$$\mathbf{v}'_i = \sum_{j \in \Omega(i)} \mathbf{W} \left[\mathbf{p}_i - \mathbf{p}_j\right] \mathbf{v}_j + \mathbf{b} \qquad (1)$$

where $\mathbf{p}_i = (x_i, y_i)^\intercal$ are pixel coordinates, $\Omega(\cdot)$ defines an $s \times s$ convolution window, and $\mathbf{b} \in \mathbb{R}^{c'}$ denotes biases. With a slight abuse of notation, we use $[\mathbf{p}_i - \mathbf{p}_j]$ to denote indexing of the spatial dimensions of an array with 2D spatial offsets. This convolution operation results in a c'-channel output, $\mathbf{v}'_i \in \mathbb{R}^{c'}$, at each pixel i. Eq. 1 highlights how the weights only depend on pixel position and thus are agnostic to image content. In other words, the weights are *spatially shared* and, therefore, *image-agnostic*. As outlined in Sec. 1, these properties of spatial convolutions are limiting: we would like the filter weights $\mathbf{W}$ to be content-adaptive.

One approach to make the convolution operation content-adaptive, rather than only based on pixel locations, is to generalize $\mathbf{W}$ to depend on the pixel features, $\mathbf{f} \in \mathbb{R}^d$:

$$\mathbf{v}'_i = \sum_{j \in \Omega(i)} \mathbf{W} \left(\mathbf{f}_i - \mathbf{f}_j\right) \mathbf{v}_j + \mathbf{b} \qquad (2)$$

where $\mathbf{W}$ can be seen as a high-dimensional filter operating in a d-dimensional feature space. In other words, we can apply Eq. 2 by first projecting the input signal $\mathbf{v}$ into a d-dimensional space, and then performing d-dimensional convolution with $\mathbf{W}$. Traditionally, such high-dimensional filtering is limited to hand-specified filters such as Gaussian filters [1]. Recent work [21] lifts this restriction and proposes a technique to freely parameterize and learn $\mathbf{W}$ in high-dimensional space. Although generic and used successfully in several computer vision applications [21, 20, 39], high-dimensional convolutions have several shortcomings. First, since data projected on a higher-dimensional space is sparse, special lattice structures and hash tables are needed to perform the convolution [1] resulting in considerable computational overhead. Second, it is difficult to learn features $\mathbf{f}$ resulting in the use of hand-specified feature spaces such as position and color features, $\mathbf{f} = (x, y, r, g, b)$. Third, we have to restrict the dimensionality d of features (say, < 10) as the projected input image can become too sparse in high-dimensional spaces. In addition, the advantages that come with spatial sharing of standard convolution are lost with high-dimensional filtering.

Pixel-adaptive convolution. Instead of bringing convolution to higher dimensions, which has the above-mentioned drawbacks, we choose to modify the spatially invariant convolution in Eq. 1 with a spatially varying kernel $K \in \mathbb{R}^{c' \times c \times s \times s}$ that depends on pixel features $\mathbf{f}$:

$$\mathbf{v}'_i = \sum_{j \in \Omega(i)} K\left(\mathbf{f}_i, \mathbf{f}_j\right) \mathbf{W} \left[\mathbf{p}_i - \mathbf{p}_j\right] \mathbf{v}_j + \mathbf{b} \qquad (3)$$

where K is a kernel function that has a fixed parametric form such as Gaussian: $K(\mathbf{f}_i, \mathbf{f}_j) = \exp(-\frac{1}{2}(\mathbf{f}_i - \mathbf{f}_j)^\intercal(\mathbf{f}_i - \mathbf{f}_j))$. Since K has a pre-defined form and is not parameterized as a high-dimensional filter, we can perform this filtering on the 2D grid itself without moving onto higher dimensions. We call the above filtering operation (Eq. 3) as "Pixel-Adaptive Convolution" (PAC) because the standard spatial convolution $\mathbf{W}$ is adapted at each pixel using pixel features $\mathbf{f}$ via kernel K. We call these pixel features $\mathbf{f}$ as "adapting features" and the kernel K as "adapting kernel". The adapting features $\mathbf{f}$ can be either hand-specified such as position and color features $\mathbf{f} = (x, y, r, g, b)$ or can be deep features that are learned end-to-end.

Generalizations. PAC, despite being a simple modification to standard convolution, generalizes several widely used filtering operations, including

- *Spatial Convolution* can be seen as a special case of PAC with adapting kernel being constant $K(\mathbf{f}_i, \mathbf{f}_j) = 1$. This can be achieved by using constant adapting features, $\mathbf{f}_i = \mathbf{f}_j, \forall i, j$. In brief, standard convolution (Eq. 1) uses fixed, spatially shared filters, while PAC allows the filters to be modified by the adapting kernel K differently across pixel locations.
- *Bilateral Filtering* [42] is a basic image processing operation that has found wide-ranging uses [33] in image processing, computer vision and also computer

graphics. Standard bilateral filtering operation can be seen as a special case of PAC, where $\mathbf{W}$ also has a fixed parametric form, such as a 2D Gaussian filter, $\mathbf{W}\left[\mathbf{p}_i - \mathbf{p}_j\right] = \exp(-\frac{1}{2}(\mathbf{p}_i - \mathbf{p}_j)^\intercal \Sigma^{-1}(\mathbf{p}_i - \mathbf{p}_j))$.

- *Pooling* operations can also be modeled by PAC. Standard average pooling corresponds to the special case of PAC where $K(\mathbf{f}_i, \mathbf{f}_j) = 1$, $\mathbf{W} = \frac{1}{s^2} \cdot \mathbf{1}$. *Detail Preserving Pooling* [36, 45] is a recently proposed pooling layer that is useful to preserve high-frequency details when performing pooling in CNNs. PAC can model the detail-preserving pooling operations by incorporating an adapting kernel that emphasizes more distinct pixels in the neighborhood, e.g., $K(\mathbf{f}_i, \mathbf{f}_j) = \alpha + \left(|\mathbf{f}_i - \mathbf{f}_j|^2 + \epsilon^2\right)^\lambda$.

The above generalizations show the generality and the wide applicability of PAC in different settings and applications. We experiment using PAC in three different problem scenarios, which will be discussed in later sections.

Some filtering operations are even more general than the proposed PAC. Examples include high-dimensional filtering shown in Eq. 2 and others such as dynamic filter networks (DFN) [22] discussed in Sec. 2. Unlike most of those general filters, PAC allows efficient learning and reuse of spatially invariant filters because it is a direct modification of standard convolution filters. PAC offers a good trade-off between standard convolution and DFNs. In DFNs, filters are solely generated by an auxiliary network and different auxiliary networks or layers are required to predict kernels for different dynamic-filter layers. PAC, on the other hand, uses learned pixel embeddings $\mathbf{f}$ as adapting features, which can be reused across several different PAC layers in a network. When related to sparse high-dimensional filtering in Eq. 2, PAC can be seen as factoring the high-dimensional filter into a product of standard spatial filter $\mathbf{W}$ and the adapting kernel K. This allows efficient implementation of PAC in 2D space alleviating the need for using hash tables and special lattice structures in high dimensions. PAC can also use learned pixel embeddings $\mathbf{f}$ instead of hand-specified ones in existing learnable high-dimensional filtering techniques such as [21].

Implementation and variants. We implemented PAC as a network layer in PyTorch with GPU acceleration[1]. Our implementation enables back-propagation through the features $\mathbf{f}$, permitting the use of learnable deep features as adapting features. We also implement a PAC variant that does pixel-adaptive transposed convolution (also called "deconvolution"). We refer to pixel-adaptive convolution shown in Eq. 3 as PAC and the transposed counterpart as PAC$^\intercal$. Similar to standard transposed convolution, PAC$^\intercal$ uses fractional striding and results in an upsampled output. Our PAC and PAC$^\intercal$ implementations allow easy and flexible specification of different options that are commonly used in standard convolution: filter size, number of input and output channels, striding, padding and dilation factor.

[1]Code will be available at https://suhangpro.github.io/pac/

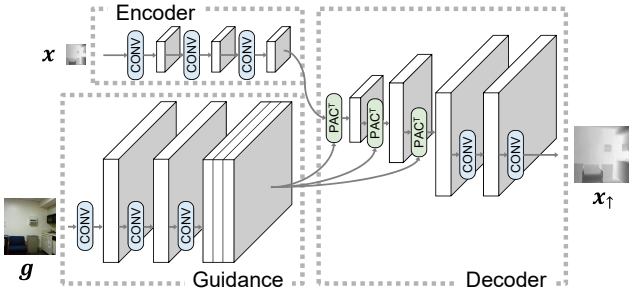

Figure 2: **Joint upsampling with PAC.** Network architecture showing encoder, guidance and decoder components. Features from the guidance branch are used to adapt PAC$^\intercal$ kernels that are applied on the encoder output resulting in upsampled signal.

4. Deep Joint Upsampling Networks

Joint upsampling is the task of upsampling a low-resolution signal with the help of a corresponding high-resolution guidance image. An example is upsampling a low-resolution depth map given a corresponding high-resolution RGB image as guidance. Joint upsampling is useful when some sensors output at a lower resolution than cameras, or can be used to speed up computer vision applications where full-resolution results are expensive to produce. PAC allows filtering operations to be guided by the adapting features, which can be obtained from a separate guidance image, making it an ideal choice for joint image processing. We investigate the use of PAC for joint upsampling applications. In this section, we introduce a network architecture that relies on PAC for deep joint upsampling, and show experimental results on two applications: joint depth upsampling and joint optical flow upsampling.

4.1. Deep joint upsampling with PAC

A deep joint upsampling network takes two inputs, a low-resolution signal $\mathbf{x} \in \mathbb{R}^{c \times h/m \times w/m}$ and a high-resolution guidance $\mathbf{g} \in \mathbb{R}^{c_g \times h \times w}$, and outputs upsampled signal $\mathbf{x}_\uparrow \in \mathbb{R}^{c \times h \times w}$. Here m is the required upsampling factor. Similar to [27], our upsampling network has three components (as illustrated in Fig. 2):

- *Encoder* branch operates directly on the low-resolution signal with convolution (CONV) layers.
- *Guidance* branch operates solely on the guidance image, and generates adapting features that will be used in all PAC$^\intercal$ layers later in the network.
- *Decoder* branch starts with a sequence of PAC$^\intercal$, which perform transposed pixel-adaptive convolution, each of which upsamples the feature maps by a factor of 2. PAC$^\intercal$ layers are followed by two CONV layers to generate the final upsampled output.

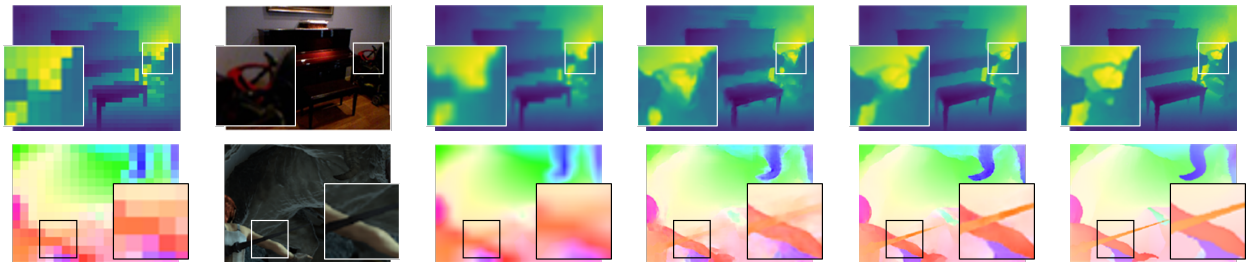

Figure 3: **Deep joint upsampling.** Results of different methods for 16× joint depth upsampling (top row) and 16× joint optical flow upsampling (bottom row). Our method produces results that have more details and are more faithful to the edges in the guidance image.

Each of the CONV and PAC$^\intercal$ layers, except the final one, is followed by a rectified linear unit (ReLU).

4.2. Joint depth upsampling

Here, the task is to upsample a low-resolution depth by using a high-resolution RGB image as guidance. We experiment with the NYU Depth V2 dataset [38], which has 1449 RGB-depth pairs. Following [27], we use the first 1000 samples for training and the rest for testing. The low-resolution depth maps are obtained from the ground-truth depth maps using nearest-neighbor downsampling. Tab. 1 shows root mean square error (RMSE) of different techniques and for different upsampling factors m (4×, 8×, 16×). Results indicate that our network outperforms others in comparison and obtains state-of-the-art performance. Sample visual results are shown in Fig. 3.

We train our network with the Adam optimizer using a learning rate schedule of [$10^{-4}\times$ 3.5k, $10^{-5}\times$ 1.5k, $10^{-6}\times$ 0.5k] and with mini-batches of 256×256 crops. We found this training setup to be superior to the one recommended in DJF [27], and also compare with our own implementation of it under such a setting ("DJF (Our impl.)" in Tab. 1). We keep the network architecture similar to that of previous state-of-the-art technique, DJF [27]. In DJF, features from the guidance branch are simply concatenated with encoder outputs for upsampling, whereas we use guidance features to adapt PAC$^\intercal$ kernels. Although with similar number of layers, our network has more parameters compared with DJF (see appendix for details). We also trained a lighter version of our network ("Ours-lite") that matches the number of parameters of DJF, and still observe better performance showing the importance of PAC$^\intercal$ for upsampling.

Table 1: **Joint depth upsampling.** Results (in RMSE) show that our upsampling network consistently outperforms other techniques for different upsampling factors.

Method	4×	8×	16×
Bicubic	8.16	14.22	22.32
MRF	7.84	13.98	22.20
GF [18]	7.32	13.62	22.03
JBU [24]	4.07	8.29	13.35
Ham *et al*. [15]	5.27	12.31	19.24
DMSG [19]	3.78	6.37	11.16
FBS [5]	4.29	8.94	14.59
DJF [27]	3.54	6.20	10.21
DJF+ [28]	3.38	5.86	10.11
DJF (Our impl.)	2.64	5.15	9.39
Ours-lite	2.55	4.82	8.52
Ours	**2.39**	**4.59**	**8.09**

4.3. Joint optical flow upsampling

We also evaluate our joint upsampling network for upsampling low-resolution optical flow using the original RGB image as guidance. Estimating optical flow is a challenging task, and even recent state-of-the-art approaches [40] resort to simple bilinear upsampling to predict optical flow at the full resolution. Optical flow is smoothly varying within motion boundaries, where accompanying RGB images can offer strong clues, making joint upsampling an appealing solution. We use the same network architecture as in the depth upsampling experiments, with the only difference being that instead of single-channel depth, input and output are two-channel flow with u, v components. We experiment with the Sintel dataset [7] (clean pass). The same training protocol in Sec. 4.2 is used, and the low-resolution optical flow is obtained from bilinear downsampling of the ground-truth. We compare with baselines of bilinear interpolation and DJF [27], and observe consistent advantage (Tab. 2). Fig. 3 shows a sample visual result indicating that our network is capable of restoring fine-structured details and also produces smoother predictions in areas with uniform motion.

Table 2: **Joint optical flow upsampling.** End-Point-Error (EPE) showing the improved performance compared with DJF [27].

	4×	8×	16×
Bilinear	0.465	0.901	1.628
DJF [27]	0.176	0.438	1.043
Ours	**0.105**	**0.256**	**0.592**

5. Conditional Random Fields

Early adoptions of CRFs in computer vision tasks were limited to region-based approaches and short-range structures [37] for efficiency reasons. Fully-Connected CRF (Full-CRF) [25] was proposed to offer the benefits of dense pairwise connections among pixels, which resorts to approximate high-dimensional filtering [1] for efficient inference. Consider a semantic labeling problem, where each pixel i in an image I can take one of the semantic labels $l_i \in \{1, ..., \mathcal{L}\}$. Full-CRF has unary potentials usually defined by a classifier such as CNN: $\psi_u(l_i) \in \mathbb{R}^{\mathcal{L}}$. And, the pairwise potentials are defined for every pair of pixel locations (i, j): $\psi_p(l_i, l_j|I) = \mu(l_i, l_j)K(\mathbf{f}_i, \mathbf{f}_j)$, where K is a kernel function and μ is a compatibility function. A common choice for μ is the Potts model: $\mu(l_i, l_j) = [l_i \neq l_j]$. [25] utilizes two Gaussian kernels with hand-crafted features as the kernel function:

$$K(\mathbf{f}_i, \mathbf{f}_j) = w_1 \exp\left\{-\frac{\|\mathbf{p}_i - \mathbf{p}_j\|^2}{2\theta_\alpha^2} - \frac{\|I_i - I_j\|^2}{2\theta_\beta^2}\right\} + w_2 \exp\left\{-\frac{\|\mathbf{p}_i - \mathbf{p}_j\|^2}{2\theta_\gamma^2}\right\} \quad (4)$$

where $w_1, w_2, \theta_\alpha, \theta_\beta, \theta_\gamma$ are model parameters, and are typically found by a grid-search. Then, inference in Full-CRF amounts to maximizing the following Gibbs distribution: $P(\mathbf{l}|I) = \exp(-\sum_i \psi_u(l_i) - \sum_{i<j} \psi_p(l_i, l_j))$, $\mathbf{l} = (l_1, l_2, ..., l_n)$. Exact inference of Full-CRF is hard, and [25] relies on mean-field approximation which is optimizing for an approximate distribution $Q(\mathbf{l}) = \prod_i Q_i(l_i)$ by minimizing the KL-divergence between $P(\mathbf{l}|I)$ and the mean-field approximation $Q(\mathbf{l})$. This leads to the following mean-field (MF) inference step that updates marginal distributions Q_i iteratively for $t = 0, 1, ...$:

$$Q_i^{(t+1)}(l) \leftarrow \frac{1}{Z_i} \exp\left\{ -\psi_u(l) - \sum_{l' \in \mathcal{L}} \mu(l, l') \sum_{j \neq i} K(\mathbf{f}_i, \mathbf{f}_j) Q_j^{(t)}(l') \right\} \quad (5)$$

The main computation in each MF iteration, $\sum_{j \neq i} K(\mathbf{f}_i, \mathbf{f}_j) Q_j^{(t)}$, can be viewed as high-dimensional Gaussian filtering. Previous work [25, 26] relies on permutohedral lattice convolution [1] to achieve efficient implementation.

5.1. Efficient, learnable CRF with PAC

Existing work [52, 21] back-propagates through the above MF steps to combine CRF inference with CNNs resulting in end-to-end training of CNN-CRF models. While there exists optimized CPU implementations, permutohedral lattice convolution cannot easily utilize GPUs because it "does not follow the SIMD paradigm of efficient GPU computation" [41]. Another drawback of relying on permutohedral lattice convolution is the approximation error incurred during both inference and gradient computation.

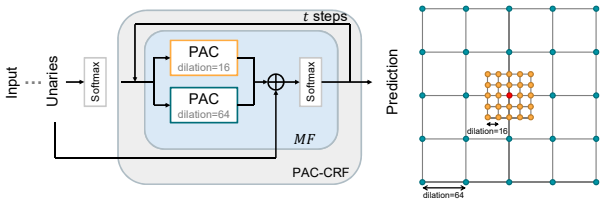

Figure 4: **PAC-CRF.** Illustration of inputs, outputs and the operations in each mean-field (MF) step of PAC-CRF inference. Also shown is the coverage of two 5×5 PAC filters, with dilation factors 16 and 64 respectively.

We propose PAC-CRF, which alleviates these computation issues by relying on PAC for efficient inference, and is easy to integrate with existing CNN backbones. PAC-CRF also has additional learning capacity, which leads to better performance compared with Full-CRF in our experiments.

PAC-CRF. In PAC-CRF, we define pairwise connections over fixed windows Ω^k around each pixel instead of dense connections: $\sum_k \sum_i \sum_{j \in \Omega^k(i)} \psi_p^k(l_i, l_j|I)$, where the k-th pairwise potential is defined as

$$\psi_p^k(l_i, l_j|I) = K^k(\mathbf{f}_i, \mathbf{f}_j)\mathbf{W}_{l_j l_i}^k[\mathbf{p}_j - \mathbf{p}_i] \quad (6)$$

Here $\Omega^k(\cdot)$ specifies the pairwise connection pattern of the k-th pairwise potential originated from each pixel, and K^k is a fixed Gaussian kernel. Intuitively, this formulation allows the label compatibility transform μ in Full-CRF to be modeled by $\mathbf{W}$, and to vary across different spatial offsets. Similar derivation as in Full-CRF yields the following iterative MF update rule (see appendix for more details):

$$Q_i^{(t+1)}(l) \leftarrow \frac{1}{Z_i} \exp\Bigg\{ -\psi_u(l) - \underbrace{\sum_k \sum_{l' \in \mathcal{L}} \sum_{j \in \Omega^k(i)} K^k(\mathbf{f}_i, \mathbf{f}_j)\mathbf{W}_{l'l}^k[\mathbf{p}_j - \mathbf{p}_i]Q_j^{(t)}(l')}_{\text{PAC}} \Bigg\} \quad (7)$$

MF update now consists of PAC instead of sparse high-dimensional filtering as in Full-CRF (Eq. 5). As outlined in Sec. 2, there are several advantages of PAC over high-dimensional filtering. With PAC-CRF, we can freely parameterize and learn the pairwise potentials in Eq. 6 that also use a richer form of compatibility transform $\mathbf{W}$. PAC-CRF can also make use of learnable features $\mathbf{f}$ for pairwise potentials instead of pre-defined ones in Full-CRF. Fig. 4 (left) illustrates the computation steps in each MF step with two pairwise PAC kernels.

Long-range connections with dilated PAC. The major source of heavy computation in Full-CRF is the dense pair-

wise pixel connections. In PAC-CRF, the pairwise connections are defined by the local convolution windows Ω^k. To have long-range pairwise connections while keeping the number of PAC parameters managable, we make use of dilated filters [10, 49]. Even with a relatively small kernel size (5×5), with a large dilation, e.g., 64, the CRF can effectively reach a neighborhood of 257×257. A concurrent work [41] also propose a convolutional version of CRF (Conv-CRF) to reduce the number of connections in Full-CRF. However, [41] uses connections only within small local windows. We argue that long-range connections can provide valuable information, and our CRF formulation uses a wider range of connections while still being efficient. Our formulation allows using multiple PAC filters in parallel, each with different dilation factors. In Fig. 4 (right), we show an illustration of the coverage of two 5×5 PAC filters, with dilation factors 16 and 64 respectively. This allows PAC-CRF to achieve a good trade-off between computational efficiency and long-range pairwise connectivity.

5.2. Semantic segmentation with PAC-CRF

The task of semantic segmentation is to assign a semantic label to each pixel in an image. Full-CRF is proven to be a valuable post-processing tool that can considerably improve CNN segmentation performance [10, 52, 21]. Here, we experiment with PAC-CRF on top of the FCN semantic segmentation network [32]. We choose FCN for simplicity and ease of comparisons, as FCN only uses standard convolution layers and does not have many bells and whistles.

In the experiments, we use scaled RGB color, $[\frac{R}{\sigma_R}, \frac{G}{\sigma_G}, \frac{B}{\sigma_B}]^\intercal$, as the guiding features for the PAC layers in PAC-CRF . The scaling vector $[\sigma_R, \sigma_G, \sigma_B]^\intercal$ is learned jointly with the PAC weights $\mathbf{W}$. We try two internal configurations of PAC-CRF: a single 5×5 PAC kernel with dilation of 32, and two parallel 5×5 PAC kernels with dilation factors of 16 and 64. 5 MF steps are used for a good balance between speed and accuracy (more details in appendix). We first freeze the backbone FCN network and train only the PAC-CRF part for 40 epochs, and then train the whole network for another 40 epochs with reduced learning rates.

Dataset. We follow the training and validation settings of FCN [32] which is trained on PascalVOC images and validated on a reduced validation set of 736 images. We also submit our final trained models to the official evaluation server to get test scores on 1456 test images.

Baselines. We compare PAC-CRF with three baselines: Full-CRF [25], BCL-CRF [21], and Conv-CRF [41]. For Full-CRF, we use the publicly available C++ code, and find the optimal CRF parameters through grid search. For BCL-CRF, we use 1-neighborhood filters to keep the runtime manageable and use other settings as suggested by the authors. For Conv-CRF, the same training procedure is used as in PAC-CRF. We use the more powerful variant of Conv-CRF with learnable compatibility transform (referred to as "Conv+C" in [41]), and we learn the RGB scales for Conv-CRF in the same way as for PAC-CRF. We follow the suggested default settings for Conv-CRF and use a filter size of 11×11 and a blurring factor of 4. Note that like Full-CRF (Eq. 4), the other baselines also use two pairwise kernels.

Table 3: **Semantic segmentation with PAC-CRF.** Validation and test mIoU scores along with the runtimes of different techniques. PAC-CRF results in better improvements than Full-CRF [25] while being faster. PAC-CRF also outperforms Conv-CRF [41] and BCL [21]. Runtimes are averaged over all validation images.

Method	mIoU (val / test)	CRF Runtime
Unaries only (FCN)	65.51 / 67.20	-
Full-CRF [25]	+2.11 / +2.45	629 ms
BCL-CRF [21]	+2.28 / +2.33	2.6 s
Conv-CRF [41]	+2.13 / +1.57	38 ms
PAC-CRF, 32	+3.01 / +2.21	39 ms
PAC-CRF, 16-64	**+3.39 / +2.62**	78 ms

Results. Tab. 3 reports validation and test mean Intersection over Union (mIoU) scores along with average runtimes of different techniques. Our two-filter variant ("PAC-CRF, 16-64") achieves better mIoU compared with all baselines, and also compares favorably in terms of runtime. The one-filter variant ("PAC-CRF, 32") performs slightly worse than Full-CRF and BCL-CRF, but has even larger speed advantage, offering a strong option where efficiency is needed. Sample visual results are shown in Fig. 5. While being quantitatively better and retaining more visual details overall, PAC-CRF produces some amount of noise around boundaries. This is likely due to a known "gridding" effect of dilation [50], which we hope to mitigate in future work.

6. Layer hot-swapping with PAC

So far, we design specific architectures around PAC for different use cases. In this section, we offer a strategy to use PAC for simply upgrading existing CNNs with minimal modifications through what we call layer *hot-swapping*.

Layer hot-swapping. Network fine-tuning has become a common practice when training networks on new data or with additional layers. Typically, in fine-tuning, newly added layers are initialized randomly. Since PAC generalizes standard convolution layers, it can directly replace convolution layers in existing networks while retaining the pre-trained weights. We refer to this modification of existing pre-trained networks as layer *hot-swapping*.

We continue to use semantic segmentation as an example, and demonstrate how layer hot-swapping can be a simple yet effective modification to existing CNNs. Fig. 6 illustrates a FCN [32] before and after the hot-swapping modifications. We swap out the last CONV layer of the last three convolution groups, CONV3_3, CONV4_3, CONV5_3,

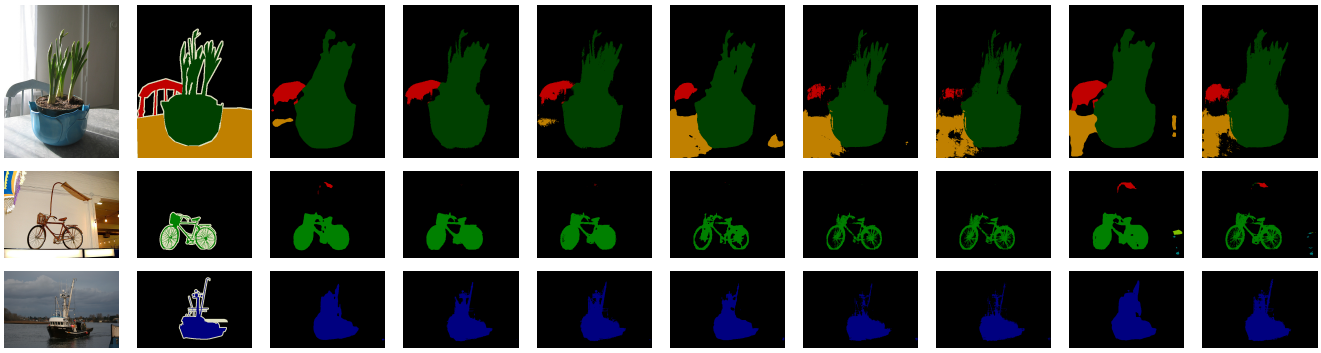

Figure 5: **Semantic segmentation with PAC-CRF and PAC-FCN.** We show three examples from the validation set. Compared to Full-CRF [25], BCL-CRF [21], and Conv-CRF [41], PAC-CRF can recover finer details faithful to the boundaries in the RGB inputs.

with PAC layers with the same configuration (filter size, input and output channels, *etc*.), and use the output of CONV2_2 as the guiding feature for the PAC layers. By this example, we also demonstrate that one could use earlier layer features (CONV2_2 here) as adapting features for PAC. Using this strategy, the network parameters do not increase when replacing CONV layers with PAC layers. All the layer weights are initialized with trained FCN parameters. To ensure a better starting condition for further training, we scale the guiding features by a small constant (0.0001) so that the PAC layers initially behave very closely to their original CONV counterparts. We use 8825 images for training, including the Pascal VOC 2011 training images and the additional training samples from [16]. Validation and testing are performed in the same fashion as in Sec. 5.

Results are reported in Tab. 4. We show that our simple modification (PAC-FCN) provides about 2 mIoU improvement on test (67.20 $\rightarrow$ 69.18) for the semantic segmentation task, while incurring virtually no runtime penalty at inference time. Note that PAC-FCN has the same number of parameters as the original FCN model. The improvement brought by PAC-FCN is also complementary to any additional CRF post-processing that can still be applied. After combined with a PAC-CRF (the 16-64 variant) and trained jointly, we observe another 2 mIoU improvement. Sample visual results are shown in Fig. 5.

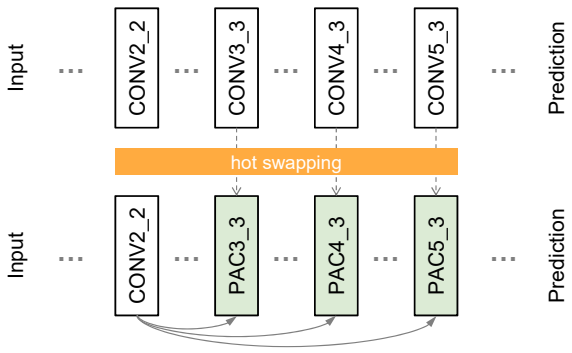

Figure 6: **Layer hot-swapping with PAC.** A few layers of a network before (top) and after (bottom) hot-swapping. Three CONV layers are replaced with PAC layers, with adapting features coming from an earlier convolution layer. All the original network weights are retained after the modification.

Table 4: **FCN hot-swapping CONV with PAC.** Validation and test mIoU scores along with runtimes of different techniques. Our simple hot-swapping strategy provides 2 IoU gain on test. Combining with PAC-CRF offers additional improvements.

Method	PAC-CRF	mIoU (val / test)	Runtime
FCN-8s	-	65.51 / 67.20	39 ms
FCN-8s	16-64	68.90 / 69.82	117 ms
PAC-FCN	-	67.44 / 69.18	41 ms
PAC-FCN	16-64	**69.87 / 71.34**	118 ms

7. Conclusion

In this work we propose PAC, a new type of filtering operation that can effectively learn to leverage guidance information. We show that PAC generalizes several popular filtering operations and demonstrate its applicability on different uses ranging from joint upsampling, semantic segmentation networks, to efficient CRF inference. PAC generalizes standard spatial convolution, and can be used to directly replace standard convolution layers in pre-trained networks for performance gain with minimal computation overhead.

Acknowledgements Hang Su and Erik Learned-Miller acknowledge support from AFRL and DARPA (#FA8750-18-2-0126)[2] and the MassTech Collaborative grant for funding the UMass GPU cluster.

[2]The U.S. Gov. is authorized to reproduce and distribute reprints for Gov. purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the AFRL and DARPA or the U.S. Gov.

References

[1] A. Adams, J. Baek, and M. A. Davis. Fast high-dimensional filtering using the permutohedral lattice. *Computer Graphics Forum*, 29(2):753–762, 2010. 3, 6

[2] V. Aurich and J. Weule. Non-linear Gaussian filters performing edge preserving diffusion. In *DAGM*, pages 538–545. Springer, 1995. 2

[3] S. P. Awate and R. T. Whitaker. Higher-order image statistics for unsupervised, information-theoretic, adaptive, image filtering. In *Proc. CVPR*, volume 2, pages 44–51. IEEE, 2005. 2

[4] S. Bako, T. Vogels, B. McWilliams, M. Meyer, J. Novák, A. Harvill, P. Sen, T. Derose, and F. Rousselle. Kernel-predicting convolutional networks for denoising monte carlo renderings. *ACM Trans. Graph.*, 36(4):97, 2017. 2

[5] J. T. Barron and B. Poole. The fast bilateral solver. In *Proc. ECCV*, pages 617–632. Springer, 2016. 5

[6] A. Buades, B. Coll, and J.-M. Morel. A non-local algorithm for image denoising. In *Proc. CVPR*, volume 2, pages 60–65. IEEE, 2005. 2

[7] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. In A. Fitzgibbon et al. (Eds.), editor, *Proc. ECCV*, Part IV, LNCS 7577, pages 611–625. Springer-Verlag, Oct. 2012. 5

[8] S. Chandra and I. Kokkinos. Fast, exact and multi-scale inference for semantic image segmentation with deep Gaussian CRFs. In *Proc. ECCV*, pages 402–418. Springer, 2016. 2

[9] L.-C. Chen, J. T. Barron, G. Papandreou, K. Murphy, and A. L. Yuille. Semantic image segmentation with task-specific edge detection using CNNs and a discriminatively trained domain transform. In *Proc. CVPR*, pages 4545–4554, 2016. 2

[10] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs. *PAMI*, 40(4):834–848, 2018. 7

[11] L.-C. Chen, A. Schwing, A. Yuille, and R. Urtasun. Learning deep structured models. In *Proc. ICML*, pages 1785–1794, 2015. 2

[12] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei. Deformable convolutional networks. *arXiv:1703.06211*, 1(2):3, 2017. 2, 3

[13] R. Gadde, V. Jampani, M. Kiefel, D. Kappler, and P. V. Gehler. Superpixel convolutional networks using bilateral inceptions. In *Proc. ECCV*, pages 597–613. Springer, 2016. 2

[14] E. S. Gastal and M. M. Oliveira. Domain transform for edge-aware image and video processing. *ACM Trans. Graph.*, 30(4):69, 2011. 2

[15] B. Ham, M. Cho, and J. Ponce. Robust image filtering using joint static and dynamic guidance. In *Proc. CVPR*, pages 4823–4831, 2015. 5

[16] B. Hariharan, P. Arbelaez, L. Bourdev, S. Maji, and J. Malik. Semantic contours from inverse detectors. In *Proc. ICCV*, 2011. 8

[17] A. W. Harley, K. G. Derpanis, and I. Kokkinos. Segmentation-aware convolutional networks using local attention masks. In *Proc. ICCV*, volume 2, page 7, 2017. 2

[18] K. He, J. Sun, and X. Tang. Guided image filtering. *PAMI*, 35(6):1397–1409, 2013. 2, 5

[19] T.-W. Hui, C. C. Loy, and X. Tang. Depth map super-resolution by deep multi-scale guidance. In *Proc. ECCV*, pages 353–369. Springer, 2016. 5

[20] V. Jampani, R. Gadde, and P. V. Gehler. Video propagation networks. In *Proc. CVPR*, 2017. 3

[21] V. Jampani, M. Kiefel, and P. V. Gehler. Learning sparse high dimensional filters: Image filtering, dense CRFs and bilateral neural networks. In *Proc. CVPR*, pages 4452–4461, 2016. 2, 3, 4, 6, 7, 8

[22] X. Jia, B. De Brabandere, T. Tuytelaars, and L. V. Gool. Dynamic filter networks. In *Proc. NIPS*, pages 667–675, 2016. 2, 4

[23] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009. 12

[24] J. Kopf, M. F. Cohen, D. Lischinski, and M. Uyttendaele. Joint bilateral upsampling. *ACM Trans. Graph.*, 26(3):96, 2007. 5

[25] P. Krähenbühl and V. Koltun. Efficient inference in fully connected CRFs with Gaussian edge potentials. In *Proc. NIPS*, pages 109–117, 2011. 2, 6, 7, 8

[26] P. Krähenbühl and V. Koltun. Parameter learning and convergent inference for dense random fields. In *Proc. ICML*, pages 513–521, 2013. 6

[27] Y. Li, J.-B. Huang, N. Ahuja, and M.-H. Yang. Deep joint image filtering. In *Proc. ECCV*, pages 154–169. Springer, 2016. 4, 5, 11, 13

[28] Y. Li, J.-B. Huang, N. Ahuja, and M.-H. Yang. Joint image filtering with deep convolutional networks. *PAMI*, 2018. 5

[29] Y. Li and R. Zemel. Mean field networks. *arXiv:1410.5884*, 2014. 2

[30] G. Lin, C. Shen, A. Van Den Hengel, and I. Reid. Efficient piecewise training of deep structured models for semantic segmentation. In *Proc. CVPR*, pages 3194–3203, 2016. 2

[31] S. Liu, S. D. Mello, J. Gu, G. Zhong, M.-H. Yang, and J. Kautz. Learning affinity via spatial propagation networks. In *Proc. NIPS*, 2017. 2

[32] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proc. CVPR*, pages 3431–3440, 2015. 7

[33] S. Paris, P. Kornprobst, J. Tumblin, F. Durand, et al. Bilateral filtering: Theory and applications. *Foundations and Trends® in Computer Graphics and Vision*, 4(1):1–73, 2009. 3

[34] N. Parmar, A. Vaswani, J. Uszkoreit, Ł. Kaiser, N. Shazeer, and A. Ku. Image transformer. *arXiv:1802.05751*, 2018. 3

[35] J.-H. Rick Chang and Y.-C. Frank Wang. Propagated image filtering. In *Proc. CVPR*, pages 10–18, 2015. 2

[36] F. Saeedan, N. Weber, M. Goesele, and S. Roth. Detail-preserving pooling in deep networks. In *cvpr*, pages 9108–9116, 2018. 2, 4

[37] J. Shotton, J. Winn, C. Rother, and A. Criminisi. Textonboost: Joint appearance, shape and context modeling for

multi-class object recognition and segmentation. In *Proc. ECCV*, 2006. 6

[38] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor segmentation and support inference from rgbd images. In *Proc. ECCV*, pages 746–760. Springer, 2012. 5

[39] H. Su, V. Jampani, D. Sun, S. Maji, E. Kalogerakis, M.-H. Yang, and J. Kautz. Splatnet: Sparse lattice networks for point cloud processing. In *Proc. CVPR*, 2018. 3

[40] D. Sun, X. Yang, M.-Y. Liu, and J. Kautz. PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume. In *Proc. CVPR*, pages 8934–8943, 2018. 5

[41] M. T. T. Teichmann and R. Cipolla. Convolutional CRFs for semantic segmentation. *arXiv:1805.04777*, 2018. 6, 7, 8

[42] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Proc. ICCV*, 1998. 2, 3

[43] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Proc. NIPS*, pages 5998–6008, 2017. 3

[44] X. Wang, R. Girshick, A. Gupta, and K. He. Non-local neural networks. In *Proc. CVPR*, 2018. 2, 3

[45] N. Weber, M. Waechter, S. C. Amend, S. Guthe, and M. Goesele. Rapid, detail-preserving image downscaling. *ACM Trans. Graph.*, 35(6):205, 2016. 4

[46] H. Wu, S. Zheng, J. Zhang, and K. Huang. Fast end-to-end trainable guided filter. In *Proc. CVPR*, pages 1838–1847, 2018. 2

[47] J. Wu, D. Li, Y. Yang, C. Bajaj, and X. Ji. Dynamic sampling convolutional neural networks. *arXiv:1803.07624*, 2018. 2

[48] T. Xue, J. Wu, K. Bouman, and B. Freeman. Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks. In *Proc. NIPS*, pages 91–99, 2016. 2

[49] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv:1511.07122*, 2015. 7

[50] F. Yu, V. Koltun, and T. Funkhouser. Dilated residual networks. In *Proc. CVPR*, pages 472–480, 2017. 7

[51] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena. Self-attention generative adversarial networks. *arXiv:1805.08318*, 2018. 3

[52] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. Torr. Conditional random fields as recurrent neural networks. In *Proc. ICCV*, pages 1529–1537, 2015. 2, 6, 7

Appendix

In the appendix, we provide additional details and results on the deep joint upsampling experiments (Sec. A) and PAC-CRF (Sec. B).

A. Deep Joint Upsampling with PAC

Network architecture. Here we provide details of our network architectures used in the joint upsampling experiments. Our networks have three branches: Encoder, Guidance, and Decoder. The layers in each branch of the joint depth upsampling networks are listed in Tab. 5. Since we use each PAC$^\intercal$ for 2× upsampling, 4×, 8×, 16× networks requires 2, 3, 4 PAC$^\intercal$ layers respectively. The final output from the guidance branch is equally divided in the channel dimension for use as adapting features for the PAC$^\intercal$ layers in the decoder. All CONV and PAC$^\intercal$ layers use 5×5 filters, and are followed by ReLU except for the last CONV. We use Gaussian kernels for K in all PAC$^\intercal$ layers.

We design two variants of our model, *standard* and *lite*. The *standard* variant has a simpler design, but has varying number of parameters for different upsampling factors, and overall consume more memory than DJF [27], a previous state-of-the-art approach on joint depth upsampling. For the *lite* variant, we reduce the number of filters and make sure the networks roughly match the number of parameters compared to DJF.

Similar network architectures are also used for optical flow upsampling. First layer of encoder and last layer in decoder are modified to fit the two (u, v) channels in optical flow instead of one channel in depth maps, i.e., using "C2" instead of "C1" in Tab. 5.

	standard			lite		
	4×	8×	16×	4×	8×	16×
Encoder	C32 C32 C32	C32 C32 C32	C32 C32 C32	C12 C16 C22	C12 C16 C16	C8 C16 C16
Guidance	C32 C32 C32	C32 C32 C48	C32 C32 C64	C12 C22 C24	C12 C16 C36	C8 C16 C40
Decoder	P32 P32 C32 C1	P32 P32 P32 C32 C1	P32 P32 P32 P32 C32 C1	P12 P16 C22 C1	P12 P16 P16 C20 C1	P8 P16 P16 P16 C16 C1
#Params	183K	222K	260K	56K	56K	56K

Table 5: **Network architectures for joint depth upsampling.** "C" stands for regular CONV, "P" stands for PAC$^\intercal$ (the transposed convolution variant of PAC), and the number after them represents the number of output channels.

Additional examples. We provide more joint upsampling visual results for depth (Fig. 7) and optical flow (Fig. 8).

B. Conditional Random Fields

Interpretations of the formulation. The pairwise potentials in Full-CRF is defined as $\psi_p(l_i, l_j|I) = \mu(l_i, l_j)K(\mathbf{f}_i, \mathbf{f}_j)$, where the kernel function K has two terms, *appearance kernel* and *smoothness kernel*

$$K(\mathbf{f}_i, \mathbf{f}_j) = w_1 \underbrace{\exp\left\{ -\frac{\|\mathbf{p}_i - \mathbf{p}_j\|^2}{2\theta_\alpha^2} - \frac{\|I_i - I_j\|^2}{2\theta_\beta^2} \right\}}_{\text{appearance kernel}} + w_2 \underbrace{\exp\left\{ -\frac{\|\mathbf{p}_i - \mathbf{p}_j\|^2}{2\theta_\gamma^2} \right\}}_{\text{smoothness kernel}} \tag{8}$$

In comparision, our pairwise potential uses (assuming using Guassian kernel and a single pairwise term)

$$\begin{aligned} K'(\mathbf{f}_i, \mathbf{f}_j) &= \mathbf{W}[\mathbf{p}_j - \mathbf{p}_i]K(\mathbf{f}_i, \mathbf{f}_j) \\ &= \mathbf{W}[\mathbf{p}_j - \mathbf{p}_i] \exp\left\{ -\frac{1}{2}\|\mathbf{f}_i - \mathbf{f}_j\|^2 \right\} \end{aligned} \tag{9}$$

There are two major differences:

1. The *smoothness kernel* is now moved out of K and is represented using filter $\mathbf{W}$. It can still be initialized as a Gaussian, but arbitrary filter is allowed to be learned.
2. The *appearance kernel* now operates on $\mathbf{f}$ directly without the need of decomposing it into multiple parts, and without the individual scaling factors $(\theta_\alpha, \dots)$.

Both changes give the pairwise potential more learning capacity. Note that $\mathbf{f}$ can be the output of some other network layers. A simple linear layer can learn appropriate scaling factors, while in other cases a more complex network may be preferred. For input with more than RGB channels (e.g., 3D data with color, depth, normal, curvature, *etc.*), hand-crafting and finding parameters for kernel functions like Eq. 8 can be time-consuming and suboptimal, and allowing the function to be learned from data in an end-to-end fashion is particularly desirable.

Note that in Eq. 9, $\mathbf{W}$ is a 2D matrix, and the corresponding pairwise potential is defined as

$$\psi_p(l_i, l_j) = \mu(l_i, l_j)\mathbf{W}[\mathbf{p}_j - \mathbf{p}_i]K(\mathbf{f}_i, \mathbf{f}_j) \tag{10}$$

where $\mu(l_i, l_j)$ is the compatibility matrix. Our final pairwise potential, $\psi_p(l_i, l_j) = K(\mathbf{f}_i, \mathbf{f}_j)\mathbf{W}_{l_j l_i}[\mathbf{p}_j - \mathbf{p}_i]$, can be seen as a further step of generalization, where $\mathbf{W}$ is now a 4D tensor. Intuitively, this formulation allows the label compatibility pattern to be spatially varying across different pixel locations. Eq. 10 can be seen as a special case factorizing the 4D tensor as the product of two 2D matrices.

Mean-field inference derivation. We will start from the mean-field update equation for general pairwise CRFs, Eq. 11. Detailed derivation for it can be found in Koller and Friedman [23, Chapter 11.5].

$$Q_i(l) = \frac{1}{Z_i} \exp\Big\{ -\psi_u(l) - \sum_{j \in \Omega(i)} \mathbf{E}_{l_j \sim Q_j} \psi_p(l, l_j) \Big\} \tag{11}$$

Considering that we use multiple neighborhoods (with different dilation factors) in parallel, the update equation becomes

$$Q_i(l) = \frac{1}{Z_i} \exp\Big\{ -\psi_u(l) - \sum_k \sum_{j \in \Omega^k(i)} \mathbf{E}_{l_j \sim Q_j} \psi_p^k(l, l_j) \Big\} \tag{12}$$

Substituting the pairwise potential with

$$\psi_p^k(l_i, l_j) = K^k(\mathbf{f}_i, \mathbf{f}_j) \mathbf{W}^k_{l_j l_i} [\mathbf{p}_j - \mathbf{p}_i] \tag{13}$$

the update rule becomes

$$\begin{aligned} Q_i(l) =& \frac{1}{Z_i} \exp\Big\{ -\psi_u(l) \\ & - \sum_k \sum_{j \in \Omega^k(i)} \mathbf{E}_{l_j \sim Q_j} \Big\{ K^k(\mathbf{f}_i, \mathbf{f}_j) \mathbf{W}^k_{l_j l} [\mathbf{p}_j - \mathbf{p}_i] \Big\} \Big\} \\ =& \frac{1}{Z_i} \exp\Big\{ -\psi_u(l) \\ & - \sum_k \sum_{l' \in \mathcal{L}} \sum_{j \in \Omega^k(i)} K^k(\mathbf{f}_i, \mathbf{f}_j) \mathbf{W}^k_{l' l} [\mathbf{p}_j - \mathbf{p}_i] Q_j(l') \Big\} \end{aligned} \tag{14}$$

Using Eq. 14 in an iterative fashion leads to the final update rule of mean-field inference:

$$\begin{aligned} Q_i^{(t+1)}(l) \leftarrow & \frac{1}{Z_i} \exp\Big\{ -\psi_u(l) \\ & - \sum_k \underbrace{\sum_{l' \in \mathcal{L}} \sum_{j \in \Omega^k(i)} K^k(\mathbf{f}_i, \mathbf{f}_j) \mathbf{W}^k_{l' l} [\mathbf{p}_j - \mathbf{p}_i] Q_j^{(t)}(l')}_{\text{PAC}} \Big\} \end{aligned} \tag{15}$$

Mean-field inference steps. Tab. 6 shows how mIoU changes with different mean-field steps. We use 5 steps for all other experiments in the paper.

Table 6: **Impact of MF steps in PAC-CRF.** Validation mIoU when using different number of MF steps in PAC-CRF.

Mean-field steps	1	3	5	7
mIoU	68.38	68.72	**68.90**	68.90
time	19 ms	49 ms	78 ms	109 ms

On the contribution of dilation. Just like standard convolution, PAC supports dilation to increase the receptive field without increasing the number of parameters. This capability is leveraged by PAC-CRF to allow long-range connections. For a similar purpose, Conv-CRF applies Gaussian blur to pairwise potentials to increase the receptive field. To quantify the improvements due to dilation, we try another baseline where we add dilation to Conv-CRF. The improved performance (+2.13/+1.57 $\rightarrow$ +2.50/+1.91) validates that dilation is indeed an important ingredient, while the remaining gap shows that the PAC formulation is essential to the full gain.

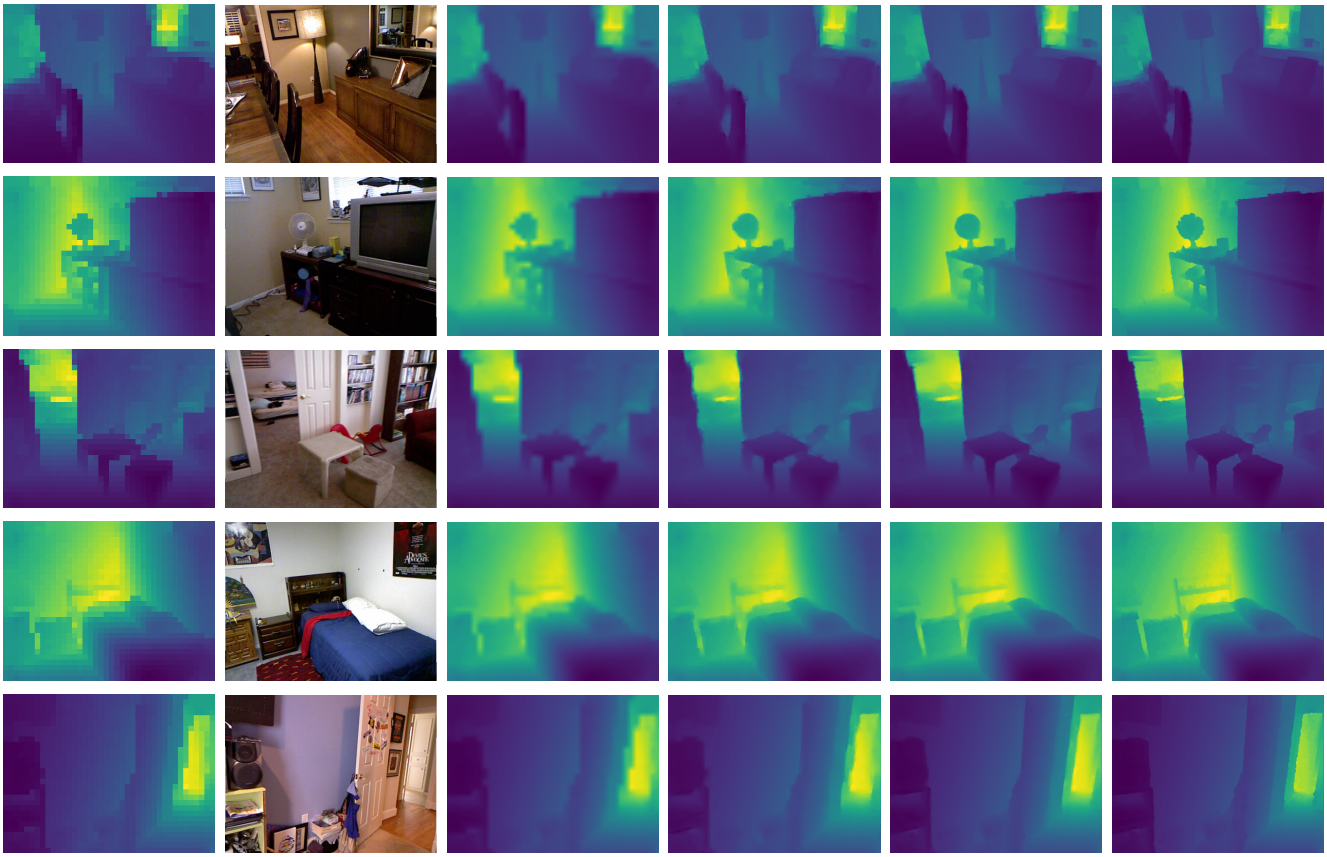

Figure 7: **Additional examples of joint depth upsampling.** Samples are from the test set of NYU Depth V2. Zoom in for full details.

Figure 8: **Additional examples of joint optical flow upsampling.** Samples are from the val set of Sintel. Zoom in for full details.