Probabilistic Pixel-Adaptive Refinement Networks

Anne S. Wannenwetsch\textsuperscript{1,2}\textsuperscript{*}, Stefan Roth\textsuperscript{2}
\textsuperscript{1}Amazon, Germany \textsuperscript{2}TU Darmstadt, Germany

Abstract

Encoder-decoder networks have found widespread use in various dense prediction tasks. However, the strong reduction of spatial resolution in the encoder leads to a loss of location information as well as boundary artifacts. To address this, image-adaptive post-processing methods have shown beneficial by leveraging the high-resolution input image(s) as guidance data. We extend such approaches by considering an important orthogonal source of information: the network’s confidence in its own predictions. We introduce probabilistic pixel-adaptive convolutions (PPACs), which not only depend on image guidance data for filtering, but also respect the reliability of per-pixel predictions. As such, PPACs allow for image-adaptive smoothing and simultaneously propagating pixels of high confidence into less reliable regions, while respecting object boundaries. We demonstrate their utility in refinement networks for optical flow and semantic segmentation, where PPACs lead to a clear reduction in boundary artifacts. Moreover, our proposed refinement step is able to substantially improve the accuracy on various widely used benchmarks.

1. Introduction

Convolutional neural networks (CNNs) have become a standard tool in computer vision. Especially in dense prediction tasks [7, 41, 49, 60], encoder-decoder or pyramid-structured CNNs are a common choice. While originating in unsupervised learning [21], such architectures have become popular also in supervised settings. The encoder builds a powerful feature representation, reducing the spatial resolution of the inputs to aggregate global information [41]. The decoder takes the feature representation from the bottleneck, enlarges its size, and transforms it into the desired output, e.g. a segmentation map or optical flow field.

While downsampling in the encoder increases the receptive field and allows to deal with large image sizes, it also leads to a drastic loss in spatial resolution. As such, valuable location information is lost and boundary artifacts can arise [18, 41], e.g. segmentation maps that are misaligned w.r.t. the underlying objects. Moreover, the decoder typically yields low-resolution outputs and simple components are used to upscale predictions to the input size. This often results in blurry outputs since estimates of different objects are combined, e.g. motion from background and foreground objects is mixed as can be seen in Fig. 1.

Several approaches have been proposed to reduce these disadvantages, e.g. skip connections [41, 49] or densely connected blocks [24, 32]. Additionally, different types of generalized convolutions, taking into account a high-resolution RGB guidance image, have shown beneficial as part of the decoder [25, 26] or in a separate upsampling and/or refinement step [31, 48, 56, 58]. Many of the above approaches require a large number of additional parameters, are computationally expensive, or restrictive w.r.t. to the filtering method and the applicable guidance data. Most recently, pixel-adaptive convolutions (PACs) were introduced by Su et al. [51]. PACs combine spatially-invariant convolution weights with a content-adaptive kernel that depends on guidance data. In [51], PACs are shown to yield state-of-the-art results in joint upsampling tasks.

In this paper, we argue that we can leverage another source of information for refinement that is complementary...
to the input image: the uncertainty of each pixel’s estimate. For dense classification networks, this quantity is generally provided implicitly. For instance, segmentation networks usually output log-probabilities of the classes, which are passed through an arg max operation at test time. Even though these uncertainties might not be well calibrated [16], we argue that they still contain valuable information for refining the predictions. Beyond classification, explicit uncertainty estimates are also gaining increased attention for dense regression problems such as geometry [47] or motion estimation [12, 27, 60]. They are, for example, helpful for applications in which the reliability of network estimates is crucial, e.g. in autonomous driving. Here, we show that we can also leverage them to refine the regression output itself.

Fig. 1 shows an optical flow field estimated by the probabilistic HD3 method [60], as well as the corresponding confidence map and endpoint error per pixel. We observe that regions of high uncertainty (b, dark gray) correspond quite well with large errors (c, dark red). When applying post-processing to the network output, it seems desirable to take the available pixel uncertainty into account. As such, only reliable pixels should be spatially propagated while uncertain pixels can be replaced. To allow for probability-aware filtering, we propose probabilistic pixel-adaptive convolutions (PPACs), and therefore extend the adaptive convolution operation of [51]. The kernels of PPACs and thus the filtering output vary dependent on two properties: guidance data, e.g. the input image, as well as a probability map estimated by the deep network, either inherently or explicitly.

This paper focuses on the application of PPACs for the refinement of outputs from dense prediction networks, illustrated with the tasks of optical flow estimation and semantic segmentation. Therefore, we introduce a PPAC refinement network, which leverages RGB guidance data and probabilities for several content- and probability-adaptive convolutions. For both tasks, PPACs not only allow to improve estimates at boundaries but also remove outliers of low reliability. As shown in Fig. 1, blurry edges in the flow field (a) are transformed into crisp boundaries and the overall prediction is smoothed (e). Along with the visual improvements, PPAC refinement leads to a clear accuracy gain in optical flow and semantic segmentation. For instance, PPACs substantially improve state-of-the-art HD3 [60] optical flow estimates on the widely used KITTI 2012 and 2015 datasets. Our proposed PPAC-HD3 method ranks 1st among published optical flow approaches2 on both benchmarks, improving the outlier rate by ~11.1% and ~7.5% over the underlying baseline method.

1By probability we refer to a measure that approximates or summarizes the marginal posterior over the network’s estimates, e.g. by taking the marginal posterior of the chosen prediction value. To ease readability, the terms probability, confidence, (uncertainty, and reliability will be used interchangeably in the paper.

2All rankings at the time of publication.

2. Related Work

Probabilistic deep networks. Combining probabilistic approaches with deep networks is an active field of research, which is pursued to cope with model and/or input uncertainty [35]. As such, we can only provide a rough summary and refer to the cited references for a broader overview.

Bayesian neural networks [5, 15, 20, 37, 42, 57] often learn parametric distributions over the network weights to capture model uncertainty. The predictive distribution over the outputs is obtained by taking an expectation over the weights through approximate inference [5, 20]. However, Bayesian neural networks introduce many additional parameters and are not always easy to handle in practice [39].

Sampling-based approaches, e.g. [2, 11, 35], are often simpler to apply and include a random component, such as dropout, in the network structure. At test time, the predictive uncertainty is computed as Monte Carlo estimates from several network passes. Similarly, an ensemble of networks can be trained and combined at test time [23, 39]. A major drawback of both avenues is the increased runtime as they require multiple forward passes.

Another line of research uses deep networks to output the parameters of an assumed predictive distribution, either directly [35, 46] or by propagation of input uncertainty [12, 54]. There, the difficulty is to find a parametric distribution that is sufficiently easy to handle in practice and appropriately describes the quantity of interest.

Beyond such general purpose probabilistic treatments, probabilistic networks have also been developed in the context of specific vision problems. Yin et al. [60] propose a method to aggregate correspondence uncertainty in the context of optical flow and stereo matching through various spatial scales. In [27], a multi-hypothesis network for optical flow estimation is trained to output an ensemble at once. Novotny et al. [47] use uncertainty estimates obtained with probabilistic losses to predict the reliability of descriptors.

Content-adaptive convolutions. One category of content-adaptive convolutions adjusts the sampling location of neighbor pixels [8, 33]. Deformable convolutions [8] predict data-dependent offsets to determine at which locations neighboring pixels should be sampled for a spatially-invariant convolution. Another line of research adjusts the convolution weights [34, 59] of standard convolutions. Dynamic filter networks [34] use a subnetwork to predict location-specific weight kernels, which have already shown benefits for optical flow estimation, e.g. [25, 26]. A common drawback is the significant amount of additional parameters, which increases the risk of overfitting – especially if only a limited amount of training data is available.

Several works, therefore, approach content-adaptive convolutions in a more constrained setting. Spatial transformers [30] as well as CARAFE [55] rearrange features
in a content-adaptive way with a global or local transformation before performing the convolution itself. However, they remain restricted to a 2D grid structure. [31, 56] perform image-adaptive convolutions with predefined or learned features in the high-dimensional permutohedral lattice [1]. Such convolutions are computationally expensive and thus not suitable for fast processing. [18] incorporates semantics by learning input-dependent attention masks but requires object classes for (pre)-training. Deep guided filters [58] extend the classical guided filter [19] to learned guidance data. In [48], the parameters for spatially-variant linear representation models of the guided filter are learned with a CNN. In both approaches only 1D guidance data can be used for each output channel. Deep joint image filtering [40] applies standard convolutions to the concatenation of pre-processed guidance data and estimates, but misses explicit knowledge on the relation of both components.

We base our approach on pixel-adaptive convolutions (PACs) [51]. Here, the convolution kernels are split into a fixed weight as well as a location-specific component that depends upon a feature embedding learned from guidance data. PACs have a small computational overhead and are easy to train in practice. Nevertheless, like most adaptive filtering approaches, PACs do not allow to explicitly leverage knowledge about the reliability of filter inputs. While such information can be included as guidance data for the content-adaptive part of the weight kernel, our explicit probabilistic formulation leads to a significant performance gain.

Probabilistic joint filtering. There are only few filtering approaches that jointly consider guidance data as well as probabilities. Different filtering methods have been applied to refine semantic segmentations [50], optical flow [43, 53], and especially depth [17, 38, 45]. However, these approaches are task-specific, tailored to certain filtering methods, and/or rely on time-intensive iterative approaches. [14] replaces unreliable pixels using a network that takes uncertainties and guidance data as input, but uncertainties are not explicitly leveraged to improve estimates. The Fast Bilateral Solver [4] and the Domain Transform Solver [3] allow to perform fast, edge-aware optimization on different estimates. They leverage uncertainty by requiring a closer connection between inputs and outputs for more reliable pixels. As these approaches optimize a predefined objective, their flexibility is restricted. Moreover, [3, 4] cannot backpropagate into the features used to determine the pixel similarity. Closest to ours with regard to probabilistic joint filtering are [22, 29], where confidences are used to extend the bilateral and the guided filter, respectively, by weighing a pixel’s importance with its confidence. However, both methods rely on predefined features for pixel similarity, hand-crafted reliability measures, and fixed filter kernels.

In comparison to previous work, our approach is very general as pixel feature embeddings, the filter weights, as well as a pre-processing of the probabilities are learned from data. Moreover, the proposed approach is fast and easily integrable into different task-specific neural networks.

3. Probabilistic Pixel-Adaptive Convolutions

We begin by presenting pixel-adaptive convolutions (PACs) as introduced in [51]. We then propose an advanced normalization approach for pixel-adaptive convolutions and finally extend PACs to allow for probabilistic filtering.

3.1. Pixel-adaptive convolutions

Assume, we aim to perform a convolution with neighborhood size \( s \) that transforms features \( v \in \mathbb{R}^d \) into features \( \tilde{v} \in \mathbb{R}^d \). We denote the corresponding convolution weights as tensor \( \mathbf{W} \in \mathbb{R}^{d 	imes d 	imes s 	imes s} \) and the bias term as \( b \in \mathbb{R}^d \). Following the notation of [51], the output of a standard convolution at pixel \( i \) is then given as

\[
\tilde{v}_i = \sum_{j \in \mathcal{N}(i)} \mathbf{W} [p_i - p_j] v_j + b,
\]

where \( \mathcal{N}(i) \) denotes the \( s \times s \) neighborhood of the pixel. The vectors \( p_i \) and \( p_j \) represent the 2D pixel positions. \( \mathbf{W} [p_i - p_j] \in \mathbb{R}^{d 	imes d} \) corresponds to the 2D slice from weight tensor \( \mathbf{W} \), evaluated at position \( p_i - p_j \).

PACs generalize such spatially-invariant convolutions by augmenting the convolution weight with an additional location-adaptive component \( K(f_i, f_j) \). In [51], the vectors \( f_i \) and \( f_j \) are denoted as “pixel features” and characterize the pixels \( i \) and \( j \), respectively. For instance, one could use the RGB components of a guidance image as feature \( f_i \), as is done in the bilateral filter [52]. However, more advanced features learned from data have shown to be advantageous [51]. The function \( K(f_i, f_j) = K(f_i - f_j) \) is a (fixed) kernel, which evaluates the difference between \( f_i \) and \( f_j \). If pixels \( i \) and \( j \) show similar characteristics, \( K(f_i, f_j) \) weighs the corresponding values \( v_j \) more than the ones of a more deviating pixel. Various choices for \( K(\cdot, \cdot) \) are possible; we will apply a Gaussian RBF kernel in the following, i.e.

\[
K(f_i, f_j) = e^{-\frac{1}{2}(f_i - f_j)^2}. \tag{2}
\]

The PAC convolution [51] is then defined as

\[
\tilde{v}_i = \sum_{j \in \mathcal{N}(i)} K(f_i, f_j) \cdot \mathbf{W} [p_i - p_j] v_j + b. \tag{3}
\]

The same feature kernel \( K \) is used for all input channels, while the weight \( \mathbf{W} \) differs dependent on the spatial location within the mask and for each feature channel.

3.2. Advanced normalization step

PACs are a powerful tool for deep dense prediction architectures, but they are not without challenges. One major issue is the fact that the number of closely related pixels, i.e. pixels \( j \) that show a high value of \( K(f_i, f_j) \), varies.
across the image. This is natural and even desirable, since only a restricted neighborhood region should be taken into account at object boundaries or similar. Nevertheless, applying PACs in different neighborhoods should lead to results with the same output magnitude as long as the input variables are equal. Otherwise, learning of convolution weights might be difficult since the output values are inevitably smaller at boundaries or in highly structured areas.

**Basic scheme.** The implementation of PACs provides an option to normalize kernels such that \( \sum_j K(f_i, f_j) = 1 \) for all pixels \( i \). However, we argue that such a kernel normalization is not sufficient. Consider the illustration of two pixels \( i \) and \( i' \) and their neighborhoods \( \mathcal{N}(i) \) and \( \mathcal{N}(i') \) in Fig. 2. For simplicity, we assume equal input values for pixels of the same object. Pixel \( i \) is part of a homogeneous area and the kernel function leads to an equal distribution of the kernel weights. In contrast, the neighborhood of pixel \( i' \) contains also elements from a different object. Here, the kernel weights are only distributed over the elements from the same object as pixel \( i' \). Even though both kernels sum to one, their convolution with an exemplary weight \( W \) leads to clearly different results. As mentioned above, this complicates the learning of PACs.

**Our advanced scheme.** We address this issue with an advanced normalization scheme. To that end, we adapt the normalization from [56] to the context of PACs. An auxiliary array \( \tilde{v}_{aux} = 1 \) with constant value 1 is defined and passed through the filtering step in Eq. (3):

\[
\tilde{v}_{i, aux} = \sum_{j \in \mathcal{N}(i)} K(f_i, f_j) \cdot W [p_i - p_j] \cdot 1.
\]  

(4)

In slight abuse of notation, we now denote by \( \tilde{v}_i \) the PAC output in Eq. (3) before adding the bias term. Then, the normalized convolution output \( \tilde{v}_{i, norm} \) is given by

\[
\tilde{v}_{i, norm} = \frac{\tilde{v}_i}{\tilde{v}_{i, aux}} + b.
\]  

(5)

With the novel normalization, not only the number of similar pixels is taken into account but also their weighting by \( W \). However, the normalization becomes invalid as soon as the weight \( W \) becomes negative [56]. We thus follow [56] to introduce an additional normalization weight \( \tilde{W} \), which replaces \( W \) in the normalization convolution in Eq. (4). \( \tilde{W} \) is ensured to remain positive and set to the same initialization as \( W \). During training, we independently update \( W \) as well as \( \tilde{W} \) using regular gradient-based optimization and thus omit to explicitly enforce their similarity.

Reconsidering the example in Fig. 2, both output values remain close when using our advanced normalization with the appropriate \( \tilde{W} \approx W \). We will show in Sec. 6.1 that the proposed scheme leads to superior results in practice.

### 3.3. Probabilistic pixel-adaptive convolutions

While the definition of PACs allows for more advanced filtering than a standard convolution, the approach is still restricted. The kernel function \( K(\cdot, \cdot) \) in Eq. (2) only takes differences of pixel features as input, thus excluding properties that cannot be reasonably expressed as such. Consider, for instance, that we have a per-pixel probability alongside the estimate. In this case, it seems beneficial to consider such information during filtering to improve unreliable estimates. As \( K(\cdot, \cdot) \) rewards similar pixel features, neighbors with a similar level of reliability are more closely connected. However, this seems counterintuitive, given that the values of reliable pixels should particularly propagate to neighbors with a very different, i.e. low, confidence.

Therefore, we extend PACs such that we are able to perform convolutions that also take unary properties – especially probabilities – into account. Let \( c_j \) describe the confidence assigned to a certain pixel location \( j \). For consistency, we assume that only one confidence estimate is given per spatial location.\(^3\) Similar to [29], we then propose to define a probabilistic pixel-adaptive convolution (PPAC) as:

\[
\tilde{v}_i = \sum_{j \in \mathcal{N}(i)} c_j \cdot K(f_i, f_j) \cdot W [p_i - p_j] v_j + b.
\]  

(6)

Here, each pixel value is not only weighted by its distance to the center pixel but also with its individual confidence.

To illustrate our proposed approach, consider Fig. 3: An outlier pixel is surrounded by more reliable estimates,\(^3\) An extension to individual confidences per channel is straightforward.
which belong to the same and a different object. For simplicity, we assume that \( W \) performs an averaging operation. If the pixel confidence was not taken into account, the outlier value would spread to the surrounding pixels due to its higher magnitude. In contrast, the proposed PPAC allows to propagate the reliable pixel values from the same object and thus almost completely replaces the outlier in the center.

The normalization step is performed as before and the PPAC outputs are divided per pixel by \( \eta_i \) before the bias (Eq. 5).

### 4. Refinement Networks with PPACs

Deep dense predictors often output predictions at a scale lower than the input resolution to save time and parameters, e.g. [7, 60]. The low-resolution estimates are then upscaled by simple methods such as bilinear interpolation. In [51, 56, 58], image-adaptive convolutions have proven very helpful in this upsampling step. Following that, we propose a PPAC refinement network, which takes image and reliability data into account to upscale and refine network outputs.

A straightforward approach is to upscale results with transposed PPACs. However, we found that this can lead to difficulties, especially for optical flow. This is due to the fact that flow networks often assume the input sizes to be divisible by a certain power of 2, e.g. \( 2^n \) [60]. As this is mostly not the case, e.g. for the Sintel benchmark [6], input images are resized and the output flow is afterwards rescaled with a non-integer factor. To apply transposed convolutions, which can only upscale by integers, one has to pad the inputs and crop the output after upscaling. We observed that this leads to severe artifacts, which clearly reduce the accuracy.

Instead, we propose to first upscale the estimates by the default method of the original network. A lightweight network with PPACs is then applied at full resolution. As we only use a small number of PPACs, the computational expense of the approach remains low and the prediction accuracy does not decrease due to padding artifacts or similar.

Our proposed refinement networks consist of three branches as illustrated in Fig. 4. In addition to the upscaled estimates, the network takes corresponding probability data, e.g. a full marginal posterior per pixel or other probability measures, and the high-resolution images as inputs. The first subnetwork transforms the probability data into scalar confidence values for the individual PPACs. The second branch processes the guidance images to generate meaningful pixel features. Both intermediate outputs as well as the underlying network predictions are then fed into the PPACs of the combination branch to create a refined estimate.

![Exemplary architecture of our proposed PPAC refinement networks.](image)

**5. Implementation**

**5.1. Network architecture**

To illustrate their capabilities, we experiment with PPAC refinement networks for the tasks of optical flow estimation and semantic segmentation. All networks are fully convolutional and include two consecutive PPACs. We use log values as inputs to the probability branch and add a sigmoid function at its end for normalization. Please refer to the supplement for a detailed description of the networks’ setup.

Here, we only highlight architectural choices that differ significantly from the ones in [51]. First, we found no benefit from increasing the number of channels in the combination branch. Moreover, using group convolutions with the number of groups being equal to the number of inputs did not decrease the performance, but significantly lowers the number of parameters. We even go further and share the convolution weights across all channels, which proved especially beneficial for semantic segmentation, possibly due to the large reduction in parameters. Following the upsampling setup in [51], we also experimented with standard convolution layers to pre- or post-process the data itself. However, we found no benefit from such convolutions and thus stick with a combination branch that only includes PPACs.

**5.2. Additional baselines**

To assess the benefits of our PPAC refinement networks, we introduce two baseline networks. The simple refinement network takes estimates, log-probabilities, and input images and processes them jointly with standard convolutions. Here, we set the channel depth such that the number of parameters used for the PPAC network is approximately equal to the simple baseline. Additionally, we use a PAC baseline that replaces all PPACs with its non-probabilistic PAC counterpart. For this network, the probability branch is removed and the probabilities are instead concatenated with the guidance images and fed to the guidance subnetwork. We again ensure that the number of parameters remains comparable for PAC and PPAC refinement networks.

**5.3. Training procedure**

For fair comparison, we only train the refinement networks and do not backpropagate into the original networks.
Table 1. Average end-point error (AEE) and 3-pixel outlier rate (out) on our Sintel and KITTI test splits for different normalizations of a PAC refinement network.

<table>
<thead>
<tr>
<th></th>
<th>Sintel (AAE)</th>
<th>KITTI</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>clean</td>
<td>final</td>
<td>AEE</td>
<td>out</td>
</tr>
<tr>
<td>HD3 [60]</td>
<td>1.672</td>
<td>1.357</td>
<td>1.990</td>
<td>6.14%</td>
</tr>
<tr>
<td>PAC w/o normalization</td>
<td>1.665</td>
<td>1.352</td>
<td>1.924</td>
<td>6.34%</td>
</tr>
<tr>
<td>PAC w/ kernel normal.</td>
<td>1.622</td>
<td>1.323</td>
<td>1.921</td>
<td>6.15%</td>
</tr>
<tr>
<td>PAC w/ adv. normal. (ours)</td>
<td><strong>1.594</strong></td>
<td><strong>1.302</strong></td>
<td><strong>1.868</strong></td>
<td><strong>5.81%</strong></td>
</tr>
</tbody>
</table>

Table 2. Average-end-point error (AEE) and 3-pixel outlier rate (out) on our Sintel and KITTI test splits for different refinements.

<table>
<thead>
<tr>
<th></th>
<th>Sintel (AAE)</th>
<th>KITTI</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>clean</td>
<td>final</td>
<td>AEE</td>
<td>out</td>
</tr>
<tr>
<td>HD3 [60]</td>
<td>1.672</td>
<td>1.357</td>
<td>1.990</td>
<td>6.14%</td>
</tr>
<tr>
<td>Simple refinement</td>
<td>1.638</td>
<td>1.334</td>
<td>1.872</td>
<td>5.92%</td>
</tr>
<tr>
<td>PAC network [51]</td>
<td>1.594</td>
<td>1.302</td>
<td>1.868</td>
<td>5.81%</td>
</tr>
<tr>
<td>PPAC network (ours)</td>
<td><strong>1.562</strong></td>
<td><strong>1.283</strong></td>
<td><strong>1.848</strong></td>
<td><strong>5.50%</strong></td>
</tr>
<tr>
<td>SL [56]</td>
<td>1.634</td>
<td>1.340</td>
<td>1.953</td>
<td>6.41%</td>
</tr>
</tbody>
</table>

Networks from the author-provided checkpoint pre-trained on FlyingChairs [9] as well as FlyingThings3D [28] and determine the best models on our validation images. Following [60], only images from Sintel final are used for fine-tuning of the Sintel model.

All refinement networks are trained for 500 epochs with the average end-point error (AEE) as loss function and a batch size of 8. The base learning rates are cut by a factor of two every 100 epochs. As augmentations, we only apply random cropping to sizes (384, 768) and (320, 896) for Sintel and KITTI data, respectively. On Sintel, individual networks are trained on the final and clean subsets.

Comparison of normalization approaches. We first demonstrate the benefit of our proposed advanced normalization scheme. Therefore, we train PAC refinement networks without normalization, similar to [51], and with kernel normalization, c.f. Sec. 3.2. Table 1 shows results evaluated on our test sets of Sintel and KITTI. Note that the results on Sintel clean tend to be worse than on final as this data has not been seen during HD3 fine-tuning. We observe that a PAC network with kernel normalization is able to improve the accuracy over the HD3 baseline as well as the PAC approach without normalization. The same network leveraging our proposed advanced normalization shows clearly the best accuracy on all test sets. We attribute this to the fact that kernel normalization does not sufficiently compensate different neighborhood conditions.

Evaluation of PPACs. We now evaluate refinement networks including PPACs in comparison to the simple and PAC baselines as introduced in Sec. 5.2. The results of flow refinement on our test splits are summarized in Table 2. The simple refinement approach based only on standard convolutions (with the same inputs) improves the flow predictions only slightly on all datasets. Applying guidance data, including the estimated probabilities, explicitly in a PAC refinement network already leads to a clear improvement. However, our proposed PPAC refinement network outperforms the PAC approach by a large margin, improving the AEE by 6.6%, 5.5% and 7.1% on Sintel clean, Sintel final...
and KITTI, respectively. We also observe that the improvement of using content-adaptive, probabilistic convolutions over a simple setup with standard convolutions is more significant on Sintel than on KITTI. We attribute this to the fact that guidance data has shown to be most effective in boundary regions [56], which play a lesser role in the KITTI dataset as the ground truth is sparse.

Even more striking than these significant accuracy gains, are the improvements in visual quality. Figs. 1 and 5 show example flow fields on Sintel final and KITTI. PPACs clearly improve the underlying HD3 estimates and lead to substantial improvements especially near motion boundaries. Additionally, our proposed approach is able to correctly propagate flow estimates into outlier regions.

To further understand the results of PPAC refinement, we evaluate an additional oracle network, which takes oracle confidences as input to the probability branch, which we take to be the inverse of the AEE. As this correctly assesses the reliability of each pixel, this networks provides an upper bound on the possible accuracy improvement from probabilistic refinement. Comparing the results in Table 2, we observe that an even more significant improvement would be possible if more precise probability estimates were available. This observation holds especially for the evaluation on KITTI, where a rather small amount of ground truth is available during fine-tuning. This suggests that future work on improved probability estimates in deep network has the potential of improving the accuracy in difficult areas further.

We finally compare our proposed PPAC refinement to other approaches from the literature. As PACs have shown to be the state-of-the-art method for joint upsampling, we restrict further comparison to the Semantic Lattice (SL) [56], which appeared concurrently to [51], and the Fast Bilateral Solver (FBS) [4] as a representative method that explicitly considers confidences for post-processing. The SL is trained as described in [56]. For the FBS, we use the probabilities of the last output layer of HD3 [60]. All FBS weight parameters and the trade-off parameter λ are determined via grid search on the validation set. Note that we were not able to find stable parameters for FBS on KITTI.

Table 3. Average-end-point error (AEE) and AEE of regions \( \leq 10 \) pixels from occlusion boundaries (d0-10) evaluated on Sintel test.

<table>
<thead>
<tr>
<th></th>
<th>clean</th>
<th>final</th>
</tr>
</thead>
<tbody>
<tr>
<td>HD3 [60]</td>
<td>4.788</td>
<td>3.225</td>
</tr>
<tr>
<td>PPAC-HD3 (ours)</td>
<td>4.589</td>
<td>2.788</td>
</tr>
</tbody>
</table>

Table 4. 3-pixel outlier rate of non-occluded/all pixels (Out-Noc/all) and runtimes evaluated on KITTI 2012 and 2015 test.

<table>
<thead>
<tr>
<th></th>
<th>KITTI 2012</th>
<th>KITTI 2015</th>
</tr>
</thead>
<tbody>
<tr>
<td>HD3 [60]</td>
<td>2.26 %</td>
<td>5.41 %</td>
</tr>
<tr>
<td>PPAC-HD3 (ours)</td>
<td>2.01 %</td>
<td>5.09 %</td>
</tr>
</tbody>
</table>

Table 2 shows that SL and FBS are both able to improve the HD3 baseline accuracy. However, PPAC refinement outperforms both previous methods by a large margin.

**Evaluation on benchmarks.** We finish our flow experiments by evaluating PPAC-HD3, i.e. HD3 optical flow [60] with PPAC refinement, on the official benchmarks. For Sintel, we initialize HD3 with the same checkpoint as used in [60]. On KITTI, we use the fine-tuned checkpoint with context module as provided online. To train our PPAC network, we leverage the entire training data provided for the Sintel and KITTI benchmarks, respectively. Again, we train separate networks for Sintel clean and final, as well as a joint refinement network for both KITTI benchmarks. In comparison to our previous experiment, we adjust the number of epochs in the learning scheme such that the total number of iterations remains approximately the same despite the larger number of images. Note that we do not use other augmentations than random cropping, since we found our refinement network to be robust w.r.t. overfitting. We attribute this to the lightweight structure of our PPAC network, which only adds approximately 12k parameters.

Results on Sintel test are summarized in Table 3. PPAC-HD3 clearly improves the accuracy of the underlying HD3 method on clean and final splits by \( \sim 4.2\% \) and \( \sim 1.4\% \), respectively. The larger improvement on clean might be partly due to the fact that no data from this pass was used during fine-tuning of the HD3 baseline. Moreover, we observe a very significant improvement of \( 7.0\% \) on the final pass and of \( \sim 13.6\% \) on Sintel clean when considering the AEE of regions closer than 10 pixels to motion boundaries (d0-10). Overall, PPAC-HD3 ranks 4th on Sintel final among published two-frame methods and is thus highly competitive.

In Table 4, we show results on the official test sets of KITTI 2012 and 2015. PPAC-HD3 outperforms its underlying method by a large margin, leading to a substantial relative improvement of \( \sim 11.1\% \) for the outlier rate of non-
Table 5. Mean intersection over union (mIoU) evaluated on our Pascal VOC 2012 test split. †Results taken from [56].

<table>
<thead>
<tr>
<th></th>
<th>mIoU</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeepLabv3+ [7]</td>
<td>82.20%</td>
</tr>
<tr>
<td>PAC refinement</td>
<td>82.39%</td>
</tr>
<tr>
<td>PPAC refinement</td>
<td>82.62%</td>
</tr>
<tr>
<td>SL [56]†</td>
<td>82.25%</td>
</tr>
<tr>
<td>FBS [4]</td>
<td>82.28%</td>
</tr>
</tbody>
</table>

occluded pixels on KITTI 2012. On KITTI 2015, PPAC refinement improves the outlier rate of all pixels from 6.55% to 6.06% (relative improvement of ∼7.5%). Our proposed method clearly ranks 1st among published optical flow approaches on both datasets. PPAC-HD3 is even able to outperform several strong scene flow methods, which leverage additional stereo data as input. Table 4 also shows computational times measured on a single GTX 1080 Ti GPU. Adding PPACs has a computational overhead of ∼1.7×, which seems justified by the strongly improved results.

6.2. Semantic segmentation

In a second set of experiments, we apply our probabilistic refinement networks to the task of semantic segmentation. We choose DeepLabv3+ [7] as a baseline using the Xception65 variant of the model. Before feeding the logits of the segmentation network into the probability branch of our PPAC network, we normalize them with a log-softmax operation. The logits are equally used as input to the combination branch. Here, we left the values unnormalized as we found a normalization to lead to inferior results.

We evaluate our probabilistic refinement network on Pascal VOC 2012 [10] and use the training, validation, and test split of [56]. When training PAC and PPAC networks, we found it important to use different learning rates for the pre-processing branches as well as the weights in the combination branch. To determine appropriate values, we first fixed the PPAC or PAC weights to a Gaussian kernel with standard deviation $\sigma = 1$ and searched for the optimal learning rate of the guidance and probability subnetworks on the validation set. In a second step, all weights are randomly initialized and a second learning rate is determined for the convolution weights in the combination branch. Furthermore, we observed improved accuracy when initializing the bias term of PACs and PPACs to zero.

Table 5 summarizes the results on our test split of Pascal VOC 2012. In this setting, our PPAC network requires on average 0.055s per image on a single GTX 1080 Ti GPU and is able to improve the segmentation accuracy even though DeepLabv3+ takes already special care of the decoder. In contrast, PAC refinement shows a considerably smaller benefit. We were not able to find a simple configuration that improved the original results. As for optical flow, we also compare to SL [56] and FBS [4], and use the probability of the most likely class as confidence input. Both previous methods are clearly outperformed by our proposed PPACs.

Fig. 6 shows examples of segmentation maps from Pascal VOC 2012. PPAC refinement leads to a better alignment with the underlying objects especially at object intersections or for smaller objects. Moreover, PPACs are able to successfully close smaller holes in the segmentation maps.

7. Conclusion

We introduced probabilistic pixel-adaptive convolutions (PPACs), which allow for filtering operations that respect guidance data as well as per-pixel confidences. Building on the work of [51], we first proposed an advanced normalization scheme, which we show to clearly improve the results in practice. Subsequently, we extend PACs to include confidence information during the filtering step to especially improve regions of low reliability. We proposed to use PPACs for the refinement of dense prediction networks and demonstrated their benefits for optical flow estimation and semantic segmentation. Here, PPAC refinement resulted in significant accuracy gains; our PPAC-HD3 clearly leads both KITTI benchmarks for optical flow. Moreover, refined estimates show fewer boundary artifacts and are smoother overall while correctly respecting object boundaries.


