A. Training Details

In this section, we describe the training setup. For consistency, all experiments in the paper were repeated five times using the pipeline detailed below.

**Data augmentation.** We augment input images during training by random scaling with values between 0.5 and 2.0, random cropping to input size $425 \times 560$ for NYUD-v2— we use the cropped version of [3]—and padded to $512 \times 512$ for PASCAL-Context), random horizontal flipping and random color jitter. Image intensities are standardized. Depth experiments, we use a minibatch size of 8 and train for 40000 iterations.

**Task losses.** The total loss of the multi-task network with parameters $\theta$ is a weighted sum of losses (for tasks $n \in \{1, \ldots, N\}$):

$$L_{\text{total}}(\theta) = \sum_{n=1}^{N} \omega_n L_n(\theta)$$ (1)

For semantic segmentation and human parts segmentation we use a cross-entropy loss (loss weights $\omega_n = 1$ and $\omega_n = 2$ respectively), for saliency estimation a balanced cross-entropy loss ($\omega_n = 5$), for depth estimation a $L_1$ loss ($\omega_n = 1$), for surface normal estimation a $L_1$ loss with unit vector normalization ($\omega_n = 10$) and for boundary detection a weighted cross-entropy loss ($\omega_n = 50$). For boundary detection, the positive pixels are weighted with 0.8 and the negative pixels with 0.2 on NYUD-v2, while on PASCAL-Context the weights are 0.95 and 0.05. $\omega_n$ for each task was determined through a logarithmic grid search over candidate values with single-task networks.

The auxiliary predictions $A_n$ are trained with a cross-entropy loss using the same loss weights as above. However, the auxiliary head backpropagation is stopped from updating parameters of the main network.

**Optimization hyperparameters.** All backbones are initialized with ImageNet pretrained weights. We use Stochastic Gradient Descent (SGD) with momentum of 0.9 and weight decay of 0.0005 to optimize the model parameters. The initial learning rate is determined through a logarithmic grid search ($..., 0.002, 0.005, 0.01, 0.02, ...$), with the option of having a 10 times higher learning rate for the heads vs. the backbone. The initial value is decayed during training according to a ‘poly’ learning rate schedule [1]. For all experiments, we use a minibatch size of 8 and train for 40000 iterations.

**Context type search.** The architecture distribution parameters $\alpha$ are initialized with zeros. We use an Adam optimizer [4] to update them, with learning rate 0.0005 (no weight decay, no learning rate scheduler). The update occurs in the same round of backpropagation as the regular model parameters (single-level optimization). Over the course of training, the Gumbel-Softmax temperature $\lambda$ is annealed linearly from 1.0 to 0.05 (following [9]). Also, to ensure a fair candidate context type selection, we disable learnable affine parameters of the last batch normalization of every context type attention mechanism.

As discussed in Sec. 3.3, we use entropy ($H$) regularization to control the sampling variance during the architecture search. Specifically, we calculate the mean entropy of the architecture parameter ($\alpha$)-distributions over all Context Pooling (CP) blocks, scale it with a weight $\omega_H$, and add it to the total loss.

$$L_{\text{search}}(\theta, \alpha) = \sum_{n=1}^{N} \omega_n L_n(\theta, \alpha) + \frac{\omega_H}{N^2} \sum_{j=1}^{N} H(\alpha_j)$$ (2)

$j$ indexes the CP blocks. The scaling factor $\omega_H$ follows a linear schedule during the search, from -0.02 to 0.06. We found that this provides an adequate balance between candidate exploration and exploitation. For a given CP block $j$, architecture search is terminated prematurely if the difference between the two largest values of $\alpha_j$ exceeds 0.3. One candidate is then sampled using $\text{argmax}$ (i.e., $\alpha_j$ becomes a one-hot vector).

After concluding five runs of the architecture search, we determine the final configuration by choosing the context type receiving the most votes over the five runs in each CP.
Table B-1. NYUD-v2 single task performances of HRNetV2-W18-small (HRNet18) and HRNetV2-W48 (HRNet48) models [8]. We compare the performances obtained using our implementation with the numbers published in [7].

<table>
<thead>
<tr>
<th>Model</th>
<th>SemSeg ↑</th>
<th>Depth ↓</th>
<th>Normal ↓</th>
<th>Bound ↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>HRNet18, [7]</td>
<td>33.18</td>
<td>0.667</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>HRNet18, ours</td>
<td>38.02</td>
<td>0.610</td>
<td>20.94</td>
<td>76.22</td>
</tr>
<tr>
<td>HRNet48, [7]</td>
<td>45.70</td>
<td>0.547</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HRNet48, ours</td>
<td>45.87</td>
<td>0.540</td>
<td>20.09</td>
<td>77.34</td>
</tr>
</tbody>
</table>

Table E-1. NYUD-v2 comparison of the performance upper bound of T-label and S-label context, using ground truth (GT) spatial region maps $A_{n}^{\text{GT}}$ (see Sec. 3.2.3).

<table>
<thead>
<tr>
<th>Model</th>
<th>SemSeg ↑</th>
<th>Depth ↓</th>
<th>Normal ↓</th>
<th>Bound ↑</th>
<th>$\Delta_n$ [%] ↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single task</td>
<td>38.02</td>
<td>0.6104</td>
<td>20.94</td>
<td>76.22</td>
<td>0.00</td>
</tr>
<tr>
<td>T-label, GT</td>
<td>46.71</td>
<td>0.5202</td>
<td>18.16</td>
<td>76.06</td>
<td>12.67</td>
</tr>
<tr>
<td>S-label, GT</td>
<td>47.71</td>
<td>0.5160</td>
<td>17.87</td>
<td>78.18</td>
<td>14.55</td>
</tr>
</tbody>
</table>

For depth prediction, our approach is inspired by [5]. The estimation error for larger depth values is naturally larger. During training, we learn a classifier to assign the pixels to the bins. During evaluation, we use a soft-weighted-sum inference: Every bin is represented by its mean depth in log space. A weighted sum of bins (weight = prediction score) is used as the final prediction.

For surface normal estimation, we use the triangular coding technique of [10]. First, a codebook is learned with k-means. The codewords form a Delaunay triangulation cover on the unit sphere. Any surface normal can thus be expressed as a weighted combination of the three codewords marking its triangle. During training, we learn a classifier to predict those codeword weights. Following [10], we choose 40 codewords (≈ 40 classes). Evaluation consists of two steps: (1) Find the triangle with maximum total probability. (2) Use the probabilities of the three codewords of that triangle to reconstruct the surface normal.

To verify the above discretization schemes, we trained single task models accordingly, and compared them to the regression models in Fig. D-1. The figure shows that the performance of classification—while slightly worse than regression—is satisfactory for both depth and surface normal estimation. The same conclusion can be drawn from a qualitative comparison, shown in Fig. D-2.

E. Label Context: Performance Upper Bound

To estimate the potential of label context for multi-modal distillation, we conduct experiments using ground truth label regions. Instead of predicting the spatial maps $A_{n}$ from the input image (see Sec. 3.2.3), we directly use the ground truth data $A_{n}^{\text{GT}}$ to partition the label space into distinct regions. This provides an upper bound for the performance of label context distillation. Table E-1 shows the results for both T-label and S-label context: The performance increases greatly (with the exception of T-label context for boundary detection), confirming that label region grouping is highly effective for multi-modal distillation.

F. Context Type Search Reliability

We consider the context type selection during architecture search as a rater decision. Since we repeat each run five times, we can evaluate the intra-rater reliability: The agreement among the five selected context types in all CP blocks.

The most intuitive way to quantify agreement is through percentage agreement (i.e., counting the fraction of times a...
Figure C-1. Schematics of the different relational context types. The grids represent individual pixels (channels not shown), the attention mechanism is shown for one target pixel (framed in red box) respectively. Normalization is applied over all pixels of the attention map. $A_s$ are the auxiliary predictions, as depicted in Fig. 2a.

Figure D-1. Performance comparison of single task depth and surface normal estimation models, using either a regression or classification framework. Their similar performance confirms that we can exploit the classification scheme to form high-quality label regions for the label context.

Figure D-2. Qualitative NYUD-v2 comparison of regression and classification schemes for depth (top two rows) and surface normal (bottom two rows) estimation. Classification achieves satisfactory results on both tasks.

G. How Important is Self-Attention in ATRC?

The permutation testing results of Sec. 4.4 can be utilized to partly address this question. We conclude there that self-attention constitutes the most important distillation module for 3 out of 4 investigated tasks. However, other cross-task connections contribute significantly too. To investigate fur-
ther, we provide in Table G-1 the performance of ATRC without self-attention. The multi-task performance $\Delta_m$ for this model is 0.87\% (v.s. 1.56\% for the full ATRC), outperforming the single task configuration. This confirms that, even though self-attention is vital according to permutation testing, the cross-task distillation modules are able to provide a substantial performance boost on their own.

<table>
<thead>
<tr>
<th>Distillation module</th>
<th>SemSeg ↑</th>
<th>Depth ↓</th>
<th>Normal ↓</th>
<th>Bound ↑</th>
<th>$\Delta_m$ [%] ↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>None (single task baseline)</td>
<td>38.02 ± 0.14</td>
<td>0.6104 ± 0.0041</td>
<td>20.94 ± 0.08</td>
<td>76.22 ± 0.07</td>
<td>0.00</td>
</tr>
<tr>
<td>None (multi-task baseline)</td>
<td>36.35 ± 0.26</td>
<td>0.6284 ± 0.0054</td>
<td>21.02 ± 0.06</td>
<td>76.36 ± 0.05</td>
<td>-1.89</td>
</tr>
<tr>
<td>ATRC (ours)</td>
<td>38.90 ± 0.43</td>
<td>0.6010 ± 0.0046</td>
<td>20.48 ± 0.02</td>
<td>76.34 ± 0.12</td>
<td>1.56</td>
</tr>
<tr>
<td>ATRC (no self-attention)</td>
<td>38.19 ± 0.46</td>
<td>0.6032 ± 0.0038</td>
<td>20.55 ± 0.05</td>
<td>76.22 ± 0.10</td>
<td>0.87</td>
</tr>
</tbody>
</table>

Table G-1. Effect of removing the self-attention blocks on NYUD-v2 with a HRNet18 backbone. First three lines correspond to the numbers reported in Table 1.

References


