Rethinking Inductive Biases for Surface Normal Estimation

Supplementary Material

1. Network architecture

Tab. 1 shows the architecture of the CNN used to extract the initial surface normals, initial hidden state, and context feature. For the ConvGRU cell and convex upsampling layer, we use the architecture of [1] and [5], respectively.

2. Data preprocessing

During training, the input image goes through the following set of data augmentation (p: the probability of applying each augmentation).

- **Downsample-and-upsample** ($p = 0.1$). Bilinearly downsample the image $(H \times W)$ into $(rH \times rW)$, where $r \sim \mathcal{U}(0.2, 1.0)$. Then upsample it back to $(H \times W)$.
- **JPEG compression** ($p = 0.1$). Apply JPEG compression with quality $q \sim \mathcal{U}(10, 90)$.
- **Gaussian blur** ($p = 0.1$). Add Gaussian blur with kernel size (11×11) and $\sigma \sim \mathcal{U}(0.1, 5.0)$.
- **Motion blur** ($p = 0.1$). Simulate motion blur by convolving the image with a 2D kernel whose value is 1.0 along a line that passes through the center and is 0.0 elsewhere. The kernel is then normalized such that its sum equals 1.0. The kernel size is drawn randomly from $[1, 3, 5, 7, 9, 11]$.
- **Gaussian noise** ($p = 0.1$). Add Gaussian noise $x \sim \mathcal{N}(0, \sigma)$ where $\sigma \sim \mathcal{U}(0.01, 0.05)$. Note that the image is pre-normalized to $[0.0, 1.0]$.
- **Color** ($p = 0.1$). Use ColorJitter in PyTorch [4] with (brightness=0.5, contrast=0.5, saturation=0.5, hue=0.2).
- **Grayscale** ($p = 0.01$). Change the image into grayscale.

We also randomize the aspect ratio of the input image. Suppose that the input has a resolution of $H \times W$. We first randomize the target aspect ratio $H^{\text{target}} \times W^{\text{target}}$, while making sure that the total number of pixels is roughly 300K (to maintain GPU memory usage). We then resize the input into $rH \times rW$, such that $rH \sim \mathcal{U}(\min(H, H^{\text{target}}), \max(H, H^{\text{target}}))$. The resized input is then cropped based on the target resolution.

3. Additional results

We provide an additional qualitative comparison to Omnidata v2 [3] in Fig. 1.

References

[1] Gwangbin Bae, Ignas Budvytis, and Roberto Cipolla. Irondepth: Iterative refinement of single-view depth using surface normal and its uncertainty. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2022. 1

[2] Weifeng Chen, Shengyi Qian, David Fan, Noriyuki Kojima, Max Hamilton, and Jia Deng. Oasis: A large-scale dataset for single image 3d in the wild. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 3

[3] Ainaz Eftekhar, Alexander Sax, Jitendra Malik, and Amir Zamir. Omnidata: A scalable pipeline for making multi-task mid-level vision datasets from 3d scans. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021. 1, 3

[4] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. 1

[5] Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020. 1

Input	Layer	Output	Output Dimension
image	-	-	$H \times W \times 3$
Encoder			
image	EfficientNet B5	F_8 F_{16} F_{32}	$H/8 \times W/8 \times 64$ $H/16 \times W/16 \times 176$ $H/32 \times W/32 \times 2048$
Decoder			
$F_{32} + \mathbf{r}_{32}$	Conv2D(ks=1, C_{out}=2048, padding=0)	x_0	$H/32 \times W/32 \times 2048$
$\text{up}(x_0) + F_{16} + \mathbf{r}_{16}$	$\begin{pmatrix}\text{Conv2D(ks=3, } C_{\text{out}}\text{=1024, padding=1),}\\ \text{GroupNorm}(n_{\text{groups}} = 8),\\ \text{LeakyReLU()}\end{pmatrix} \times 2$	x_1	$H/16 \times W/16 \times 1024$
$\text{up}(x_1) + F_8 + \mathbf{r}_8$	$\begin{pmatrix}\text{Conv2D(ks=3, } C_{\text{out}}\text{=512, padding=1),}\\ \text{GroupNorm}(n_{\text{groups}} = 8),\\ \text{LeakyReLU()}\end{pmatrix} \times 2$	x_2	$H/8 \times W/8 \times 512$
Prediction Heads			
$x_2 + \mathbf{r}_8$	Conv2D(ks=3, C_{out}=128, padding=1), ReLU(), Conv2D(ks=1, C_{out}=128, padding=0), ReLU(), Conv2D(ks=1, C_{out}=3, padding=0), Normalize(), rayReLU()	$\mathbf{n}^{t=0}$	$H/8 \times W/8 \times 3$
$x_2 + \mathbf{r}_8$	Conv2D(ks=3, C_{out}=128, padding=1), ReLU(), Conv2D(ks=1, C_{out}=128, padding=0), ReLU(), Conv2D(ks=1, C_{out}=64, padding=0)	$\mathbf{f}$	$H/8 \times W/8 \times 64$
$x_2 + \mathbf{r}_8$	Conv2D(ks=3, C_{out}=128, padding=1), ReLU(), Conv2D(ks=1, C_{out}=128, padding=0), ReLU(), Conv2D(ks=1, C_{out}=64, padding=0)	$\mathbf{h}^{t=0}$	$H/8 \times W/8 \times 64$

Table 1. **Network architecture.** In each 2D convolutional layer, "ks" and C_{out} are the kernel size and the number of output channels, respectively. F_N represents the feature-map of resolution $H/N \times W/N$, and $\mathbf{r}_N$ is a dense map of per-pixel ray direction in the same resolution. $X + Y$ means that the two tensors are concatenated, and $\text{up}(\cdot)$ is bilinear upsampling by a factor of 2.

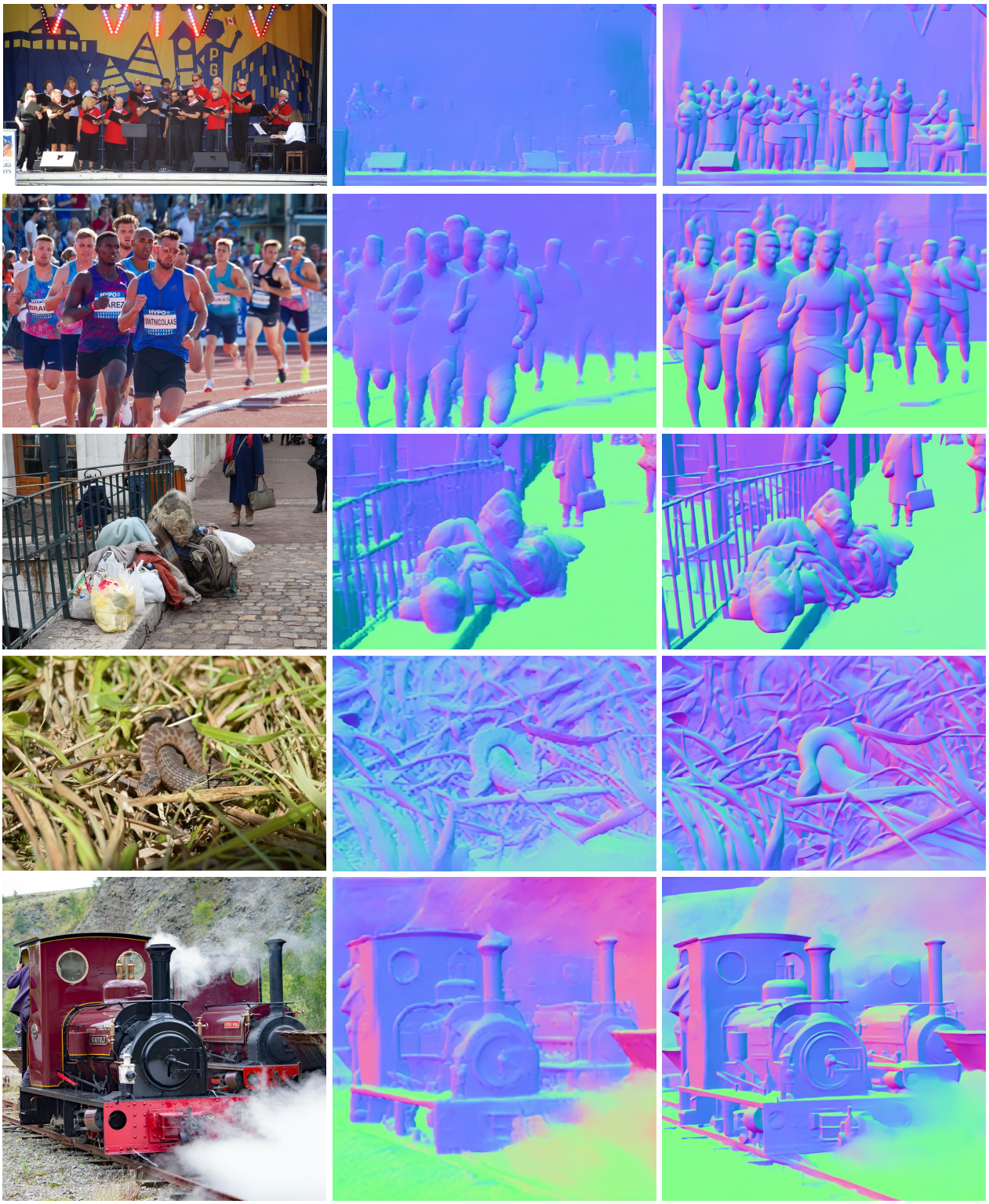

Figure 1. **Additional comparison to Omnidata v2 [3] on in-the-wild images from the OASIS dataset [2].**