Estimating Spatially-Varying Lighting in Urban Scenes with Disentangled Representation (Supplemental Material)

Jiajun Tang[1], Yongjie Zhu[2], Haoyu Wang[1], Jun Hoong Chan[1], Si Li[2], and Boxin Shi[1,3] ✉

[1] NERCVT, School of Computer Science, Peking University
[2] School of Artificial Intelligence, Beijing University of Posts and Telecommunications
[3] Peng Cheng Laboratory

In this supplementary material, we provide more details about our data collection, implementation details and network architectures. We also show additional results on our enhanced synthetic dataset, captured real data and collected in-the-wild data.

7 Appendix

7.1 Data Collection Details

For synthetic data generation, we collect 30 PBR materials from ambientCG [1] in 4 categories: bricks, concrete, pavement, and grass field; we then mark the 3D city model with the category labels and randomly replace the materials following the category labels in Blender [2][4]. After material enhancement, we follow the camera sampling and rendering process proposed in SOLID-Net [13]. Specifically, the view point (x, y, z) of the local lighting is 10 cm away from the surface at the designated pixel position (u, v) along its normal.

When capturing the real data of the outdoor spatially-varying lighting, we use a level to ensure we capture the correct sun elevation angle relative to the skyline and we use a compass to ensure we capture the correct sun azimuth angle in the panoramas relative to the observation direction of the limited-FoV images. Due to the long capture time and the moving clouds, we discard the data if the lighting environment changes largely in its capture process, which makes our data capture very time-consuming, especially in (partly-)cloudy weathers.

7.2 Implementation Details

In our implementation, we use 128 parameters in total to model spatially-varying lighting. Among them, $z_{\text{sky}} \in \mathbf{R}^{16}$, $z_{\text{sun}} \in \mathbf{R}^{45}$, M_{sun} is determined by 2 parameters, $z_{\text{local}} \in \mathbf{R}^{64}$, and we treat the spatially-varying sun visibility z_{vis} as another parameter. The image resolutions are 320×240 for limited-FoV images I and 128×64 for panoramic maps (128×32 for panoramic maps of the sky-dome).

[4] We use the PBR materials in Blender following the document at https://docs.ambientcg.com/books/using-the-assets/page/pbr-materials-in-blender

Our full pipeline is implemented in PyTorch [10] and trained step-wise. We first train our global lighting encoder-decoder on Laval Sky dataset [6] for 30 epochs and finetune on the mixture of Laval Sky dataset [6] and our global lighting data for 5 epochs. Then we extract the global lighting codes z_{sky} and z_{sun} of our global lighting data and train our local content encoder-renderer using our local-global paired data for 20 epochs. At last, the local lighting data are encoded into z_{local} and we train our spatially-varying lighting estimator using these code labels for 20 epochs. When training the global lighting encoder-decoder and the local content encoder-renderer, we apply the radiometric distortion [3] and random rotation at the probability of 0.5 and 0.6 respectively, and we use the ground truth sun position as input, leaving the sun position estimation as a disentangled task in the estimator. The batch size is set to 16 in all of our experiments. The sun position map M_{sun} has the resolution of 128×32 and only the 8×8 patch at the (predicted) sun position is set as 1 otherwise 0. We use Adam optimizer [5] in all of our experiments, the initial learning rates are set to 4×10^{-3} and are halved every 5 epochs.

Since outdoor HDR images have extremely bright intensities in sun pixels, for a more stable training using CNN, all HDR images are tone-mapped before being fed into the networks [4]:

$$T = \frac{\log(1 + \mu H)}{\log(1 + \mu)}, \tag{6}$$

where linear HDR images H are mapped into log-compressed images T, and we empirically set $\mu = 16$.

7.3 Sun Visibility Mask

In our proposed methos, we unify the occlusion and weather factors into one sun visibility component. The sun visibility mask M_{vis} used in our method has the same resolution as the limited-FoV image I and indicates whether a pixel position is directly illuminated by the sun light, which is not identical to a shadow mask in non-sunny weathers. However it's hard to get the accurate mask strictly following the aforementioned definition. To get the approximate mask for training supervision, we use the Lambertian diffuse material to override the original materials in the 3D city model and render limited-FoV images using the same cameras with the shadow visibility set on and off in the Blender [2] Cycles engine as I_{ws} and I_{wos}, then we calculate the intensity of the difference map $D = \text{gray}(I_{\text{wos}} - I_{\text{ws}})$. The higher pixel value in D, the more likely the pixel is in the cast shadow or attached shadow and is not directly illuminated by the sun light. We use the combination of a fixed threshold of the minimum difference $t_{\text{min}} = 0.002$ and an adaptive threshold t_{otsu} from the Otsu's method [9] to convert the different map D into the approximate binary label of sun visibility mask M_{vis}. For cloudy scenes, we set all pixels as non-visible (0). Since the mask is an approximation and to make our prediction robust to noisy pixels, we use the predicted mask M'_{vis} in a conservative manner: We constrain the sun light

visibility in the estimated local lighting only when all pixel values in the 7×7 patch of the pixel position are smaller than 0.2, otherwise we do nothing.

7.4 Training Losses

Our full pipeline is trained with the following losses:

$$\mathcal{L} = \mathcal{L}_G + \mathcal{L}_L + \mathcal{L}_E, \tag{7}$$

where $\mathcal{L}_G$ is the loss term for our global lighting encoder-decoder, $\mathcal{L}_L$ is the loss term for our local content encoder-renderer, and $\mathcal{L}_E$ is the loss term for our spatially-varying lighting estimator.

For global lighting encoder-decoder, $\mathcal{L}_G$ is defined as:

$$\mathcal{L}_G = \mathcal{L}_{\text{sky}} + \mathcal{L}_{\text{sun}} + \mathcal{L}_{\text{info}}, \tag{8}$$

where $\mathcal{L}_{\text{sky}}$ is the L_1 reconstruction loss for sky light:

$$\mathcal{L}_{\text{sky}} = ||P_{\text{sky}} \odot (1 - M_{\text{sun}}) - P'_{\text{sky}} \odot (1 - M_{\text{sun}})||_1, \tag{9}$$

$\mathcal{L}_{\text{sun}}$ is the L_2 reconstruction loss for sun light:

$$\mathcal{L}_{\text{sun}} = ||P_{\text{sun}} \odot M_{\text{sun}} - P'_{\text{sun}}||_2, \tag{10}$$

and $\mathcal{L}_{\text{info}}$ is the same as that in HDSky [11].

For local content encoder-renderer, $\mathcal{L}_L$ is defined as:

$$\mathcal{L}_L = \mathcal{L}_{\text{app}} + \mathcal{L}_{\text{sil}} + \mathcal{L}_{\text{id}} + \mathcal{L}_{\text{cr}}, \tag{11}$$

where $\mathcal{L}_{\text{app}}$ is the L_1 reconstruction loss for local appearances:

$$\mathcal{L}_{\text{app}} = ||P_{\text{local}} \odot M_{\text{sil}} - P'_{\text{app}} \odot M_{\text{sil}}||_1, \tag{12}$$

and $\mathcal{L}_{\text{sil}}$ is the L_1 reconstruction loss for M_{sil}. $\mathcal{L}_{\text{id}}$ and $\mathcal{L}_{\text{cr}}$ are self-supervised losses introduced in Sec. 4.3.

For spatially-varying lighting estimator, $\mathcal{L}_E$ is defined as:

$$\mathcal{L}_E = \mathcal{L}_{\text{vis}} + \mathcal{L}_{\text{pos}} + \mathcal{L}_{\text{code}} + \mathcal{L}_{\text{pano}}, \tag{13}$$

where $\mathcal{L}_{\text{vis}}$ is an L_2 loss for M_{vis}, $\mathcal{L}_{\text{pos}}$ is the cross-entropy loss for flattened M_{sun}, $\mathcal{L}_{\text{code}}$ is an L_2 loss for z_{sky}, z_{sun}, and z_{local}. We also impose $\mathcal{L}_{\text{pano}}$, an L_1 loss on decoded/rendered panoramic maps P'_{sky}, P'_{sun}, P'_{sil}, and P'_{app}.

7.5 Implementation of Baseline Methods

SOLID-Net [13]. We train SOLID-Net [13] on our enhanced dataset using their original code implementation. Following their paper, we render and compute normal maps, depth maps, (diffuse) shading maps, (diffuse) shadow maps and albedo maps as learning targets to train their I-Net. Following their geometry

projection, we prepare incomplete panoramas projected from the limited-FoV images at different local positions as the input of their P-Net. Their I-Net and P-Net are trained separately for 30 epochs and 20 epochs respectively.

SkyNet [3]. We implement SkyNet [3] in PyTorch [10] following the network structure shown in their paper. SkyNet [3] is designed to separately predict a sun azimuth angle and a normalized panoramic environment map where the sun is azimuth-centered. To fit their data requirement, we compute the sun azimuth of our data and compute the normalized panoramic environment maps as the training targets of SkyNet [3]. Their sky autoencoder and image encoders are trained separately for 30 epochs and 20 epochs respectively.

Ours$_{\mathbf{SH}}$. We use the same network backbone for Ours$_{\mathrm{SH}}$ as our proposed method (see Sec. 7.7) to predict 5-th order SH coefficients as representation for local lighting. The ground truth SH coefficients can be directly computed from the orthogonality of SH basis. The model is trained for 30 epochs.

Ours$_{\mathbf{SG}}$. We use the same network backbone for Ours$_{\mathrm{SG}}$ as our proposed method (see Sec. 7.7) to predict 24 SG lobes as representation for local lighting. The corresponding parameters of SG lobes may not be unique given an environment map, which poses a problem to use SG parameters as learning targets. Here we follow [7] to reparameterize SG parameters to constrain each lobe in the certain range of the sphere (we roughly divide the sphere into 3×8 regions) and optimize the ground truth SG parameters by LBFGS method [8]. Note that this process is very time-consuming[5], meaning that using optimized SG lobes to represent outdoor lighting might not be an efficient choice. The model is trained for 30 epochs.

7.6 Additional Results

Qualitative results. We show the quantitative results of sun position estimation on our captured real data in Table 4[6]. We can see from the figures that though the real data are more challenging than synthetic dataset, our method still achieves a better performance than baseline methods. It's worth noting that SOLID-Net [13] trained on synthetic dataset *before enhancement* (SOLID-Net w/o en) shows an significant drop compared with the same model trained on synthetic dataset *after enhancement* (SOLID-Net), indicating the enhanced material diversity is helpful for real-world performance. We also show the quantitative evaluation of local lighting estimation and relighting performance on our captured real data in Table 5, and the trends are similar as on our enhanced synthetic dataset, indicating our method can consistently produce more realistic relighting results.

Synthetic dataset enhancement. To further confirm the effectiveness of our synthetic dataset enhancement, we test the SOLID-Net [13] trained on synthetic

[5] https://github.com/lzqsd/InverseRenderingOfIndoorScene#differences-from-the-original-paper

[6] Note that Ours$_{\mathrm{SH}}$ and Ours$_{\mathrm{SG}}$ is not designed for global sun position estimation, we use the maximum points of the predicted local lighting maps given the pixel positions in the bright regions as the predicted sun positions only as a reference here.

dataset without and with enhancement on the same real-captured data, and the qualitative results are shown in Figure 12. As we can seen, the SOLID-Net [13] with enhanced dataset (blue) can recover clearer scene layouts and colors in ground truth (green) than without enhancement (red) with the presence of irregular grasses and leafs, which would lead to noisy intrinsic estimation results.

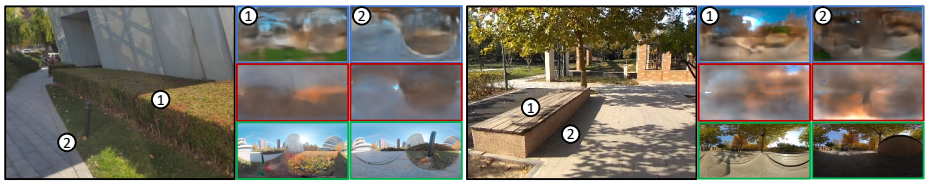

Fig. 12. Local lighting estimation results on real data of SOLID-Net [13] trained on synthetic dataset without enhancement (the first row, blue box) and with enhancement (the second row, red box), compared with the ground truth local lighting (the third row, green box). Panoramas are shown in tone-mapped HDR.

Controlled scene test. To better understand the bounds of our proposed method, we conduct a controlled scene test in Figure 13. As we can see, our method works well as expected in the sunny (hard shadow) urban scenes and can generalize to (partly-)cloudy (soft shadow) weathers (Figure 13 left and middle) and different proximity to the buildings (Figure 13 middle and right). The two known common types of failure cases (Figure 13 left) happen around complex local shapes (blue point) and highlight glass reflections (orange point).

Fig. 13. Local lighting estimation and relighting results of controlled scene test. The red (green) points in three images correspond to the same spot which is sun-(in)visible. The organge and blue points show two typical types of failure cases. Colors of boxes indicate the correspondence with local pixel points marked in the same color. Panoramas and relighting results are shown in tone-mapped HDR.

Editability. To further illustrate the editability of our disentangled representation of global and local lighting, we show more qualitative examples. For global lighting editing, we show examples of sun position editing in Figure 14, sun information editing in Figure 15, and sky information editing in Figure 16. For local lighting editing, we show more results in Figure 17. We can see from the results that by

Table 4. Quantitative evaluation of sun position estimation on our captured real data. We report the average angular error $\mathcal{E}_{\text{ang}}$, azimuth error $\mathcal{E}_{\text{az}}$, and elevation error $\mathcal{E}_{\text{el}}$ in degrees. The results of Ours_{SH} and Ours_{SG} are calculated from the predicted local lighting maps for reference purpose. Lower is better.

Method	$\mathcal{E}_{\text{ang}}$	$\mathcal{E}_{\text{az}}$	$\mathcal{E}_{\text{el}}$
Ours	**21.15**	**20.76**	**8.44**
SkyNet [3]	32.83	32.81	14.20
SOLID-Net [13]	30.82	33.92	11.95
SOLID-Net [13] w/o en	49.07	61.20	**8.03**
$\textbf{Ours}_{\textbf{SH}}$	48.91	64.55	21.96
$\textbf{Ours}_{\textbf{SG}}$	60.87	68.84	24.31

Table 5. Quantitative evaluation of spatially-varying local lighting estimation on our captured real data. 'Panorama' stands for the errors of panoramic HDR lighting maps before tone mapping. 'Relighting' stands for the errors of rendered HDR images using predicted lighting maps. Lower is better.

Method	Panorama		Relighting	
	MAE	RMSE	MAE	RMSE
Ours	0.240	4.872	**0.259**	**0.437**
Ours w/o M_{vis}	0.274	6.496	0.299	0.552
$\textbf{Ours}_{\textbf{SH}}$	0.244	**2.367**	0.269	0.469
$\textbf{Ours}_{\textbf{SG}}$	**0.179**	2.748	0.318	0.560
SOLID-Net [13]	0.390	4.206	0.310	0.601

predicting disentangled global and local representations according to our problem formulation, the predicted global and local lighting remain flexible editability.

Quantitative results. More quanlitative results of global lighting estimation and local lighting estimation are shown in Figure 18 and Figure 19 respectively. In each figure, we show the estimated lighting maps, relighting results using the (predicted) lighting maps and the quantitative error metrics of the lighting maps and relighting results. Our method generally performs better qualitatively than baseline methods while may not be the best in the error metrics, showing the inconsistency of the quantitative metrics and visual perception.

In-the-wild performance. To test the generalization ability of the compared methods, we show the local relighting results of in-the-wild data in Figure 20 and Figure 21. The in-the-wild data are collected from the Google Street View dataset [12] and Internet photos, which cover diverse scenes and lighting conditions. Although the data distribution is far different from our synthetic training data, our proposed method can capture many spatially-varying properties of outdoor local lighting and give reasonable relighting results.

7.7 Detailed Network Archetectures

We show the detailed network archtectures of the global lighting encoder-decoder, local content encoder-renderer, spatially-varying lighting estimator and our baseline methods Ours_{SH} and Ours_{SG} from Table 6 to Table 9, with the structures and default settings of common blocks shown in Figure 22.

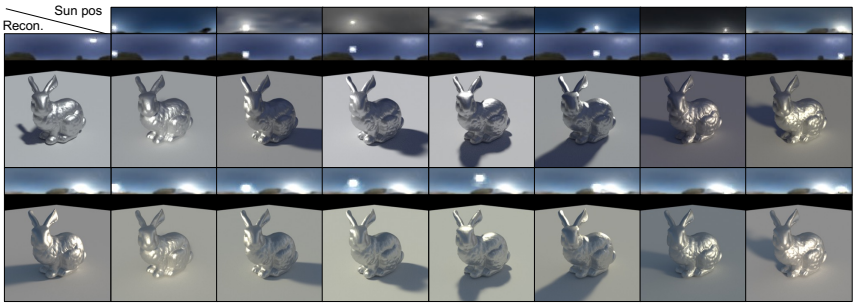

Fig. 14. Global lighting editing by changing the sun positions (in each row) and the relighting results (shown in tone-mapped HDR).

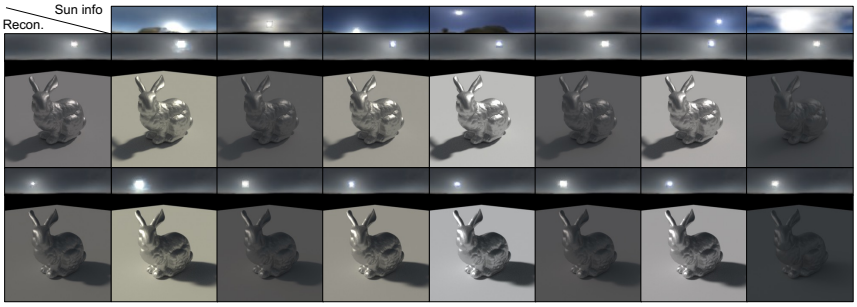

Fig. 15. Global lighting editing by changing the sun info (in each row), and the relighting results (shown in tone-mapped HDR).

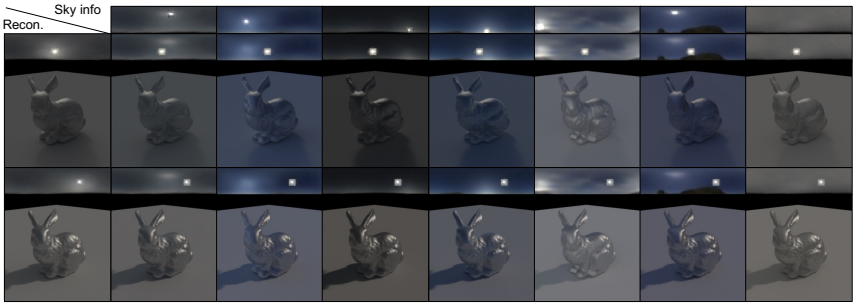

Fig. 16. Global lighting editing by changing the sky info (in each row), and the relighting results (shown in tone-mapped HDR).

Fig. 17. Local lighting editing by changing global lighting conditions (in each row) and local contents (in each column), and the relighting results (shown in tone-mapped HDR).

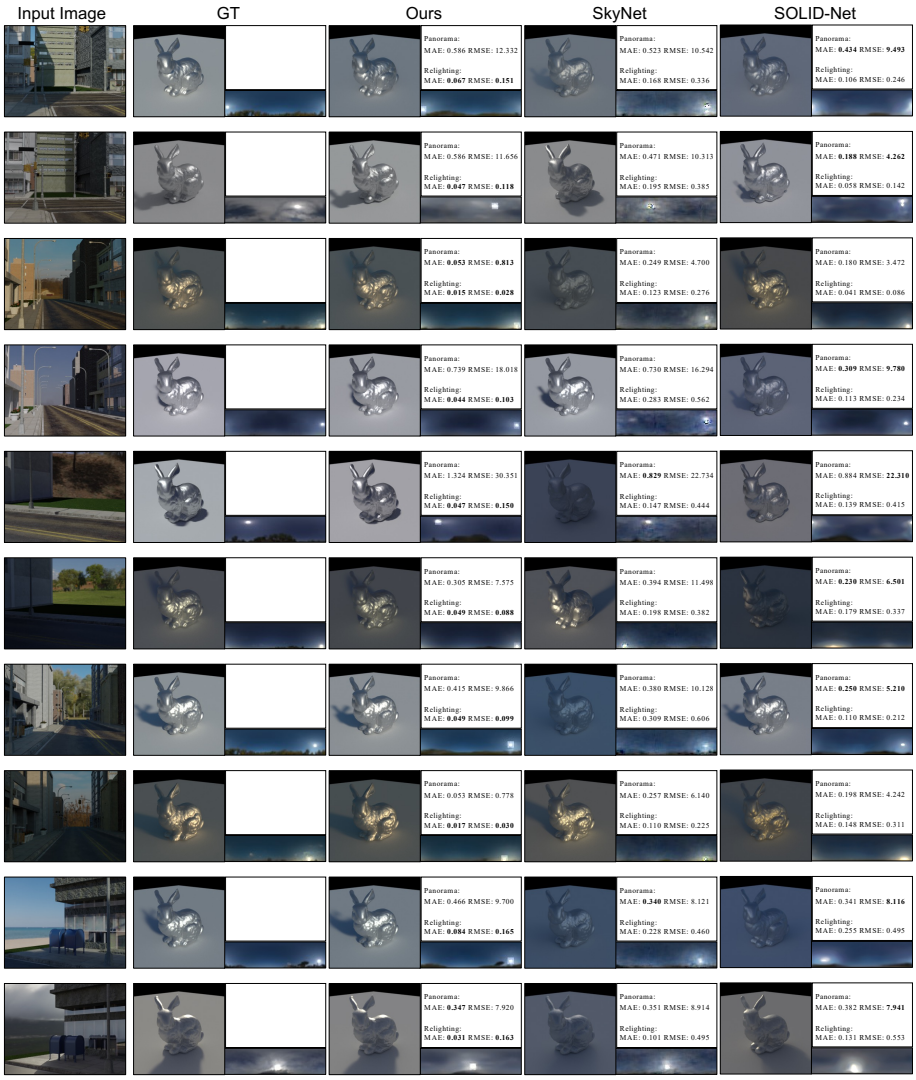

Fig. 18. Global lighting estimation and relighting results on our enhanced synthetic dataset (shown in tone-mapped HDR). Errors of instances are shown in text boxes.

Fig. 19. Local lighting estimation and relighting results (shown in tone-mapped HDR). The first three examples are from our enhanced synthetic dataset, and the last five examples are from our captured real data. Errors of instances are shown below each image (from left to right: MAE of panoramic maps, RMSE of panoramic maps, MAE of relighting results and RMSE of relighting results).

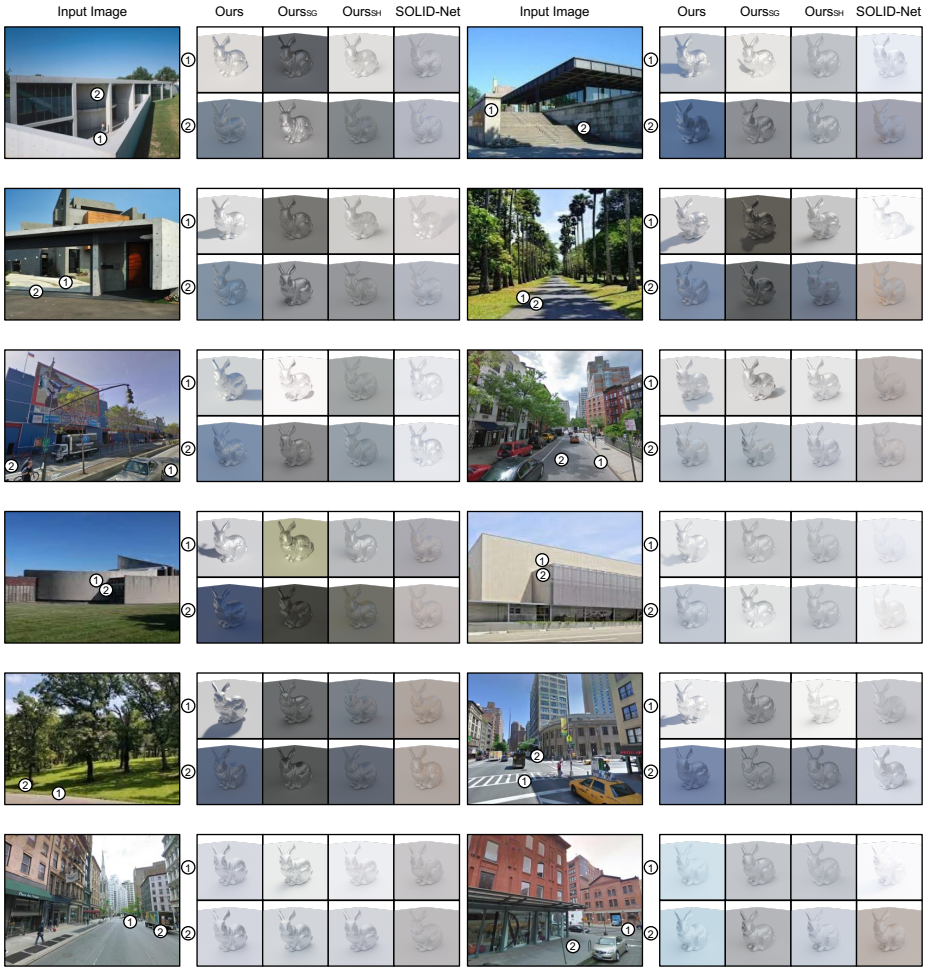

Fig. 20. More examples of local lighting estimation on in-the-wild data.

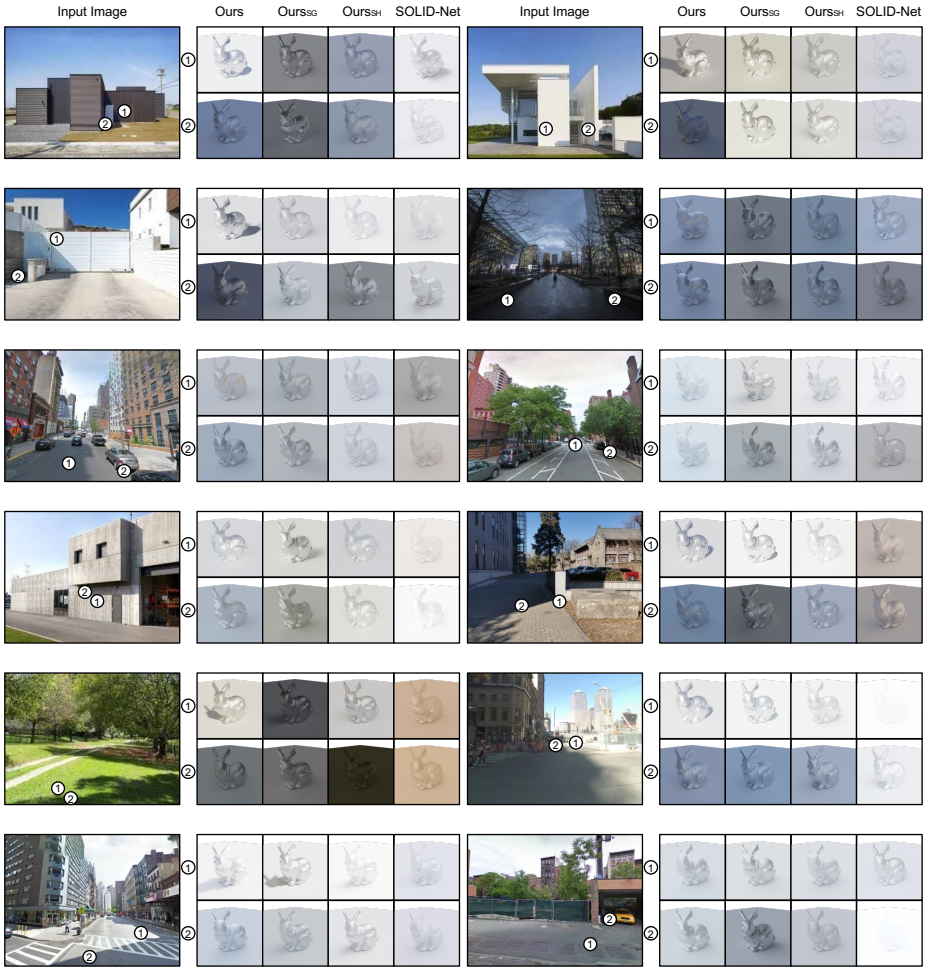

Fig. 21. More examples of local lighting estimation on in-the-wild data.

Table 6. Network architectures of our global lighting encoder-decoder.

Global Lighting Encoder-Decoder			
Name	Layer Description	Input	Output Dim.
P_{global}	Global lighting maps	–	(3, 64, 128)
z_{sky}	Latent code for sky light	P_{global}	(16)
z_{sun}	Latent code for sun light	P_{global}	(45)
M_{sun}	Mask for sun position	P_{global}	(1, 64, 128)
Encoders for z_{sky} ($c_{\text{out}} = 16$) and z_{sun} ($c_{\text{out}} = 45$)			
Conv1_1	Conv., k=5, s=1	P_{global}	(32, 32, 128)
Conv1_2	Res. I, k=3, s=1	Conv1_1	(32, 32, 128)
Conv1_3	Conv., k=3, s=2	Conv1_2	(64, 16, 64)
Conv2_1	Conv., k=3, s=1	Conv1_3	(64, 16, 64)
Conv2_2	Res. I, k=3, s=1	Conv2_1	(64, 16, 64)
Conv2_3	Conv., k=3, s=2	Conv2_2	(128, 8, 32)
Conv3_1	Conv., k=3, s=1	Conv2_3	(128, 8, 32)
Conv3_2	Res. I, k=3, s=1	Conv3_1	(128, 8, 32)
Conv3_3	Conv., k=3, s=2	Conv3_2	(128, 4, 16)
Conv4_1	Conv., k=3, s=1	Conv3_3	(128, 4, 16)
Conv4_2	Res. I, k=3, s=1, no norm	Conv4_1	(128, 4, 16)
Conv4_3	Conv., k=3, s=2, no norm	Conv4_2	(64, 2, 8)
Conv5	Conv., k=3, s=1, no norm, no activ	Conv4_3	(c_{out}, 2, 8)
Pool	Global Avg. Pool.	Conv5	(c_{out})
Decoder for P_{sky}			
Fc	Linear 16×512, and reshape	z_{sky}	(8, 4, 16)
Conv1_1	Conv., k=3, s=1	Fc	(64, 4, 16)
Conv1_2	2×Res. I, k=3, s=1	Conv1_1	(64, 4, 16)
Up1	Deconv., k=3, s=1	Conv1_2	(64, 8, 32)
Conv2_1	Conv., k=3, s=1	Up1	(128, 8, 32)
Conv2_2	2×Res. I, k=3, s=1	Conv2_1	(128, 8, 32)
Up2	Deconv., k=3, s=1	Conv1_2	(128, 16, 64)
Conv3_1	Conv., k=3, s=1	Up2	(64, 16, 64)
Conv3_2	2×Res. I, k=3, s=1	Conv3_1	(64, 16, 64)
Up3	Deconv., k=3, s=1	Conv1_2	(64, 32, 128)
Conv4_1	Conv., k=3, s=1	Up3	(32, 32, 128)
Conv4_2	2×Res. I, k=3, s=1, no norm	Conv4_1	(32, 32, 128)
Conv5_1	Conv., k=3, s=1, no norm	Conv4_2	(16, 32, 128)
Conv5_2	Conv., k=3, s=1, no norm, no activ	Conv5_1	(3, 32, 128)
Decoder for P_{sun}			
Merge	repeat z_{sun} and concate with M_{sun}	z_{sun}, M_{sun}	(46, 32, 128)
Conv1_1	Conv., k=3, s=1	Merge	(64, 32, 128)
Conv1_2	Res. I, k=3, s=1	Conv1_1	(64, 32, 128)
Conv2_1	Conv., k=3, s=1	Conv1_2	(128, 32, 128)
Conv2_2	Res. I, k=3, s=1	Conv2_1	(128, 32, 128)
Conv3_1	Conv., k=3, s=1	Conv2_2	(64, 32, 128)
Conv3_2	Res. I, k=3, s=1	Conv3_1	(64, 32, 128)
Conv4_1	Conv., k=3, s=1	Conv3_2	(32, 32, 128)
Conv4_2	2×Res. I, k=3, s=1, no norm	Conv4_1	(32, 32, 128)
Conv5_1	Conv., k=3, s=1, no norm	Conv4_2	(16, 32, 128)
Conv5_2	Conv., k=3, s=1, no norm, no activ	Conv5_1	(3, 32, 128)

Table 7. Network architectures of our local appearance encoder-renderer.

Local Appearance Encoder-Renderer			
Name	Layer Description	Input	Output Dim.
P_{sv}	Local lighting maps	–	(3, 64, 128)
z_{local}	Latent code for local content	P_{sv}	(64)
z_{sky}	Latent code for sky light	P_{global}	(16)
z_{sun}	Latent code for sun light	P_{global}	(45)
M_{cos}	Cosine mask for sun light	M_{sun}	(1, 64, 128)
Encoder for z_{local}			
Conv1_1	Conv., k=5, s=1	P_{global}	(32, 64, 128)
Conv1_2	Res. I, k=3, s=1	Conv1_1	(32, 64, 128)
Conv1_3	Conv., k=3, s=2	Conv1_2	(64, 32, 64)
Conv2_1	Conv., k=3, s=1	Conv1_3	(64, 32, 64)
Conv2_2	Res. I, k=3, s=1	Conv2_1	(64, 32, 64)
Conv2_3	Conv., k=3, s=2	Conv2_2	(128, 16, 32)
Conv3_1	Conv., k=3, s=1	Conv2_3	(128, 16, 32)
Conv3_2	Res. I, k=3, s=1	Conv3_1	(128, 16, 32)
Conv3_3	Conv., k=3, s=2	Conv3_2	(128, 8, 16)
Conv4_1	Conv., k=3, s=1	Conv3_3	(128, 8, 16)
Conv4_2	Res. I, k=3, s=1, no norm	Conv4_1	(128, 8, 16)
Conv4_3	Conv., k=3, s=2, no norm	Conv4_2	(64, 4, 8)
Conv5	Conv., k=3, s=1, no norm, no activ	Conv4_3	(64, 2, 8)
Pool	Global Avg. Pool.	Conv5	(64)
Decoder for M_{sil}			
Fc	Linear 64 × 256, and reshape	z_{local}	(8, 4, 8)
Conv1_1	Conv., k=3, s=1	Fc	(32, 4, 8)
Up1	Deconv., k=3, s=1, nearest upsample	Conv1_1	(32, 8, 16)
Conv1_2	2×Res. I, k=3, s=1	Up1	(32, 8, 16)
Conv2_1	Conv., k=3, s=1	Conv1_2	(64, 8, 16)
Up2	Deconv., k=3, s=1, nearest upsample	Conv2_1	(64, 16, 32)
Conv2_2	2×Res. I, k=3, s=1	Up2	(64, 16, 32)
Conv3_1	Conv., k=3, s=1	Conv2_2	(64, 16, 32)
Up3	Deconv., k=3, s=1, nearest upsample	Conv3_1	(64, 32, 64)
Conv3_2	2×Res. I, k=3, s=1	Up3	(64, 32, 64)
Conv4_1	Conv., k=3, s=1	Conv3_2	(32, 32, 64)
Up4	Deconv., k=3, s=1, nearest upsample	Conv4_1	(32, 64, 128)
Conv4_2	2×Res. I, k=3, s=1	Up4	(32, 64, 128)
Conv5_1	Conv., k=3, s=1, no norm	Conv4_2	(16, 64, 128)
Conv5_2	Conv., k=3, s=1, no norm, Sigmoid	Conv5_1	(1, 64, 128)
Renderer for P_{local}			
Concat1	concate z_{local} with z_{sky}	z_{local}, z_{sky}	(80)
Fc	Linear 80 × 1024, and reshape	Concat1	(8, 8, 16)
Conv1_1	Conv., k=3, s=1	Fc	(64, 8, 16)
Conv1_2	2×Res. I, k=3, s=1	Conv1_1	(64, 8, 16)
Up1	Deconv., k=3, s=1	Conv1_2	(64, 16, 32)
Conv2_1	Conv., k=3, s=1	Up1	(128, 16, 32)
Conv2_2	2×Res. I, k=3, s=1	Conv2_1	(128, 16, 32)
Up2	Deconv., k=3, s=1	Conv2_2	(128, 32, 64)
Conv3_1	Conv., k=3, s=1	Up2	(128, 32, 64)
Conv3_2	2×Res. I, k=3, s=1	Conv3_1	(128, 32, 64)
Up3	Deconv., k=3, s=1	Conv3_2	(128, 64, 128)
Conv4_1	Conv., k=3, s=1	Up3	(128, 64, 128)
Conv4_2	2×Res. I, k=3, s=1, no norm	Conv4_1	(128, 64, 128)
Concat2	concate M_{cos} with Conv4_2	Conv4_2, M_{cos}	(129, 64, 128)
Merge	repeat z_{sun} and concate with Concat2	Concat2, z_{sun}	(174, 64, 128)
Conv5_1	Conv., k=3, s=1	Merge	(128, 64, 128)
Conv5_2	2×Res. I, k=3, s=1	Conv5_1	(128, 64, 128)
Conv6_1	Conv., k=3, s=1	Conv5_2	(128, 64, 128)
Conv6_2	2×Res. I, k=3, s=1	Conv6_1	(128, 64, 128)
Conv7_1	Conv., k=3, s=1	Conv6_2	(64, 64, 128)
Conv7_2	2×Res. I, k=3, s=1	Conv7_1	(64, 64, 128)
Conv8_1	Conv., k=3, s=1	Conv7_2	(32, 64, 128)
Conv8_2	2×Res. I, k=3, s=1	Conv8_1	(32, 64, 128)
Conv9_1	Conv., k=3, s=1, no norm	Conv8_2	(16, 64, 128)
Conv9_2	Conv., k=3, s=1, no norm, no activ	Conv9_1	(3, 64, 128)

Table 8. Network architectures of lighting estimator and baselines.

Spatially-varying Lighting Estimator			
Name	Layer Description	Input	Output Dim.
I	Limited-FoV images	–	(3, 240, 320)
l	local pixel positions	–	(2)
F_{global}	extracted global feature	I	(512, 30, 40)
F_{local}	extracted local feature	F_{global}	(128, 60, 80)
F_{pix}	pixel-aligned local feature	F_{local}	(128, 3, 3)
Global Feature Backbone for F_{global}			
Conv1	Conv., k=7, s=1	I	(64, 240, 320)
Conv2	Conv., k=4, s=2	Conv1	(128, 120, 160)
Conv3	Conv., k=4, s=2	Conv2	(256, 60, 80)
Conv4	Conv., k=4, s=2	Conv3	(512, 30, 40)
Conv5	2×Res. I, k=3, s=1	Conv4	(512, 30, 40)
Estimator for M_{vis}			
Conv1	2×Res. I, k=3, s=1	F_{global}	(512, 30, 40)
Up1	Deconv., k=5, s=1, ReLU	Conv1	(256, 60, 80)
Up2	Deconv., k=5, s=1, ReLU	Conv2	(128, 120, 160)
Up3	Deconv., k=5, s=1, ReLU	Conv3	(64, 240, 320)
Conv2	Conv., k=7, s=1, no norm, Sigmoid	Up3	(1, 240, 320)
Estimators for z_{sky} ($c_{\text{out}} = 16$) and z_{sun} ($c_{\text{out}} = 45$)			
Conv1_1	Conv., k=3, s=1	F_{global}	(256, 30, 40)
Conv1_2	Res. I, k=3, s=1	Conv1_1	(256, 30, 40)
Conv2_1	Conv., k=3, s=2	Conv1_2	(128, 15, 20)
Conv2_2	Res. I, k=3, s=1	Conv2_1	(128, 15, 20)
Conv3_1	Conv., k=3, s=2	Conv2_2	(128, 8, 10)
Conv3_2	Res. I, k=3, s=1	Conv3_1	(128, 8, 10)
Conv4_1	Conv., k=3, s=1	Conv3_2	(64, 8, 10)
Conv4_2	Res. I, k=3, s=1	Conv4_1	(64, 8, 10)
Conv5_1	Conv., k=3, s=2, no norm	Conv4_2	(32, 4, 5)
Conv5_2	Conv., k=3, s=1, no norm, no activ	Conv5_1	(c_{out}, 4, 5)
Pool	Global Avg. Pool.	Conv5_2	(c_{out})
Estimators for M_{sun}			
Conv1_1	Conv., k=3, s=1	F_{global}	(256, 30, 40)
Conv1_2	Res. I, k=3, s=1	Conv1_1	(256, 30, 40)
Pool1	Avg. Pool., k=2, s=2	Conv1_2	(256, 15, 20)
Conv2	Conv., k=3, s=1	Pool1	(128, 15, 20)
Pool2	Avg. Pool., k=5, s=5	Conv2	(128, 3, 4)
Fc	reshape and Linear 1536×256	Pool2	(256)
Softmax	Softmax and reshape	Fc	(1, 8, 32)
Estimators for z_{local} ($c_{\text{out}} = 64$) or SH coefficients in Ours_{SH} ($c_{\text{out}} = 108$) or SG lobe axes ($c_{\text{out}} = 48$), lambdas ($c_{\text{out}} = 24$) and weights ($c_{\text{out}} = 72$) in Ours_{SG}			
Extract	3×3 feature patch of F_{pix} pixel-aligned to l	F_{local}, l	(128, 3, 3)
Reshape	flatten	Extract	(1152)
Percp1	Linear 1152×512, and LReLU (0.1)	Reshape	(512)
Percp2	Linear 512×256, and LReLU (0.1)	Percp1	(256)
Percp3	Linear 256×128, and LReLU (0.1)	Percp2	(128)
Percp4	Linear $128 \times c_{\text{out}}$, and LReLU (0.1)	Percp3	(c_{out})

Table 9. Network architectures of stacked hourglass structure.

Stacked Hourglass Structure for F_{local}			
Up	ConvT., k=4, s=2, IN, LReLU (0.1)	F_{global}	(256, 60, 80)
Conv1	Res. I, k=3, s=1	Up	(256, 60, 80)
Hg1	*Hourglass structure of depth=2*	Conv1	(256, 60, 80)
Conv1_t	Res. II, k=3, s=1	Hg1	(256, 60, 80)
Conv1_l	Conv., k=1, s=1, no norm, no activ	Conv1_t	(128, 60, 80)
Conv1_a	Conv., k=1, s=1, no norm, no activ	Conv1_l	(256, 60, 80)
Conv1_b	Conv., k=1, s=1, no norm, no activ	Conv1_t	(256, 60, 80)
Input2	Input2 = Conv1_t+Conv1_a+Conv1_b		(256, 60, 80)
Hg2	Hourglass structure of depth=2	Input2	(256, 60, 80)
Conv2_t	Res. II, k=3, s=1	Hg2	(256, 60, 80)
Conv2_l	Conv., k=1, s=1, no norm, no activ	Conv2_t	(128, 60, 80)

Hourglass Structure of depth=2			
Name	Layer Description	Input	Output Dim.
F	feature for hourglass input	–	(C, H, W)
Conv1_b1	Res. II, k=3, s=1	F	(C, H, W)
Pool1	Avg. Pool., k=2, s=2	F	$(C, H/2, W/2)$
Conv1_b2	Res. II, k=3, s=1	Pool1	$(C, H/2, W/2)$
Conv2_b1	Res. II, k=3, s=1	Conv1_b2	$(C, H/2, W/2)$
Pool2	Avg. Pool., k=2, s=2	Conv1_b2	$(C, H/4, W/4)$
Conv2_b2	Res. II, k=3, s=1	Pool2	$(C, H/4, W/4)$
Conv2_b2p	Res. II, k=3, s=1	Pool2	$(C, H/4, W/4)$
Conv2_b3	Res. II, k=3, s=1	Conv2_b2p	$(C, H/4, W/4)$
Up2	Deconv., no conv, no activ	Conv2_b3	$(C, H/2, W/2)$
Output2	Output2 = Conv2_b1 + Up2		$(C, H/2, W/2)$
Conv1_b3	Res. II, k=3, s=1	Output2	$(C, H/2, W/2)$
Up1	Deconv., no conv, no activ	Conv1_b3	(C, H, W)
Output1	Output1 = Conv1_b1 + Up1		(C, H, W)
Output	output of the hourglass	Output1	(C, H, W)

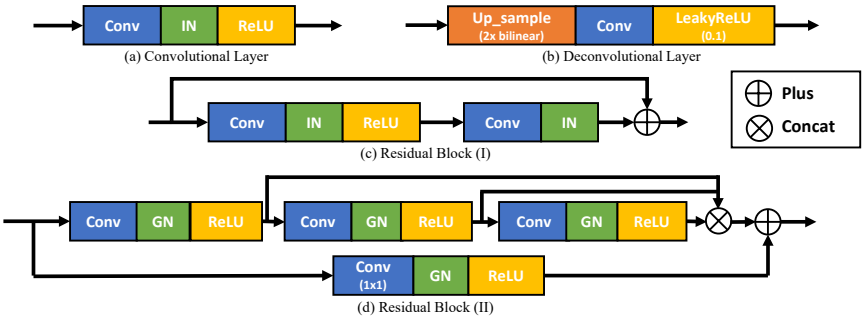

Fig. 22. Common blocks used in our network architectures (with default settings).

References

1. ambientCG, public domain materials for physically based rendering. [licensed under CC0 1.0 universal], https://ambientcg.com 17
2. Blender., https://www.blender.org 17, 18
3. Hold-Geoffroy, Y., Athawale, A., Lalonde, J.F.: Deep sky modeling for single image outdoor lighting estimation. In: Proc. of Computer Vision and Pattern Recognition (2019) 18, 20, 22
4. Kalantari, N.K., Ramamoorthi, R.: Deep high dynamic range imaging of dynamic scenes. ACM Transactions on Graphics (Proc. of ACM SIGGRAPH) **36**(4) (2017) 18
5. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Proc. of International Conference on Learning Representations (2015) 18
6. Lalonde, J.F., Asselin, L.P., Becirovski, J., Hold-Geoffroy, Y., Garon, M., Gardner, M.A., Zhang, J.: The laval HDR sky database. [free license for academic or government-sponsored researchers] (2016), http://sky.hdrdb.com 18
7. Li, Z., Shafiei, M., Ramamoorthi, R., Sunkavalli, K., Chandraker, M.: Inverse rendering for complex indoor scenes: Shape, spatially-varying lighting and SVBRDF from a single image. In: Proc. of Computer Vision and Pattern Recognition (2020) 20
8. Liu, D.C., Nocedal, J.: On the limited memory BFGS method for large scale optimization. Mathematical Programming **45**(1-3), 503–528 (1989) 20
9. Otsu, N.: A threshold selection method from gray-level histograms. IEEE Transactions on Systems, Man, and Cybernetics **9**(1), 62–66 (1979) 18
10. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: PyTorch: An imperative style, high-performance deep learning library. In: Proc. of Neural Information Processing Systems (2019) 18, 20
11. Yu, P., Guo, J., Huang, F., Zhou, C., Che, H., Ling, X., Guo, Y.: Hierarchical disentangled representation learning for outdoor illumination estimation and editing. In: Proc. of International Conference on Computer Vision (2021) 19
12. Zamir, A.R., Shah, M.: Image geo-localization based on multiple nearest neighbor feature matching using generalized graphs. IEEE Transactions on Pattern Analysis and Machine Intelligence **36**(8), 1546–1558 (2014) 22
13. Zhu, Y., Zhang, Y., Li, S., Shi, B.: Spatially-varying outdoor lighting estimation from intrinsics. In: Proc. of Computer Vision and Pattern Recognition (2021) 17, 19, 20, 21, 22