Supplementary Material: Cross-Domain Synthetic-to-Real In-the-Wild Depth and Normal Estimation for 3D Scene Understanding

Jay Bhanushali[1] Manivannan Muniyandi[1] Praneeth Chakravarthula[2]
[1]Haptics Lab, Indian Institute of Technology Madras
[2]UNC Chapel Hill

1. Supplementary Material

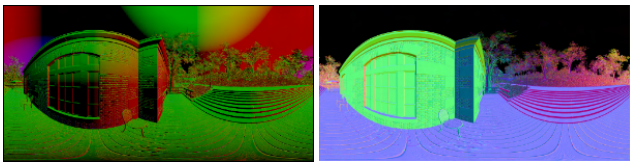
In this supplementary material, we discuss our approach on generating the OmniHorizon dataset in Unreal Engine 4. We elaborate on the factors and certain assumptions that we made in order to render the dataset. Additionally, we discuss about training the UBotNet on indoor datasets and architecture choices. Finally, we demonstrate additional results for depth and normal estimation from real-world images in the wild.

1.1. Depth clamping

Rendering engines such as Unreal Engine 4 work with a larger depth range compared to that captured by physical sensors. However, we were interested in exploring the range of depth information that can be used for covering a wide range of objects in outdoor scenarios. This motivated us to simulate the limitations of the physical sensors and restrict the depth range to 150 m, similar to the Fukuoka dataset [5]. The engine places the far plane at infinity, which results in depth values being generated for extremely distant objects. To avoid this, we modify the depth material to visualise the impact of constraining the depth to a maximum specified value. We show the results for the clamping of depth at a range of 10m, 75m and 150m in Figure 2. At a depth of 10 m, only the truck is visible. When the depth range is raised to 75 m, cars and building start to appear in the background. At 150 m, the trees and most of the background are visible. By limiting the depth in outdoor environments, it is possible to focus solely on nearby items, or, depending on the application, on distant objects as well.

1.2. View-space vs world-space normals

The view space normals are calculated relative to the camera orientation, whereas the world space normals are calculated with respect to the global axes of the scene. The normals in view space are desired when using a perspective camera as they are tied to the camera pose (extrinsic parameters). However, the panoramic image is obtained by rotating the camera on both the horizontal and vertical axis in increments of fixed angle steps (5°), followed by merging the multiple views.



(a) view-space normals (b) world-space normals

Figure 1. *Comparison between view-space and world-space normals.* The normals captured in view-space appear as gradient with lack of clear distinction between the basis vectors. Normal maps recorded in world-space follow a consistent coordinate system.

Since the coordinate system is relative to the camera in view space, it also gets modified with the rotation. This results in a gradient of normals with no basis vectors. The normals obtained in world space are absolute and independent of camera pose. Figure 1 shows the difference between the view-space and world-space normals. Therefore, we captured the normals in world space as it was consistent for both within and between the scenes. We show the convention used for the world-space normals in Figure 3.

1.3. Virtual Avatars

As discussed in main paper, we utilised Metahumans [4] for the virtual avatars in the scene. We have used premade MetaHumans available in the Quixel bridge. It allowed us to bring in highly detailed characters and more diversity in the pedestrians. But there were certain challenges while using the Metahumans for the dataset. They are generated with multiple level of details (LODs) for perfomance optimisation. As a result, there would be sudden popups and other artifacts when the camera is approaching a character. Figure 4 illustrates how the character hair and details change when the camera is approaching the character. Lower LOD level (LOD 8) indicates lowest detailed polygon mesh with no advanced features such as detail normal maps or hairs. The higher LOD level (Level 0/1) has higher polygons with extra detail maps for the skin and hair grooming system.

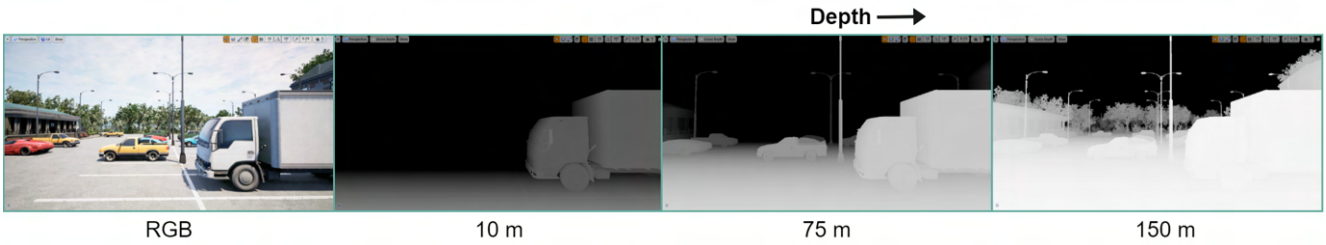


Figure 2. *Depth clamping experiment.* Comparison between various depth ranges after clamping to a specific range: 10 m, 75 m and 150 m. Inverted depth maps are shown for better visualization.

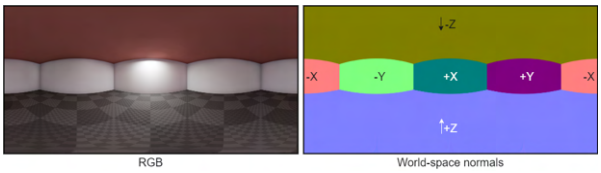


Figure 3. *Convention for the world-space normals.*

Additionally, we also observed artifacts in the normal maps for the characters with detailed grooming such as facial hair. Figure 5 shows the issues with the normal maps of a character in the region with facial hair. For such characters we used LOD 1 or LOD 2 to resolve the problems.

1.4. Assumptions in the Dataset

Our dataset renders several realistic outdoor and indoor environments with dynamic scene components. While curating this dataset, we made certain assumptions especially about the outdoor scenes which we list below:

1. The sky is assumed to be situated at infinitely large distance from the camera, and is represented as a spherical mesh of large radius encompassing the entire scene. Additionally, normals are not rendered for the sky region. It is represent using black which indicates invalid normal values. This allows us to distinguish sky from other regions in the scene.

2. Transparent and transluscent materials such as water, windows of the buildings and windshields of vehicles are replaced with fully reflective materials. We observed that inferring depth of such materials from color images is challenging and this limitation, for example, also applies to real-world datasets captured using lidars [8]. Figure 6 depicts the limitation of using transparent and translucent materials in the dataset. The original water shader in the scene was designed in such a way that it acted as a see-through material in case of depth. As a result, the depth map captures the terrain hidden underneath the water surface. We modified the the water shader to a reflective surface and thus depth is correctly rendered as a planar surface. We observed a similar case for the glass shader used for windows in the vehicles. The vehicles indeed have detailed indoors but due to reflections on the glass, the inside is not clearly visible. However, the depth map has much cleaner view of the indoors. To avoid this conflict of information, we use fully opaque and reflective materials for the windows.

2. UBotNet

UBotNet for Indoor datasets. In the main paper, we discussed about the UBotNet architecture and the results from training on the OmniHorizon dataset. We additonally trained UBotNet on real-world indoor dataset Pano3D [1] to validate the performance of the network on other datasets. Pano3D is proposed as a modification of Matteport3D [2] and Gibson3D [10]. We used the official splits provided by the authors for Matterport3D for training and validation. For, Gibson, we used the *GibsonV2 Full Low Resolution* for training and validated on Matterport. All the images used for training were of 512 x 256 resolution. We used the loss function and training parameters outlined in our main paper. We trained UBotNet Lite on the both the datasets for 60 epochs.

Table 1. *Quantitative results for depth estimation using UBotNet Lite validated on indoor dataset - Matterport3D.*

	Depth Error ↓			Depth Accuracy ↑		
Dataset	RMSE	MRE	RMSE log	$\delta 1$	$\delta 2$	$\delta 3$
Matterport3D	0.639	0.142	0.064	0.817	0.952	0.981
Gibson 3D	0.591	0.154	0.061	0.830	0.965	0.986

Table 1 shows the quantitative results for the task of depth estimation by UBotNet Lite evaluated on Matterport3D. We also show the qualitative results for the validation task in Figure 8. We observed better performance in overall metrics and the visual results when the network is trained on the Gibson3D.

Figure 4. *Dynamic LODs vs Constant LOD.* a) The Dynamic LOD system loads different meshes with various level of details based on the proximity to camera. This however results in sudden poping up of the meshes which generates artefacts in the data. b) Default LOD settings used by the engine. c) The modified LOD system is used to maintain LODs at a fixed LOD so that the avatar's appearance is unaffected by distance. d) The LOD of the character is locked to 1 using Forced LOD.

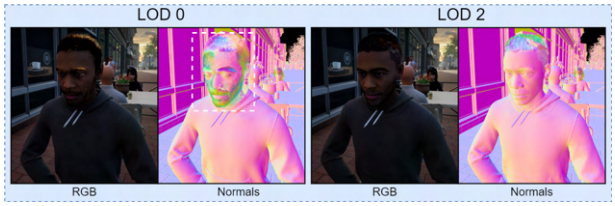


Figure 5. *Artefacts in normal maps for facial hairs.* When the camera is very close to the characters, the engine uses additional detail meshes for characters with facial hair at the highest LOD level (LOD 0). As a result, artefacts appear in the normal maps.We use LOD 1 or 2 for such characters.

Absolute vs Relative positional encoding. We utilised relative positional encoding [6] for self-attention in our proposed UBotNet architecture. We compare it against the absolute positional embeddings and show the quantitative results in Table 2. The absolute positional embeddings perform inferior to the relative positional embeddings used for self-attention. Moreover, the differences are more prominent in case of normal estimation. This is reaffirmed by the visual differences shown in Figure 7. The network loses the context required for learning the consistent representation of the normals. It behaves similar to the UNet_{128} network discussed in the main paper.

Network architecture Table 3 shows the detailed layout of the UBotNet architecture. The three major sections of the architecture are: UNet Encoder, Bottleneck Transformer and UNet Decoder.

3. Addition Results

Figure 9 shows the results for the networks discussed in main paper for depth and normal estimation on 360 images captured from real-world locations. As evident from the results, other methods struggle when estimating the normal information. On the other hand, UBotNet leverages this information and estimates both depth and normal information with increased accuracy. Note that some of networks also struggle with changing sky conditions and hence produce artifacts in those regions. UBotnet is more robust to diverse outdoor lighting and sky conditions. Interestingly, other networks also fail to identify vertical structures (as shown in last image of Figure 9) whereas both UBotNet and UbotNet Lite are able to segment ground from the walls and boundaries.

Figure 10 demonstrates the qualitative results for the architectures that perform only depth estimation. The visual output of depth estimation from the Bifuse [9] support the quantitative results in the main paper where Bifuse performs really well in OmniHorizon benchmark. Figure 11 shows additional examples of depth and normal estimation by UBotNet on real-world images. We test the network in overcast cast conditions, uneven terrain and reflective floors. UBotNet also performs well with diverse vegetation scenarios ranging from small shrubs to complex forests. We also show few results for indoor scenarios where the network performs well even though it was trained for outdoor scenarios. Note that all the networks used for the evaluation and results discussed in this section were trained purely on OmniHorizon dataset.

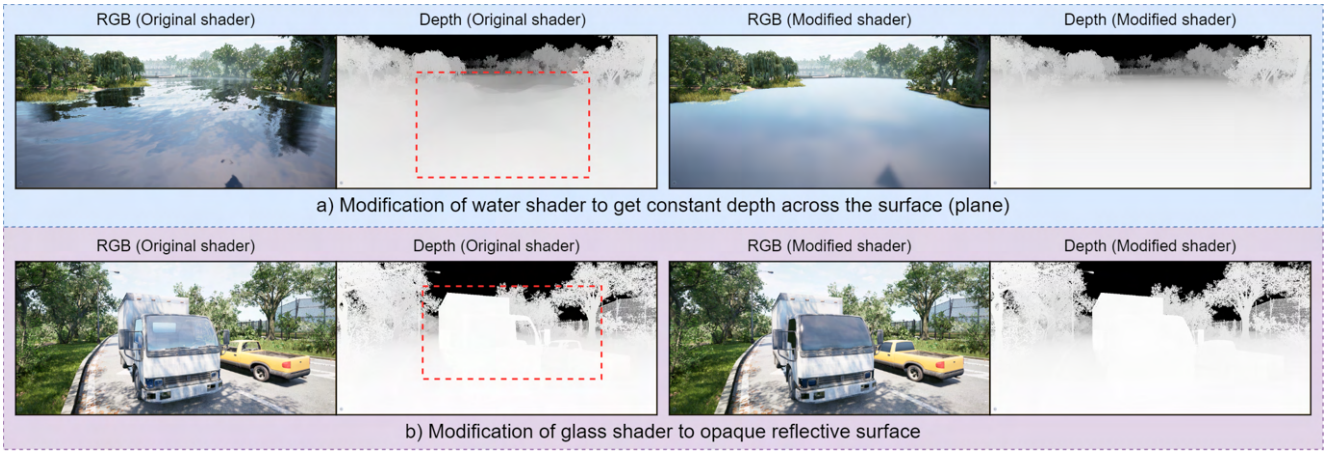


Figure 6. *Assumptions for the dataset.* a) Modification of water shader to achieve constant depth across the surface of the water. b) Modification of glass shader into opaque reflective surface which hides the interior parts of the vehicles.

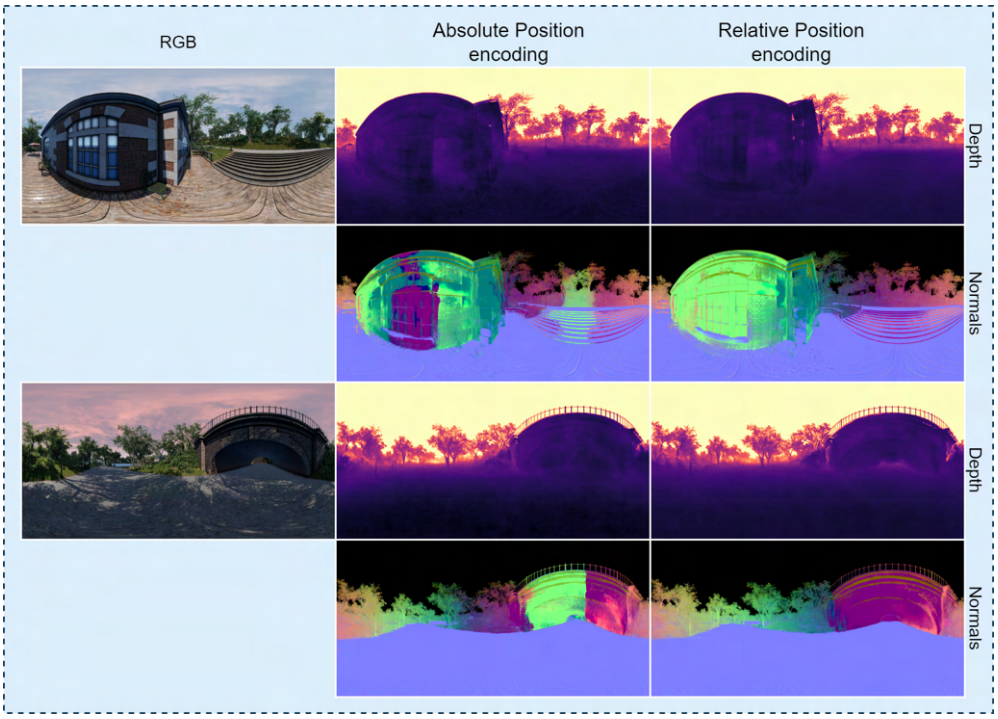


Figure 7. *Comparison between Abs. and Rel. positional embedding.* Absolute positional embedding loses the context required for learning the normals when used for self-attention.

Table 2. *Quantitative results for the comparison between the positional embedding used in the UBotNet architecture for self-attention.* The results for the Relative Positional Embedding are repeated from our main paper for the comparison.

	Depth Error ↓			**Depth Accuracy** ↑			**Normal Error** ↓			**Normal Accuracy** ↑		
Method	RMSE	MRE	RMSE log	$\delta 1 < 1.25$	$\delta 2 < 1.25^2$	$\delta 3 < 1.25^3$	Mean	Median	RMSE	5.0°	7.5°	11.25°
Absolute Pos. Emb.	**0.053**	0.290	0.152	0.691	0.871	0.925	8.65	3.98	13.99	54.26	63.00	73.23
Relative Pos. Emb.	0.054	**0.271**	**0.151**	**0.712**	**0.875**	**0.926**	**7.44**	**3.61**	**12.12**	**56.80**	**67.28**	**78.52**

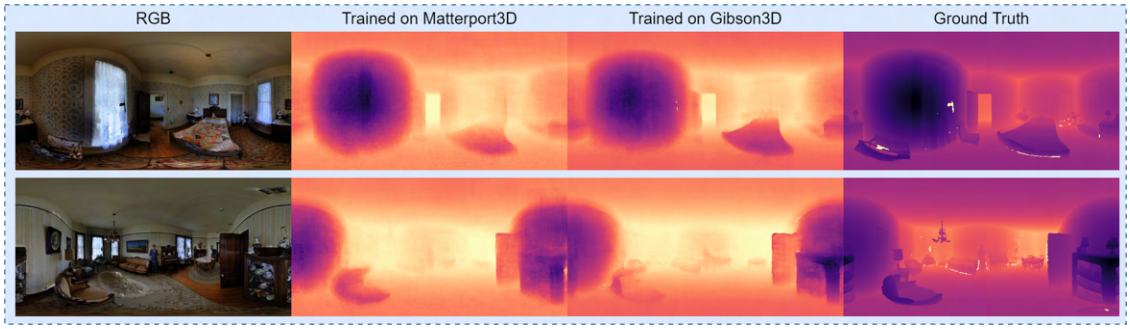


Figure 8. *Qualitative results for UBotNet Lite trained on Indoor datasets - Matterport3D and Gibson3D.*

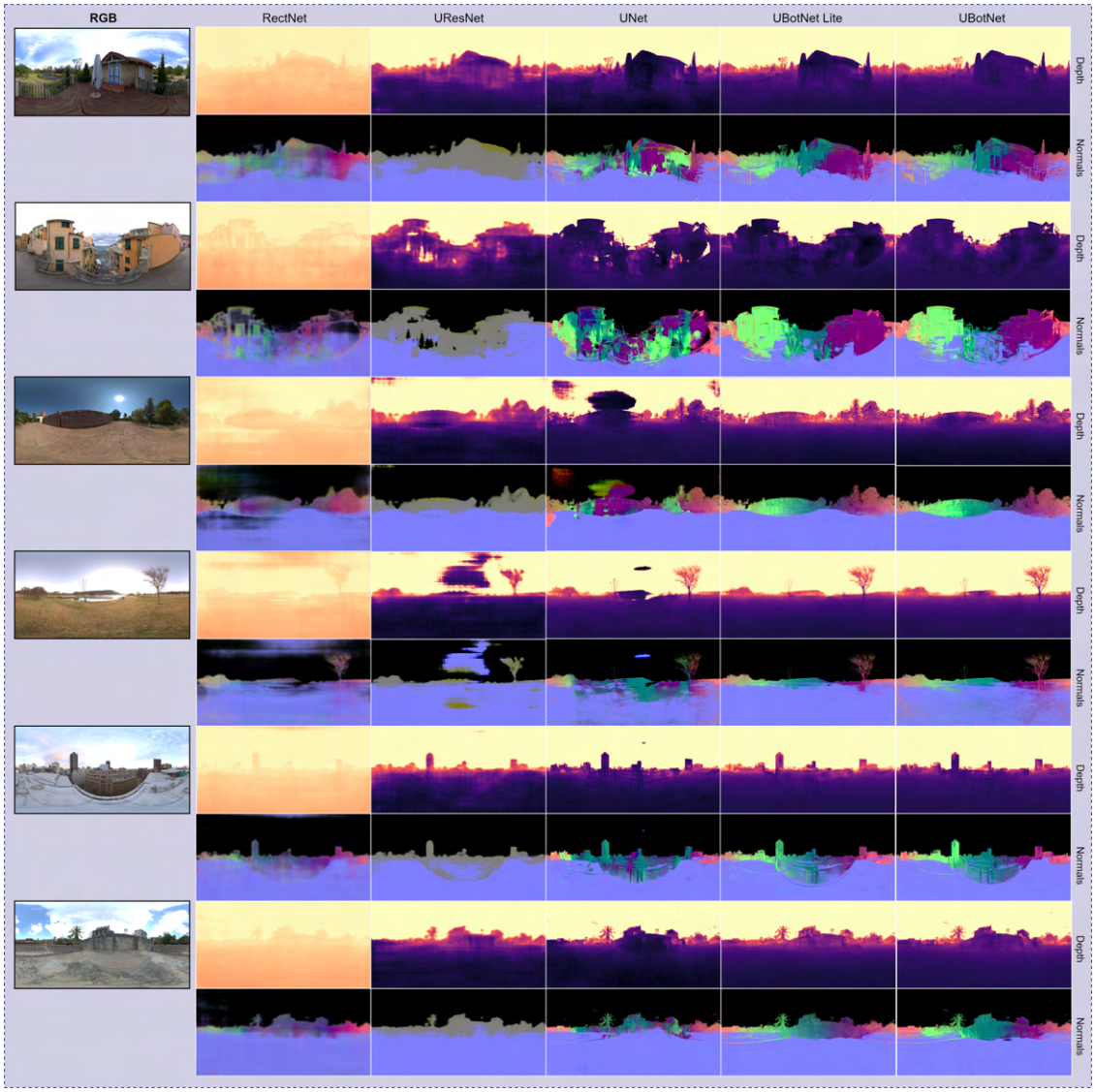


Figure 9. *Depth and Normal estimation on real-world images in the wild.* Comparison between all the networks discussed in main paper for depth and normal estimation on real world images.

Figure 10. *Qualitative results for monocular depth estimation by [7], [3] and [9] on OmniHorizon and real-world images.*

Figure 11. *Examples of depth and normal estimation using UBotNet on real-world images in the wild.*

References

[1] Georgios Albanis, Nikolaos Zioulis, Petros Drakoulis, Vasileios Gkitsas, Vladimiros Sterzentsenko, Federico Alvarez, Dimitrios Zarpalas, and Petros Daras. Pano3d: A holistic benchmark and a solid baseline for 360° depth estimation. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 3722–3732, 2021. 2

[2] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. *International Conference on 3D Vision (3DV)*, 2017. 2

[3] Giovanni Pintore et al. SliceNet: deep dense depth estimation from a single indoor panorama using a slice-based representation. In *CVPR*, 2021. 6

[4] Epic Games. Metahumans, 2022. https://www.unrealengine.com/en-US/metahuman. 1

[5] Oscar Martinez Mozos, Kazuto Nakashima, Hojung Jung, Yumi Iwashita, and Ryo Kurazume. Fukuoka datasets for place categorization. *The International Journal of Robotics Research*, 38(5):507–517, 2019. 1

[6] Aravind Srinivas, Tsung-Yi Lin, Niki Parmar, Jonathon Shlens, Pieter Abbeel, and Ashish Vaswani. Bottleneck transformers for visual recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16519–16529, June 2021. 3

[7] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Hohonet: 360 indoor holistic understanding with latent horizontal features. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2573–2582, 2021. 6

[8] Igor Vasiljevic, Nick Kolkin, Shanyi Zhang, Ruotian Luo, Haochen Wang, Falcon Z. Dai, Andrea F. Daniele, Mohammadreza Mostajabi, Steven Basart, Matthew R. Walter, and Gregory Shakhnarovich. DIODE: A Dense Indoor and Outdoor DEpth Dataset. *CoRR*, abs/1908.00463, 2019. 2

[9] Fu-En Wang, Yu-Hsuan Yeh, Min Sun, Wei-Chen Chiu, and Yi-Hsuan Tsai. Bifuse: Monocular 360 depth estimation via bi-projection fusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 462–471, 2020. 3, 6

[10] Fei Xia, Amir R. Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world perception for embodied agents. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 2

Table 3. *UBotNet Network Architecture.* The network consists of three major segments: UNet Encoder, Bottleneck Transformer and UNet Decoder. Note: To simplify the table we have included the final dense and sigmoid layers in the Decoder towards the end.

UNet Encoder			**Bottleneck Transformer**			**UNet Decoder**		
Layer	**Output Shape**	**Params**	**Layer**	**Output Shape**	**Params**	**Layer**	**Output Shape**	**Params**
Conv2d-1	128, 512, 256	3,456	Identity-64	2048, 32, 16	0	ConvTranspose2d-158	1024, 64, 32	8,389,632
BatchNorm2d-2	128, 512, 256	256	Conv2d-65	512, 32, 16	1,048,576	Conv2d-159	1024, 64, 32	18,874,368
ReLU-3	128, 512, 256	0	BatchNorm2d-66	512, 32, 16	1,024	BatchNorm2d-160	1024, 64, 32	2,048
Conv2d-4	128, 512, 256	147,456	ReLU-67	512, 32, 16	0	ReLU-161	1024, 64, 32	0
BatchNorm2d-5	128, 512, 256	256	ReLU-68	512, 32, 16	0	Conv2d-162	1024, 64, 32	9,437,184
ReLU-6	128, 512, 256	0	ReLU-69	512, 32, 16	0	BatchNorm2d-163	1024, 64, 32	2,048
DoubleConv-7	128, 512, 256	0	ReLU-70	512, 32, 16	0	ReLU-164	1024, 64, 32	0
MaxPool2d-8	128, 511, 255	0	ReLU-71	512, 32, 16	0	DoubleConv-165	1024, 64, 32	0
MaxPool2d-9	128, 511, 255	0	ReLU-72	512, 32, 16	0	UNet-up-block-166	1024, 64, 32	0
ReflectionPad2d-10	128, 514, 258	0	Conv2d-73	1024, 32, 16	524,288	ConvTranspose2d-167	512, 128, 64	2,097,664
ReflectionPad2d-11	128, 514, 258	0	Conv2d-74	512, 32, 16	262,144	Conv2d-168	512, 128, 64	4,718,592
BlurPool-12	128, 256, 128	0	RelPosEmb-75	4, 512, 512	0	BatchNorm2d-169	512, 128, 64	1,024
BlurPool-13	128, 256, 128	0	Softmax-76	4, 512, 512	0	ReLU-170	512, 128, 64	0
Conv2d-14	256, 256, 128	294,912	MHSA-77	512, 32, 16	0	Conv2d-171	512, 128, 64	2,359,296
BatchNorm2d-15	256, 256, 128	512	Identity-78	512, 32, 16	0	BatchNorm2d-172	512, 128, 64	1,024
ReLU-16	256, 256, 128	0	BatchNorm2d-79	512, 32, 16	1,024	ReLU-173	512, 128, 64	0
Conv2d-17	256, 256, 128	589,824	ReLU-80	512, 32, 16	0	DoubleConv-174	512, 128, 64	0
BatchNorm2d-18	256, 256, 128	512	ReLU-81	512, 32, 16	0	UNet-up-block-175	512, 128, 64	0
ReLU-19	256, 256, 128	0	ReLU-82	512, 32, 16	0	ConvTranspose2d-176	256, 256, 128	524,544
DoubleConv-20	256, 256, 128	0	ReLU-83	512, 32, 16	0	Conv2d-177	256, 256, 128	1,179,648
UNet-down-block-21	256, 256, 128	0	ReLU-84	512, 32, 16	0	BatchNorm2d-178	256, 256, 128	512
MaxPool2d-22	256, 255, 127	0	ReLU-85	512, 32, 16	0	ReLU-179	256, 256, 128	0
MaxPool2d-23	256, 255, 127	0	Conv2d-86	2048, 32, 16	1,048,576	Conv2d-180	256, 256, 128	589,824
ReflectionPad2d-24	256, 258, 130	0	BatchNorm2d-87	2048, 32, 16	4,096	BatchNorm2d-181	256, 256, 128	512
ReflectionPad2d-25	256, 258, 130	0	ReLU-88	2048, 32, 16	0	ReLU-182	256, 256, 128	0
BlurPool-26	256, 128, 64	0	ReLU-89	2048, 32, 16	0	DoubleConv-183	256, 256, 128	0
BlurPool-27	256, 128, 64	0	ReLU-90	2048, 32, 16	0	UNet-up-block-184	256, 256, 128	0
Conv2d-28	512, 128, 64	1,179,648	ReLU-91	2048, 32, 16	0	ConvTranspose2d-185	128, 512, 256	131,200
BatchNorm2d-29	512, 128, 64	1,024	ReLU-92	2048, 32, 16	0	Conv2d-186	128, 512, 256	294,912
ReLU-30	512, 128, 64	0	ReLU-93	2048, 32, 16	0	BatchNorm2d-187	128, 512, 256	256
Conv2d-31	512, 128, 64	2,359,296	BoTBlock-94	2048, 32, 16	0	ReLU-188	128, 512, 256	0
BatchNorm2d-32	512, 128, 64	1,024	Identity-95	2048, 32, 16	0	Conv2d-189	128, 512, 256	147,456
ReLU-33	512, 128, 64	0	Conv2d-96	512, 32, 16	1,048,576	BatchNorm2d-190	128, 512, 256	256
DoubleConv-34	512, 128, 64	0	BatchNorm2d-97	512, 32, 16	1,024	ReLU-191	128, 512, 256	0
UNet-down-block-35	512, 128, 64	0	ReLU-98	512, 32, 16	0	DoubleConv-192	128, 512, 256	0
MaxPool2d-36	512, 127, 63	0	ReLU-99	512, 32, 16	0	UNet-up-block-193	128, 512, 256	0
MaxPool2d-37	512, 127, 63	0	ReLU-100	512, 32, 16	0	Linear-194	512, 256, 512	66,048
ReflectionPad2d-38	512, 130, 66	0	ReLU-101	512, 32, 16	0	ReLU-195	512, 256, 512	0
ReflectionPad2d-39	512, 130, 66	0	ReLU-102	512, 32, 16	0	Dropout-196	512, 256, 512	0
BlurPool-40	512, 64, 32	0	ReLU-103	512, 32, 16	0	Linear-197	512, 256, 128	65,664
BlurPool-41	512, 64, 32	0	Conv2d-104	1024, 32, 16	524,288	ReLU-198	512, 256, 128	0
Conv2d-42	1024, 64, 32	4,718,592	Conv2d-105	512, 32, 16	262,144	Dropout-199	512, 256, 128	0
BatchNorm2d-43	1024, 64, 32	2,048	RelPosEmb-106	4, 512, 512	0	Linear-200	512, 256, 1	129
ReLU-44	1024, 64, 32	0	Softmax-107	4, 512, 512	0	Sigmoid-201	512, 256, 1	0
Conv2d-45	1024, 64, 32	9,437,184	MHSA-108	512, 32, 16	0	Linear-202	512, 256, 3	387
BatchNorm2d-46	1024, 64, 32	2,048	Identity-109	512, 32, 16	0	Sigmoid-203	512, 256, 3	0
ReLU-47	1024, 64, 32	0	BatchNorm2d-110	512, 32, 16	1,024			
DoubleConv-48	1024, 64, 32	0	ReLU-111	512, 32, 16	0			
UNet-down-block-49	1024, 64, 32	0	ReLU-112	512, 32, 16	0			
MaxPool2d-50	1024, 63, 31	0	ReLU-113	512, 32, 16	0			
MaxPool2d-51	1024, 63, 31	0	ReLU-114	512, 32, 16	0			
ReflectionPad2d-52	1024, 66, 34	0	ReLU-115	512, 32, 16	0			
ReflectionPad2d-53	1024, 66, 34	0	ReLU-116	512, 32, 16	0			
BlurPool-54	1024, 32, 16	0	Conv2d-117	2048, 32, 16	1,048,576			
BlurPool-55	1024, 32, 16	0	BatchNorm2d-118	2048, 32, 16	4,096			
Conv2d-56	2048, 32, 16	18,874,368	ReLU-119	2048, 32, 16	0			
BatchNorm2d-57	2048, 32, 16	4,096	ReLU-120	2048, 32, 16	0			
ReLU-58	2048, 32, 16	0	ReLU-121	2048, 32, 16	0			
Conv2d-59	2048, 32, 16	37,748,736	ReLU-122	2048, 32, 16	0			
BatchNorm2d-60	2048, 32, 16	4,096	ReLU-123	2048, 32, 16	0			
ReLU-61	2048, 32, 16	0	ReLU-124	2048, 32, 16	0			
DoubleConv-62	2048, 32, 16	0	BoTBlock-125	2048, 32, 16	0			
UNet-down-block-63	2048, 32, 16	0	Identity-126	2048, 32, 16	0			
			Conv2d-127	512, 32, 16	1,048,576			
			BatchNorm2d-128	512, 32, 16	1,024			
			ReLU-129	512, 32, 16	0			
			ReLU-130	512, 32, 16	0			
			ReLU-131	512, 32, 16	0			
			ReLU-132	512, 32, 16	0			
			ReLU-133	512, 32, 16	0			
			ReLU-134	512, 32, 16	0			
			Conv2d-135	1024, 32, 16	524,288			
			Conv2d-136	512, 32, 16	262,144			
			RelPosEmb-137	4, 512, 512	0			
			Softmax-138	4, 512, 512	0			
			MHSA-139	512, 32, 16	0			
			Identity-140	512, 32, 16	0			
			BatchNorm2d-141	512, 32, 16	1,024			
			ReLU-142	512, 32, 16	0			
			ReLU-143	512, 32, 16	0			
			ReLU-144	512, 32, 16	0			
			ReLU-145	512, 32, 16	0			
			ReLU-146	512, 32, 16	0			
			ReLU-147	512, 32, 16	0			
			Conv2d-148	2048, 32, 16	1,048,576			
			BatchNorm2d-149	2048, 32, 16	4,096			
			ReLU-150	2048, 32, 16	0			
			ReLU-151	2048, 32, 16	0			
			ReLU-152	2048, 32, 16	0			
			ReLU-153	2048, 32, 16	0			
			ReLU-154	2048, 32, 16	0			
			ReLU-155	2048, 32, 16	0			
			BoTBlock-156	2048, 32, 16	0			
			BoTStack-157	2048, 32, 16	0			