Supplemental Material : Revisiting Near/Remote Sensing with Geospatial Attention

Scott Workman, DZYNE Technologies
M. Usman Rafique, Kitware, Inc.
Hunter Blanton, University of Kentucky
Nathan Jacobs, University of Kentucky

1. Dataset Details

We extend the Brooklyn and Queens dataset [2] with two new per-pixel labeling tasks, estimating land cover and estimating height. The original dataset contains non-overlapping overhead images downloaded from Bing Maps (zoom level 19, approximately 30 cm per pixel) and street-level panoramas from Google Street View. The Brooklyn subset consists of 43,605 overhead images and 139,327 panoramas. The held-out Queens subset, used solely for evaluation, consists of 10,044 overhead images and 38,603 panoramas. Including our two new tasks, there are five tasks for this dataset: estimating land use, building age, building function, land cover, and height. For all experiments, we include the 20 closest street-level panoramas to each overhead image. For evaluation, we use the original train/test splits.

2. Qualitative Results

We show qualitative results for building function estimation in Figure 1. Due the the large number of classes (206 building types), we visualize results for this task as a top-k image where each pixel is assigned a color (from green to red) by the rank of the correct class in the posterior distribution. Bright green corresponds to rank one and red corresponds rank 10 or more. We show additional qualitative results for the other tasks in Figure 2.

3. Attention Visualization

Figure 3 visualizes the spatial attention maps for several input images as the target location changes. For this experiment, we use our full method and output from the height estimation task. Each image is color-coded and the capture location is represented by the same-colored dot in the overhead image. Similarly, the attention maps are color-coded, with the target location represented by the same-colored square in the overhead image. As observed, the region of high attention is generally oriented toward the target pixel. Our approach is able to learn these geometric relationships without requiring direct correspondences.

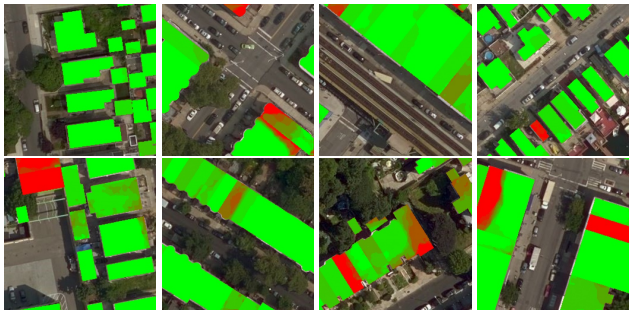

Figure 1. Qualitative results for building function. Each pixel represents the rank of the correct class in the posterior distribution (green to red). Bright green corresponds to rank one and red corresponds to rank 10 or more.

Similarly, Figure 4 visualizes the spatial attention maps for several pairs of input images and target locations, for three different tasks. For each overhead image, the top row of attention maps corresponds to the $\square$ in the overhead image, and the bottom row corresponds to the $\times$. As expected, the region of high attention is generally oriented toward the target pixel and the attention maps are task dependent. These results demonstrate that our approach is able to learn rich geometric relationships without explicitly providing such supervision and without requiring direct correspondences or other strong geometric assumptions, such as single-image depth estimation.

4. Extended Evaluation on Queens

Following the standard protocol, all our models are trained exclusively on the training subset of the Brooklyn portion of the "Brooklyn and Queens" dataset [2] (aside from pre-training). In the main paper, we presented results on the held-out testing subset of the Brooklyn portion of the dataset. Here we extend this analysis to show how the model generalizes to the Queens portion. This benchmark is known to be challenging due to large differences in the underlying label distributions and building appearance between the two portions.

Table 1 shows the results of our approach versus baselines on Queens. Our approach, which integrates geospatial attention, generally matches or outperforms two prior methods as well as the single-modality baselines. While there is clearly work left to be done to improve domain adaptation, this result demonstrates that our model is not just over-fitting to the Brooklyn region.

Table 2 extends the ablation study from the main paper, which highlights the importance of the different input features used for geospatial attention, to the remaining tasks (building age, building function, land cover and height). As before, our full approach outperforms baselines, with the geometric features being essential for achieving good performance.

5. Detailed Architecture

We provide detailed architecture descriptions for the components of our network. Table 3 and Table 4 show the feature encoders used for the overhead (EfficientNet-B4) and ground-level (ResNet-50) imagery, respectively. Table 5 shows the architecture for forming the dense ground-level feature map using geospatial attention. Table 6 corresponds to the fusion network for combining the overhead feature with the dense ground-level feature map. Finally, Table 7 shows our U-Net style decoder used for generating the segmentation output.

6. Computational Analysis

While our method offers significantly improved metrics over the overhead-image only method, it comes at an increase in computational cost. This difference is especially pronounced during training, where a single training run for our full method takes around 67 hours but the overhead-only baseline (*remote*) only required around 8 hours. The ground-only baseline (*proximate*) required around 54 hours to train. We conclude that the primary computational increase is due to the inclusion of the ground-level images. However, we did not extensively optimize for training time computational efficiency. While training time is important, inference time is often a much more important factor in remote sensing applications. We found in our unoptimized implementation that our method requires $\sim$0.09 seconds for a single overhead image (and the corresponding ground-level images). This compares to $\sim$0.03 seconds for the overhead-only baseline.

References

[1] Rui Cao, Jiasong Zhu, Wei Tu, Qingquan Li, Jinzhou Cao, Bozhi Liu, Qian Zhang, and Guoping Qiu. Integrating aerial and street view images for urban land use classification. *Remote Sensing*, 10(10):1553, 2018. 4

[2] Scott Workman, Menghua Zhai, David J. Crandall, and Nathan Jacobs. A Unified Model for Near and Remote Sensing. In *IEEE International Conference on Computer Vision*, 2017. 1, 4

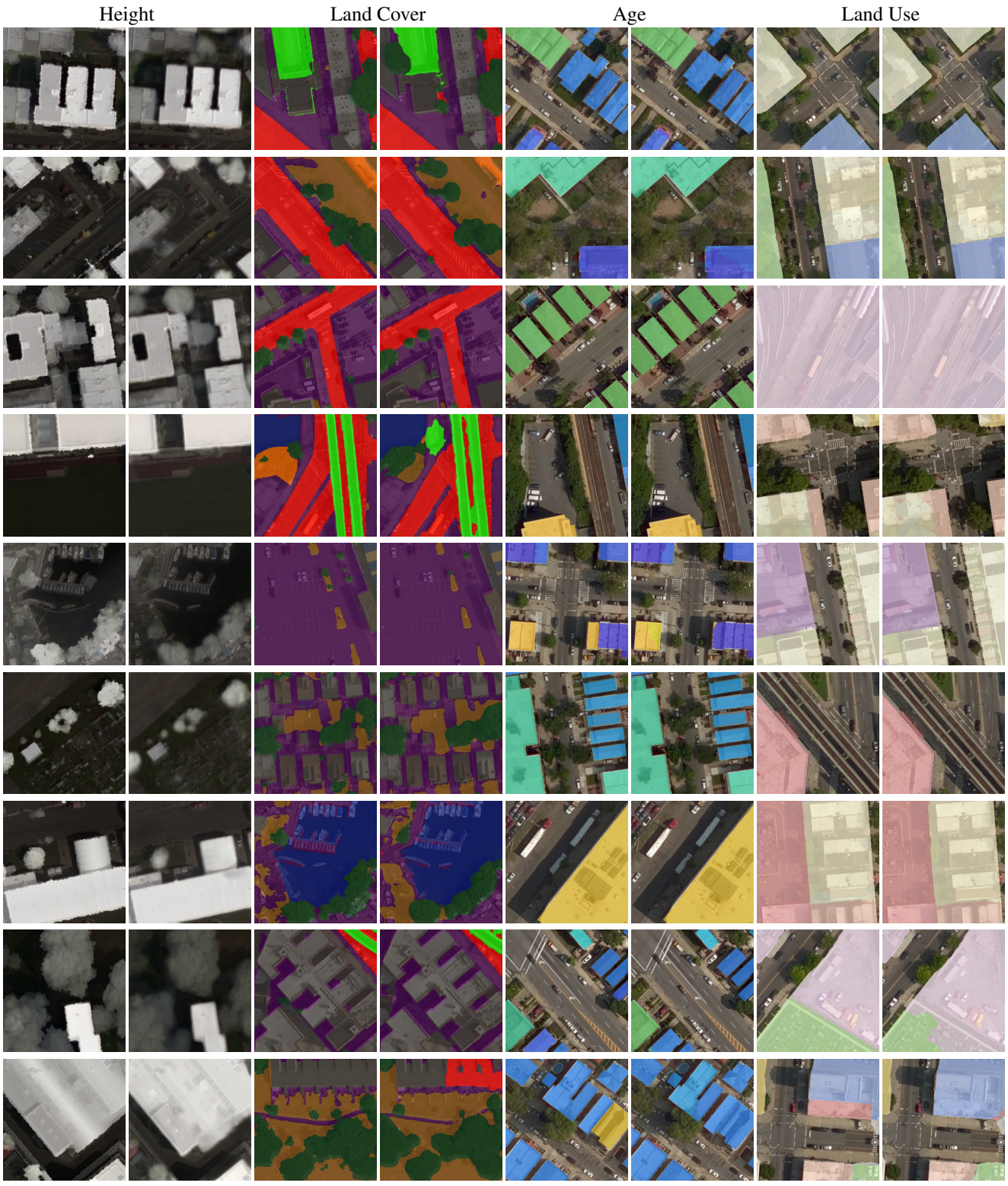

Figure 2. Additional qualitative results: (left) ground truth and (right) *ours*.

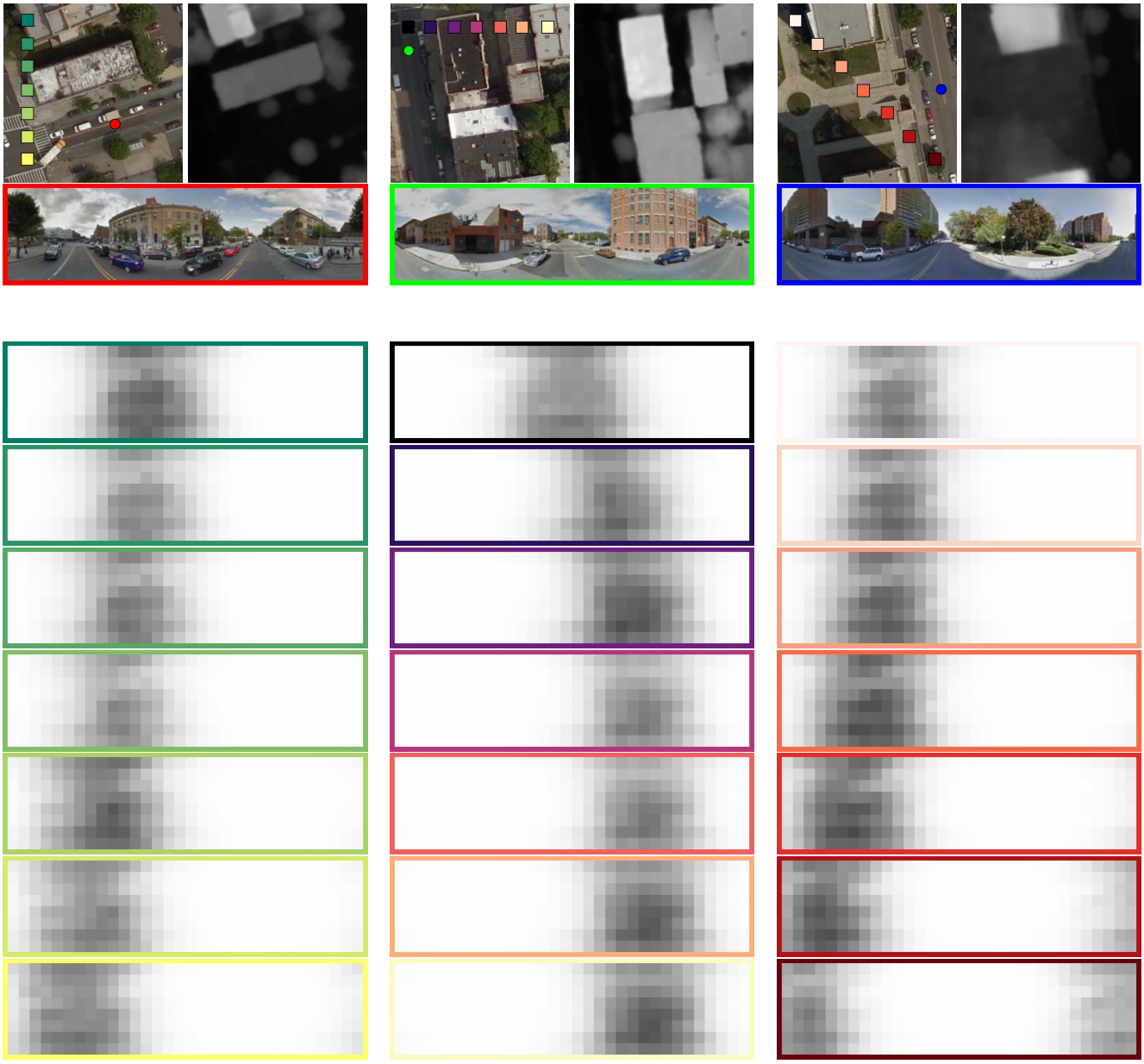

Figure 3. Visualizing spatial attention maps from our full method as the target location changes (height prediction task). Each column shows attention maps for one panorama, with the location of the panorama represented by the same-colored dot in the overhead image. Similarly, the attention maps are color-coded corresponding to the target location, which is represented by the same-colored square in the overhead image.

Table 1. Queens evaluation results.

	Land Use		Age		Function		Land Cover		Height	
	mIOU	Acc	mIOU	Acc	mIOU	Acc	mIOU	Acc	RMSE	RMSE log
Workman et al. [2]	33.48%	70.55%	9.53%	29.76%	3.73%	**34.13%**				
Cao et al. [1]	39.40%	74.87%								
proximate	33.84%	68.88%	10.44%	30.13%	3.63%	33.27%	30.02%	59.97%	4.597	1.236
remote	34.16%	72.30%	8.31%	22.91%	2.85%	29.46%	**62.63%**	83.54%	3.319	0.988
ours	**42.93%**	**76.85%**	**12.88%**	**32.93%**	**4.08%**	34.04%	61.24%	**83.82%**	**3.003**	**0.946**

Figure 4. Spatial attention maps for several ground-level images and target locations using our full method. The location of each panorama is represented by the same-colored dot in the overhead image. For each panorama, the top row of attentions maps corresponds to using the orange □ in the overhead image as the target location, while the bottom row corresponds to using the purple × as the target location. From top to bottom, the tasks correspond to height estimation, land cover segmentation, and building age prediction.

Table 2. Ablation study highlighting the importance of different input features for geospatial attention.

			Age		Function		Land Cover		Height	
Panorama	Overhead	Geometry	mIOU	Acc	mIOU	Acc	mIOU	Acc	RMSE	RMSE log
✓			33.52%	54.47%	13.60%	46.53%	72.95%	87.49%	3.128	0.766
	✓		32.49%	53.15%	14.11%	46.44%	73.41%	87.57%	3.135	0.781
		d	39.28%	60.02%	17.58%	51.40%	73.83%	87.56%	3.001	0.755
		θ	37.40%	58.57%	17.74%	50.46%	72.90%	87.61%	3.041	0.762
		d, θ	51.07%	70.04%	24.21%	59.07%	72.61%	87.93%	2.878	0.747
✓	✓	d, θ	**51.70%**	**70.34%**	**27.40%**	**60.31%**	**74.59%**	**88.10%**	**2.845**	**0.747**

Table 3. Overhead image feature extractor.

Layer (type:depth-idx)	Input Shape	Kernel Shape	Output Shape	Param #
EfficientNet: 1	–	–	–	–
— ModuleList: 2-1	–	–	–	–
— Conv2dStaticSamePadding: 2-2	[1, 3, 256, 256]	[3, 48, 3, 3]	[1, 48, 128, 128]	–
— — ZeroPad2d: 3-1	[1, 3, 256, 256]	–	[1, 3, 257, 257]	–
— BatchNorm2d: 2-3	[1, 48, 128, 128]	[48]	[1, 48, 128, 128]	96
— MemoryEfficientSwish: 2-4	[1, 48, 128, 128]	–	[1, 48, 128, 128]	–
— ModuleList: 2-1	–	–	–	–
— — MBConvBlock: 3-2	[1, 48, 128, 128]	–	[1, 24, 128, 128]	2,940
— — MBConvBlock: 3-3	[1, 24, 128, 128]	–	[1, 24, 128, 128]	1,206
— — MBConvBlock: 3-4	[1, 24, 128, 128]	–	[1, 32, 64, 64]	11,878
— — MBConvBlock: 3-5	[1, 32, 64, 64]	–	[1, 32, 64, 64]	18,120
— — MBConvBlock: 3-6	[1, 32, 64, 64]	–	[1, 32, 64, 64]	18,120
— — MBConvBlock: 3-7	[1, 32, 64, 64]	–	[1, 32, 64, 64]	18,120
— — MBConvBlock: 3-8	[1, 32, 64, 64]	–	[1, 56, 32, 32]	25,848
— — MBConvBlock: 3-9	[1, 56, 32, 32]	–	[1, 56, 32, 32]	57,246
— — MBConvBlock: 3-10	[1, 56, 32, 32]	–	[1, 56, 32, 32]	57,246
— — MBConvBlock: 3-11	[1, 56, 32, 32]	–	[1, 56, 32, 32]	57,246
— — MBConvBlock: 3-12	[1, 56, 32, 32]	–	[1, 112, 16, 16]	70,798
— — MBConvBlock: 3-13	[1, 112, 16, 16]	–	[1, 112, 16, 16]	197,820
— — MBConvBlock: 3-14	[1, 112, 16, 16]	–	[1, 112, 16, 16]	197,820
— — MBConvBlock: 3-15	[1, 112, 16, 16]	–	[1, 112, 16, 16]	197,820
— — MBConvBlock: 3-16	[1, 112, 16, 16]	–	[1, 112, 16, 16]	197,820
— — MBConvBlock: 3-17	[1, 112, 16, 16]	–	[1, 112, 16, 16]	197,820
— — MBConvBlock: 3-18	[1, 112, 16, 16]	–	[1, 160, 16, 16]	240,924
— — MBConvBlock: 3-19	[1, 160, 16, 16]	–	[1, 160, 16, 16]	413,160
— — MBConvBlock: 3-20	[1, 160, 16, 16]	–	[1, 160, 16, 16]	413,160
— — MBConvBlock: 3-21	[1, 160, 16, 16]	–	[1, 160, 16, 16]	413,160
— — MBConvBlock: 3-22	[1, 160, 16, 16]	–	[1, 160, 16, 16]	413,160
— — MBConvBlock: 3-23	[1, 160, 16, 16]	–	[1, 160, 16, 16]	413,160
— — MBConvBlock: 3-24	[1, 160, 16, 16]	–	[1, 272, 8, 8]	520,904
— — MBConvBlock: 3-25	[1, 272, 8, 8]	–	[1, 272, 8, 8]	1,159,332
— — MBConvBlock: 3-26	[1, 272, 8, 8]	–	[1, 272, 8, 8]	1,159,332
— — MBConvBlock: 3-27	[1, 272, 8, 8]	–	[1, 272, 8, 8]	1,159,332
— — MBConvBlock: 3-28	[1, 272, 8, 8]	–	[1, 272, 8, 8]	1,159,332
— — MBConvBlock: 3-29	[1, 272, 8, 8]	–	[1, 272, 8, 8]	1,159,332
— — MBConvBlock: 3-30	[1, 272, 8, 8]	–	[1, 272, 8, 8]	1,159,332
— — MBConvBlock: 3-31	[1, 272, 8, 8]	–	[1, 272, 8, 8]	1,159,332
— — MBConvBlock: 3-32	[1, 272, 8, 8]	–	[1, 448, 8, 8]	1,420,804
— — MBConvBlock: 3-33	[1, 448, 8, 8]	–	[1, 448, 8, 8]	3,049,200
— Conv2dStaticSamePadding: 2-5	[1, 448, 8, 8]	[448, 1792, 1, 1]	[1, 1792, 8, 8]	–
— — Identity: 3-34	[1, 448, 8, 8]	–	[1, 448, 8, 8]	–
— BatchNorm2d: 2-6	[1, 1792, 8, 8]	[1792]	[1, 1792, 8, 8]	3,584
— MemoryEfficientSwish: 2-7	[1, 1792, 8, 8]	–	[1, 1792, 8, 8]	–

Table 4. Ground-level image feature extractor.

Layer (type:depth-idx)	Input Shape	Kernel Shape	Output Shape	Param #
Sequential: 1-1	[20, 3, 128, 500]	–	[20, 1024, 8, 32]	–
— Conv2d: 2-1	[20, 3, 128, 500]	[3, 64, 7, 7]	[20, 64, 64, 250]	(9,408)
— BatchNorm2d: 2-2	[20, 64, 64, 250]	[64]	[20, 64, 64, 250]	(128)
— ReLU: 2-3	[20, 64, 64, 250]	–	[20, 64, 64, 250]	–
— MaxPool2d: 2-4	[20, 64, 64, 250]	–	[20, 64, 32, 125]	–
— Sequential: 2-5	[20, 64, 32, 125]	–	[20, 256, 32, 125]	–
— — Bottleneck: 3-1	[20, 64, 32, 125]	–	[20, 256, 32, 125]	(75,008)
— — Bottleneck: 3-2	[20, 256, 32, 125]	–	[20, 256, 32, 125]	(70,400)
— — Bottleneck: 3-3	[20, 256, 32, 125]	–	[20, 256, 32, 125]	(70,400)
— Sequential: 2-6	[20, 256, 32, 125]	–	[20, 512, 16, 63]	–
— — Bottleneck: 3-4	[20, 256, 32, 125]	–	[20, 512, 16, 63]	(379,392)
— — Bottleneck: 3-5	[20, 512, 16, 63]	–	[20, 512, 16, 63]	(280,064)
— — Bottleneck: 3-6	[20, 512, 16, 63]	–	[20, 512, 16, 63]	(280,064)
— — Bottleneck: 3-7	[20, 512, 16, 63]	–	[20, 512, 16, 63]	(280,064)
— Sequential: 2-7	[20, 512, 16, 63]	–	[20, 1024, 8, 32]	–
— — Bottleneck: 3-8	[20, 512, 16, 63]	–	[20, 1024, 8, 32]	1,512,448
— — Bottleneck: 3-9	[20, 1024, 8, 32]	–	[20, 1024, 8, 32]	1,117,184
— — Bottleneck: 3-10	[20, 1024, 8, 32]	–	[20, 1024, 8, 32]	1,117,184
— — Bottleneck: 3-11	[20, 1024, 8, 32]	–	[20, 1024, 8, 32]	1,117,184
— — Bottleneck: 3-12	[20, 1024, 8, 32]	–	[20, 1024, 8, 32]	1,117,184
— — Bottleneck: 3-13	[20, 1024, 8, 32]	–	[20, 1024, 8, 32]	1,117,184
Conv2d: 1-2	[20, 1024, 8, 32]	[1024, 128, 1, 1]	[20, 128, 8, 32]	131,200
LayerNorm: 1-3	[20, 128, 8, 32]	[8, 128, 32]	[20, 128, 8, 32]	65,536

Table 5. Grid architecture.

Layer (type:depth-idx)	Input Shape	Kernel Shape	Output Shape	Param #
Grid: 1-1	–	–	[1, 128, 32, 32]	–
— GeoAttention: 2-1	–	–	[1024, 20, 8, 32]	–
— — Conv2d: 3-1	[20480, 8, 8, 32]	[8, 1, 3, 3]	[20480, 1, 8, 32]	73
— — Conv2d: 3-2	[20480, 8, 8, 32]	[8, 1, 5, 5]	[20480, 1, 8, 32]	201
— — Conv2d: 3-3	[20480, 2, 8, 32]	[2, 1, 1, 1]	[20480, 1, 8, 32]	3
— —Sigmoid: 3-4	[1024, 20, 8, 32]	–	[1024, 20, 8, 32]	–
BatchNorm2d: 1-2	[1, 128, 32, 32]	[128]	[1, 128, 32, 32]	256

Table 6. Fusion (dense ground-level/overhead feature map) architecture.

Layer (type:depth-idx)	Input Shape	Kernel Shape	Output Shape	Param #
Conv2d: 1-1	[1, 184, 32, 32]	[184, 160, 3, 3]	[1, 160, 32, 32]	265,120
BatchNorm2d: 1-2	[1, 160, 32, 32]	[160]	[1, 160, 32, 32]	320
ReLU: 1-3	[1, 160, 32, 32]	–	[1, 160, 32, 32]	–
Conv2d: 1-4	[1, 160, 32, 32]	[160, 160, 3, 3]	[1, 160, 32, 32]	230,560
BatchNorm2d: 1-5	[1, 160, 32, 32]	[160]	[1, 160, 32, 32]	320
ReLU: 1-6	[1, 160, 32, 32]	–	[1, 160, 32, 32]	–
Conv2d: 1-7	[1, 160, 32, 32]	[160, 160, 3, 3]	[1, 160, 32, 32]	230,560
BatchNorm2d: 1-8	[1, 160, 32, 32]	[160]	[1, 160, 32, 32]	320
ReLU: 1-9	[1, 160, 32, 32]	–	[1, 160, 32, 32]	–
MaxPool2d: 1-10	[1, 160, 32, 32]	–	[1, 160, 16, 16]	–
Conv2d: 1-11	[1, 160, 16, 16]	[160, 448, 3, 3]	[1, 448, 16, 16]	645,568
BatchNorm2d: 1-12	[1, 448, 16, 16]	[448]	[1, 448, 16, 16]	896
ReLU: 1-13	[1, 448, 16, 16]	–	[1, 448, 16, 16]	–
Conv2d: 1-14	[1, 448, 16, 16]	[448, 448, 3, 3]	[1, 448, 16, 16]	1,806,784
BatchNorm2d: 1-15	[1, 448, 16, 16]	[448]	[1, 448, 16, 16]	896
ReLU: 1-16	[1, 448, 16, 16]	–	[1, 448, 16, 16]	–
Conv2d: 1-17	[1, 448, 16, 16]	[448, 448, 3, 3]	[1, 448, 16, 16]	1,806,784
BatchNorm2d: 1-18	[1, 448, 16, 16]	[448]	[1, 448, 16, 16]	896
ReLU: 1-19	[1, 448, 16, 16]	–	[1, 448, 16, 16]	–
MaxPool2d: 1-20	[1, 448, 16, 16]	–	[1, 448, 8, 8]	–

Table 7. Decoder architecture.

Layer (type:depth-idx)	Input Shape	Kernel Shape	Output Shape	Param #
Upsample: 1-1	[1, 448, 8, 8]	–	[1, 448, 16, 16]	–
DoubleConv: 1-2	[1, 448, 16, 16]	–	[1, 448, 16, 16]	–
— Sequential: 2-1	[1, 448, 16, 16]	–	[1, 448, 16, 16]	–
— — Conv2d: 3-1	[1, 448, 16, 16]	[448, 448, 3, 3]	[1, 448, 16, 16]	1,806,784
— — BatchNorm2d: 3-2	[1, 448, 16, 16]	[448]	[1, 448, 16, 16]	896
— — ReLU: 3-3	[1, 448, 16, 16]	–	[1, 448, 16, 16]	–
— — Conv2d: 3-4	[1, 448, 16, 16]	[448, 448, 3, 3]	[1, 448, 16, 16]	1,806,784
— — BatchNorm2d: 3-5	[1, 448, 16, 16]	[448]	[1, 448, 16, 16]	896
— — ReLU: 3-6	[1, 448, 16, 16]	–	[1, 448, 16, 16]	–
Upsample: 1-3	[1, 608, 16, 16]	–	[1, 608, 32, 32]	–
DoubleConv: 1-4	[1, 608, 32, 32]	–	[1, 160, 32, 32]	–
— Sequential: 2-2	[1, 608, 32, 32]	–	[1, 160, 32, 32]	–
— — Conv2d: 3-7	[1, 608, 32, 32]	[608, 160, 3, 3]	[1, 160, 32, 32]	875,680
— — BatchNorm2d: 3-8	[1, 160, 32, 32]	[160]	[1, 160, 32, 32]	320
— — ReLU: 3-9	[1, 160, 32, 32]	–	[1, 160, 32, 32]	–
— — Conv2d: 3-10	[1, 160, 32, 32]	[160, 160, 3, 3]	[1, 160, 32, 32]	230,560
— — BatchNorm2d: 3-11	[1, 160, 32, 32]	[160]	[1, 160, 32, 32]	320
— — ReLU: 3-12	[1, 160, 32, 32]	–	[1, 160, 32, 32]	–
Upsample: 1-5	[1, 216, 32, 32]	–	[1, 216, 64, 64]	–
DoubleConv: 1-6	[1, 216, 64, 64]	–	[1, 56, 64, 64]	–
— Sequential: 2-3	[1, 216, 64, 64]	–	[1, 56, 64, 64]	–
— — Conv2d: 3-13	[1, 216, 64, 64]	[216, 56, 3, 3]	[1, 56, 64, 64]	108,920
— — BatchNorm2d: 3-14	[1, 56, 64, 64]	[56]	[1, 56, 64, 64]	112
— — ReLU: 3-15	[1, 56, 64, 64]	–	[1, 56, 64, 64]	–
— — Conv2d: 3-16	[1, 56, 64, 64]	[56, 56, 3, 3]	[1, 56, 64, 64]	28,280
— — BatchNorm2d: 3-17	[1, 56, 64, 64]	[56]	[1, 56, 64, 64]	112
— — ReLU: 3-18	[1, 56, 64, 64]	–	[1, 56, 64, 64]	–
Upsample: 1-7	[1, 88, 64, 64]	–	[1, 88, 128, 128]	–
DoubleConv: 1-8	[1, 88, 128, 128]	–	[1, 88, 128, 128]	–
— Sequential: 2-4	[1, 88, 128, 128]	–	[1, 88, 128, 128]	–
— — Conv2d: 3-19	[1, 88, 128, 128]	[88, 88, 3, 3]	[1, 88, 128, 128]	69,784
— — BatchNorm2d: 3-20	[1, 88, 128, 128]	[88]	[1, 88, 128, 128]	176
— — ReLU: 3-21	[1, 88, 128, 128]	–	[1, 88, 128, 128]	–
— — Conv2d: 3-22	[1, 88, 128, 128]	[88, 88, 3, 3]	[1, 88, 128, 128]	69,784
— — BatchNorm2d: 3-23	[1, 88, 128, 128]	[88]	[1, 88, 128, 128]	176
— — ReLU: 3-24	[1, 88, 128, 128]	–	[1, 88, 128, 128]	–
Upsample: 1-9	[1, 88, 128, 128]	–	[1, 88, 256, 256]	–
DoubleConv: 1-10	[1, 88, 256, 256]	–	[1, 88, 256, 256]	–
— Sequential: 2-5	[1, 88, 256, 256]	–	[1, 88, 256, 256]	–
— — Conv2d: 3-25	[1, 88, 256, 256]	[88, 88, 3, 3]	[1, 88, 256, 256]	69,784
— — BatchNorm2d: 3-26	[1, 88, 256, 256]	[88]	[1, 88, 256, 256]	176
— — ReLU: 3-27	[1, 88, 256, 256]	–	[1, 88, 256, 256]	–
— — Conv2d: 3-28	[1, 88, 256, 256]	[88, 88, 3, 3]	[1, 88, 256, 256]	69,784
— — BatchNorm2d: 3-29	[1, 88, 256, 256]	[88]	[1, 88, 256, 256]	176
— — ReLU: 3-30	[1, 88, 256, 256]	–	[1, 88, 256, 256]	–
Conv2d: 1-11	[1, 88, 256, 256]	[88, 13, 1, 1]	[1, 13, 256, 256]	1,157