Supplementary Material:
Reversible GANs for Memory-efficient Image-to-Image Translation

Tycho F.A. van der Ouderaa
University of Amsterdam
tychovdo@gmail.com

Daniel E. Worrall
University of Amsterdam
d.e.worrall@uva.nl

1. Implementation Details

We provide a Pytorch [9] implementation on Github. Our code extends the image-to-image translation framework from [13] with several reversible models in 2D and 3D. The reversible blocks are implemented using a modified version of MemCNN [12].

1.1. Generator architecture

2d Architecture All 2d models adapt network architectures similar to those used in [13] and [4]. The encoders $\mathrm{Enc}_X, \mathrm{Dec}_Y$ consist of a 7×7 convolutional layer that maps 3 input channels to K channels, followed by two 3×3 convolutional layers with stride 2 that spatially downsample $(/4)$ the signal and increase $(\times 2)$ the channel dimension. We also refer to K as the *width* of our network. As reversible core C, we use R sequential reversible residual layers (with $R = 6$ for 128×128 *Cityscapes* data and $R = 9$ for 256×256 *Maps* data). We consider the amount of reversible residual layers in the core to be the *depth* of our network. The decoders Dec_X and Dec_Y are build out of two 3×3 fractionally-strided convolutional layers [1], followed by a 7×7 convolutional layer projecting the final features to 3 output channels.

We apply reflection padding before every convolution to avoid spatial downsampling. Each convolutional layer is followed by an instance normalization layer [11] and a ReLU nonlinearity, except for the last convolutional layer which is directly followed by a Tanh non-linearity to scale the output within $[-1, 1]$, just like the normalized data.

A full schematic version of the 2D architecture can be found in Figure 1. A diagram of the (identical) $\mathtt{NN}_1$ and $\mathtt{NN}_2$ functions used in the 2D reversible block are shown in Figure 2.

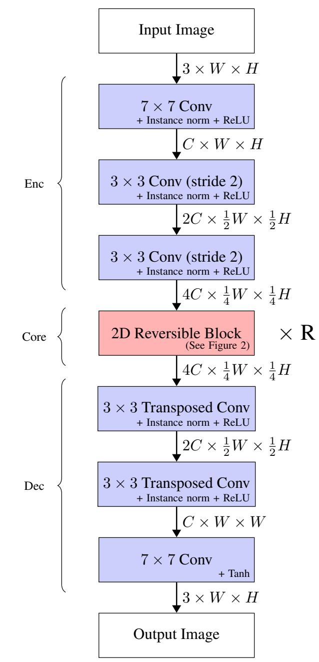
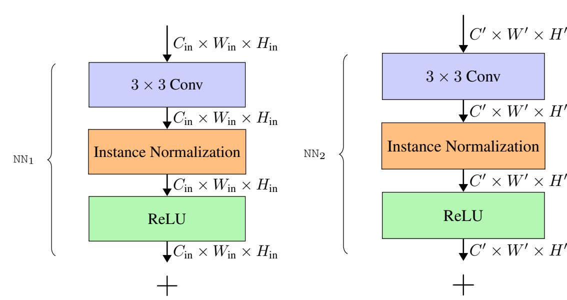


Figure 1. 2D Generator Architecture



(a) $\mathtt{NN}_1$ (b) $\mathtt{NN}_2$

Figure 2. Schematic representation of $\mathtt{NN}_1$ and $\mathtt{NN}_2$ in 2D Reversible Residual Block.

[1] 'Fractionally-strided convolutional layers' or 'transposed convolutions' are sometimes referred to as 'deconvolutions' in literature. To avoid confusion, especially in the context of invertibility, we follow this [2] guide on convolutional arithmetic, and only refer to the term 'deconvolution' when we speak of the mathematical inverse of a convolution, which is different from the fractionally-strided convolution.

3d Architecture For the 3-dimensional super-resolution task (*HTC Brains*), we consider our input and output to be equally sized. Therefore, we first up-sample the images from the low-resolution input domain, before feeding them to the model. It is known that this method also helps to prevent checkerboard-like artifacts [8]. The first layer in our model is a $3 \times 3 \times 3$ convolution layer that increases the channel dimension to K, and is directly followed by an instance normalization layer and a ReLU non-linearity. Then we apply an arbitrary amount of 3D reversible blocks using additive coupling, with the following sequence for $\mathtt{NN}_1$ and $\mathtt{NN}_2$: a $3 \times 3 \times 3$ convolutional layer, an instance normalization layer, a ReLU non-linearity and another $3 \times 3 \times 3$ convolution. We use reflection padding of 1 to ensure that the $\mathtt{NN}_1$ and $\mathtt{NN}_2$ are volume-preserving. Also, we initialize the reversible blocks perform as the identity mapping, by initializing the weights of the last convolutional layer in the reversible block with zeros. This trick has previously shown to be effective in the context of reversible networks [6].

A full schematic version of the 3D generator can be found in Figure 3. A diagram illustrating $\mathtt{NN}_1$ and $\mathtt{NN}_2$ used in the 3D reversible block is shown in Figure 4.

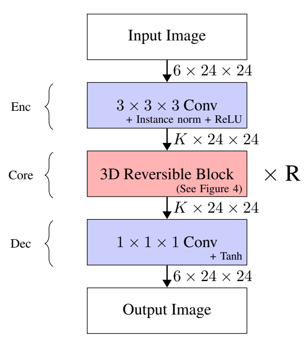
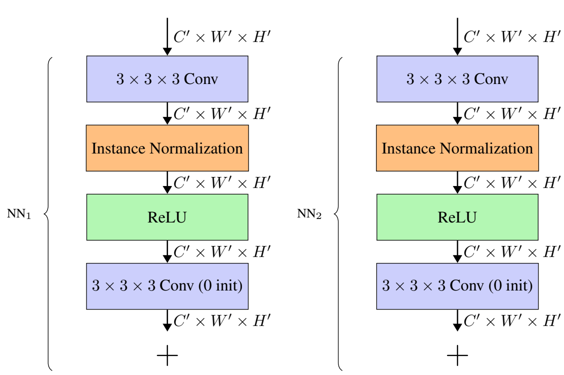


Figure 3. 3D RevGAN Architecture



(a) (b)

Figure 4. 3D RevGAN Architecture

1.2. Discriminator Architecture

For the discriminator, we adapt the same architecture as used in [13], also known as PatchGAN. We use subsequent 4×4 convolutional layers with stride 2 followed by LeakyReLU (with 0.2 slope) non-linearities. The first layer projects the input to 64 layers, followed by three layers each doubling the channel dimension. Finally, we obtain a 1-dimensional outputs by applying a 1×1 convolution followed by a Sigmoid. The 3D models use a very similar architecture and solely replacing the 2D convolutional kernels by equally sized 3D convolutional layers (e.g. 3×3 kernels become $3 \times 3 \times 3$ kernels).

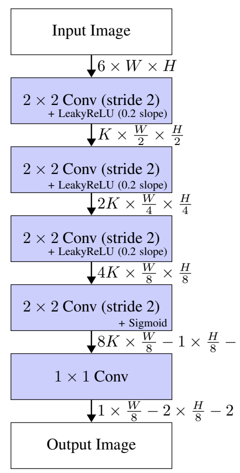


Figure 5. 2D PatchGAN Discriminator

1.3. Hyper-parameters

A summary of the used hyper-parameters can be found in Table 1 below.

Parameter	**2D**	**3D**
Data size	$3 \times 128 \times 128$ or $3 \times 256 \times 256$	$6 \times 24 \times 24 \times 24$
Weight initialization	$\mathcal{N}(\mu = 0, \sigma = 0.02)$	
Normalization	Instance Norm	
Dropout	No	
Optimizer	Adam [5]	
Optimizer params	$\beta_1 = 0.5, \beta_2 = 0.999$	
Epochs	200	20
Batch size	1	
Learning rate	0.002	
Learning rate decay	Keep fixed first half of epochs. Linearly decay to 0 in second half of epochs.	

Table 1. Summary of hyper-parameters

Depth	Width	Params	Memory Model	Naive		Memory Saving	
				+ Activations	Training Time (s / sample)	+ Activations	Training Time (s / sample)
CycleGAN †	32	3.9 M	367.0 ± 0.00	**650.0** ± 0.00	0.63 ± 0.02	n/a	n/a
Unpaired RevGAN	32	1.3 M	334.8 ± 0.43	682.3 ± 0.43	0.67 ± 0.03	**366.53** ± 0.50	0.63 ± 0.02
Unpaired RevGAN †	56	3.9 M	357.8 ± 0.43	1184.5 ± 0.50	0.91 ± 0.02	**640.50** ± 0.50	1.03 ± 0.02
Pix2pix †	32	3.9 M	341.0 ± 0.00	163.0 ± 0.00	0.31 ± 0.00	n/a	n/a
Paired RevGAN	32	1.3 M	333.5 ± 0.50	341.0 ± 1.00	0.46 ± 0.03	183.0 ± 0.00	0.44 ± 0.02
Paired RevGAN †	56	3.9 M	356.0 ± 0.00	592.0 ± 0.00	0.58 ± 0.02	320.0 ± 0.59	0.59 ± 0.02

Table 2. Measurements of memory usage and computation time while performing the *Cityscapes* experiments. LEFT Model configurations. CENTER Memory usage to store model parameters. RIGHT Memory usage to store activations and training time per sample while taking advantage of the memory-efficiency of reversible residual layers (Memory Saving) and without (Naive). TOP Unpaired models. BOTTOM Paired models.

Model	Width	Params	Memory Model	Naive		Memory Saving	
				+ Activations	Training Time (s / sample)	+ Activations	Training Time (s / sample)
CycleGAN †	32	5.7 M	391.0 ± 0.00	**800.0** ± 0.00	0.73 ± 0.01	n/a	n/a
Unpaired RevGAN	32	1.7 M	337.5 ± 0.50	844.5 ± 0.50	0.82 ± 0.02	**338.3** ± 0.49	0.74 ± 0.01
Unpaired RevGAN †	58	5.6 M	371.1 ± 0.69	1540.0 ± 0.52	1.17 ± 0.02	**663.7** ± 0.47	1.34 ± 0.02
Unpaired RevGAN	64	6.8 M	404.0 ± 0.00	1687.0 ± 0.00	1.19 ± 0.01	**723.0** ± 0.00	1.39 ± 0.01
Pix2pix †	32	5.7 M	353.0 ± 0.00	200.0 ± 0.00	0.37 ± 0.00	n/a	n/a
Paired RevGAN	32	1.7 M	336.0 ± 0.00	422.0 ± 0.00	0.54 ± 0.02	183.0 ± 0.00	0.50 ± 0.02
Paired RevGAN †	58	5.6 M	368.0 ± 0.00	770.0 ± 0.00	0.72 ± 0.02	332.0 ± 0.00	0.79 ± 0.01
Paired RevGAN	64	6.8 M	417.0 ± 0.00	830.0 ± 0.00	0.72 ± 0.01	361.0 ± 0.00	0.82 ± 0.01

Table 3. Measurements of memory usage and computation time while performing the *Maps* experiments. LEFT Model configurations. CENTER Memory usage to store model parameters. RIGHT Memory usage to store activations and training time per sample while taking advantage of the memory-efficiency of reversible residual layers (Memory Saving) and without (Naive). TOP Unpaired models. BOTTOM Paired models.

Model	Width	Depth	Params	Memory Model	Naive		Memory Saving	
					+ Activations	Training Time (s / sample)	+ Activations	Training Time (s / sample)
CycleGAN	32	6	3.9 M	**367.0** ± 0.00	**650** ± 0.00	0.61 ± 0.02	n/a	n/a
CycleGAN †	32	9	5.7 M	**391.0** ± 0.00	**800** ± 0.00	0.73 ± 0.01	n/a	n/a
CycleGAN	32	12	7.5 M	**415.7** ± 0.00	**950** ± 0.00	0.84 ± 0.02	n/a	n/a
CycleGAN	32	18	11.0 M	**463.7** ± 0.00	**1250** ± 0.00	1.07 ± 0.02	n/a	n/a
CycleGAN	32	30	18.1 M	**559.0** ± 0.00	**1850** ± 0.00	1.51 ± 0.02	n/a	n/a
Unpaired RevGAN	58	6	1.2 M	**358.78** ± 0.64	1243.28 ± 0.72	0.95 ± 0.03	**663.37** ± 0.70	1.10 ± 0.02
Unpaired RevGAN †	58	9	1.7 M	**371.19** ± 0.78	1540.11 ± 0.53	1.16 ± 0.02	**663.67** ± 0.47	1.35 ± 0.02
Unpaired RevGAN	58	12	2.1 M	**382.87** ± 0.77	1837.33 ± 0.47	1.39 ± 0.02	**663.50** ± 0.50	1.64 ± 0.02
Unpaired RevGAN	58	18	3.1 M	**406.88** ± 0.32	2431.43 ± 0.59	1.83 ± 0.02	**663.12** ± 0.32	2.18 ± 0.02
Unpaired RevGAN	58	30	4.8 M	**454.69** ± 0.46	3619.00 ± 0.75	2.74 ± 0.03	**663.31** ± 0.46	3.27 ± 0.01

Table 4. Measurements of memory usage and computation time while performing the *Maps* dataset using the CycleGAN and Unpaired RevGAN model at different depths. LEFT Model configurations. CENTER Memory usage to store activations and training time per sample while taking advantage of the memory-efficiency of reversible residual layers (Memory Saving) and without (Naive). TOP CycleGAN models at different depths. BOTTOM Unpaired RevGAN models at different depths.

2. Memory Cost and Training Times

To further evaluate the model performance, we report the memory cost split out in the cost to store the model parameters and the cost to store activations. For the latter, we measure the memory consumption both using the memory-efficiency of the reversible residual layers (*Memory Saving*), if possible, and without (*Naive*). For each experiment, we also report the average training time per sample. In Table 2, the memory costs and training time for the *Cityscapes* experiments are shown. In Table 3, the memory costs and training time for the experiments on the *Maps* dataset can be found. In Table 4, the memory cost and training time for CycleGAN and Unpaired RevGAN models at different depths are given.

The measurements in Table 2, Table 3 and Table 4 were obtained by training models on a NVIDIA K40m GPU using a warm-up period of 100 training samples after which the GPU memory usage was measured over the next 100 samples by querying the `nvidia-smi` toolkit. We report means and standard deviations.

3. Negative Results

- We tried to replace *additive coupling* with *affine coupling*, which has been applied succesfully in the context of reversible networks by [6]. In theory, affine coupling layers are more general and more expressive than additive coupling. We found, however, that affine coupling degraded performance and made training more unstable. Nevertheless, it would be interesting to see whether affine coupling outperforms additive coupling combined with other architectures or hyperparameters.

- We tried to replace the down-sampling and up-sampling layers with sub-pixel convolutions [10] in our 2D and 3D models, which have also been applied succesfully in the context of invertible architectures [3], but found that it degraded performance. Sub-pixel convolutions were originally proposed to save memory in super-resolution problems by applying convolutions in lower-dimensional space rather than in the higher-dimensional target space. The RevGAN model, on the other hand, saves memory by not having to store the activations of the reversible layers.

- We tried to replace the transposed convolutions used for up-sampling in our model with nearest-neighbour and bilinear upsampling to prevent checkerboard-like aftifacts as explained in [8], but found that it degraded performance. Furthermore, we observed that the checkerboard appeared in early training stages, but that they disappeared after a sufficient amount of training iterations.

- We tried *Consensus Optimization* [7] to stabilize training by encouraging agreement between the discriminators and the generators. Consensus optimization boils down to regularization term over the second-order derivative over our gradients, which is a computationally intensive task. We stopped using it because it slowed down training too much.

- We found that the invertible core can be replaced with a continuous-depth residual networks introduced in [1] of which the forward and inverse pass are trained using an ordinary differential equation (ODE) solver. Due to time constraints, we were not able to evaluate the performance of this method. Some benefits of the method are constant $\mathcal{O}(1)$ memory cost as a function of depth (similar to reversible layers) and explicit control over the numerical error. In future work we plan to explore the use of neural ordinary (or even stochastic) differential equations in the context of image-to-image translation.

References

[1] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud. Neural ordinary differential equations. *arXiv preprint arXiv:1806.07366*, 2018.

[2] V. Dumoulin and F. Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

[3] J.-H. Jacobsen, A. Smeulders, and E. Oyallon. i-revnet: Deep invertible networks. *arXiv preprint arXiv:1802.07088*, 2018.

[4] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, pages 694–711. Springer, 2016.

[5] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[6] D. P. Kingma and P. Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *arXiv preprint arXiv:1807.03039*, 2018.

[7] L. Mescheder, S. Nowozin, and A. Geiger. The numerics of gans. In *Advances in Neural Information Processing Systems*, pages 1825–1835, 2017.

[8] A. Odena, V. Dumoulin, and C. Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016.

[9] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. *NIPS 2017 Workshop*, 2017.

[10] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1874–1883, 2016.

[11] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Instance normalization: the missing ingredient for fast stylization. corr abs/1607.0 (2016).

[12] S. C. van de Leemput, J. Teuwen, and R. Manniesing. Memcnn: a framework for developing memory efficient deep invertible networks. *ICLR 2018 Workshop*, 2018.

[13] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *arXiv preprint*, 2017.

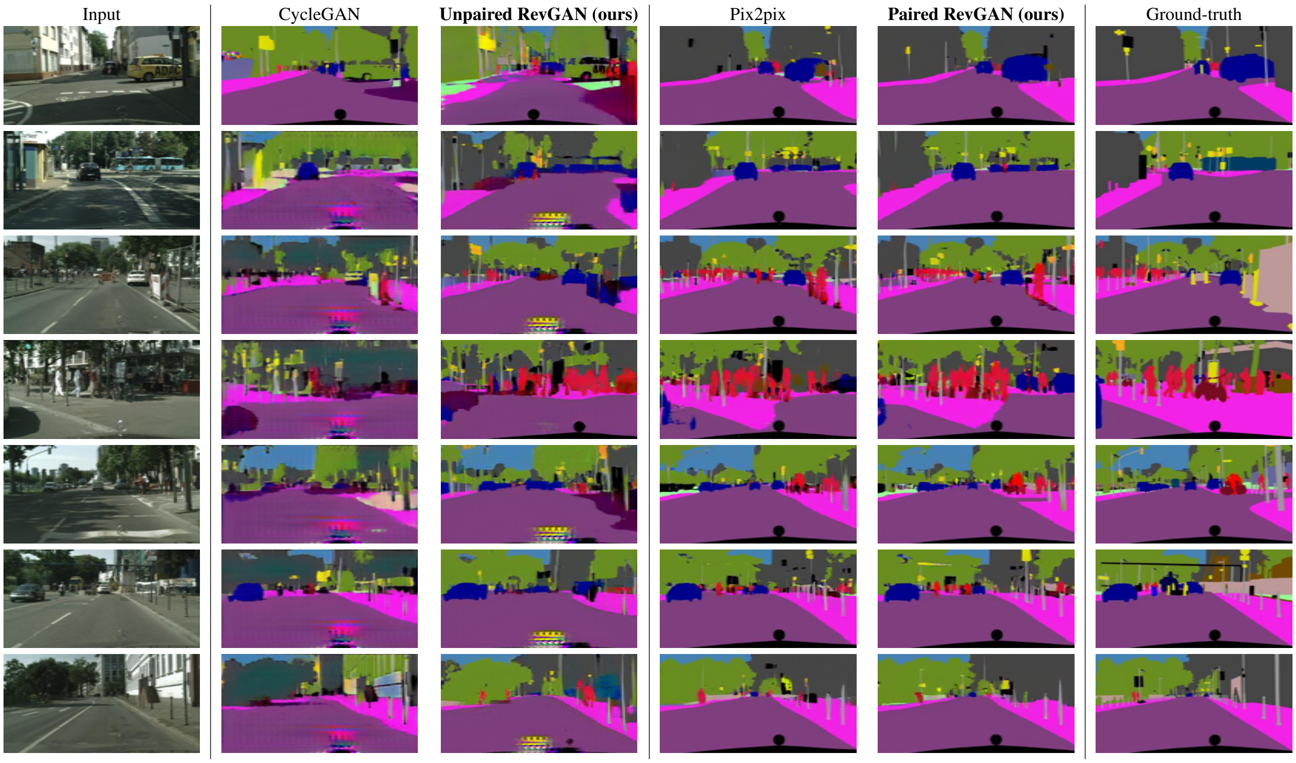


Figure 6. Additional image mappings for *photo→label* on the *Cityscapes* test set.

Figure 7. Additional image mappings for *label→photo* on the *Cityscapes* test set.

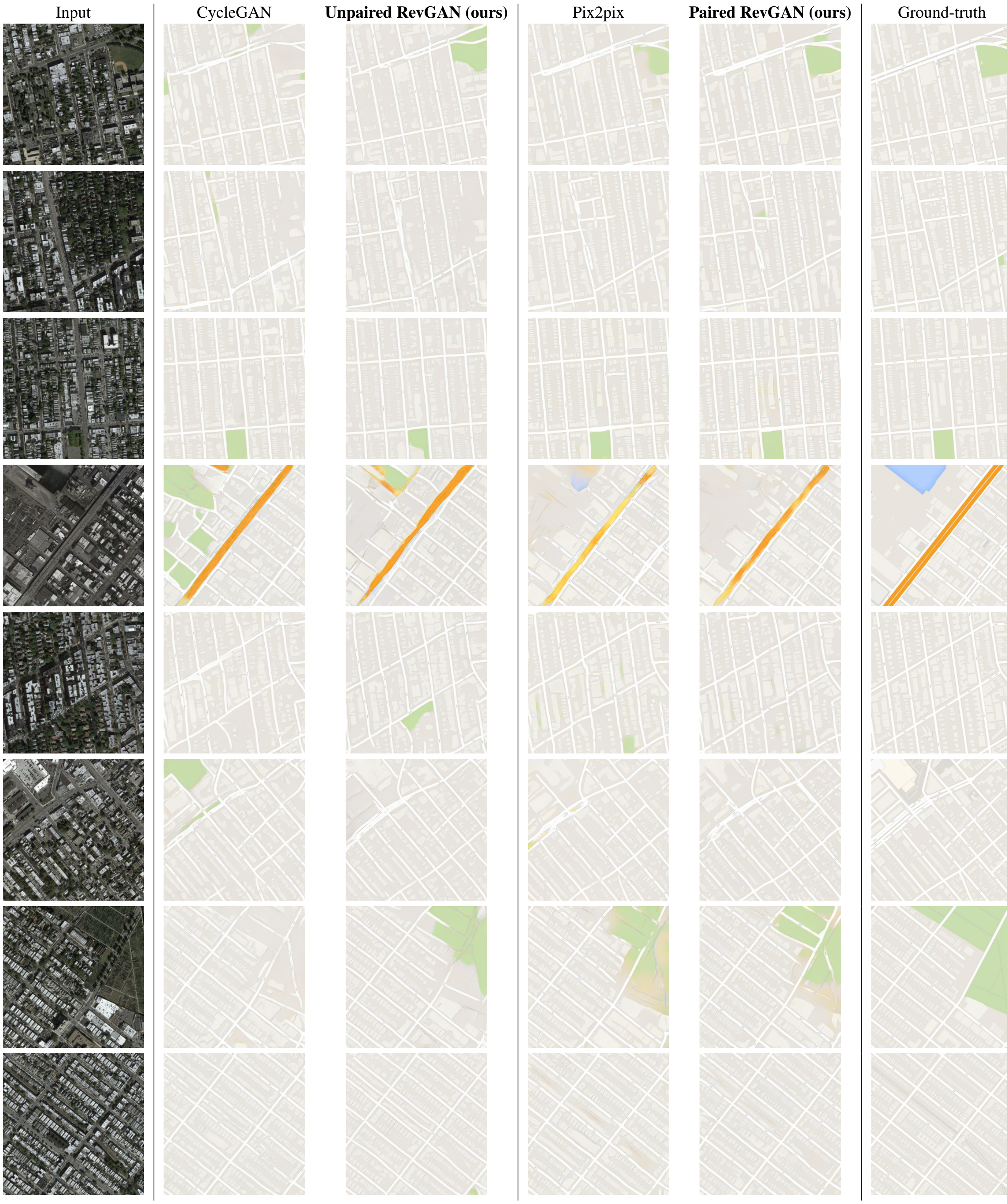


Figure 8. Additional image mappings for *satellite→maps* on *Maps* test set.

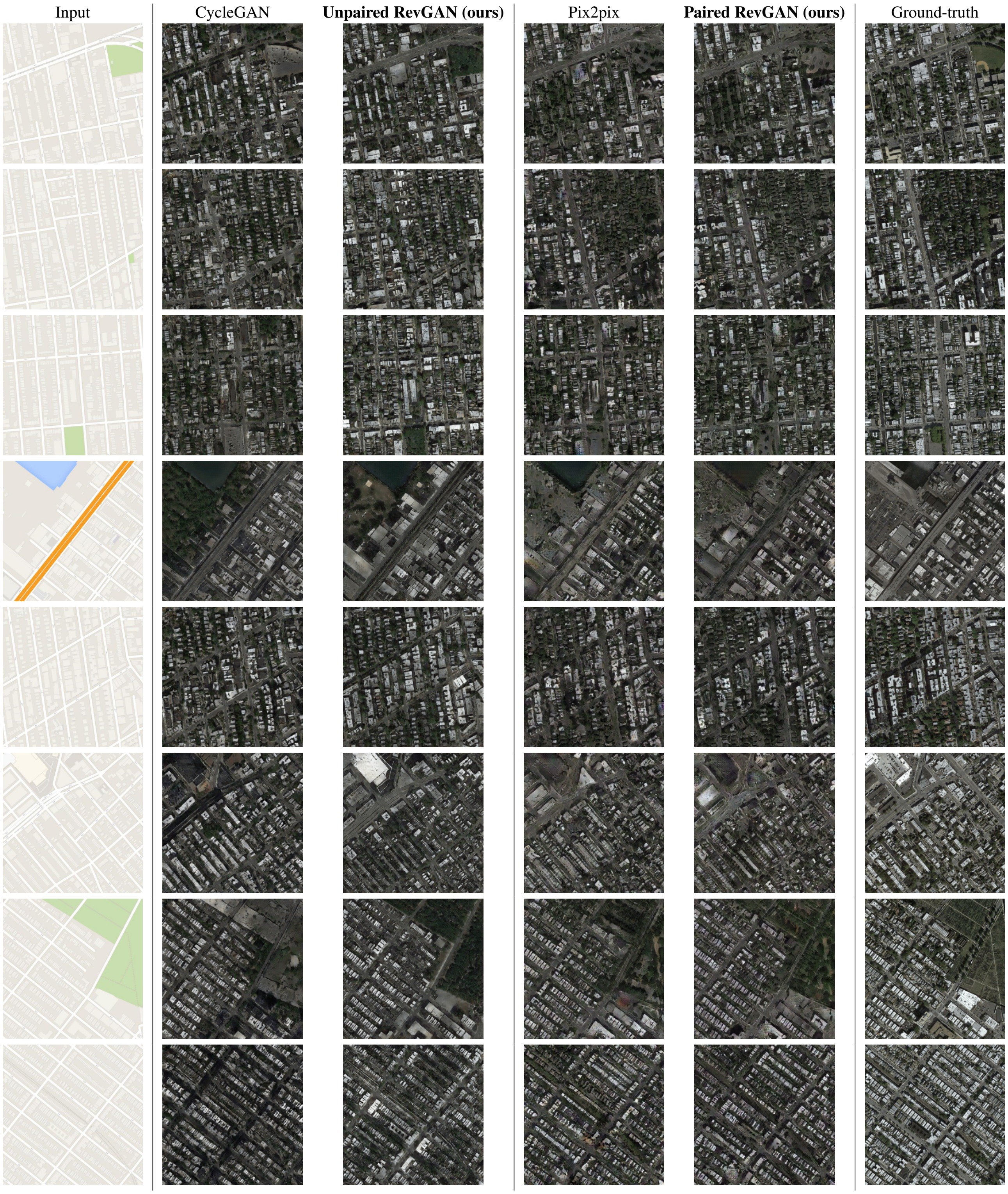


Figure 9. Additional image mappings for *maps*→*satellite* on *Maps* test set.