Supplemental Material: Orientation-Aware Semantic Segmentation on Icosahedron Spheres

Chao Zhang*[1] Stephan Liwicki*[1] William Smith[2] Roberto Cipolla[1,3]

[1]Toshiba Research Europe Limited, Cambridge, United Kingdom
[2]University of York, United Kingdom [3]University of Cambridge, United Kingdom

1. CNN Operation on the Icosahedron Mesh

We include the pseudo code of our main CNN operators, applied to the icosahedron mesh components, denoted $\{C_i\}_{i=0}^4$. Note, many operations will be a direct result of a combination of these operators (*i.e.* Pyramid Pooling Layers [4]). First we detail padding in Algorithm 1. Our orientation-aware hexagonal convolutions with arc-based interpolations for north-alignment are given in Algorithm 2. Algorithm 3 presents up-sampling. We emphasize, convolutions with kernel size 1, pooling, batch normalization, non-linearities and biases are directly computed on the spherical components without padding, through standard unchanged CNN operators.

Algorithm 1: Padding & WestPadding (top & left only)

Result: Given sphere components $\{C_i\}_{i=0}^4$ of height $2W$ and width W compute padded $\{P_i\}_{i=0}^4$

for $i \leftarrow \{0, \dots, 4\}$ **do** `// pad each component`

$C_w \leftarrow C_{\mathtt{mod}(i-1,5)}$; `// west neighbor`

$T \leftarrow \begin{bmatrix} C_w(W,W) & \text{to} & C_w(1,W) & 0 \end{bmatrix}$;

$L \leftarrow \begin{bmatrix} \begin{bmatrix} C_w(W+1,W) & \text{to} & C_w(2W,W) \end{bmatrix}^\intercal \\ \begin{bmatrix} C_w(2W,W-1) & \text{to} & C_w(2W,1) \end{bmatrix}^\intercal \\ 0 \end{bmatrix}$;

$P_i \leftarrow \begin{bmatrix} T \\ \begin{bmatrix} L & C_i \end{bmatrix} \end{bmatrix}$; `// top & left`

if *pad all sides* **then**

$C_e \leftarrow C_{\mathtt{mod}(i+1,5)}$; `// east neighbor`

$B \leftarrow \begin{bmatrix} 0 & C_e(2W,1) & \text{to} & C_e(W+1,1) \end{bmatrix}$;

$R \leftarrow \begin{bmatrix} 0 \\ \begin{bmatrix} C_e(1,W) & \text{to} & C_e(1,1) \end{bmatrix}^\intercal \\ \begin{bmatrix} C_e(1,1) & \text{to} & C_e(W+1,1) \end{bmatrix}^\intercal \end{bmatrix}$;

$P_i \leftarrow \begin{bmatrix} \begin{bmatrix} P_i \\ B \end{bmatrix} & R \end{bmatrix}$; `// bottom & right`

end

end

Algorithm 2: Hexagonal Convolution (HexConv)

Result: Given components $\{C_i\}_{i=0}^4$ and precomputed interpolation weights $\{A_i\}_{i=0}^4$ get filter results $\{F_i\}_{i=0}^4$ of same size.

$\{C_i\}_{i=0}^4 \leftarrow \text{Padding}(\{C_i\}_{i=0}^4)$; `// Alg. 1`

$\mathbf{W}_1 \leftarrow \begin{bmatrix} w_2 & w_1 & 0 \\ w_3 & w_7 & w_6 \\ 0 & w_4 & w_5 \end{bmatrix}$; `// Hexagon filter`

$\mathbf{W}_2 \leftarrow \begin{bmatrix} w_3 & w_2 & 0 \\ w_4 & w_7 & w_1 \\ 0 & w_5 & w_6 \end{bmatrix}$; `// Shift weights`

for $i \leftarrow \{0, \dots, 4\}$ **do**

$F_i^1 \leftarrow \text{conv2d}(C_i, \mathbf{W}_1)$; `// standard 2D conv`

$F_i^2 \leftarrow \text{conv2d}(C_i, \mathbf{W}_2)$;

`// Element-wise Interpolation`

$F_i \leftarrow A_i \otimes F_i^1 + (1 - A_i) \otimes F_i^2$

end

Algorithm 3: Bi-linear Up-sampling on Sphere

Result: Given components $\{C_i\}_{i=0}^4$ get bi-linear up-sampling $\{F_i\}_{i=0}^4$.

$\{C_i\}_{i=0}^4 \leftarrow \text{WestPadding}(\{C_i\}_{i=0}^4)$; `// Alg. 1`

for $i \leftarrow \{0, \dots, 4\}$ **do**

$F_i \leftarrow \text{upsample}(C_i)$; `// 2× up-sampling`

Cut 1 pixel width from all sides of F_i;

end

Additionally, for purpose of visualisation, we illustrate the resulting convolutions, after north alignment in Figure 1.

2. Evaluation Details

In this section, we include details of network architectures and parameters used in our experiments.

*Equal contribution: {chao.zhang, stephan.liwicki}@crl.toshiba.co.uk

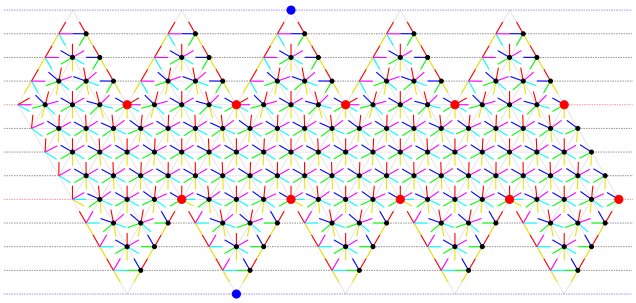

Figure 1. Interpolated convolutions on the unfolded mesh ($r = 2$). Orientations are north-south aligned, while 5-degree connections (red) are padded through duplication, and poles (blue) are ignored.

Level	a	Block	b	c	s
4	1	HexConv	–	16	1
4	16	ResBlock	64	64	2
3	64	ResBlock	256	256	2
2	256	MaxPool	–	–	1
–	256	Dense	–	10	1

Table 1. HexRUNet-C architecture used in Omni-MNIST experiments. a, b, c stands for input channels, bottleneck channels, and output channels. s stands for strides: 2 means downsampling.

Branch 1	Branch 2
–	Conv2D 1/1, (a, b),pool,BN,f
–	HexConv, (b, b),BN,f
Conv2D 1/1, (a, c),pool,BN	Conv2D 1/1, (b, c),BN
add, f	

Table 2. Residual block (ResBlock), where a, b, c stands for input channels, bottleneck channels and output channels. BN is short for Batch Normalization, and f stands for Rectified Linear Unit activation function (ReLU).

2.1. Omni-MNIST

The input signal for this experiment is on a level-4 mesh, we use max pooling before the final dense layer rather than average pooling used in [1]. We train our network HexRUNet-C with a batch size of 15, initial learning rate of 0.001, and use the Adam optimizer. We use the cross-entropy loss for the digits classification task. The network structure is illustrated in Table 1. The residual block is used across this and other networks, and is shown in Table 2. Total number of parameters of HexRUNet-C is 74,730.

2.2. Climate Pattern

The input signal for this experiment is on a level-5 mesh, the number of input channels is 16. We use the same architecture as the semantic segmentation task in §2.3 (Table 3). We have included two variants using 8 or 32 as the feature maps in the first HexConv operation, called HexRUNet-8 and HexRUNet-32. We train our network with a batch size 60, initial learning rate of 0.001 with Adam optimizer. We train using weighted cross-entropy loss, due to the unbalanced classes distributions. Total number of parameters is 7,543,331 for HexRUNet-32 and 476,747 for HexRUNet-8. UGSCNN [1] uses 8 initial feature maps with a total of 328,339 parameters.

Level	a	Block	b	c	s
5	4	HexConv	–	16	1
5	16	ResBlock	16	32	2
4	32	ResBlock	32	64	2
3	64	ResBlock	64	128	2
2	128	ResBlock	128	256	2
1	256	ResBlock	256	256	2
0	256	HexConv$^{\mathrm{T}}$	–	256	0.5
1	256x2	ResBlock	128	128	0.5
2	128x2	ResBlock	64	64	0.5
3	64x2	ResBlock	32	32	0.5
4	32x2	ResBlock	16	16	0.5
5	16x2	ResBlock	16	16	1
5	16	HexConv	–	13	1

Table 3. HexRUNet architecture used in 2D3DS semantic segmentation experiments. a, b, c stands for input channels, bottleneck channels, and output channels. s stands for strides. When $s = 2$, down-sampling is performed, and when $s = 0.5$, up-sampling is done (using up-sampling and point-wise HexConv).

2.3. 2D3DS

The input signal for this experiment is on a level-5 mesh, and the number of input channels is 4 for RGB and Depth. The network structure is illustrated in Table 3. The network contains two parts: encoder layers and decoder layers. At level-0, we apply upsampling and a point-wise HexConv operation to increase the resolution to level-1 (denoted HexConv$^{\mathrm{T}}$). In the subsequent layers, the input will be concatenated with the output from previous layers at corresponding levels. This can be seen for rows in which the input size is doubled.

We train our network with a batch size 32, initial learning rate of 0.001 with Adam optimizer, up to 500 epochs. In contrast to UGSCNN [1] which uses 32 feature maps and 5,180,239 parameters, we employ 16 feature maps for the first layer, resulting in 1,585,885 parameters to ensure a competitive comparison between the frameworks. Following [1], class-wise weighted cross-entropy loss is used to balance the class examples. Note that the number of output channels is 13 rather than 15, since the 2D3DS dataset has two invalid classes ("unknown" and "invalid"), which are not evaluated during validation.

2.4. Omni-SYNTHIA

We create our own Omni-SYNTHIA dataset from SYNTHIA data [3]. In the original SYNTHIA data, synthetic views are captured with a stereo set-up consisting of 2 clusters of 4 cameras. For Omni-SYNTHIA we use the left-

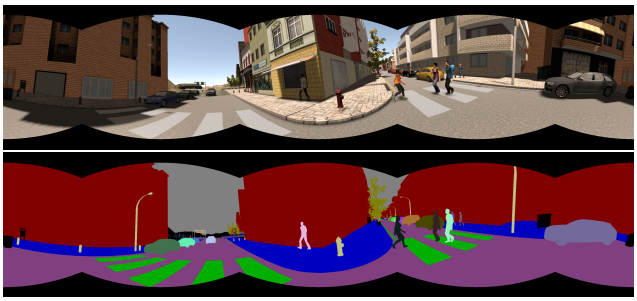

Figure 2. An example of our Omni-SYNTHIA dataset images (top) and labels (bottom). (Top and bottom parts of the images are cropped only for visualization.)

Input	Operator	Output
a	HexConv,BN,f	c
c	HexConv,BN,f	c
c	Pool	c

Table 4. U-Net encoder block (Encoder), where a, c stands for input channels and output channels. BN is short for Batch Normalization, and f stands for Rectified Linear Unit activation function (ReLU).

Input	Operator	Output
a	HexConv,BN,f	b
b	HexConv,BN,f	b
b	Up	b
b	Conv2D 1/1, BN, f	c

Table 5. U-Net decoder block (Decoder), where a, b, c stands for input channels, middle channels and output channels. BN is short for Batch Normalization, and f stands for Rectified Linear Unit activation function (ReLU).

stereo cluster which captures 4 viewpoints with a common camera center. The views capture 90° intervals with a filed of view of 100° each. We use the visual overlap to create an omnidirectional view to our needs (fig. 2). Since perspective images are of resolution 760×1280 the final equirectangular RGB images are set to 2096×4192. In particular, we keep height/width ratio 1 to 2, and compute the overlap between adjacent viewpoints to find the needed equirectangular resolution.

Multiple sequences are acquired simulating different cities of four seasons with drastic change of appearance. Ground-truth includes pixel-wise semantic labels of 14 classes (including "invalid"). In our experiments, the five "SUMMER" sequences are chosen to make our omnidirectional dataset. Specifically, sequences simulating New York-like (1 and 2) and Highway-like (5 and 6) scenes are used as training set, while European-like sequence (4) is employed for validation. For each sequence, we choose every second frame. In total, 2,269 equirectangular RGB images are generated (1818 for training, 451 for testing). Depth maps are not used in this experiment.

Level	a	Block	b	c	s
6	3	Encoder	–	32	2
5	32	Encoder	–	64	2
4	64	Encoder	–	128	2
3	128	Encoder	–	256	2
2	256	Decoder	512	256	0.5
3	256x2	Decoder	256	128	0.5
4	128x2	Decoder	128	64	0.5
5	64x2	Decoder	64	32	0.5
6	32x2	HexConv,BN,f	–	32	1
6	32	HexConv,BN,f	–	32	1
6	32	HexConv	–	13	1

Table 6. HexUNet architecture used in Omni-SYNTHIA semantic segmentation experiments. a, b, c stands for input channels, bottleneck channels, and output channels. s stands for strides. When $s = 2$, downsampling is performed, and when $s = 0.5$, up-sampling is applied using bi-linear up-sampling and a point-wise convolution.

In this experiment, we use the standard U-Net architecture [2] to facilitate weight transfer from planar U-Net. We call this "HexUNet", and the architecture is illustrated in Table 6. Table 4 and 5 show the detailed encoder and decoder block. Our model has a total of 7,245,101 parameters. Batch size 32, 8 and 2 are used for resolution level 6, 7 and 8 respectively to ensure memory fit on our GPU.

Comparision with state of the art Spherical input at level-6 is the maximum resolution we could fit in GPU using the provided implementation of [1], so we choose to compare our method to UGSCNN using data sampled at level-6 mesh. Planar U-Net [2] using original perspective images is also evaluated. Images are sub-sampled to match the icosahedron resolution.

Specifically, we count the number of vertices on the icosahedron mesh that fall onto the image region. We then set the image resolution to be approximately equivalent to this number of vertices, resulting in image resolution 48×80 for level-6, 96×160 for level-7 and 192×320 for level-8 meshes. To compare network efficiency in terms of training time, we show the average training time on the Omni-SYNTHIA dataset. Evaluations are performed on a single Nvidia 1080Ti GPU with 11 Gb memory. Average training times are obtained by averaging the first 10 epochs.

Evaluation of Perspective Weights Transfer Using an orientation-aware hexagonal convolution kernel, our method allows direct weights transfer from perspective networks. Initialized with the learned filters of U-Net, we report the results as HexUNet-T in Table 7. To show the effectiveness of direct weights transfer, we limit weight refinement to up to 10 epochs. Results after just one retraining epoch are also shown in Table 7. We emphazie,

Method	$r = 6$	$r = 7$	$r = 8$
UNet (Perspective)	38.8	44.6	43.8
HexUNet-T (1 epoch)	29.4	30.3	35.9
HexUNet-T (10 epochs)	36.7	38.0	45.3
HexUNet (10 epochs)	20.6	10.9	15.1
HexUNet (500 epochs)	43.6	48.3	47.1

Table 7. Comparison of perspective weights transfer on Omni-SYNTHIA.

rather than to further improve overall accuracy, we aim to reduce the number of training epochs *with spherical data* by weights transferal. In particular, the empirical evaluation shows the effectiveness of the weights transfer, as it reaches competitive results with much less training cost (in terms of epochs), especially at resolution $r = 8$. We additionally comparing to source network UNet and our spherical HexUNet trained on up to 10 and 500 epochs.

3. Additional Results

We show additional semantic segmentation results for semantic segmentation on 2D3DS and Omni-SYNTHIA.

3.1. 2D3DS Results

We show additional semantic segmentation results in Fig. 3. Cases (a-c) demonstrate examples on which both our proposed method and UGSCNN [1] fail to predict the correct labels for some objects. The RGB data of case (a) shows bright illuminations which our method wrongly consider as windows. Windows and bookcases are confused in case (b). As shown in case (c), it is challenging to segment the boundary of a bookcase. Our method failed to recognize windows and chairs in case (d). It also poses a hard sample for UGSCNN. Finally, we argue that some of the ground-truth labels are not accurate, for example, while case (e) presents a bookcase some parts of it are labeled as wall. Even though, our method could adequately predict the bookcase. We also believe that with better input resolution, and with better architectures, our proposed method is capable of improving over these cases.

3.2. Omni-SYNTHIA

We conclude with additional results for the Omni-SYNTHIA resolution evaluation. In general, finer segmentation is achieved at higher resolution. Furthermore, as observed in case (a), (c) and (f) segmentation of small objects and boundaries is improved. In (b) an instance of a cyclist is shown. Here the model classifies misc for low resolution, and pedestrian for $r = 7$ and $r = 8$. An indication that the network architecture is not ideal for capturing context at all resolutions is case (d), where buildings are misclassified as misc.

References

[1] Chiyu Max Jiang, Jingwei Huang, Karthik Kashinath, Prabhat, Philip Marcus, and Matthias Nießner. Spherical CNNs on unstructured grids. In *ICLR'19*, 2019. 2, 3, 4, 5

[2] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI'15*, pages 234–241, 2015. 3

[3] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *CVPR'16*, pages 3234–3243, 2016. 2

[4] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017. 1

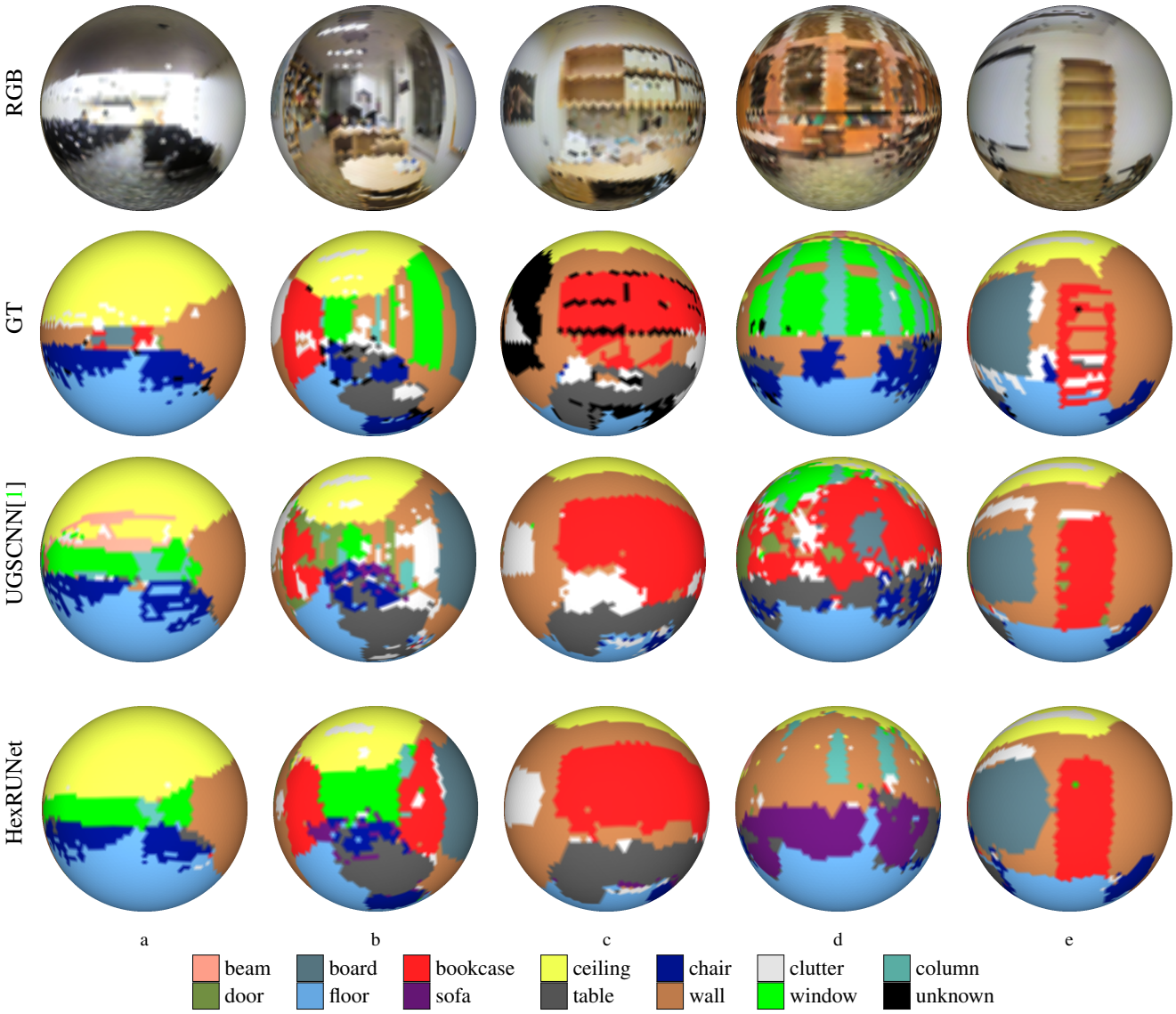

Figure 3. Qualitative segmentation results and failed cases on 2D3DS dataset.

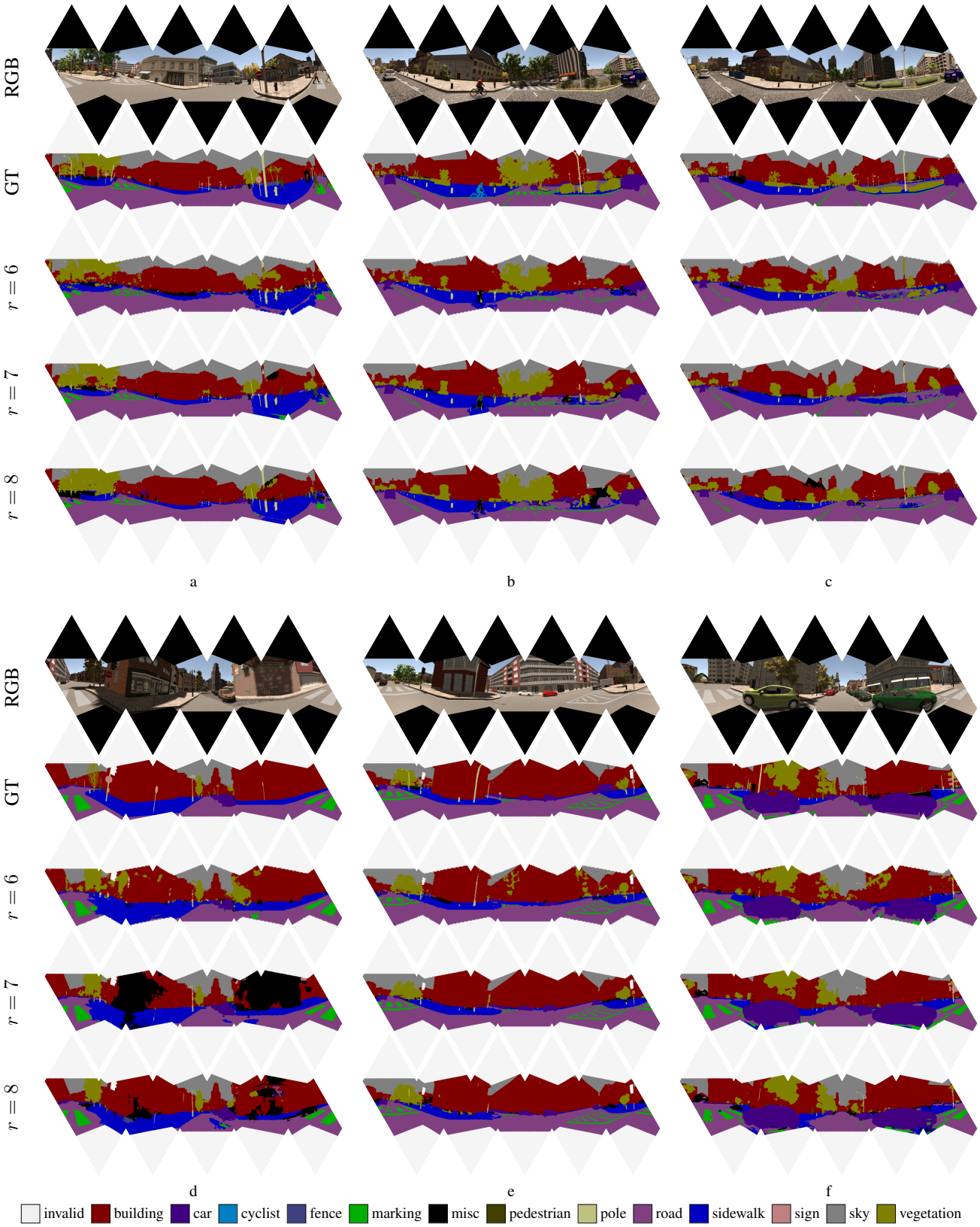

Figure 4. Unfolded visualizations of semantic segmentation results on Omni-SYNTHIA dataset at different resolutions.