Supplementary Material: Task-aligned Part-aware Panoptic Segmentation through Joint Object-Part Representations

Daan de Geus Gijs Dubbelman
Eindhoven University of Technology
{d.c.d.geus, g.dubbelman}@tue.nl

1. Overview

In this document, we provide the following material in addition to the main manuscript:

- In Sec. 2, we present the results of additional experiments, in which we evaluate the effect of different loss weights and data augmentation techniques, and assess the efficiency of TAPPS and other approaches.
- In Sec. 3, we provide more details of the experimental setup, including the implementation details of TAPPS and the strong baseline.
- In Sec. 4, we show examples of predictions by TAPPS on the Pascal-PP and Cityscapes-PP datasets, and qualitatively compare TAPPS to the strong baseline and existing work.

The code for TAPPS is made publicly available through `https://tue-mps.github.io/tapps/`.

2. Additional experiments

Loss weights. In Tab. 1, we show the impact of using different loss weights to balance the losses for object-level segmentation and part-level segmentation, using the weights λ_{obj} and λ_{pt} (see Eq. 1 of the main manuscript). We find that balancing the losses with $\lambda_{\text{obj}} = \lambda_{\text{pt}} = 1.0$ yields the best performance. As expected, the object-level segmentation performance, reflected in the PQ metric, drops when λ_{obj} is decreased. Conversely, the part-level segmentation performance, reflected in the PartSQ$^{\text{Pt}}$ metric, drops when λ_{pt} is decreased.

Data augmentation techniques. As explained in Sec. 3.1 of this document, we use large-scale jittering data augmentation for our experiments on Pascal-PP. However, the existing works Panoptic-PartFormer [8] and Panoptic-PartFormer++ [9] use less aggressive data augmentation techniques during training. To show that the difference in data augmentation techniques is not the main reason that TAPPS outperforms these methods, we train TAPPS with the same data augmentation techniques that are used by these methods, according to the official code repository of Panoptic-PartFormer. Specifically, we apply a random horizontal flip, and then directly resize the image such that the smallest side is 800 pixels. The results in Tab. 2 show that TAPPS still significantly outperforms both existing methods when using these data augmentation techniques. This shows that the improvement by TAPPS is mainly caused by the methodology and network architecture, and not by the data augmentation differences. Finally, we note that there are no differences in data augmentation techniques between our method and existing works on the Cityscapes-PP dataset, so these results in Tab. 2 of the main manuscript are directly comparable.

λ_{pt}	λ_{obj}	PartPQ			PartSQ	PQ
		Pt	No pt	All	Pt	All
1.0	1.0	67.2	50.4	54.7	75.1	57.7
0.5	1.0	67.0	50.1	54.4	75.0	57.5
0.2	1.0	65.9	50.1	54.1	73.6	57.5
0.1	1.0	64.9	50.2	53.9	72.0	57.6
1.0	0.5	67.3	50.2	54.5	75.5	57.4
1.0	0.2	66.6	49.1	53.5	75.5	56.4
1.0	0.1	66.3	48.0	52.7	75.6	55.5

Table 1. **Loss weights.** Evaluated on Pascal-PP [1, 4, 5, 14], with pre-training on COCO panoptic segmentation [10].

Method	Data augmentation	PartPQ			PartSQ	PQ
		Pt	No pt	All	Pt	All
TAPPS (ours)	*Default*	*67.2*	*50.4*	*54.7*	*75.1*	*57.7*
TAPPS (ours)	Flip & resize [8]	**65.8**	**47.2**	**51.9**	**74.8**	**54.9**
Panoptic-PartFormer [8]	Flip & resize [8]	56.1	38.8	43.2	66.8	47.6
Panoptic-PartFormer++ [9]	Flip & resize [8]	52.6	42.6	45.1	60.4	51.6

Table 2. **Comparison with state-of-the-art methods using identical data augmentation techniques.** Evaluated on Pascal-PP [1, 4, 5, 14], with pre-training on COCO panoptic segmentation [10]. For the default data augmentation techniques of TAPPS, see Sec. 3.1.1.

Method	Inference speed	Memory	# Params
Baseline	14.9 fps	2.1 GB	44M
TAPPS (predict all parts)	12.7 fps	7.8 GB	48M
TAPPS (predict compat. parts)	16.1 fps	1.7 GB	48M

(a) **Different versions of TAPPS,** with ResNet-50 backbone.

Method	Inference speed	# Params	PartPQ
Panoptic-PartFormer [8]	11.5 fps	40M	43.2
TAPPS (ours)	16.1 fps	48M	54.7

(b) **TAPPS vs. existing work,** with ResNet-50 backbone.

Table 3. **Efficiency.** We evaluate the average inference speed in frames per second (fps) and the maximum required GPU memory on the Pascal-PP *val* set [1, 4, 5, 14], using an Nvidia A100 GPU.

Efficiency. In Tab. 3a, we show the effect of predicting *only compatible parts* on the model's efficiency. Most importantly, we observe that the default version of TAPPS, which only predicts the masks for the N^{c} *compatible* parts, is much more efficient than the version that predicts masks for *all* N^{pc} part classes, in terms of both inference speed and memory. Moreover, by only considering compatible parts, TAPPS is also more efficient than the baseline that uses *separate* object-level and part-level queries. Another reason that TAPPS is more efficient than the baseline is that its part-level queries do not participate in the self-attention and cross-attention operations in the decoder, unlike those of the baseline. This shows the strength of the simplicity of TAPPS.

In Tab. 3b, we compare TAPPS to existing method Panoptic-PartFormer [8]. The results show that TAPPS obtains a considerably better PartPQ, while using only 8M more parameters, and even being much faster.

3. Experimental setup

In this section, we provide further details about the experimental setup. First, in Sec. 3.1, we describe the implementation details more extensively, for TAPPS, the strong baseline, and the TAPPS version that conducts dynamic part segmentation. Second, in Sec. 3.2, we explain how we obtain the PartPQ scores for Panoptic-PartFormer [8] and Panoptic-PartFormer++ [9] on Pascal-PP.

3.1. Implementation details

The most important implementation details have already been provided in Sec. 4 of the main manuscript. This subsection provides a more comprehensive overview of the implementation details.

3.1.1 General

This subsection describes the implementation details that apply to both TAPPS and the baseline. For completeness, we repeat some of the details already mentioned in the main manuscript.

Both TAPPS and the baseline are implemented on top of the publicly available code of Mask2Former [2], which uses Detectron2 [16]. All experiments are conducted with a batch size of 16, using 4 Nvidia A100 GPUs in total. Following Mask2Former, we optimize all networks using AdamW [12], using a polynomial learning rate schedule with an initial learning rate of 10^{-4}, a power of 0.9, and a weight decay of 0.05. When we apply ImageNet pre-training, we initialize the backbone with weights pre-trained on ImageNet-1K [15]. In case of COCO pre-training, we initialize both the backbone and the compatible decoder layers with weights pre-trained on COCO panoptic segmentation [7, 10]; we use the weights provided in the official repository of Mask2Former. Like Mask2Former, TAPPS applies deep supervision [2]. This means that the segmentation masks and classes are predicted after each transformer layer in the decoder, and that a loss is calculated for these predictions at each of these layers. The overall loss is the sum of the total losses at all transformer layers.

For experiments on *Pascal-PP* [1, 4, 5, 14], we train for 60k iterations in case of ImageNet pre-training. In case of COCO pre-training, we train for 10k iterations, to avoid overfitting. Following state-of-the-art panoptic segmentation implementations on COCO [10], during training, we apply a random horizontal flip, followed by large-scale jittering with a scale between 0.1 and 2.0 and a random crop of 1024×1024 pixels. During inference, we resize the image such that the shortest side is 800 pixels.

For experiments on *Cityscapes-PP* [3, 4], we train for 90k iterations for both ImageNet and COCO pre-training. We follow the conventional data augmentation steps for Cityscapes during training [2, 8]: random horizontal flip with a probability of 0.5, scaling the image with a random factor between 0.5 and 2.0, and finally a random crop of 512×1024 pixels. During inference, we feed the full-resolution images of 1024×2048 pixels.

3.1.2 TAPPS

For TAPPS, we use $N^{\mathrm{q}} = 100$ shared queries. This is equal to the default number of object-level queries used by Mask2Former, because each shared query still represents only one object-level segment. Following Mask2Former, query embedding dimension $E = 256$. By default, the adaptation layer in the JOPS head is an MLP with two fully connected layers with 256 input and output channels and a ReLU activation in between; see Tab. 4 of the main manuscript for ablations.

During inference, TAPPS outputs PPS predictions without requiring rule-based post-processing, as it does not have to assign object-instance-unaware parts to individual ob-

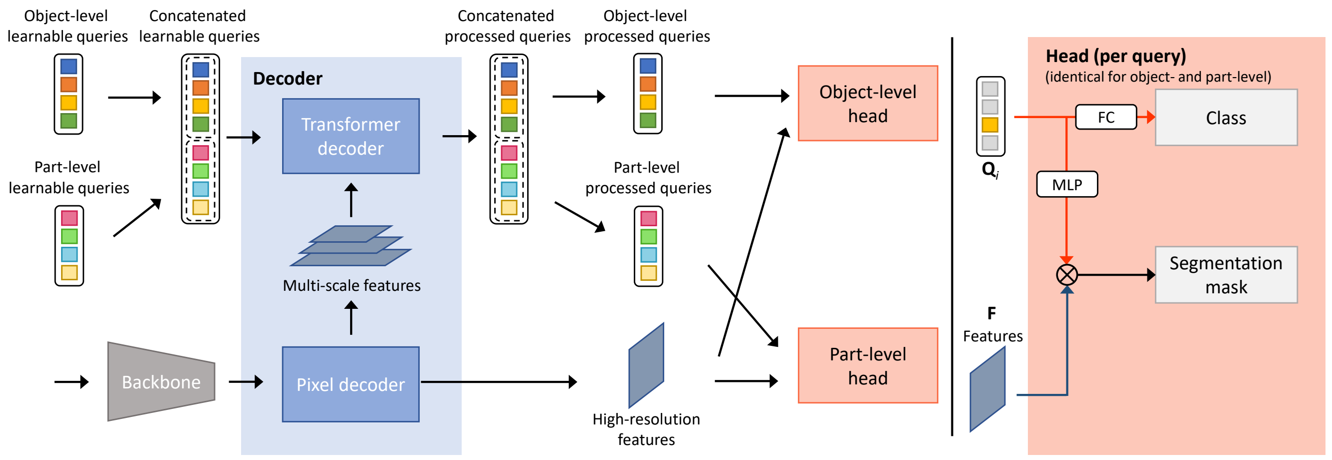

Object-level learnable queries
Concatenated learnable queries
Part-level learnable queries
Decoder
Transformer decoder
Multi-scale features
Backbone
Pixel decoder
Concatenated processed queries
Object-level processed queries
Part-level processed queries
High-resolution features
Object-level head
Part-level head
$\mathbf{Q}_i$
F
Features
Head (per query)
(identical for object- and part-level)
FC
Class
MLP
Segmentation mask

Figure 1. **Baseline network architecture.** Our strong baseline uses two separate sets of queries, one set for object-level segmentation and another set for part-level segmentation. Using these two sets of queries, this baseline network separately predicts object-level segments and object-unaware part-level segments. Operator $\otimes$ denotes a matrix multiplication.

jects, or resolve conflicts between object- and part-level predictions. Specifically, for each query, TAPPS simply outputs (a) an object-level class and mask, and (b) a part-level mask for each part-level class that is compatible with the predicted object-level class. For each pixel within the object-level mask, TAPPS keeps only the highest-scoring part mask prediction, applying `argmax`. This results in a set of compatible part-level segments that belong to an individual object-level segment. Applying this to all shared queries, we output a set of predictions that comply with the PPS task definition.

3.1.3 Baseline

Like Mask2Former, our baseline uses 100 queries for object-level segmentation. Additionally, to also conduct part-level semantic segmentation, it uses 100 additional queries. In Fig. 1, we depict the network architecture for this strong baseline. As seen in this figure, both sets of queries are concatenated when entering the Transformer decoder, so there can be interaction between object-level and part-level queries through self-attention. Note that, although they are concatenated, these queries are not 'mixed' or shared. The first 100 queries are still object-level queries, which learn to represent object-level segments, and the final 100 queries are part-level queries, which learn to represent object-instance-unaware part-level segments. At the end of the decoder, the queries are again split into two sets of queries, and fed into separate heads. The object-level head predicts the object-level segmentation masks and classes, resulting in object-level panoptic segmentation predictions. The part-level head predicts the part-level segmentation masks and classes, resulting in object-instance-unaware part segmentation predictions, *i.e.*, semantic segmentation for part classes.

During training, we apply the cross-entropy loss to the object-level and part-level classes, and we use both the cross-entropy and Dice loss [13] for the object-level and part-level segmentation masks. The total loss is the sum of the object-level mask and classification losses, and the part-level mask and classification losses.

During inference, because the part-level predictions are object-instance-unaware and not explicitly linked to individual objects, and because there can be incompatibilities between object- and part-level predictions, we apply the default rule-based merging strategy [4] used by existing work to generate the final PPS predictions.

3.1.4 Dynamic part segmentation

In Tab. 7 of the main manuscript, we compare our default version of TAPPS to a version that applies *dynamic* part segmentation. By default, as explained in Sec. 3.3.2 of the main manuscript, we generate a set of *fixed* per-object part queries in the JOPS head. That means that each per-object part query corresponds to a fixed, pre-determined part-level class, and that this query predicts a mask for this class. Alternatively, we can use a set of dynamic queries, which do not correspond to a fixed class, like we do for object-level segmentation. As depicted in Fig. 2, for each query $\mathbf{Q}_i$, we apply N^{dyn} fully-connected (FC) layers to generate a set of dynamic per-object part queries $\mathbf{Q}_i^{\text{dyn}} \in \mathbb{R}^{N^{\text{dyn}} \times E}$. As these queries are dynamic, they do not correspond to a fixed class, so for each of these queries we predict (a) a part-level class with a single fully-connected layer, and (b) a part-level segmentation mask by first applying a 3-layer MLP and then taking the product of the resulting mask queries with the features $\mathbf{F}$. We use $N^{\text{dyn}} = 50$.

Method	Backbone	Pre-training	As originally reported [8, 9]			With official evaluation [4]		
			PartPQ$^{\text{Pt}}$	PartPQ$^{\text{NoPt}}$	PartPQ	PartPQ$^{\text{Pt}}$	PartPQ$^{\text{NoPt}}$	PartPQ
Panoptic-PartFormer [8]	ResNet-50 [6]	I,C	–	–	37.8	56.1	38.8	43.2
Panoptic-PartFormer++ [9]	ResNet-50 [6]	I,C	–	–	42.2	52.6	42.6	45.1
Panoptic-PartFormer [8]	Swin-B [11]	I,C	–	–	47.4	64.3	50.6	54.1
Panoptic-PartFormer++ [9]	Swin-B [11]	I,C	–	–	49.3	48.9	52.1	51.3

Table 4. **Re-evaluation of existing work on Pascal-PP [1, 4, 5, 14].** After discovering an evaluation bug in the official code of Panoptic-PartFormer [8] which caused the PartPQ scores to be lower than they actually are, we re-evaluate the predictions by Panoptic-PartFormer [8] and Panoptic-PartFormer++ [9] using the official PPS evaluation repository [4]. We use these *higher* correct numbers in our comparisons in the main manuscript. I = ImageNet [15], C = COCO panoptic [10] pre-training.

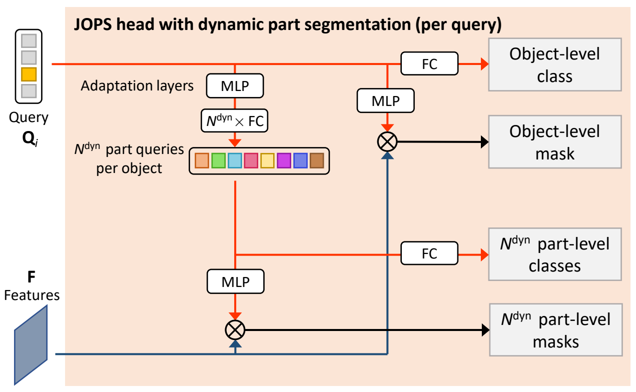

JOPS head with dynamic part segmentation (per query)
Query
$\mathbf{Q}_i$
Adaptation layers
MLP
$N^{\text{dyn}} \times$ FC
N^{dyn} part queries per object
FC
MLP
Object-level class
Object-level mask
FC
N^{dyn} part-level classes
F
Features
MLP
N^{dyn} part-level masks

Figure 2. **Dynamic part segmentation.** When conducting dynamic part segmentation, the JOPS head uses N^{dyn} fully-connected (FC) layers to generate N^{dyn} per-object part queries. Each per-object part query dynamically learns to represent at most one part-level segment within an object. For each per-object part query, we predict (a) a part-level class and (b) a part-level mask.

To supervise these part-level predictions during training, we assign each per-object part query to at most one part-level ground-truth segment using the same Hungarian matching algorithm we use for object-level segmentation [2]. This matching is applied separately within each object-level segment. If there is no matching ground-truth segment for a part query, we do not supervise the segmentation mask and supervise a 'no-part' class label. The part class prediction by this dynamic version of TAPPS is supervised with a cross-entropy loss. The other losses remain the same.

During inference, the procedure is the same as for the default TAPPS. The only difference is the source of the part-level class prediction. This *dynamic* version explicitly predicts it, whereas, for the default *fixed* version, it is known because each per-object part query is associated with a predetermined part class.

3.2. Evaluation of existing work

As mentioned in Tab. 2 of the main manuscript, the PartPQ scores of Panoptic-PartFormer [8] and its extension Panoptic-PartFormer++ [9] on Pascal-PP reported in our work are higher than the scores that these works originally reported [8, 9]. This is due to an evaluation bug that we discovered in the official code repository of Panoptic-PartFormer [8], which caused the resulting PartPQ scores to be lower than they actually are. Note that this bug only applies to Pascal-PP and not to Cityscapes-PP. We notified the authors of this bug, and they confirmed it. To assess whether this problem also occurred for Panoptic-PartFormer++, for which the code is not available, we requested the authors to send us the predictions by Panoptic-PartFormer++, so we could re-evaluate them on Pascal-PP. We are thankful that the authors have sent us these predictions. In Tab. 4, we provide both the originally reported scores, and the scores after our re-evaluation given the official PPS evaluation repository [4]. For all methods, the overall PartPQ scores are higher when using the correct evaluation. Therefore, we compare against these higher values in Tab. 2 of the main manuscript.

4. Qualitative results

In Fig. 3 and Fig. 4, we compare TAPPS against the strong baseline that we describe in Sec. 3.1.3. These examples show some of the advantages of TAPPS over the baseline. Specifically, we observe that TAPPS (1) makes more accurate part segmentation predictions within identified objects, and (2) is better able to separate different object instances.

Comparing TAPPS to state-of-the-art existing model Panoptic-PartFormer [8] in Fig. 5 and Fig. 6, we observe even more significant differences. In addition to the object-level segmentation quality, the part segmentation quality within objects is considerably better for TAPPS. This applies to large and small objects across different classes.

In Fig. 7 and Fig. 8, we show examples of predictions by TAPPS with a Swin-B [11] backbone, which achieves new state-of-the-art PPS performance. These examples show the high segmentation quality that TAPPS can achieve, across different types of objects and classes.

Finally, Fig. 9 shows examples of typical errors made by TAPPS. Notably, TAPPS struggles with images in which objects are seen from uncommon perspectives, and images with many objects and complex occlusions.

References

[1] Xianjie Chen, Roozbeh Mottaghi, Xiaobai Liu, Sanja Fidler, Raquel Urtasun, and Alan Yuille. Detect What You Can: Detecting and Representing Objects using Holistic Models and Body Parts. In *CVPR*, 2014. 1, 2, 4, 6, 8, 10, 12

[2] Bowen Cheng, Ishan Misra, Alexander G Schwing, Alexander Kirillov, and Rohit Girdhar. Masked-attention Mask Transformer for Universal Image Segmentation. In *CVPR*, 2022. 2, 4

[3] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The Cityscapes Dataset for Semantic Urban Scene Understanding. In *CVPR*, 2016. 2, 7, 9, 11, 12

[4] Daan de Geus, Panagiotis Meletis, Chenyang Lu, Xiaoxiao Wen, and Gijs Dubbelman. Part-aware Panoptic Segmentation. In *CVPR*, 2021. 1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12

[5] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The Pascal Visual Object Classes (VOC) Challenge. *IJCV*, 88(2):303–338, 2010. 1, 2, 4, 6, 8, 10, 12

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *CVPR*, 2016. 4, 6, 7, 8, 9, 12

[7] Alexander Kirillov, Kaiming He, Ross Girshick, Carsten Rother, and Piotr Dollar. Panoptic Segmentation. In *CVPR*, 2019. 2

[8] Xiangtai Li, Shilin Xu, Yibo Yang, Guangliang Cheng, Yunhai Tong, and Dacheng Tao. Panoptic-PartFormer: Learning a Unified Model for Panoptic Part Segmentation. In *ECCV*, 2022. 1, 2, 4, 8, 9

[9] Xiangtai Li, Shilin Xu, Yibo Yang, Haobo Yuan, Guangliang Cheng, Yunhai Tong, Zhouchen Lin, and Dacheng Tao. PanopticPartFormer++: A Unified and Decoupled View for Panoptic Part Segmentation. *arXiv preprint arXiv:2301.00954*, 2023. 1, 2, 4

[10] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common Objects in Context. In *ECCV*, 2014. 1, 2, 4, 6, 7, 8, 9, 10, 11, 12

[11] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin Transformer: Hierarchical Vision Transformer Using Shifted Windows. In *ICCV*, 2021. 4, 10, 11

[12] Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *ICLR*, 2019. 2

[13] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentatio. In *3DV*, 2016. 3

[14] Roozbeh Mottaghi, Xianjie Chen, Xiaobai Liu, Nam-Gyu Cho, Seong-Whan Lee, Sanja Fidler, Raquel Urtasun, and Alan Yuille. The Role of Context for Object Detection and Semantic Segmentation in the Wild. In *CVPR*, 2014. 1, 2, 4, 6, 8, 10, 12

[15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 115(3):211–252, 2015. 2, 4

[16] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. `https://github.com/facebookresearch/detectron2`, 2019. 2

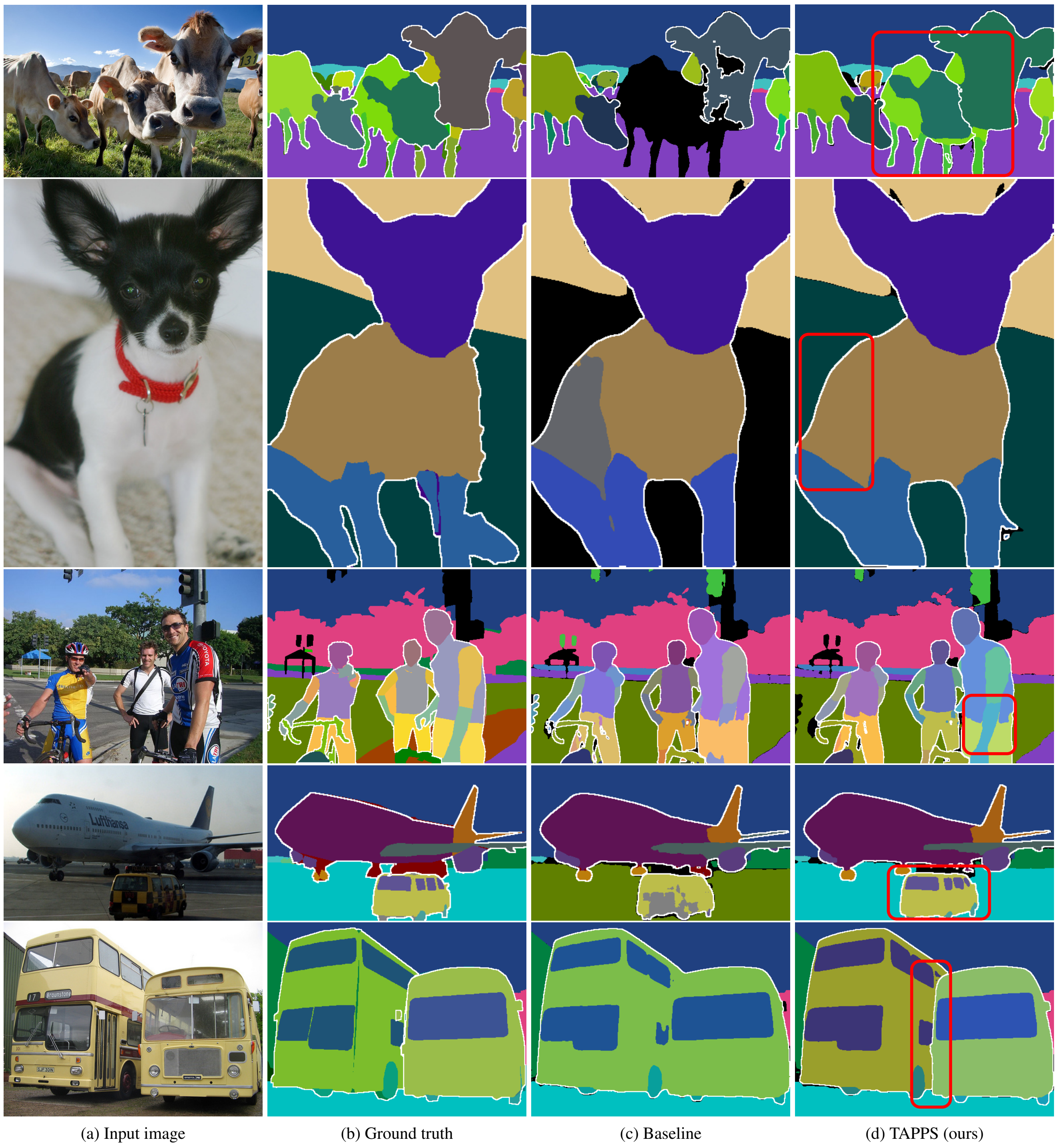
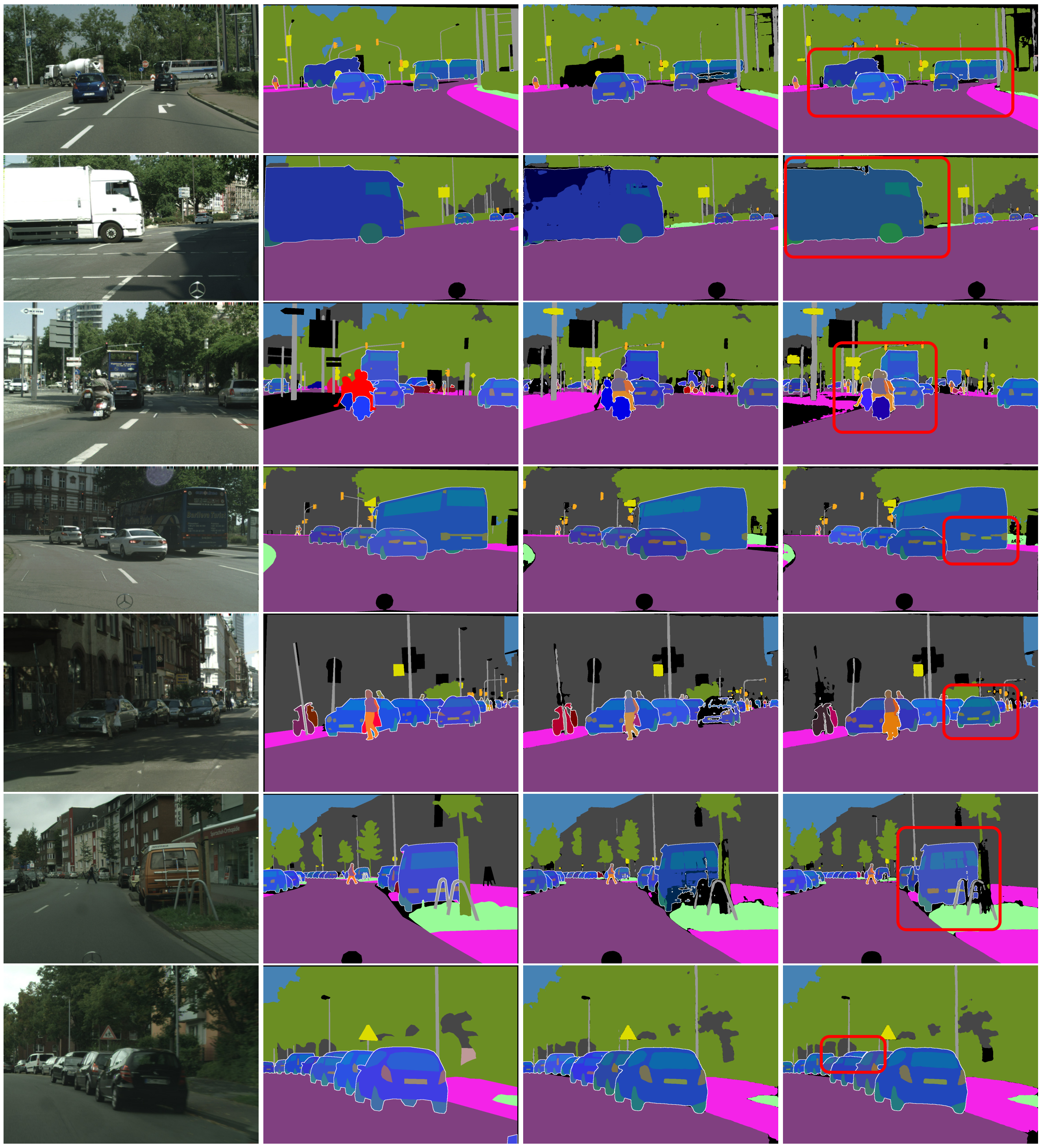

(a) Input image (b) Ground truth (c) Baseline (d) TAPPS (ours)

Figure 3. **Qualitative examples of TAPPS and our strong baseline on Pascal-PP [1, 4, 5, 14].** Both networks use ResNet-50 [6] with COCO pre-training [10]. White borders separate different object-level instances; color shades indicate different categories. Note that the colors of part-level categories are not identical across instances; there are different shades of the same color. In these examples, we can see how TAPPS improves both the instance separability and part segmentation quality with respect to the strong baseline. The red boxes indicate regions in which these differences are best visible. Best viewed digitally.



(a) Input image (b) Ground truth (c) Baseline (d) TAPPS (ours)

Figure 4. **Qualitative examples of TAPPS and our strong baseline on Cityscapes-PP [3, 4].** Both networks use ResNet-50 [6] with COCO pre-training [10]. White borders separate different object-level instances; color shades indicate different categories. Note that the colors of part-level categories are not identical across instances; there are different shades of the same color. In these examples, we can see how TAPPS improves both the instance separability and part segmentation quality with respect to the strong baseline. The red boxes indicate regions in which these differences are best visible. Best viewed digitally.

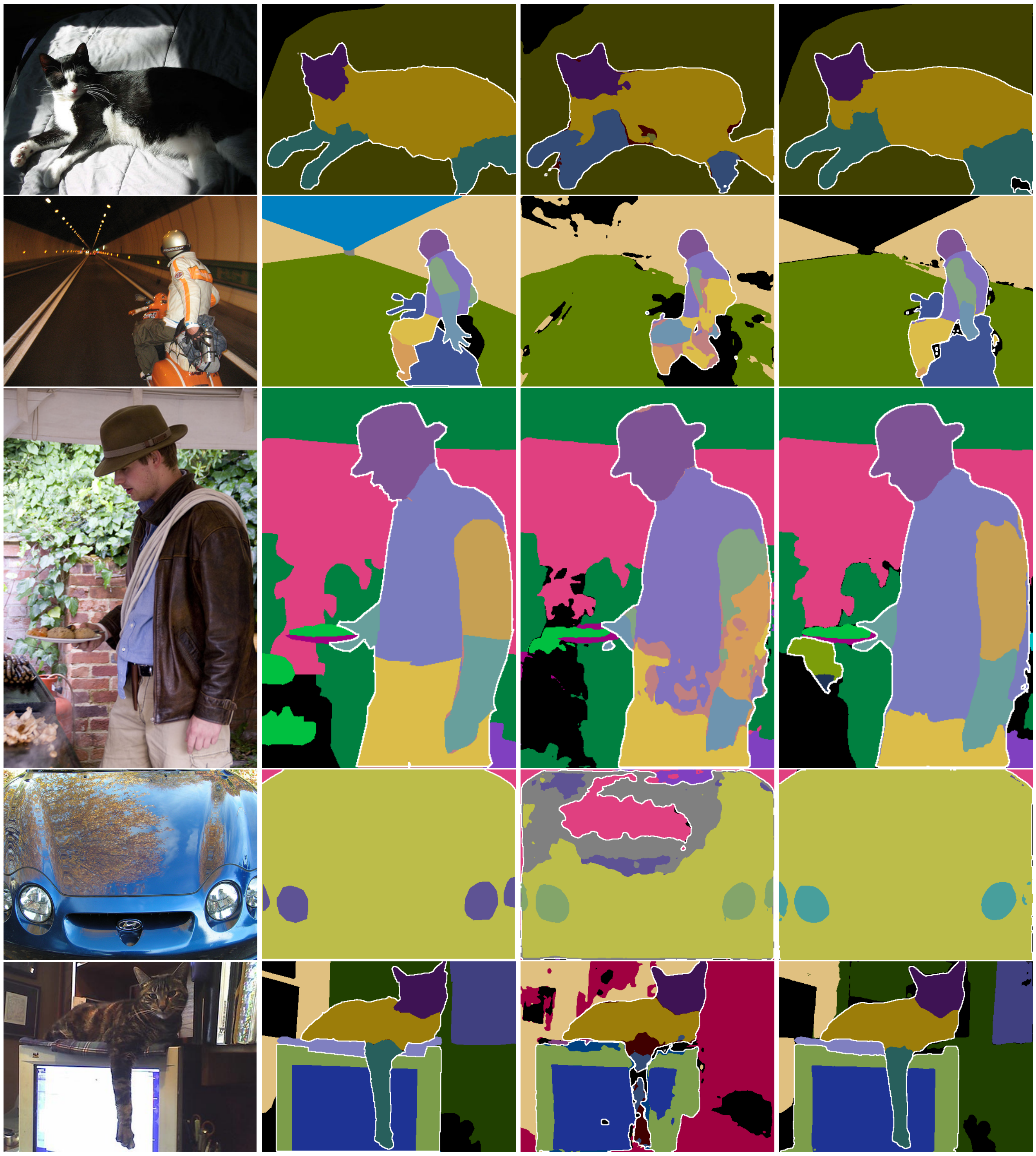


Figure 5. **Qualitative examples of TAPPS and Panoptic-PartFormer [8] on Pascal-PP [1, 4, 5, 14].** Both networks use ResNet-50 [6] with COCO pre-training [10]. White borders separate different object-level instances; color shades indicate different categories. Note that the colors of part-level categories are not identical across instances; there are different shades of the same color. Best viewed digitally.

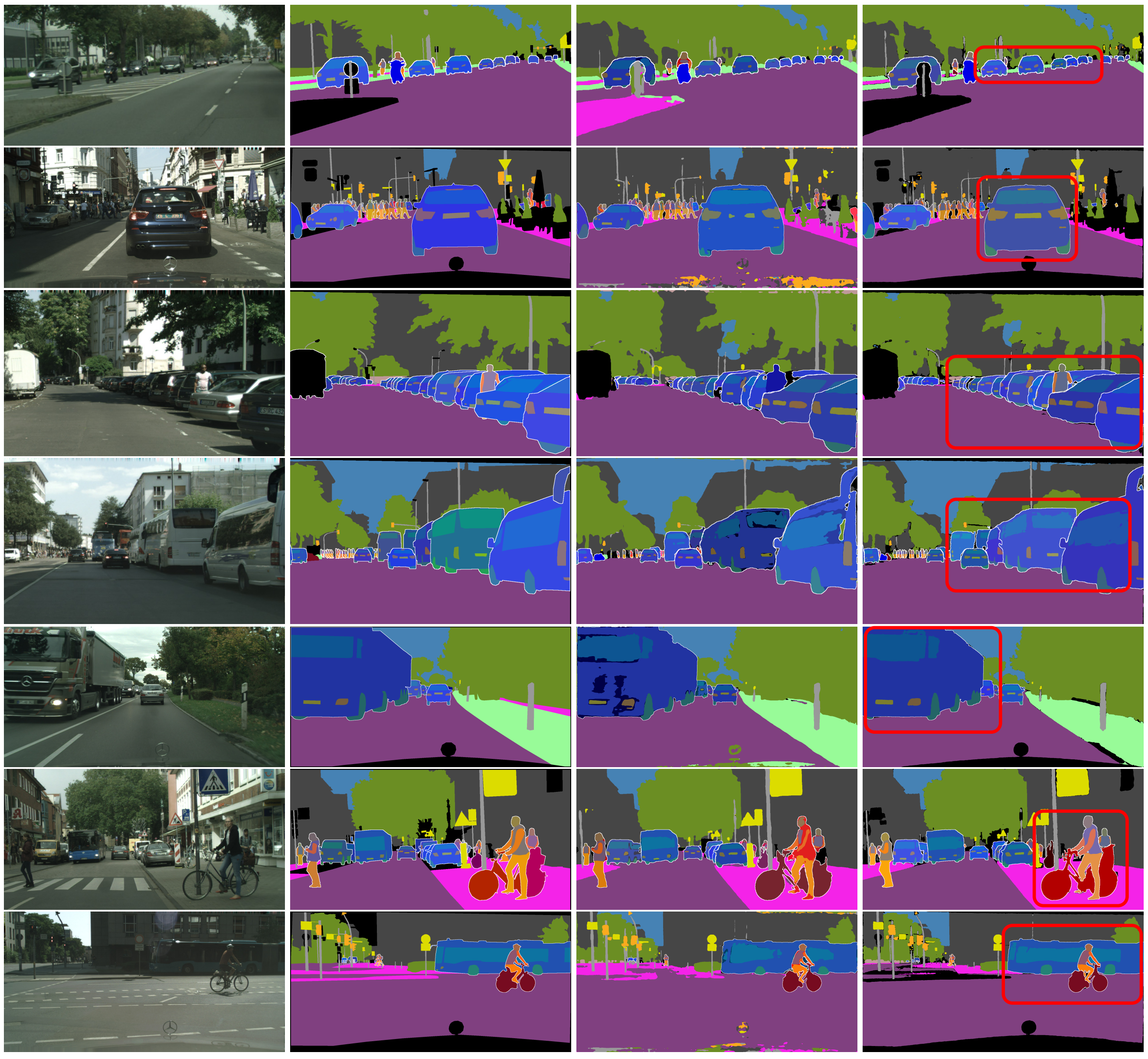

(a) Input image (b) Ground truth (c) Panoptic-PartFormer [8] (d) TAPPS (ours)

Figure 6. **Qualitative examples of TAPPS and Panoptic-PartFormer [8] on Cityscapes-PP [3, 4].** Both networks use ResNet-50 [6] with COCO pre-training [10]. White borders separate different object-level instances; color shades indicate different categories. Note that the colors of part-level categories are not identical across instances; there are different shades of the same color. Best viewed digitally.

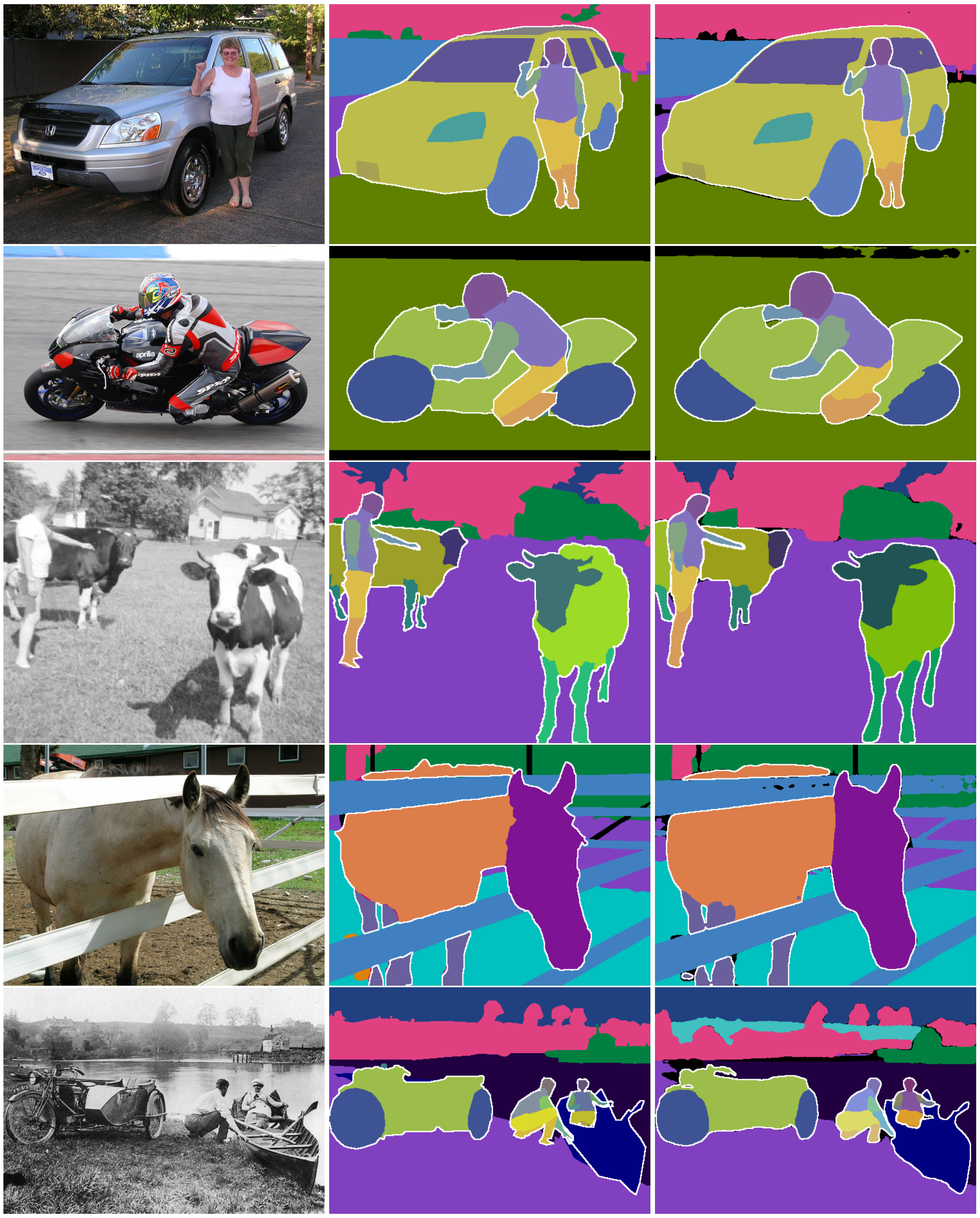


Figure 7. **TAPPS with Swin-B [11] on Pascal-PP [1, 4, 5, 14].** The Swin-B backbone is pre-trained on COCO panoptic [10]. White borders separate different object-level instances; color shades indicate different categories. Note that the colors of part-level categories are not identical across instances; there are different shades of the same color. Best viewed digitally.

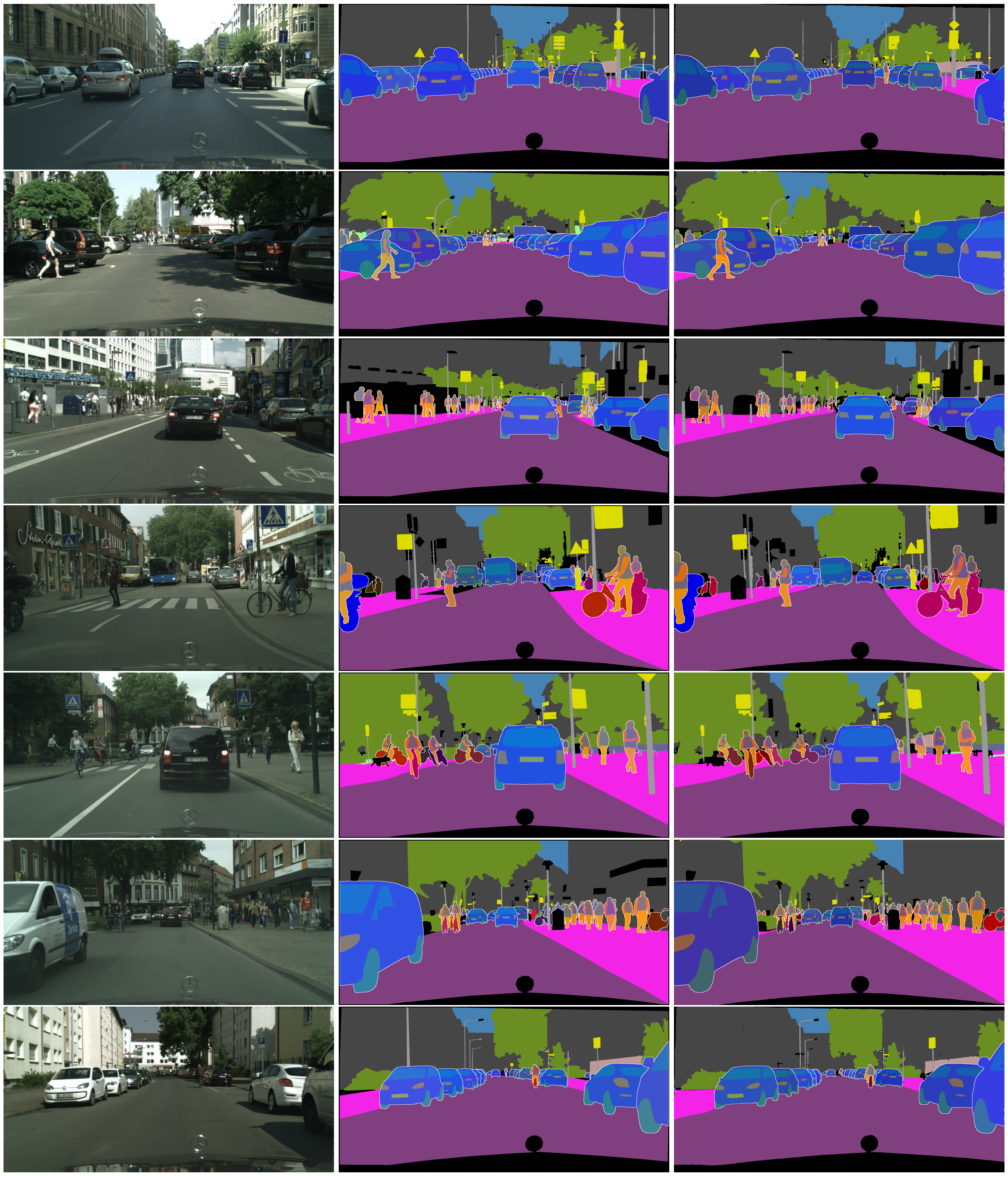

(a) Input image (b) Ground truth (c) TAPPS (ours)

Figure 8. **TAPPS with Swin-B [11] on Cityscapes-PP [3, 4].** The Swin-B backbone is pre-trained on COCO panoptic [10]. White borders separate different object-level instances; color shades indicate different categories. Note that the colors of part-level categories are not identical across instances; there are different shades of the same color. Best viewed digitally.

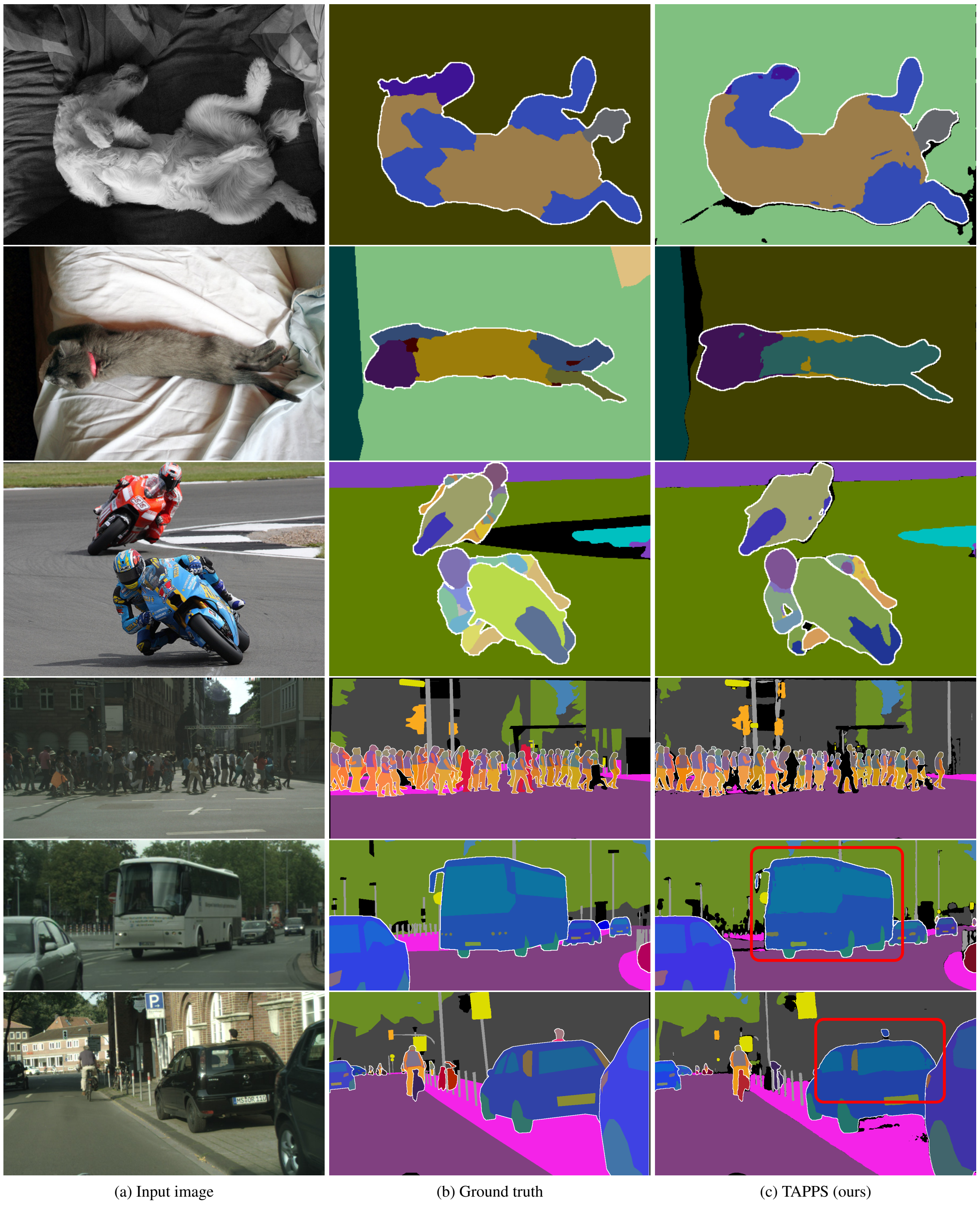


Figure 9. **Examples of errors in TAPPS predictions.** The predictions are made by TAPPS that uses a ResNet-50 [6] backbone pre-trained on COCO panoptic [10]. Top three images are from Pascal-PP validation [1, 4, 5, 14], bottom three images are from Cityscapes-PP val [3, 4]. White borders separate different object-level instances; color shades indicate different categories. Note that the colors of part-level categories are not identical across instances; there are different shades of the same color. Best viewed digitally.