# Panoptic-DeepLab in PyTorch

Bowen Cheng[1], Maxwell D. Collins[2], Yukun Zhu[2], Ting Liu[2], Liang-Chieh Chen[2]

[1]UIUC [2]Google Research

## Abstract

*In this technical report, we present a PyTorch reimplementation of Panoptic-DeepLab [8, 9]. We further introduce technical details that are not mentioned in any paper, to let everyone be able to reproduce state-of-the-art results with limited resources (4 or 8 GPUs vs 32 TPUs in [8, 9]). We also analyze Panoptic-DeepLab and find center prediction is almost perfect and the bottleneck of bottom-up method still lies in semantic segmentation. Code and models are available at* https://github.com/bowenc0221/panoptic-deeplab.

## 1. Introduction

Panoptic segmentation [18] has received a lot of attention from the community. We observe an interesting phenomenon that most top-performing panoptic segmentation methods [17, 28] are based on top-down Mask R-CNN [14]. This is probably due to most open-sourced PyTorch toolboxes [27, 2] for instance and panoptic segmentation are based on Mask R-CNN [14] and researchers adopt this two-stage paradigm because their codes are easily accessible.

Recently, there is an increasing number of works using bottom-up approaches [26, 12, 19], but there still exists some gap between top-down methods. Panoptic-DeepLab [8, 9] closes this gap with a simple but extremely effective bottom-up paradigm and even surpasses top-down methods by a large margin on some datasets (*e.g.* Cityscapes [11]). However, the original Panoptic-DeepLab is implemented in TensorFlow [1] and there are also very few other bottom-up methods open-source their codes, making it hard for researchers using PyTorch [23] to find a suitable bottom-up baseline.

Towards the goal of providing a high-quality codebase for bottom-up panoptic and instance segmentation, we reimplement Panoptic-DeepLab in PyTorch. Apart from open-sourcing our PyTorch implementation, we further introduce the data structure we use for data pre-processing and output post-processing in detail. We hope these details could help the community to push forward the performance of bottom-up methods. Furthermore, our analysis suggests center prediction is almost perfect and the bottleneck lies in semantic segmentation.

## 2. Technical details

### 2.1. Data pre-processing details

We first introduce how data is processed for training. An input image and label pair will go through three steps: (1) pre-augmentation transform, (2) data augmentation, and (3) post-augmentation transform.

#### 2.1.1 Data structure

Different from top-down methods that have standard data pre-processing pipeline, bottom-up methods need to first define a good data structure (of labels) for efficient data pre-processing. Mask R-CNN based top-down methods store 'thing' instances in a list of binary masks and 'stuff' regions in a semantic map the same way as semantic segmentation (with 'thing' classes merged into a single class). As bottom-up methods usually have heavier data augmentation, storing instances in a list is not efficient.

In Panoptic-DeepLab, we adopt storing 'thing' instances and 'stuff' regions together using a single 'panoptic label' [18, 11] defined as follow, into a single semantic map:

$$panoptic = semantic \times label\_divisor + instance\_id \quad (1)$$

where $label\_divisor$ is an integer that is greater than the max possible $instance\_id$. $instance\_id$ for stuff regions is 0 and starting from 1 for thing instances.

In this way, we can directly apply the same data augmentation to a pair of image and panoptic label.

#### 2.1.2 Pre-augmentation transform

As its name suggests, pre-augmentation transform is performed before data augmentation. For some datasets like COCO [20] and Mapillary Vistas [21], images have various shapes. Thus, we first resize images to some fixed shape

| Framework | PQ (%) | SQ (%) | RQ (%) | AP (%) | mIoU (%) | Net speed (ms) | Post speed (ms) |
|---|---|---|---|---|---|---|---|
| TensorFlow (original [9]) | 59.8 | - | - | - | - | 117 (V100) | included ($\sim$ 2) |
| PyTorch (ours) | 59.0 | 80.0 | 72.4 | 24.5* | 79.1 | 126 (1080TI) | 107$^+$ |

Table 1. Comparison between our reimplementation and the original implementation [9] on Cityscapes *val* set, with ResNet-50 as backbone. The difference in PQ is probably because: (1) our reimplementation uses output stride 32, (2) our reimplementation uses smaller batchsize (4 vs 32), and (3) we use $7 \times 7$ convolution in stem while [9] uses two $3 \times 3$ convolutions in stem. *: we focus on reproducing the PQ metric and do not tune parameters for AP. $^+$: Our PyTorch implementation still uses 'for-loop' for merging, it could be optimized by merging in parallel [8, 9].

before data augmentation. It is done by resizing the longer side and padding the shorter side to make all images having the same size. For datasets like Cityscapes [11], images have the same size and we do not need pre-augmentation transform.

#### 2.1.3 Data augmentation

Since Panoptic-DeepLab is built upon DeepLab [4, 5, 6, 7], we use the same data augmentation with minor modification. Following the open-sourced DeepLab [1], we first apply a random scaling with scales sampled from 0.5 to 2.0 with a step size of 0.1 (which is different from the default 0.25 in DeepLab). Then, we randomly crop a region defined by a `crop_size`. If the `crop_size` is larger then the image, we perform a random padding. Unlike the padding in DeepLab that only pads the bottom and right part of the image, we find random padding is beneficial for instance branch. We believe this is because random padding can generates more diverse locations for objects. Lastly, we perform a random horizontal flip.

#### 2.1.4 Post-augmentation transform

The post-augmentation transform (or target transform) only applies to the panoptic label map to generate targets for Panoptic-DeepLab. Specifically, we need 6 targets: (1) semantic labels, (2) center heatmap, (3) offsets and their associated pixel-wise loss weights. These targets are generated as follow:

1. Semantic label: From Eq. (1), we can easily recover semantic label by `panoptic_label // label_divisor`, where `//` is the Python3 operator that only takes the integer part of the division.

2. Center heatmap: For each unique panoptic label, we can test it is 'thing' or 'stuff' by investigating its instance id, which is `panoptic_label % label_divisor`, `%` is the Python3 mod operator. For every instance mask, we calculates its centroid and apply a Gaussian centered at the centroid location. To combine Gaussian from the new instance with existing Gaussians, we simply take the maximum value.

3. Offset: This can be easily acquired by taking the difference between coordinate of current location with instance center it belongs to [22]. We do not learn offset for pixels belonging to 'stuff' regions.

4. Semantic loss weight: We set loss weights to 3 for pixels belonging to an instance with area less than 4096. Loss weights for other locations are set to 1 [29].

5. Center loss weight: We set loss weights to 1 for every pixel except pixels belonging to an 'ignore' region.

6. Offset loss weight: We set loss weights to 1 for every pixel belonging to a 'thing' class and 0 for other.

#### 2.1.5 Odd crop size

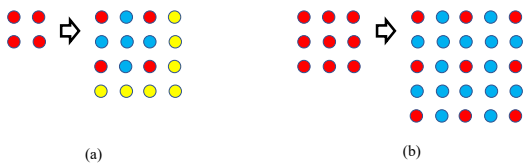


Figure 1. Effect of upsampling even-value or odd-value input size. (a) Upsampling a feature map with even-value resolution will cause unaligned boundary (yellow). (b) Upsampling a feature map with odd-value resolution resolves unaligned boundary.

Using odd-value input size (e.g., $1025 \times 2049$ instead of $1024 \times 2048$) is first proposed in the very first version of DeepLab [4] to align features between different spatial resolutions. For example, with an input size $1025 \times 2049$, the intermediate feature map resolutions are $513 \times 1025$, $257 \times 513$, $129 \times 257$, and $65 \times 129$. When performing bilinear upsampling on a feature map with odd-value input size, the corner features between two resolutions will be matched, while the remaining features will be bilinearly interpolated. We illustrate this effect of feature alignment in Fig. 1, and refer to the open source DeepLab library.

[1] https://github.com/tensorflow/models/tree/master/research/deeplab

| GT Semantic | GT Foreground | GT Center | GT Offset | PQ (%) | SQ (%) | RQ (%) | AP (%) | mIoU (%) |
|---|---|---|---|---|---|---|---|---|
| | | | | 59.0 | 80.0 | 72.4 | 24.5 | 79.1 |
| ✓ | | | | 78.1 | 91.5 | 84.4 | 38.4 | 100.0 |
| | ✓ | | | 62.0 | 82.3 | 74.4 | 35.0 | 79.1 |
| | | ✓ | | 59.9 | 80.1 | 73.5 | 27.0 | 79.1 |
| | | | ✓ | 62.9 | 83.6 | 74.4 | 42.2 | 79.1 |
| ✓ | ✓ | | | 80.6 | 93.3 | 85.6 | 42.0 | 100.0 |
| ✓ | | ✓ | | 79.7 | 91.7 | 86.1 | 44.0 | 100.0 |
| ✓ | | | ✓ | 87.3 | 95.5 | 91.3 | 68.1 | 100.0 |
| | ✓ | ✓ | | 63.2 | 82.5 | 75.7 | 39.7 | 79.1 |
| | ✓ | | ✓ | 69.3 | 86.5 | 79.4 | 59.5 | 79.1 |
| | | ✓ | ✓ | 65.9 | 84.0 | 77.7 | 46.5 | 79.1 |
| | ✓ | ✓ | ✓ | 71.1 | 87.5 | 80.5 | 70.1 | 79.1 |
| ✓ | | ✓ | ✓ | 90.1 | 96.7 | 93.1 | 83.5 | 100.0 |
| ✓ | ✓ | | ✓ | 91.6 | 98.0 | 93.5 | 77.4 | 100.0 |
| ✓ | ✓ | ✓ | | 82.6 | 93.8 | 87.6 | 49.4 | 100.0 |
| ✓ | ✓ | ✓ | ✓ | 96.5 | 100.0 | 96.5 | 100.0 | 100.0 |
| ✓ | ✓ | ✓ | ✓ | 100.0* | 100.0 | 100.0 | 100.0 | 100.0 |

Table 2. Oracle experiments on Cityscapes *val* set, with ResNet-50 as bakcbone. **GT Semantic**: Use ground truth semantic prediction. **GT Foreground**: Use ground truth foreground prediction. **GT Center**: Use ground truth center heatmap prediction. **GT Offset**: Use ground truth offset prediction. *: Disable stuff area threshold, setting stuff area threshold to 4096 makes PQ less than 100% when replacing all predictions with ground truth.

## 2.2. Training details

Parameters for training are discussed in [8, 9]. Here we discuss some details that may be easily ignored.

### 2.2.1 Batch size, iteration and learning rate schedule

Since the 'poly' learning rate schedule [5] is widely used for semantic segmentation, we also adopt this schedule. However, the 'linear scaling rule' [13] may not hold for this learning rate schedule as some constraints for approximation in [13] does not hold. Currently we do not know the correct way to scale learning rate and iteration based on batch size yet.

### 2.2.2 Synchronizd BatchNorm

Some researches [25, 24] show that training with large mini-batch size and finetune the BatchNorm [16] layer could improve performance. In our experiments, we find that if the input crop size is large enough, there is not need to use large batch size. In our experiments, we use a batch size of 4 but a crop size of $1025 \times 2049$ and we do not see any instability in finetuning the BatchNorm layer.

## 2.3. Post-processing details

The post-processing for Panoptic-DeepLab is extremely simple, which only consists of thress steps: (1) find center points, (2) assign an instance id to every pixel, and (3) merge semantic and instance prediction.

### 2.3.1 Find center (point NMS)

Point NMS has been widely used in keypoint detection [30, 10]. We apply a $k \times k$ max pooling with stride 1 and compare values before and after max pooling. Center points are position where values are the same before and after max pooling.

### 2.3.2 Assign instance id

We first calculates the predicted center points of every pixel location by adding the offset with its coordinate, resulting in a $H \times W \times 2$ tensor. Then, we reshape tensor to $HW \times 2$ and the center points tensor is also reshaped to $K \times 2$. Next, both tensors are broadcast to $HW \times K \times 2$ and we calculate the $l2$-norm between two tensors on the third axis, resulting in a tensor of shape $HW \times K$. The instance id for every pixel is acquired by simply taking the argmax along the second axis (plus 1, because 0 is reserved for 'stuff'). Finally, we set the instance id to 0 for pixels belonging to 'stuff' class

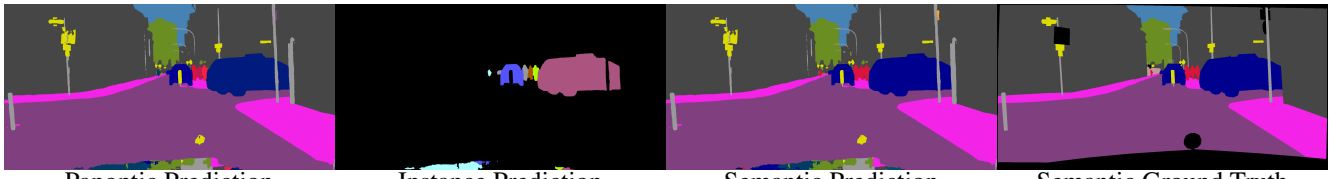


Figure 2. Visualization of Panoptic-DeepLab with ResNet-50 on Cityscapes *val* set. Showing problems when handling VOID regions. For the semantic segmentation task, the comparison between prediction and ground truth is done pixel-wise, thus VOID pixels are ignored for prediction. In instance and panoptic segmentation, VOID pixels could be associated with other instances in Panoptic-DeepLab, which degrades the IoU for that instance in evaluation.

from the semantic segmentation output.

#### 2.3.3 Merging semantic and instance prediction

Merging is done by a straightforward majority voting. For every instance, we take the majority of thing classes within this region and generate panoptic label by (1). To avoid 'overflow', we set individual instance id counters for different thing classes (*i.e*. instance id always starts from 1 for every 'thing' class). In our PyTorch implementation, we did not optimize majority voting as [8, 9].

## 3. Analysis of Panoptic-DeepLab

### 3.1. Reproduced results

Tab. 1 shows comparison of original Panoptic-DeepLab and our reimplementation on Cityscapes [11] val set with ResNet-50 [15] backbone. Our reproduced result is 0.8 PQ lower than the original one. There are several factors:

1. Our ResNet-50 encoder has output stride 32, where as [9] uses output stride 16.
2. We train the model use smaller batch size (4 vs 32 in [9]).
3. We use $7 \times 7$ convolution in stem while [9] uses two $3 \times 3$ convolutions in stem.

### 3.2. Oracle results

We further perform an oracle experiment by replacing different prediction with ground truth in Tab. 2. This oracle experiment provides us with information on how well each component is doing. From these results, we can see that **center points prediction is doing well and the bottleneck of Panoptic-DeepLab is still on semantic segmentation**.

## 4. Problem with 'egocar' region

We notice that the 'ignore' region in semantic segmentation might be problematic in Panoptic-DeepLab. Fig. 2 demonstrates the problem. Some parts of the 'egocar' region (ignored in Cityscapes) are grouped with instances. The cause of this problem is we do not train semantic segmentation branch for 'ignore' region, so that the behavior is random in those regions. It won't cause any problem in semantic segmentation as semantic segmentation performs pixel-to-pixel comparison. Thus, those regions are ignored. On the other hand, instance segmentation performs a instance-to-instance comparison. Including this 'ignore' region will lower the IoU and thus making true positive a false positive. Recently, Chen *et al*. [3] has identified this problem and find it harmful in semi-supervised learning. [3] solves it by removing 'egocar' region from human annotation. A potential solution in our case is to add a foreground segmentation branch the predicts 'egocar' region to be background.

## Disclaimer

The propose of open sourcing this PyTorch implementation of Panoptic-DeepLab is to share more technical details on implementing bottom-up panoptic segmentation methods. We do not guarantee to reproduce all numbers in [9]. Please compare with original numbers in [9] when making comparison.

## References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 1

[2] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang,

Chen Change Loy, and Dahua Lin. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019. 1

[3] Liang-Chieh Chen, Raphael Gontijo Lopes, Bowen Cheng, Maxwell D Collins, Ekin D Cubuk, Barret Zoph, Hartwig Adam, and Jonathon Shlens. Semi-supervised learning in video sequences for urban scene segmentation. *arXiv:2005.10266*, 2020. 4

[4] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. In *ICLR*, 2015. 2

[5] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE TPAMI*, 2017. 2, 3

[6] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv:1706.05587*, 2017. 2

[7] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *ECCV*, 2018. 2

[8] Bowen Cheng, Maxwell D Collins, Yukun Zhu, Ting Liu, Thomas S Huang, Hartwig Adam, and Liang-Chieh Chen. Panoptic-deeplab. In *ICCV COCO + Mapillary Joint Recognition Challenge Workshop*, 2019. 1, 2, 3, 4

[9] Bowen Cheng, Maxwell D Collins, Yukun Zhu, Ting Liu, Thomas S Huang, Hartwig Adam, and Liang-Chieh Chen. Panoptic-deeplab: A simple, strong, and fast baseline for bottom-up panoptic segmentation. In *CVPR*, 2020. 1, 2, 3, 4

[10] Bowen Cheng, Bin Xiao, Jingdong Wang, Honghui Shi, Thomas S Huang, and Lei Zhang. Higherhrnet: Scale-aware representation learning for bottom-up human pose estimation. In *CVPR*, 2020. 3

[11] 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. 1, 2, 4

[12] Naiyu Gao, Yanhu Shan, Yupei Wang, Xin Zhao, Yinan Yu, Ming Yang, and Kaiqi Huang. Ssap: Single-shot instance segmentation with affinity pyramid. In *ICCV*, 2019. 1

[13] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017. 3

[14] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *ICCV*, 2017. 1

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 4

[16] Sergey Ioffe and Christian Szegedy. Batch normalization: accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 3

[17] Alexander Kirillov, Ross Girshick, Kaiming He, and Piotr Dollár. Panoptic feature pyramid networks. In *CVPR*, 2019. 1

[18] Alexander Kirillov, Kaiming He, Ross Girshick, Carsten Rother, and Piotr Dollár. Panoptic segmentation. In *CVPR*, 2019. 1

[19] Qizhu Li, Xiaojuan Qi, and Philip HS Torr. Unifying training and inference for panoptic segmentation. In *CVPR*, 2020. 1

[20] 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

[21] Gerhard Neuhold, Tobias Ollmann, Samuel Rota Bulò, and Peter Kontschieder. The mapillary vistas dataset for semantic understanding of street scenes. In *ICCV*, 2017. 1

[22] George Papandreou, Tyler Zhu, Liang-Chieh Chen, Spyros Gidaris, Jonathan Tompson, and Kevin Murphy. Personlab: Person pose estimation and instance segmentation with a bottom-up, part-based, geometric embedding model. In *ECCV*, 2018. 2

[23] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017. 1

[24] Chao Peng, Tete Xiao, Zeming Li, Yuning Jiang, Xiangyu Zhang, Kai Jia, Gang Yu, and Jian Sun. Megdet: A large mini-batch object detector. In *CVPR*, 2018. 3

[25] Samuel Rota Bulò, Lorenzo Porzi, and Peter Kontschieder. In-place activated batchnorm for memory-optimized training of dnns. In *CVPR*, 2018. 3

[26] Konstantin Sofiiuk, Olga Barinova, and Anton Konushin. Adaptis: Adaptive instance selection network. In *ICCV*, 2019. 1

[27] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019. 1

[28] Yuwen Xiong, Renjie Liao, Hengshuang Zhao, Rui Hu, Min Bai, Ersin Yumer, and Raquel Urtasun. Upsnet: A unified panoptic segmentation network. In *CVPR*, 2019. 1

[29] Tien-Ju Yang, Maxwell D Collins, Yukun Zhu, Jyh-Jing Hwang, Ting Liu, Xiao Zhang, Vivienne Sze, George Papandreou, and Liang-Chieh Chen. Deeperlab: Single-shot image parser. *arXiv:1902.05093*, 2019. 2

[30] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. *arXiv:1904.07850*, 2019. 3