

# Towards Bounding-Box Free Panoptic Segmentation

Ujwal Bonde<sup>1</sup> Pablo F. Alcantarilla<sup>1</sup> Stefan Leutenegger<sup>1,2</sup>  
 firstname@slamcore.com

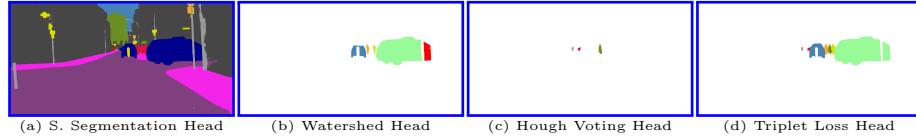
<sup>1</sup> SLAMcore Ltd. <sup>2</sup> Imperial College London

**Abstract.** In this work we introduce a new Bounding-Box Free Network (BBFNet) for panoptic segmentation. Panoptic segmentation is an ideal problem for proposal-free methods as it already requires per-pixel semantic class labels. We use this observation to exploit class boundaries from off-the-shelf semantic segmentation networks and refine them to predict instance labels. Towards this goal BBFNet predicts coarse watershed levels and uses them to detect large instance candidates where boundaries are well defined. For smaller instances, whose boundaries are less reliable, BBFNet also predicts instance centers by means of Hough voting followed by mean-shift to reliably detect small objects. A novel triplet loss network helps merging fragmented instances while refining boundary pixels. Our approach is distinct from previous works in panoptic segmentation that rely on a combination of a semantic segmentation network with a computationally costly instance segmentation network based on bounding box proposals, such as Mask R-CNN, to guide the prediction of instance labels using a Mixture-of-Expert (MoE) approach. We benchmark our proposal-free method on Cityscapes and Microsoft COCO datasets and show competitive performance with other MoE based approaches while outperforming existing non-proposal based methods on the COCO dataset. We show the flexibility of our method using different semantic segmentation backbones.

## 1 Introduction

Panoptic segmentation is the joint task of predicting semantic scene segmentation together with individual instances of objects present in the scene. Historically this has been explored under different umbrella terms of scene understanding [42] and scene parsing [36]. In [17], Kirillov *et al.* coined the term and gave a more concrete definition by including the suggestion from Forsyth *et al.* [10] of splitting the objects categories into *things* (countable objects like persons, cars, *etc.*) and *stuff* (uncountable like sky, road, *etc.*) classes. While *stuff* classes require only semantic label prediction, *things* need both the semantic and instance labels. Along with this definition, *Panoptic Quality* (PQ) measure was proposed to benchmark different methods. Since then, there has been a more focused effort towards panoptic segmentation with multiple datasets [7,24,25] supporting it.

Existing methods for panoptic segmentation can be broadly classified into two groups. The first group uses a proposal based approach for predicting *things*.



**Fig. 1.** BBFNet gradually refines the class boundaries of the semantic segmentation network to predict panoptic segmentation. The Watershed head detects candidates for large instances whereas the Hough voting head detects small object instances. The Triplet Loss network refines and merges the detection to obtain the final instance labels.

Traditionally these methods use completely separate instance and scene segmentation networks. Using a MoE approach, the outputs are combined either heuristically or through another sub-network. Although, more recent works propose sharing a common feature backbone for both networks [16,28], this split of tasks restricts the backbone network to the most complex branch. Usually this restriction is imposed by the instance segmentation branch, performed by Mask-RCNN [13].

The second group of work uses a proposal free approach for instance segmentation allowing for a more efficient design. An additional benefit of these methods is that they do not need bounding-box predictions. While bounding-box detection based approaches have been popular and successful, they require predicting auxiliary quantities like scale, width and height which do not directly contribute to instance segmentation. Furthermore, the choice of bounding-boxes for object-detection had been questioned in the past [30]. We believe panoptic segmentation to be an ideal problem for a bounding-box free approach since it already contains structured information from semantic segmentation.

In this work, we exploit this using a flexible panoptic segmentation head that can be added to any off-the-shelf semantic segmentation network. We coin this as *Bounding-Box Free Network* (BBFNet) which is a proposal free network and predicts *things* by gradually refining the class boundaries predicted by the base network. To achieve this we exploit previous works in non-proposal based methods for instance segmentation [2,4,27]. Based on the output of a semantic segmentation network, BBFNet first detects noisy and fragmented large instance candidates using a watershed-level prediction head (see Fig. 1). These candidate regions are clustered and their boundaries improved with a triplet loss based head. The remaining smaller instances, with unreliable boundaries, are detected using a Hough voting head that predicts the offsets to the center of the instance. Mean-shift clustering followed by vote back-tracing is used to reliably detect the smaller instances. Without using MoE our method produces comparable results to proposal based approaches while outperforming proposal-free methods on the COCO dataset.

To summarise, we present BBFNet for panoptic segmentation which is a flexible, bounding-box free non-MoE approach that does not use the output of any instance segmentation or detection network while outperforming existing non-proposal based methods.

## 2 Related Work

Despite the recent introduction of panoptic segmentation, there have already been multiple works attempting to address this [12,19,22,40]. This is in part due to its importance to the wider community, success in individual subtasks of instance and semantic segmentation and publicly available datasets to benchmark different methods.

**Panoptic Segmentation:** Most current works in panoptic segmentation fall under the proposal based approach for detecting *things*. In [17], Kirillov *et al.* use separate networks for semantic segmentation (*stuff*) and instance segmentation (*things*) with a heuristic MoE fusion of the two results for the final prediction. Realising the duplication of feature extractors in the two related tasks, [16,19,22,28,40] propose using a single backbone feature extractor network. This is followed by separate branches for the two sub-tasks with a heuristic or learnable MoE head to combine the results. While panoptic Feature Pyramid Networks (FPN) [16] uses Mask R-CNN [13] for the *things* classes and fills in the *stuff* classes using a separate FPN branch, UPSNet [40] combines the resized logits of the two branches to predict the final output. In AUNet [22], attention masks predicted from the Region Proposal Network (RPN) and the instance segmentation head help fusing the results of the two tasks. Instead of relying only on the instance segmentation branch, TASCNet [19] predicts a coherent mask for the *things* and *stuff* classes using both branches. This is later filled with the respective outputs. All these methods rely on Mask R-CNN [13] for predicting *things*. Mask R-CNN is a two-stage instance segmentation network which uses a RPN to predict initial candidates for instance. The proposed candidates are either discarded or refined and a separate head produces segmentation for the remaining candidates. The two-stage serial approach makes Mask R-CNN accurate albeit computationally expensive and inflexible thus slowing progress towards real-time panoptic segmentation.

In FPSNet [12], the authors replace Mask R-CNN with a computationally less expensive detection network and use its output as a soft attention mask to guide the prediction of *things* classes. This trade off is at a cost of considerable reduction in accuracy while continuing to use a computationally expensive backbone (ResNet50 [14]). In [21] the authors make up for the reduced accuracy by using an affinity network but this is at the cost of computational complexity. Both these methods still use bounding-boxes for predicting *things*. In [35], the detection network is replaced with an object proposal network which predicts instance candidates. In contrast, we propose a flexible panoptic segmentation head that relies only on a semantic segmentation network which, when replaced with faster networks [32,33] allows for a more efficient solution.

A parallel direction gaining increased popularity is the use of proposal-free approach for predicting *things*. In [37], the authors predict the direction to the center and replace bounding box detection with template matching using these predicted directions as a feature. Instead of template matching, [1,20] use a dynamically initiated conditional random field graph from the output of an object detector to segment instances. In the more recent work of Gao *et al.* [11],

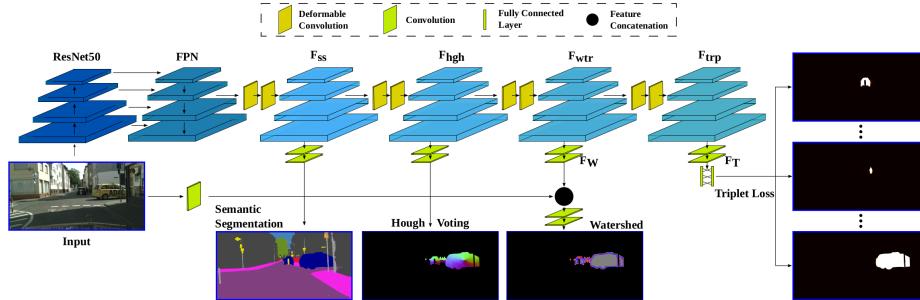
cascaded graph partitioning is performed on the predictions of a semantic segmentation network and an affinity pyramid computed within a fixed window for each pixel. Cheng *et al.* [5] simplify this process by adopting a parallelizable grouping algorithm for *thing* pixel to predict instance segmentation. In comparison, our flexible panoptic segmentation head predicts *things* by refining the segmentation boundaries obtained from any backbone semantic segmentation network. Furthermore, our post-processing steps are computationally more efficient compared to other proposal-free approaches while outperforming them on multiple datasets.

**Instance segmentation:** Traditionally predicting instance segmentation masks relied on obtaining rough boundaries followed by refining them [18,34]. With the success of deep neural networks in predicting object proposals [29,31], and the advantages of an end-to-end learning method, proposal based approaches have become more popular. Recent works have suggested alternatives to predicting proposals in an end-to-end trainable network. As these are most relevant to our work, we only review these below.

In [2], the authors propose predicting quantised watershed energies [38] using a Deep Watershed Transform network (DWT). Connected-components on the second-lowest watershed energy level are used to predict the instance segments. While this does well on large instances it suffers on small and thin instances. Moreover, fragmented regions of occluded objects end up being detected as different instances. In comparison, [4] embed the image into a transformed feature space where pixels of the same instance cluster together and pixels of different instances are pushed apart. While this method is not affected by object fragmentation, poor clustering often leads to either clustering multiple objects as single instance (under-segmentation) or segmenting large objects into multiple instances (over-segmentation). In [26], the authors try to address this by using variable clustering bandwidths predicted by the network. In this work, we observe the complementary advantages of these methods and exploit it towards our goal of an accurate, bounding-box free panoptic segmentation.

### 3 Panoptic Segmentation

In this section we introduce our non-bounding box approach to panoptic segmentation. Fig. 2 shows the various blocks of our network and Table 1 details the main components of BBFNet. The backbone semantic segmentation network consists of a ResNet50 followed by an FPN [23]. In FPN, we only use the P2, P3, P4 and P5 feature maps which contain 256 channels each and are 1/4, 1/8, 1/16 and 1/32 of the original scale respectively. Each feature map then passes through the same series of eight Deformable Convolutions(DC) [8]. Intermediate features after every couple of DC are used to predict semantic segmentation (§3.1), Hough votes (§3.2), watershed energies (§3.3) and features for the triplet loss [39] network. We first explain each of these components and their corre-



**Fig. 2.** BBFNet gradually refines the class boundaries of the backbone semantic segmentation network to predict panoptic segmentation. The watershed head predicts quantized watershed levels (shown in different colours) which is used to detect large instance candidates. For smaller instances we use Hough voting with fixed bandwidth. The output shows offsets ( $X_{\text{off}}, Y_{\text{off}}$ ) colour-coded to represent the direction of the predicted vector. Triplet head refines and merges the detection to obtain the final instance labels. We show the class probability (colour-map *hot*) for different instances with their center pixels used as  $f_a$ . Table 1 lists the components of individual heads while §3 explains them in detail.

sponding training loss. In (§3.5) we explain our training and inference steps. Through ablation studies we show the advantages of each block in (§4.2).

### 3.1 Semantic Segmentation

The first head in BBFNet is used to predict semantic segmentation. This allows BBFNet to quickly predict *things* ( $C_{\text{things}}$ ) and *stuff* ( $C_{\text{stuff}}$ ) labels while the remainder of BBFNet improves *things* boundaries using semantic segmentation features  $F_{\text{seg}}$ . We use per-pixel cross-entropy loss to train this head given by:

$$L_{ss} = \sum_{c \in \{C_{\text{stuff}}, C_{\text{thing}}\}} y_c \log(p_c^{ss}), \quad (1)$$

where  $y_c$  and  $p_c^{ss}$  are respectively the one-hot ground truth label and predicted softmax probability for class  $c$ .

### 3.2 Hough Voting

The Hough voting head is similar to the semantic segmentation head and is used to refine  $F_{ss}$  to give Hough features  $F_{\text{hgh}}$ . These are then used to predict offsets for the center of each *things* pixel. We use a  $\tanh$  non-linearity to squash the predictions and obtain normalised offsets ( $\hat{X}_{\text{off}}$  and  $\hat{Y}_{\text{off}}$ ). Along with the centers we also predict the uncertainty in the two directions ( $\sigma_x$  and  $\sigma_y$ ) making the number of predictions from the Hough voting head equal to  $4 \times C_{\text{things}}$ . The

predicted center for each pixel  $(x, y)$ , is then given by:

$$\begin{aligned}\hat{X}_{\text{center}}(x, y) &= \hat{x} + \hat{X}_{\text{off}}^{C(x,y)}(x, y), \\ \hat{Y}_{\text{center}}(x, y) &= \hat{y} + \hat{Y}_{\text{off}}^{C(x,y)}(x, y),\end{aligned}\quad (2)$$

where  $C$  is the predicted class and  $(\hat{x}, \hat{y})$  are image normalised pixel location.

Hough voting is inherently noisy [3] and requires clustering or mode seeking methods like mean-shift [6] to predict the final object centers. As instances could have different scales, tuning clustering hyper-parameters is difficult. For this reason we use Hough voting primarily to detect small objects and to filter predictions from other heads. We also observe that the dense loss from the Hough voting head helps convergence of deeper heads in our network.

The loss for this head is only for the *thing* pixels and is given by:

$$L_{\text{hgh}} = w \left( \frac{(X_{\text{off}} - \hat{X}_{\text{off}})^2}{\sigma_x} + \frac{(Y_{\text{off}} - \hat{Y}_{\text{off}})^2}{\sigma_y} \right) - \frac{1}{2} \left( \log(\sigma_x) + \log(\sigma_y) \right), \quad (3)$$

where  $X_{\text{off}}$  and  $Y_{\text{off}}$  are ground truth offsets and  $w$  is the per pixel weight. To avoid bias towards large objects, we inversely weigh the instances based on the number of pixels. This allows it to accurately predict the centers for objects of all sizes. Note that we only predict the centers for the visible regions of an instance and do not consider its occluded regions.

### 3.3 Watershed Energies

Our watershed head is inspired from DWT [2]. Similar to that work, we quantise the watershed levels into fixed number of bins ( $K = 4$ ). The lowest bin ( $k = 0$ ) corresponds to background and regions that are within 2 pixels inside the instance boundary. Similarly,  $k = 1, k = 2$  are for regions that are within 5 and 15 pixels away from the instance boundary, respectively, while  $k = 3$  is for the remaining region inside the instance.

In DWT, the bin corresponding to  $k = 1$  is used to detect large instance boundaries. While this does reasonably well for large objects, it fails for smaller objects producing erroneous boundaries. Furthermore, occluded instances that are fragmented cannot be detected as a single object. For this reason we use this head only for predicting large object candidates which are filtered and refined using predictions from other heads.

Due to the fine quantisation of watershed levels, rather than directly predicting the upsampled resolution, we gradually refine the lower resolution feature maps while also merging higher resolution features from the backbone semantic segmentation network.  $F_{\text{hgh}}$  is first transformed into  $F_{\text{wtr}}$  followed by further refining into  $F_W$  as detailed in Table 1. Features from the shallowest convolution block of ResNet are then concatenated with  $F_W$  and further refined with two 1 convolution to predict the four watershed levels.

We use a weighted cross-entropy loss to train this given by:

$$L_{\text{wtr}} = \sum_{k \in (0,3)} w_k W_k \log(p_k^{\text{wtr}}), \quad (4)$$

Input	Blocks	Output
FPN	dc-256-256, dc-256-128	$F_{ss}$
$F_{ss}$	ups, cat, conv-512- $(C_{stuff} + C_{thing})$ , ups	Segmentation
$F_{ss}$	$2 \times$ dc-128-128	$F_{hgh}$
$F_{hgh}$	ups, cat, conv-512-128, conv-128- $(4 \times C_{thing})$ , ups	Hough
$F_{hgh}$	$2 \times$ dc-128-128	$F_{wtr}$
$F_{wtr}$	ups, cat, conv-512-128, conv-128-16, ups	$F_W^*$
$F_{wtr}$	$2 \times$ dc-128-128	$F_{trp}$
$F_{trp}$	ups, cat, conv-512-128, conv-128-128, ups	$F_T^*$

**Table 1.** Architecture of BBFNet. *dc*, *conv*, *ups* and *cat* stand for deformable convolution [8],  $1 \times 1$  convolution, upsampling and concatenation respectively. The two numbers that follow *dc* and *conv* are the input and output channels to the blocks.\* indicates that more processing is done on these blocks as detailed in §3.3 and §3.4.

where  $W_k$  is the one-hot ground truth for  $k^{th}$  watershed level,  $p_k^{wtr}$  its predicted probability and  $w_k$  its weights.

### 3.4 Triplet Loss Network

The triplet loss network is used to refine and merge the detected candidate instances in addition to detecting new instances. Towards this goal, a popular choice is to formulate it as an embedding problem using triplet loss [4]. This loss forces features of pixels belonging to the same instance to group together while pushing apart features of pixels from different instances. Margin-separation loss is usually employed for better instance separation and is given by:

$$L(f_a, f_p, f_n) = \max((f_a - f_p)^2 - (f_a - f_n)^2 + \alpha, 0), \quad (5)$$

where  $f_a, f_p, f_n$  are the anchor, positive and negative pixel features respectively and  $\alpha$  is the margin. Choosing  $\alpha$  is not easy and depends on the complexity of the feature space [26]. Instead, we opt for a fully-connected network to classify the pixel features and formulate it as a binary classification problem:

$$T(f_a, f_*) = \begin{cases} 1 & \text{if, } f_* = f_p, \\ 0 & \text{if, } f_* = f_n, \end{cases} \quad (6)$$

We use the cross-entropy loss to train this head:

$$L_{trp} = \sum_{c \in (0, 1)} T_c \log(p_c^{trp}), \quad (7)$$

$T_c$  is the ground truth one-hot label for the indicator function and  $p^{trp}$  the predicted probability.

The pixel feature used for this network is a concatenation of  $F_T$  (see Table 1), its normalised position in the image  $(x, y)$  and the outputs of the different heads ( $p^{seg}, p^{wtr}, \hat{X}_{\text{off}}, \hat{Y}_{\text{off}}, \sigma_x$  and  $\sigma_y$ ).

### 3.5 Training and Inference

We train the whole network along with its heads in an end-to-end fashion using a weighted loss function:

$$L_{\text{total}} = \alpha_1 L_{ss} + \alpha_2 L_{hgh} + \alpha_3 L_{wtr} + \alpha_4 L_{trp}. \quad (8)$$

For the triplet loss network, training with all pixels is prohibitively expensive. Instead we randomly choose a fixed number of anchor pixels  $N_a$  for each instance. Hard positive examples are obtained by sampling from the farthest pixels to the object center and correspond to watershed level  $k = 0$ . For hard negative examples, neighbouring instances' pixels closest to the anchor and belonging to the same class are given higher weight. Only half of the anchors use hard example mining while the rest use random sampling.

We observe that large objects are easily detected by the watershed head while Hough voting based center prediction does well when objects are of the same scale. To exploit this observation, we detect large object candidates ( $I_{L'}$ ) using connected components on the watershed predictions correspond to  $k \geq 1$  bins. We then filter out candidates whose predicted Hough center ( $I_{L'}^{\text{center}}$ ) does not fall within their bounding boxes ( $BB_{L'}$ ). These filtered out candidates are fragmented regions of occluded objects or false detections. Using the center pixel of the remaining candidates ( $I_{L''}$ ) as anchors points, the triplet loss network refines them over the remaining pixels allowing us to detect fragmented regions while also improving their boundary predictions.

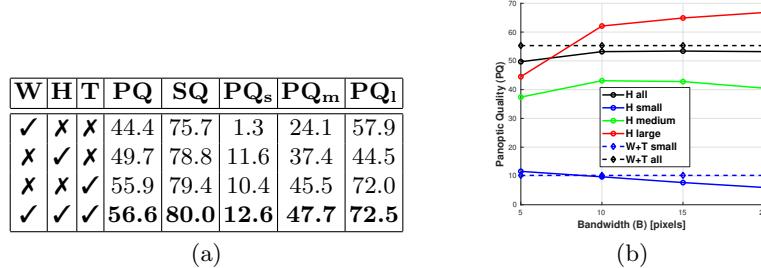
After the initial watershed step, the unassigned *thing* pixels corresponding to  $k = 0$  and primarily belong to small instances. We use mean-shift clustering with fixed bandwidth ( $B$ ) to predict candidate object centers,  $I_S^{\text{center}}$ . We then back-trace pixels voting for their centers to obtain the Hough predictions  $I_S$ .

Finally, from the remaining unassigned pixels we randomly pick an anchor point and test it with the other remaining pixels. We use this as candidates regions that are filtered ( $I_R$ ) based on their Hough center predictions, similar to the watershed candidates. The final detections are the union of these predictions. We summarise these steps in algorithm provided in the supplementary material.

## 4 Experiments

In this section we evaluate the performance of BBFNet and present the results we obtain. We first describe the datasets and the evaluation metrics used. In §4.1 we describe the implementation details of our network. §4.2 then discusses the performance of individual heads and how its combination helps improve the overall accuracies. We presents both the qualitative and quantitative results in §4.3 and show the flexibility of BBFNet in §4.4. We end this section by presenting some of the failure cases in §4.5 and comparing them with other MoE+BB based approaches.

**Datasets:** The Cityscapes dataset [7] contains driving scenes with  $2048 \times 1024$  resolution images recorded over various cities in Germany and Switzerland. It



**Table 2.** (a) Performance of different heads (W- Watershed, H- Hough Voting and T- Triplet Loss Network) on Cityscapes validation set. BBFNet exploits the complementary performance of watershed (large objects  $> 10k$  pixels) and Hough voting head (small objects  $< 1k$  pixels) resulting in higher accuracy. PQ<sub>s</sub>, PQ<sub>m</sub> and PQ<sub>l</sub> are the PQ scores for small, medium and large objects respectively. Bold is for best results. (b) Performance of Hough voting head (H) with varying  $B$  for different sized objects,  $s$ -small  $< 1k$  pixels,  $l$ -large  $> 10k$  pixels and  $m$ -medium sized instances. For reference we also plot the performance of Watershed+Triplet loss (W+T) head (see Table 2).

consists of 2975 densely annotated images training images and a further 500 validation images. For the panoptic challenge, a total of 19 classes are split into 8 *things* and 11 *stuff* classes.

Microsoft COCO [24] is a large scale object detection and segmentation dataset with over 118k training (2017 edition) and 5k validation images with varying resolutions. The labels consists of 133 classes split into 80 *things* and 53 *stuff*.

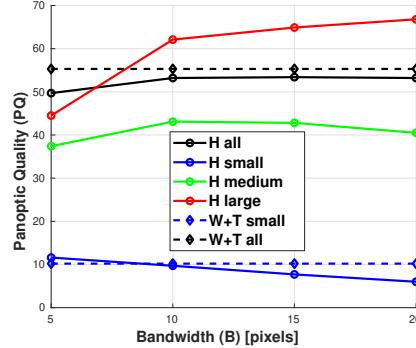
**Evaluation Metrics:** We benchmark using the Panoptic Quality (PQ) measure which was proposed in [16]. This measure comprises of two terms, Recognition Quality (RQ) and Segmentation Quality (SQ), to measure individual performance on recognition and segmentation tasks:

$$PQ = \underbrace{\frac{\sum_{(p,g) \in TP} IoU(p,g)}{|TP|}}_{SQ} \underbrace{\frac{|TP|}{|TP| + \frac{1}{2}|FP| + \frac{1}{2}|FN|}}_{RQ}, \quad (9)$$

where, IoU is the intersection-over-union measure, (p,g) are the matching predicted and ground-truth regions ( $> 0.5$  IoU), TP, FP and FN are true-positive, false-positive and false-negative respectively.

#### 4.1 Implementation Details

We use the pretrained ImageNet [9] models for ResNet50 and FPN and train the BBFNet head from scratch. We keep the backbone fixed for initial epochs before training the whole network jointly. In the training loss (*eq. 8*), we set  $\alpha_1, \alpha_2, \alpha_3$  and  $\alpha_4$  parameters to 1.0, 0.1, 1.0 and 0.5 respectively, since we found this to be a good balance between the different losses. The mean-shift bandwidth is



**Fig. 3.** Performance of Hough voting head (H) with varying  $B$  for different sized objects,  $s$ -small  $< 1k$  pixels,  $l$ -large  $> 10k$  pixels and  $m$ -medium sized instances. For reference we also plot the performance of Watershed+Triplet loss (W+T) head (see Table 2).

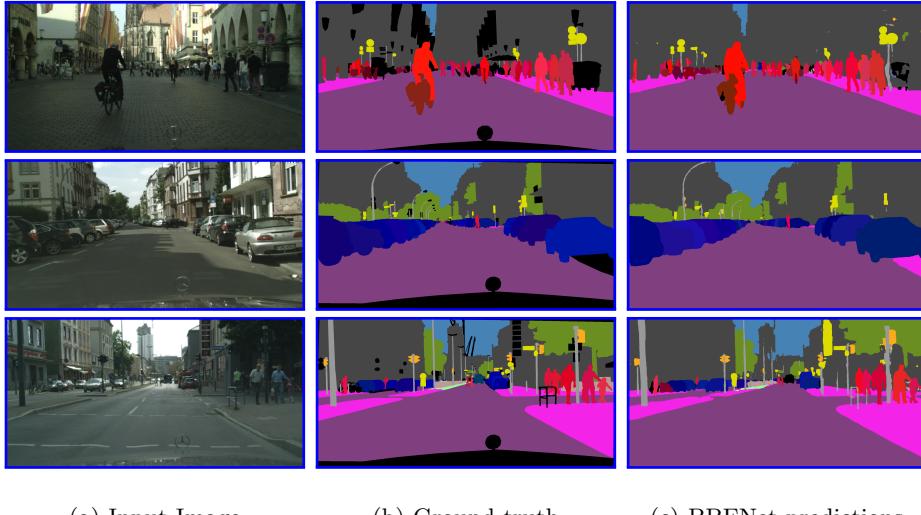
set to reduced pixels of  $B = 10$  to help the Hough voting head detect smaller instances. In the watershed head, the number of training pixels decreases with  $K$  and needs to be offset by higher  $w_k$ . We found the weights 0.2, 0.1, 0.05, 0.01 to work best for our experiments. Moreover, these weights help the network focus on detecting pixels corresponding to lower bins on whom the connected-component is performed. To train the triplet-loss network head we set the number of pixels per object  $N_a = 1000$ . For smaller instance, we sample with repetition so as to give equal importance to objects of all sizes. All experiments were performed on NVIDIA Titan 1080Ti.

To improve robustness we augment the training data by randomly cropping the images and adding alpha noise, flipping and affine transformations. No additional augmentation was used during testing. All experiments were performed on NVIDIA Titan 1080Ti. Cityscapes dataset is trained with full resolution. For COCO, the longest edge of each image is resized to 1024 while keeping the aspect ratio same.

A common practice during inference is to remove prediction with low detection probability to avoid penalising twice (FP and FN) [40]. In BBFNet, these correspond to regions with poor segmentation (class or boundary). We use the mean segmentation probability over the predicted region as the detection probability and filter regions with low probability ( $< 0.65$ ). Furthermore, we also observe boundaries shared between multiple objects to be frequently predicted as different instances. We filter these by having a threshold (0.1) on the IoU between the segmented prediction and its corresponding bounding box.

#### 4.2 Ablation studies

We conduct ablation studies here to show the advantage of each individual head and how BBFNet exploits them. Table 2(a) shows the results of our experiments



**Fig. 4.** Sample qualitative results of BBFNet on Cityscapes dataset. BBFNet is able to handle complex scenes with multiple occluded and fragmented objects.

on Cityscapes. We use the validation sets for all our experiments. We observe that watershed or Hough voting heads alone do not perform well. In the case of watershed head this is because performing connected component analysis on  $k = 1$  level (as proposed in [2]) leads to poor SQ. Note that performing the watershed cut at  $k = 0$  is also not optimal as this leads to multiple instances that share boundaries being grouped into a single detection. By combining the Watershed head with a refining step from the triplet loss network we observe over 10 point improvement in accuracy.

On the other hand, the performance of the Hough voting head depends on the bandwidth  $B$  that is used. Table 2(b) plots its performance with varying  $B$ . As  $B$  increases from 5 to 20 pixels we observe an initial increase in overall PQ before it saturates. This is because while the performance increases on large objects ( $> 10k$  pixels), it reduces on small ( $< 1k$  pixels) and medium sized objects. However, we observe that at lower  $B$  it outperforms the Watershed+triplet loss head on smaller objects. We exploit this in BBFNet (see §3.5) by using the watershed+triplet loss head for larger objects while using Hough voting head primarily for smaller objects.

### 4.3 Experimental Results

Table 3 benchmarks the performance of BBFNet with existing methods on the Cityscapes and COCO datasets. As all state-of-the-art methods report results with ResNet50+FPN networks while using the same pre-training dataset (ImageNet) we also follow this convention and report our results with this setup except where highlighted. Multi-scale testing along with horizontal-flipping were

Method	BB	Cityscapes				COCO				
		PQ	PQ <sub>Th</sub>	PQ <sub>St</sub>	IoU	PQ	PQ <sub>Th</sub>	PQ <sub>St</sub>	IoU	PQ <sub>test</sub>
FPSNet [12]	✓	55.1	48.3	60.1	-	-	-	-	-	-
Li <i>et al.</i> [21]	✓	<b>61.4</b>	54.7	<b>66.6</b>	77.8	<b>43.4</b>	<b>48.6</b>	<b>35.5</b>	<b>53.7</b>	<b>47.2</b> <sup>*</sup>
Porzi <i>et al.</i> [28]	✓	60.3	<b>56.1</b>	63.6	77.5	-	-	-	-	-
TASCNet [19]	✓	55.9	50.5	59.8	-	-	-	-	-	40.7
AUNet [22]	✓	56.4	52.7	59.0	73.6	39.6	49.1	25.2	45.1	45.2 <sup>*</sup>
P. FPN [16]	✓	57.7	51.6	62.2	75.0	39.0	45.9	28.7	41.0	40.9 <sup>*</sup>
AdaptIS [35]	✓	59.0	55.8	61.3	75.3	35.9	29.3	40.3	-	42.8 <sup>*</sup>
UPSNet [40]	✓	59.3	54.6	62.7	75.2	42.5	48.5	33.4	54.3	46.6 <sup>*</sup>
DIN [20]	✗	53.8	42.5	<u>62.1</u>	71.6	-	-	-	-	-
DeeperLab [41]	✗	56.5 $\pm$	-	-	-	33.8 $\pm$	-	-	-	34.4 $\pm$
SSAP [11]	✗	56.6	49.2	-	75.1	36.5 <sup>*</sup>	-	-	-	36.9 <sup>*</sup>
P. DeepLab [5]	✗	<b>59.7</b>	-	-	<b>80.5</b>	35.1	-	-	-	41.4 $\pm$
BBFNet	✗	56.6	<u>49.9</u>	61.1	76.5	<u>37.1</u>	42.9	<u>28.5</u>	<b>54.9</b>	<u>42.9</u> <sup>*</sup>

**Table 3.** Panoptic segmentation results on the Cityscapes and COCO dataset. All methods use the same pretraining (ImageNet) and backbone (ResNet50+FPN), except those with \* (ResNet101) and  $\pm$  (Xception-71). Bold is for overall best results and underscore is the best result in non-BB based methods.

used in some works but we omit those results here as this can be applied to any existing work including BBFNet to improve performance. From the results we observe that BBFNet, without using an MoE or BB, has comparable performance to other MoE+BB based methods while outperforming non-BB based methods on the more complicated COCO dataset. Fig. 4 shows some qualitative results on the Cityscapes validation dataset.

#### 4.4 Flexibility and Efficiency

To highlight BBFNets ability to work with different segmentation backbones we compare its generalisation with different segmentation networks. As it is expected we observe an increase in performance with more complex backbones and with DC’s but at a cost of reduced efficiency (see Table 4). For reference we also show the performance of a baseline proposal-based approach (UPSNet) and a proposal-free approach (SSAP). We used the author provided code of UPSNet<sup>1</sup> for computing efficiency figures. Note, that since UPSNet uses Mask R-CNN its backbone cannot be replaced and it is not as flexible as BBFNet.

As BBFNet does not use a separate instance segmentation head, it is computationally more efficient using only  $\approx 28.6M$  parameters compared to  $44.5M$  UPSNet. We find a similar pattern when we compare the number of FLOPs on a  $1024 \times 2048$  image with BBFNet taking 0.38 TFLOPs compared to 0.425 TFLOPs of UPSNet when using the same ResNet50 backbone. The authors of SSAP [11] do not provide details about their number of parameters, FLOPs

<sup>1</sup> Source code available from <https://github.com/uber-research/UPSNet>

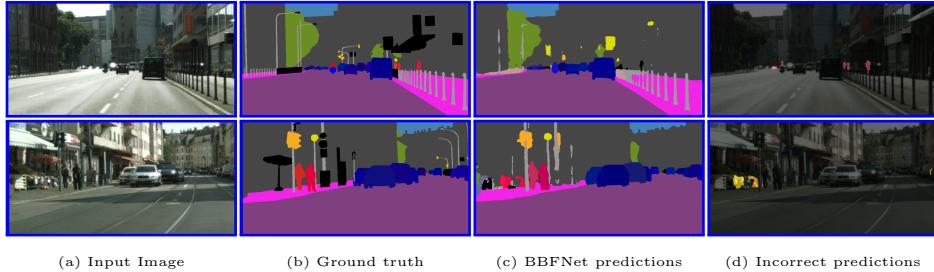


**Fig. 5.** Sample qualitative results of BBFNet on COCO dataset. BBFNet can handle different object classes with multiple instances.

and inference time. However, they provide timing information for their post-processing step which is a cascaded graph partitioning approach that uses the predictions of a semantic segmentation network and an affinity pyramid net-

Network	Backbone	PQ	SQ	RQ	IoU	Flops	Params	T <sub>I</sub>	T <sub>PP</sub>
		(T)	(M)					(sec)	(sec)
BBFNet	ERFNet [32] (w/o DC)	46.8	76.2	58.7	69.0	<b>0.07</b>	<b>2.58</b>	<b>0.12</b>	<b>0.15</b>
	MobileNetV2 [33]	48.2	77.0	60.3	70.1	0.1	4.26	0.22	0.15
	ResNet50 [14] (w/o DC)	50.4	76.3	62.4	68.9	0.49	28.2	0.16	0.15
	ResNet50 [14]	56.6	80.0	69.3	76.5	0.38	29.5	0.32	0.15
	ResNet101 [14]	57.8	<b>80.7</b>	70.2	<b>78.6</b>	0.53	48.49	0.34	0.15
SSAP [11]	ResNet50 [14]	56.6	-	-	75.1	-	-	-	$\geq 0.26$
UPSNet [40]	ResNet50 [14]	<b>59.3</b>	79.7	<b>73.0</b>	75.2	0.425	44.5	0.2	0.33

**Table 4.** Panoptic segmentation results showing the trade-off between performance and efficiency with different semantic segmentation backbones on the Cityscapes dataset. For efficiency we use flops, parameters, inference time (T<sub>I</sub>) and the post-processing time (T<sub>PP</sub>). We compare this with a baseline proposal based network (UPSNet) and a proposal free network (SSAP).



**Fig. 6.** Sample results where BBFNet fails. First row shows an example where low confidence of semantic segmentation network leads to missed detection while the second row shows examples of false positives due to wrong class label prediction. Without MoE these errors from the semantic segmentation network cannot be corrected by BBFNet.

work. This cascaded graph partition module solves a multicut optimisation problem [15] and takes between 0.26–1.26 seconds depending on the initial resolution for the cascaded graph partition. We believe that BBFNet post-processing step is simpler and presumably faster than the one in SSAP.

#### 4.5 Error Analysis

We discuss the reasons for performance difference between our bounding-box free method and ones that use bounding-box proposals. UPSNet [40] is used as a benchmark as it shares common features with other methods. Table 5 depicts the number of predictions made for different sized objects in the Cityscapes validation dataset. We report the True Positive (TP), False Positive (FP) and the False Negative (FN) values.

One of the areas where BBFNet performs poorly is the number of small object detections. BBFNet detects 2/3 of the smaller objects compared to UPSNet. Poor segmentation (wrong class label or inaccurate boundary prediction) also leads to a relatively higher FP for medium and large sized objects. Figure 6 shows some sample examples. The multi-head MoE approach helps addressing these issues but at the cost of additional complexity and computation time (§4.3). For applications where time or memory are more critical compared to detecting smaller objects, BBFNet would be a more suited solution.

Network	Small			Medium			Large		
	TP	FP	FN	TP	FP	FN	TP	FP	FN
UPSNet	1569	722	2479	3496	401	954	1539	49	82
BBFNet	1067	666	2981	3446	680	1004	1527	82	94

**Table 5.** Performance comparison of BBFNet with an MoE+BB method (UPSNet). Due to a non-MoE approach, errors from the backbone semantic segmentation network (low TP-small and high FP-medium, large) cannot be corrected by BBFNet.

## 5 Conclusions and Future Work

We presented an efficient bounding-box free panoptic segmentation method called BBFNet. Unlike previous methods, BBFNet does not use any instance segmentation network to predict *things*. It instead refines the boundaries from the semantic segmentation output obtained from any off-the-shelf segmentation network. This allows us to be flexible while out-performing proposal-free methods on the more complicated COCO benchmark.

In the next future we would work on making the network end-to-end trainable and improving the efficiency by removing the use of DCN while maintaining similar accuracy.

## 6 Acknowledgment

We would like to thank Prof. Andrew Davison and Dr. Alexandre Morgand for their critical feedback during the course of this work.

## References

1. Arnab, A., Torr, P.H.: Pixelwise instance segmentation with a dynamically instantiated network. In: IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) (2017)
2. Bai, M., Urtasun, R.: Deep watershed transform for instance segmentation. In: IEEE Conf. on Computer Vision and Pattern Recognition (CVPR). pp. 2858–2866 (2017)
3. Ballard, D.H.: Generalizing the hough transform to detect arbitrary shapes. Pattern Recognition 13(2), 111–122 (1981)
4. Brabandere, B.D., Neven, D., Gool, L.V.: Semantic instance segmentation with a discriminative loss function. arXiv preprint arXiv:arXiv:1708.02551 (2017)
5. Cheng, B., Collins, M., Zhu, Y., Liu, T., Huang, T., Adam, H., Chen, L.: Panoptic-deeplab:a simple, strong, and fast baseline for bottom-up panoptic segmentation. In: IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) (2020)
6. Cheng, Y.: Mean shift, mode seeking, and clustering. IEEE Trans. Pattern Anal. Machine Intell. 17(8), 790–799 (1995)
7. Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., Schiele, B.: The cityscapes dataset for semantic urban scene understanding. In: IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) (2016)
8. Dai, J., Qi, H., Xiong, Y., Li, Y., Zhang, G., Hu, H., Wei, Y.: Deformable convolutional networks. In: Intl. Conf. on Computer Vision (ICCV) (2017)
9. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) (2009)
10. Forsyth, D., Malik, J., Fleck, M., Greenspan, H., Leung, T., Belongie, S., Carson, C., Bregler, C.: Finding pictures of objects in large collections of images. In: Intl. workshop on object representation in computer vision (1996)
11. Gao, N., Shan, Y., Wang, Y., Zhao, X., Yu, Y., Yang, M., Huang, K.: SSAP: Single-shot instance segmentation with affinity pyramid. In: Intl. Conf. on Computer Vision (ICCV) (2019)
12. de Geus, D., Meletis, P., Dubbelman, G.: Fast panoptic segmentation network. arXiv preprint arXiv:arXiv:1910.03892 (2019)
13. He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask r-cnn. In: Intl. Conf. on Computer Vision (ICCV) (2017)
14. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) (2015)
15. Keuper, M., Levinkov, E., Bonneel, N., Lavoué, G., Brox, T., Andres, B.: Efficient decomposition of image and mesh graphs by lifted multicut. In: Intl. Conf. on Computer Vision (ICCV) (2015)
16. Kirillov, A., Girshick, R., He, K., Dollár, P.: Panoptic feature pyramid networks. In: IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) (2019)
17. Kirillov, A., He, K., Girshick, R., Rother, C., Dollár, P.: Panoptic segmentation. In: IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) (2019)
18. Ladický, L., Russell, C., Kohli, P., Torr, P.H.S.: Associative hierarchical CRFs for object class image segmentation. In: IEEE Conf. on Computer Vision and Pattern Recognition (CVPR). pp. 739–746 (2009)
19. Li, J., Raventos, A., Bhargava, A., Tagawa, T., Gaidon, A.: Learning to fuse things and stuff. arXiv preprint arXiv:arXiv:1812.01192 (2019)

20. Li, Q., Arnab, A., Torr, P.H.: Weakly-and semi-supervised panoptic segmentation. In: Eur. Conf. on Computer Vision (ECCV) (2018)
21. Li, Q., Qi, X., Torr, P.: Unifying training and inference for panoptic segmentation. In: IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) (2020)
22. Li, Y., Chen, X., Zhu, Z., Xie, L., Huang, G., Du, D., Wang, X.: Attention-guided unified network for panoptic segmentation. In: IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) (2019)
23. Lin, T., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S.: Feature pyramid networks for object detection. In: IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) (2017)
24. Lin, T., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft COCO: Common objects in context. In: Eur. Conf. on Computer Vision (ECCV) (2014)
25. Neuhold, G., Ollmann, T., Bulò, S.R., Kontschieder, P.: The Mapillary Vistas dataset for semantic understanding of street scenes. In: Intl. Conf. on Computer Vision (ICCV) (2017), <https://www.mapillary.com/dataset/vistas>
26. Neven, D., Brabandere, B.D., Proesmans, M., Gool, L.V.: Instance segmentation by jointly optimizing spatial embeddings and clustering bandwidth. In: IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) (2019)
27. Neven, D., Brabandere, B.D., Georgoulis, S., Proesmans, M., Gool, L.V.: Fast scene understanding for autonomous driving. arXiv preprint arXiv:arXiv:1708.02550 (2017)
28. Porzi, L., Bulò, S.R., Colovic, A., Kontschieder, P.: Seamless scene segmentation. In: IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) (2019)
29. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) (2016)
30. Redmon, J., Farhadi, A.: Yolov3: An incremental improvement. arXiv preprint arXiv:arXiv:1804.02767 (2018)
31. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards real-time object detection with region proposal networks. In: Advances in Neural Information Processing Systems (NIPS) (2015)
32. Romera, E., Álvarez, J.M., Bergasa, L.M., Arroyo, R.: Erfnet: Efficient residual factorized convnet for real-time semantic segmentation. In: IEEE Trans. on Intelligent Transportation Systems. vol. 19, pp. 263–272 (2018)
33. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) (2018)
34. Shotton, J., Winn, J., Rother, C., Criminisi, A.: Textonboost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context. IEEE Trans. Pattern Anal. Machine Intell. 81(1), 2–23 (2009)
35. Sofiiuk, K., Barinova, O., Konushin, A.: Adaptis: Adaptive instance selection network. In: Intl. Conf. on Computer Vision (ICCV) (2019)
36. Tighe, J., Niethammer, M., Lazebnik, S.: Scene parsing with object instances and occlusion ordering. In: IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) (2014)
37. Uhrig, J., Cordts, M., Franke, U., Brox, T.: Pixel-level encoding and depth layering for instance-level semantic labeling. In: German Conference on Pattern Recognition (GCPR) (2016)

38. Vincent, L., Soille, P.: Watersheds in digital spaces: an efficient algorithm based on immersion simulations. *IEEE Trans. Pattern Anal. Machine Intell.* 13(6), 583–598 (1991)
39. Weinberger, K.Q., Saul, L.K.: Distance metric learning for large margin nearest neighbor classification. *J. of Machine Learning Research* (2009)
40. Xiong, Y., Liao, R., Zhao, H., Hu, R., Bai, M., Yumer, E., Urtasun, R.: UPSNet: A unified panoptic segmentation network. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* (2019)
41. Yang, T., Collins, M., Zhu, Y., Hwang, J., Liu, T., Zhang, X., Sze, V., Papandreou, G., Chen, L.: Deeperlab: Single-shot image parser. arXiv preprint arXiv:arXiv:1902.05093 (2019)
42. Yao, J., Fidler, S., Urtasun, R.: Describing the scene as a whole: joint object detection. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* (2012)

**Algorithm 1** Compute Instance segments  $I$ 


---

**Require:** Watershed levels  $W_k$ , predicted class  $c$ , probability  $p_c^{ss}$ ,  $\hat{X}_{\text{center}}$  and  $\hat{Y}_{\text{center}} \vee c \in C_{\text{thing}}$

- 1  $I_{L'} \leftarrow$  connected-components on  $W_k \geq 1$ .  $\triangleright$  Large instance candidates
- 2  $BB_{I_{L'}} \leftarrow$  bounding-box of  $I_{L'}$ .
- 3  $I_{L'}^{\text{center}} \leftarrow \frac{\sum_{x \in I_{L'}} \hat{X}_{\text{center}} p_c^{ss}}{\sum_{x \in I_{L'}} p_c^{ss}}, \frac{\sum_{y \in I_{L'}} \hat{Y}_{\text{center}} p_c^{ss}}{\sum_{y \in I_{L'}} p_c^{ss}}$ .
- 4  $I_{L''} \leftarrow I_{L'}^{\text{center}} \in BB_{I_{L'}}$ .  $\triangleright$  Filter candidates
- 5  $I_L \leftarrow I_{L''} \cup T(f_a, f_*) = 1 \vee f_a = I_{L''}^{\text{center}} \& f_* = c \in C_{\text{thing}} / I_{L''}$ . 6
- 6  $I_S^{\text{center}} \leftarrow \text{meanshift } \vee c \in C_{\text{thing}} / I_L$ .  $\triangleright$  Small instances
- 7  $I_S \leftarrow$  Back-trace pixels voting for  $I_S^{\text{center}}$
- 8 **while**  $c \notin \emptyset$  **do**  $\triangleright$  Remaining instances
- 9    $I_{R'} \leftarrow (\cup T(f_a, f_*) = 1) \vee f_a = \text{Random}(c) \& f_* = c \in C_{\text{thing}} / I_L / I_S$ . 6
- 10 **end while**
- 11  $BB_{I_{R'}} \leftarrow$  bounding-box of  $I_{R'}$ .
- 12  $I_{R'}^{\text{center}} \leftarrow \frac{\sum_{x \in I_{R'}} \hat{X}_{\text{center}} p_c^{ss}}{\sum_{x \in I_{R'}} p_c^{ss}}, \frac{\sum_{y \in I_{R'}} \hat{Y}_{\text{center}} p_c^{ss}}{\sum_{y \in I_{R'}} p_c^{ss}}$ .
- 13  $I_R \leftarrow I_{R'}^{\text{center}} \in BB_{I_{R'}}$ .  $\triangleright$  Filter candidates
- 14  $I \leftarrow I_L \cup I_S \cup I_R$

---

## 7 Supplementary

### 7.1 Inference Algorithm

We summarise the inference steps detailed in §3.5 in an algorithm 1

### 7.2 Cityscapes dataset

Table 6 gives the per-class results for the Cityscapes dataset. The first 11 classes are *stuff* while the rest 8 are *thing* label.

### 7.3 COCO dataset

Tables 7, 8 and 9 give the per-class results for the COCO dataset. The first 80 classes are *things* while the rest 53 are *stuff* label.

class	PQ	SQ	RQ	PQ <sub>s</sub>	PQ <sub>m</sub>	PQ <sub>l</sub>
road	97.9	98.2	99.7	0.0	0.0	0.0
sidewalk	74.9	84.0	89.2	0.0	0.0	0.0
building	87.4	89.2	98.0	0.0	0.0	0.0
wall	26.2	72.0	36.4	0.0	0.0	0.0
fence	27.6	72.9	37.8	0.0	0.0	0.0
pole	50.8	65.2	77.9	0.0	0.0	0.0
T. light	40.7	68.4	59.4	0.0	0.0	0.0
T. sign	64.8	76.4	84.7	0.0	0.0	0.0
vegetation	88.3	90.3	97.8	0.0	0.0	0.0
terrain	27.6	72.4	38.1	0.0	0.0	0.0
sky	85.1	91.9	92.7	0.0	0.0	0.0
person	48.0	76.3	62.9	22.9	62.0	81.9
rider	43.8	71.2	61.6	11.2	54.3	71.7
car	64.7	84.5	76.5	32.2	72.2	91.5
truck	48.2	84.5	57.0	6.7	37.3	72.3
bus	69.1	88.5	78.1	0.0	49.6	85.0
train	46.1	80.7	57.1	0.0	10.7	64.2
motorcycle	36.9	72.5	50.9	8.9	44.3	56.6
bicycle	40.6	70.2	57.9	17.4	47.1	56.8

**Table 6.** Per-class results for cityscapes dataset. The first 11 classes are from *stuff* while the rest 8 are from *thing* label.

class	PQ	SQ	RQ	PQ <sub>s</sub>	PQ <sub>m</sub>	PQ <sub>l</sub>
person	51.7	77.7	66.5	32.0	55.7	71.1
bicycle	17.6	66.9	26.4	7.9	19.5	33.2
car	42.1	81.0	52.0	30.9	54.9	56.0
motorcycle	40.6	74.1	54.8	13.7	35.7	58.9
airplane	56.8	78.0	72.7	45.4	37.5	72.3
bus	52.0	87.8	59.3	0.0	34.1	76.4
train	50.0	84.2	59.4	0.0	16.8	56.2
truck	24.3	78.4	31.0	13.3	21.5	36.7
boat	23.1	68.2	33.9	10.9	32.2	37.5
T. light	36.7	77.3	47.4	31.4	51.5	69.8
F. hydrant	77.5	87.1	88.9	0.0	71.6	91.3
S. sign	80.4	91.3	88.0	36.5	88.5	92.6
P. meter	56.2	87.9	64.0	0.0	48.6	82.0
bench	17.2	67.9	25.4	11.0	23.4	13.5
bird	28.2	73.5	38.4	15.0	47.5	78.6
cat	86.3	91.2	94.6	0.0	78.7	89.0
dog	69.3	86.0	80.6	0.0	58.5	82.9
horse	56.5	78.7	71.8	0.0	47.6	71.5
sheep	49.5	79.0	62.6	23.7	59.1	80.7
cow	42.3	82.5	51.4	0.0	32.4	70.1
elephant	63.0	83.9	75.0	0.0	37.4	71.5
bear	64.5	85.0	75.9	0.0	56.2	75.8
zebra	74.3	88.2	84.2	0.0	71.6	81.9
giraffe	73.1	82.2	88.9	0.0	77.0	72.4
backpack	9.6	83.5	11.5	2.8	16.4	34.7
umbrella	50.2	81.9	61.3	21.6	57.2	64.3
handbag	12.8	74.6	17.2	2.8	20.4	29.7
tie	29.8	77.6	38.5	0.0	56.2	51.4
suitcase	51.6	79.9	64.6	16.7	51.6	70.2
frisbee	70.4	85.8	82.1	51.1	77.7	93.0
skis	4.5	71.2	6.3	0.0	12.4	0.0
snowboard	24.2	65.3	37.0	9.5	34.3	0.0
kite	27.1	72.4	37.5	25.8	21.7	43.6
B. bat	23.8	67.9	35.0	35.0	8.5	0.0
B. glove	37.7	83.6	45.2	18.6	74.3	0.0
skateboard	37.3	71.5	52.2	0.0	48.8	50.6
surfboard	48.5	75.2	64.4	29.8	49.0	69.0
T. racket	58.1	83.0	70.0	27.1	68.6	86.7
bottle	38.6	80.7	47.8	29.5	49.4	81.8
wine glass	38.7	79.3	48.8	0.0	44.4	86.1
cup	48.5	88.1	55.0	15.9	70.9	75.6
fork	8.5	63.5	13.3	7.2	10.4	0.0
knife	17.7	78.7	22.5	0.0	26.3	68.2
spoon	20.2	76.4	26.4	0.0	36.9	0.0
bowl	29.9	78.6	38.0	17.3	32.2	39.8
banana	16.5	76.4	21.6	4.0	22.1	35.5
apple	30.4	87.5	34.8	8.0	63.6	51.3
sandwich	31.8	88.4	36.0	0.0	34.2	32.9
orange	59.8	88.3	67.7	36.1	37.9	82.8
broccoli	22.4	74.9	30.0	0.0	20.3	42.6
carrot	17.3	74.2	23.3	12.4	24.1	0.0
hot dog	26.5	68.6	38.6	13.7	29.6	27.5
pizza	44.5	83.2	53.5	12.6	37.5	54.5
donut	44.5	86.5	51.4	45.2	26.2	72.2

**Table 7.** Per-class results for COCO dataset. Continued in Table 8

class	PQ	SQ	RQ	PQ <sub>s</sub>	PQ <sub>m</sub>	PQ <sub>l</sub>
cake	49.9	90.2	55.3	0.0	31.6	62.3
chair	24.0	74.3	32.3	7.4	33.6	41.5
couch	44.1	80.8	54.5	0.0	32.4	52.4
P. plant	27.2	74.1	36.7	16.6	33.1	27.3
bed	48.4	82.0	59.0	0.0	0.0	57.2
D. table	13.0	71.5	18.2	0.0	7.7	21.0
toilet	73.2	86.9	84.2	0.0	58.3	78.5
tv	57.2	86.8	66.0	0.0	49.4	72.2
laptop	57.2	81.7	70.0	0.0	44.0	67.9
mouse	68.2	86.6	78.8	44.3	81.0	62.6
remote	20.7	80.1	25.8	6.8	48.8	0.0
keyboard	52.4	85.2	61.5	0.0	46.8	72.2
cell phone	46.1	84.9	54.3	15.0	66.2	58.1
microwave	61.3	91.9	66.7	0.0	60.7	94.8
oven	33.3	79.1	42.1	0.0	19.5	42.4
toaster	0.0	0.0	0.0	0.0	0.0	0.0
sink	49.5	81.8	60.5	30.8	56.8	45.7
refrigerator	30.6	87.2	35.1	0.0	12.0	41.9
book	8.1	70.6	11.5	6.3	11.6	13.1
clock	59.3	86.4	68.7	40.9	68.1	92.5
vase	31.8	80.5	39.4	22.4	35.3	42.5
scissors	0.0	0.0	0.0	0.0	0.0	0.0
teddy bear	49.0	82.4	59.4	0.0	39.8	72.8
hair drier	0.0	0.0	0.0	0.0	0.0	0.0
toothbrush	0.0	0.0	0.0	0.0	0.0	0.0
banner	5.5	79.9	6.9	0.0	0.0	0.0
blanket	0.0	0.0	0.0	0.0	0.0	0.0
bridge	22.0	71.3	30.8	0.0	0.0	0.0
cardboard	16.6	75.7	21.9	0.0	0.0	0.0
counter	19.7	67.8	29.0	0.0	0.0	0.
curtain	45.6	83.0	54.9	0.0	0.0	0.0
door-stuff	24.4	72.8	33.6	0.0	0.0	0.0
floor-wood	35.5	82.7	43.0	0.0	0.0	0.0
flower	12.5	65.8	19.0	0.0	0.0	0.0
fruit	5.4	65.0	8.3	0.0	0.0	0.0
gravel	11.6	63.5	18.2	0.0	0.0	0.0
house	13.5	72.5	18.6	0.0	0.0	0.0
light	16.1	67.5	23.8	0.0	0.0	0.0
mirror-stuff	28.2	80.4	35.1	0.0	0.0	0.0
net	33.7	84.3	40.0	0.0	0.0	0.0
pillow	0.0	0.0	0.0	0.0	0.0	0.0
platform	10.3	92.5	11.1	0.0	0.0	0.0
playingfield	69.4	87.6	79.2	0.0	0.0	0.0
railroad	25.5	72.9	35.0	0.0	0.0	0.0
river	22.2	82.1	27.0	0.0	0.0	0.0
road	45.6	83.1	54.9	0.0	0.0	0.0
roof	5.2	80.8	6.5	0.0	0.0	0.0
sand	40.6	91.4	44.4	0.0	0.0	0.0
sea	71.0	91.6	77.5	0.0	0.0	0.0
shelf	8.8	76.3	11.5	0.0	0.0	0.0
snow	81.0	91.8	88.2	0.0	0.0	0.0
stairs	10.9	65.4	16.7	0.0	0.0	0.0
tent	5.3	53.3	10.0	0.0	0.0	0.0
towel	16.8	77.7	21.6	0.0	0.0	0.0

**Table 8.** Per-class results for COCO dataset. Continued in table 9

class	PQ	SQ	RQ	PQ <sub>s</sub>	PQ <sub>m</sub>	PQ <sub>l</sub>
wall-brick	24.7	77.6	31.8	0.0	0.0	0.0
wall-stone	10.0	92.1	10.8	0.0	0.0	0.0
wall-tile	35.2	75.7	46.5	0.0	0.0	0.0
wall-wood	14.3	76.2	18.8	0.0	0.0	0.0
water-other	20.9	80.3	26.1	0.0	0.0	0.0
window-blind	44.6	84.7	52.6	0.0	0.0	0.0
window-other	22.2	73.7	30.0	0.0	0.0	0.0
tree-merged	64.6	80.7	80.0	0.0	0.0	0.0
fence-merged	19.7	74.9	26.3	0.0	0.0	0.0
ceiling-merged	57.3	81.8	70.1	0.0	0.0	0.0
sky-other-merged	76.9	90.4	85.1	0.0	0.0	0.0
cabinet-merged	33.1	79.7	41.5	0.0	0.0	0.0
table-merged	15.9	72.1	22.0	0.0	0.0	0.0
floor-other-merged	29.5	80.3	36.7	0.0	0.0	0.0
pavement-merged	36.4	78.9	46.2	0.0	0.0	0.0
mountain-merged	39.7	76.9	51.6	0.0	0.0	0.0
grass-merged	50.3	81.2	61.9	0.0	0.0	0.0
dirt-merged	27.4	77.0	35.6	0.0	0.0	0.0
paper-merged	4.7	74.6	6.3	0.0	0.0	0.0
food-other-merged	14.0	78.7	17.8	0.0	0.0	0.0
building-other-merged	29.3	76.4	38.4	0.0	0.0	0.0
rock-merged	31.0	78.4	39.6	0.0	0.0	0.0
wall-other-merged	45.6	79.2	57.6	0.0	0.0	0.0
rug-merged	38.3	82.7	46.4	0.0	0.0	0.0

**Table 9.** Per-class results for COCO dataset. The first 80 classes are from the *thing* while the rest 53 are from *stuff* label.