Fast Panoptic Segmentation Network

Daan de Geus, Panagiotis Meletis, and Gijs Dubbelman

Abstract—In this work, we present an end-to-end network for fast panoptic segmentation. This network, called Fast Panoptic Segmentation Network (FPSNet), does not require computationally costly instance mask predictions or rule-based merging operations. This is achieved by casting the panoptic task into a custom dense pixel-wise classification task, which assigns a class label or an instance id to each pixel. We evaluate FPSNet on the Cityscapes and Pascal VOC datasets, and find that FPSNet is faster than existing panoptic segmentation methods, while achieving better or similar panoptic segmentation performance. On the Cityscapes validation set, we achieve a Panoptic Quality score of 55.1%, at prediction times of 114 milliseconds for images with a resolution of 1024 × 2048 pixels. For lower resolutions of the Cityscapes dataset and for the Pascal VOC dataset, FPSNet achieves prediction times as low as 45 and 28 milliseconds, respectively.

Index Terms—Semantic scene understanding, object detection, segmentation and categorization, deep learning in robotics and automation.

I. INTRODUCTION

Panoptic segmentation [1] is a task for which the goal is to predict a class label and an instance id for each pixel in an image. A distinction is made between things and stuff classes. For things classes, which have countable objects (e.g. person, car), the instance id is used to distinguish between different objects, whereas all stuff classes receive the same instance id, as these parts of the image are usually uncountable (e.g. sky, water). In this work, we present an end-to-end deep neural network architecture for fast panoptic segmentation, that is able to achieve real-time inference speeds.

Panoptic segmentation is closely related to the tasks of semantic segmentation and instance segmentation. For semantic segmentation, the goal is to predict a class label – for both stuff and things classes – for each pixel in an image, whereas instance segmentation aims at finding pixel-level masks for all things instances in an image. Current panoptic segmentation methods [2]–[6] exploit this relation between these tasks. Instead of training the panoptic task directly, i.e. fully end-to-end, they train the instance segmentation and semantic segmentation tasks separately, and fuse the outputs into the panoptic format. This requires solving conflicts between instance segmentation and semantic segmentation predictions. Firstly, instance segmentation predictions can overlap each other, and secondly, pixels can also get different predictions from the instance segmentation and semantic segmentation output. These conflicts are problematic because panoptic segmentation allows for only one prediction per pixel. Current state-of-the-art methods rely on a rule-based post-processing step to merge the outputs and resolve these conflicts [2], [4], [6]. We propose a fast, end-to-end approach that is able to learn to resolve these conflicts without needing a rule-based merging step.

Although the existing panoptic segmentation methods achieve state-of-the-art panoptic segmentation quality, there are several...
To overcome these drawbacks, we present the Fast Panoptic Segmentation Network (FPSNet), an end-to-end architecture that is able to learn how to resolve conflicts between classes and instances. It does not need computationally expensive instance mask predictions or merging operations. Our FPSNet architecture, that is detailed in Section III, is compatible with any object detection backbone that is able to generate a single feature map for dense full-image segmentation.

To summarize, we present FPSNet, a fast panoptic segmentation architecture with the following contributions:

- FPSNet uses a novel architecture for end-to-end panoptic segmentation that does not require a) instance mask predictions or b) rule-based merging operations.
- With this architecture, we are able to achieve inference speeds significantly faster than existing methods [2], [3], [7], while achieving similar or better Panoptic Quality.

In the remainder of this paper, we will first discuss the related work in Section II. In Section III, we define the problem of achieving fast panoptic segmentation, and explain how we solve it with FPSNet. The experiments to evaluate FPSNet are explained in Section IV, and the results are presented in Section V. Finally, we provide conclusions in Section VI.

II. RELATED WORK

Panoptic segmentation [1] unifies the typically distinct tasks of semantic segmentation and instance segmentation. Earlier forms of panoptic segmentation have been investigated by many authors [8]–[10], but only recently it was formulated as a well defined problem by Kirillov et al. [1]. Existing approaches solve this problem either by using separate networks and then fusing the partial results [1], [8], [11] or by using a common backbone, and applying a specific head for each subtask followed by late fusion [2], [3], [5], [6], [12].

A baseline solution for panoptic segmentation is given in [1], according to which two state-of-the-art networks are trained independently. In that work, the outputs of Mask R-CNN [13] for instance segmentation and PSPNet [14] for semantic segmentation are fused by solving conflicts with a rule-based merging post-processing step. A clear downside of this method is the fact that it needs two separate networks, which is computationally costly. For this reason, several single network approaches have been presented for the task of panoptic segmentation.

JSIS-Net [12], Panoptic FPN [2] and the method by Porzi et al. [4] all introduce a common backbone, in order to reduce computation and to benefit from subtasks similarity, and connect it to two separate heads corresponding to the subtasks.

TASCNet [5] also consists of a common backbone and two heads, and is augmented with a Things And Stuff Consistency (TASC) loss to enforce per-pixel consistency between the heads. Although this extra loss offers higher consistency in the output distributions, it does not force outputs to be in the panoptic format, and thus the rule-based merging step is still needed. AUNet [6] applies a single network architecture and extends it with two sources of attention at mask and proposal level, to improve the segmentation of stuff classes. To improve the performance for things classes, OANet [15] proposes a spatial ranking module for solving occlusions between different things instances of the same class. Although all single network approaches described above improve computational efficiency by using a single common backbone, they still need to make expensive instance segmentation predictions, and rely on post-processing to generate the final panoptic output.

Another single network approach, UPSNet [3], applies rule-based merging within the network, and directly outputs the panoptic segmentation predictions, thereby improving the prediction speed. However, UPSNet still makes costly instance segmentation predictions using the two-stage Mask R-CNN [13] method. DeeperLab [7] solves the panoptic segmentation problem aiming at efficiency using a single-stage, five-head network, which generates per-pixel semantic and instance predictions. This leads to an efficient neural network, but this method still needs computationally costly merging post-processing to fuse the predictions into a coherent panoptic output.

The methods discussed so far, all require making instance mask predictions, using a rule-based method for merging outputs, or both. The method by Li et al. [11], which is based on the Dynamically Instantiated Network (DIN) [8], approaches panoptic segmentation in a different way. Here, cues from an external object detector are fused with a semantic segmentation output using a Conditional Random Field (CRF) in order to segment the semantic segmentation output into instances. In earlier work, methods like InstanceCut [16] and the work by Uhrig et al. [17] solved the same task with single unified networks, also relying on post-processing steps to split semantic segmentation predictions into instances. However, they are outperformed by DIN. Compared to the above methods that split predictions instead of merge, FPSNet has the following differences: 1) FPSNet does not make explicit semantic segmentation predictions for things classes first. 2) Our method does not need complex post-processing or CRFs to split semantic segmentation outputs into instances. 3) Most importantly, for FPSNet, the entire panoptic segmentation task is learned in an end-to-end fashion.

In our experiments, we compare with several state-of-the-art approaches including [2]–[4], [6], [11].

III. FAST PANOPTIC SEGMENTATION NETWORK

To achieve fast panoptic segmentation, we aim for a method that does not require:

1) making instance segmentation predictions;
2) a post-processing step to merge or split predictions.

We achieve this by introducing a novel convolutional neural network module, which we call the panoptic head. This head has two inputs: 1) a feature map on which we can perform dense
segmentation, and 2) attention masks indicating the presence of things instances, which are generated using detections from a regular bounding box object detector. From this, the model is trained to 1) perform semantic segmentation for stuff classes, 2) morph the attention masks into complete pixel-wise instance masks for things instances, and 3) output the predictions for both the stuff classes and things instances in a single map, on which we can do pixel-wise classification. This module is trained end-to-end in a single network, together with the required feature extractor and bounding box object detector.

We call our network the Fast Panoptic Segmentation Network (FPSNet), and introduce its components in more detail in the next sections. In Section III-A, we present the novel panoptic module and explain how it is trained. The network backbone is discussed in Section III-B. A diagram of the FPSNet architecture is depicted in Fig. 3.

### A. Panoptic Module

In our novel panoptic module for fast panoptic segmentation, we assume that we have bounding box object detections from a regular object detector, as well as a single feature map to apply dense image segmentation. The bounding boxes are used to generate attention masks to indicate the location of things in the image, and determine the order of the things in the output. The attention masks are first shuffled, then concatenated to the feature map, and finally applied to a fully convolutional network, i.e. the panoptic head.

At the output of the panoptic head, we predict, for each pixel, either a stuff class or a things instance id, which can directly be related back to a things class predicted by the bounding box object detector. This panoptic head is trained to morph the attention masks into coherent things instance masks. In essence, in the output features of the panoptic head, the stuff classes and the things instance ids are treated the same.

#### 1) Attention Mask Generation:

To indicate the location of things in the image, we generate attention masks, based on the bounding box object detections. We do this by projecting the bounding boxes on a tensor with the dimensions of the feature map, and filling the bounding box for each pixel $X = (x, y)$ with values of a 2D bell-shaped exponential function

$$f(X) = \exp \left(-\frac{1}{2}(X - C)S^{-1}(X - C)^T\right),$$

with center $C = (x_c, y_c)$ and spread $S = \text{Diag}(\frac{1}{4}w_b, \frac{1}{4}h_b)$, where $(x_c, y_c)$, $w_b$, $h_b$ are the center coordinates, width and height of the bounding box, respectively. This is similar to a Gaussian that is normalized such that the probability at $C$ is 1. Outside the bounding boxes, values are 0. This is depicted in Fig. 4. We opt for these so-called soft attention masks rather than hard masks - with a constant value for all pixels - because we assume that it is more likely that the object is located at the center of the bounding box, and therefore it should receive more attention there. In experiments, we show that soft masks indeed lead to better performance, see Section V-B. In total, we use $N_{\text{att}}$ attention masks. If there are more than $N_{\text{att}}$ objects detected by the object detector, we pick the $N_{\text{att}}$ bounding boxes with the highest class scores. If there are less, we use tensors filled with only zeros for the remaining masks, effectively applying no attention.

After the masks are generated, we shuffle the masks, so that objects of different sizes, class scores and class id are divided among the channels and filters of the convolutional layers as equally as possible during training. This is done to let each feature dimension related to attention masks to be treated equally
as possible. With experiments, we show that mask shuffling boosts the performance (see Section V-B).

2) Panoptic Head: After the attention masks are generated and shuffled, they are stacked in the outer, so-called channel dimension of a tensor, so that the tensor has the shape \([N_b, H, W, N_{att}]\), where \(N_b\) is the batch size, \(H\) and \(W\) are the height and width of the feature map, respectively, and \(N_{att}\) is the number of attention masks. The masks are then multiplied with a constant, \(C_{att}\), to make sure that the attention masks are in the same order of magnitude as the features, to facilitate learning.

The feature map from the backbone is then concatenated to the attention masks along the channel dimension, resulting in a tensor with the shape \([N_b, H, W, N_{att} + F_{dim}]\), where \(F_{dim}\) is the depth of the feature map from the backbone. Subsequently, we apply a 3 \(\times\) 3 convolution with ReLU activation [18] and batch normalization [19], to merge the concatenated feature map, before feeding it to the head architecture. This head consists of four more 3 \(\times\) 3 convolutional layers with ReLU activation and batch normalization. The panoptic head architecture is depicted in Fig. 5.

To get the final panoptic prediction, we apply a 1 \(\times\) 1 convolution to predict \(N_{out}\) outputs for each pixel, with \(N_{out} = N_{att} + N_{stuff}\). Here, \(N_{stuff}\) is the number of stuff classes. The pixels that are predicted for the first \(N_{att}\) outputs can be related back to the input attention masks. The pixels for the \(n\)th output belong to the \(n\)th attention mask (after shuffling) and its corresponding class. For each pixel, we get the final prediction by picking the instance or class with the highest score, i.e. applying \(\text{argmax}\), after bilinearly upsampling the logits to the dimensions of the input image. We then have single prediction for each pixel, i.e. a things instance or a stuff class. For the things instances, we can assign a unique \(id\) to each instance, but the class label does not directly follow from the panoptic head. To retrieve this class, recall that we used attention masks to make these predictions, which are generated from bounding box object detections that have class labels. As a result, we have a class label for each individual predicted things instance. Since instance \(ids\) do not play a role for stuff classes, we then have an output in the panoptic format. This process is depicted in more detail in Fig. 6.

3) Training: Because panoptic segmentation only allows a single prediction for each pixel, we treat the problem as a semantic segmentation problem during training. We construct a ground-truth consisting of a single prediction for each pixel, and apply a softmax cross-entropy loss. The desired output for each pixel is either a stuff class or a things instance \(id\), keeping in consideration that it is the responsibility of the object detector to provide the class label for each things instance.

The main challenge is to make sure that the order of the feature dimensions in the output tensor of the panoptic head related to things instance \(ids\), is the same as the order in which the attention masks are stacked in the input tensor of the panoptic head. Otherwise, it is not possible to relate a feature dimension in the output back to a things class of the object detector. We achieve this by matching the predicted attention masks to the ground-truth things instances, and simply re-ordering the matched ground-truth things instances such that their order corresponds to the order of the attention masks in the input tensor of the panoptic head. When training the panoptic head, we assume that we have accurate attention masks. This means that, during training, we only assign one single predicted attention mask to a ground-truth instance, and vice versa. Thus, after the attention masks are gathered, we discard the ones that do not have an Intersection over Union (IoU) greater than 0.5 with a ground-truth instance, and we assign each things instance only to the attention mask for which the IoU is highest.

The supervision for the stuff classes is the same as for a semantic segmentation problem. In our case, the stuff ground-truth is concatenated to the things ground-truth instance masks. During training, we add two additional outputs to deal with the pixels of the ground truth things instances that are not matched to any attention mask, and the unlabeled pixels.

By constructing the loss and ground-truth in this fashion, the network learns to output the things instances in the same order as the input attention masks. An example is shown in Fig. 4. Note that we effectively apply class-agnostic instance segmentation using these attention masks, and that the relevant things classes are retrieved from the object detector using the order-preserving nature of the panoptic head.

B. Backbone

For the FPSNet framework, we need a backbone that performs object detection and is able to generate a single feature map. The single feature map is necessary to make dense panoptic segmentation predictions, and the object detection network is required to output the bounding boxes that are used to generate attention masks for things instances.

1) Object Detection Network: Since FPSNet can work with various types of object detectors, and we aim to achieve fast, but also accurate, panoptic segmentation, we pick RetinaNet [20] as our object detection network. RetinaNet is a single-stage object detector that achieves state-of-the-art performance at high
inference speeds. In our implementation, we use the version of RetinaNet with a ResNet-50-based Feature Pyramid Network (FPN) as backbone [20]–[22]. This can be seen in Fig. 3.

2) Single Feature Map: The output of the ResNet-50-based Feature Pyramid Network is a set of feature maps from different levels of the feature extractor. However, to make dense panoptic segmentation predictions, we need a single feature map. In [2], the authors encountered a similar problem of performing the task of semantic segmentation on a multi-scale feature map. They solved this by upsampling and merging the different layers of the feature map, to finally generate a single feature map. For our implementation, we maintain a similar approach. As seen in Fig. 3, the output of the FPN is a set of feature maps \{P3, P4, P5, P6, P7\}, with strides \{8, 16, 32, 64, 128\}, respectively. We use feature maps P3, P4, and P5 to generate our single feature map, with a stride of 8. We apply two upsampling steps to P5 and one upsampling step to P4, creating S5 and S4, respectively. Each upsampling step consists of a 3 × 3 convolutional layer with ReLU, followed by 2 × bilinear upsampling. We get S3 by applying a 3 × 3 convolutional layer with ReLU to P3. Finally, we generate the final feature map S with \(S = S3 + S4 + S5\). Note that this is very similar to the process maintained in Panoptic FPN, except for the fact that we do not use a feature map with a stride of 4, to save computation time and resources. RetinaNet [20] applies a similar strategy to achieve more efficient object detection than FPN [22].

3) Training: The object detection head of the network is trained in the usual fashion, as explained in [20].

IV. EXPERIMENTS

We conduct the following experiments to demonstrate FPSNet and evaluate its performance:

- **Speed and accuracy:** Since FPSNet is designed for both speed and accuracy, we evaluate both at different resolutions, and compare with existing methods. For these experiments, we use the Cityscapes dataset [24].
- **Ablation study:** We conduct ablation experiments to show the effect of various design choices, i.e. attention mask shuffling, the use of hard attention masks, and tuning \(N_{\text{att}}\) and \(C_{\text{att}}\). Again, we evaluate on the Cityscapes dataset.
- **Performance on Pascal VOC:** To demonstrate the general applicability of FPSNet, we evaluate on the Pascal VOC dataset [25].

A. Metrics

We evaluate the performance of our panoptic segmentation method using the Panoptic Quality (PQ) metric [1]. This metric includes both the recognition and segmentation capabilities of the network. We also assess the performance of our network for things and stuff classes separately, through \(PQ_{\text{th}}\) and \(PQ_{\text{st}}\), respectively.

To assess the prediction speed of the network, we also measure its inference time. We report single image inference time on an Nvidia Titan RTX GPU, averaged over all images in the validation set. For methods relying on rule-based merging to generate the final panoptic segmentation output, we also provide the time required for these operations. Since FPSNet does not require such rule-based merging, the total time required for a prediction is directly given by the network inference time.

B. Datasets

We evaluate FPSNet on Cityscapes [24]. Cityscapes is a specialized dataset consisting of 5 k street scene images. It has annotations for 8 things and 11 stuff classes. To prevent overfitting, we apply a data augmentation strategy similar to the one described in [2]. We randomly scale the image with a factor between 0.5 and 1.5, and use a random crop of 512 × 1024 pixels as input to the network.

To test the applicability of FPSNet on other datasets, we also train and test on Pascal VOC [25]. Pascal VOC is a more general computer vision dataset. As in other related work [7], [11], we generate a training set by merging the Pascal VOC 2012 training set and the additional annotations from the SBD dataset [26]. This results in 10582 training images. For validation, we use the Pascal VOC 2012 validation set. This dataset has annotations for 20 things classes and no stuff classes. We randomly resize the images to square images between 512 × 512 and 800 × 800 pixels, and train on random crops of 512 × 512 pixels.

C. Implementation Details

Since FPSNet applies both object detection and panoptic segmentation, the loss function is given by

\[
L = \lambda_{\text{det}}L_{\text{det}} + \lambda_{\text{pan}}L_{\text{pan}},
\]

where \(L_{\text{det}}\) are the RetinaNet detection losses defined in [20], \(L_{\text{pan}}\) is our softmax cross-entropy loss for panoptic segmentation, and \(\lambda_{\text{det}}\) and \(\lambda_{\text{pan}}\) are the respective loss weights. In our implementation, \(\lambda_{\text{det}} = 0.5\) and \(\lambda_{\text{pan}} = 1.0\), as we found that this led to the best results. We train our network by optimizing the loss using stochastic gradient descent with a momentum of 0.9. The weight decay is 0.001. We train all networks on a single GPU, with a batch size of 4 images. We use polynomial learning rate schedule (as in [27]) with an initial learning rate of 0.001 and a power of 0.9. For the main speed and accuracy experiments, we train the network for 200 k steps. For the ablation experiments, we train all networks for 100 k steps. By default, we use \(N_{\text{att}} = 50\) and \(C_{\text{att}} = 50\); ablations are provided in Section V-B. Before training, the backbone is initialized with weights from a model pre-trained on ImageNet [28].

V. RESULTS

A. Speed and Accuracy

In Table I, we present PQ scores and prediction times for FPSNet and existing methods that report prediction times. From Table I, it follows that FPSNet is considerably faster than existing panoptic segmentation methods, while still achieving competitive scores on Panoptic Quality. Comparing with DeepLab [7], a panoptic segmentation method designed for speed and efficiency, it becomes clear that FPSNet achieves higher PQ scores at lower inference times. At a PQ score of 52.3, DeepLab is almost three times slower than FPSNet at a PQ.
score of 55.1. UPSNet [3] does score higher than FPSNet, but it is also significantly slower than our slowest implementation. In Fig. 1, the different prediction times and Panoptic Quality scores are visualized. Qualitative results on the Cityscapes validation set are shown in Figs. 2 and 7.

Additionally, we also compare FPSNet to our own re-implementation of Panoptic FPN [2]. Based on inference time alone, FPSNet is already 2× as fast. When we take the rule-based merging operations into account as well, our method becomes over 3× faster than Panoptic FPN.

### Table I

| Method                | Backbone      | Resolution | Accuracy | Inference (ms) | Rule-based merging (ms) | Total (ms) |
|-----------------------|---------------|------------|----------|----------------|-------------------------|------------|
| DeeperLab [7]         | LW-MNV2       | 512 × 1024 | 39.2     | 41             | 45                      | 86         |
| FPSNet (ours)         | ResNet-50-FPN | 512 × 1024 | 46.7     | 45             | n/a                     | 45†        |
| DeeperLab [7]         | LW-MNV2       | 1024 × 2048| 48.1     | 97             | 154                     | 251†       |
| DeeperLab [7]         | W-MNV2        | 1024 × 2048| 52.3     | 149            | 154                     | 303†       |
| DeeperLab [7]         | Xception-71   | 1024 × 2048| 56.5     | 308            | 154                     | 462†       |
| UPSNet [3]            | ResNet-50-FPN | 1024 × 2048| 59.3     | 176            | n/a                     | 176†       |
| Panoptic FPN (ours)   | ResNet-50-FPN | 1024 × 2048| 57.0     | 226            | 152                     | 378†       |
| FPSNet (ours)         | ResNet-50-FPN | 1024 × 2048| 55.1     | 114            | n/a                     | 114†       |

1 Evaluated on the same machine; inference with an Nvidia Titan RTX GPU. 2 Timing taken from [7]; inference with an Nvidia V100 GPU. (L)W-MNV2 is (Light) Wider MobileNetV2.

### Table II

| Method                | Backbone      | PQ   | PQ50 | PQ80 | Time (ms) |
|-----------------------|---------------|------|------|------|-----------|
| De Geus et al. [24]   | RN-50         | 45.9 | 39.2 | 50.8 | -         |
| Li et al. [11]        | RN-101        | 47.5 | 39.6 | 52.9 | -         |
| TASCA-Net [5]         | RN-50-FPN     | 55.9 | 50.5 | 59.8 | -         |
| DeeperLab [7]         | XC-71         | 56.3 | 52.7 | 59.0 | 462       |
| AU-Net [6]            | RN-50-FPN     | 56.4 | 52.7 | 59.0 | -         |
| Panoptic FPN [2]      | RN-50-FPN     | 57.7 | 51.6 | 62.2 | -         |
| UPSNet [3]            | RN-50-FPN     | 59.3 | 54.6 | 62.7 | 176       |
| Porzi et al. [4]      | RN-50-FPN     | 60.2 | 55.6 | 63.6 | -         |
| FPSNet (ours)         | RN-50-FPN     | 55.1 | 48.3 | 60.1 | 114       |

### B. Ablation Study

We conduct several ablation experiments on the Cityscapes validation set. We evaluate the method using both the original attention masks, gathered from the detection branch output, and attention masks generated using ground-truth bounding boxes. We use these ground-truth bounding boxes for a fair analysis of the performance of the panoptic head.

1) **Attention Mask Shuffling:** To show that attention mask shuffling boosts the performance, we conduct an experiment without shuffling. From Table III, it follows that the performance for things classes increases when we introduce attention mask shuffling. The gap is 2 points when the original attention masks are used, but increases to 15 points when using ground-truth bounding boxes as attention masks. In the latter case, there are more attention masks, and they are ordered differently. This shows that not all filters of the convolutional layers in the panoptic head receive adequate supervision, and that the network learns a specific order of things instances instead.

2) **Hard Attention Masks:** We replace our soft attention masks with hard attention masks, where all pixels within the
TABLE III
SEVERAL ABLATIONS ON CITYSCAPES VALIDATION SET

| Attention Mask Shuffling | Hard Attention Masks | GT Bounding Boxes | PQ  | PQ_{Th} | PQ_{St} |
|-------------------------|----------------------|-------------------|-----|---------|---------|
| -                       | -                    | -                 | 53.1 | 44.7    | 39.4    |
| ✓                       | -                    | -                 | 54.1 | 46.7    | 39.5    |
| ✓                       | ✓                    | -                 | 52.7 | 43.7    | 39.2    |
| ✓                       | -                    | ✓                 | 30.5 | 39.7    | 38.4    |
| ✓                       | ✓                    | ✓                 | 57.5 | 54.7    | 59.5    |
| ✓                       | ✓                    | ✓                 | 61.1 | 51.5    | 59.4    |

TABLE IV
ABLATIONS FOR N_{att} AND C_{att} ON CITYSCAPES VALIDATION SET

| N_{att} | C_{att} | PQ  | PQ_{Th} | PQ_{St} | Time (ms) |
|---------|---------|-----|---------|---------|-----------|
| 50      | 50      | 54.1 | 46.7    | 39.5    | 114       |
| 25      | 50      | 53.7 | 45.8    | 39.5    | 109       |
| 100     | 50      | 52.6 | 44.5    | 38.5    | 128       |
| 50      | 25      | 52.6 | 43.9    | 38.9    | 114       |
| 50      | 100     | 53.2 | 45.7    | 58.6    | 114       |

bounding box get the value C_{att}. As expected, the results in Table III show that using hard attention masks reduces the performance for things classes, given that PQ_{Th} is reduced by 3.0 points.

3) N_{att} and C_{att}: We train FPSNet with different values for N_{att} and C_{att} and report the results in Table IV. We find that changing the number of attention masks, i.e. N_{att}, has a slight effect on the performance. Using 25 attention masks instead of 50 slightly decreases the scores on things classes, which is caused by a lower performance on images with more than 25 things instances. With N_{att} = 100, the performance drops for both stuff and things classes. In this case, N_{att} appears to be much larger than the amount of things instances actually present in the image, which means that many output channels of the panoptic head are not supervised with a ground-truth things instance. This hinders the learning process, and leads to a lower performance. From the results, it also follows that C_{att} = 50 seems to be the optimal value. In this case, there is a good balance between the magnitude of the attention mask tensors and the features from the feature map. It is likely that there are better ways to create this balance, e.g. with various normalization techniques, but we leave this for future work to address.

When assessing the prediction times in Table IV, it can be seen that changing the value of C_{att} has no effect. This is as expected, as this is merely a multiplication factor that is applied to the attention masks. Changing N_{att} does have an effect: the higher N_{att}, the higher the prediction time. This shows that there is a trade-off between speed and accuracy for lower values of N_{att}.

C. Performance on Pascal VOC

We evaluate our results on the Pascal VOC 2012, and compare with other methods in Table V, in terms of PQ and total prediction time. Again, it is clear that FPSNet is much faster than DeeperLab [7], whilst achieving competitive PQ scores. Since this dataset only consists of things classes, and hence the balance between stuff and things classes is completely different than for Cityscapes, it is possible that changing hyperparameters can substantially improve the performance. Qualitative results on the Pascal VOC 2012 validation set are shown in Fig. 8. In the bottom two rows of this figure, it can be seen that our method also works for objects with non-compact shape, even though the weight of a soft attention mask is in the center of the bounding box. Moreover, the performance for Pascal VOC drops from 57.8 to 54.9 when using hard attention masks instead of soft masks, similar to the drop for Cityscapes in Table III.

VI. CONCLUSIONS

In this work, we presented FPSNet, an end-to-end framework for fast panoptic segmentation. FPSNet makes dense panoptic segmentation predictions in a fashion that does not require computationally expensive instance mask predictions or rule-based merging operations. This is facilitated by a novel panoptic
head design and a tailored panoptic training strategy. With extensive experiments, we have shown that FPSNet is faster than existing state-of-the-art panoptic segmentation networks, and is able to achieve inference times as low as 28 milliseconds for images of $512 \times 512$ pixels, which is equivalent to a real-time frame rate of 35 frames per second. While being fast, FPSNet also achieves a competitive Panoptic Quality score of 55.1 on the Cityscapes validation set. With this work, we have made a significant step in bringing high-quality panoptic segmentation to real-time applications in robotics and intelligent vehicles.

REFERENCES

[1] A. Kirillov, K. He, R. Girshick, C. Rother, and P. Dollár, “Panoptic segmentation,” in Proc. IEEE Conf. Comput. Vision Pattern Recognit., Jun. 2019, pp. 9396–9405.
[2] A. Kirillov, R. Girshick, K. He, and P. Dollár, “Panoptic feature pyramidal networks,” in Proc. IEEE Conf. Comput. Vision Pattern Recognit., Jun. 2019, pp. 6392–6401.
[3] Y. Xiong et al., “UPSNet: A unified panoptic segmentation network,” in Proc. IEEE Conf. Comput. Vision Pattern Recognit., Jun. 2019, pp. 8810–8818.
[4] L. Porzi, S. R. Bulo, A. Colovic, and P. Kontschieder, “Seamless scene segmentation,” in Proc. IEEE Conf. Comput. Vision Pattern Recognit., Jun. 2019, pp. 8269–8278.
[5] J. Li, A. Raventos, A. Bhargava, T. Tagawa, and A. Gaidon, “Learning to fuse things and stuff,” Dec. 2018, arXiv:1812.01192.
[6] Y. Li, “Attention-Guided unified network for panoptic segmentation,” in Proc. IEEE Conf. Comput. Vision Pattern Recognit., Jun. 2019, pp. 7019–7028.
[7] T.-J. Yang et al., “DeeperLab: Single-Shot image parser,” 2019, arXiv:1905.05993.
[8] A. Arnab and P. H. S. Torr, “Pixelwise instance segmentation with a dynamically instantiated network,” in Proc. IEEE Conf. Comput. Vision Pattern Recognit., Jul. 2017, pp. 879–888.
[9] S. Fidler, J. Yao, and R. Urtasun, “Describing the scene as a whole: Joint object detection, scene classification and semantic segmentation,” in Proc. IEEE Conf. Comput. Vision Pattern Recognit., Jun. 2012, pp. 702–709.
[10] Z. Tu, X. Chen, A. Yuille, and S. Zhu, “Image parsing: Unifying segmentation, detection, and recognition,” Int. J. Comput. Vision, vol. 63, no. 2, pp. 113–140, 2005.
[11] Q. Li, A. Arnab, and P. Torr, “Weakly- and semi-supervised panoptic segmentation,” in Proc. Eur. Conf. Comput. Vision, Sep. 2018, pp. 106–124.
[12] D. de Geus, P. Meletis, and G. Dubbelman, “Panoptic segmentation with a joint semantic and instance segmentation network,” Sep. 2018, arXiv:1809.02110.
[13] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask R-CNN,” in Proc. IEEE Conf. Comput. Vision, Oct. 2017, pp. 2980–2988.
[14] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, “Pyramid scene parsing network,” in Proc. IEEE Conf. Comput. Vision Pattern Recognit., Jul. 2017, pp. 6230–6239.
[15] H. Liu et al., “An End-To-end network for panoptic segmentation,” in Proc. IEEE Conf. Comput. Vision Pattern Recognit., Jun. 2019, pp. 6165–6174.
[16] A. Kirillov, E. Levinkov, B. Andres, B. Savekhynskyi, and C. Rother, “InstanceCut: From edges to instances with multicut,” in Proc. IEEE Conf. Comput. Vision Pattern Recognit., Jul. 2017, pp. 7322–7331.
[17] J. Uhrig, M. Cordts, U. Franke, and T. Brox, “Pixel-level encoding and depth layering for instance-level semantic labeling,” in Proc. German Conf. Pattern Recognit. Springer, 2016, pp. 14–25.
[18] V. Nair and G. Hinton, “Rectified linear units improve restricted boltzmann machines,” in Proc. 27th Int. Conf. Mach. Learn., 2010, pp. 807–814.
[19] S. Ioffe and C. Szegedy, “Batch normalization: accelerating deep network training by reducing internal covariate shift,” in Proc. 32nd Int. Conf. Mach. Learn., 2015, pp. 448–456.
[20] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, “Focal loss for dense object detection,” in Proc. IEEE Int. Conf. Comput. Vision, Oct. 2017, pp. 2999–3007.
[21] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE Conf. Comput. Vision Pattern Recognit., Jun. 2016, pp. 770–778.
[22] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, “Feature pyramid networks for object detection,” in Proc. IEEE Conf. Comput. Vision Pattern Recognit., Jul. 2017, pp. 936–944.
[23] D. de Geus, P. Meletis, and G. Dubbelman, “Single network panoptic segmentation for street scene understanding,” in Proc. IEEE Intell. Vehicles Symp. (IV), Jun. 2019, pp. 709–715.
[24] M. Cordts et al., “The cityscapes dataset for semantic urban scene understanding,” in Proc. IEEE Conf. Comput. Vision Pattern Recognit., Jun. 2016, pp. 3213–3223.
[25] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes (VOC) challenge,” Int. J. Comput. Vision, vol. 88, no. 2, pp. 303–338, Jun. 2010.
[26] B. Hariharan, P. Arbelaez, L. Bourdev, S. Maji, and J. Malik, “Semantic contours from inverse detectors,” in Proc. Int. Conf. Comput. Vision, 2011, pp. 991–998.
[27] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, “DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 40, no. 4, pp. 834–848, Apr. 2018.
[28] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei, “ImageNet: A large-scale hierarchical image database,” in Proc. IEEE Conf. Comput. Vision Pattern Recognit., Jun. 2009, pp. 248–255.