FASSST: Fast Attention Based Single-Stage Segmentation Net for Real-Time Instance Segmentation

Anonymous WACV submission

Paper ID 760

Abstract

*Real-time instance segmentation is crucial in various AI applications. This work designs a network named **F**ast **A**ttention based **S**ingle-**S**tage **S**egmentation Ne**T** (FASSST) that performs instance segmentation with video-grade speed. Using an* instance attention module (IAM)*, FASSST quickly locates target instances and segments with* region of interest (ROI) feature fusion (RFF) *aggregating ROI features from* pyramid mask layers*. The module employs an efficient* single-stage feature regression*, straight from features to instance coordinates and class probabilities. Experiments on COCO and CityScapes datasets show that FASSST achieves state-of-the-art performance under competitive accuracy: real-time inference of* 47.5*FPS on a GTX1080Ti GPU and* 5.3*FPS on a Jetson Xavier NX board with only* 71.6*GFLOPs.*

1. Introduction

Various computer vision applications, such as object detection and semantic segmentation, have undergone remarkable progress in recent years [5, 11, 6]. Nevertheless, as a more complex task, instance segmentation requires precise locations and semantic masks of all instances in a frame, which still remains a great challenge especially its implementation on resource-constrained edge/terminal devices. Modern researches on instance segmentation mainly fall into two categories: **i)** Pixel-wise approach [10, 12] which learns an affinity relation between image pixels and segments image by segregating pixels of different instances and grouping pixels of same instance. However, a *post-processing* is needed to separate instances, leading to unnecessary computational complexity and low speed. **ii)** Proposal-based approach [13, 9] which first proposes object candidates by bounding boxes, then selects interested ones of them, and at last performs masking. This strategy avoids handling all pixels of an image, but still requires *multiple steps* of computationally expensive candidate proposal. Also, a large amount of segmentation time is wasted on the unadopted candidates or overlapped areas, making it hard to achieve a real-time speed.

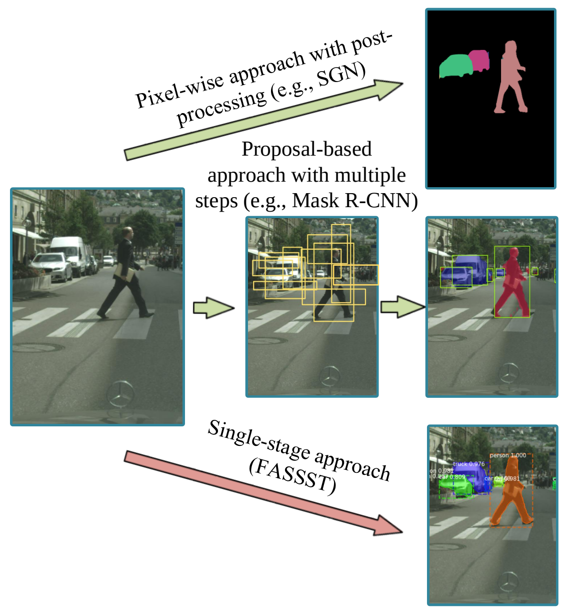

Figure 1. Different instance segmentation approaches.

To overcome these hurdles, we design FASSST (**F**ast **A**ttention-based **S**ingle-**S**tage **S**egmentation Ne**T**) for real-time instance segmentation. The contributions are threefold: **1)** an *instance attention module (IAM)* is devised to locate and segment the target instances, instead of learning pixel-wise relations or proposing object candidates; **2)** a *single-stage feature regression* strategy that produces instance coordinates and class probabilities straight from features is used for *video speed* signal processing; **3)** segmentation is done via a *region of interest (ROI) feature fusion (RFF)*, aggregating ROI features from the *pyramid mask layers* and delivering competitive accuracy with fewer layers.

Figure 1 compares several related works and highlights the difference of the proposed FASSST. Experimental results on COCO [21] and CityScapes [7] show that FASSST achieves state-of-the-art performance under competitive accuracy: real-time inference of 47.5FPS on a GTX1080Ti

WACV #760

GPU and 5.3FPS on a Jetson Xavier NX board with only 71.6GFLOPs. In what follows, Section 2 reviews some related works. Section 3 illustrates the FASSST design. Section 4 presents experiments on two large-scale datasets and Section 5 draws the conclusion.

2. Related Instance Segmentation Work

Pixel-wise: Existing pixel-wise approaches for instance segmentation are usually realized by grouping instance pixels into an arbitrary number of instances. Recent work [10] proposes a discriminative loss function to learn pixel-wise relations by pushing away pixels belonging to different instances and grouping pixels in the same instance. Later, SSAP [12] uses a pixel-pair affinity pyramid to group two pixels each time. And SGN [22] reframes the instance segmentation problem into a sequence of sub-grouping problems. However, these methods suffer from unsatisfactory accuracy and speed due to their per-pixel grouping and expensive post-processing.

Proposal-based: Driven by the advancement of object detection networks, recent works perform instance segmentation with R-CNN to first propose object candidates and then segment interested ones of them. The work in [8] utilizes the shared convolutional features among object candidates in segmentation layers. DeepMask [25] is developed for learning mask proposals based on Fast R-CNN. Multi-task cascaded network [9] is developed with an instance-aware semantic segmentation on object candidates. Mask R-CNN [13] is developed as the extension of Faster R-CNN with a mask branch. All these approaches require *multiple steps* that first generate object candidates, then segment interested ones of them, and at last detect and recognize the correct ones. Apparently, such object proposal methods waste unnecessary computation on the unadopted candidates and overlapped areas of candidates.

Single-stage: Lately, there are attempts to produce a single-stage instance segmentation [3, 29, 17, 27, 4]. FCIS [18] assembles the position-sensitive score maps within the ROI to directly predict instance masks. YOLACT [2] tries to combine the prototype masks and predicted coefficients and then crops with a segmented bounding box. PolarMask [30] introduces the polar representation to formulate pixel-wise instance segmentation as a distance regression problem. SOLO [28] divides network into two branches to generate instance segmentation with predicted object locations. However, they still require significant amounts of pre- or post-processing before or after localization.

3. FASSST

We now elaborate FASSST that leverages an instance attention module (IAM) to achieve a single-stage real-time instance segmentation. There are three main design goals: small size, high speed and high accuracy.

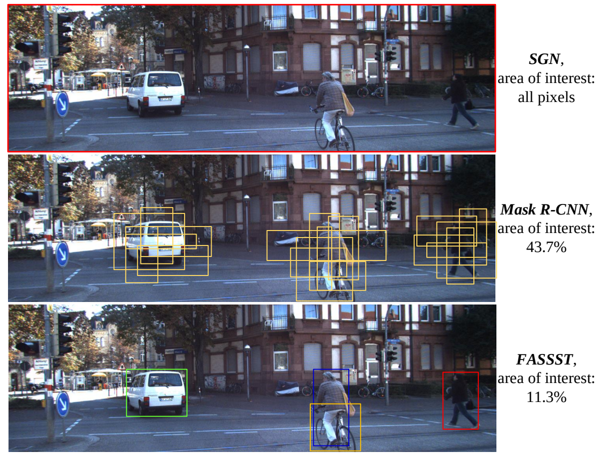

Figure 2. Comparison of area of interest among different instance segmentation schemes.

3.1. Observations

Instance segmentation usually requires the correct separation of all parts in a frame. In practical application such as autonomous driving and robotics, to precisely detect the free-space area and predict trajectories, the frames are of high-resolution (e.g., 2048×1024), which contain a large number of pixels. We divide frame pixels into two parts: **i)** Target object pixels, which are important but practically minority in frames. **ii)** Background pixels, which are the majority in most situations. This implies significant processing time can be saved if instances in a frame can be quickly and precisely located. The proportions of area of interest among different approaches are compared in Figure 2 wherein we calculate the proportions by: *area of interest/frame size*, it can be seen the former two approaches need to handle much more area than in FASSST.

With such analysis, we present the full architecture of FASSST in Figure 3. Assuming $\boldsymbol{F} \in \mathbb{R}^{W_f \times H_f \times D_f}$ is a frame, where W, H and D represent the mode dimensions. First, we use several front convolutional layers of the network backbone as "network head" to extract raw features $\boldsymbol{E} \in \mathbb{R}^{W_e \times H_e \times D_e}$ of the whole frame. The specific settings of network head will be further analyzed in the experiment section. Then, the feature tensor $\boldsymbol{E}$ is parallelly delivered into the following layers and the IAM. The IAM is applied to learn instance information tensor $\boldsymbol{I} \in \mathbb{R}^{W_i \times H_i \times D_i}$, including instance coordinates and class probabilities, from raw features. Next, we use the instance information to locate ROIs on several pyramid mask layers and obtain the fused ROI features $\boldsymbol{R} \in \mathbb{R}^{W_r \times H_r \times D_r}$ by an ROI feature fusion module (RFF). Note that the fused ROI feature tensor carries both local and global context information. Finally, the representation $\boldsymbol{R}$ is fed into the subsequent small-size convolutional layers to get the final instance segmentation

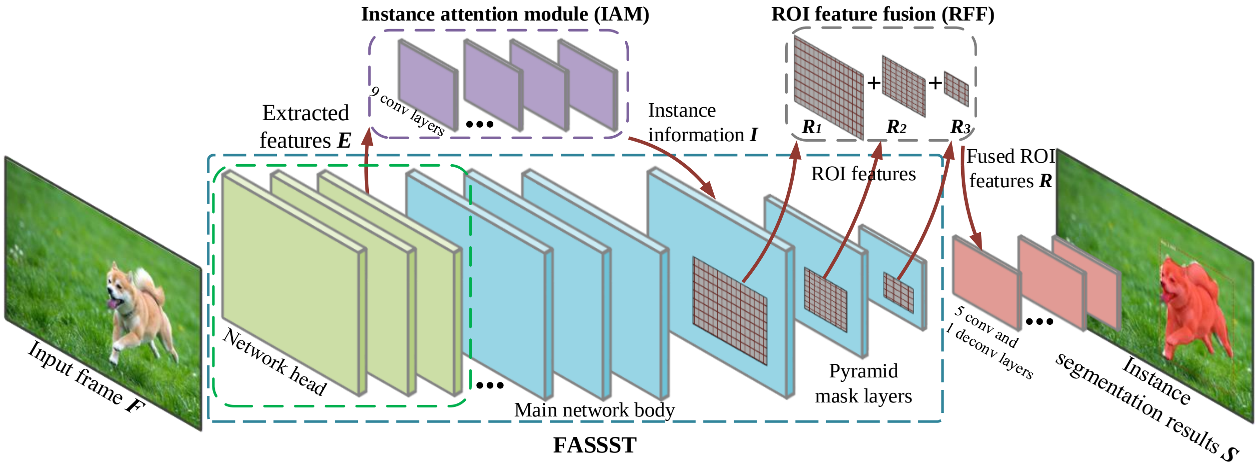

Figure 3. Overview of the FASSST architecture.

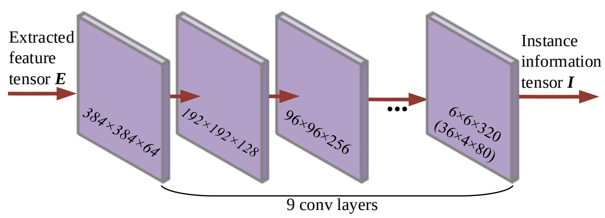

Figure 4. Instance attention module (IAM) architecture.

results $\boldsymbol{S} \in \mathbb{R}^{W_s \times H_s}$.

3.2. Instance Attention Module

The feature regression schemes for object detection (e.g., YOLO [26] and SSD [23]) have been proposed to learn structured output regression to localize instances and proved to be efficient. Similarly, in the proposed IAM module, we regard the instance attention as a single-stage regression problem and *directly* learn instance coordinates and class probabilities from raw features. First, the raw feature tensor $\boldsymbol{E}$ is generated by the network head:

$$\boldsymbol{E} = extr(\boldsymbol{F}), \tag{1}$$

where $extr$ denotes the feature extractor to extract raw features from image pixels. Then, as shown in Figure 3, the IAM further produces the instance locality information $\boldsymbol{I}$:

$$\boldsymbol{I} = attn(\boldsymbol{E}), \tag{2}$$

where $attn$ represents the instance attention process. $attn$ regards the instance attention as a single-stage regression problem, which directly learns instance locality information $\boldsymbol{I}$ from raw features $\boldsymbol{E}$ [26]. Specifically, $\boldsymbol{I}$ is structured as an $n \times c \times s$ tensor (that is $W_i = n$, $H_i = c$ and $D_i = s$), where n is the largest number of instances for each frame which varies for different datasets (e.g., in the COCO experiments we set $n = 36$), c denotes 4 coordinate predictions of an instance: top-most t, left-most l, bottom-most b, right-most r, and dimension of s being the number of classes and respective confidence scores of class probabilities are stored

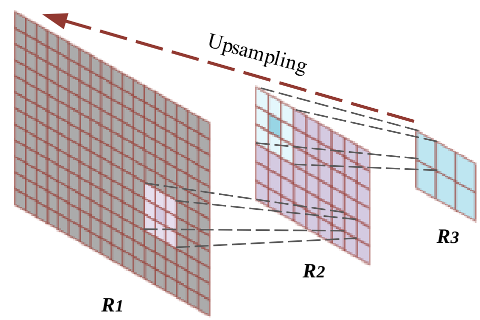

Figure 5. ROI feature fusion architecture.

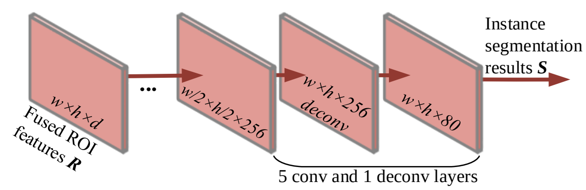

Figure 6. Mask generation progress.

along the s-axis. These trained coefficients can provide accurate instance coordinates and class probabilities for a frame. Some detailed sizes of the adopted convolutional layers are specified in Figure 4. The particular settings of layer scales and depths for network head, IAM, and later mask "tail" will be discussed in Section 4.3. The instance information $\boldsymbol{I}$ will be fed back to the main network body and applied to locate the ROI features. It should be noted that the overlapped areas of instances are multi-time processed in FASSST.

3.3. ROI Feature Fusion

After obtaining the important instance coordinates, the ROI features can be located on layers. First, we apply a series of pyramid mask layers to exploit deep features. Note that it has been proved that the shallow layers explore more on the instance contours, while the deeper layers focus on the full instances [31, 19]. Then, we employ the RFF to fuse the features from ROIs of the pyramid mask layers. The fused ROI features carry both local (instance core) and

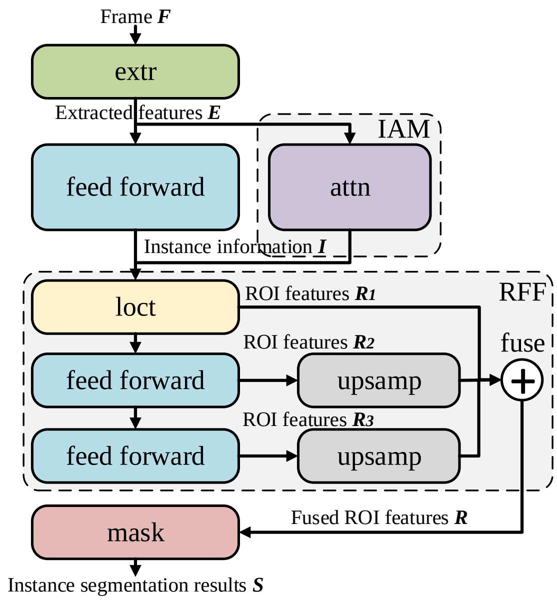

Figure 7. Workflow of FASSST.

global (instance contour) context information, delivering a high accuracy. Inside the RFF, we apply upsampling and $ReLU$ operations to aggregate different features. As shown in Figure 5, the input to this module contains three ROI features $\boldsymbol{R_1}$, $\boldsymbol{R_2}$ and $\boldsymbol{R_3}$. Note that $\boldsymbol{R_2}$ has a spatial size double that of $\boldsymbol{R_3}$, and $\boldsymbol{R_1}$ is double that of $\boldsymbol{R_2}$. To form these, we first perform upsampling on $\boldsymbol{R_3}$ with a rate of 2 through bilinear interpolation:

$$\boldsymbol{R_3^u} = upsamp(\boldsymbol{R_3}), \tag{3}$$

where $\boldsymbol{R_3^u}$ is the upsampled $\boldsymbol{R_3}$ feature tensor and $upsamp$ represents the corresponding upsampling function. Then, we upsample $\boldsymbol{R_3^u}$ to $\boldsymbol{R_2}$ and apply the $upsamp$ operation again. Finally, the fused feature tensor $\boldsymbol{R}$ is processed by:

$$\begin{aligned} \boldsymbol{R} =& fuse(\boldsymbol{R_1}, \boldsymbol{R_2}, \boldsymbol{R_3}) \\ =& ReLU(\boldsymbol{R_1} + upsamp(\boldsymbol{R_2} + upsamp(\boldsymbol{R_3}))), \end{aligned} \tag{4}$$

where $fuse$ denotes the feature fusion function. Note that a $ReLU$ function is further applied to refine the upsampled features.

3.4. Mask Generation

To generate instance masks from the fused ROI features, we further apply several small-size convolutional layers as the "tail" part of the framework, as shown in Figure 6. We use 5 convolutional layers and 1 deconvolution layer to learn the mask representation. Assuming the size of fused ROI feature tensor $\boldsymbol{R}$ is $w \times h \times d$, we use 5 convolutional layers and 1 deconvolution layer to learn the mask representation. With the mask outputs produced, we can obtain the final instance segmentation results $\boldsymbol{S}$ of the proposed framework FASSST. The whole workflow of FASSST is summarized in Figure 7, where $loct$ represents the ROI localization process, and $mask$ represents the final mask generation.

Algorithm 1 *Forwards Propagation of FASSST Training*

Require: Frame data $\boldsymbol{F}$, training epoch T.
Ensure: Training accuracy $\boldsymbol{P}$.

```
for k = 1 : T do
    Feed F into the network head
    Obtain the extracted features E^k ← extr(F)
    Feed E^k to IAM
    I^k ← attn(E^k), parallelly process the main network
        body to pyramid mask layers
    Locate the ROIs: R_1^k, R_2^k, R_3^k ← loct(I^k)
    RFF: R^k ← fuse(R_1^k, R_2^k, R_3^k)
    Feed R^k to mask "tail": S^k ← mask(R^k)
end for
Get the training accuracy P
```

3.5. Training Strategy

The forward propagation of FASSST training is presented in Algorithm 1. Different from the two- or multi-stage training of proposal-based instance segmentation approaches, the training of FASSST is a single-stage end-to-end process.

The loss function in backward propagation of FASSST training is built on mask loss L_m, localization loss L_l and classification loss L_c:

$$L = \lambda_m L_m + \lambda_l L_l + \lambda_c L_c, \tag{5}$$

where L is the total loss, λ_m, λ_l and λ_c are set as 5.75, 3 and 1.25, respectively. Specifically, the L_m is based on Dice Loss [15]:

$$L_m = 1 - Dice(mask_p, mask_g), \tag{6}$$

where $Dice$ is the corresponding function for dice coefficients, $mask_p$ and $mask_g$ are predicted masks and ground truth masks, respectively. Moreover, L_l and L_c are based on the conventional Focal Loss [20].

4. Experiments

We present a thorough evaluation and ablation study of the proposed FASSST. Our experimental setup employs Caffe for coding; a single NVIDIA GTX-1080Ti GPU card for hardware realization; and an NVIDIA Jetson Xavier NX board for terminal implementation. Benchmarking is made on two instance segmentation datasets: COCO and CityScapes. Note that in all comparisons, the accuracy and efficiency data of some open source models are practically evaluated in our machine. Moreover, the plain FASSST represents main network body with MobileNet-54-V2 backbone and network head with input frame scale 416×416. We will emphasize by suffix if different settings are used. All these settings will be further discussed in Section 4.3 on ablation study.

Figure 8. Sample visual results of FASSST on COCO.

Category	Approach	Backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Pixel-wise	SGN [22]	-	25.0	44.9	25.8	-	-	-
	SSAP [12]	ResNet-101-FPN	29.4	48.1	28.8	-	28.6	-
Proposal-based	FCIS [18]	ResNet-101-C5-dilated	29.2	49.5	-	7.1	31.3	50.0
	FCIS+++ [18]	ResNet-101-C5-dilated	33.6	54.5	**37.9**	-	-	-
	MNC [9]	ResNet-101-C4	24.6	44.3	24.8	4.7	25.9	43.6
	Mask R-CNN [13]	ResNet-101-FPN	**35.7**	**58.0**	37.8	**15.5**	**38.1**	52.4
Single-stage	ExtremeNet [32]	Hourglass-104	18.9	44.5	13.7	10.4	20.4	28.3
	YOLACT [2]	ResNet-101-FPN	31.2	50.6	32.8	12.1	33.3	47.1
	SOLO [28]	ResNet-101-FPN	**37.8**	**59.5**	**40.4**	**16.4**	**40.6**	**54.2**
	SipMask [17]	ResNet-101-FPN	32.8	53.4	34.3	9.3	35.6	**54.0**
	CenterMask [17]	ResNet-50-FPN	32.9	-	-	12.9	34.7	48.7
	PolarMask [30]	ResNet-101-FPN	30.4	51.9	31.0	13.4	32.4	42.8
Proposed	**FASSST**	MobileNet-54-V2	**34.2**	**56.4**	**38.1**	**14.9**	**36.7**	**53.8**

i) - represents not reported or no open source for evaluation.
ii) **red**: ranking 1^{st}; **yellow**: ranking 2^{nd}; **blue**: ranking 3^{rd}.

Table 1. Accuracy comparison with state-of-the-arts on COCO.

4.1. Evaluation on COCO

We first train and evaluate FASSST with the COCO2017 segmentation benchmark that involves 80 foreground instance classes and one background class. The original dataset contains 118K (train) and 41K (test) instance pixel-level labeled images. Specifically, we perform training on *train2017* and evaluation on *test-dev*. Using a batch size as 8, epochs as 100 and a learning rate as 0.005, each full training on COCO costs $3 \sim 4$ days. Some visual results by FASSST are shown in Figure 8 where we sample a wide range of instance sizes. It is observed that existing instances are located and segmented in the frames by FASSST.

4.1.1 Accuracy Analysis

The accuracy of FASSST on COCO is measured in terms of the standard average precision (AP) metrics, namely, AP_{50},

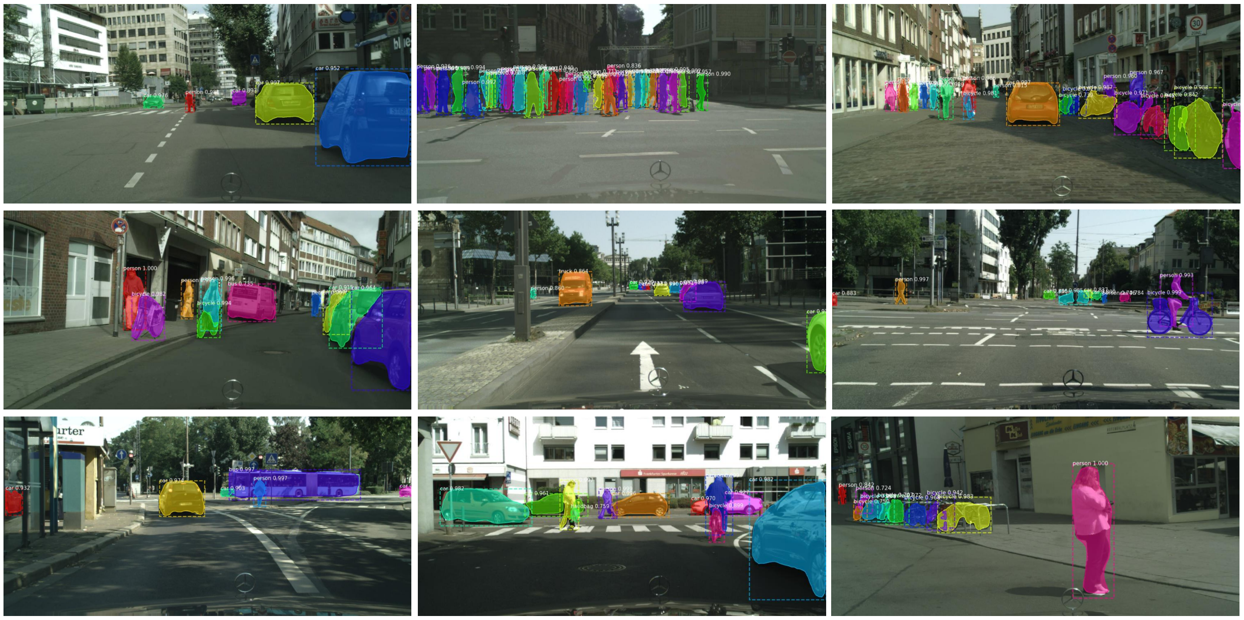

Figure 9. Sample visual results of FASSST on Cityscapes.

AP_{75}, and AP_S, AP_M, AP_L. Here AP_{50} and AP_{75} represent the AP scores over IoU thresholds at 0.5 and 0.75, respectively; AP_S, AP_M and AP_L denote the AP scores for small objects (area$< 32^2$), medium objects ($32^2 <$area$< 96^2$) and large objects (area$> 96^2$), respectively. In Table 1, we report the accuracy comparison on COCO between FASSST and state-of-the-art pixel-wise and proposal-based models. We conclude that FASSST can achieve competitive accuracy as well as video speed using the *more compact backbone (MobileNet-54-V2)* for the main network body. The average AP of FASSST reaches 34.2, which outperforms various state-of-the-arts and is only slightly lower than Mask R-CNN and SOLO. We argue that FASSST will obtain higher accuracy if the same complex backbones (e.g., ResNet-101-FPN and ResNeXt-101-FPN) or same training tricks (e.g., multi-scale train/test) are adopted.

4.1.2 Efficiency Analysis

Here we evaluate the inference speed, computational complexity and storage of FASSST. Table 2 compares the FPS (frames per second), FLOPs (floating-point operations per second) and storage size between FASSST and state-of-the-arts. Note that all listed results are practically measured on one single GTX-1080Ti card. In particular, FASSST exhibits a major niche in the inference speed which reaches 59.2FPS and is 5.7× faster than the popular Mask R-CNN. This video-grade speed can be considered to be *"very fast"* for instance segmentation. Also, the proposed framework requires the least FLOPs (71.6G) and storage (36.3MB) among all schemes, which are 3.8× and 6.7× smaller than the Mask R-CNN, respectively.

Approach	FPS	FLOPs (G)	Storage (MB)
SSAP [12]	5.5	-	-
FCIS [18]	6.2	364.1	**207.0**
Mask R-CNN [13]	10.3	273.6	242.3
RetinaMask [20]	6.8	358.3	423.6
MS R-CNN [14]	11.5	-	-
YOLACT-550 [2]	**41.7**	**97.3**	**121.8**
SOLO [28]	22.5	-	422.0
PolarMask-400 [30]	**23.1**	**248.7**	409.3
FASSST	**59.2**	**71.6**	**36.3**

Table 2. Efficiency comparison with state-of-the-arts on COCO.

4.2. Evaluation on CityScapes

We further test FASSST on the CityScapes, a large-scale dataset with high quality pixel-level annotations of 5000 images of 2048 × 1024 resolution collected in street scenes from 50 different cities. Following the evaluation protocol for instance segmentation, we select 8 instance labels for training: *person*, *rider*, *car*, *truck*, *bus*, *train*, *motorcycle* and *bicycle* (belonging to two super categories: *human* and *vehicle*, and all other labels are considered as background), which are regarded as the most important classes in autonomous driving. The training and testing sets contain 2975 and 1525 images, respectively. Sample visual instance segmentation results on CityScapes are presented in Figure 9. Again, we conclude that FASSST can accurately locate and mask the designated instances, even for crowds in the distance.

4.2.1 Accuracy Analysis

We evaluate the standard metrics AP and AP_{50}, which are the same with COCO experiments, and individual AP scores for every instance class. Here we present the accuracy com-

Approach	AP	AP_{50}	*person*	*rider*	*car*	*truck*	*bus*	*train*	*motorcycle*	*bicycle*
InstanceCut [16]	13.0	27.9	10.0	8.0	23.7	14.0	19.5	15.2	9.3	4.7
SGN [22]	25.0	44.9	21.8	20.1	39.4	24.8	33.2	30.8	17.7	12.4
SegNet [1]	29.5	**55.6**	29.9	23.4	43.4	**29.8**	**41.0**	**33.3**	18.7	**16.7**
SSAP [12]	**32.7**	51.8	**35.4**	**25.5**	**55.9**	**33.2**	**43.9**	**31.9**	**19.5**	16.2
Mask R-CNN [13]	26.2	49.9	30.5	23.7	46.9	22.8	32.2	18.6	19.1	16.0
Mask R-CNN[COCO] [13]	**32.0**	**58.1**	**34.8**	**27.0**	**49.1**	**30.1**	**40.9**	**30.9**	**24.1**	**18.7**
GMIS [24]	27.3	45.6	31.5	25.2	42.3	21.8	37.2	28.9	18.8	12.8
FASSST-768[COCO]	**31.1**	**56.2**	**34.5**	**26.8**	**49.9**	28.7	38.3	27.8	**24.2**	**18.7**

"[COCO]" means with pretrained COCO model.

Table 3. Accuracy comparison with state-of-the-arts on CityScapes.

Approach	FPS	FLOPs (G)	Storage (MB)
SegNet [1]	2.4	604.7	**112.0**
SSAP [12]	3.4	-	-
Mask R-CNN [13]	6.9	463.5	245.6
YOLACT-700 [2]	**21.7**	**214.3**	**192.0**
PolarMask-800 [30]	**18.3**	**324.8**	705.4
FASSST-768	**47.5**	**112.8**	**43.7**

Table 4. Efficiency comparison with state-of-the-arts on CityScapes.

parison on CityScapes between FASSST and state-of-the-art methods in Table 3. The proposed FASSST with lightweight MobileNet-54-V2 backbone outperforms various state-of-the-arts on all AP scores.

4.2.2 Efficiency Analysis

We further report efficiency analysis of FASSST on CityScapes. As shown in Table 4, FASSST achieves 47.5FPS on the a single GTX1080Ti GPU, which is a 2.2× speedup versus the representative single-stage instance segmentation method YOLACT. The FLOPs and model size of FASSST are only 112.8G and 41.3MB, i.e., 1.9× and 4.6× smaller than YOLACT, respectively. In addition, we further evaluate FASSST on a terminal device of NVIDIA Jetson Xavier NX board, the inference speed achieves remarkable 5.3FPS. Therefore, we conclude that FASSST provides a real-time and hardware-friendly instance segmentation for edge computing.

4.3. Ablation Study

We run a series of ablations to further analyze FASSST. Note that all experiments are evaluated on COCO and CityScapes with the same software-hardware setting.

4.3.1 Network Head

The first concern arises from the beginning of network. As the network head extracts important features for the subsequent parts, the input frame scale and depth should be investigated. In Table 5, we compare different heads' scales and depths. At a frame scale of 416, changing the head depth from 4 to 5 provides 3.7 AP gains while 5 to 6 provides 0.1 AP gains and the accuracy becomes stable. Therefore, we conclude that 5 is the best choice for layer depth of network head in the main network body. Next, setting depth to be 5, changing input frame scale from 416 to 768 provides 2.2 AP gain, and causes 11.7FPS loss. In practice, we keep both scales for network head and apply FASSST-416 as the default, and enable FASSST-768 when the frame sizes are large (e.g., in CityScapes). Note that same investigation of depths has been thoroughly performed on the IAM and mask "tail" modules, and hence we determine the current settings (9 conv layers for the IAM, and 5 conv and 1 deconv layers for the mask "tail").

4.3.2 Backbone Architecture

For the backbone architecture, we avoid using the commonly used complex backbones like ResNet-101-FPN and ResNeXt-101-FPN. In Table 6, we evaluate FASSST with two different backbones. The results show that ResNet-50-FPN obtains better accuracy (0.7 higher on AP) than MobileNet-54-V2 but loses much speed (20.4FPS). Subsequently, we employ MobileNet-54-V2 as the default backbone due to its compactness and decent accuracy. Nevertheless, FASSST with more complex ResNet-50-FPN already achieves 38.8FPS and outperforms most approaches in Table 2 except YOLACT-550. We stress that FASSST can get competitive accuracy with the lightweight MobileNet-54-V2 when compared with much larger scale networks (cf. Table 1).

4.3.3 Number of Boxes

The number of boxes n in IAM plays an important role in the instance localization prediction, which is set to balance the performance and computational complexity of IAM. In Table 7, we report the AP scores on both COCO and CityScapes with different n values which are the squared numbers from 4 to 9. As we can see, the speeds FPS_{coco} and FPS_{city} get lower smoothly as n goes up. Among all schemes, $n = 36$ and $n = 49$ get the highest AP_{coco} 34.2 and AP_{city} 31.1, and thus are determined to be the best settings on COCO and CityScapes, respectively. The comparison of visual results on COCO with different n (16, 36 and 81) is further shown in Figure 10. It can be observed that $n = 36$ delivers the best

WACV #760

Scale	Depth	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	FPS
	4	30.5	50.3	31.8	12.4	33.1	46.2	62.3
416	**5**	**34.2**	**56.4**	**38.1**	**14.9**	**36.7**	**53.8**	**59.2**
	6	34.3	56.8	38.1	15.1	36.4	53.9	50.6
	4	33.3	52.6	35.4	14.3	35.2	48.6	52.1
768	**5**	**36.4**	**56.7**	**39.6**	**16.1**	**38.3**	**54.1**	**47.5**
	6	36.8	56.9	39.7	16.5	38.7	54.3	40.6

The chosen ones are in bold.

Table 5. Network Head: Larger and deeper head brings higher accuracy, while too large or deep head highly slows down the speed on COCO.

Backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	FPS
MobileNet-54-V2	**34.2**	**56.4**	**38.1**	**14.9**	**36.7**	**53.8**	**59.2**
ResNet-50-FPN	34.9	57.6	39.1	15.8	37.0	55.1	38.8

Table 6. Backbone Architecture: Backbone with higher complexity gains expected benefits but lowers the speed on COCO.

Number of Boxes	AP_{coco}	AP_{city}	FPS_{coco}	FPS_{city}
16	31.5	27.3	65.1	52.1
25	33.3	29.5	63.5	51.0
36	**34.2**	**30.6**	**59.2**	**49.5**
49	34.0	31.1	55.6	47.5
64	32.6	30.8	49.6	42.3
81	30.1	28.6	41.9	34.7

"coco" and "city" mean on COCO and CityScapes datasets, respectively.

Table 7. Number of Boxes: More boxes in IAM bring accuracy benefits but speed decreases, while too many boxes cause accuracy loss due to overfitting.

COCO model	AP	AP_{50}	$person$	$rider$	car	$truck$	bus	$train$	$motorcycle$	$bicycle$
with	**31.1**	**56.2**	**34.5**	**26.8**	**49.9**	**28.7**	**38.3**	**27.8**	**24.2**	**18.7**
without	25.8	49.2	29.5	21.7	44.9	23.1	33.5	21.0	18.4	14.2

Table 8. Pretrained COCO Model: Pretrained model on COCO remarkably improves the accuracy on CityScapes.

RFF	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
with	**34.2**	**56.4**	**38.1**	**14.9**	**36.7**	**53.8**
without	29.5	52.0	33.2	11.4	32.2	46.7

Table 9. RFF: Fused ROI features make significant difference to instance segmentation accuracy.

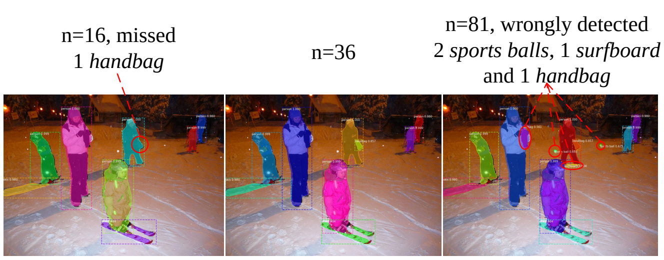

Figure 10. Visual results on COCO with different numbers of boxes.

performance for which all instances are precisely located and segmented.

4.3.4 RFF

The proposed RFF has significant impact on the performance of instance segmentation results. Table 9 shows the accuracy results with/without RFF. Note that we directly feed ROI features of the first pyramid mask layer to the following part if without RFF. It can be observed that RFF brings a 4.7 improvement on AP, which verifies its importance.

4.3.5 COCO Pretrained Model

Finally we evaluate the impacts of COCO pretrained model adopted in CityScapes training. Table 8 reports the AP scores on CityScapes with/without COCO pretrained model. We have the observation that the COCO pretrained model provides a 5.3 AP improvement on CityScapes.

5. Conclusion

This work has developed a network named FASSST for real-time instance segmentation with video-grade speed. An instance attention module is proposed to locate and segment the target instances. A single-stage feature regression strategy is applied to map features to instance coordinates and class probabilities, followed by ROI feature fusion to aggregate information from the pyramid mask layers for final mask generation. Experiments on the large-scale COCO and CityScapes datasets demonstrate the state-of-the-art performance of FASSST: 47.5FPS on a GTX1080Ti GPU and 5.3FPS on a Jetson Xavier NX board with only 71.6GFLOPs.

References

[1] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *TPAMI*, 39(12):2481–2495, 2017. 7

[2] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. Yolact: Real-time instance segmentation. *arXiv preprint arXiv:1904.02689*, 2019. 2, 5, 6, 7

[3] Jiale Cao, Rao Muhammad Anwer, Hisham Cholakkal, Fahad Shahbaz Khan, Yanwei Pang, and Ling Shao. Sipmask: Spatial information preservation for fast image and video instance segmentation. *arXiv preprint arXiv:2007.14772*, 2020. 2

[4] Hao Chen, Kunyang Sun, Zhi Tian, Chunhua Shen, Yongming Huang, and Youliang Yan. Blendmask: Top-down meets bottom-up for instance segmentation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8573–8581, 2020. 2

[5] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. In *CVPR*, pages 1907–1915, 2017. 1

[6] Yuan Cheng, Guangya Li, Ngai Wong, Hai-Bao Chen, and Hao Yu. Deepeye: A deeply tensor-compressed neural network hardware accelerator. In *ICCAD*, pages 1–8, 2019. 1

[7] Marius Cordts, Mohamed Omran, and Sebastian Ramos. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, pages 3213–3223, 2016. 1

[8] Jifeng Dai, Kaiming He, and Sun Jian. Convolutional feature masking for joint object and stuff segmentation. In *CVPR*, 2015. 2

[9] Jifeng Dai, Kaiming He, and Jian Sun. Instance-aware semantic segmentation via multi-task network cascades. In *CVPR*, pages 3150–3158, 2016. 1, 2, 5

[10] Bert De Brabandere, Davy Neven, and Luc Van Gool. Semantic instance segmentation with a discriminative loss function. *arXiv preprint arXiv:1708.02551*, 2017. 1, 2

[11] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional two-stream network fusion for video action recognition. In *CVPR*, pages 1933–1941, 2016. 1

[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*, pages 642–651, 2019. 1, 2, 5, 6, 7

[13] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask r-cnn. *TPAMI*, PP(99):1–1, 2017. 1, 2, 5, 6, 7

[14] Zhaojin Huang, Lichao Huang, Yongchao Gong, Chang Huang, and Xinggang Wang. Mask scoring r-cnn. In *CVPR*, pages 6409–6418, 2019. 6

[15] Hoel Kervadec, Jihene Bouchtiba, Christian Desrosiers, Eric Granger, Jose Dolz, and Ismail Ben Ayed. Boundary loss for highly unbalanced segmentation. In *MIDL*, pages 285–296, 2019. 4

[16] Alexander Kirillov, Evgeny Levinkov, Bjoern Andres, Bogdan Savchynskyy, and Carsten Rother. Instancecut: from edges to instances with multicut. In *CVPR*, pages 5008–5017, 2017. 7

[17] Youngwan Lee and Jongyoul Park. Centermask: Real-time anchor-free instance segmentation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13906–13915, 2020. 2, 5

[18] Yi Li, Haozhi Qi, Jifeng Dai, Xiangyang Ji, and Yichen Wei. Fully convolutional instance-aware semantic segmentation. In *CVPR*, pages 2359–2367, 2017. 2, 5, 6

[19] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2117–2125, 2017. 3

[20] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *ICCV*, pages 2980–2988, 2017. 4, 6

[21] 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*, pages 740–755, 2014. 1

[22] Shu Liu, Jiaya Jia, Sanja Fidler, and Raquel Urtasun. Sgn: Sequential grouping networks for instance segmentation. In *ICCV*, pages 3496–3504, 2017. 2, 5, 7

[23] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *ECCV*, pages 21–37, 2016. 3

[24] Yiding Liu, Siyu Yang, Bin Li, Wengang Zhou, Jizheng Xu, Houqiang Li, and Yan Lu. Affinity derivation and graph merge for instance segmentation. In *ECCV*, pages 686–703, 2018. 7

[25] Pedro O Pinheiro, Ronan Collobert, and Piotr Dollár. Learning to segment object candidates. In *NeurIPS*, pages 1990–1998, 2015. 2

[26] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. 3

[27] Zhi Tian, Chunhua Shen, and Hao Chen. Conditional convolutions for instance segmentation. *arXiv preprint arXiv:2003.05664*, 2020. 2

[28] Xinlong Wang, Tao Kong, Chunhua Shen, Yuning Jiang, and Lei Li. Solo: Segmenting objects by locations. *arXiv preprint arXiv:1912.04488*, 2019. 2, 5, 6

[29] Xinlong Wang, Rufeng Zhang, Tao Kong, Lei Li, and Chunhua Shen. Solov2: Dynamic and fast instance segmentation. *Advances in Neural Information Processing Systems*, 2020. 2

[30] Enze Xie, Peize Sun, Xiaoge Song, Wenhai Wang, Xuebo Liu, Ding Liang, Chunhua Shen, and Ping Luo. Polarmask: Single shot instance segmentation with polar representation. *arXiv preprint arXiv:1909.13226*, 2019. 2, 5, 6, 7

[31] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *CVPR*, pages 2881–2890, 2017. 3

[32] Xingyi Zhou, Jiacheng Zhuo, and Philipp Krahenbuhl. Bottom-up object detection by grouping extreme and center points. In *CVPR*, pages 850–859, 2019. 5