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

Yuan Cheng*,†, Rui Lin†, Peining Zhen*, Tianshu Hou*, Chiu Wa Ng†, Hai-Bao Chen*, Hao Yu‡, Ngai Wong†
*Shanghai Jiao Tong University †The University of Hong Kong
‡Southern University of Science and Technology

cyuan328@sjtu.edu.cn

Abstract

Real-time instance segmentation is crucial in various AI applications. This work designs a network named FASSST (Fast Attention based Single-Stage Segmentation Net) 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 groupings 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.

To overcome these hurdles, we design FASSST (Fast Attention-based Single-Stage Segmentation Net) 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 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 pre-waste unnecessary computation on the unadopted candidates and overlapped areas of candidates. 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.

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.

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/area of interest: small size, high speed and high accuracy.

![Figure 2. Comparison of area of interest among different instance segmentation schemes.](image)

With such analysis, we present the full architecture of FASSST in Figure 3. Assuming $F \in \mathbb{R}^{W \times H \times D_f}$ 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 $E \in \mathbb{R}^{W_e \times H_e \times D_e}$ of the whole frame. The specific settings of network head will be further analyzed in the experiment section. Then, the feature tensor $E$ is parallelly delivered into the following layers and the IAM. The IAM is applied to learn instance information tensor $I \in \mathbb{R}^{W_i \times H_i \times D_i}$, 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 $R \in \mathbb{R}^{W_r \times H_r \times D_r}$ by an ROI feature fusion module (RFF). Note that the fused ROI feature tensor
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 $E$ is generated by the network head:

$$E = \text{extr}(F),$$  

(1)

where $\text{extr}$ 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 $I$:

$$I = \text{attn}(E),$$  

(2)

where $\text{attn}$ represents the instance attention process. $\text{attn}$ regards the instance attention as a single-stage regression problem, which directly learns instance locality information $I$ from raw features $E$ [26]. Specifically, $I$ 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 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 $I$ 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.

Figure 3. Overview of the FASSST architecture.

Figure 4. Instance attention module (IAM) architecture.

Figure 5. ROI feature fusion architecture.

Figure 6. Mask generation progress.
The fused ROI features carry both local (instance core) and 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 \( R_1, R_2 \) and \( R_3 \). Note that \( R_3 \) has a spatial size double that of \( R_2 \), and \( R_1 \) is double that of \( R_2 \). To form these, we first perform upsampling on \( R_3 \) with a rate of 2 through bilinear interpolation:

\[
R_3^u = \text{upsamp}(R_3),
\]

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

\[
R = \text{fuse}(R_1, R_2, R_3) \\
= \text{ReLU} \left( R_1 + \text{upsamp}(R_2 + \text{upsamp}(R_3) \right),
\]

where \text{fuse} denotes the feature fusion function. Note that a \text{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 \( R \) 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 \( S \) of the proposed framework FASSST. The whole workflow of FASSST is summarized in Figure 7, where \text{loct} represents the ROI localization process, and \text{mask} represents the final mask generation.

### 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 \( L_c \):

\[
L = \lambda_m L_m + \lambda_l L_l + \lambda_c L_c,
\]

where \( L \) is the total loss, \( \lambda_m, \lambda_l \) and \( \lambda_c \) are set as 5.75, 3 and 1.25, respectively. Specifically, the \( L_m \) is based on Dice Loss [15]:

\[
L_m = 1 - \text{Dice}(\text{mask}_p, \text{mask}_g),
\]

where \( \text{Dice} \) is the corresponding function for dice coefficients, \( \text{mask}_p \) and \( \text{mask}_g \) 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 CityScape. 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-S4-V2 backbone and network head with input frame scale \( 416 \times 416 \). We will emphasize by suffix if different settings are used. All these settings will be further discussed in Section 4.3 on ablation study.

---

**Algorithm 1 Forwards Propagation of FASSST Training**

**Require:** Frame data \( F \), training epoch \( T \).

**Ensure:** Training accuracy \( P \).

1: for \( k = 1 : T \) do
2: Feed \( F \) into the network head
3: Obtain the extracted features \( E^k \leftarrow \text{extr}(F) \)
4: Feed \( E^k \) to IAM
5: \( I^k \leftarrow \text{attn}(E^k) \), parallelly process the main network body to pyramid mask layers
6: Locate the ROIs: \( R^k_1, R^k_2, R^k_3 \leftarrow \text{loct}(I^k) \)
7: RFF: \( R^k \leftarrow \text{fuse}(R^k_1, R^k_2, R^k_3) \)
8: Feed \( R^k \) to mask “tail”: \( S^k \leftarrow \text{mask}(R^k) \)
9: end for
10: Get the training accuracy \( P \)
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 ~ 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, AP50, AP75, APs, APm, and APl.

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

<table>
<thead>
<tr>
<th>Category</th>
<th>Approach</th>
<th>Backbone</th>
<th>AP</th>
<th>AP50</th>
<th>AP75</th>
<th>APs</th>
<th>APm</th>
<th>APl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixel-wise</td>
<td>SGN [22]</td>
<td>-</td>
<td>25.0</td>
<td>44.9</td>
<td>25.8</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>SSAP [12]</td>
<td>ResNet-101-FPN</td>
<td>29.4</td>
<td>48.1</td>
<td>28.8</td>
<td>28.6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>FCIS [18]</td>
<td>ResNet-101-C5-dilated</td>
<td>29.2</td>
<td>49.5</td>
<td>-</td>
<td>7.1</td>
<td>31.3</td>
<td>50.0</td>
</tr>
<tr>
<td></td>
<td>FCIS+++ [18]</td>
<td>ResNet-101-C5-dilated</td>
<td>33.6</td>
<td>54.5</td>
<td>37.9</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Proposal-based</td>
<td>MNC [9]</td>
<td>ResNet-101-C4</td>
<td>24.6</td>
<td>44.3</td>
<td>24.8</td>
<td>4.7</td>
<td>25.9</td>
<td>43.6</td>
</tr>
<tr>
<td></td>
<td>Mask R-CNN [13]</td>
<td>ResNet-101-FPN</td>
<td>35.7</td>
<td>58.0</td>
<td>37.8</td>
<td>15.5</td>
<td>38.1</td>
<td>52.4</td>
</tr>
<tr>
<td>Single-stage</td>
<td>ExtremeNet [32]</td>
<td>Hourglass-104</td>
<td>18.9</td>
<td>44.5</td>
<td>13.7</td>
<td>10.4</td>
<td>20.4</td>
<td>28.3</td>
</tr>
<tr>
<td></td>
<td>YOLACT [2]</td>
<td>ResNet-101-FPN</td>
<td>31.2</td>
<td>50.6</td>
<td>32.8</td>
<td>12.1</td>
<td>33.3</td>
<td>47.1</td>
</tr>
<tr>
<td></td>
<td>SOLO [28]</td>
<td>ResNet-101-FPN</td>
<td>37.8</td>
<td>59.5</td>
<td>40.4</td>
<td>16.4</td>
<td>40.6</td>
<td>54.2</td>
</tr>
<tr>
<td></td>
<td>SipMask [17]</td>
<td>ResNet-101-FPN</td>
<td>32.8</td>
<td>53.4</td>
<td>34.3</td>
<td>9.3</td>
<td>35.6</td>
<td>54.0</td>
</tr>
<tr>
<td></td>
<td>CenterMask [17]</td>
<td>ResNet-50-FPN</td>
<td>32.9</td>
<td>-</td>
<td>-</td>
<td>12.9</td>
<td>34.7</td>
<td>48.7</td>
</tr>
<tr>
<td></td>
<td>PolarMask [30]</td>
<td>ResNet-101-FPN</td>
<td>30.4</td>
<td>51.9</td>
<td>31.0</td>
<td>13.4</td>
<td>32.4</td>
<td>42.8</td>
</tr>
<tr>
<td>Proposed</td>
<td>FASSST</td>
<td>MobileNet-54-V2</td>
<td>34.2</td>
<td>56.4</td>
<td>38.1</td>
<td>14.9</td>
<td>36.7</td>
<td>53.8</td>
</tr>
</tbody>
</table>

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

Figure 8. Sample visual results of FASSST on COCO.
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.

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.

4.2. Evaluation on CityScapes

We further test FASSST on the CityScapes, a large-scale dataset with high quality pixel-level annotations of 5000 im-

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

Table 2. Efficiency comparison with state-of-the-arts on COCO. ages 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 comparison 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.
We further report efficiency analysis of FASSST on AP gain, and causes 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.

### 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 $\text{FPS}_{\text{coco}}$ and $\text{FPS}_{\text{city}}$ get lower smoothly as $n$ goes up. Among all schemes, $n = 36$ and $n = 49$ get the highest $\text{AP}_{\text{coco}}$ 34.2 and $\text{AP}_{\text{city}}$ 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 $n$ is further shown in Figure 10. It can be observed that $n = 36$ delivers the best 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...
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.

<table>
<thead>
<tr>
<th>Scale</th>
<th>Depth</th>
<th>AP</th>
<th>AP_{50}</th>
<th>AP_{75}</th>
<th>AP_S</th>
<th>AP_M</th>
<th>AP_L</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>416</td>
<td>5</td>
<td>34.2</td>
<td>56.4</td>
<td>38.1</td>
<td>14.9</td>
<td>36.7</td>
<td>53.8</td>
<td>59.2</td>
</tr>
<tr>
<td>768</td>
<td>5</td>
<td>36.4</td>
<td>56.7</td>
<td>39.6</td>
<td>16.1</td>
<td>38.3</td>
<td>54.1</td>
<td>47.5</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>36.8</td>
<td>56.9</td>
<td>39.7</td>
<td>16.5</td>
<td>38.7</td>
<td>54.3</td>
<td>40.6</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Backbone</th>
<th>AP</th>
<th>AP_{50}</th>
<th>AP_{75}</th>
<th>AP_S</th>
<th>AP_M</th>
<th>AP_L</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileNet-54-V2</td>
<td>34.2</td>
<td>56.4</td>
<td>38.1</td>
<td>14.9</td>
<td>36.7</td>
<td>53.8</td>
<td>59.2</td>
</tr>
<tr>
<td>ResNet-50-FPN</td>
<td>34.9</td>
<td>57.6</td>
<td>39.1</td>
<td>15.8</td>
<td>37.0</td>
<td>55.1</td>
<td>38.8</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Number of Boxes</th>
<th>AP_{coco}</th>
<th>AP_{city}</th>
<th>FPS_{coco}</th>
<th>FPS_{city}</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>31.5</td>
<td>27.3</td>
<td>65.1</td>
<td>52.1</td>
</tr>
<tr>
<td>25</td>
<td>33.3</td>
<td>29.5</td>
<td>63.5</td>
<td>51.0</td>
</tr>
<tr>
<td>36</td>
<td>34.2</td>
<td>30.6</td>
<td>59.2</td>
<td>49.5</td>
</tr>
<tr>
<td>49</td>
<td>34.0</td>
<td>31.1</td>
<td>55.6</td>
<td>47.5</td>
</tr>
<tr>
<td>64</td>
<td>32.6</td>
<td>30.8</td>
<td>49.6</td>
<td>42.3</td>
</tr>
<tr>
<td>81</td>
<td>30.1</td>
<td>28.6</td>
<td>41.9</td>
<td>34.7</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>COCO model</th>
<th>AP_{50}</th>
<th>person</th>
<th>rider</th>
<th>car</th>
<th>truck</th>
<th>bus</th>
<th>train</th>
<th>motorcycle</th>
<th>bicycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>with</td>
<td>31.1</td>
<td>56.2</td>
<td>34.5</td>
<td>26.8</td>
<td>49.9</td>
<td>28.7</td>
<td>38.3</td>
<td>27.8</td>
<td>24.2</td>
</tr>
<tr>
<td>without</td>
<td>25.8</td>
<td>49.2</td>
<td>29.5</td>
<td>21.7</td>
<td>44.9</td>
<td>23.1</td>
<td>33.5</td>
<td>21.0</td>
<td>18.4</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>RFF</th>
<th>AP</th>
<th>AP_{50}</th>
<th>AP_{75}</th>
<th>AP_S</th>
<th>AP_M</th>
<th>AP_L</th>
</tr>
</thead>
<tbody>
<tr>
<td>with</td>
<td>34.2</td>
<td>56.4</td>
<td>38.1</td>
<td>14.9</td>
<td>36.7</td>
<td>53.8</td>
</tr>
<tr>
<td>without</td>
<td>29.5</td>
<td>52.0</td>
<td>33.2</td>
<td>11.4</td>
<td>32.2</td>
<td>46.7</td>
</tr>
</tbody>
</table>

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.

Acknowledgement

This work is supported in part by the Seed Funding of the HKU-TCL Joint Research Center for Artificial Intelligence, and in part by the General Research Fund (GRF) project 17206020.
References


