YolactEdge: Real-time Instance Segmentation on the Edge

Haotian Liu*, Rafael A. Rivera Soto*, Fanyi Xiao, and Yong Jae Lee

***Abstract*— We propose *YolactEdge*, the first competitive instance segmentation approach that runs on small edge devices at real-time speeds. Specifically, YolactEdge runs at up to 30.8 FPS on a Jetson AGX Xavier (and 172.7 FPS on an RTX 2080 Ti) with a ResNet-101 backbone on 550x550 resolution images. To achieve this, we make two improvements to the state-of-the-art image-based real-time method YOLACT [1]: (1) applying TensorRT optimization while carefully trading off speed and accuracy, and (2) a novel feature warping module to exploit temporal redundancy in videos. Experiments on the YouTube VIS and MS COCO datasets demonstrate that YolactEdge produces a 3-5x speed up over existing real-time methods while producing competitive mask and box detection accuracy. We also conduct ablation studies to dissect our design choices and modules. Code and models are available at https://github.com/haotian-liu/yolact_edge.**

I. Introduction

Instance segmentation is a challenging problem that requires the correct detection and segmentation of each *object instance* in an image. A fast and accurate instance segmenter would have many useful applications in robotics, autonomous driving, image/video retrieval, healthcare, security, and others. In particular, a *real-time* instance segmenter that can operate on *small edge devices* is necessary for many real-world scenarios. For example, in safety critical applications in complex environments, robots, drones, and other autonomous machines may need to perceive objects and humans in real-time on device – without having access to the cloud, and in resource constrained settings where bulky and power hungry GPUs (e.g., Titan Xp) are impractical. However, while there has been great progress in real-time instance segmentation research [1], [2], [3], [4], [5], [6], [7], thus far, there is no method that can run accurately at real-time speeds on small edge devices like the Jetson AGX Xavier.

In this paper, we present *YolactEdge*, a novel real-time instance segmentation approach that runs accurately on edge devices at real-time speeds. Specifically, with a ResNet-101 backbone, YolactEdge runs at up to 30.8 FPS on a Jetson AGX Xavier (and 172.7 FPS on an RTX 2080 Ti GPU), which is 3-5x faster than existing state-of-the-art real-time methods, while being competitive in accuracy.

In order to perform inference at real-time speeds on edge devices, we build upon the state-of-the-art image-based real-time instance segmentation method, YOLACT [1], and make two fundamental improvements, one at the system-level and the other at the algorithm-level: (1) we apply NVIDIA's TensorRT inference engine [8] to quantize the network parameters to fewer bits while systematically balancing any tradeoff in accuracy, and (2) we leverage temporal redundancy in video (i.e., temporally nearby frames are highly correlated), and learn to transform and propagate features over time so that the deep network's expensive backbone feature computation does not need to be fully computed on every frame.

The proposed shift to video from static image processing makes sense from a practical standpoint, as the real-time aspect matters much more for video applications that require low latency and real-time response than for image applications; *e.g.*, for real-time control in robotics and autonomous driving, or real-time object/activity detection in security and augmented reality, where the system must process a stream of video frames and generate instance segmentation outputs in real-time. Importantly, all existing real-time instance segmentation methods (including YOLACT) are static image-based, which makes YolactEdge the first *video-dedicated* real-time instance segmentation method.

In sum, our contributions are: (1) we apply TensorRT optimization while carefully trading off speed and accuracy, (2) we propose a novel feature warping module to exploit temporal redundancy in videos, (3) we perform experiments on the benchmark image MS COCO [9] and video YouTube VIS [10] datasets, demonstrating a 3-5x faster speed compared to existing real-time instance segmentation methods while being competitive in accuracy, and (4) we publicly release our code and models to facilitate progress in robotics applications that require on device real-time instance segmentation.

II. Related Work

Real-time instance segmentation in images. YOLACT [1] is the first real-time instance segmentation method to achieve competitive accuracy on the challenging MS COCO [9] dataset. Recently, CenterMask [2], BlendMask [5], and SOLOv2 [3] have improved accuracy in part by leveraging more accurate object detectors (*e.g.*, FCOS [11]). All existing real-time instance segmentation approaches [1], [2], [5], [6], [3] are image-based and require bulky GPUs like the Titan Xp / RTX 2080 Ti to achieve real-time speeds. In contrast, we propose the first *video-based* real-time instance segmentation approach that can run on small edge devices like the Jetson AGX Xavier.

Feature propagation in videos has been used to improve speed and accuracy for video classification and video object detection [12], [13], [14]. These methods use off-the-shelf optical flow networks [15] to estimate pixel-level object motion and warp feature maps from frame to frame. However, even the most lightweight flow networks [15], [16] require non-negligible memory and compute, which are obstacles

[1]Fanyi Xiao is with Amazon Web Services, Inc., the rest are with the University of California, Davis. {lhtliu, riverasoto, fyxiao, yongjaelee}@ucdavis.edu (* *Haotian Liu and Rafael A. Rivera Soto are co-first authors.*)

for real-time speeds on edge devices. In contrast, our model estimates object motion and performs feature warping directly at the feature level (as opposed to the input pixel level), which enables real-time speeds.

Improving model efficiency. Designing lightweight yet performant backbones and feature pyramids has been one of the main thrusts in improving deep network efficiency. MobileNetv2 [17] introduces depth-wise convolutions and inverted residuals to design a lightweight architecture for mobile devices. MobileNetv3 [18], NAS-FPN [19], and EfficientNet [20] use neural architecture search to automatically find efficient architectures. Others utilize knowledge distillation [21], [22], [23], model compression [24], [25], or binary networks [26], [27]. The CVPR Low Power Computer Vision Challenge participants have used TensorRT [8], a deep learning inference optimizer, to quantize and speed up object detectors such as Faster-RCNN on the NVIDIA Jetson TX2 [28]. In contrast to most of these approaches, YolactEdge retains large expressive backbones, and exploits temporal redundancy in video together with a TensorRT optimization for fast and accurate instance segmentation.

III. Approach

Our goal is to create an instance segmentation model, YolactEdge, that can achieve real-time (>30 FPS) speeds on edge devices. To this end, we make two improvements to the image-based real-time instance segmentation approach YOLACT [1]: (1) applying TensorRT optimization, and (2) exploiting temporal redundancy in video.

A. TensorRT Optimization

The edge device that we develop our model on is the NVIDIA Jetson AGX Xavier. The Xavier is equipped with an integrated Volta GPU with Tensor Cores, dual deep learning accelerator, 32GB of memory, and reaches up to 32 TeraOPS at a cost of $699. Importantly, the Xavier is the only architecture from the NVIDIA Jetson series that supports both FP16 and INT8 Tensor Cores, which are needed for TensorRT [29] optimization.

TensorRT is NVIDIA's deep learning inference optimizer that provides mixed-precision support, optimal tensor layout, fusing of network layers, and kernel specializations [8]. A major component of accelerating models using TensorRT is the quantization of model weights to INT8 or FP16 precision. Since FP16 has a wider range of precision than INT8, it yields better accuracy at the cost of more computational time. Given that the weights of different deep network components (backbone, prediction module, etc.) have different ranges, this speed-accuracy trade-off varies from component to component. Therefore, we convert each model component to TensorRT independently and explore the optimal mix between INT8 and FP16 weights that maximizes FPS while preserving accuracy.

Table I shows this analysis for YOLACT [1], which is the baseline model that YolactEdge directly builds upon. Briefly, YOLACT can be divided into 4 components: (1) a feature backbone, (2) a feature pyramid network [30] (FPN), (3) a ProtoNet, and (4) a Prediction Head; see Fig. 1 (right) for the network architecture. (More details on YOLACT will be provided in Sec. III-B.) The second row in Table I represents YOLACT, with all components in FP32 (i.e., no TensorRT optimization), and results in only 6.6 FPS on the Jetson AGX Xavier with a ResNet-101 backbone. From there, INT8 or FP16 conversion on different model components leads to various improvements in speed and changes in accuracy. Notably, conversion of the Prediction Head to INT8 (last four rows) always results in a large loss of instance segmentation accuracy. We hypothesize that this is because the final box and mask predictions require more than $2^8 = 256$ bins to be encoded without loss in the final representation. Converting every component to INT8 except for the Prediction Head and FPN (row highlighted in gray) achieves the highest FPS with little mAP degradation. Thus, this is the final configuration we go with for our model in our experiments, but different configurations can easily be chosen based on need.

Backbone	FPN	ProtoNet	PredHead	TensorRT	mAP	FPS
FP32	FP32	FP32	FP32	N	**29.8**	6.4
FP16	FP16	FP16	FP16	N	29.7	12.1
FP32	FP32	FP32	FP32	Y	29.6	19.1
FP16	FP16	FP16	FP16	Y	29.7	21.9
INT8	FP16	FP16	FP16	Y	29.9	26.3
INT8	FP16	INT8	FP16	Y	29.9	26.5
INT8	**INT8**	**FP16**	**FP16**	Y	29.7	**27.7**
INT8	INT8	INT8	FP16	Y	29.8	27.4
INT8	FP16	FP16	INT8	Y	25.4	26.2
INT8	FP16	INT8	INT8	Y	25.4	25.9
INT8	INT8	FP16	INT8	Y	25.2	26.9
INT8	INT8	INT8	INT8	Y	25.2	26.5

TABLE I: **Effect of Mixed Precision** on YOLACT [1] with a ResNet-101 backbone on the MS COCO val2017 dataset with a Jetson AGX Xavier using 100 calibration images. Mixing precision across the modules results in different instance segmentation mean Average Precision (mAP) and FPS for each instantiation of YOLACT. All results are averaged over 5 runs, with a standard deviation less than 0.6 FPS.

In order to quantize model components to INT8 precision, a calibration step is necessary: TensorRT collects histograms of activations for each layer, generates several quantized distributions with different thresholds, and compares each of them to the reference distribution using KL Divergence [31]. This step ensures that the model loses as little performance as possible when converted to INT8 precision. Table VIa shows the effect of the calibration dataset size. We observe that calibration is necessary for accuracy, and generally a larger calibration set provides a better speed-accuracy trade-off.

B. Exploiting Temporal Redundancy in Video

The TensorRT optimization leads to a ∼4x improvement in speed, and when dealing with static images, this is the version of YolactEdge one should use. However, when dealing with *video*, we can exploit temporal redundancy to make YolactEdge even faster, as we describe next.

Given an input video as a sequence of frames $\{I_i\}$, we aim to predict masks for each object instance in each frame $\{y_i = \mathcal{N}(I_i)\}$, in a fast and accurate manner. For our video

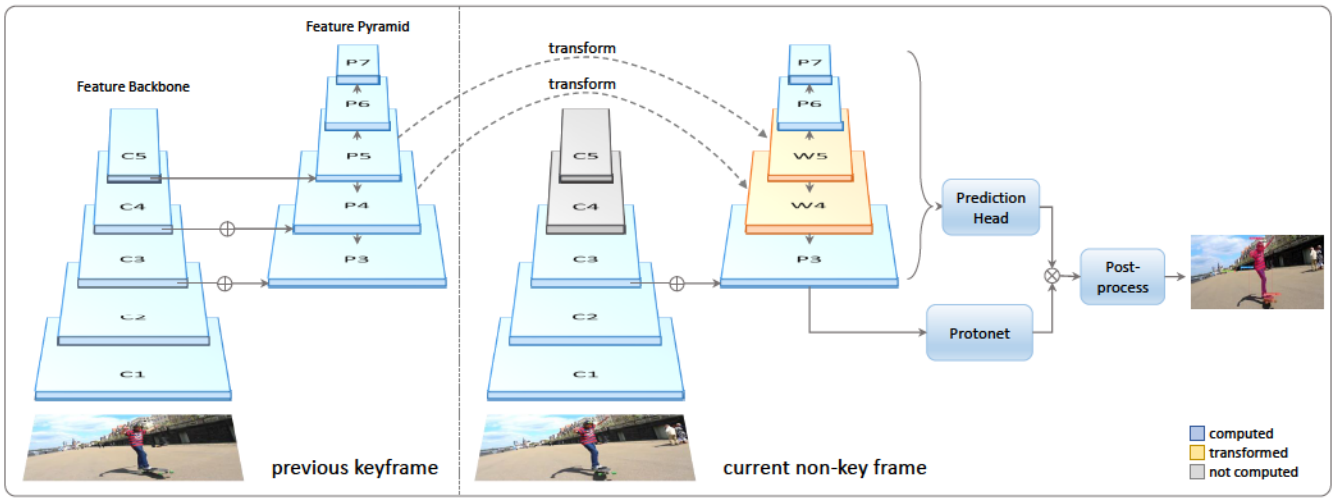


Fig. 1: **YolactEdge** extends YOLACT [1] to video by transforming a subset of the features from keyframes (left) to non-keyframes (right), to reduce expensive backbone computation. Specifically, on non-keyframes, we compute C_3 features that are cheap while crucial for mask prediction given its high-resolution. This largely accelerates our method while retaining accuracy on non-keyframes. We use blue, orange, and grey to indicate computed, transformed, and skipped blocks, respectively.

instance segmentation network $\mathcal{N}$, we largely follow the YOLACT [1] design for its simplicity and impressive speed-accuracy tradeoff. Specifically, on each frame, we perform two parallel tasks: (1) generating a set of prototype masks, and (2) predicting per-instance mask coefficients. Then, the final masks are assembled through linearly combining the prototypes with the mask coefficients.

For clarity of presentation, we decompose $\mathcal{N}$ into $\mathcal{N}_{feat}$ and $\mathcal{N}_{pred}$, where $\mathcal{N}_{feat}$ denotes the feature backbone stage and $\mathcal{N}_{pred}$ is the rest (*i.e.*, prediction heads for class, box, mask coefficients, and ProtoNet for generating prototype masks) which takes the output of $\mathcal{N}_{feat}$ and make instance segmentation predictions. We selectively divide frames in a video into two groups: keyframes I^k and non-keyframes I^n; the behavior of our model on these two groups of frames only varies in the backbone stage.

$$y^k = \mathcal{N}_{pred}(\mathcal{N}_{feat}(I^k)) \quad (1)$$

$$y^n = \mathcal{N}_{pred}(\widetilde{\mathcal{N}}_{feat}(I^n)) \quad (2)$$

For keyframes I^k, our model computes all backbone and pyramid features ($C_1 - C_5$ and $P_3 - P_7$ in Fig. 1). Whereas for non-keyframes I^n, we compute only a subset of the features, and transform the rest from the temporally closest previous keyframe using the mechanism that we elaborate on next. This way, we strike a balance between producing accurate predictions while maintaining a fast runtime.

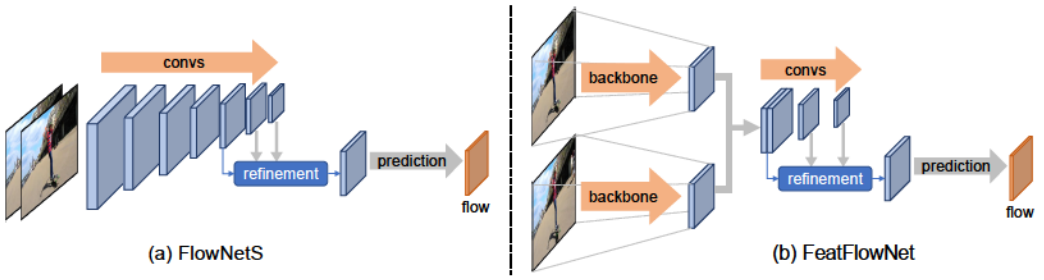
Partial Feature Transform. Transforming (*i.e.*, warping) features from neighboring keyframes was shown to be an effective strategy for reducing backbone computation to yield fast video bounding box object detectors in [12]. Specifically, [12] transforms all the backbone features using an off-the-shelf optical flow network [15]. However, due to inevitable errors in optical flow estimation, we find that it fails to provide sufficiently accurate features required for pixel-level tasks like instance segmentation. In this work, we propose to perform *partial feature transforms* to improve the quality of the transformed features while still maintaining a fast runtime.

Specifically, unlike [12], which transforms all features (P_3^k, P_4^k, P_5^k in our case) from a keyframe I^k to a non-keyframe I^n, our method computes the backbone features for a non-keyframe only up through the high-resolution C_3^n level (i.e., skipping C_4^n, C_5^n and consequently P_4^n, P_5^n computation), and only transforms the lower resolution P_4^k/P_5^k features from the previous keyframe to approximate P_4^n/P_5^n (denoted as W_4^n/W_5^n) in the current non-keyframe, as shown in Fig. 1 (right). It computes P_6^n/P_7^n by downsampling W_5^n in the same way as YOLACT. With the computed C_3^n features and transformed W_4^n features, it then generates P_3^n as $P_3^n = C_3^n + up(W_4^n)$, where $up(\cdot)$ denotes upsampling. Finally, we use the P_3^n features to generate pixel-accurate prototypes. This way, in contrast to [12], we can preserve high-resolution details for generating the mask prototypes, as the high-resolution C_3 features are computed instead of transformed and thus are immune to errors in flow estimation.

Importantly, although we compute the C_1-C_3 backbone features for every frame (*i.e.*, both key and non-keyframes), we avoid computing the most expensive part of the backbone, as the computational costs in different stages of pyramid-like networks are highly imbalanced. As shown in Table II, more than 66% of the computation cost of ResNet-101 lies in C_4, while more than half of the inference time is occupied by backbone computation. By computing only lower layers of the feature pyramid and transforming the rest, we can largely accelerate our method to reach real-time performance.

In summary, our *partial feature transform* design produces higher quality feature maps that are required for instance segmentation, while also enabling real-time speeds.

Efficient Motion Estimation. In this section, we describe



(a) FlowNetS (b) FeatFlowNet

Fig. 2: **Flow estimation**. Illustration of the difference between FlowNetS [15] (a) and our FeatFlowNet (b).

	C_1	C_2	C_3	C_4	C_5
# of convs	1	9	12	**69**	9
TFLOPS	0.1	0.7	1.0	**5.2**	0.8
%	1.5	8.7	13.2	**66.2**	10.3

(a) **ResNet-101 Backbone**

Stage	%	Stage	%
Backbone	**54.7**	FPN	6.4
ProtoNet	7.8	Pred	10.6
Detect	6.6	Other	13.1

(b) **YOLACT**

TABLE II: **Computational cost breakdown** for different stages of (a) ResNet-101 backbone, and (b) YOLACT.

how we efficiently compute flow between a keyframe and non-keyframe. Given a non-keyframe I^n and its preceding keyframe I^k, our model first encodes object motion between them as a 2-D flow field $\mathcal{M}(I^k, I^n)$. It then uses the flow field to transform the features $F^k = \{P_4^k, P_5^k\}$ from frame I^k to align with frame I^n to produce the warped features $\widetilde{F}^n = \{W_4^n, W_5^n\} = \mathcal{T}(F^k, \mathcal{M}(I^k, I^n))$.

In order to perform fast feature transformation, we need to estimate object motion efficiently. Existing frameworks [12], [13] that perform flow-guided feature transform directly adopt off-the-shelf pixel-level optical flow networks for motion estimation. FlowNetS [15] (Fig. 2a), for example, performs flow estimation in three stages: it first takes in raw RGB frames as input and computes a stack of features; it then refines a subset of the features by recursively upsampling and concatenating feature maps to generate coarse-to-fine features that carry both high-level (large motion) and fine local information (small motion); finally, it uses those features to predict the final flow map.

In our case, to save computation costs, instead of taking an off-the-shelf flow network that processes raw RGB frames, we reuse the features computed by our model's backbone network, which already produces a set of semantically rich features. To this end, we propose FeatFlowNet (Fig. 2b), which generally follows the FlowNetS architecture, but in the first stage, instead of computing feature stacks *from raw RGB image inputs*, we re-use features from the ResNet backbone (C_3) and use fewer convolution layers. As we demonstrate in our experiments, our flow estimation network is much faster while being equally effective.

Feature Warping. We use FeatFlowNet to estimate the flow map $\mathcal{M}(I^k, I^n)$ between the previous keyframe I^k and the current non-keyframe I^n, and then transform the features from I^k to I^n via inverse warping: by projecting each pixel x in I^n to I^k as $x + \delta x$, where $\delta x = \mathcal{M}_x(I^k, I^n)$. The pixel value is then computed via bilinear interpolation $F^{k \to n}(x) = \sum_u \theta(u, x + \delta x) F^k(x)$, where θ is the bilinear interpolation weight at different spatial locations.

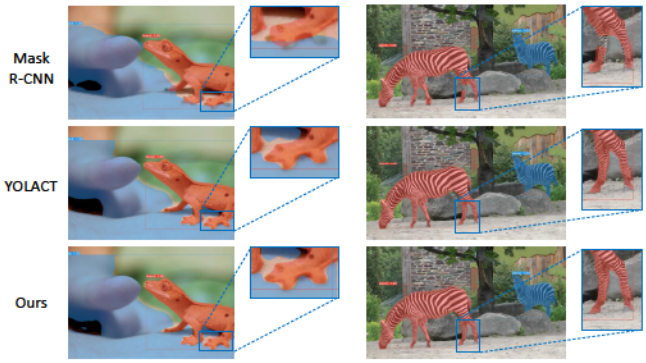


Fig. 3: **Mask quality**. Our masks are as high quality as YOLACT even on non-keyframes, and are typically higher quality than those of Mask R-CNN [32].

Loss Functions. For the instance segmentation task, we use the same losses as YOLACT [1] to train our model: classification loss L_{cls}, box regression loss L_{box}, mask loss L_{mask}, and auxiliary semantic segmentation loss L_{aux}. For flow estimation network pre-training, like [15], we use the endpoint error (EPE).

IV. Results

In this section, we analyze YolactEdge's instance segmentation accuracy and speed on the Jetson AGX Xavier and RTX 2080 Ti. We compare to state-of-the-art real-time instance segmentation methods, and perform ablation studies to dissect our various design choices and modules.

Implementation details. We train with a batch size of 32 on 4 GPUs using ImageNet pre-trained weights. We leave the pre-trained batchnorm (*bn*) unfrozen and do not add any extra bn layers. We first pre-train YOLACT with SGD for 500k iterations with 5×10^{-4} initial learning rate. Then, we freeze YOLACT weights, and train FeatFlowNet on FlyingChairs [33] with 2×10^{-4} initial learning rate. Finally, we fine-tune all weights except ResNet backbone for 200k iterations with 2×10^{-4} initial learning rate. When pre-training YOLACT, we apply all data augmentations used in YOLACT; during fine-tuning, we disable *random expand* to allow the warping module to model larger motions. For all training stages, we

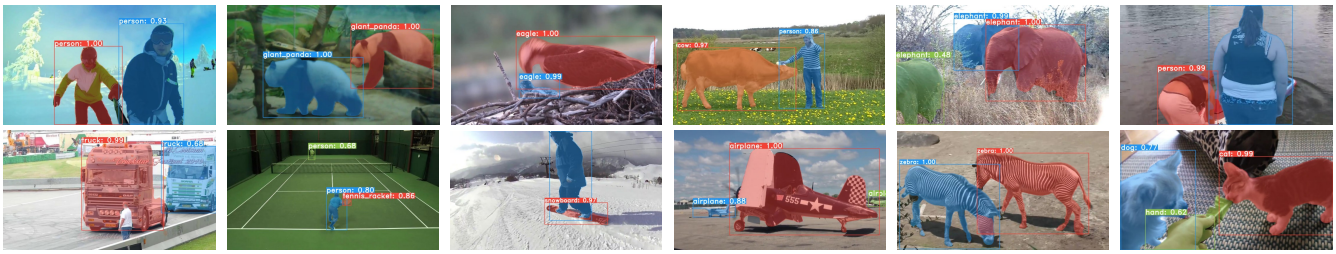


Fig. 4: **YolactEdge results on YouTube VIS** on non-keyframes whose subset of features are warped from a keyframe 4 frames away (farthest in sampling window). Our mask predictions can tightly fit the objects, due to partial feature transform.

Method	Backbone	mask AP	box AP	RTX FPS
Mask R-CNN [32]	R-101-FPN	43.1	47.3	14.1
CenterMask-Lite [2]	V-39-FPN	41.6	45.9	34.4
BlendMask-RT [5]	R-50-FPN	44.0	47.9	49.3
SOLOv2-Light [3]	R-50-FPN	46.3	–	43.9
YOLACT [1]	R-50-FPN	44.7	46.2	59.8
YOLACT [1]	R-101-FPN	**47.3**	**48.9**	42.6
Ours				
YolactEdge (w/o TRT)	R-50-FPN	44.2	45.2	**67.0**
YolactEdge (w/o TRT)	R-101-FPN	46.9	47.8	61.2
YolactEdge	R-50-FPN	44.0	45.1	**177.6**
YolactEdge	R-101-FPN	46.2	47.1	172.7

TABLE III: **Comparison to state-of-the-art real-time methods on YouTube VIS**. We use our sub-training and sub-validation splits for YouTube VIS and perform joint training with COCO using a 1:1 data sampling ratio. (Box AP is not evaluated in the authors' code base of SOLOv2.)

Method	Backbone	mask AP	box AP	AGX FPS	RTX FPS
YOLACT [1]	MobileNet-V2	**22.1**	**23.3**	15.0	35.7
YolactEdge (w/o video)	MobileNet-V2	20.8	22.7	**35.7**	**161.4**
YOLACT [1]	R-50-FPN	**28.2**	**30.3**	9.1	45.0
YolactEdge (w/o video)	R-50-FPN	27.0	30.1	**30.7**	**140.3**
YOLACT [1]	R-101-FPN	**29.8**	**32.3**	6.6	36.5
YolactEdge (w/o video)	R-101-FPN	29.5	32.1	**27.3**	**124.8**

TABLE IV: **YolactEdge (w/o video) comparision to YOLACT on MS COCO [9] test-dev split.** AGX: Jetson AGX Xavier; RTX: RTX 2080 Ti.

use cosine learning rate decay schedule, with weight decay 5×10^{-4}, and momentum 0.9. We pick the first of every 5 frames as the keyframes. We use 100 images from the training set to calibrate our INT8 model components (backbone, prototype, FeatFlowNet) for TensorRT, and the remaining components (prediction head, FPN) are converted to FP16. We do not convert the warping module to TensorRT, as the conversion of the sampling function (needed for inverse warp) is not natively supported, and is also not a bottleneck for our feature propagation to be fast. We limit the output resolution to be a maximum of 640x480 while preserving the aspect ratio.

Datasets. YouTube VIS [10] is a video instance segmentation dataset for detection, segmentation, and tracking of object instances in videos. It contains 2883 high-resolution YouTube videos of 40 common objects such as person, animals, and vehicles, at a frame rate of 30 FPS. The train, validation, and test set contain 2238, 302, and 343 videos, respectively. Every 5th frame of each video is annotated with pixel-level

Method	Backbone	mask AP	box AP	AGX FPS	RTX FPS
YOLACT [1]	R-50-FPN	**44.7**	**46.2**	8.5	59.8
YolactEdge (w/o TRT)	R-50-FPN	44.2	45.2	10.5	67.0
YolactEdge (w/o video)	R-50-FPN	44.5	46.0	32.0	**185.7**
YolactEdge	R-50-FPN	44.0	45.1	**32.4**	177.6
YOLACT [1]	R-101-FPN	**47.3**	**48.9**	5.9	42.6
YolactEdge (w/o TRT)	R-101-FPN	46.9	47.8	9.5	61.2
YolactEdge (w/o video)	R-101-FPN	46.9	48.4	27.9	158.2
YolactEdge	R-101-FPN	46.2	47.1	**30.8**	**172.7**

TABLE V: **YolactEdge ablation results on Youtube VIS.**

instance segmentation ground-truth masks. Since we only perform instance segmentation (without tracking), we cannot directly use the validation server of YouTube VIS to evaluate our method. Instead, we further divide the training split into two train-val splits with a 85%-15% ratio (1904 and 334 videos). To demonstrate the validity of our own train-val split, we created two more splits, and configured them so that any two splits have video overlap of less than 18%. We evaluated Mask R-CNN, YOLACT, and YolactEdge on all three splits, the AP variance is within ± 2.0.

We also evaluate our approach on the MS COCO [9] dataset, which is an image instance segmentation benchmark, using the standard metrics. We train on the train2017 set and evaluate on the val2017 and test-dev sets.

A. Instance Segmentation Results

We first compare YolactEdge to state-of-the-art real-time methods on YouTube VIS using the RTX 2080 Ti GPU in Table III. YOLACT [1] with a R101 backbone produces the highest box detection and instance segmentation accuracy over all competing methods. Our approach, YolactEdge, offers competitive accuracy to YOLACT, while running at a much faster speed (177.6 FPS on a R50 backbone). Even without the TensorRT optimization, it still achieves over 60 FPS for both R50 and R101 backbones, demonstrating the contribution of our partial feature transform design which allows the model to skip a large amount of redundant computation in video.

In terms of mask quality, because YOLACT/YolactEdge produce a final mask of size 138x138 directly from the feature maps without repooling (which potentially misalign the features), their masks for large objects are noticeably higher quality than Mask R-CNN. For instance, in Fig. 3, both YOLACT and YolactEdge produce masks that follow the boundary of the feet of lizard and zebra, while those of Mask R-CNN have more artifacts. This also explains YOLACT/YolactEdge's stronger quantitative performance

#Calib. Img.	mAP	FPS
0	24.4	–
5	29.6	27.4
50	**29.8**	27.4
100	29.7	**27.5**

(a) **INT8 calibration** Effect of the number of calibration images.

Warp layers	mAP	FPS
C_4, C_5	**39.2**	59.7
P_4, P_5	**39.2**	63.2
C_3, C_4, C_5	37.8	59.1
P_3, P_4, P_5	38.0	**64.1**

(b) **Partial feature transform** We warp P_4 & P_5 as it is both fast and accurate.

Channels	mAP	FPS
1x	**47.0**	48.3
1/2x	46.9	53.6
1/4x	46.9	61.2
1/8x	–	**62.2**

(c) **FeatFlowNet** We reduce channels for accuracy/speed tradeoff.

Method	mAP	FPS
w/o flow	31.8	**72.5**
FlowNetS	**39.2**	43.3
FeatFlowNet	**39.2**	61.2

(d) **FeatFlowNet** is faster and equally effective compared to FlowNetS.

TABLE VI: **Ablations**. (a) is on COCO val2017 using YOLACT with a R101 backbone. (b-d) are YolactEdge (w/o TRT) on our YouTube VIS sub-train/sub-val split ((b)&(d) without COCO joint training). We highlight our design choices in gray.

over Mask R-CNN on YouTube VIS, which has many large objects. Moreover, our proposed partial feature transform allows the network to take the computed high resolution C_3 features to help generate prototypes. In this way, our method is less prone to artifacts brought by misalignment compared to warping all features (as in [12]) and thus can maintain similar accuracy to YOLACT which processes all frames independently. See Fig. 4 for more qualitative results.

We next compare YolactEdge to YOLACT on the MS COCO [9] dataset in Table IV. Here YolactEdge is without video optimization since MS COCO is an image dataset. We compare three backbones: MobileNetv2, ResNet-50, and ResNet-101. Every YolactEdge configuration results in a loss of AP when compared to YOLACT due to the quantization of network parameters performed by TensorRT. This quantization, however, comes at an immense gain of FPS on the Jetson AGX and RTX 2080 Ti. For example, using ResNet-101 as a backbone results in a loss of 0.3 mask mAP from the unquantized model but results in a 20.7/88.3 FPS improvement on the AGX/RTX. We note that the MobileNetv2 backbone has the fastest speed (35.7 FPS on AGX) but has a very low mAP of 20.8 when compared to the other configurations.

Finally, Table V shows ablations of YolactEdge. Starting from YOLACT, which is equivalent to YolactEdge without TensorRT and video optimization, we see that with a ResNet-101 backbone, both our video and TensorRT optimizations lead to significant improvements in speed with a bit of degradation in mask/box mAP. The speed improvement for instantiations with a ResNet-50 backbone are not as prominent, because video optimization mainly exploits the redundancy of computation in the backbone stage and its effect diminishes in smaller backbones.

B. Which feature layers should we warp?

As shown in Table VIb, computing C_3/P_3 features (rows 2-3) yields 1.2-1.4 higher AP than warping C_3/P_3 features (rows 4-5). We choose to perform partial feature transform over P instead of C features, as there is no obvious difference in accuracy while it is much faster to warp P features.

C. FeatFlowNet

To encode pixel motion, FeatFlowNet takes as input C_3 features from the ResNet backbone. As shown in Table VIc, we choose to reduce the channels to 1/4 before it enters FeatFlowNet as the AP only drops slightly while being much faster. If we further decrease it to 1/8, the FPS does not increase by a large margin, and flow pre-training does not converge well. As shown in Table VId, accurate flow maps are crucial for transforming features across frames. Notably, our FeatFlowNet is equally effective for mask prediction as FlowNetS [15], while being faster as it reuses C_3 features for pixel motion estimation (whereas FlowNetS computes flow starting from raw RGB pixels).

D. Temporal Stability

Finally, although YolactEdge does not perform explicit temporal smoothing, it produces temporally stable masks.[1] In particular, we observe less mask jittering than YOLACT. We believe this is due to YOLACT only training on static images, whereas YolactEdge utilizes temporal information in videos both during training and testing. Specifically, when producing prototypes, our partial feature transform implicitly aggregates information from both the previous keyframe and current non-keyframe, and thus "averages out" noise to produce stable segmentation masks.

V. Discussion of Limitations

Despite YolactEdge's competitiveness, it still falls behind YOLACT in mask mAP. We discuss two potential causes.

a) Motion blur: We believe part of the reason lies in the feature transform procedure – although our partial feature transform corrects certain errors caused by imperfect flow maps (Table VIb), there can still be errors caused by motion blur which lead to mis-localized detections. Specifically, for non-keyframes, P_4 and P_5 features are derived by transforming features of previous keyframes. It is not guaranteed that the randomly selected keyframes are free from motion blur. A smart way to select keyframes would be interesting future work.

b) Mixed-precision conversion: The accuracy gap can also be attributed to mixed precision conversion – even with the optimal conversion and calibration configuration (Table I,VIa), the precision gap between training (FP32) and inference (FP16/INT8) is not fully addressed. An interesting direction is to explore *training* with mixed-precision, with which the model could potentially learn to compensate for the precision loss and adapt better during inference.

Acknowledgements. This work was supported in part by NSF IIS-1751206, IIS-1812850, and AWS ML research award. We thank Joohyung Kim for helpful discussions.

[1] See supplementary video: https://youtu.be/GBCK9SrcCLM.

REFERENCES

[1] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. Yolact: real-time instance segmentation. In *ICCV*, 2019.

[2] Youngwan Lee and Jongyoul Park. Centermask: Real-time anchor-free instance segmentation. *arXiv preprint arXiv:1911.06667*, 2019.

[3] Xinlong Wang, Rufeng Zhang, Tao Kong, Lei Li, and Chunhua Shen. Solov2: Dynamic, faster and stronger. *arXiv preprint arXiv:2003.10152*, 2020.

[4] Rufeng Zhang, Zhi Tian, Chunhua Shen, Mingyu You, and Youliang Yan. Mask encoding for single shot instance segmentation. *arXiv preprint arXiv:2003.11712*, 2020.

[5] Hao Chen, Kunyang Sun, Zhi Tian, Chunhua Shen, Yongming Huang, and Youliang Yan. Blendmask: Top-down meets bottom-up for instance segmentation. *arXiv preprint arXiv:2001.00309*, 2020.

[6] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. Yolact++: Better real-time instance segmentation. *TPAMI*, 2020.

[7] Sida Peng, Wen Jiang, Huaijin Pi, Hujun Bao, and Xiaowei Zhou. Deep snake for real-time instance segmentation. *arXiv preprint arXiv:2001.01629*, 2020.

[8] Nvidia tensorrt. https://developer.nvidia.com/tensorrt. Accessed: 2020.

[9] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014.

[10] Linjie Yang, Yuchen Fan, and Ning Xu. Video instance segmentation. In *ICCV*, 2019.

[11] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: Fully convolutional one-stage object detection. In *ICCV*, 2019.

[12] Xizhou Zhu, Yuwen Xiong, Jifeng Dai, Lu Yuan, and Yichen Wei. Deep feature flow for video recognition. In *CVPR*, 2017.

[13] Xizhou Zhu, Yujie Wang, Jifeng Dai, Lu Yuan, and Yichen Wei. Flow-guided feature aggregation for video object detection. In *ICCV*, 2017.

[14] Xizhou Zhu, Jifeng Dai, Lu Yuan, and Yichen Wei. Towards high performance video object detection. In *CVPR*, 2018.

[15] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. In *ICCV*, 2015.

[16] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. In *CVPR*, 2018.

[17] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018.

[18] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. *CoRR*, abs/1905.02244, 2019.

[19] Golnaz Ghiasi, Tsung-Yi Lin, Ruoming Pang, and Quoc V. Le. NAS-FPN: learning scalable feature pyramid architecture for object detection. *CoRR*, abs/1904.07392, 2019.

[20] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019.

[21] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[22] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. *CoRR*, abs/1802.05668, 2018.

[23] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108, 2019.

[24] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ICLR*, 2016.

[25] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size. *arXiv:1602.07360*, 2016.

[26] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016.

[27] Adrian Bulat and Georgios Tzimiropoulos. Xnor-net++: Improved binary neural networks. In *BMVC*, 2019.

[28] Sergei Alyamkin, Matthew Ardi, Alexander C. Berg, Achille Brighton, Bo Chen, Yiran Chen, Hsin-Pai Cheng, Zichen Fan, Chen Feng, Bo Fu, Kent Gauen, Abhinav Goel, Alexander Goncharenko, Xuyang Guo, Soonhoi Ha, Andrew Howard, Xiao Hu, Yuanjun Huang, Donghyun Kang, Jaeyoun Kim, Jong-gook Ko, Alexander Kondratyev, Junhyeok Lee, Seungjae Lee, Suwoong Lee, Zichao Li, Zhiyu Liang, Juzheng Liu, Xin Liu, Yang Lu, Yung-Hsiang Lu, Deeptanshu Malik, Hong Hanh Nguyen, Eunbyung Park, Denis Repin, Liang Shen, Tao Sheng, Fei Sun, David Svitov, George K. Thiruvathukal, Baiwu Zhang, Jingchi Zhang, Xiaopeng Zhang, and Shaojie Zhuo. Low-power computer vision: Status, challenges, opportunities. *CoRR*, abs/1904.07714, 2019.

[29] Tensorrt hardware support matrix. https://docs.nvidia.com/deeplearning/tensorrt/support-matrix/index.html#hardware-precision-matrix. Accessed: 2020.

[30] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *CVPR*, 2017.

[31] Tensorrt int8 calibration. https://on-demand.gputechconf.com/gtc/2017/presentation/s7310-8-bit-inference-with-tensorrt.pdf. Accessed: 2020.

[32] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *ICCV*, 2017.

[33] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. v.d. Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. In *ICCV*, 2015.