ACCV2020
#422

Real-Time Segmentation Networks should be Latency Aware

Anonymous ACCV 2020 submission

Paper ID 422

Abstract. As scene segmentation systems reach visually accurate results, many recent papers focus on making these network architectures faster, smaller and more efficient. In particular, studies often aim at designing 'real-time' systems. Achieving this goal is particularly relevant in the context of real-time video understanding for autonomous vehicles, and robots.

In this paper, we argue that the commonly used performance metric of mean Intersection over Union (mIoU) does not fully capture the information required to estimate the true performance of these networks when they operate in 'real-time'. We propose a change of objective in the segmentation task, and its associated metric that encapsulates this missing information in the following way: We propose to predict the future output segmentation map that will match the *future* input frame at the time when the network finishes the processing. We introduce the associated latency-aware metric, from which we can also determine a ranking.

We perform latency timing experiments of some recent networks on different hardware and assess the performances of these networks on our proposed task. We propose improvements to scene segmentation networks to better perform on our task by using multi-frames input and increasing capacity in the initial convolutional layers.

1 Introduction

Recent image segmentation networks achieve near-human level performance due to their expressive power, and more focus is on designing architectures that are faster, and can run on smaller hardware with less memory and computing power. In particular, enabling real-time segmentation is critical for applications in robotics, autonomous driving or medical imaging during surgery.

The primary way currently used to assess performance is a task whose objective is the prediction of the input frame's segmentation, which is compared to the input frame's ground-truth segmentation using a given metric (*e.g.* mIoU). In what follows, we will use 'accuracy' to refer to such a metric. For networks aiming at low-latency, researchers also estimate efficiency with the *Frames Per Second* (FPS) metric, or its inverse the Seconds Per Frame metric, also called *latency*.

On real-time segmentation benchmarks, networks are ranked according either to some accuracy metric or latency. Often, accuracy-latency charts also allows

to quickly estimate a new network overall performances. However, we claim there still is critical information missing to the practitioner: What is the actual accuracy of the system when deployed and used in practice? In other terms, we want to help answer the question of how the system's latency will affect the relevance of its predictions.

We propose an intuitive extension to the usual video segmentation task by introducing a change in the objective. We change the goal from predicting input frame segmentation to predicting future frame segmentation. Going beyond introducing a useful metric, our 'latency-aware' task aims at encouraging researchers to focus on a more relevant goal for real-time contexts, i.e. designing *anticipatory* networks.

The change we propose in the objective definition is straightforwardly applicable to a wide range of problem domains (*e.g.* object tracking, object detection, object segmentation, pose estimation). In the remainder of this paper, we will focus on the scene semantic segmentation task and perform our experiments on it.

Our contributions are as follows:

- We propose a simple, and relevant task that aims to assess actual performance of real-time networks,
- we highlight the associated metric and discuss its benefits,
- we analyse the relevance of the metric through multiple experiments on different scene segmentation networks,
- we propose improvements to a fast image-segmentation network for better performance on our task by taking multiple frames as input and increasing the number of channels of early convolutional layers.

We will make our code publicly available at the time of the conference.

2 Related Work

2.1 Image Semantic Segmentation

Most popular approaches for tackling Semantic Segmentation use a variant of powerful deep classification networks that are made fully convolutionnal, with all final fully connected layers replaced by convolutions. That seminal idea is at the core of the FCN paper [1].

The main issue coming with this technique is that it significantly reduces the image resolution in order to retrieve semantic information. Subsequent models for semantic segmentation are built as a "fully convolutional network" and attempt to cope with the dimension reductions, while increasing the Receptive Field.

One commonly used techniques is to use a *decoder network* plugged after the FCN to upsample the segmentation map using transposed convolution, as first did [2] and [3] with SegNet and U-Net. This setup allows to merge spatially rich shallow layers into semantically rich deeper layers.

ACCV2020
#422

DeepLab v2[4] later proposed to use *dilated convolutions* [5] to avoid downsampling. This allows to process images with a large field of view without having to reduce them, but it comes with a larger computational complexity.

Finally, [6] proposed to use a "*Spatial Pyramidal Pooling*" (SPP) module [7] for segmentation. SPP pools the image simultaneously at different resolutions over a grid, therefore enlarging the Receptive Field. This allows to incorporate a larger context, and take into account higher-level semantic.

Many works followed with techniques to produce high-quality segmentation [8–11], including better ways to extract features [12–15] and to take into account context [16, 17]. Some recent works also proposed attention-based methods [18–23] and neural architecture search for image segmentation [24–26].

2.2 Real-time Semantic Segmentation

Reducing the computational cost and the memory cost of deep segmentation systems is critical for many applications that need to run real-time on slow hardware. A precursor in fast segmentation is ICNet [27], which is a fast network that uses multi-scale processing with a special fuse block to merge multi-scale information.

One way of optimising neural network architecture for speed is by using factorised convolutional blocks, *e.g.* factorizing kernels $k \times k$ into $1 \times k$ and $k \times 1$ kernels as does ERFNet [28]. It can also be achieved using group convolutions, and methods such as ShuffleNet [29] propose different ways to create connections between groups.

One can also use depthwise separable convolution (DSC), which are the combination of depthwise and pointwise convolutions. These DSC are used to lower the number of parameters and makes the inference faster, at the cost of accuracy. They are used broadly in MobileNets [30, 31].

Another important idea of these networks is to quickly downsample images in order to perform most of the processing at a smaller resolution and avoid full resolution processing. This idea is key to the design of ENet [32].

BiSeNet and BisenetV2 [33, 34] proposed a way to separate the localization problem from the semantic extraction problem, and then to merge the two information appropriately.

Recent work such as FasterSeg network [35] also use Neural Architecture Search to succesfully discover fast neural architectures for semantic segmentation.

Among fast segmentation networks, Swiftnet [36] is another recent work that proposes an architecture with a light-weight ImageNet-pretrained Resnet followed by a simple decoder using lateral connections similarly to U-Net. For our work, we choose SwiftNet as one of our base networks for its simplicity and its speed.

2.3 Video Scene Segmentation Networks

Another part of the literature focuses on designing *video* segmentation systems. More specifically, these works try to leverage the temporal correlation of consecutive frames in a video to improve the next-frame prediction and reduce

computation and latency. However, most works in this domain are more focused on improving segmentation accuracy than reducing the latency.

The Clockwork net in [37] is a model that leverages temporal correlation by running different parts of the network at each time-step conditionally to how much the video has changed from the previous frame. This technique has the disadvantage of not providing a fixed frame-rate.

Another direction to address the problem is to try propagating previous features to consecutive frames to avoid recomputing very similar features for following frames, as is done in [38], even though their design is not meant for real-time.

The work [39] built on these two previous ideas. Their network decides at each frame whether to propagate previous features or to recompute the entire segmentation map. They improved the clockwork design to reduce the maximum latency but did not reach real-time.

Other works use predictive learning, that is predicting future frames or flow motion using past frames and segmentations to help current segmentation [40, 41].

Video temporal coherence is also used along with representation warping in [42] to produce better output segmentation maps. This work is not focused on time efficiency.

Recently, a Temporally Distributed Network [43] was proposed for fast video segmentation. It uses a teacher-student design where fast student networks have to predict in turns part of the feature map of the teacher network.

3 A new task for real-time networks

3.1 Motivation for latency awareness

Real-time network performance is usually assessed through accuracy-latency charts that help in understanding a network's trade-offs. These charts provide *instant accuracy* of networks, *i.e.* the accuracy between the network's prediction and the input's ground truth. In practice however, networks may need a few seconds before they make a prediction. During that time, the scene has changed and the network prediction does not match that change. It is then particularly useful to compare the network prediction with this new scene's segmentation. This important comparison is missing from latency-accuracy charts as they do not provide the actual accuracy (compensated for time-delay) that one will get in practice. More, it does not provide neither a total order relation nor a ranking to compare various real-time networks, as can be seen on benchmark websites such as *paperswithcode.com*[1], We believe that it is therefore relevant to introduce a new objective for the segmentation task that takes into account network latency. This allows to get a meaningful accuracy information and practical benchmarking of networks.

[1] https://paperswithcode.com/task/video-object-segmentation

3.2 Defining a new objective for real-time networks

We propose to change the objective of the segmentation task: currently, the objective of the task is to predict the input frame segmentation. Instead, we propose as objective to predict the segmentation of the "future" frame *at the time the network finishes its computation.*

Formally, let us consider a video sequence and let I_t and S_t denote respectively the frame at time t and its ground truth segmentation. Let F denote the operation of a network (say semantic segmentation) that takes l_F milliseconds to process the current input I_t. The common objective is to improve the metric:

$$\text{acc}\left(F(I_t), S_t\right) \tag{1}$$

while our task proposes to optimise for:

$$\text{acc}\left(F(I_t), S_{t+l_F}\right) \tag{2}$$

Instead of predicting the segmentation of the input frame, our task expects systems to predict the segmentation of a future frame, thus acknowledging the network latency.

This objective is particularly relevant for real-time applications in which we are usually more interested in what is currently happening than in what was a few seconds before : it is useful compare the information we get at a given time t using a network $(F(I_{t-l_F}))$ with the information we ideally would like to get at that time (S_t).

As said earlier, this change of objective is applicable to a whole range of different tasks such as object segmentation, object detection, object tracking, pose detection, etc. We focus on the scene segmentation task for this work.

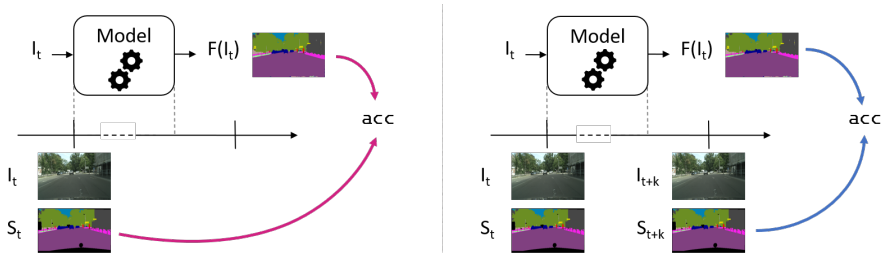

Fig. 1: Left: How mIoU is computed now; the output of the network is compared to the ground truth segmentation of the *image in input of the network*. Right: Our proposed way to measure mIoU; the output of the network is compared to the ground truth segmentation of the *future image* when network finishes processing.

3.3 Corresponding Latency-Aware Metric

For scene semantic segmentation, the metric commonly used is the mean Intersection over Union (mIoU). Our new task naturally defines a metric that depends on the latency of the network. We term it "Latency-Aware mIoU" (LAmIoU), which is defined as per Eq. (2).
Considering this metric is interesting as it carries an additional practical meaning compared to the classical instant mIoU: the accuracy (LAmIoU) that this metric outputs is the accuracy that one will see in practice when running this network in a real-time setting on the given hardware device.

3.4 Use with video datasets

In practice, a video sequence is collected with a specific sampling frequency (there is some time delay d between two sampled frames), and thus the dataset does not have a frame for every time t. For our task we therefore use the frame sampled just *after* the model has output a prediction as shown in Fig. 2.

More precisely, let's assume that $t = 0$ when frame of index 0 enters the network and consider a video sequence with a delay d between two frames (fps $= 1/d$). Then, the index of the segmentation map that the metric would use as ground truth is:

$$k_F = \lceil l_F/d \rceil \tag{3}$$

In what follows, when we refer to $t + k_F \times d$, we will simplify notation and write $t + k_F$.

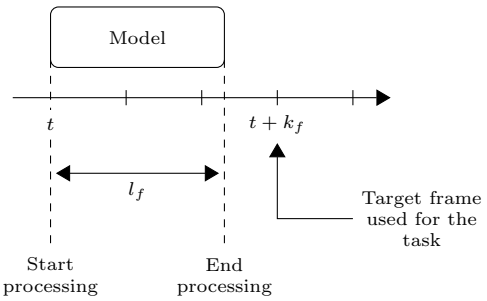

Fig. 2: Target frame used for our proposed objective

Note that the performance on this task is hardware-dependent. Indeed, as network latency depends on the device, the frame used for metric computation depends on it as well. Knowing the value of this metric for a network on multiple hardware allows one to pick the right network and hardware depending on precision needs.

4 Dataset, Models and Experimental setup

4.1 Dataset

Dense pixel-level manual annotation of videos for scene segmentation is not feasible due to the time and economic costs involved. Cityscapes dataset [44] was estimated to take about 90 minutes per-frame for annotation and verification, and thus only provides sparse annotations of one frame per video sequence. We choose this dataset to conduct our experiments as it contains video sequences and its use is widespread as a segmentation benchmark.

This dataset contains 2,975 training, 500 validation and 1,525 testing video sequences. Each sequence contains 30 frames, and the 20^{th} frame is annotated with fine pixel-level class labels for 19 object categories. A sequence is 1.8s long, therefore the frame rate is approximately 16.6 fps and there is about 60ms between each frame. As Cityscapes contains only the ground truth segmentation for one image per sequence, we process as follows:

1. We time the latency l_F of the network
2. We determine how many frames of offset k_F this time corresponds to: $k_F = \lceil l_F/0.06 \rceil$ ($0.06 = 60$ms)
3. We use as input of the network the frame of index $20 - k_F$, as we only have the 20^{th} frame's ground truth segmentation

Note that the Cityscapes framerate is about half of the one usually encountered in videos. This may be slightly detrimental as the higher the framerate is, the more precise the latency-aware metric will be.

4.2 Networks

For our experiments, we choose two image segmentation networks: SwiftNet [36] and DeepLab-V3+ [45] with 2 different encoders : ResNet-101 and MobileNet v2 [31].

SwiftNet SwiftNet is a state-of-the-art network in real-time segmentation. For our experiments, we build this network as detailed in the original paper and describe it below. It is a network with an encoder-decoder structure:

- The encoder backbone is a classical ResNet-18 whose fully connected layers have been removed to make it fully convolutionnal.
- A Spatial Pyramidal Pooling Module with 4 different pooling layers of grid size (1,2,4,8) is plugged in output of the encoder to increase its receptive field.
- Finally, a decoder with 3 upsampling modules recovers original image resolution. An upsampling module upsamples the previous layer's output and then merges it with a skip connection coming from the encoder.

We will refer to it as **SwiftNet-R18**. It has approximately 12M parameters. On Cityscapes validation set, it reaches 75% mIoU and runs at about 40 fps on a GTX 1080 Ti. On this hardware, SwiftNet has a latency of 26 ms. This means we have to use $k_F = \lceil 26/60 \rceil = 1$ frame offset to compute the latency-aware mIoU.

ACCV2020
#422

DeepLab v3+ DeepLab v3+ is a state of the art network for image segmentation. It has an encoder-decoder architecture very similar to that of SwiftNet:

- We use two different encoder backbones :
  - A dilated ResNet-101 network stripped of its fully connected layers. We use an output stride of 16, so the last two residual blocks make use of dilated convolutions to enlarge the receptive field.
  - A MobileNet-V2 network as described in [31]. It uses depthwise separable convolutions within "inverted" residuals blocks separated by linear bottlenecks.
- An Atrous Spatial Pyramidal Pooling module is plugged after this encoder. It convolves the encoder output with 4 atrous convolutions using different dilation rates: (1, 6, 12, 18).
- Finally, a small decoder upsamples the ASPP output and concatenates it with low-level features from the encoder. The decoder blends them with a convolution and upsamples the output to the original input image size.

When using a ResNet-101 as backbone, the whole network has approximately 60M parameters and reaches 77% mIoU on Cityscapes validation set and runs at 5 fps. We will refer to it as **DeepLab-R101**. On our hardware, it has a latency of 195 ms, which means we have to use $k_F = \lceil 195/60 \rceil = 4$ frame offsets to compute the latency-aware mIoU.

When using a MobileNet backbone, the model has about 5.5M parameters. It reaches 72% mIoU on Cityscapes and runs at 13 fps. We will refer to it as **DeepLab-MN.** It has a latency of 76 ms, so we have to use $k_F = \lceil 76/60 \rceil = 2$ frame offsets.

4.3 Adapting SwiftNet for our task

As we will discuss further, it is useful to input previous frames along with the current frame when training to predict a future segmentation map (that corresponds to the latency-aware objective). When we added more input channels to the first layer of the network, we noticed it is beneficial to increase the initial layers capacity by enlarging the number of channels. We construct a variant of SwiftNet that takes multiple frames in input, and where we expand the number of kernels in the initial convolution layers to deal with the increased input size. Specifically, in the case of two input frames, we replace the first layer:

$$\text{conv}(3, 64, 7 \times 7, s = 2)$$

with the following block of four layers:

$$\begin{gathered}\text{conv}(6, 130, 7 \times 7, s = 2)\\ \text{BN}(130)\\ \text{ReLU}\\ \text{conv}(130, 64, 3 \times 3, s = 1)\end{gathered}$$

ACCV2020
#422

Note that we cannot introduce changes affecting multiple encoder layers as this would prevent us from reusing pretrained weights for the ResNet encoder, which represents SwiftNet's main strength.

The newly created convolutions were initialized using He's initialization [46] rule. We will refer to our extended SwiftNet version as **SwiftNet-R18-X**.

4.4 Training

All experiments are performed with the PyTorch framework. We use ImageNet-1k pretrained weights for all encoders in our networks.

Data augmentation We use image crops of 768×768. We do standard image augmentation with random horizontal flip, random scaling from 0.75 to 1.5 and random Gaussian blur.

SwiftNet For SwiftNet, we use a batch size of 12 and train using the Adam optimiser with default parameters. We use a learning rate of $5e{-}4$ and a weight decay of $1e{-}4$. We also set a smaller learning rate of $1e{-}4$ for the part that is ImageNet-pretrained. We train the network for 200 epochs and use a cosine annealing schedule with $\eta_{min} = 1e{-}6$.

DeepLab v3+ For DeepLab, we use a batch size of 10 and train using the SGD optimiser with momentum of 0.9. We use a learning rate of $5e{-}2$ and a weight decay of $5e{-}4$. We similarly set a smaller learning rate of $5e{-}3$ for the part that is ImageNet-pretrained. We train the network for 200 epochs and use a poly schedule with a power of 3.

5 Experiments and Results

We perform experiments to evaluate the effect of network latency on the LAmIoU and to understand the changes in the training to suit the proposed objective.

5.1 Effect of latency on the LAmIoU

The experiments described in this subsection are performed with *DeepLab-R101* and *SwiftNet-R18*. These two networks are modified to accept in input 2 frames X_{t-1}, X_t, as will be detailed later.

Decay of the LAmIoU with the frame offset In Fig. 3, we plot the LAmIoU vs frame offsets. The two networks considered here are both trained and evaluated on the future segmentation map (with offset).

We notice that the LAmIoU drops quickly and the decrease is consistent between networks: an offset of 2 frames is enough to lose 10% mIoU for both networks on Cityscapes. In practice, we can expect this order of magnitude of mIoU drop, depending on the hardware.

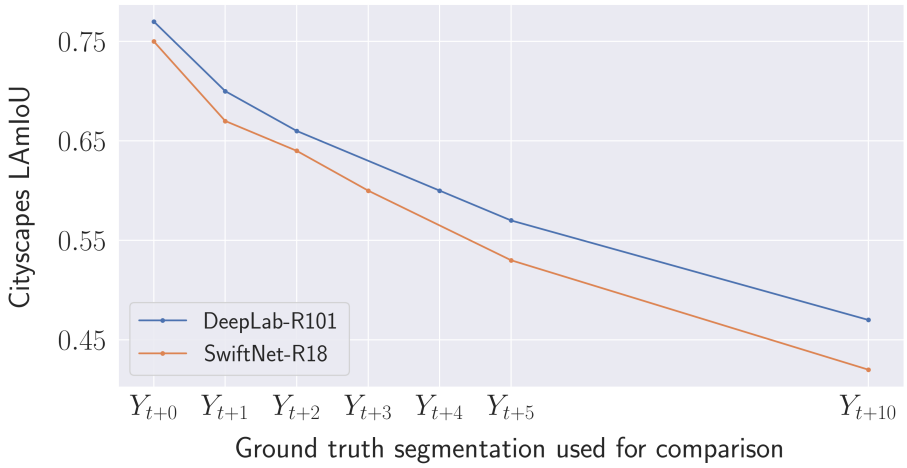

Fig. 3: LAmIoU decay with different offsets in the objective on Cityscapes validation set

Decay of the LAmIoU with the hardware Each network has a different latency per hardware. Therefore, the frame offset used for training and computing the metric is also different per hardware.

We perform timing experiments on *Tesla V100*, *GTX 1080 Ti* and *Tesla K80* GPUs for our two networks. We estimate latencies on these hardware and then train the networks with the corresponding frame offsets. In Fig. 4, we report the LAmIoU with respect to the inverse hardware speed (inverse of flops).

This plot exhibits an interesting and foreseeable fact: the slower deeplab network, whose "instant" mIoU is higher, performs worse than the faster SwiftNet network on slow hardware when measuring the LAmIoU. This graph illustrates the need for a latency aware metric in real-time contexts, which allows for a simple and fairer comparison of networks.

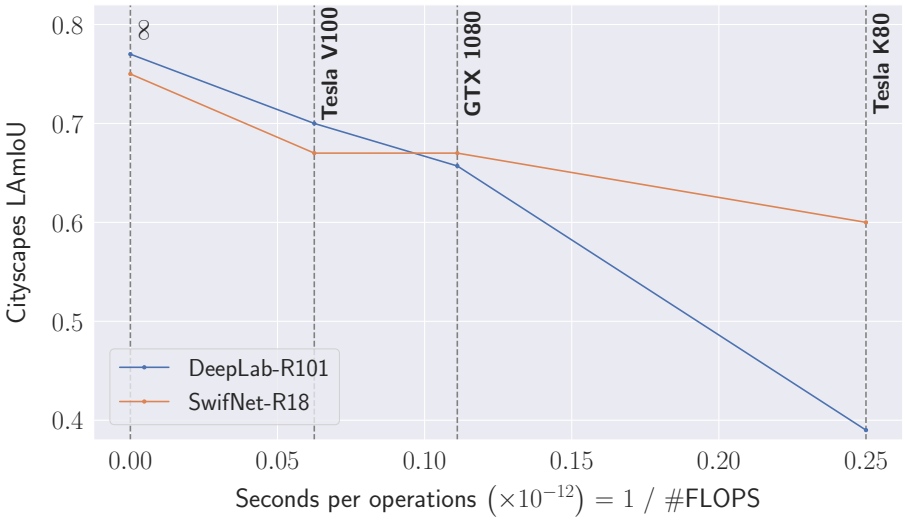

Fig. 4: LAmIoU decay with hardware speed on Cityscapes validation set

5.2 Optimising network training for our latency-aware objective

In this subsection, we investigate how changing the inputs and training target affects the LAmIoU. Precisely, we train using three different configurations and comment on the noticeable differences in the networks output. These experiments are performed on a *GTX 1080 Ti* GPU for each of the four networks described in Section 4.2.

First configuration First, we evaluate the four networks when trained with the usual objective (input I_t and target S_t) but evaluated with LAmIoU. The results are reported in second line of Table 1. Compared to their instant mIoU, we notice a significant drop between 10% and 30 %.

Second configuration We train the networks for the proposed objective of predicting the future segmentation ground-truth used by the LAmIoU (input I_t and target S_{t+k}). The value of k is different for each network since each has a different latency. Therefore, they are trained and evaluated with a differently offset ground-truth.

The results are reported in third line of Table 1. We can see a slight but consistent increase of the LAmIoU metric for all networks. In Fig. 5, we show some qualitative results of SwiftNet-R18 overlaying images I_t and I_{t+1}. We notice that the segmentation mask is slightly blurry, as one would expect. However, we note that the blur is often surprisingly anisotropic, *i.e.* the segmentation blur is not surrounding the object, but favours a specific direction.

We conjecture that the network is able to predict some objects' movement based on their orientation. For instance, a person facing left in image I_t is likely to have moved left in the next image I_{t+1}. Similarly, a car facing the camera is more likely to be coming toward the camera, and thus is probably going to look bigger in the following frame.

(a) I_t and $F(I_t)$ (b) I_{t+1} and $F(I_t)$ (c) I_{t+1} and S_{t+1} (G.T.)

Fig. 5: Output segmentation of SwiftNet-R18 trained to predict S_{t+1} from I_t. We observe anisotropic blur as the network is able to infer some objects movement directions from their orientation.

Third configuration Finally, in order to allow the networks to infer speeds and directions, we train them using both I_{t-1} and I_t as inputs. The training objective remains to predict the future segmentation ground-truth S_{t+k}. Results reported in fourth line of Table 1 show a consistent improvement of the LAmIoU metric. These numbers confirms the relevance of using previous frames for our latency-aware objective. We reported in Fig. 6 examples of the output segmentation of SwiftNet-R18 overlaid on image I_t and I_{t+1} where it is clear that using an additional input is useful to produce sharper and more accurate segmentation maps.

For practical applications, we have seen that it is relevant to consider a future ground-truth as objective. When doing so, we need to change our training objective, and the result of this last configuration shows that it becomes necessary to use previous frames to get better predictions. While this result may not be surprising, the great majority of real-time scene segmentation works only use the current input when designing and training their networks. This last result

emphasises the fact that networks constructed for real-time contexts should use previous frames to better predict a future target.

(a) I_t and $F(I_t)$ (b) I_{t+1} and $F(I_t)$ (c) I_{t+1} and S_{t+1} (G.T.)

Fig. 6: Output segmentation of SwiftNet-R18 trained to predict S_{t+1} from (I_{t-1}, I_t). We observe a more precise segmentation as the network has a way to infer relative speeds and directions.

5.3 Input translations experiment for increased Receptive Field

When processing simultaneously images from different time-steps I_{t-1} and I_t, it is important for the network to have a large receptive field (to be able to map objects from one frame to the other). To do so without the need of big kernels, we try to offset this load on the input. The idea is to trade part of the computational cost usually associated with the use of big convolutional kernels for the memory cost of having more inputs.

Practically, we concatenate to the current input of the network various translations of the previous image I_{t-1}. When introducing translations, we want to compensate for the use of big kernels by allowing the model to simultaneously attend different parts of the image that a normal convolution kernel would not process simultaneously. Particularly, we change the inputs $\{I_{t-1}, I_t\}$ of the previous experiment to $\{T_1(I_{t-1}), \cdots, T_N(I_{t-1}), I_{t-1}, I_t\}$. corresponding to N different fixed translations T_i. The translations offsets were chosen to span a regular grid around the origin.

While initial experiments seemed promising, we further discovered that only the reason for improved LAmIoU was the additional convolutional layer with higher number of kernels that we added at the head of the model to handle the

Table 1: Results for the 3 training configurations for the 4 networks. The first line gives the instant mIoU, the next three lines reports value of the LAmIoU metric for different training configurations. The last three lines give information about latency, frame offset used for the network as explained in Section 4.1, and fps of these networks. Note that the k temporal offset depends on the network, hardware and on the dataset framerate: this offset is greater and leads to poorer performance for slow processing.

	Input	Target (train)	Target (test)	DeepLab-R101	DeepLab-MN	SwiftNet-R18	SwiftNet-R18-X
	I_t	S_t	S_t	**0.77**	0.72	0.75	0.74
LAmIoU	I_t	S_t	S_{t+k}	0.5	0.56	**0.64**	0.63
LAmIoU	I_t	S_{t+k}	S_{t+k}	0.53	0.57	**0.65**	0.64
LAmIoU	I_{t-1}, I_t	S_{t+k}	S_{t+k}	0.60	0.58	0.67	**0.69**
	Frame offset (k)			4	2	1	1
	Latency (ms)			195	76	26	38
	FPS			5	13	38	26

translations where we had replaced the first convolution layer conv(3, 64, 7×7, $s = 2$) with the following block:

$$\begin{aligned} &\text{conv}(6 + 3 \times N, 8 \times N, 7 \times 7, s = 2) \\ &\text{BN}(8 \times N) \\ &\text{ReLU} \\ &\text{conv}(8 \times N, 64, 3 \times 3, s = 1). \end{aligned}$$

with N the number of translations. Noticing that using additional inputs requires increasing capacity in the early convolutionnal layers eventually lead to the design of SwiftNet-R18-X, in which we increased the number of kernels in the first two layers.

6 Conclusion

We proposed a change in the usual objective of the segmentation task that makes real-time networks account for their latency when making their predictions. In addition to providing a new latency-aware ranking, the associated LAmIoU metric is of particular practical relevance as it represents the actual mIoU value one may obtain on a given hardware when taking into account network latency.

We argued the reasons why real-time networks should be latency-aware and we believe introducing a new latency-aware segmentation objective encourages research in anticipatory networks. While the focus of this paper is specifically toward video scene segmentation, the same change of objective is relevant and applicable to a wide range of other "real-time tasks" not limited to computer vision.

References

1. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. (2015) 3431–3440
2. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical image computing and computer-assisted intervention, Springer (2015) 234–241
3. Badrinarayanan, V., Kendall, A., Cipolla, R.: Segnet: A deep convolutional encoder-decoder architecture for image segmentation. IEEE transactions on pattern analysis and machine intelligence **39** (2017) 2481–2495
4. Chen, L.C., Papandreou, G., Kokkinos, I., Murphy, K., Yuille, A.L.: Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. IEEE transactions on pattern analysis and machine intelligence **40** (2017) 834–848
5. Yu, F., Koltun, V., Funkhouser, T.: Dilated residual networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. (2017) 472–480
6. Zhao, H., Shi, J., Qi, X., Wang, X., Jia, J.: Pyramid Scene Parsing Network (PSPNet). arXiv:1612.01105 [cs] (2016) arXiv: 1612.01105.
7. He, K., Zhang, X., Ren, S., Sun, J.: Spatial pyramid pooling in deep convolutional networks for visual recognition. IEEE transactions on pattern analysis and machine intelligence **37** (2015) 1904–1916
8. Tang, M., Perazzi, F., Djelouah, A., Ben Ayed, I., Schroers, C., Boykov, Y.: On regularized losses for weakly-supervised cnn segmentation. In: Proceedings of the European Conference on Computer Vision (ECCV). (2018) 507–522
9. Takikawa, T., Acuna, D., Jampani, V., Fidler, S.: Gated-scnn: Gated shape cnns for semantic segmentation. In: Proceedings of the IEEE International Conference on Computer Vision. (2019) 5229–5238
10. Zhu, Y., Sapra, K., Reda, F.A., Shih, K.J., Newsam, S., Tao, A., Catanzaro, B.: Improving semantic segmentation via video propagation and label relaxation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2019) 8856–8865
11. Valada, A., Mohan, R., Burgard, W.: Self-supervised model adaptation for multimodal semantic segmentation. International Journal of Computer Vision (2019) 1–47
12. Peng, C., Zhang, X., Yu, G., Luo, G., Sun, J.: Large kernel matters–improve semantic segmentation by global convolutional network. In: Proceedings of the IEEE conference on computer vision and pattern recognition. (2017) 4353–4361
13. Zhang, Z., Zhang, X., Peng, C., Xue, X., Sun, J.: Exfuse: Enhancing feature fusion for semantic segmentation. In: Proceedings of the European Conference on Computer Vision (ECCV). (2018) 269–284
14. He, J., Deng, Z., Qiao, Y.: Dynamic multi-scale filters for semantic segmentation. In: Proceedings of the IEEE International Conference on Computer Vision. (2019) 3562–3572
15. Wu, H., Zhang, J., Huang, K., Liang, K., Yu, Y.: Fastfcn: Rethinking dilated convolution in the backbone for semantic segmentation. arXiv preprint arXiv:1903.11816 (2019)
16. Ding, H., Jiang, X., Shuai, B., Qun Liu, A., Wang, G.: Context contrasted feature and gated multi-scale aggregation for scene segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2018) 2393–2402

17. Yuan, Y., Chen, X., Wang, J.: Object-contextual representations for semantic segmentation. arXiv preprint arXiv:1909.11065 (2019)
18. Li, Y., Chen, X., Zhu, Z., Xie, L., Huang, G., Du, D., Wang, X.: Attention-guided unified network for panoptic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2019) 7026–7035
19. Fu, J., Liu, J., Tian, H., Li, Y., Bao, Y., Fang, Z., Lu, H.: Dual attention network for scene segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2019) 3146–3154
20. Huang, Z., Wang, X., Huang, L., Huang, C., Wei, Y., Liu, W.: Ccnet: Criss-cross attention for semantic segmentation. In: Proceedings of the IEEE International Conference on Computer Vision. (2019) 603–612
21. Tao, A., Sapra, K., Catanzaro, B.: Hierarchical multi-scale attention for semantic segmentation. arXiv preprint arXiv:2005.10821 (2020)
22. Zhang, H., Wu, C., Zhang, Z., Zhu, Y., Zhang, Z., Lin, H., Sun, Y., He, T., Mueller, J., Manmatha, R., et al.: Resnest: Split-attention networks. arXiv preprint arXiv:2004.08955 (2020)
23. Choi, S., Kim, J.T., Choo, J.: Cars can't fly up in the sky: Improving urban-scene segmentation via height-driven attention networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. (2020) 9373–9383
24. Liu, C., Chen, L.C., Schroff, F., Adam, H., Hua, W., Yuille, A.L., Fei-Fei, L.: Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. (2019) 82–92
25. Zhang, X., Xu, H., Mo, H., Tan, J., Yang, C., Ren, W.: Dcnas: Densely connected neural architecture search for semantic image segmentation. arXiv preprint arXiv:2003.11883 (2020)
26. Nekrasov, V., Chen, H., Shen, C., Reid, I.: Architecture search of dynamic cells for semantic video segmentation. In: The IEEE Winter Conference on Applications of Computer Vision. (2020) 1970–1979
27. Zhao, H., Qi, X., Shen, X., Shi, J., Jia, J.: ICNet for Real-Time Semantic Segmentation on High-Resolution Images. arXiv:1704.08545 [cs] (2017) arXiv: 1704.08545.
28. Romera, E., Alvarez, J.M., Bergasa, L.M., Arroyo, R.: Erfnet: Efficient residual factorized convnet for real-time semantic segmentation. IEEE Transactions on Intelligent Transportation Systems **19** (2017) 263–272
29. Zhang, X., Zhou, X., Lin, M., Sun, J.: Shufflenet: An extremely efficient convolutional neural network for mobile devices. In: Proceedings of the IEEE conference on computer vision and pattern recognition. (2018) 6848–6856
30. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv:1704.04861 [cs] (2017) arXiv: 1704.04861.
31. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. (2018) 4510–4520
32. Paszke, A., Chaurasia, A., Kim, S., Culurciello, E.: Enet: A deep neural network architecture for real-time semantic segmentation. arXiv preprint arXiv:1606.02147 (2016)
33. Yu, C., Wang, J., Peng, C., Gao, C., Yu, G., Sang, N.: BiSeNet: Bilateral Segmentation Network for Real-time Semantic Segmentation. arXiv:1808.00897 [cs] (2018) arXiv: 1808.00897.

34. Yu, C., Gao, C., Wang, J., Yu, G., Shen, C., Sang, N.: Bisenet v2: Bilateral network with guided aggregation for real-time semantic segmentation. arXiv preprint arXiv:2004.02147 (2020)
35. Chen, W., Gong, X., Liu, X., Zhang, Q., Li, Y., Wang, Z.: Fasterseg: Searching for faster real-time semantic segmentation. arXiv preprint arXiv:1912.10917 (2019)
36. Orsic, M., Kreso, I., Bevandic, P., Segvic, S.: In defense of pre-trained imagenet architectures for real-time semantic segmentation of road-driving images. In: Proceedings of the IEEE conference on computer vision and pattern recognition. (2019) 12607–12616
37. Shelhamer, E., Rakelly, K., Hoffman, J., Darrell, T.: Clockwork Convnets for Video Semantic Segmentation. arXiv:1608.03609 [cs] (2016) arXiv: 1608.03609.
38. Zhu, X., Xiong, Y., Dai, J., Yuan, L., Wei, Y.: Deep Feature Flow for Video Recognition. arXiv:1611.07715 [cs] (2016) arXiv: 1611.07715.
39. Li, Y., Shi, J., Lin, D.: Low-latency video semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2018) 5997–6005
40. Jin, X., Li, X., Xiao, H., Shen, X., Lin, Z., Yang, J., Chen, Y., Dong, J., Liu, L., Jie, Z., et al.: Video scene parsing with predictive feature learning. In: Proceedings of the IEEE International Conference on Computer Vision. (2017) 5580–5588
41. Jin, X., Xiao, H., Shen, X., Yang, J., Lin, Z., Chen, Y., Jie, Z., Feng, J., Yan, S.: Predicting scene parsing and motion dynamics in the future. In: Advances in Neural Information Processing Systems. (2017) 6915–6924
42. Gadde, R., Jampani, V., Gehler, P.V.: Semantic Video CNNs through Representation Warping. arXiv:1708.03088 [cs] (2017) arXiv: 1708.03088.
43. Hu, P., Caba, F., Wang, O., Lin, Z., Sclaroff, S., Perazzi, F.: Temporally distributed networks for fast video semantic segmentation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. (2020) 8818–8827
44. Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., Schiele, B.: The cityscapes dataset for semantic urban scene understanding. In: Proceedings of the IEEE conference on computer vision and pattern recognition. (2016) 3213–3223
45. Chen, L.C., Zhu, Y., Papandreou, G., Schroff, F., Adam, H.: Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation (DeepLabv3+). arXiv:1802.02611 [cs] (2018) arXiv: 1802.02611.
46. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision. (2015) 1026–1034