PAUMER: Patch Pausing Transformer for Semantic Segmentation

Evann Courdier*[1,2]
evann.courdier@idiap.ch
Prabhu Teja S*[1,2]
prabhu.teja@idiap.ch
François Fleuret[1,2,3]
francois.fleuret@unige.ch

[1] Idiap Research Institute, Switzerland.
[2] EPFL, Switzerland.
[3] University of Geneva, Switzerland.

Abstract

We study the problem of improving the efficiency of segmentation transformers by using disparate amounts of computation for different parts of the image. Our method, PAUMER, accomplishes this by pausing computation for patches that are deemed to not need any more computation before the final decoder. We use the entropy of predictions computed from intermediate activations as the pausing criterion, and find this aligns well with semantics of the image. Our method has a unique advantage that a single network trained with the proposed strategy can be effortlessly adapted at inference to various runtime requirements by modulating its pausing parameters. On two standard segmentation datasets, Cityscapes and ADE20K, we show that our method operates with about a 50% higher throughput with an mIoU drop of about 0.65% and 4.6% respectively.

1 Introduction

Vision transformers [6, 21] (ViT) have recently demonstrated very strong performance on large scale image classification tasks. These networks break the images into a collection of patches (or tokens, interchangeably) and progressively refine their representation by processing them through a series of residual self-attention layers [26]. While their genesis was for image classification, recent methods have adapted transformer architectures to various computer vision tasks [1, 12, 28], and specifically to semantic segmentation [22, 29, 34].

While these large transformer architectures have lead the progress on the accuracy front, there have been several efforts to make them more efficient to be able to process more data, and faster [23]. One way to achieve this is to reduce the number of processed patches. Some works use multiscale approaches [28, 29] that gradually reduce the number of patches as the processing progresses. Another option has been to drop the patches that are not informative to the classification task [15, 17, 18, 20, 32]. For example, it is possible to classify an image as that of a dog with only the patches that belong to the dog, while refraining from processing the rest of the patches.

* Equal Contribution.

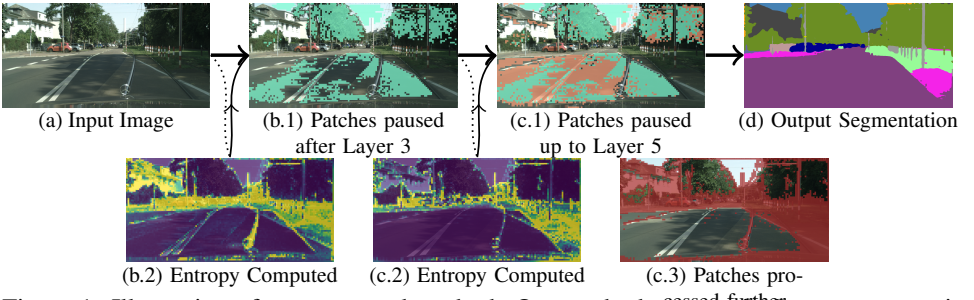


Figure 1: Illustration of our proposed method. Our method progressively stops processing patches after they reach a low enough prediction entropy. First column (a) shows the input image. Second column (b.1) shows the patches that are stopped from being processed after the third transformer layer in green. Third column shows additional patches that are paused after the fifth layer in pink. In the bottom row, (b.2) and (c.2), we show the entropy computed from the auxiliary decoders that is used to decide which patches to pause (Section 2.1). It is apparent that the network automatically pauses easy parts of the image while allocating more computation to the parts that correspond to boundaries, and to smaller and rarer classes, as shown in (c.3) in red. Figure best viewed on a reader with zooming capability. Full details are presented in Section 2.

In this work, we are interested in patch dropping in the context of semantic segmentation. Differing from the case of image classification, it is not possible to drop patches in semantic segmentation, as we have to predict the labels for all the pixels. Instead, we redefine the problem in the context of semantic segmentation to *patch pausing*: pausing a patch at a certain layer signifies that its representation is not going to be updated by any subsequent encoder layer, it does not contribute to the feature computation of other patches, and it is fed directly to the decoder. Consider segmenting natural road scenes from Cityscapes [4] in Figure 1, it is apparent that some parts of the scene are relatively simpler to segment (say, the sky, and the road). So, we allocate lesser computation power to these patches by pausing their feature computation, and feed them as-is to the decoder to produce the final segmentation map. Our argument is supported by the findings in Raghu et al. [19]; they find that the representations of tokens is primarily modified in the first half of the network, and relies on the residual connections to only marginally refine them in the later stages. This opens up an opportunity to reuse the representations, instead of recomputing them, and thus improving the efficiency of segmentation transformers.

Our criterion for token pausing is the time-tested posterior entropy of the segmentation labels. We find that entropy is strongly indicative of lower error. Our method, called Patch pAUsing segmentation transforMER (PAUMER), adds a simple linear auxiliary decoder to predict labels and compute entropy, and processes only the patches whose class prediction is of high entropy, *i.e*. the network is not confident about predicting the labels of these patches, and processes them more. Based on the Segmenter [22] architecture, we show the performance of our method on the standard benchmark suite of ADE20K [35] and Cityscapes [35]. Our method pushes the pareto front of the speed-accuracy trade-offs, and we find that we can operate at a 50% higher throughput with a drop in mean intersection of union of 4.6% and 0.65% on ADE20K and Cityscapes respectively, and for doubling the throughput, the drop is about 10.7% and 4.5% respectively.

2 Patch pausing transformer for Semantic Segmentation

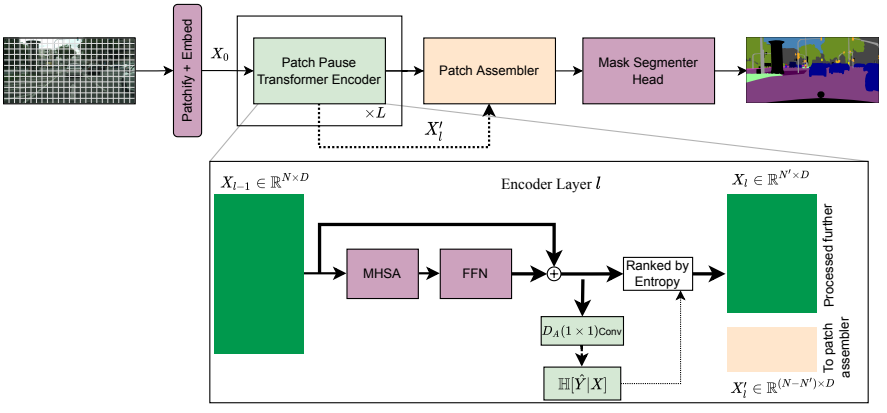


Figure 2: Schematic of our proposed method. We modify Segmenter [22] to enable pausing of patches, and feeding them directly to the decoder. Our proposed PAUMER encoder adds a simple auxiliary decoder (a 1×1 convolution), and uses the predicted posterior entropy $\mathbb{H}[\hat{Y}|X]$ of each component of X to reorder the feature representation X. A portion (τ) of this feature representation would be paused and fed to the decoder directly. The rest of the features (of size $N' < N$) are processed further.

Patch pausing for semantic segmentation is tied to the notion that computation should be non-uniformly distributed across the image, with some parts of the input needing more computation than others to obtain an accurate segmentation. This notion is difficult to realize in the case of convolutional networks as convolution implementations in popular deep learning frameworks handle only inputs with uniform coordinate grid, and thus need software optimizations [13], or require architectural simplifications like the use of 1×1 convolutions in the network [27]. On the other hand, ViTs are ideally suited for this purpose, as each transformer layer consumes a matrix of patch representations without any regard to its inputs spatial location. Removal of patches from computation does not require any additional modifications to the transformer networks for them to apply heterogeneously distributed computation across an image. This restricts the pausing pattern to operate at a patch level, and these paused patches can be non-uniformly distributed over the image coordinate grid. Our primary experiments are based on the architecture Segmenter [22], which uses a ViT backbone to extract patch representations, and predicts a segmentation map using a transformer-based mask decoder. We describe this in detail in Appendix C

2.1 Using Entropy as a criterion for patch-pausing

How do we determine which tokens to pause, *i.e.* which one do not need more processing? Consider the unrealistic case when we have access to the ground truth labels. We could decode after each layer $l \in \{1 \dots L\}$, and stop the processing of tokens for which the prediction is accurate enough.

In the absence of ground truth to determine which tokens can be paused, we propose to use the entropy of label predictions as a proxy for the correctness of the network's predictions. We posit that our models, when confident about their prediction, are likely to be correct. To sanity-check the aptness of entropy as a pausing criterion, we plot in Figure 3 the

entropy of predictions computed after every second layer in a segmenter. Specifically, we use a Segmenter with ViT-Ti backbone pretrained on Cityscapes, freeze its weights, and only train the one-layer linear auxiliary predictor added after each layer. Each vertical plot is a histogram, and we see that in the initial layers, there is a higher overlap of correct and incorrect predictions' entropies. When a token has been processed enough to predict the correct label, the entropy of the prediction is generally low. Thus, pausing patches based on entropy results in representations that have been refined enough to result in correct predictions.

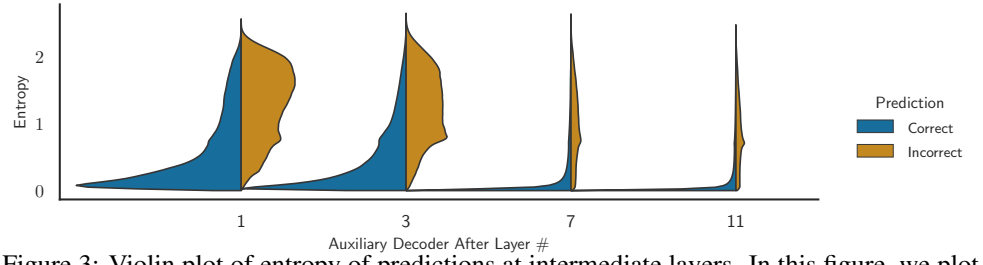


Figure 3: Violin plot of entropy of predictions at intermediate layers. In this figure, we plot the entropy distribution of the auxiliary predictions for 10% of images in Cityscapes validation set. The x-axis marks after which layer the prediction was done. For each layer, the entropy distribution is shown for tokens correctly (in blue) and incorrectly (in orange) classified. We see that the entropies of the predictions for tokens correctly classified accumulate in the low values in the later layers (blue spike on the bottom-right)

2.2 Training PAUMER- One training for many pause configurations

We base our network on Segmenter's architecture [22] detailed in Appendix C. A pause configuration (or configuration) refers to the proportion of patches paused at each layer of the network. An obvious method is to train and test the same patch pausing configuration that satisfies our run-time requirements. Any changes to the run-time requirements requires retraining the network. For this reason, we propose a more general strategy that enables multiple patch pausing configurations at inference with just one trained model. For each transformer layer l, we define a range of patch pausing proportions $(\tau_l^{\text{lo}}, \tau_l^{\text{hi}})$. For each batch of training samples, we sample uniformly one layer $l \in \{3, \dots, L\}$ and a patch pausing proportion $\tau_l \sim \mathcal{U}[\tau_l^{\text{lo}}, \tau_l^{\text{hi}}]$, where $\mathcal{U}$ refers to a uniform distribution over the parameters. To facilitate the patch pausing, we employ a single auxiliary decoder D_A, parametrized by a 1×1 convolution, after the operations of layer l (see Figure 2).

The outputs of the main branch of the network and the auxiliary branch are trained using the traditional cross entropy loss.

$$\mathcal{L}_{\text{main}} = \text{CE}(\boldsymbol{y}, \hat{\boldsymbol{y}}) \tag{1}$$

$$\mathcal{L}^l_{\text{aux}} = \text{CE}(\boldsymbol{y}, \hat{\boldsymbol{y}}^{\boldsymbol{l}}) \tag{2}$$

where $\boldsymbol{y}$ is the ground truth, $\hat{\boldsymbol{y}}$ refers to the logits predicted by the main decoder (mask transformer), and $\hat{\boldsymbol{y}}^{\boldsymbol{l}}$ is the auxiliary decoder's output at the l^{th} layer. The total loss used to train is a combination of losses in Equations (1) and (2).

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{main}} + \lambda \mathcal{L}^i_{\text{aux}} \tag{3}$$

where λ is a scalar used to scale the contribution of additional losses.

At layer l, entropy is computed for each component of X_l as

$$H_l := \mathbb{H}[\hat{Y}_l|X_l] = \mathbb{H}[\sigma(D_A(X_l))], \tag{4}$$

where σ is the softmax function applied to each pixel independently. With this entropy, we pause the computation of a proportion τ_l of tokens with the lowest entropy and store them as X'_l, and continue with the computation using the rest of the tokens X_l (see Figure 1). Note that there are other ways to use H_l to pause tokens, for example by using a threshold on entropy. However, doing so results in pausing and removing different amounts of tokens in each image of a batch, which would add a substantial overhead as padding is not processed efficiently on a GPU. Additionally, pausing a fixed amount of tokens allows for a deterministic computation time.

In order to re-assemble in the original order the patches that have been fully processed and the ones that have been paused, we use a patch assembler module (see Appendix A for PyTorch-like code). It takes X_L and X'_l, and reassembles them into the original shape of X_0. In order to do so, pausing mechanism stores the indices of the patches that have been chosen to be paused, in addition to the current feature representation. The assembler copies the paused representation into the same indices stored previously. This re-assembled output is finally fed into the decoder (mask segmenter head) to compute the segmentation map.

Inference Our training procedure of randomized pause configurations gives us the advantage to choose a pausing configuration that is informed by the run-time requirements *i.e.* mIoU and number of images processed per second. This configuration can have multiple pause locations, each with different pause proportions. We show some results for some configurations listed in Table 1. The patch assembler accordingly assembles multiple paused patches $\{X'_l\}$, and the final representation X_L. The specific configurations are chosen to display the adaptability of the trained network to various inference time requirements, and do not hold any specific importance. Pause configurations can be added easily, as it does not influence the training, but only requires testing over the validation set.

3 Related Work

We now discuss some important prior work related to our method, before showing the experimental evidence.

Segmentation using Transformers Transformers that were originally proposed for language processing tasks [26] have been incorporated into vision [5, 25] and several improvements have been proposed [12]. SETR [34] adapted the standard vision transformer (ViT) to segmentation by using multiscale decoder on all the image patches. Segmenter [22] improved the decoder design by using a learnable per-class token that acts as weighting mechanism over the tokens' representations. Segformer [29] redesigned the architecture with a multiscale backbone that does not use positional encoding, and an MLP based decoder. Several improvements to the transformer backbone have been shown to have impact on the down-stream segmentation performance [1, 16, 28, 31]. These improvements to the transformer backbones have indeed improved the efficiency, measured by frames processed per second, number of floating point operations per second (FLOPs), or images processed per second.

Token sparsification methods Several components of the whole transformer architecture have been improved, by approximations, and simplifications to attention mechanism. Interested readers can refer to Tay et al. [23]. Orthogonal to the architectural improvements,

recent work has focused on the reduction of the data processed, and our proposed method is a form of input dependent reduction. Graves [8] proposed Adaptive Computation Time (ACT), where the amount of processing for each input to an RNN is decided by the network by determining a halting distribution. It was adapted to residual networks by Figurnov et al. [7], that dynamically decides to apply differential number of residual units to different parts of the input. This has been adapted to transformers too [32], where the tokens are progressively halted as they are determined have been processed enough according to a similar criterion as ACT. DynConv [27] uses an auxiliary network to predict pixel masks using which indicate pixels of the image that are not processed by a residual block. DynamicViT [20] extends this formulation to transformers where they, similarly, use an auxiliary network to predict which patches are dropped from being refined further. The auxiliary branches are trained using the Gumbel-softmax trick [10] in both these methods. We consider simplicity of the steps the strength of our proposed method. Unlike DynamicViT [20], we do not need techniques like Gumbel-softmax that are harder to optimize, and additional tailored losses. Additionally, both A-ViT and DynamicViT drop a fixed amount of patches for a given image, and do not provide the flexibility to vary the number of patches dropped, as our method does.

Early-exit methods Dynamic neural networks [9] adapt the architectures or parameters in an input adaptive fashion. Specifically, early-exit methods find that deep neural networks can overthink where a network can correctly predict before all layers process the input, and it can even result in wrong predictions due to over-processing [11]. Several methods to determine when to exit the network have been proposed. Branchynet [24] and Shallow-Deep nets [11] uses auxiliary classifiers to predict the output class for vision convolutional networks, and stops processing a sample if the entropy of a branch's predictions is lower than a predefined threshold. This idea was further exploited in NLP literature. Zhou et al. [36] extends this by using a patience parameter that tracks number of auxiliary classifiers which predict the same class. DeeBERT [30] proposes a two stage training, where the auxiliary decoders are trained after the main networks is trained, and frozen. Li et al. [14] propose a layer cascade for convolutional segmentation networks, that processes easy to hard parts progressively through the network. Their method needs modifications of the network architecture, whereas we show that our proposed method can be added with very little efforts. Our method is an early exit strategy, specifically for the case of segmentation transformers. While similar methods have been examined in literature, to the best of our knowledge, we are the first to examine the patch pausing problem of semantic segmentation. Also, the randomized training presented in Section 2.2 has not been used in this context, though similar ideas to reduce network width were studied in slimmable networks [33].

4 Experiments

4.1 Datasets and Evaluation

We show the performance of our method using networks trained on Cityscapes [4] and ADE20K [35]. Cityscapes (CS) consists of $2,975$ images in the training set, in which each pixel belongs to one of 19 classes, and 500 images in the validation set which are used to benchmark the performance of our method. ADE20K is substantially larger, with a training set of $25,574$ with 150 classes, and $2,000$ images to validate the performance. The results for Cityscapes and ADE20K are presented below.

Our primary performance measure is based on the speed-accuracy trade-off, measured

by mean Intersection over Union (mIoU) metric and throughput in images per second. To determine the number of images processed per second (IMPS), following Strudel et al. [22], we use images of size 512×512 with a batch size that optimally occupies a V100 GPU.

Methods compared To assess the performance of our proposed method, we use the following baselines for comparison:

a. **Baseline set by Segmenter**, without any patch pausing.
b. **Random Pausing (RP)**: We train the network to handle pausing a proportion τ of randomly chosen patches, instead of the lowest entropy ones.

We examined an additional simple baseline of random pausing (without training), and found the results not competitive enough to warrant reporting here. Also, some methods in Section 3 are capable of dropping different patches per image. These methods can result in a decrease in FLOPs (floating point operations), but this reduction cannot be realized in wall clock improvements as modern GPUs parallelize computation over batch elements.

Pause Layer	Pause configurations													
3	0.2	0.4	0.6					0.2	0.3	0.4	0.2	0.3	0.4
5				0.2	0.4	0.6	0.8	0.2	0.3	0.4	0.2	0.3	0.4
7											0.2	0.3	0.4

Table 1: Table of configurations. Each column represents a pause configuration, *e.g.* the first column represents the configuration pausing 20% of tokens after layer 3, using the notation introduced in Section 2.2. Each configuration here corresponds to a marker in Figures 4a, 4b and 13.

Training hyperparameters and Inference Configurations: For our main experiments, we use Segmenter with ViT-Ti and ViT-S backbones (details in Appendix B). During training, we follow the procedure in Section 2.2 for every network and dataset, and in particular we pause a random amount of tokens $\tau_l \sim \mathcal{U}[0.2, 0.8]$ at a random layer $l \in \{3,4,5,6,7,8,9\}$. We initialize the model for our training with pretrained segmenter weights, as we find this results in better performance, and train the model for $80,000$ steps for Cityscapes and $160,000$ steps for ADE20K. The auxiliary loss-weight λ is set to 0.1. Rest of the hyperparameters as kept the same as in Strudel et al. [22]. We implement our method using mmsegmentation [3], and use their pretrained models whenever available. At inference, we test the networks with the pause configurations in Table 1. This list of pause configuration is not exhaustive, and does not hold any specific importance, but have been chosen to show the efficiency of our method in trading off mIoU for higher IMPS.

Choosing pause configurations: Determining the appropriateness of a pausing configuration incurs little additional cost, as it only requires inference with a validation set for each configuration of interest (see Figures 4a and 4b). On a new dataset for which we train the network with the proposed training procedure, we foresee two scenarios:

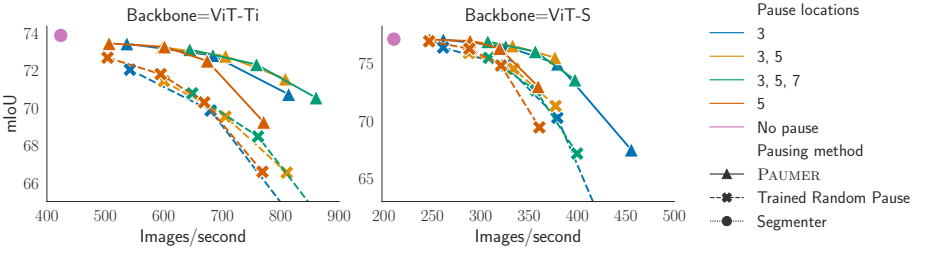
1. If the objective is to attain a specific throughput, we can easily find configurations that match the requirement with a sweep over them (Figures 9 and 10) by only timing them, and then evaluating the mIoU of the ones that meet the time requirement.

2. If the objective is to find a good throughput-mIoU trade-off: First, we sweep through configurations that pause at only one layer (Figure 9) and we pick the first layer and

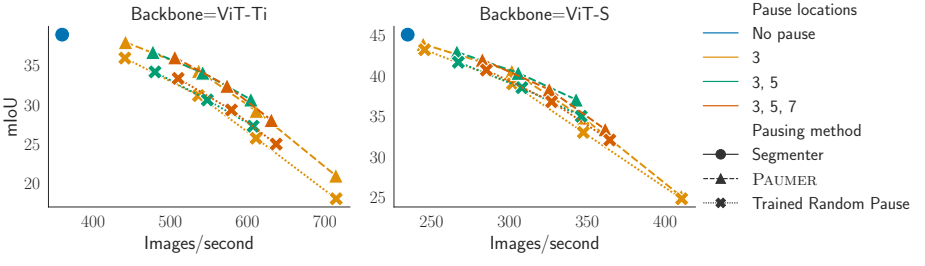
proportion. We fix this layer and ratio, then sweep through pausing configurations at a second layer (Figure 10). We repeat this procedure until adding more layers is no longer beneficial.

4.2 Results

Performance analysis In Figure 4a, we plot the mIoU versus the number of images per second achieved by baselines and our entropy patch pausing for different configurations, for Cityscapes. Each point is the average value of three training runs. The left-most point corresponds to the original Segmenter model, in solid line our proposed pausing strategy, and in dashed line random pausing with training. For both ViT-Ti and ViT-S backbones, a 50% increase of IMPS can be achieved with an mIoU drop of about 0.7% and 0.6% respectively. Further, for doubling the IMPS, we see that the mIoU drops about 3.2% and 5.9% respectively. For the trained random pausing using ViT-Ti, a strong baseline, the equivalent drops in mIoU are about 2.9% and 8.8% to increase the IMPS by 50% and 100%, respectively. We show results for ViT-Ti and ViT-S for ADE20K in Figure 4b. For ViT-Ti backbone, we



(a) mIoU vs Images processed per second for ViT-Ti and ViT-S backbones on Cityscapes val set.



(b) mIoU vs Images per second for ViT-Ti and ViT-S backbones on ADE20K val set.

Figure 4: Results on Cityscapes and ADE20K for our proposed method PAUMER. Each marker is a configuration from Table 1. We train a single model capable of handling various pause configurations that can be chosen based on run-time requirements. It is apparent that ADE20K suffers from a higher drop in performance when patch pausing is employed. However, PAUMER consistently outperforms the random training baseline.

see that for a 50% increase in IMPS, mIoU drops by about 2.8%, and a 100% increase in IMPS with a mIoU drop of about 10.7%, compared to random pausing performance of 5.4% and 13.5%. The drop in mIoU is in contrast with the results of Cityscapes, where we could achieve similar increase in throughput with a much lesser drop in performance. We chalk this difference up to dataset characteristics; ADE20K has almost an order of magnitude more

classes, and the images are smaller with cluttered scenes and numerous small objects, which may require more processing to be correctly classified.

Generating these performance plots *i.e.* mIoU vs IMPS is inexpensive, as no retraining is involved and only needs inference on a validation set with reasonably chosen pause configurations.

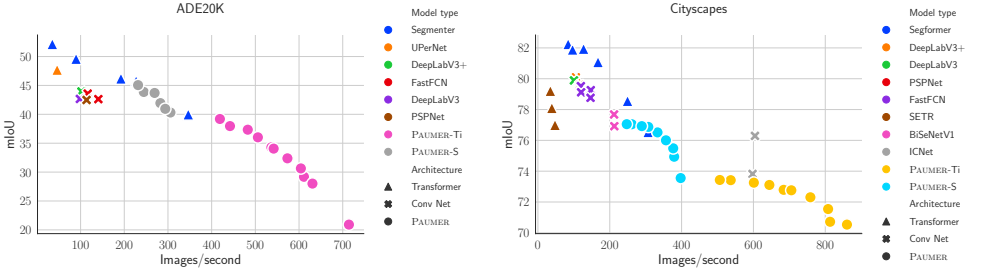


Figure 5: Performance comparison for ADE20K and Cityscapes. We compare to pretrained models available in mmsegmentation [3] for each of these datasets. Architectures devised for speed or accuracy outperform us on those criteria, but PAUMER has the unique advantage that we can trade off one for the other using a tunable hyperparameter. Here, we show the pareto front of PAUMERẆe use different colors for different architectures, and different shapes for architectural families.

Comparison to other architectures In Figure 5, we compare our method to a broad array of architectures for which pretrained models are available in mmsegmentation [3]. Here we plot only the best performances obtained for each throughput across pause configurations. We estimate this by computing the skyline queries. We include both convolution based architectures, and transformer based ones. In the transformer family, available networks have focussed on improving the performance, and thus are slower, but more accurate than the PAUMER family of models. CNN based ones (say ICNet for Cityscapes) that have been designed to be more efficient are competitive or better in speed than our models. However, we have the unique ability to tune the mIoU-throughput scores of our model without having to retrain them.

5 Discussion

The improvement in images processed per second is obtained by reducing the number of patches processed. This might not necessarily hold true in the case of networks with convolutional layers interspersed, such as SegFormer [29] that uses convolution instead of positional encoding, as convolution on unstructured sparse inputs is not highly optimized in CUDA implementations. Thus, our method is not readily applicable to all transformer models.

In addition to architecture dependence, patch pausing's performance maybe dependent on the dataset itself. We attributed the difference between Cityscapes and ADE20K to inherent dataset complexities by examining the performance of the auxiliary classifier; for ViT-Ti, it reaches around 60% accuracy for ADE20K and 90% accuracy for Cityscapes. Examining the possible relationship between patch pausing performance and the dataset difficulty [6] might shed some light on this issue.

Additionally, patch pausing assumes that a paused token is not important to the feature computation of other tokens, as it will not contribute further to the attention computation to refine the representation of other patches. Performance (mIoU) indicates that it might have little bearing, but this assumption needs to be investigated further.

Our method, while being input adaptive in choosing the patches to pause, chooses a fixed proportion of them. This design is to exploit the batch level parallelism on GPUs. Choosing the number of patches depending on the input batch has not been dealt with in this paper.

6 Conclusion

Our method, PAUMER, is a first step in the direction of post-hoc design for efficient inference in semantic segmentation transformers. We do so by applying dissimilar amounts of computation to various patches of an input image. Patches with high predicted auxiliary entropy are processed further, whereas the rest of them are fed directly to the decoder skipping all the intermediate computation. To run at a specified throughput (images per second), our method offers the flexibility to choose an appropriate pause configuration, without having to retrain the network.

Acknowledgements Evann Courdier and Prabhu Teja are supported by the "Swiss Center for Drones and Robotics - SCDR" of the Swiss Department of Defence, Civil Protection and Sport via armasuisse S+T under project No 050-38.

References

[1] Xiangxiang Chu, Zhi Tian, Yuqing Wang, Bo Zhang, Haibing Ren, Xiaolin Wei, Huaxia Xia, and Chunhua Shen. Twins: Revisiting the design of spatial attention in vision transformers. *Advances in Neural Information Processing Systems*, 34, 2021.

[2] Xiangxiang Chu, Zhi Tian, Yuqing Wang, Bo Zhang, Haibing Ren, Xiaolin Wei, Huaxia Xia, and Chunhua Shen. Twins: Revisiting the design of spatial attention in vision transformers. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 9355–9366. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/4e0928de075538c593fbdabb0c5ef2c3-Paper.pdf.

[3] MMSegmentation Contributors. MMSegmentation: Openmmlab semantic segmentation toolbox and benchmark. https://github.com/open-mmlab/mmsegmentation, 2020.

[4] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

[5] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Syl-

vain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*, 2021.

[6] Kawin Ethayarajh, Yejin Choi, and Swabha Swayamdipta. Understanding dataset difficulty with $\mathcal{V}$-usable information. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 5988–6008. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/ethayarajh22a.html.

[7] Michael Figurnov, Maxwell D Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry Vetrov, and Ruslan Salakhutdinov. Spatially adaptive computation time for residual networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1039–1048, 2017.

[8] Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

[9] Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang. Dynamic neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2021. doi: 10.1109/TPAMI.2021.3117837.

[10] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[11] Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. Shallow-deep networks: Understanding and mitigating network overthinking. In *International conference on machine learning*, pages 3301–3310. PMLR, 2019.

[12] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in vision: A survey. *ACM Comput. Surv.*, dec 2021. ISSN 0360-0300. doi: 10.1145/3505244. URL https://doi.org/10.1145/3505244.

[13] Andrew Lavin and Scott Gray. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4013–4021, 2016.

[14] Xiaoxiao Li, Ziwei Liu, Ping Luo, Chen Change Loy, and Xiaoou Tang. Not all pixels are equal: Difficulty-aware semantic segmentation via deep layer cascade. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3193–3202, 2017.

[15] Youwei Liang, Chongjian GE, Zhan Tong, Yibing Song, Jue Wang, and Pengtao Xie. EVit: Expediting vision transformers via token reorganizations. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=BjyvwnXXVn_.

[16] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[17] Dmitrii Marin, Jen-Hao Rick Chang, Anurag Ranjan, Anish Prabhu, Mohammad Rastegari, and Oncel Tuzel. Token pooling in vision transformers, 2022. URL https://openreview.net/forum?id=EGtUVDm991w.

[18] Bowen Pan, Yi fan Jiang, Rameswar Panda, Zhangyang Wang, Rogério Schmidt Feris, and Aude Oliva. Ia-red2: Interpretability-aware redundancy reduction for vision transformers. In *NeurIPS*, 2021.

[19] Maithra Raghu, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. Do vision transformers see like convolutional neural networks? In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=Gl8FHfMVTZu.

[20] Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. Dynamicvit: Efficient vision transformers with dynamic token sparsification. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[21] Andreas Steiner, Alexander Kolesnikov, , Xiaohua Zhai, Ross Wightman, Jakob Uszkoreit, and Lucas Beyer. How to train your vit? data, augmentation, and regularization in vision transformers. *arXiv preprint arXiv:2106.10270*, 2021.

[22] Robin Strudel, Ricardo Garcia, Ivan Laptev, and Cordelia Schmid. Segmenter: Transformer for semantic segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7262–7272, 2021.

[23] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *arXiv preprint arXiv:2009.06732*, 2020.

[24] Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. Branchynet: Fast inference via early exiting from deep neural networks. *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469, 2016.

[25] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herve Jegou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, volume 139, pages 10347–10357, July 2021.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[27] Thomas Verelst and Tinne Tuytelaars. Dynamic convolutions: Exploiting spatial sparsity for faster inference. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition*, pages 2320–2329, 2020.

[28] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 568–578, 2021.

[29] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. *Advances in Neural Information Processing Systems*, 34, 2021.

[30] Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. DeeBERT: Dynamic early exiting for accelerating BERT inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2246–2251, Online, July 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.acl-main.204.

[31] Weijian Xu, Yifan Xu, Tyler Chang, and Zhuowen Tu. Co-scale conv-attentional image transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9981–9990, 2021.

[32] Hongxu Yin, Arash Vahdat, Jose Alvarez, Arun Mallya, Jan Kautz, and Pavlo Molchanov. A-ViT: Adaptive tokens for efficient vision transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

[33] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=H1gMCsAqY7.

[34] Sixiao Zheng, Jiachen Lu, Hengshuang Zhao, Xiatian Zhu, Zekun Luo, Yabiao Wang, Yanwei Fu, Jianfeng Feng, Tao Xiang, Philip HS Torr, et al. Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6881–6890, 2021.

[35] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 633–641, 2017.

[36] Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. Bert loses patience: Fast and robust inference with early exit. *Advances in Neural Information Processing Systems*, 33:18330–18341, 2020.

Supplementary Sections

Appendix A Pseudocode for Patch Pauser and Assembler

In Algorithm 1, we present pseudo-code for both patch pausing mechanism and patch assembler. The code isn't meant to be functional but only for illustrative purposes. Comments describe the functionality.

Algorithm 1 Patch Pauser and Assembler Pseudocode

```
def patch_pauser(tokens, pause_ratio, keep_indices, paused_tokens):

    # tokens (X_l) refers to current set of tokens being processed.
    # Note that this might not be the total number of tokens, as one
    # or more patch pausing stages could have happened.

    # pause ratio is τ, pausing proportion

    # keep_indices, paused_tokens are temporary arrays to store
    # details for assembling (see below).

    _, total_tokens, _ = tokens.shape
    to_process_count = N - int(pause_ratio * N)

    # Compute aux entropy of tokens
    aux_prediction = auxiliary_classifier(tokens)
    probs = aux_prediction.softmax(dim=-1)
    entropy = compute_entropy(probs)

    ## Instead of sorting, we use topk. This is faster on GPU.
    topk_inds = entropy.topk(to_process_count)
    kept_tokens = tokens[:, topk_inds]

    keep_indices.append(topk_inds)
    paused_tokens.append(tokens) ## This is X'_l

    return kept_tokens ## This is X_{l+1}

def patch_assembler(X_L, paused_tokens, keep_indices):
    # X_L (X_L) refers to the final feature representation at the end of the
    # encoder. One or more stages of pausing have occurred before this.
    # X_L is of the shape BxN'xD.

    # paused_tokens: the feature representations of the tokens prior to
    # removing the paused ones.

    # keep_indices: The indices of the argsort of the auxiliary decoders'
    # entropy.
    for indices, tokens in zip(keep_indices[::-1], paused_tokens[::-1]):
        tokens[:, indices] = X_L
        X_L = tokens

    return X_L
```

Appendix B Backbone details

In this paper, we use two transformer backbones: ViT-Ti(ny) and ViT-S(mall). We do not experiment with ViT-B, ViT-L architectures, due to our computational resource constraints. In Table 2, we describe the main architecture details of the ViT backbones.

Model Name	Layers	Embedding dim	Heads	Params
ViT-Ti	12	192	3	5.9M
ViT-S	12	384	6	22.5M

Table 2: ViT architectural details used

Appendix C Brief introduction to Segmenter

Segmenter [27] network ingests an input image $I \in \mathbb{R}^{W\times H\times 3}$ and assigns one of the K output classes to each input pixel. From I, the model first extracts non-overlapping patches of size P, creating a total of $N = \frac{WH}{P^2}$ patches (also called tokens). Each of those patches are then transformed using a linear embedding layer $E : \mathbb{R}^{3P^2} \to \mathbb{R}^D$, giving a feature representation $X_0 \in \mathbb{R}^{N\times D}$, as shown in Figure 2.

This feature representation is refined by processing through L transformer encoder layers T_l ($l \in [L]$), where each transformer encoder layer consists of a multi-head self-attention (MHSA) block followed by a two layers perceptron (FFN). The overall operation can be represented as:

$$X_L := T_L \circ T_{L-1} \circ \cdots \circ T_1 \circ E(I) \in \mathbb{R}^{N\times D}$$

Each T_i is residual in nature, *i.e.* $T_i(x) = x + A(x)$ where A encompasses the self-attention and the multi layer perceptron.

After L layers of such processing, the refined features X_L are fed into a decoder M_D. The paper investigated two kinds of decoders: (a) a linear decoder that takes in the features in $X_L \in \mathbb{R}^{N\times D}$ and produces logits $\in \mathbb{R}^{N\times K}$ using a 1×1 convolution, (b) a mask transformer which learns K class embeddings that are jointly processed with X_L through several transformer encoder layers (à la T_i), and produces a logits $\in \mathbb{R}^{N\times K}$ as a dot product between the features and the learned class embeddings. The output of either decoder is reshaped to $\mathbb{R}^{\frac{W}{P}\times\frac{H}{P}\times K}$, and then bilinearly upsampled to produce a logit map of size $W\times H\times K$. A softmax layer is used to obtain a categorical distribution over the labels for every pixel. All the layers, E, T_i, M_D are trained using the standard cross entropy loss.

Appendix D Patch pausing's limitations

In the main paper, we investigated pausing up to three times in the network. It is indeed tempting to pause at more layers, but this entails two additional costs: first to compute the entropy and rank the patches, and then the patch assembly as detailed in Appendix A. These costs are (ideally) off-set by the reduction in number of patches processed.

To illustrate this further, let us take the case of our experiment with ViT-T backbone (with 12 layers). In this experiment, we are interested in how the pausing patterns affect the

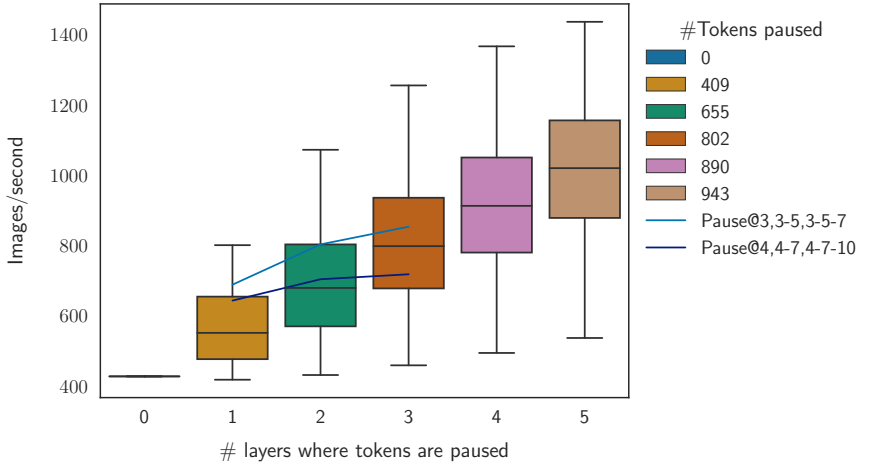


Figure 6: Evolution of the throughput with the number of layers using patch pausing.

throughput computed in images processed per second. To simplify the analysis, we assume that we pause a fixed proportion $\tau = 0.4$.

In Figure 6, we show the distribution of throughput (in images/sec) vs the number of layers we pause tokens at. We use a box-plot where horizontal lines indicates quartiles. For each value k on the x-axis, there are $\binom{12}{k}$ pause configurations.

Consider the case of pausing once. In this case, not all configurations of pausing are useful; pausing too late may in fact be slower than the baseline of not pausing at all, as it incurs an additional cost of auxiliary decoding and patch re-assembling that may offset the time gain of not processing some patches. This trend is visible on Figure 6 and holds even as the number of layers to pause at increases.

We now focus on two cases of pausing: pausing after layers 3, (3, 5), (3, 5, 7) and 4, (4, 7), (4, 7, 10). This two configurations are plotted as lines on Figure 6. We see clearly that pausing more has benefits in number of images processed, but that this benefit can quickly plateau if we pause at later layers of the network. Additionally, this analysis does not consider the mIoU at all. Indeed, while pausing early-on and at many layers is tempting, the drop in mIoU becomes too high for those pause configurations to be useful (see Figures 4a and 4b). Thus the primary limitation is posed by the drop in mIoU rather than throughput.

Appendix E Influence of the training pause ratio $\tau_l - \tau_h$

In Figure 7, we plot the mIoU of different pause configurations as a function of the throughput for different values of the range of the pause ratio $\tau_l - \tau_h$ introduced in Section 2.2. We see that the results for various train pause ranges is relatively stable for low amounts of inference pausing ratios. Segmenter's standard deviation (over 3 runs) is 0.35%, and we see that the absolute difference in the performance at a given IMPS is about 0.5%. This, however, changes when the amount of pausing increases (each colored curve's right corners), when the performance difference is higher ($\approx 1\%$). More aggressive pausing at training seems beneficial ($0 - 0.9$ performs the best). However, as a middle ground to the multiple configurations investigated, we use $0.2 - 0.8$ for all our experiments.

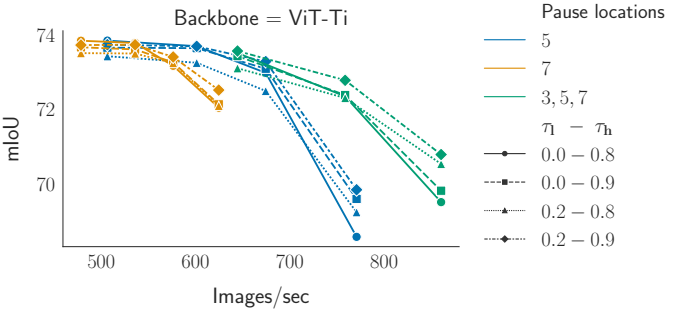


Figure 7: mIoU vs throughput for different values of the range of the pause ratio $\tau_l - \tau_h$ introduced in Section 2.2.

Appendix F Trading off mIoU for higher throughput

We present results for trading off mIoU for throughput. Specifically, we take all our runs (mIoU vs IMPS) data, and fit a linear spline, and use the resultant function to predict the mIoU for 8 intermediate IMPS within the range for which we have experimental results for ViT-Ti in Figure 4a. This is to illustrate that we can choose a pause configuration that fits our run-time requirements (IMPS), and that it works with the performance specified here.

Backbone									
ViT-S 210 im/s	Images / second	252	276	300	325	349	373	397	421
	mIoU of PAUMER	77.04	76.96	76.89	76.41	76.17	74.89	73.60	71.11
	Diff to Segmenter	-0.03	-0.12	-0.19	-0.66	-0.90	-2.18	-3.47	-5.96
	mIoU of RP	76.68	76.08	75.77	74.79	73.32	70.75	67.60	62.64
	Diff to Segmenter	-0.39	-0.99	-1.30	-2.28	-3.75	-6.32	-9.47	-14.44
ViT-Ti 424 im/s	Images / second	508	557	605	654	702	751	799	847
	mIoU of PAUMER	73.42	73.35	73.23	72.91	72.76	72.37	70.99	70.58
	Diff to Segmenter	-0.42	-0.50	-0.61	-0.94	-1.09	-1.48	-2.86	-3.27
	mIoU of RP	72.59	71.96	71.35	70.64	69.56	68.66	65.07	64.98
	Diff to Segmenter	-1.26	-1.89	-2.50	-3.20	-4.28	-5.19	-8.78	-8.87

Table 3: Trading off mIoU for speed. In Section 4.2, we showed the performance of mIoU for 50%, and 100% increase in IMPS. Here we show numbers for a finer grid of IMPS, up to doubling of IMPS.

Appendix G Influence of the auxiliary loss weight λ

In Figure 8, we plot the mIoU of different pause configurations as a function of the throughput for different values of the auxiliary loss weight λ introduced in Section 2.2. We can see that increasing λ pushes the network to be more robust to token-pausing but leads to lower performance when pausing fewer tokens. Thus, λ can be tuned depending on the use-case to favor either pausing lesser or a larger number of tokens.

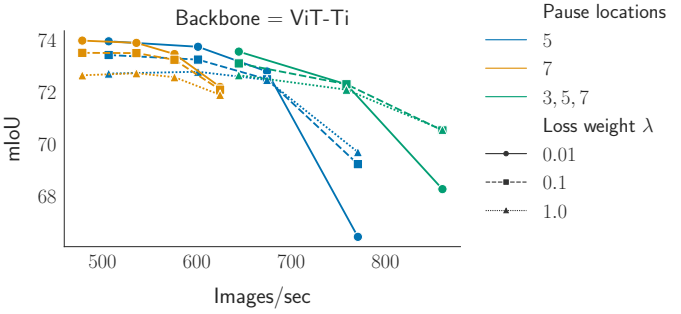


Figure 8: mIoU vs throughput for different values of the auxiliary loss weight λ introduced in Section 2.2. Lower values of λ lead to better performance when pausing few tokens but worse performance when pausing more, and conversely for higher values of λ. Our chosen value of $\lambda = 0.1$ is a trade-off that can also be modified depending on the use-case.

Appendix H Interplay of Pause location and Pausing proportion τ

We showed the results for some configurations in Table 1. In this section, we study the interplay between pause location and pause proportion τ. In Figure 9, we show a sweep over pause configrations. For each layer, we choose 20 pause proportions in $(0, 1)$. It is apparent that dropping at a later layer results in lesser drop in mIoU but also does not result in a large gain in IMPS. Thus depending on the desired run-time, one can choose a pause location and pause proportion that gives the required performance.

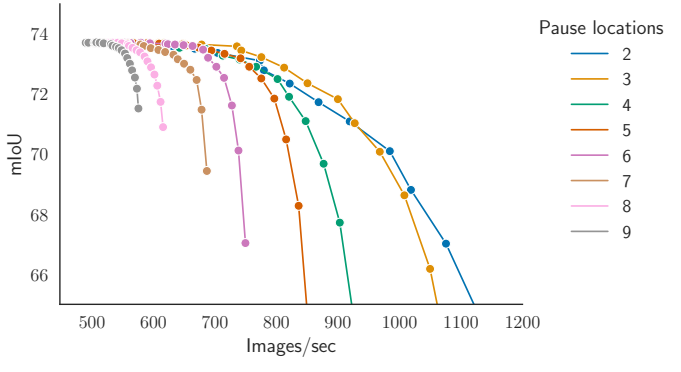


Figure 9: Pausing once at various layers for various pause proportions for ViT-Ti on Cityscapes. We see that the highest gains in IMPS are achieved by dropping in earlier layers.

To examine this further, we plot the performance of pausing twice in Figure 10. Similarly to the case of pausing once, here we sweep over 10 thresholds for each location, thereby generating 100 configurations for a given tuple of layers. For those 100 configurations, we plot the pareto front of performance in Figure 10. We focus on the first pausing layer also, as it has a larger influence on the IMPS gain. We can see that pausing at earlier layers leads to a higher increase in IMPS, that pausing small proportions at these layers leads to slightly

higher drop in mIoU than pausing a higher amount in later layers.

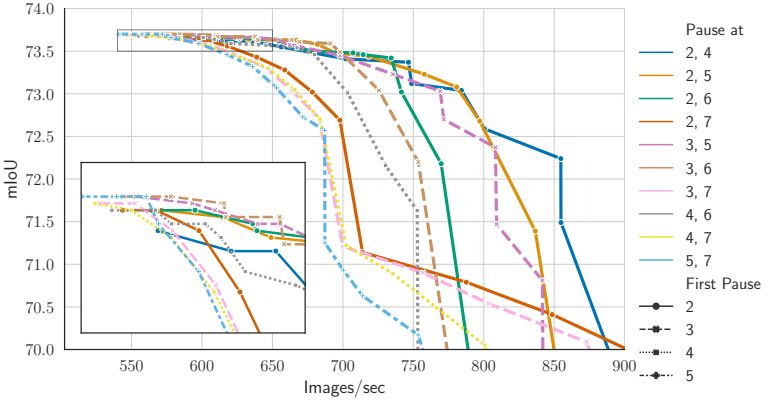


Figure 10: Pareto front of pausing at two layers for ViT-Ti on Cityscapes.

Appendix I Using Early Exit at test time

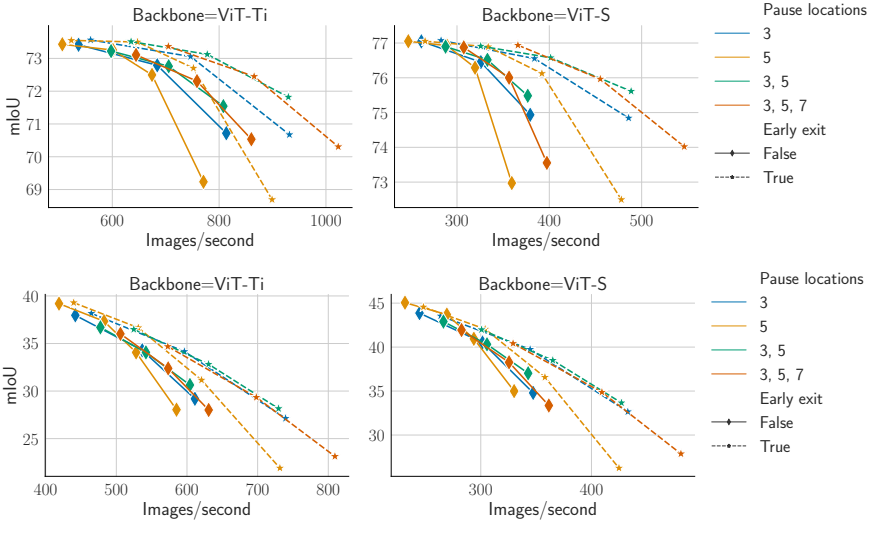


Figure 11: Comparison when segmenting with and without early exit at test time on CityScapes and ADE20K.

In Figure 11, we study the use of early exit on a trained PAUMER. Early exit (see works in Section 3) refers to stopping processing of an input once it is deemed to have been processed enough. In our method, we pause tokens, *i.e.* we stop processing a token by the encoder, and feed it to the decoder to predict the segmentation label. Here, we compare it with directly using the predictions of the auxiliary decoder itself, without stopped tokens being processed by the main decoder. PAUMER can be run with or without early-exit depending on the task, and using early exit on a trained Segmenter is straightforward as it does not need any

retraining due to the use of auxiliary decoders. For the same pausing configurations as in Figure 4a, a network with early exit runs at a higher throughput with less FLOPs by design, but it may run with a lower mIoU. On Figure 11, we see that for Cityscapes, it is beneficial to use PAUMER with early exit. This finding might not hold in general, as a complex mask decoder maybe needed for different datasets.

Appendix J Comparing SETR, Segmenter, EarlyExit using Segmenter

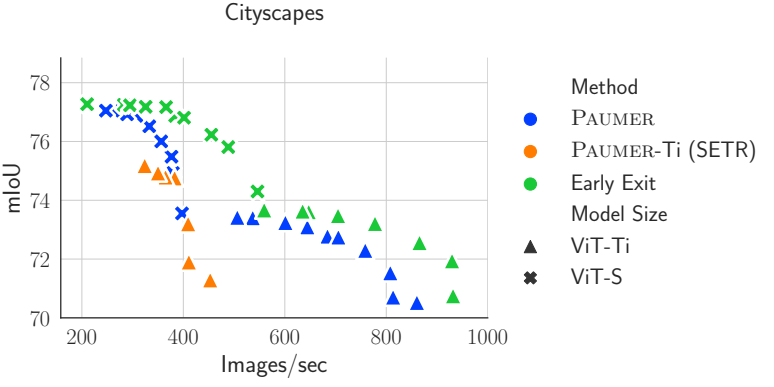


Figure 12: We compare PAUMER on Segmenter, on SETR, and using an early exit variant of PAUMER on Segmenter. Early exit fares better than PAUMER on Cityscapes. This may be attributable to the dataset, as Cityscapes might not need more complex mask decoder for accuracy.

In Appendix I we showed the results of early exit. Here we examine the best performances obtained for each throughput across pause configurations. We estimate this by computing the skyline queries. We see that early exit performs consistently better on Cityscapes.

Additionally, we implement our patch pausing strategy, PAUMER, on the network architecture SETR [54]. SETR's performance drops off more rapidly than Segmenter based patch pausing. Note that we adapt SETR's PUP decoder to use it with a ViT-Tiny backbone. In particular, we reduce the number of channels of the decoder to 192, the number of convolutions in the decoder from 4 to 2 and the upscale factor from 2 to 4.

Appendix K Importance of task specific pretraining

In Figure 13, we study importance of initialization. We compare ViT-Ti pretrained on Cityscapes (task specific), and pretrained on ImageNet (generic) and study their impact on performance. When using a generic pretrained model, our training with PAUMER is increased to 160K iterations instead of 80K. It is apparent that using a task specific pretrained model brings a relatively consistent benefit in this context.

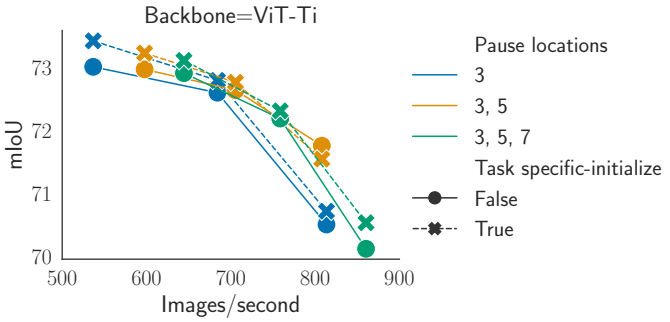


Figure 13: Importance of initialization for PAUMER. Task specific initialization benefits performances. We train ViT-Ti PAUMER: solid lines are from ImageNet pretrained backbones, dashed lines are from Cityscapes pretrained Segmenters. Each marker is a configuration from Table 1.

Appendix L Entropy as a measure of patch-pausing

In Section 2.1, we argued that entropy is a reasonable indicator of completion of processing. For that, we used the illustration in Figure 3 to show the increase in separation of entropy histograms for pixels predicted correctly and incorrectly. We expand that in Figure 14, to analysing that to each class individually. The larger separation in entropy in the first few layers of network in prevalent in large classes like road, building, vegetation, car. As seen in Figure 1 too these larger classes are paused to gain IMPS. Smaller, rarer classes like train, motorcycle, rider are tougher to learn and are unlikely to be paused (as evidenced by their higher entropy).

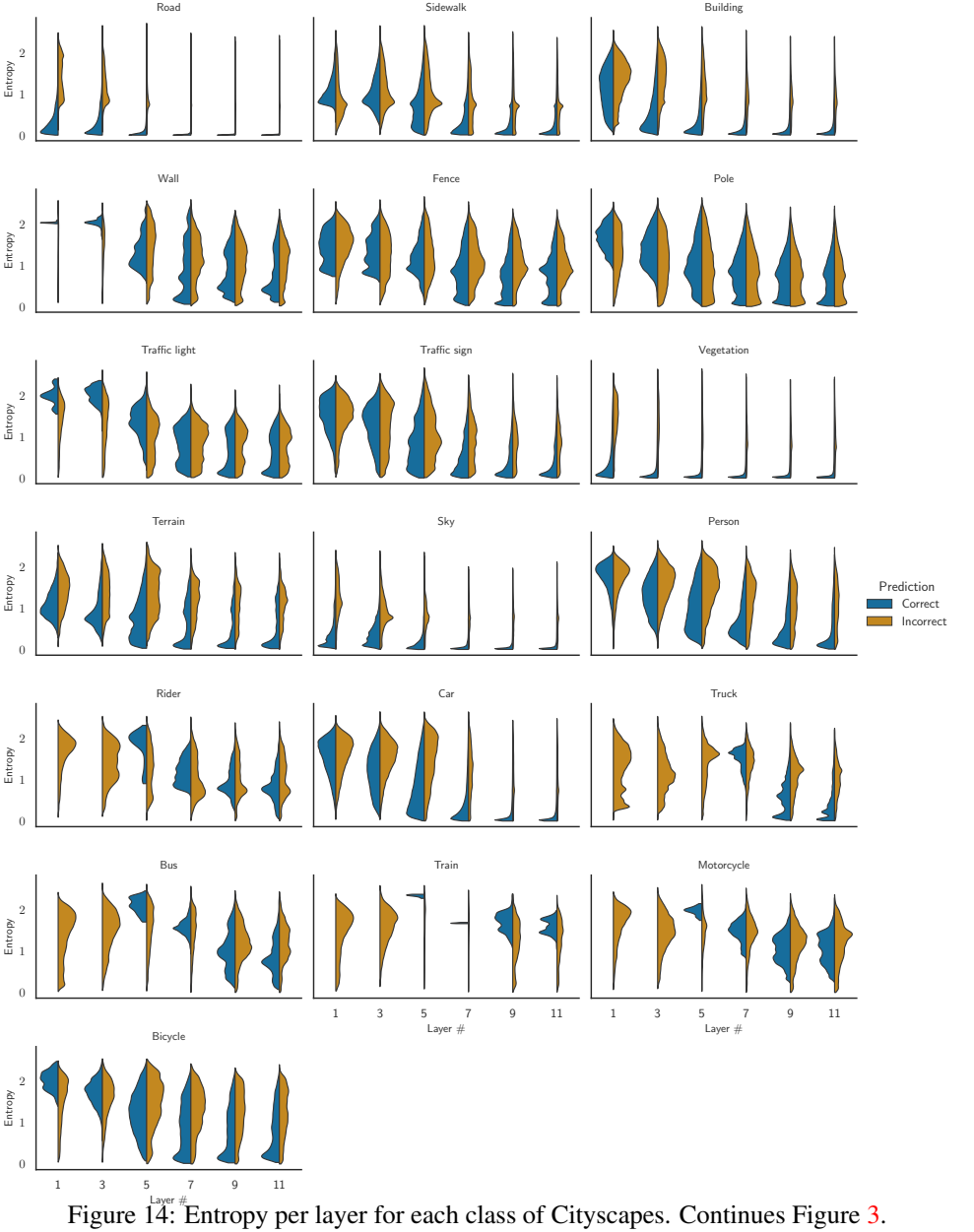


Figure 14: Entropy per layer for each class of Cityscapes. Continues Figure 3.