

Fast and Future: Towards Efficient Forecasting in Video Semantic Segmentation

Présentée le 2 février 2024

Faculté des sciences et techniques de l'ingénieur
Laboratoire de l'IDIAP
Programme doctoral en informatique et communications

pour l'obtention du grade de Docteur ès Sciences

par

Evann Pierre Guy COURDIER

Acceptée sur proposition du jury

Prof. P. Frossard, président du jury
Prof. F. Fleuret, Prof. A. Roshan Zamir, directeurs de thèse
Dr D. Grangier, rapporteur
Dr G. Synnaeve, rapporteur
Prof. M. Jaggi, rapporteur

Acknowledgements

This thesis has been a long journey, filled with joy and hardships, which I shared with several people I would like to acknowledge here.

To begin, I want to thank my advisor, François. I am genuinely grateful for the opportunity I got to interact with him over the past 5 years. I admired his authenticity, which is likely only rivaled by his Twitter addiction. I loved how he would challenge my political views, suggest some great dinner recipes, and share his latest toy project epiphanies all in the same call. Thank you for your time, your trust, and the freedom and support you gave me when I needed it.

I want to thank all the people at Idiap and EPFL who have shared my life in one way or another. To Teja, Suraj, Angelos, Florian, Iulia, Louise, Martin, Matteo, Una, Miranda, Anastasia, Prakhar, Praneeth, Aymeric, Thjis, Jean-Baptiste, Bojana, Paola, Jade, Uday, Vinitra, Ekaterina, Alexandre and many others.

A special thanks to Angelos, to whom I am particularly grateful for giving me the opportunity to intern at Apple and discover the US! He was a great colleague, advisor, friend, and climbing partner. These thanks extend to all my other Apple colleagues in the Cybertron team, notably Tom, Tom and Arthur.

I would like to express my gratitude to my flatmates Benoit, Jonathan and Louise. They were fantastic companions for skiing, hiking, climbing, game nights, and much more. I would also like to extend my thanks to numerous friends, including Annaelle, Gabrielle, Camille, Gwenola, Marine, Kamila, Walentyna, Imene, Gil, and Iryna who supported me or took time out to visit me in Switzerland.

I thank my family, Caroline, Remy, Satchel, and Margaux for their unconditional love, support and faith in me, as well as my grandmas Majo and Marie-Madeleine.

La Réunion, January 19, 2024

Evann Courdier

Abstract

Deep learning has revolutionized the field of computer vision, a success largely attributable to the growing size of models, datasets, and computational power. Simultaneously, a critical pain point arises as several computer vision applications are deployed on low-power embedded devices, necessitating real-time processing capabilities. This challenge intensifies for semantic segmentation, a dense prediction task demanding substantial memory and computational resources. This thesis explores techniques to streamline real-time segmentation networks, enhance their efficiency, and deal with potential ambiguity.

First, we introduce a latency-aware segmentation metric, a measure that combines the mean Intersection over Union with the network processing time, providing a practical metric for applied settings. Emphasis is placed on the concept of "anticipation" in real-time networks - these systems should be capable of predicting future input segmentation. Consequently, we then design an anticipatory convolutional network incorporating an inventive convolution layer. This novel layer reduces computation by reusing features from previous video frame computations, exploiting their temporal coherence. Next, we present a method to accelerate transformer-based segmentation networks called 'patch-pausing'. This technique halts the processing of image patches deemed to be already correctly segmented by assessing the network's confidence in its prediction. Remarkably, our experimental results indicate that more than half of the patches can be paused early in the process, with a minimal impact on segmentation accuracy. This study concludes with the introduction of a discrete diffusion model for segmentation. This model allows for the sampling of multiple potential segmentations for a given input while accurately following the training data distribution. Combining this diffusion model within an autoregressive scheme, we successfully showcase its capacity to generate long-term future predictions of segmentation.

The implementation and evaluation of these approaches contribute to the ongoing efforts to improve real-time segmentation networks and facilitate more efficient deployment of computer vision applications on low-power devices.

Keywords: semantic segmentation, real-time segmentation, future segmentation, efficient transformers, patch pausing, ambiguous segmentation, discrete diffusion.

Résumé

L'apprentissage profond a révolutionné le domaine de la vision par ordinateur, un succès largement attribuable à la taille croissante des modèles, des ensembles de données et de la puissance de calcul. Cependant, de nombreuses applications de vision par ordinateur sont déployées sur des appareils embarqués de faible puissance et nécessitent des capacités de traitement en temps réel. Cela est particulièrement le cas pour la segmentation sémantique, une tâche de prédiction dense exigeant une importante capacité de calcul et de mémoire. Dans cette thèse, nous approfondissons divers aspects des réseaux de segmentation en temps réel et formulons des approches visant à accroître leur efficacité et à traiter l'ambiguïté inhérente de certains contextes.

Tout d'abord, nous introduisons une métrique de segmentation qui tient compte de la latence et fournit une métrique utile pour les situations pratiques. Nous soulignons le rôle clé de 'l'anticipation' dans les réseaux temps réel, c'est à dire la capacité de ces systèmes à prédire la segmentation des entrées futures. En conséquence, nous proposons un réseau convolutif à anticipation qui tire parti d'une couche convolutive innovante. Celle-ci exploite la cohérence temporelle des séquences vidéo et réutilise des opérations faites précédemment afin de réduire la charge de calcul. Par la suite, nous nous intéressons aux réseaux de segmentation de type *Transformer* et proposons une méthode pour les accélérer en utilisant une 'mise en pause des patches'. Notre réseau interrompt le traitement des patches dès lors qu'il juge qu'ils ont déjà été correctement segmentés, en se basant sur une mesure de confiance quant à ses prédictions. Nos résultats expérimentaux révèlent que plus de la moitié des patches peuvent être mis en pause tôt dans le processus, avec un impact pratiquement négligeable sur la précision. Cette étude se conclut par l'introduction d'un modèle de diffusion discrète pour la segmentation. Ce modèle permet de générer plusieurs segmentations potentielles pour une entrée donnée tout en suivant la distribution des données d'apprentissage. En combinant ce modèle de diffusion avec un schème autoregressif, nous mettons en évidence sa capacité à générer des prédictions de segmentation à long terme.

La mise en œuvre et l'évaluation de ces approches contribuent aux efforts en cours pour améliorer les réseaux de segmentation en temps réel et favorisent un déploiement plus efficient des applications de vision par ordinateur sur des dispositifs à faible puissance.

Zusammenfassung

Deep Learning hat das Feld Computer Vision revolutioniert, ein Erfolg, der hauptsächlich auf die zunehmende Größe von Modellen, die Erweiterung von Datensätzen und gesteigerte Rechenleistung zurückzuführen ist. Allerdings werden zahlreiche Anwendungen der Computer Vision auf energieeffizienten eingebetteten Systemen eingesetzt und erfordern Echtzeitverarbeitungsfähigkeiten, welche die Größe moderner Modelle oft nicht bietet. Dies trifft insbesondere auf die semantische Segmentierung zu, eine dichte Klassifizierungsaufgabe, die speicher- und rechenintensiv ist. In dieser Arbeit untersuchen wir verschiedene Aspekte von Echtzeit-Segmentierungsnetzen und stellen Methoden vor, um sie effizienter zu gestalten und 'Ambiguität' zu bewältigen.

Zunächst führen wir eine Latenz-bewusste mIoU-Metrik ein, die die Netzwerklatenz berücksichtigt und eine praktische Metrik für Fachleute bietet. Wir erklären, warum Echtzeitsysteme vorausschauend sein sollten, um die Segmentierung zukünftiger Eingaben vorhersagen zu können. Infolgedessen schlagen wir ein vorausschauendes Convolutional Network vor, das eine innovative Convolution-Schicht nutzt. Diese Schicht nutzt die zeitliche Kohärenz von Videoframes, um die Gesamtberechnung zu reduzieren, indem sie zuvor berechnete Features wiederverwendet. Darüber hinaus schlagen wir eine Methode vor, um *Transformer*-basierte Segmentierungsnetze zu beschleunigen, indem wir Patch-Pausen einlegen. Unser Netzwerk stoppt die Verarbeitung von Patches, von denen angenommen wird, dass sie bereits korrekt segmentiert sind, indem es ein Maß zur Bewertung des Vertrauens des Netzwerks in seine Vorhersage verwendet. Unsere experimentellen Ergebnisse zeigen, dass mehr als die Hälfte der Patches frühzeitig pausiert werden können, mit fast vernachlässigbaren Auswirkungen auf die Genauigkeit. Zum Schluss stellen wir ein Modell für diskrete Diffusion für die Segmentierung vor. Dieses Modell ermöglicht die Generierung mehrerer potenzieller Segmentierungen für eine gegebene Eingabe basierend auf der Verteilung der Trainingsdaten. Darüber hinaus integrieren wir das Diffusionsmodell in ein autoregressives Schema, welches zukünftige Vorhersagen erzeugen kann, und demonstrieren die Effektivität dieses Schemas.

Die Umsetzung und Auswertung dieser Ansätze tragen zu den laufenden Bemühungen bei, die Echtzeit-Segmentierungsnetzwerke zu verbessern und einen effizienteren Einsatz von Computer Vision-Anwendungen auf energiesparenden Geräten zu fördern.

Contents

Acknowledgements	i
Abstract (English / Français / Deutsch)	iii
Introduction	1
1 Real-Time Segmentation Networks should be Latency Aware	11
1.1 Introduction	11
1.2 Related Work	12
1.2.1 Image Semantic Segmentation	12
1.2.2 Real-time Semantic Segmentation	13
1.2.3 Video Scene Segmentation Networks	14
1.3 A new task for real-time networks	15
1.3.1 Motivation for latency awareness	15
1.3.2 Defining a new objective for real-time networks	16
1.3.3 Corresponding Latency-Aware Metric	17
1.3.4 Use with video datasets	17
1.4 Dataset, Models and Experimental setup	18
1.4.1 Dataset	18
1.4.2 Networks	19
1.4.3 Adapting SwiftNet for our task	21
1.4.4 Training	22
1.5 Experiments and Results	22
1.5.1 Effect of latency on the LAmIoU	22
1.5.2 Optimizing network training for our latency-aware objective	24
1.5.3 Input translations experiment for increased Receptive Field	26
1.6 Conclusion	29
2 Convolutions with partial channel update for future video segmentation	31
2.1 Introduction	31
2.2 Related Works	32
2.3 Method	34

Contents

2.3.1	Idea	34
2.3.2	Masked convolutional layers	34
2.3.3	Constraints on binary masks	35
2.3.4	Mask generation	36
2.4	Experimental Setup and Models	37
2.4.1	Dataset	37
2.4.2	Networks	37
2.4.3	Slimming networks	38
2.4.4	Evaluation details	39
2.4.5	Training details	39
2.5	Results	40
2.5.1	Future segmentation prediction	40
2.5.2	Controlling the speed/accuracy trade-off	41
2.5.3	Evaluation Offset	41
2.5.4	Training Offset	42
2.5.5	Bi-sequence training	43
2.5.6	Ablation on CWM convolutions position	45
2.5.7	Bi-step vs Random generator	45
2.5.8	Performance on T+1, T+2, T+3	45
2.6	Conclusion	47
3	PAUMER: Patch Pausing Transformer for Semantic Segmentation	49
3.1	Introduction	49
3.2	Patch pausing transformer for Semantic Segmentation	52
3.2.1	Using Entropy as a criterion for patch-pausing	53
3.2.2	Training PAUMER - One training for many pause configurations	53
3.3	Related Work	55
3.3.1	Segmentation using Transformers	55
3.3.2	Token sparsification methods	56
3.3.3	Early-exit methods	56
3.4	Experiments	57
3.4.1	Datasets and Evaluation	57
3.4.2	Results	59
3.5	Discussion	60
3.6	Conclusion	62
	Appendix - Chapter 3	63
3.A	Pseudocode for Patch Pauser and Assembler	63
3.B	Backbone details	63
3.C	Brief introduction to Segmenter	63

3.D	Patch pausing's limitations	65
3.E	Influence of the training pause ratio $\tau_l - \tau_h$	66
3.F	Trading off mIoU for higher throughput	66
3.G	Influence of the auxiliary loss weight λ	67
3.H	Interplay of Pause location and Pausing proportion τ	68
3.I	Using Early Exit at test time	70
3.J	Comparing SETR, Segmenter, EarlyExit using Segmenter	71
3.K	Importance of task specific pretraining	71
3.L	Entropy as a measure of patch-pausing	72
4	Handling ambiguous contexts: An application of Discrete Diffusion model to Segmentation	75
4.1	Introduction	75
4.2	Related Work	77
4.3	Method	78
4.3.1	Diffusion Models	78
4.3.2	Discrete Diffusion	78
4.3.3	Discrete Diffusion Models for Segmentation	79
4.3.4	Motivating Example: The rectangle world	80
4.4	Experiments	82
4.4.1	Lung Cancer Dataset	82
4.4.2	Car Intersection Simulator	84
4.4.3	Cityscapes	87
4.5	Conclusion	91
Appendix - Chapter 4		92
4.A	Model Architecture details	92
4.B	LIDC experiment details	92
4.B.1	Training details	92
4.B.2	Metric	92
4.B.3	Qualitative results	93
4.C	Car simulator experiment details	95
4.C.1	Training details	95
4.C.2	Transformer-Encoder architecture	95
4.C.3	Impact of the number of samples on the miss rate	96
4.C.4	Trajectory extraction and comparison	96
4.C.5	Qualitative results	98
4.D	Cityscapes experiment details	99
4.D.1	Training details	99
4.D.2	Moving Object Results	99

Contents

4.D.3 Qualitative Results	99
4.E Motivating Example Details	100
Conclusion	105
Bibliography	109
Curriculum Vitae	125

Introduction

The sharp increase in data availability and computational power in the past decades has ushered in a new age for machine learning and artificial intelligence. In the rapidly changing machine learning landscape, the field of Computer Vision has greatly benefited from this evolution. Among the diverse branches of this thriving field, segmentation has emerged as a particular area of interest. This thesis delves into the intricacies of semantic segmentation, specifically focusing on tackling the computation challenges that arise when aiming for real-time implementation.

Semantic Segmentation

Semantic segmentation is the intricate process of assigning a class label to every pixel within an image, effectively partitioning it into distinct, semantically meaningful components. This is a key task in computer vision as it aims to grasp the fine-grained details of image content at the pixel level. The output of a semantic segmentation system is a dense prediction, *i.e.*, a pixel-wise classification of the image, referred to as a segmentation map.

As image segmentation is a broad area, semantic segmentation needs to be distinguished from other tasks, which have related but different objectives:

- **Object Detection:** Delineate bounding boxes around objects within the image.
- **Instance Segmentation:** Identify individual object instances by assigning each pixel to a specific instance. It does not differentiate between different classes.
- **Semantic Segmentation:** Categorize pixels into various semantic classes without concern for individual instances within the image.
- **Panoptic Segmentation:** The combination of semantic and instance segmentation, where each pixel is assigned a class label and an instance label.

Introduction

Segmentation benefits numerous fields and has a wide array of applications. In autonomous driving, semantic segmentation enables vehicles to accurately discern the nature and positions of nearby elements, such as pedestrians, vehicles, and traffic signs. This information, fused with additional sensor inputs, enables the autonomous system to respond appropriately and navigate complex environments. In medical imaging, semantic segmentation is used to identify and delineate anatomical structures in images such as MRI and CT scans, which is crucial for diagnosis and treatment planning. In particular, real-time segmentation can play a crucial role in the context of interventional surgery, where the ability to segment and analyze anatomical structures or anomalies in real-time can aid in guiding medical procedures, reducing risks, and improving patient outcomes. Similarly, in remote sensing and GIS, image segmentation aids in the process of object-based image analysis. It can help segment different geographical features from satellite imagery like vegetation, water bodies, built-up areas, and other land features. Beyond these domains, semantic segmentation has also been applied to robotics, allowing robots to navigate and interact with their surroundings, to agriculture, video surveillance, and more. Some examples of semantic segmentation applied to different domains are displayed in Figure 1.

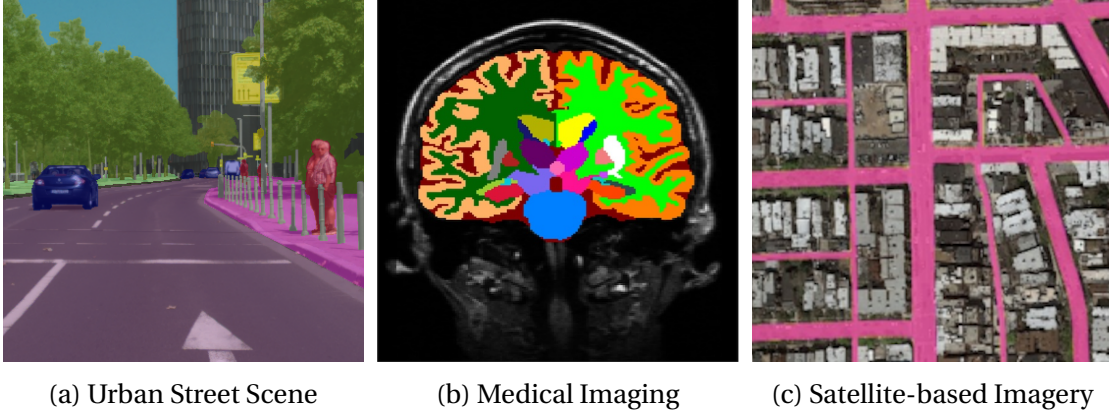


Figure 1: Examples of semantic segmentation in different domains. Different colors represent different semantic categories.

The usual metric for evaluating the segmentation performance for a given class is the Intersection over Union, often written IoU , which is a measure of the overlap between the predicted segmentation map and the ground truth. Precisely, the IoU of class i is the ratio of the intersection area between the prediction and the ground truth of class i to the union area of the two:

$$IoU_i = \frac{TP_i}{TP_i + FP_i + FN_i},$$

where True Positive (TP) are the pixels correctly classified as belonging to class i , False Positives (FP) are the pixels wrongly classified as class i (they belong to a different class), and False Negatives (FN) are the pixels of class i that were missed by the classifier (they were classified as a different class).

The IoU ranges from 0 to 1, with 1 being a perfect match between the prediction and the ground truth, and 0 when they have no intersection. When computing the IoU over a dataset rather than for an individual image, we aggregate the intersection and union areas from all images contained within the dataset. Specifically, for the numerator, we calculate the cumulative intersection area - that is, the shared area between our predicted segmentation and the actual ground truth, summed across all images. For the denominator, we compute the cumulative union area, which is the combined area of both our predicted segmentation and the respective ground truth, aggregated over all images. When the dataset comprises multiple classes, the overall segmentation performance is evaluated using the Mean Intersection over Union, referred to as *mIoU*. This calculates the IoU for each class separately and then averages these class scores. Therefore, the mIoU of a set of two predictions is in general not equal to the average of the mIoU of each prediction.

Real-time Semantic Segmentation

Any real-time system has a specific latency requirement, which sets the maximum amount of time allowed to process a single input. Designing such a system therefore requires a careful balance between the accuracy of the system, its latency, and its memory footprint. Given the real-time nature of numerous applications involving segmentation, it is becoming increasingly important to focus on the development of more efficient networks, and even more so since semantic segmentation, as a dense prediction task, is notably computationally intensive.

Over the course of this doctoral study, the body of work addressing this issue grew rapidly. Initially, the pursuit of faster segmentation networks primarily involved replacing sluggish encoder backbones with faster counterparts initially designed for classification tasks, such as MobileNet (Howard et al., 2017). Progressively, custom multi-scale designs were developed and aimed at reducing latency by performing the bulk of the processing at lower resolutions. With the introduction of Transformers into the field, various techniques emerged, including multiple patch-dropping and merging methods. To provide a more comprehensive overview, we can categorize existing efforts to enhance network efficiency into three distinct and orthogonal categories:

- **Modifying the Network Architecture:**
 - Improving design: This consists in designing the overall network with speed in mind, and in general relies on the development of efficient individual components, such as substituting the conventional convolution layer with depthwise separable convolutions. The MobileNet family (Howard et al., 2017, 2019; Sandler et al., 2018) and SqueezeNet (Iandola et al., 2016) are examples of such classification networks designed for speed. For semantic segmentation, multiple custom models were developed, including Enet (Paszke et al., 2016), ICNet (Zhao et al., 2017), Bisenet (Yu et al., 2020) and SwiftNet (Orsic et al., 2019) for convolutional ones, along with SegFormer (Xie et al., 2021) and RTFormer (Wang et al., 2022) for Transformer based ones. Chapters 1 and 2 of this work are in this category.
 - Post-processing the model: This consists in using quantization and pruning (He et al., 2021) techniques to reduce the number of parameters and accelerate the processing, and is generally done after training.
- **Modifying the Training Process:** By using Knowledge Distillation (Hinton et al., 2015), one aims to retain as much predictive power as possible in a much smaller and more efficient model. This process involves training a smaller, more efficient ‘student’ model under the guidance of a larger, pretrained ‘teacher’ network. Works from Xie et al. (2018) and Liu et al. (2019b) successfully applied this technique to segmentation networks.
- **Modifying the Data:** This consists in modifying the amount of data that goes through the network. These methods have gained more traction post the transformer revolution as they leverage the particular architecture of transformers. For classification, various methods proposed to drop (Yin et al., 2022; Rao et al., 2021) or merge (Xu et al., 2022; Liang et al., 2022b; Kong et al., 2022; Bolya et al., 2022) patches not relevant for the task. Our method, PAUMER, presented in Chapter 3, belongs to this category. It is tailored for segmentation by proposing to pause the processing of some patches.

However, it is important to highlight that what most real-time systems generally require is the segmentation of consecutive video frames, which is a key distinction because consecutive frames are highly correlated. This observation holds significant importance for our research, as it hints that the processing time of segmentation networks could potentially be reduced by repurposing previous activations and segmentation maps.

As we elaborate on in Chapter 1, real-time video segmentation and future segmenta-

tion are logically related by the need for many real-time systems to anticipate upcoming states. We therefore review below the various approaches to future segmentation.

Future Segmentation

Future semantic segmentation is the task of predicting semantic labels in forthcoming video frames. It is a challenging task as we need to solve two problems simultaneously: predicting the semantic segmentation map and anticipating the motion of objects. Multiple solutions have emerged to tackle this problem, some of which are based on optical flow, directly or indirectly, while others are based on direct forecasting (at the pixel level) or feature forecasting (at the feature level). Here, we provide a brief overview of the most prominent approaches.

Optical Flow: Optical flow techniques aim to reconstruct dense 2D motion between consecutive image frames. They often rely on deep convolutional models (Dosovitskiy et al., 2015; Sun et al., 2018) to estimate motion, due to their capability to guess motion where the correspondence is absent or ambiguous. However, since the optical flow is estimated between a past frame and a future frame not yet observed, it is bound to be inaccurate, especially with articulated objects like humans.

Temporal Alignment: Temporal alignment techniques focus on warping features or labels from a segmented key-frame to the current frame, which aids in speeding up semantic segmentation in video (Zhu et al., 2017). These methods are also used to enlarge training datasets by warping ground truth labels to neighboring unlabeled frames, and they aim to enforce temporal consistency with past frames (Gadde et al., 2017; Saemann et al., 2019).

Semantic to Semantic (S2S) Forecasting: Luc et al. (2017) were the first to introduce direct semantic forecasting, also known as S2S, in future semantic segmentation. Their model directly predicts future semantic segmentation based on past segmentations. Rochan et al. (2018) improved it using a convolutional LSTM, which was later enhanced with multiscales and attention by Chen and Han (2019).

Flow-Based (M2M) Forecasting: Extensive training data is required in the S2S approach to grasp motion patterns, but its efficiency can be optimized by incorporating geometric features indicative of 2-D motion (Jin et al., 2017b). A method that enhances this concept is Flow-Based Forecasting, also referred to as M2M, which combines optical flow with direct semantic forecasting. Specifically, M2M warps the last dense prediction based on forecasted optical flow. A prime example of this approach is presented in (Terwilliger et al., 2019), where a convolutional LSTM is used to process

Introduction

optical flows from three observed frames, generating a prediction for future optical flow.

Feature to Feature (F2F) Forecasting: Feature-level forecasting, also known as F2F forecasting, maps past features to their future counterparts. The pioneering method was developed by [Luc et al. \(2018\)](#) who harnessed the power of the Mask R-CNN model integrated with a feature pyramid network — each level of the pyramid was independently forecasted employing dilated convolutions. Progressing from this approach, further research ([Sun et al., 2019](#); [Hu et al., 2021](#)) incorporated a convolutional LSTM module at every pyramid level, and inter-level connections to share context. However, forecasting at fine resolution is computationally expensive ([Couprie et al., 2018](#)), alternative research suggests performing single-level feature forecasting at the coarsest level ([Šarić et al., 2019](#); [Chiu et al., 2020](#)). The model F2MF ([Saric et al., 2020](#)) expands this concept by integrating a feature-to-motion head and a correlation module to capture spatio-temporal feature relations. [Lin et al. \(2021\)](#) also extend single-level feature forecasting through the incorporation of an autoencoder. The encoder computes a condensed representation of the multilevel feature pyramid, predicts the future representation in the latent space using bidirectional ConvLSTM, and subsequently decodes this forecasted representation to derive the future feature pyramid.

Before delving into the details of our work, we finally provide an overview of the history and current state of semantic segmentation in the context of deep learning. This historical background will prove beneficial in understanding the evolution of techniques that have been developed to meet the specific challenges posed by real-time processing requirements.

Deep Learning for Segmentation

Deep learning for segmentation began with the seminal Fully Convolutional Network (FCN) from [Long et al. \(2015\)](#) that proposed to substitute final fully connected layers with convolutions in the convolutional networks initially built for classification in order to obtain segmentation maps. However, this technique significantly reduces the image resolution in order to retrieve semantic information. Therefore, multiple techniques were introduced to address the challenges of capturing contextual information while preserving spatial resolution. One common way is to use a decoder network with skip-connections, as first proposed [Ronneberger et al. \(2015\)](#) and [Badrinarayanan et al. \(2017\)](#) with the U-Net (fig. 2) and SegNet architecture respectively. The more recent SwiftNet ([Orsic et al., 2019](#)) follows these principles and uses a pretrained ResNet backbone, which makes it both fast and accurate. This model has proven to be

a strong baseline and has been employed as such in several chapters of this work. In addition to the use of a decoder, the latest models of the Deeplab family (Chen et al., 2014, 2017a,b, 2018) also propose to incorporate dilated convolutions to alleviate the need for downsampling the image.

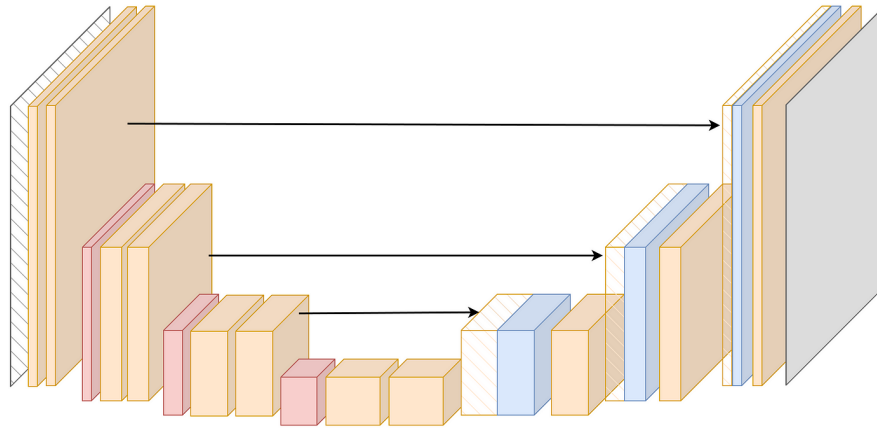


Figure 2: The seminal U-Net architecture (Ronneberger et al., 2015), is composed of an encoder, a decoder, and skip connections. Orange blocks represent convolutional layers, red blocks represent max-pooling layers, and blue blocks represent upsampling layers. Arrowed lines represent skip connections. (Source: *Towards Data Science*).

The Transformer architecture has been very successfully applied in recent years to the field of computer vision, notably with the apparition of the Vision Transformer (ViT) (Dosovitskiy et al., 2021). Transformers have quickly ventured into the field of segmentation and demonstrated state-of-the-art results. Pioneering networks include SETR (Zheng et al., 2021), Swin Transformer (Liu et al., 2021), Segmenter (Strudel et al., 2021), and SegFormer (Xie et al., 2021).

However, they show some disadvantages both for training and inference. First, they are in general slower than their convolutional counterparts, in part due to the computationally expensive self-attention operation. Additionally, they often demonstrate a slightly lower segmentation precision on high-frequency objects, a consequence of partitioning images into patches and classifying entire patches as a single category. Some hierarchical variants of ViT such as Swin (Liu et al., 2021) and PVT (Wang et al., 2021) have been designed to mitigate this issue and introduce vision-specific inductive biases. Furthermore, they can be less flexible to iterate with. Indeed, their reliance on pretraining with large classification datasets, such as ImageNet, is typically crucial for achieving strong segmentation performance (as exemplified by Segmenter). This pretraining process is generally very computationally demanding, and contrary to convolutional networks, making even slight modifications to the transformer architecture can compromise the usability of the pretrained weights. These limitations

Introduction

restricted research on architectural improvements and sparked particular interest in accelerating segmentation transformers.

Nowadays, state-of-the-art (SOTA) networks in semantic segmentation are for a large part adapted from best-performing networks on ImageNet classification, using them as backbones inside well-performing segmentation architectures such as UperNet (Xiao et al., 2018) or Mask2Former (Cheng et al., 2022). An increasing number of these SOTA classification networks are multi-modal language and vision models, such as CoCa (Yu et al., 2022) or PaLI (Chen et al., 2022c). Such multi-modal networks have also been applied to the segmentation task, with the recent introduction of SEEM (Kirillov et al., 2023) and SAM (Zou et al., 2023), two models leveraging a joint visual-semantic space and that can segment any object in an image, showing strong zero-shot segmentation capabilities.

Thesis outline and contributions

The subsequent four chapters in this dissertation are derived from four conference proceedings papers. The first three (Courdier and Fleuret, 2020, 2022; Courdier et al., 2022) are broadly focusing on enhancing the efficiency of segmentation networks, both in terms of computational power and latency. The last one (Courdier et al., 2023) proposes a way to model the inherent uncertainty of certain segmentation contexts. Each chapter stands alone and is designed to be readable in any order.

In Chapter 1, which is based on Courdier and Fleuret (2020), we focus on real-time segmentation networks and show that existing metrics fall short in conveying the actual accuracy obtained in practice for a real-time system. We propose an alternative metric that takes latency into account, named LAmIoU, along with a corresponding shift in the objective of real-time segmentation tasks. Consequently, we introduce the concept of *anticipatory* networks. These are networks specifically designed to factor in their latency during operation and are optimized based on the objective set by our proposed metric. Finally, we adapt and train some common networks for this new objective and confirm its practical relevance.

In the context of future segmentation and building upon the necessity for the design of anticipatory networks outlined in Chapter 1, we propose in Chapter 2 a temporal masking scheme for convolutional networks based on Courdier and Fleuret (2022). This novel design is specifically crafted for real-time video processing. We introduce a temporally masked convolution that leverages previously computed features to perform fewer operations while having access to past information. We show that substituting all conventional convolutions with their temporally masked counterparts

within widely used segmentation architectures enhances the predictions of future segmentations but also results in a speed boost of approximately 20%.

In Chapter 3, which is based on [Courdier et al. \(2022\)](#), following the advent of transformer networks in Computer Vision, we saw a significant opportunity for speed improvement. As we elaborate on in this chapter, different regions of an image exhibit varying levels of segmentation complexity, and we leverage this diversity to dynamically allocate computational resources to the areas that require it the most. Transformer networks are particularly well-suited for such dynamic allocation because they operate on unordered image patches, which we can decide to process more or less. We propose to use the patch's intermediate entropy as an allocation criterion and show that patches that are processed further are indeed on boundaries or contain higher-frequency elements. While results vary with the dataset complexity, our experiments on Cityscapes revealed that we can get a 2X speed improvement with a minimal reduction of less than 1% in absolute mIoU.

Finally, in Chapter 4, which is based on [Courdier et al. \(2023\)](#), we tackle another aspect of future segmentation networks, namely the inherent ambiguity of the prediction. We delve into the application of generative models as a solution to model uncertainty in non-deterministic scenarios. More specifically, we explore the utilization of discrete diffusion models within the segmentation task, focusing on situations where segmentation outcomes are not completely determined by the input data. We apply our approach to the field of medical imaging and demonstrate that our segmentation diffusion models effectively allow the sampling of different output maps. For the future segmentation task, we also introduce an autoregressive scheme that enables longer-term forecasting - a strategy we validate using a custom car intersection simulator we developed.

1 Real-Time Segmentation Networks should be Latency Aware

This chapter is based on

Courdier, E. and Fleuret, F. (2020). Real-time segmentation networks should be latency aware. In *Proceedings of the Asian Conference on Computer Vision*

1.1 Introduction

Recent image segmentation networks achieve near-human level performance due to their expressive power, and more focus is put 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 to predict the input frame's segmentation, which is compared to the input frame's ground truth 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*.

When aiming for real-time, one has to strike a balance between speed and accuracy. This becomes clear when analyzing real-time segmentation benchmarks, where networks are ranked according to both accuracy metrics and latency. Often, accuracy-latency charts also allow to quickly estimate a new network's 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

Chapter 1. Real-Time Segmentation Networks should be Latency Aware

terms, we want to help answer the question of how the system’s latency will affect the relevance of its predictions.

In this first chapter, we have set as our objective to devise a relevant metric for real-time networks. To this end, 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 a specific 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). This thesis being centered around scene semantic segmentation, we will focus on this task and perform our experiments on it.

This chapter is organized as follows:

- In Section 1.2, we review available methods for semantic segmentation at the time of this research, both for images and videos, and in particular those tackling real-time contexts.
- In Section 1.3, we motivate and define our proposed objective for semantic segmentation, along with the associated metric.
- In Section 1.4, we detail the dataset, models, and experimental setup that we employed for our study.
- Finally, in Section 1.5, we report the results of our experiments. Specifically, we delve into the influence of latency and the training configurations on the performance metric we have introduced.

1.2 Related Work

1.2.1 Image Semantic Segmentation

Most popular approaches for tackling Semantic Segmentation use a variant of powerful deep classification networks that are made fully convolutional, with all final fully connected layers replaced by convolutions. That seminal idea is at the core of the FCN paper (Long et al., 2015).

The main issue with this technique is that it significantly reduces the image reso-

lution 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 technique is to use a *decoder network* plugged after the FCN to upsample the segmentation map using transposed convolution, as first did [Ronneberger et al. \(2015\)](#) and [Badrinarayanan et al. \(2017\)](#) with SegNet and U-Net. This setup allows to merge spatially rich shallow layers into semantically rich deeper layers.

DeepLab v2 ([Chen et al., 2017a](#)) later proposed to use *dilated convolutions* ([Yu et al., 2017](#)) 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, [Zhao et al. \(2016\)](#) proposed to use a “*Spatial Pyramidal Pooling*” (SPP) module ([He et al., 2015b](#)) 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 semantics.

Many works followed with techniques to produce high-quality segmentation ([Tang et al., 2018](#); [Takikawa et al., 2019](#); [Zhu et al., 2019b](#); [Valada et al., 2019](#)), including better ways to extract features ([Peng et al., 2017](#); [Zhang et al., 2018b](#); [He et al., 2019](#); [Wu et al., 2019](#)) and to take into account context ([Ding et al., 2018](#); [Yuan et al., 2019](#)). Some recent works also proposed attention-based methods ([Li et al., 2019b](#); [Fu et al., 2019](#); [Huang et al., 2019](#); [Tao et al., 2020](#); [Zhang et al., 2020a](#); [Choi et al., 2020](#)) and neural architecture search for image segmentation ([Liu et al., 2019a](#); [Zhang et al., 2020b](#); [Nekrasov et al., 2020](#)).

On a different application domain, similarly to the change we propose, backbones with enlarged front-end have been used with success for object detection ([Zhu et al., 2019a](#)) where the authors are training their network without ImageNet pretraining.

1.2.2 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 central concept in the development of fast networks revolves around the early downsampling of images. This allows the bulk of the processing to occur at a reduced resolution, thus circumventing the need for extensive full-resolution computation. This idea is key to the design of the precursors ENet ([Paszke et al., 2016](#)) and ICNet ([Zhao et al., 2017](#)). The latter, in particular, performs multi-scale processing with a special fuse block to merge information from different scales.

One way of optimizing neural network architecture for speed is to replace regular convolutions with a more efficient alternative. One such method is to use factorized convolutional blocks, *e.g.* factorizing kernels $k \times k$ into $1 \times k$ and $k \times 1$ kernels as does ERFNet (Romera et al., 2017). It can also be achieved using group convolutions, and methods such as ShuffleNet (Zhang et al., 2018a) propose different ways to create connections between groups. Another way is to employ Depthwise Separable Convolutions (DSC), which are the combination of depthwise and pointwise convolutions. These DSC are used to lower the number of parameters and make the inference faster, at the cost of accuracy. They are used broadly in MobileNets (Howard et al., 2017; Sandler et al., 2018).

BiSeNet and BisenetV2 (Yu et al., 2018a, 2020) proposed a way to separate the localization problem from the semantic extraction problem, and then merge the two information appropriately.

Other works such as the FasterSeg network (Chen et al., 2019) also use Neural Architecture Search to successfully discover fast neural architectures for semantic segmentation.

Within the realm of fast segmentation networks, Swiftnet, introduced by Orsic et al. (2019), stands out for its architectural approach. It leverages a lightweight ImageNet-pretrained Resnet as its backbone and augments it with a straightforward decoder that utilizes lateral connections, much like the U-Net architecture. For the research presented in this chapter, we opt for SwiftNet as one of our base networks due to its simple design and processing speed.

1.2.3 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 (Shelhamer et al., 2016) 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 (Zhu et al., 2017), even though their design is not meant for real-time.

The work (Li et al., 2018) 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 (Luc et al., 2017; Jin et al., 2017a,b).

Video temporal coherence is also used along with representation warping to produce better future segmentation maps. Warping is either applied at the feature level (Saric et al., 2020; Gadde et al., 2017) or directly at the segmentation map level (Terwilliger et al., 2019), and possibly combined with existing features to produce the output. However, these works are not focused on time efficiency.

A Bayesian approach for multi-modal future prediction of scene segmentation was also proposed in (Bhattacharyya et al., 2018) that allows to take into account model and observation uncertainty.

Temporally Distributed Network (Hu et al., 2020) was introduced for fast video segmentation. It uses a teacher-student design where fast student networks have to predict - in turn - part of the feature map of the teacher network.

1.3 A new task for real-time networks

1.3.1 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. Unfortunately, accuracy-latency charts fall short in providing this critical information as they cannot demonstrate the actual accuracy compensated for the time-delay that one would meet in practical scenarios. Moreover, it does not provide either a total order relation or a ranking to compare various real-time networks, as can be seen on benchmark websites such as *paper-*

*paperswithcode.com*¹. As a result, we believe that it is relevant to introduce a new objective for the segmentation task that takes into account network latency. This allows to get meaningful accuracy information and practical benchmarking of networks.

1.3.2 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 an 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:

$$\text{acc}(F(I_t), S_t) \tag{1.1}$$

while our task proposes to optimize for:

$$\text{acc}(F(I_{t-l_F}), S_t) \tag{1.2}$$

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. Comparing the received information at time t , $F(I_{t-l_F})$, with the ideal information for that instant, S_t , emphasizes the importance of predictive accuracy in dynamic environments.

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.

¹<https://paperswithcode.com/task/video-object-segmentation>

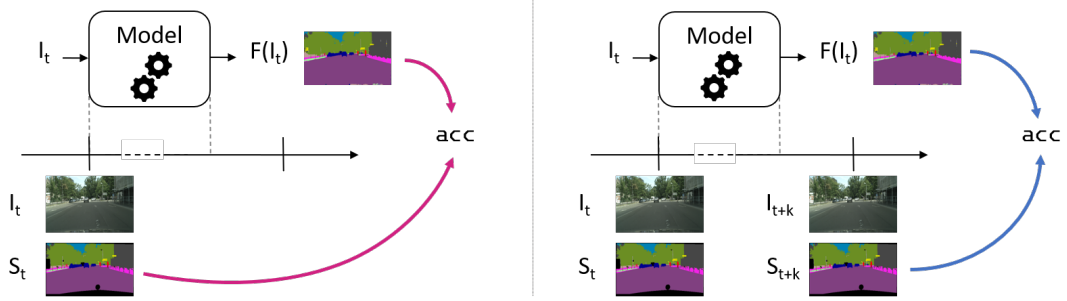


Figure 1.1: Left: How mIoU is computed now; the output of the network is compared to the ground truth segmentation of the *image in input of the network*. Right: Our proposed way to measure mIoU; the output of the network is compared to the ground truth segmentation of the *future image* when the network finishes processing.

1.3.3 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 Equation (1.2).

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.

1.3.4 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 Figure 1.2.

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

$$k_F = \lceil l_F / d \rceil \quad (1.3)$$

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

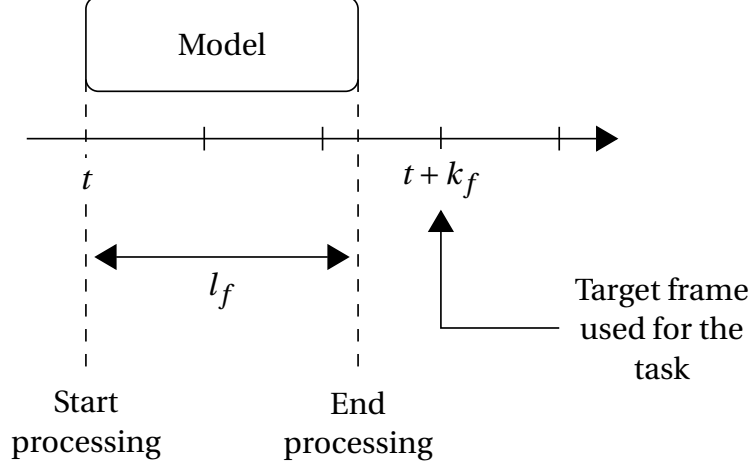


Figure 1.2: Target frame used for our proposed objective

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.

1.4 Dataset, Models and Experimental setup

1.4.1 Dataset

Dense pixel-level manual annotation of videos for scene segmentation is not feasible due to the time and economic costs involved. Cityscapes dataset (Cordts et al., 2016) 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 20th 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 are 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

1.4 Dataset, Models and Experimental setup

2. We determine how many frames of offset k_F this time corresponds to: $k_F = \lceil l_F / 0.06 \rceil$ ($0.06 = 60\text{ms}$)
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.

1.4.2 Networks

For our experiments, we choose two image segmentation networks: SwiftNet ([Orsic et al., 2019](#)) and DeepLab-V3+ ([Chen et al., 2018](#)) with 2 different encoders : ResNet-101 and MobileNet v2 ([Sandler et al., 2018](#)).

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 convolutional.
- 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 the 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.

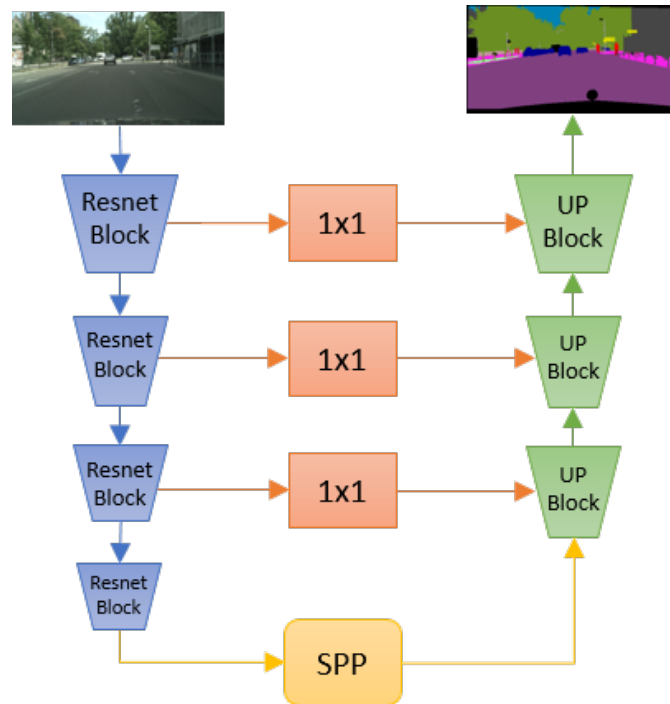


Figure 1.3: SwiftNet architecture

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 (Sandler et al., 2018). It uses depth-wise separable convolutions within “inverted” residual 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.

1.4.3 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 was 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 in which 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:

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

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. We have experimented with non-pretrained ResNet and observed a 6% mIoU drop on average.

The newly created convolutions were initialized using He's initialization ([He et al., 2015a](#)) rule. We will refer to our extended SwiftNet version as **SwiftNet-R18-X**.

1.4.4 Training

All experiments are performed with the PyTorch framework. We use ImageNet-1k pre-trained weights for all encoders in our networks. Both the classical training objectives and latency-aware objectives that is detailed further are used to train the model.

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 optimizer 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. The network is trained for 200 epochs and uses 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 optimizer 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. The network is trained for 200 epochs and uses a poly schedule with a power of 3.

1.5 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.

1.5.1 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 take 2 frames X_{t-1}, X_t as input.

Decay of the LAmIoU with the frame offset

In Figure 1.4, 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.

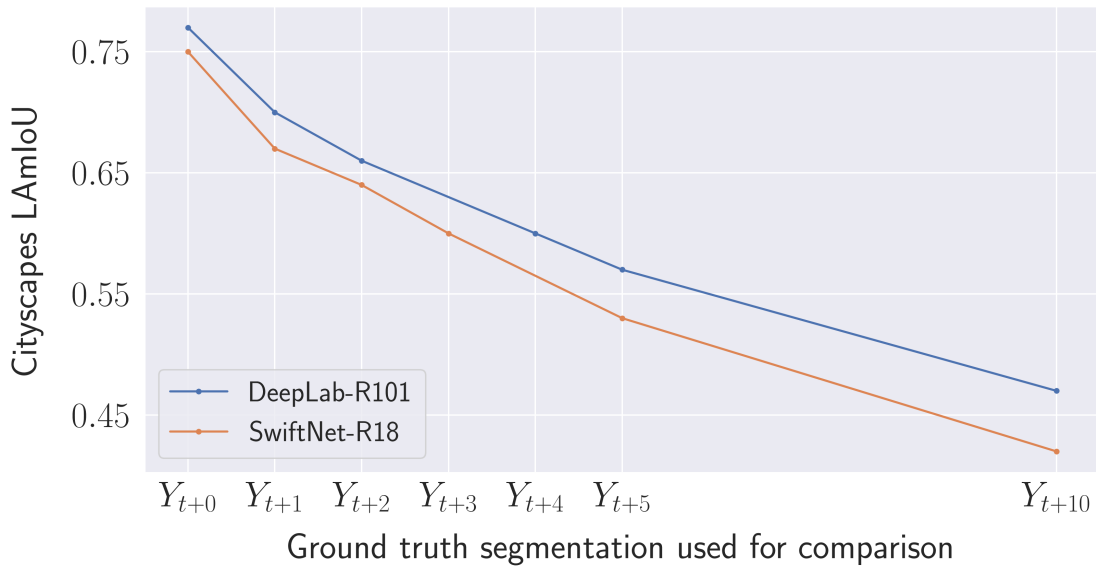


Figure 1.4: LAmIoU decay with different offsets in the objective on Cityscapes validation set

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 this hardware and then train the networks with the corresponding frame offsets. In Figure 1.5, 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.

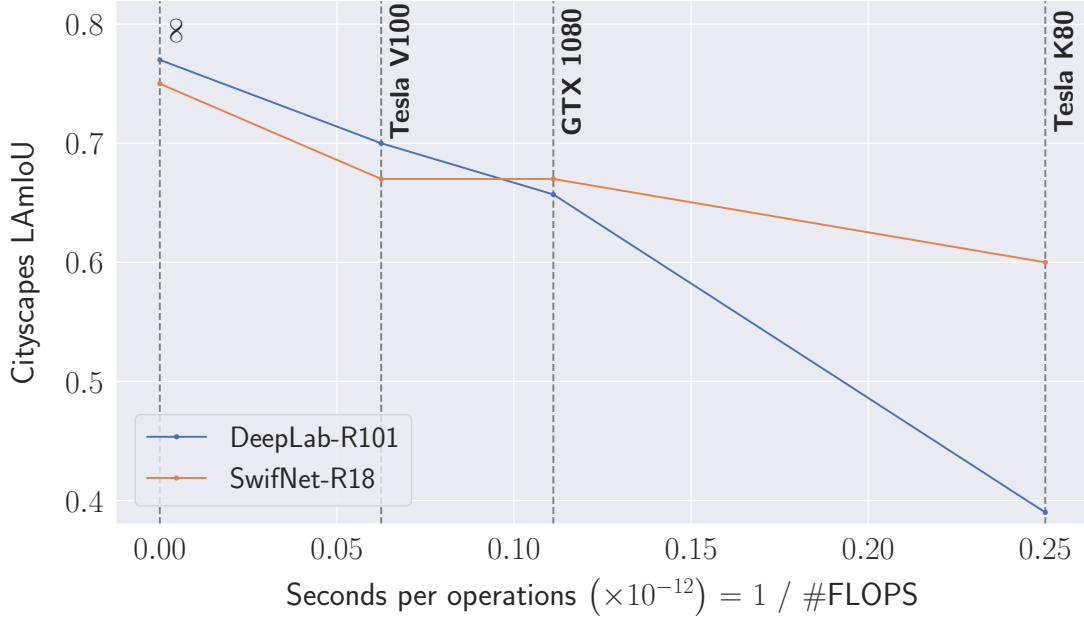


Figure 1.5: LAmIoU decay with hardware speed on Cityscapes' validation set

1.5.2 Optimizing 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 network output. These experiments are performed on a *GTX 1080 Ti* GPU for each of the four networks described in Section 1.4.2.

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 the second line of Table 1.1. 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 the third line of Table 1.1. We can see a slight but consistent increase in the LAmIoU metric for all networks. In Figure 1.6, 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 favors a specific direction.

We conjecture that the network is able to predict some objects' movements 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. However, the network has no way to infer the relative speed of the different instances in the image which makes its prediction relatively inaccurate.

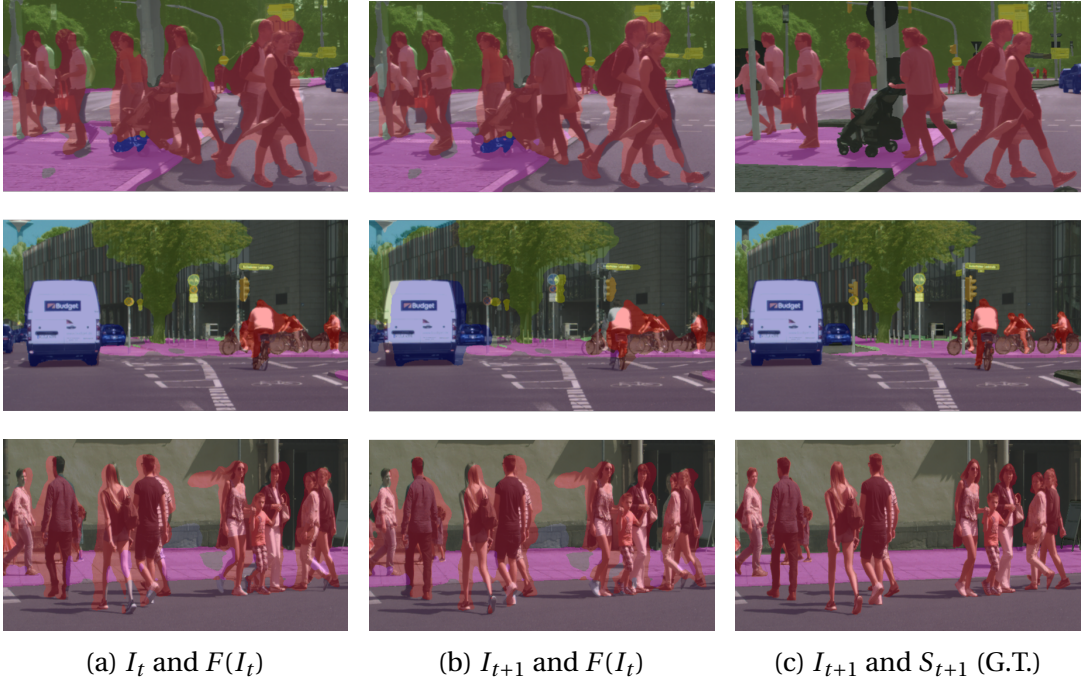


Figure 1.6: Output segmentation of SwiftNet-R18 trained to predict S_{t+1} from I_t . We observe anisotropic blur as the network is able to infer some objects' movement directions from their orientation.

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 the fourth line of Table 1.1 show a consistent improvement of the LAmIoU metric. These numbers confirm the relevance of using previous frames for our latency-aware objective. We reported in Figure 1.7 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 emphasizes the fact that networks constructed for real-time contexts should use previous frames to better predict a future target.

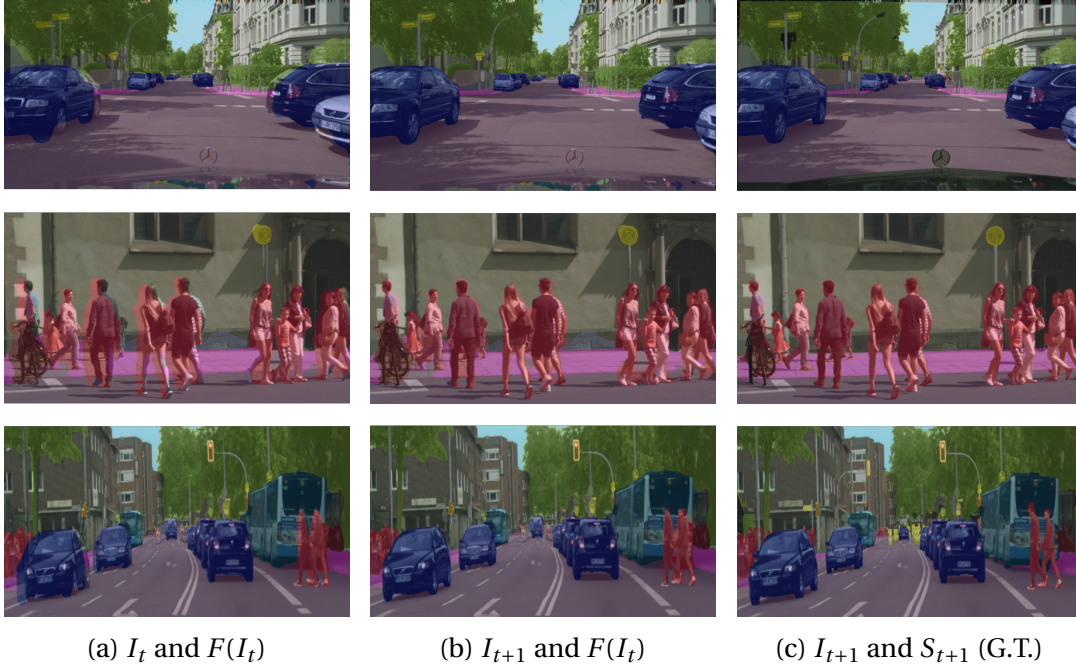


Figure 1.7: Output segmentation of SwiftNet-R18 trained to predict S_{t+1} from (I_{t-1}, I_t) . We observe a more precise segmentation as the network has a way of inferring relative speeds and directions.

1.5.3 Input translations experiment for increased Receptive Field

When processing simultaneously images from different time steps I_{t-1} and I_t , the network needs to have a large receptive field to be able to map objects from one frame to the other. To do so without the need for big kernels, we try to offset this load on the

1.5 Experiments and Results

Table 1.1: Results for the 3 training configurations for the 4 networks. The first line gives the instant mIoU and the next three lines report the 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 1.4.1, and fps of these networks when using a GTX 1080 Ti. Note that the k temporal offset depends on the network, hardware, and the dataset frame rate: 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
LAmIoU	I_t	S_t	S_t	0.77	0.72	0.75	0.74
	I_t	S_t	S_{t+k}	0.50	0.56	0.64	0.63
	I_t	S_{t+k}	S_{t+k}	0.53	0.57	0.65	0.64
	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

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 $\{T_1(I_{t-1}), \dots, T_N(I_{t-1}), I_{t-1}, I_t\}$ corresponding to N different fixed translations T_i . The translation offsets were chosen to span a regular grid around the origin.

While initial experiments seemed promising, we further discovered that the main reason for improved LAmIoU was the additional convolutional layer with a higher number of kernels that we added at the head of the model to handle the translations where we had replaced the first convolution layer

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

with the following block:

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

Chapter 1. Real-Time Segmentation Networks should be Latency Aware

with N the number of translations. Noticing that using additional inputs requires increasing capacity in the early convolutional layers eventually led to the design of SwiftNet-R18-X presented in Section [1.4.3](#) in which we increased the number of kernels in the first two layers.

1.6 Conclusion

In this chapter, we have proposed a different approach to the segmentation task, shifting the conventional objective to incorporate real-time considerations into network predictions. Our novel LAmIoU metric precisely quantifies the achievable mIoU value on a given hardware when accounting for network latency. This metric also enables us to derive a latency-aware per-device ranking, offering a solution to the initial question of identifying the optimal network for real-time applications. It is important to note that this latency-aware metric becomes more pertinent as the dataset's frame rate increases, while lower frame rates may result in a less clear differentiation among networks.

Our focus here was specifically on video scene segmentation, but the same change of objective is relevant and applicable to a wide range of other real-time tasks. We believe that introducing a new latency-aware segmentation objective may encourage research in anticipatory networks. In line with this, we proposed additions to networks and training to perform better under this new metric, and we will continue this exploration of anticipatory designs in the next chapter.

2 Convolutions with partial channel update for future video segmentation

This chapter is based on

Courdier, E. and Fleuret, F. (2022). Borrowing from yourself: Faster future video segmentation with partial channel update. In *2022 26th International Conference on Pattern Recognition (ICPR)*, pages 1–8. IEEE

2.1 Introduction

In the pursuit of improving real-time segmentation with a keen eye on minimizing and accounting for latency, we have emphasized in the previous chapter the need for more designs of anticipatory networks. Building on this insight, this chapter keeps the focus on predicting future frames’ segmentation and proposes to that end a change in the usual convolution operation.

As we have previously argued, properly leveraging the inherent sequential nature of videos may allow substantial time gain by reusing previous computation while also bringing relevant insight from past frames, as did both [Shelhamer et al. \(2016\)](#); [Li et al. \(2018\)](#) by running parts of their network conditionally to the amount of change there is from the previous frame. The changes in convolution layers we introduce in this chapter are in line with this idea and aim at accelerating the segmentation process while at the same time permitting information from previous frames to be available.

Specifically, we introduce **channel-wise masked convolutions**. These convolutions process a full input as standard convolutions but compute only a subset of their output channels at a given time step, following a pre-defined masking schedule. Some previous works proposed to reduce computation by dropping part of the image ([Verelst and Tuytelaars, 2020](#)) depending on what changed ([Habibian et al., 2021](#)), we propose

instead to drop part of the convolution kernel.

While there is a clear gain in FLOPs due to the reduced number of computation operations, the same cannot be said for latency. This is primarily due to the fact that our proposed masked convolution entails the execution of two tensor indexing operations, which can be highly time-consuming on GPU hardware, potentially negating any time saved through the convolutions themselves. Therefore, care has to be taken during the design of the channel mask to ensure that the overall model attains lower latency.

Our proposed convolution design is specifically relevant for future prediction tasks dealing with a stream of correlated inputs, such as video streams. It indeed allows every convolutional layer to have access to part of the previous timestep’s output to make its prediction.

We apply this idea to existing segmentation networks to save FLOPs and wall-clock time. Particularly, it limits the performance drop for future prediction compared to original networks which have just been “slimmed” to run faster but cannot leverage the sequential nature of videos. We also apply channel-wise masked convolutions to slimmed networks to take advantage of both techniques.

This chapter is organized as follows:

- Section 2.2 presents relevant related literature.
- Section 2.3 introduces our time-dependent channel-wise masking scheme for convolutional layers and details their design constraints.
- In Section 2.4, we describe the application of our technique to three segmentation networks and detail the adaptations made to the training and evaluation procedure.
- Section 2.5 presents a series of experiments conducted using these layers to explore their advantages and limitations. The full code is made available via a public repository¹.

2.2 Related Works

Semantic segmentation, just like the rest of the computer vision literature, was shaken to the core by the success of deep learning. The seminal work (Long et al., 2015) introduced fully convolutional networks for segmentation, and several important methods followed, proposing to use decoder networks (Ronneberger et al., 2015;

¹<https://github.com/theevann/fast-cwm-segmentation>

Badrinarayanan et al., 2017), spatial pyramidal pooling (He et al., 2015b; Zhao et al., 2016), dilated convolutions (Yu et al., 2017; Chen et al., 2017a), and current best performing models are now mainly transformer based networks (Xie et al., 2021; Yuan et al., 2021; Yan et al., 2022; Wu et al., 2021; Strudel et al., 2021).

One of the main limitations with current deep segmentation methods is the long inference time, and numerous works have addressed this (Li et al., 2019a; Zhao et al., 2017; Paszke et al., 2016; Orsic et al., 2019; Yu et al., 2018a, 2020; Hu et al., 2020; Li et al., 2020; Gao, 2021; Hong et al., 2021; Fan et al., 2021; Chao et al., 2019; Nirkin et al., 2021). These improvements allow carefully designed networks to run real-time, but any real-time network should be built as a future prediction network (Courdier and Fleuret, 2020).

On the future segmentation forecasting task, different approaches exist. Direct semantic forecasting introduced by (Luc et al., 2017) directly predicts the future segmentation map from past ones (Vora et al., 2018; Bhattacharyya et al., 2018; Rochan et al., 2018; Chen and Han, 2019). Flow-based forecasting (Patraucean et al., 2015; Zhu et al., 2017; Gadde et al., 2017; Nilsson and Sminchisescu, 2018; Terwilliger et al., 2019) uses optical flow computed from past frames to warp past segmentation into future ones. Feature level forecasting predicts future intermediate features from past ones (Saric et al., 2020; Luc et al., 2018; Chiu et al., 2020). Our technique belongs to the direct semantic forecasting category, as it directly predicts future segmentation maps.

One could note that our training with masked convolution kernels resembles that of slimmable networks (Yu et al., 2018b; Yu and Huang, 2019) which perform a different sort of channel-wise masking with varying contiguous masks, although these masks are not designed nor appropriate for future video segmentation.

Our idea also echoes the fast TDNet (Hu et al., 2020) method. When TDNet uses a different small subnetwork at each time step and combines features extracted from several past inputs, our CWM models use different small convolution masks at each time step and replace part of the past features with new ones.

2.3 Method

This section motivates our main idea and describes our proposed convolutional layer and its masking scheme.

2.3.1 Idea

Our goal is to speed up video segmentation networks. As we specifically deal with videos, consecutive frames are alike and lead to similar computed features. We propose leveraging this temporal correlation by only updating a subset of the output channels of convolutions at each time step and re-using the features of the previous frame for the rest of the channels that are not computed.

We introduce a “channel-wise masked” convolution (Section 2.3.2) that works like a normal one but uses a binary mask to select which output channels to compute. This convolution has lower FLOPs by design, but we find that we have to put constraints on the masks (Section 2.3.3) to also have lower latency. Additionally, we choose to use a predefined finite set of masks and define a generator that will create these masks (Section 2.3.4).

Our proposed channel-wise masked (CWM) convolutions can be used in place of the normal ones in any convolutional model to make it faster. We will denote a model using CWM convolutions a CWM-model.

2.3.2 Masked convolutional layers

A channel-wise masked (CWM) convolution is a convolutional layer that uses a binary mask at each forward pass to select which output channels are computed. In practice, this boils down to indexing the kernel tensor with the mask before performing the computation. The result of this convolution is then combined with the previous time step’s result by simply replacing the older features with fresh ones depending on the mask, as can be seen below on Figure 2.1.

This design implies that we have to save the output of every convolution for the next time step. At the very first time step though, there is no previous output to use, so we perform a full convolution without masking. We noticed this is effective in properly initializing the saved outputs.

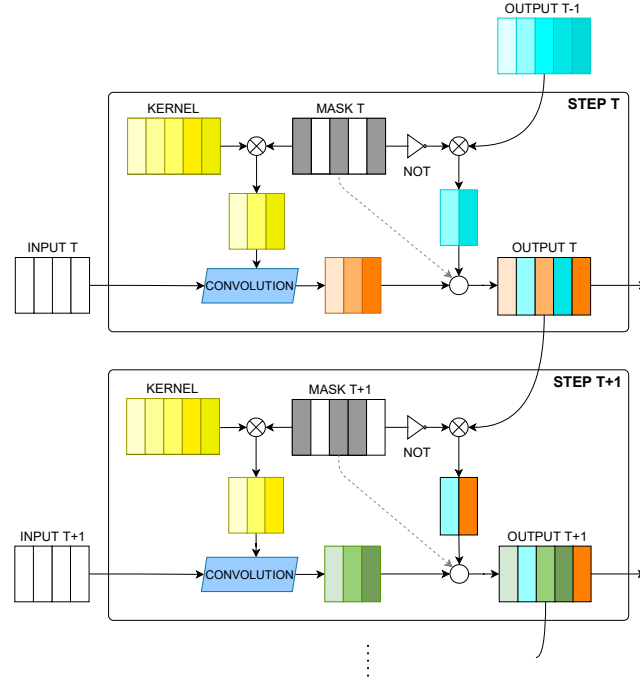


Figure 2.1: Operations performed in a channel-wise masked convolution. The active channels in the mask are greyed. The crossed circle is a masked select. The blank circle represents the interlacing of the current and previous output following the mask indexing information represented by the dashed line.

2.3.3 Constraints on binary masks

Depending on the time step, our masked convolutions perform different computations. All input channels are still used, but each time step sees a different group of convolution's output channels computed, specified through the use of binary masks. However, the improper design of these masks can lead to a significant latency increase, and we therefore have to put constraints on their structure and generation.

Pre-defined While these masks could be chosen online, our initial experiments showed that doing so comes with a significant increase in processing time without accuracy gain. Therefore, for the rest of the chapter, we only use a chosen number of pre-defined masks picked ahead of time. The masks are used sequentially, and once all masks are exhausted we restart from the first mask.

Same number of channels When two masks have a different number of active channels, CWM convolutions using those masks also have different latencies. Since our main goal is to reduce maximum latency, we need each time step to have a similar computation time, so we have to activate the same number of channels in every mask.

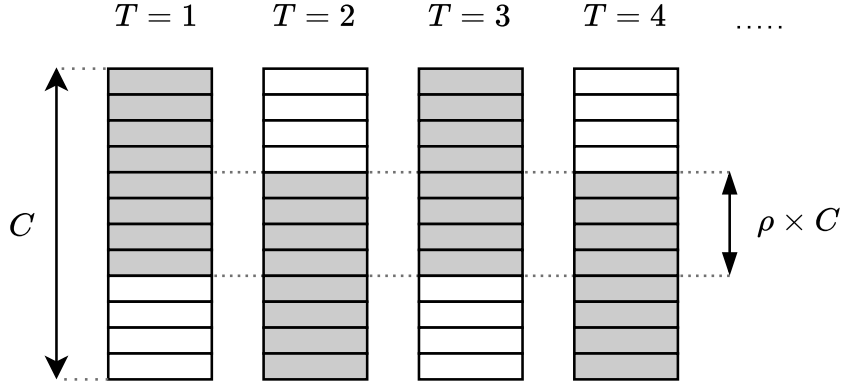


Figure 2.2: Masks provided for the 4 first time steps in a bi-step generator (BG) for a convolution with C output channels. Active channels are greyed. There are only two distinct masks used every other step. The central part which is always active has $\rho \times C$ channels. The generator is denoted ρ -BG.

Contiguous Dealing with latency is deceptively complex, especially when working with GPU hardware. In particular, indexing non-contiguous tensors on GPUs can be quite slow, and in our case, two indexing operations are required first to extract the convolution kernel and then to copy the result into the previous output. While this aspect may seem to be a detail, we empirically observed that it matters significantly. In fact, using non-contiguous masks in some initial experiments increased the latency up to 15% of its initial value. Therefore, to realize the wall clock benefits of using fewer convolutional kernels (via masking), we enforce the masks to be contiguous.

2.3.4 Mask generation

The mask generation process has to respect the constraints mentioned above and create a set of masks that are contiguous and have the same number of active channels. We delegate the generation process to a “generator” object that creates the set of masks and, given a time step, returns the corresponding mask.

In the rest of the chapter, we will use a specific generator called **bi-step generator (BG)** that creates only two different masks for two different time steps. The first mask has a proportion of channels activated from the start, while the second mask has the same proportion of channels activated from the end. This proportion is at least equal to 50%, and can be higher, which creates an overlap that corresponds to channels that are always activated. Using contiguous masks in this manner makes it possible for some channels to be activated at every step while others will be activated at every other step.

We will denote this type of mask generator using the notation ρ -BG, where $\rho \in [0, 1]$ is a ratio that corresponds to the proportion of always active channels, as shown in Figure 2.2. Therefore, a 1-BG will make two full-ones masks (all channels always active), and a 0-BG will make two half-full masks (no commonly active channel).

This bi-step generator design is interesting for two reasons. First, it keeps features “up to date” as any output channel is recomputed at least every other step. Second, the kernel weights used every other step can learn to distinguish recent from old input channels. Other generators with more steps and masks can be used, but respecting the constraints greatly reduces the set of usable masks. We compared the performance between a bi-step generator and a random contiguous mask generator in Section 2.5.

2.4 Experimental Setup and Models

2.4.1 Dataset

The Cityscapes dataset (Cordts et al., 2016) is one of the few dense pixel-level scene semantic segmentation datasets. We picked it as it provides a high number of video sequences and not only isolated frames. Each sequence contains 30 frames, and the 20th frame is annotated with fine pixel-level class labels for 19 object categories. In total, it contains 2975 training, 500 validation, and 1525 testing video sequences.

During training, we use image crops of 700×700 . We perform standard image augmentations with random horizontal flip, random scaling from 0.75 to 1.5, and random Gaussian blur. We apply the same augmentation to all images in a sequence.

Our focus is on future segmentation prediction, *i.e.*, networks are trained to predict a segmentation at $T + 1$ from an image at time T . As we use Cityscapes, we take the 19th frame of the sequence as input, and the network is trained to predict the 20th frame segmentation (for which we have ground truth).

2.4.2 Networks

For our experiments, we apply our channel-wise masked (CWM) convolution to three different networks. We replace every convolution layer with its CWM counterpart, except for the very first and very last convolution, as well as convolutions used for skip connection (when the block’s input and output have different sizes). We use:

- **SwiftNet** (Orsic et al., 2019), a performant light-weight network for real-time

Chapter 2. Convolutions with partial channel update for future video segmentation

segmentation, with a ResNet-18 (He et al., 2016) backbone. We train it using the Adam optimizer with default parameters. We use an initial learning rate of $4e-5$ and a weight decay of $1e-5$, along with a cosine annealing lr-schedule with $\eta_{min} = 1e-6$.

- **SFNet** (Li et al., 2020), the state-of-the-art for real-time segmentation network on Cityscapes, with a ResNet-18 backbone. We train it using the SGD optimizer with a momentum of 0.9, an initial learning rate of $5e-3$, and a weight decay of $5e-4$. We use a poly lr schedule with a power of 0.9.
- **DeepLab V3+** (Chen et al., 2018), an accurate deep segmentation network, with a ResNet-18 backbone. We train it using the SGD optimizer with a momentum of 0.9, an initial learning rate of $5e-2$, and a weight decay of $5e-4$. We use a poly lr schedule with a power of 3.

All hyperparameters mentioned above are taken from respective papers, and we train all networks for 500 epochs with batch size 6. However, we lower original learning rates since we initialize models with pretrained weights from original networks trained to predict the segmentation of the current input.

Unless otherwise specified, we use a ρ -BG (defined in Section 2.3.4) as mask generator in our experiments. Specifically, we conduct most of our experiments using two generators: 0-BG and 0.25-BG.

2.4.3 Slimming networks

Replacing convolutions with their CWM version reduces FLOPs and latency, but hurts accuracy. To assess the advantage of using CWM convolutions, we use as baseline the original network, which has been slimmed in the way presented in (Howard et al., 2017).

This technique consists in thinning a network uniformly at each layer by choosing a width multiplier $\alpha \in (0, 1]$ and multiplying the number of input and output channels in every convolution by α . Original networks are slimmed ahead of time and then trained normally. Such slimming reduces the FLOPs, the latency, and the number of parameters, and hence allows comparing slimmed-network - our baseline - and CWM-network performance at the same computational cost.

In addition, we note that slimming is a simple complementary approach to our method to trade off computation for performance. Therefore, our main experiments combine the two techniques, using the four width-multipliers $\alpha \in \{0.5, 0.65, 0.8, 1\}$.

2.4.4 Evaluation details

As specified in Section 2.3.2, our CWM models perform normal (full) convolutions at the first time step. Therefore, we expect the model’s behavior in the first few steps to be slightly different from its steady-state’s behavior, *i.e.* after the model has processed multiple images.

To have a correct estimation of the model’s steady-state performance, we introduce Asymptotic Behavior Testing (ABT), which consists in feeding the model with inputs from $T - k$ up to $T - 1$, with k high enough for the network to run in steady-state. In our case, we use the highest possible value $k = 19$ permitted by this dataset. Note that ABT is equivalent to a normal evaluation for original models, which do not use the sequential nature of the inputs.

Moreover, our CWM models have a different behavior every other step. Thus, to have a fair estimation of its expected asymptotic performance, we average the mIoU obtained with ABT using $k = 19$ and $k = 18$ in all experiments.

The performance metric used in all our experiments is the mean Intersection over Union (mIoU), computed on Cityscapes’ validation set. All timing measurements are done on a GTX1080 GPU.

2.4.5 Training details

Length of the input sequence Similar to the evaluation process, we also train our model using a sequence of images starting at offset j . This entails feeding the model with inputs from time step $T - j$ to $T - 1$ and utilizing the final prediction for optimization. For our main experiment (Section 2.5.1), we empirically set this offset to $j = 9$. Unless otherwise specified, we use $j = 7$ for our additional experiments, striking a balance between training time and testing performance.

Two-sequences training Since our models have a different behavior every other step, we propose training our models with two input sequences starting at consecutive offsets. Concretely, for each image sequence, we first process and perform an optimization step with the subsequence $\{T - j \dots T - 1\}$, and then again with the subsequence $\{T - j + 1 \dots T - 1\}$.

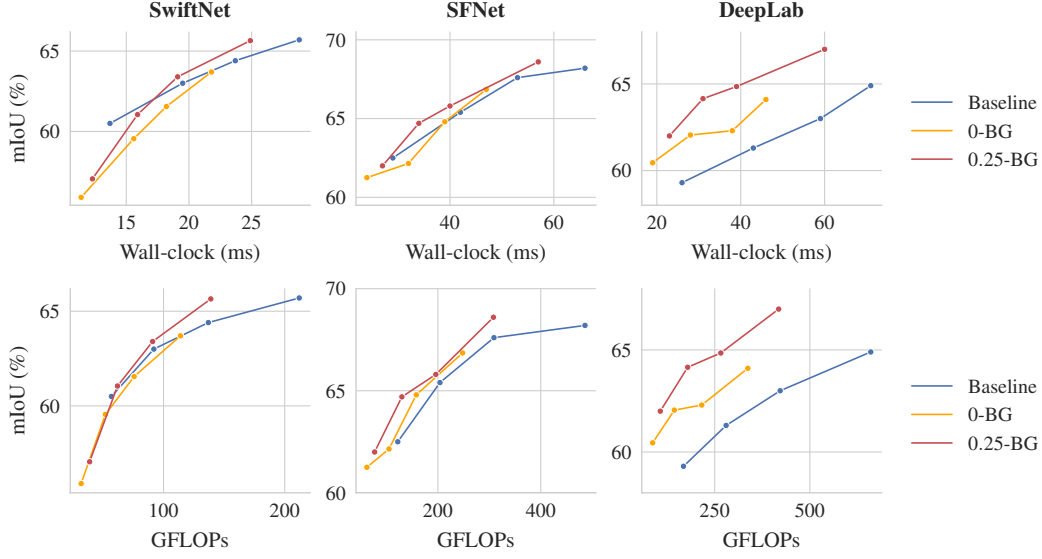


Figure 2.3: mIoU for three different models with and without CWM convolutions. Baseline is the original model, 0-BG and 0.25-BG are our CWM versions of the model using bi-step generators defined in Section 2.3.4. Each line corresponds to a model for which the speed/accuracy trade-off is modulated with the width multipliers $\alpha \in \{0.5, 0.65, 0.8, 1\}$ to slim the network (Section 2.4.3). Our un-slimmed models using 0.25-BG, corresponding to the rightmost red points, all have a better mIoU and a lower latency than the original models.

2.5 Results

In this section, we first evaluate the performance of CWM models compared to original models for future segmentation prediction. Then, we study the relevance and influence of various design choices and hyperparameters. In all experiments, we compute the mIoU with asymptotic behavior testing (presented in Section 2.4.4) on Cityscapes’ validation set. We mostly included here plots of wall-clock times, additional plots of FLOPs are in Appendix.

2.5.1 Future segmentation prediction

In Figure 2.3, we compare baseline models to our CWM versions of those with generators ρ -BG for $\rho \in \{0, 0.25\}$. Each connected line corresponds to a model for which the trade-off between computation and accuracy is controlled by changing the width multipliers $\alpha \in \{0.5, 0.65, 0.8, 1\}$ (Section 2.4.3).

For every line, the rightmost points correspond to the un-slimmed version of the model. On all figures, un-slimmed 0.25-BG models not only use, as expected, less

time and FLOPs but also have a higher mIoU than base models. This is likely because the layers of our CWM-models have access to part of previous feature maps. Having access to both previous and current features allows to more accurately estimate speeds and predict the future position of a moving object.

As we slim more and more with a lower α , the number of active channels per time step gets too small to make proper predictions. This is especially true for SwiftNet and SFNet which are already both very optimized networks. With these models, we see that using the CWM convolutions is better than using the original ones when networks are slightly slimmed. In particular, using un-slimmed CWM models with 0.25-BG decreases wall-clock by 15% and FLOPs by 35% for the same mIoU. When slimming these networks more, we then get quite comparable performances as using CWM convolutions.

For DeepLab, using CWM convolutions leads to much more competitive results. In particular, we see that a slimmed model using a 0.25-BG can reach a similar mIoU as the base model with a decrease of 45% wall-clock and 60% FLOPs. Moreover, the un-slimmed 0.25-BG model (rightmost red dot) has about 2% higher mIoU than the base model while being 15% faster.

Generally, we can see that using CWM-convolutions is a better way to accelerate a model than a simple slimming. These results demonstrate a real potential for CWM convolutions to speed up future prediction networks.

2.5.2 Controlling the speed/accuracy trade-off

When using bi-step generators ρ -BG, the higher the value of ρ , the greater the number of channels processed at each time step and the longer the processing time. In Figure 2.4, we plot the mIoU of CWM-SwiftNet with different generators ρ -BG, for $\rho \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. We use a fixed training offset $j = 9$.

This plot highlights that the parameter ρ allows controlling the trade-off between speed and accuracy, which is a desired and practical feature of our CWM models.

2.5.3 Evaluation Offset

For evaluating our model performance, we use Asymptotic Behavior Testing (ABT), as explained in Section 2.4.4. We remind that ABT consists in feeding the model with inputs from $T - k$ to $T - 1$ before evaluating. We additionally average the mIoU obtained with two consecutive k .

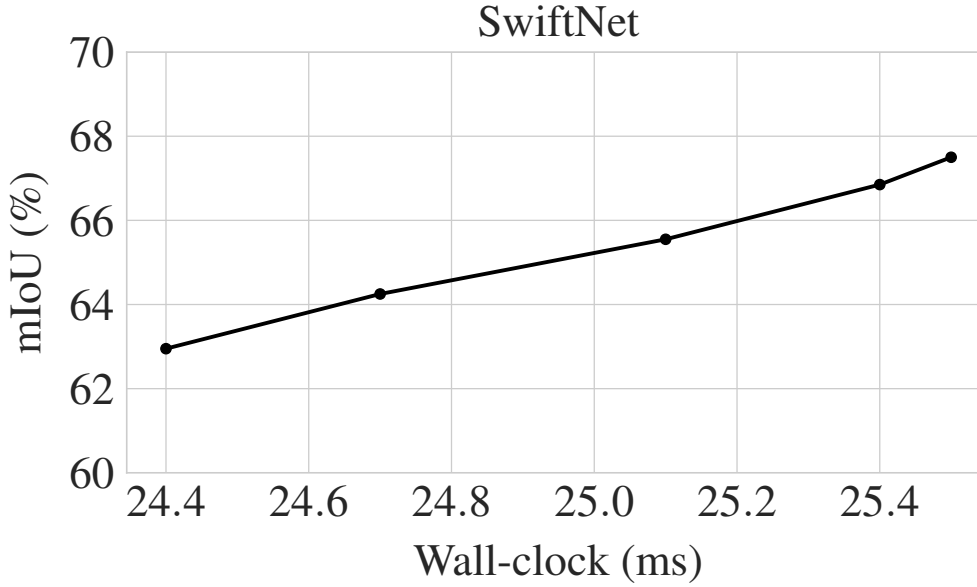


Figure 2.4: mIoU of CWM-SwiftNet trained with different bi-step generators. Each dot represents a CWM-SwiftNet model using a ρ -BG for $\rho \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. While Figure 2.3 picks two values for ρ and varies the width-multiplier α , here we set $\alpha = 1$ and vary ρ . We see that ρ allows to *modulate* the trade-off between speed and accuracy.

Figure 2.5 plots our CWM-SwiftNet’s mIoU depending on the number of frames k it has processed, for all $k \in [3 \dots 19]$, and confirms the relevance of ABT and the averaging.

Indeed, for low k values, the model is not in steady-state, and as k increases, the model performance stabilizes, which is a strong argument for ABT. In addition, the oscillation observed for higher k is coherent with the use of bi-step generators that create two masks used every other step and validates the need for averaging the mIoU computed with two consecutive k .

2.5.4 Training Offset

In CWM models, the output prediction depends on the current input frame and previous computations, which is why we have introduced ABT, an evaluation procedure that evaluates the model’s steady-state performance. Therefore, it seems relevant to train the network when it operates in steady-state. To do so, we feed the model with j inputs from $T - j$ to $T - 1$, as explained in Section 2.4.5.

In this experiment, we plot the mIoU reached by models as a function of the number of frames j used in training. The models used are 2 CWM-SwiftNet models with different mask generators 0-BG and 0.25-BG. The results in Figure 2.6 suggest that

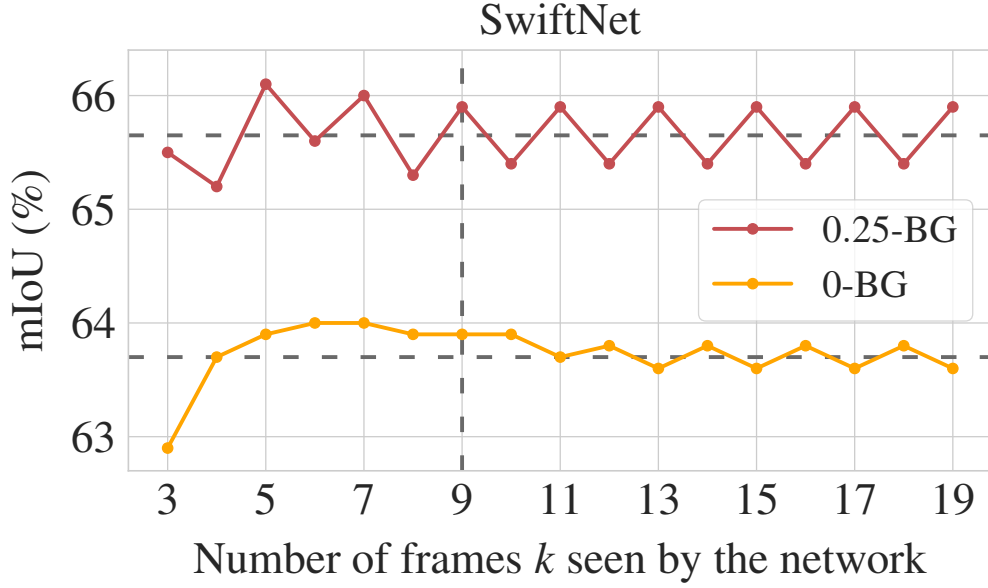


Figure 2.5: mIoU of CWM-SwiftNet as a function of the number of frames k the network has seen. Horizontal dashed lines are values taken as the final mIoU of each network. The vertical dashed line marks the number of frames that networks have been trained with. We see that steady-state is reached after about 9 frames for 0.25-BG and 12 frames for 0-BG. The oscillation for higher k is due to the two alternating masks of bi-step generators.

good steady-state performances start around $j = 7$ and are best around $j = 9$, which is why we used this last value in our main experiments. However, we refrain from using a higher j , which increases training time and does not bring higher mIoU.

2.5.5 Bi-sequence training

All experiments so far have used the “two-sequences training” presented in Section 2.4.5. Table 2.1 reports other multi-sequence training setups with SwiftNet. In particular, it shows the single sequence training where a sequence is only used once with starting offset $T - j$, with $j = 7$ in this experiment. We also trained using 3 and 4 sequences.

The results are shown in Table 2.1, and it appears that two-sequence training is better than single-sequence training (+2.5%). Moreover, using additional offset sequences does not seem to improve the final mIoU, which motivates our choice of two-sequence training for our other experiments.

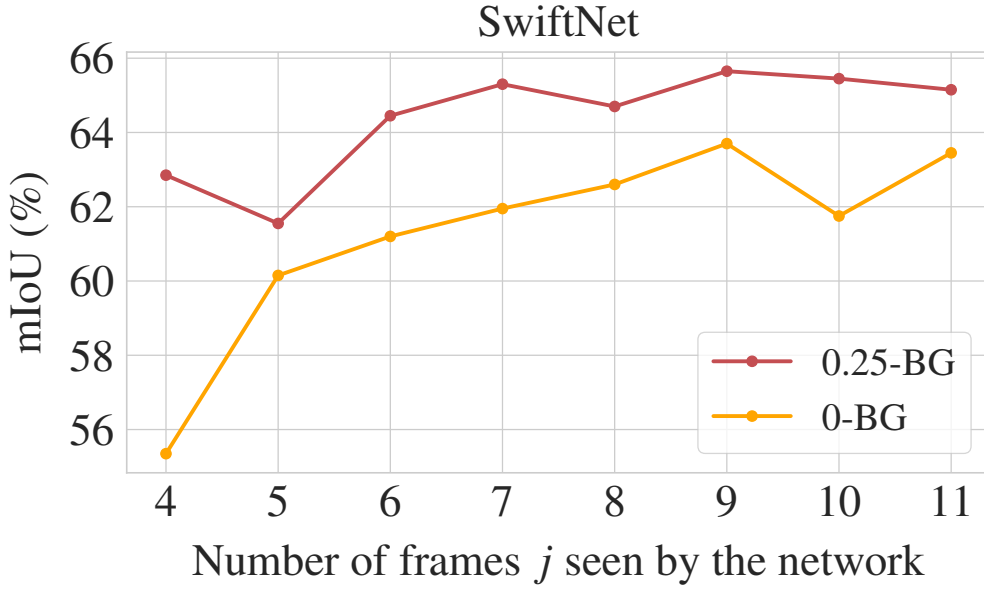


Figure 2.6: mIoU of a CWM-SwiftNet as a function of the length j of the sequence it has been trained with. The mIoU appears to plateau from $j = 7$, and $j = 9$ gives the highest values.

Starting offset of sequences	mIoU (%)
$T - 7$	62.8
$T - 7, T - 6$	65.3
$T - 7, T - 6, T - 5$	64.1
$T - 7, T - 6, T - 5, T - 4$	64.3

Table 2.1: mIoU of a CWM-SwiftNet 0.25-BG with different multi-sequence setup. The first column represents starting offset of sequences. For instance, “ $T - 7, T - 6$ ” is the usual two-sequences training, where one input sequence is used twice to optimize the model. It is used first as the subsequence $[T - 7 \dots T - 1]$, then as the subsequence $[T - 6 \dots T - 1]$. We see that two-sequence is better than single-sequence and that using more sequences does not help.

2.5.6 Ablation on CWM convolutions position

When altering a network to use our CWM convolutions, we specified in Section 2.4.2 that we do not replace the very first convolution in the stem and those used in skip connections.

In Table 2.2, we study the performance in configurations where we use CWM convolutions instead of normal ones in the stem layer or in skip connections. The model is a CWM-SwiftNet using a 0.25-BG. This table confirms that we should use normal convolutions in the stem and skip connections as it brings a substantial mIoU increase.

Stem	Skip	mIoU (%)
CWM	CWM	63.3
CWM	Standard	64
Standard	CWM	65
Standard	Standard	65.3

Table 2.2: Ablation experiment on CWM convolutions in the stem and skip connections of SwiftNet. CWM indicates a CWM convolution, Standard indicates normal convolution. This confirms that we should use normal convolutions in the stem and skip connections.

2.5.7 Bi-step vs Random generator

To evaluate the relevance of bi-step generators, that create two masks alternated every other step, we propose to compare their performance with those of random mask generators which randomly select a contiguous subset of channels at each step.

We trained both generators so that they have the same amount of activated channels per time step. For a bi-step generator 0-BG, there are 50% of channels activated per step, for 0.25-BG it is 62%, and for 0.5-BG it is 75%.

Results are shown in Table 2.3 below. These numbers suggest that the fixed update schedule of bi-step generator is better than a random update rule, but that the difference reduces as the percentage of active channels increases.

2.5.8 Performance on T+1, T+2, T+3

In this additional experiment, we study the mIoU decay for SwiftNet models when they are trained for a later objective, namely for $T + 2$ or $T + 3$ instead of $T + 1$. The models used are the SwiftNet baseline and 2 CWM-SwiftNet models with different

Chapter 2. Convolutions with partial channel update for future video segmentation

Channels activated each step	Bi-step Generator	Random Generator
50 %	62 %	50.8 %
62 %	65.3 %	61.7 %
75 %	67.5 %	66.2 %

Table 2.3: Comparison between bi-step and random generator for the same amount of activated channels at each step.

mask generators 0-BG and 0.25-BG. We report results on Figure 2.7, on which we can see that the slope of the decay appears higher for non-CWM models, which do not have access to past information.

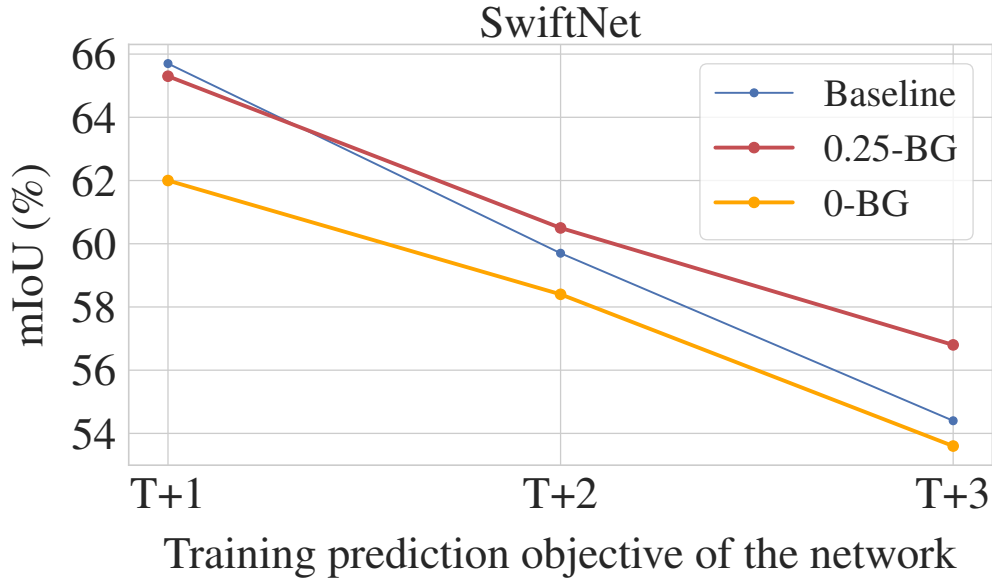


Figure 2.7: Models mIoU depending on the prediction objectives. The mIoU worsens as we try to predict further into the future, but the slope of the decay is higher for the baseline than for CWM-models.

2.6 Conclusion

In this chapter, we delved into the design of anticipatory networks by exploring a possible improvement of convolution layers that makes them more suitable for future frame segmentation. Leveraging the temporal coherence of successive frames, our simple layer design both trims computational requirements and enables access to previously calculated features, which speeds up and improves future segmentation prediction.

Our proposed CWM layer is not restricted to segmentation networks and can be applied to any task involving prediction from a correlated input sequence. Moreover, the ρ parameter of our ρ -BG generators allows for the modulation of the speed/accuracy trade-off. Experimentally, our proposed CWM-models achieve the same mIoU while performing fewer FLOPs, and taking less wall clock time than the original models in most cases. Better yet, un-slimmed CWM-models even achieve higher mIoU than the originals.

This chapter proposed an approach to mitigate network latency through the reduction of the number of parameters effectively used at each run. Moving forward, the upcoming chapter presents another strategy for achieving faster processing, shifting focus from minimizing parameter utilization toward reducing the volume of data processed by the model.

3 PAUMER: Patch Pausing Transformer for Semantic Segmentation

This chapter is based on

Courdier, E., Sivaprasad, P. T., and Fleuret, F. (2022). Paumer: Patch pausing transformer for semantic segmentation. In *33th British Machine Vision Conference 2022, London, UK, 21 - 24 November 2022*

3.1 Introduction

In the previous chapter, we looked at a way to make the model faster by skipping some parts of the convolutions in the network, effectively reducing the number of parameters employed. In this chapter, we delve into an alternative approach to accelerate networks that involves reducing the amount of overall data processed. More specifically, we propose to progressively reduce the amount of data processed by the segmentation network as we go deeper into the network. The latency-reduction method proposed in this chapter is specifically tailored to the Vision Transformers (Dosovitskiy et al., 2021; Steiner et al., 2021) class of networks. Vision Transformers (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 (Vaswani et al., 2017). Although originally developed for image classification, recent methods have adapted transformer architectures to various computer vision tasks (Khan et al., 2021; Wang et al., 2021; Chu et al., 2021), and specifically to semantic segmentation (Zheng et al., 2021; Strudel et al., 2021; Xie et al., 2021).

Chapter 3. PAUMER: Patch Pausing Transformer for Semantic Segmentation

While these large transformer architectures have led 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 (Tay et al., 2020). One way to achieve this is to reduce the number of processed patches. Some works use multiscale approaches (Wang et al., 2021; Xie et al., 2021) 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 (Yin et al., 2022; Pan et al., 2021; Marin et al., 2022; Rao et al., 2021; Liang et al., 2022a). 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.

In this chapter, 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 (Cordts et al., 2016) in Figure 3.1, 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 feeding them as-is to the decoder to produce the final segmentation map. Our argument is supported by the findings in Raghu et al. (2021); they find that the representations of tokens are primarily modified in the first half of the network, and rely 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, 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 (Strudel et al., 2021) architecture, we show the performance of our method on the standard benchmark suite of ADE20K (Zhou et al., 2017) and Cityscapes (Cordts et al., 2016). 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 over Union (mIoU) 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.

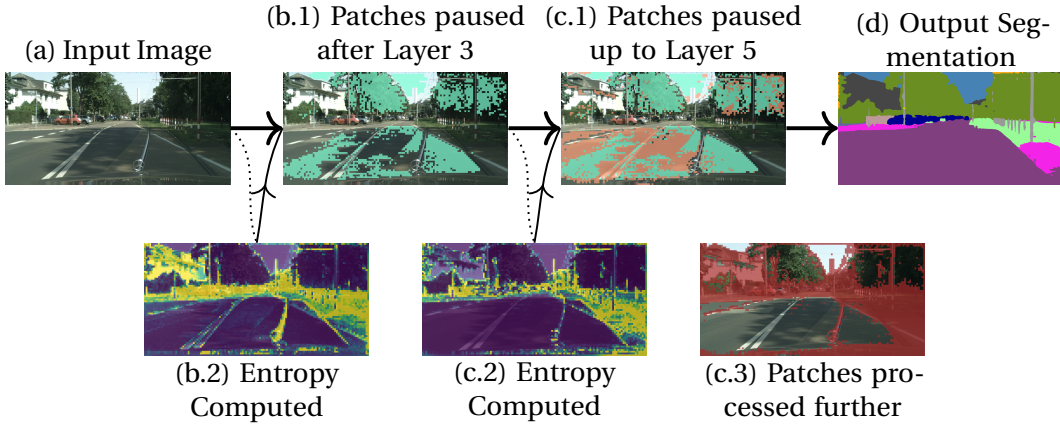


Figure 3.1: Illustration of our proposed method. Our method progressively stops processing patches after they reach a low enough prediction entropy. First column (a) shows the input image. Second column (b.1) shows the patches that are stopped from being processed after the third transformer layer in **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 3.2.1). 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 3.2.

This chapter is organized as follows:

- In Section 3.2, we describe our newly proposed method, PAUMER. We first describe the motivation behind our method, then provide a justification for our patch-pausing criterion, and finally describe the training procedure and its advantages.
- In Section 3.3, we review related existing literature. In particular, methods that perform token sparsification, early exit, and dynamic computation time are discussed.
- Section 3.4 presents the experimental results we achieved with PAUMER on the Cityscapes and ADE20K datasets. Further, we provide context into how PAUMER stands compared to other established rapid segmentation methods.
- In Section 3.5, we discuss the results obtained in the previous section and the variations observed between different datasets. We also acknowledge the limitations of our work, shedding light on several design choices and their implications.

3.2 Patch pausing transformer for Semantic Segmentation

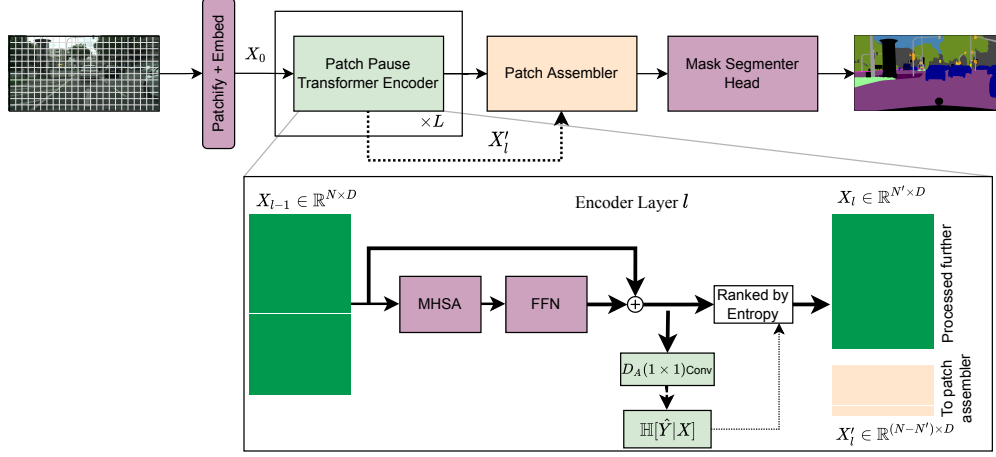


Figure 3.2: Schematic of our proposed method. We modify Segmenter (Strudel et al., 2021) to enable the pausing of patches and feeding them directly to the decoder. Our proposed PAUMER encoder adds a simple auxiliary decoder (a 1×1 convolution), and uses the predicted posterior entropy $\mathbb{H}[\hat{Y}|X]$ 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 (Lavin and Gray, 2016), or require architectural simplifications like the use of 1×1 convolutions in the network (Verelst and Tuytelaars, 2020). 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 input’s 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 from Strudel et al. (2021), 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 Section 3.3.C.

3.2.1 Using Entropy as a criterion for patch-pausing

How do we determine which tokens to pause, *i.e.* which ones 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 3.3 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.

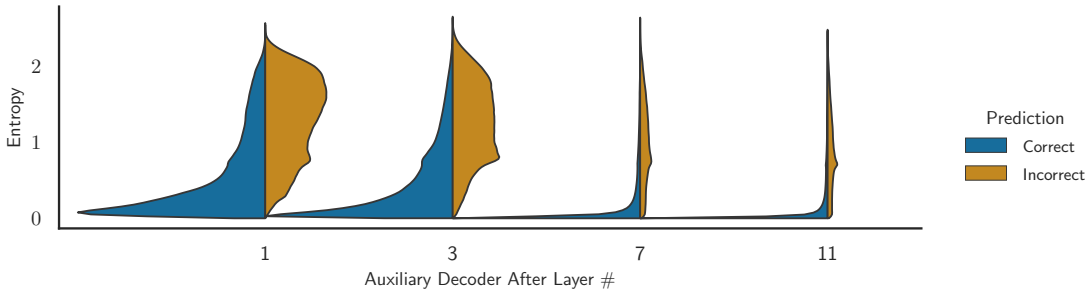


Figure 3.3: Violin plot of entropy of predictions at intermediate layers. In this figure, we plot the entropy distribution of the auxiliary predictions for 10% of images in the Cityscapes validation set. The x-axis marks after which layer the prediction was done. For each layer, the entropy distribution 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)

3.2.2 Training PAUMER - One training for many pause configurations

We base our network on Segmenter’s architecture (Strudel et al., 2021) detailed in Section 3.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

Chapter 3. PAUMER: Patch Pausing Transformer for Semantic Segmentation

test the same patch pausing configuration that satisfies our run-time requirements. Any changes to the run-time requirements require 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 any layer l (see Figure 3.2).

We choose to limit the number of locations where we pause patches, as each decoding incurs an additional cost of passing the representation through a linear layer (see Section 3.D for additional analysis).

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

$$\mathcal{L}_{\text{main}} = \text{CE}(\mathbf{y}, \hat{\mathbf{y}}) \quad (3.1)$$

$$\mathcal{L}_{\text{aux}}^l = \text{CE}(\mathbf{y}, \hat{\mathbf{y}}^l) \quad (3.2)$$

where \mathbf{y} is the ground truth, $\hat{\mathbf{y}}$ refers to the logits predicted by the main decoder (mask transformer), and $\hat{\mathbf{y}}^l$ is the auxiliary decoder’s output at the l^{th} layer. The total loss used to train is a combination of losses in Equations (3.1) and (3.2).

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{main}} + \lambda \mathcal{L}_{\text{aux}}^l \quad (3.3)$$

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

$$H_l := \mathbb{H}[\hat{Y}_l | X_l] = \mathbb{H}[\sigma(D_A(X_l))], \quad (3.4)$$

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 3.1). 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 pro-

cessed and the ones that have been paused, we use a patch assembler module (see Section 3.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, the 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 grants us the advantageous ability to choose *at run-time* a pausing configuration that is tailored to 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 3.1. 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.

3.3 Related Work

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

3.3.1 Segmentation using Transformers

Transformers that were originally proposed for language processing tasks (Vaswani et al., 2017) have been incorporated into vision (Dosovitskiy et al., 2021; Touvron et al., 2021) and several improvements have been proposed (Khan et al., 2021). SETR (Zheng et al., 2021) adapted the standard vision transformer (ViT) to segmentation by using a multiscale decoder on all the image patches. Segmenter (Strudel et al., 2021) improved the decoder design by using a learnable per-class token that acts as a weighting mechanism over the tokens' representations. Segformer (Xie et al., 2021) 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 an impact on the downstream segmentation performance (Wang et al., 2021; Xu et al., 2021; Chu et al., 2021; Liu et al., 2021). 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.

3.3.2 Token sparsification methods

Several components of the whole transformer architecture have been improved, by approximations, and simplifications to the attention mechanism. Interested readers can refer to [Tay et al. \(2020\)](#). 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 \(2016\)](#) 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. \(2017\)](#), which dynamically decides to apply a differential number of residual units to different parts of the input. This has been adapted to transformers too ([Yin et al., 2022](#)), where the tokens are progressively halted as they are determined to have been processed enough according to a similar criterion as ACT. DynConv ([Verelst and Tuytelaars, 2020](#)) uses an auxiliary network to predict pixel masks which indicate pixels of the image that are not processed by a residual block. DynamicViT ([Rao et al., 2021](#)) 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 ([Jang et al., 2016](#)) in both these methods. We consider the simplicity of the steps the strength of our proposed method. Unlike DynamicViT ([Rao et al., 2021](#)), 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.

3.3.3 Early-exit methods

Dynamic neural networks ([Han et al., 2021](#)) 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 ([Kaya et al., 2019](#)). Several methods to determine when to exit the network have been proposed. Branchynet ([Teerapittayanon et al., 2016](#)) and Shallow-Deep nets ([Kaya et al., 2019](#)) uses auxiliary classifiers to predict the output class for vision convolutional networks and stop 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. \(2020\)](#) extends this by using a patience parameter that tracks the number of auxiliary

classifiers that predict the same class. DeeBERT (Xin et al., 2020) proposes a two-stage training, where the auxiliary decoders are trained after the main network is trained and frozen. Li et al. (2017) 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 effort. Our method is an early exit strategy, specifically for the case of segmentation transformers. While similar methods have been examined in the literature, to the best of our knowledge, we are the first to perform patch-pausing in the context of semantic segmentation. Also, the randomized training presented in Section 3.2.2 has not been used in this context, though similar ideas to reduce network width were studied in slimmable networks (Yu et al., 2018b).

3.4 Experiments

3.4.1 Datasets and Evaluation

We show the performance of our method using networks trained on Cityscapes (Cordts et al., 2016) and ADE20K (Zhou et al., 2017). 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. (2021), 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

Chapter 3. PAUMER: Patch Pausing Transformer for Semantic Segmentation

methods in Section 3.3 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 3.1: 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 3.2.2. Each configuration here corresponds to a marker in Figures 3.4a, 3.4b and 3.K.1.

Training hyperparameters and Inference Configurations: For our main experiments, we use Segmenter with ViT-Ti and ViT-S backbones (details in Section 3.B). During training, we follow the procedure in Section 3.2.2 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. The rest of the hyperparameters as kept the same as in Strudel et al. (2021). We implement our method using MMSegmentation (Contributors, 2020) and use their pretrained models whenever available. At inference, we test the networks with the pause configurations in Table 3.1. This list of pause configurations is not exhaustive and does not hold any specific importance, but it has 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 3.4a and 3.4b). 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 3.H.1 and 3.H.2) 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 3.H.1) and we pick the first layer and proportion. We fix this layer and ratio, then sweep through pausing configurations at a second layer (Figure 3.H.2). We repeat this procedure until adding more layers is no longer beneficial.

3.4.2 Results

Performance analysis In Figure 3.4a, 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.

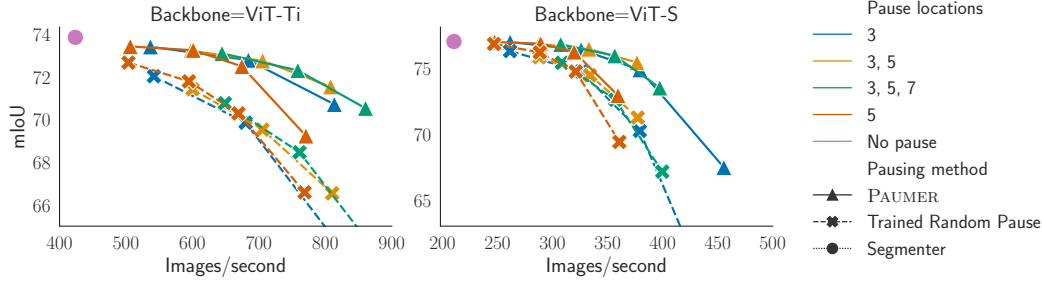
We show results for ViT-Ti and ViT-S for ADE20K in Figure 3.4b. For ViT-Ti backbone, we see that for a 50% increase in IMPS, mIoU drops by about 2.8%, and a 100% increase in IMPS with a mIoU drop of about 10.7%, compared to random pausing performance of 5.4% and 13.5%. The drop in mIoU is in contrast with the results of Cityscapes, where we could achieve a 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.

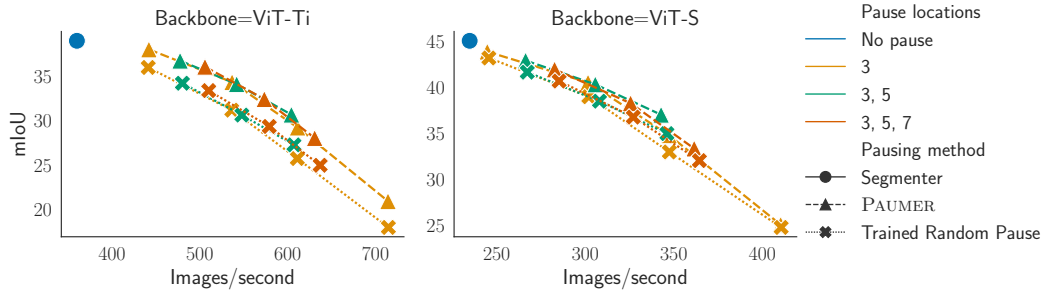
Comparison to other architectures

In Figure 3.5, we compare our method to a broad array of architectures for which pretrained models are available in MMSegmentation (Contributors, 2020). 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

Chapter 3. PAUMER: Patch Pausing Transformer for Semantic Segmentation



(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 3.4: Results on Cityscapes and ADE20K for our proposed method PAUMER. Each marker is a configuration from Table 3.1. 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.

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.

3.5 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 (Xie et al., 2021) 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.

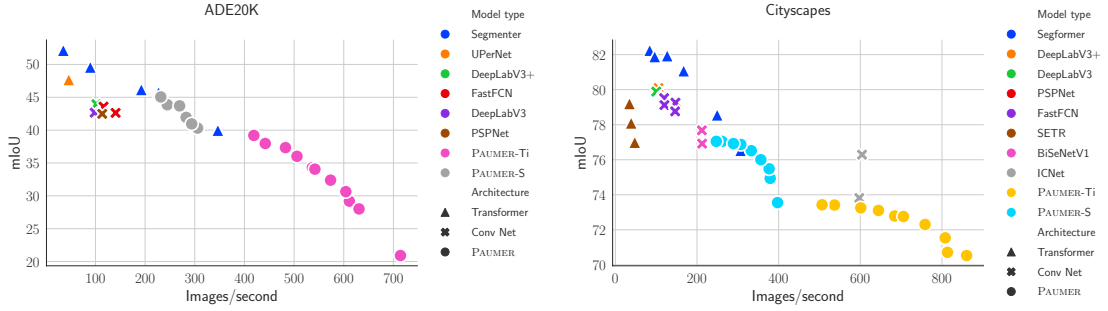


Figure 3.5: Performance comparison for ADE20K and Cityscapes. We compare to pretrained models available in MMSegmentation (Contributors, 2020) for each of these datasets. Architectures devised for speed or accuracy outperform us on those criteria, but PAUMER has the unique advantage that we can trade off one for the other using a tunable hyperparameter. Here, we show the Pareto front of PAUMER. We use different colors for different architectures and different shapes for architectural families.

In addition to architecture dependence, patch pausing’s performance may be 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 (Ethayarajh et al., 2022) 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.

While our method adaptively selects which patches to pause based on the input, it always chooses a fixed proportion of them. This design is specifically implemented to leverage batch-level parallelism on GPUs. Choosing the number of patches depending on the input batch has not been dealt with in this research.

To study the stability of our training procedure, we study the variability between training runs by training both Segmenter and our proposed PAUMER with ViT-Ti backbone on the Cityscapes dataset over three random seeds. Segmenter’s mIoU has a standard deviation of 0.37, whereas for PAUMER it depends on the amount of pausing. When pausing 40% at layer 3, the variance is 0.35, whereas when pausing 40% at layers

Chapter 3. PAUMER: Patch Pausing Transformer for Semantic Segmentation

3,5 and 7, it raises to 0.75. It is apparent that pausing more tokens results in a higher variance of performance. We hypothesize this is attributable to instabilities in patch selection early on in the training process, and thus some training runs happen to perform better. This, however, needs additional validation.

3.6 Conclusion

Both this chapter and the previous one propose a practical method for efficiency gains, not only theoretical ones. The efficiency-driven modifications are motivated by how operations are scheduled and memory adjacency requirements. This is why it is beneficial to skip the computation of some feature maps in a convolutional network and some spatial locations in a vision transformer. In this chapter, we shifted focus from convolutional to transformer networks, and within this context proposed a token-reduction technique for faster inference. Our method, PAUMER, is a first step in the direction of post-hoc design for efficient inference in semantic segmentation transformers.

PAUMER increases the inference speed by applying dissimilar amounts of computation to different patches of an image, judiciously pausing the processing of patches with an estimated low complexity. We have seen that the entropy of an auxiliary intermediate predictor can be a reliable proxy for the patch complexity and thus serves as a useful criterion for patch pausing. In addition, our method offers the flexibility to pick different pause configurations without having to retrain the network, which allows the model to run at the desired throughput. On Cityscapes, we have shown that our method can double the inference speed with little loss of mIoU.

With this chapter, we conclude the part of this research focusing on reducing network latency, whether in the context of swift convolutional networks (Chapter 2) or precise transformer networks (Chapter 3). The next and final chapter will look at another critical aspect of anticipatory networks: the predictions' ambiguity.

Appendix - Chapter 3

3.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.

3.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 3.B.1, 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 3.B.1: ViT architectural details used

3.C Brief introduction to Segmenter

Segmenter (Strudel et al., 2021) 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 $E: \mathbb{R}^{3P^2} \rightarrow \mathbb{R}^D$, giving a feature representation $X_0 \in \mathbb{R}^{N \times D}$, as shown in Figure 3.2.

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:

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

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.

Chapter 3. PAUMER: Patch Pausing Transformer for Semantic Segmentation

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  $\tau$ , 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  $B \times N' \times D$ .  
  
    # 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
```

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.

3.D Patch pausing's limitations

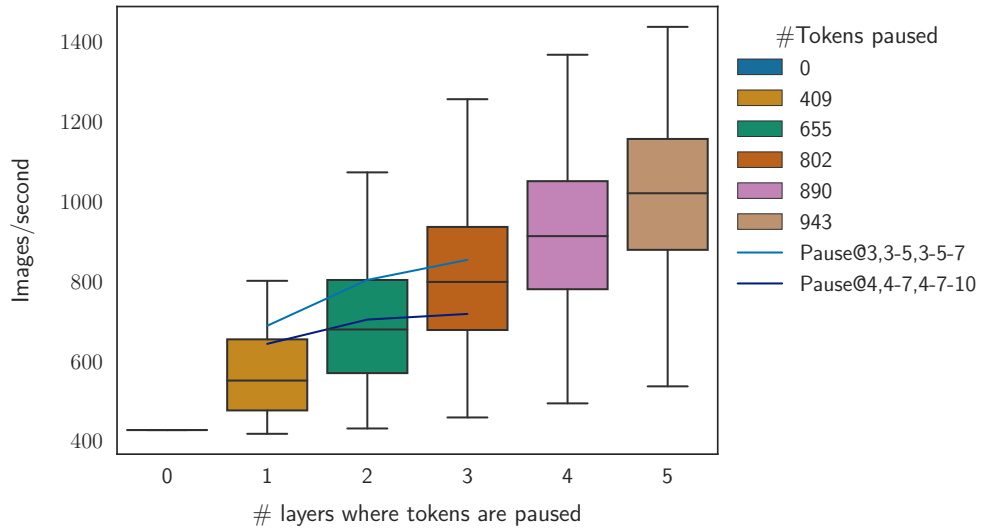


Figure 3.D.1: Evolution of the throughput with the number of layers using patch pausing.

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 Section 3.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 throughput computed in images processed per second. To simplify the analysis, we assume that we pause a fixed proportion $\tau = 0.4$.

Chapter 3. PAUMER: Patch Pausing Transformer for Semantic Segmentation

In Figure 3.D.1, 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 3.D.1 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). These two configurations are plotted as lines on Figure 3.D.1. 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 3.4a and 3.4b). Thus the primary limitation is posed by the drop in mIoU rather than throughput.

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

In Figure 3.E.1, 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 3.2.2. 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.

3.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 3.4a. This is to illustrate that we can choose a pause configuration that fits our run-time requirements (IMPS), and that it works

3.G Influence of the auxiliary loss weight λ

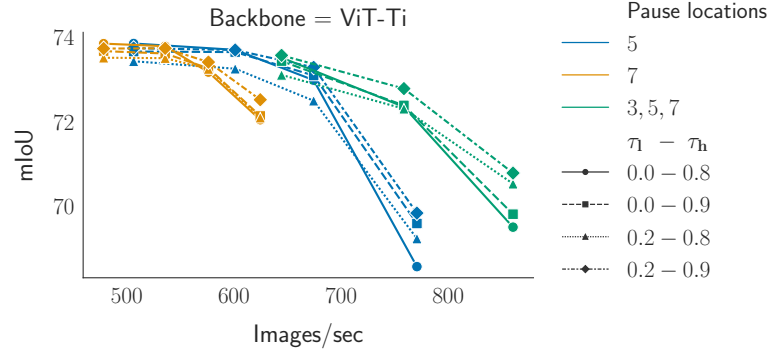


Figure 3.E.1: mIoU vs throughput for different values of the range of the pause ratio $\tau_l - \tau_h$ introduced in Section 3.2.2.

with the performance specified here.

Backbone									
	Images / second	252	276	300	325	349	373	397	421
ViT-S 210 im/s	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
	Images / second	508	557	605	654	702	751	799	847
ViT-Ti 424 im/s	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.F.1: Trading off mIoU for speed. In Section 3.4.2, 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.

3.G Influence of the auxiliary loss weight λ

In Figure 3.G.1, 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 3.2.2. 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.

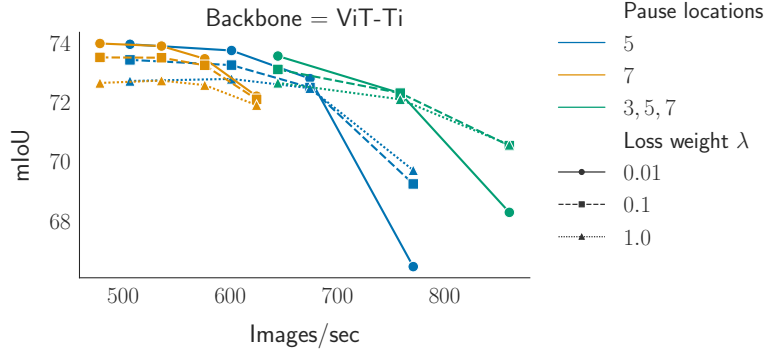


Figure 3.G.1: mIoU vs throughput for different values of the auxiliary loss weight λ introduced in Section 3.2.2. Lower values of λ lead to better performance when pausing few tokens but worse performance when pausing more, and conversely for higher values of λ . Our chosen value of $\lambda = 0.1$ is a trade-off that can also be modified depending on the use-case.

3.H Interplay of Pause location and Pausing proportion

τ

We showed the results for some configurations in Table 3.1. In this section, we study the interplay between pause location and pause proportion τ . In Figure 3.H.1, we show a sweep over pause configurations. 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 runtime, one can choose a pause location and pause proportion that gives the required performance.

3.I Using Early Exit at test time

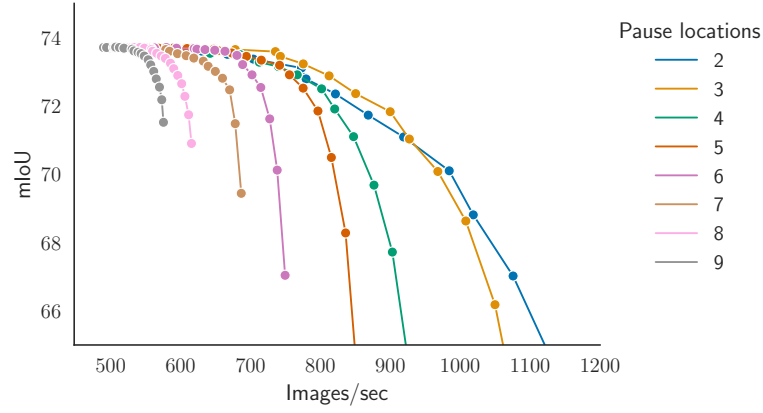


Figure 3.H.1: Pausing once at various layers for various pause proportions for ViT-Ti on Cityscapes. We see that the highest gains in IMPS are achieved by dropping in earlier layers.

To examine this further, we plot the performance of pausing twice in Figure 3.H.2. 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 3.H.2. 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.

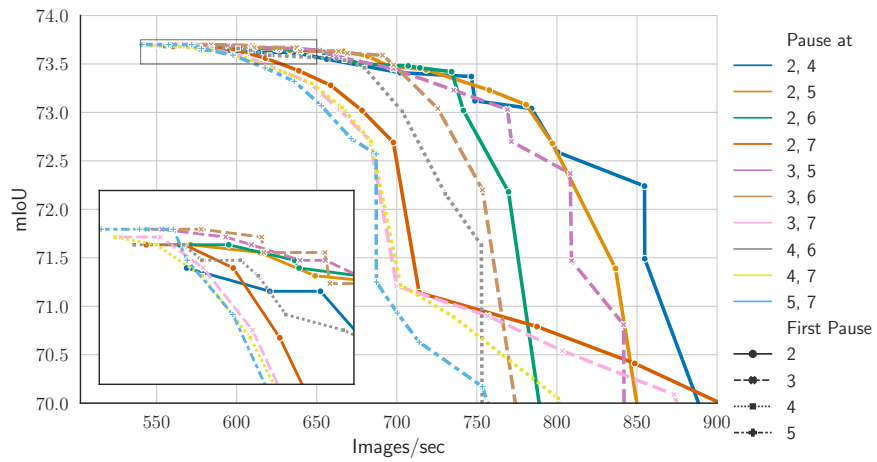


Figure 3.H.2: Pareto front of pausing at two layers for ViT-Ti on Cityscapes.

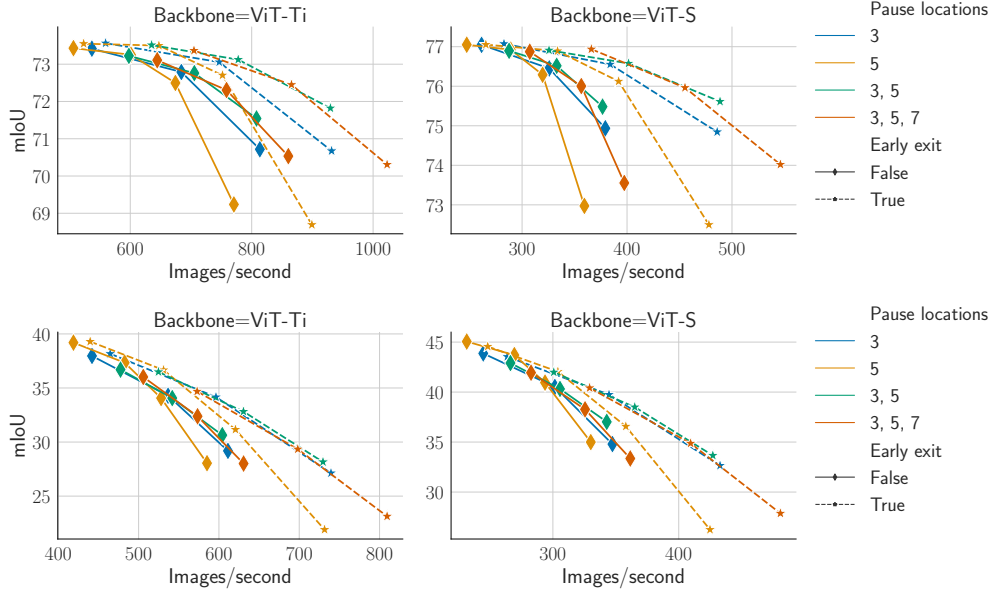
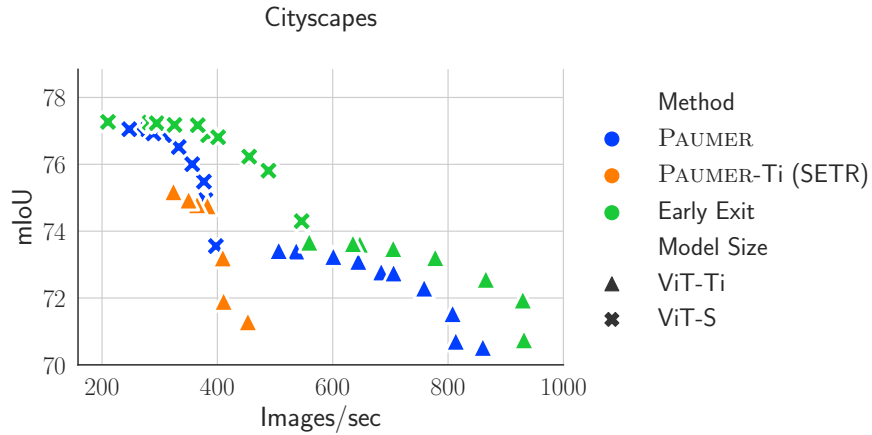


Figure 3.I.1: Comparison when segmenting with and without early exit at test time on CityScapes and ADE20K.

3.I Using Early Exit at test time

In Figure 3.I.1, we study the use of early exit on a trained PAUMER. Early exit (see works in Section 3.3) 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 3.4a, 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 3.I.1, 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.

3.J Comparing SETR, Segmenter, EarlyExit using Segmenter



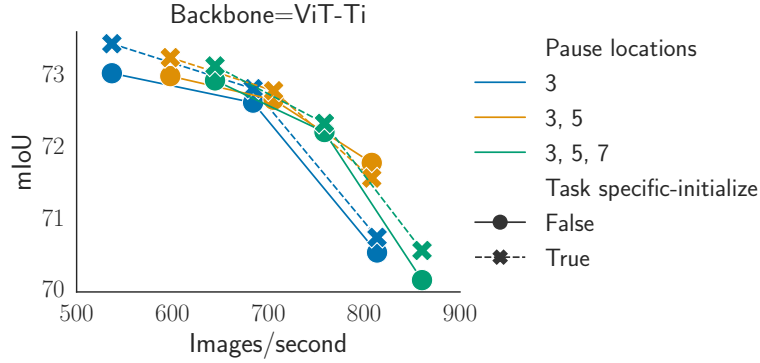


Figure 3.K.1: Importance of initialization for PAUMER. Task specific initialization benefits performances. We train ViT-Ti PAUMER: solid lines are from ImageNet pretrained backbones, dashed lines are from Cityscapes pretrained Segmenters. Each marker is a configuration from Table 3.1.

3.L Entropy as a measure of patch-pausing

In Section 3.2.1, we argued that entropy is a reasonable indicator of completion of processing. For that, we used the illustration in Figure 3.3 to show the increase in separation of entropy histograms for pixels predicted correctly and incorrectly. We expand that in Figure 3.L.1, to analysing that to each class individually. The larger separation in entropy in the first few layers of network is prevalent in large classes like road, building, vegetation, car. As seen in Figure 3.1 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).

3.L Entropy as a measure of patch-pausing

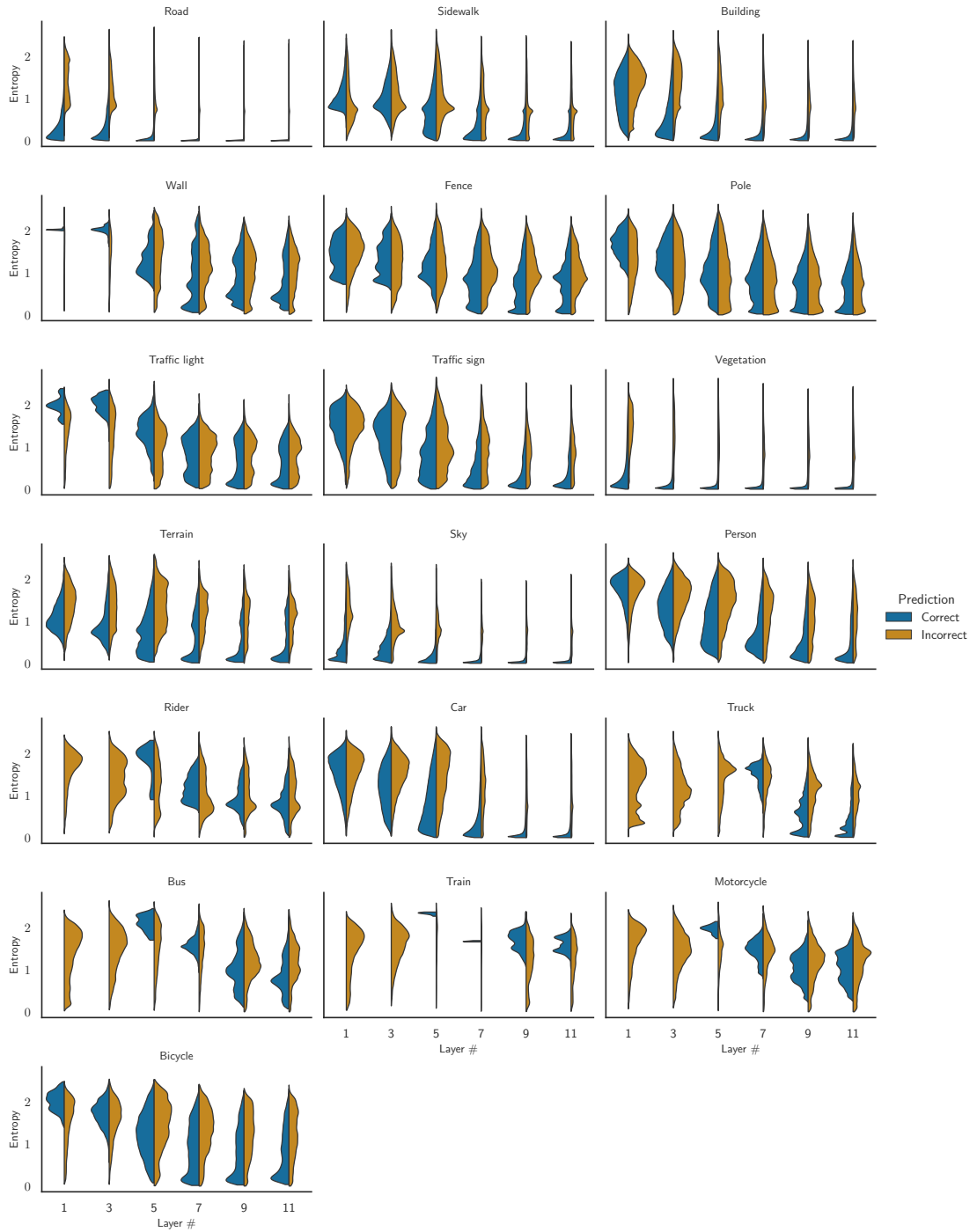


Figure 3.L.1: Entropy per layer for each class of Cityscapes. Continues Figure 3.3.

4 Handling ambiguous contexts: An application of Discrete Diffusion model to Segmentation

This chapter is based on

Courdier, E., Katharopoulos, A., and Fleuret, F. (2023). Segmenting the unknown: Discrete diffusion models for non-deterministic segmentation. Submitted to the *International Conference on Learning Representations (ICLR)*.

4.1 Introduction

In this last chapter, we propose to investigate another aspect of anticipatory networks, specifically the inherent non-deterministic nature of future predictions. While the short-term future can be deemed as nearly deterministic, the long-term future prediction task holds a substantial degree of ambiguity. In autonomous driving applications, for example, we need to model the decision-making processes of drivers and pedestrians to predict their positions in the future.

As we delved further into computer vision models capable of handling ambiguity and uncertainty, we observed that other real-world applications also require such systems. For example, in medical imaging physicians oftentimes do not agree on a single diagnosis, hence we desire models capable of providing multiple plausible predictions. However, the majority of models are still trained in a deterministic manner, meaning they provide only one prediction for each input. As a result, they can provide only one possible output for a given input, regardless of inherent uncertainty. Moreover, these models often exhibit high confidence in their predictions even when they are incorrect.

In this chapter, we investigate the use of discrete diffusion models ([Austin et al., 2021](#))

Chapter 4. Handling ambiguous contexts: An application of Discrete Diffusion model to Segmentation

for handling ambiguous predictions in semantic segmentation. In particular, we tackle two non-deterministic tasks: future prediction and medical image segmentation. Existing works in medical image segmentation (Kohl et al., 2019) utilize generative models such as VAEs to model the uncertainty of the prediction. On the other hand, although future segmentation is also inherently ambiguous, existing works (Lin et al., 2021) approach it as a deterministic task. Conversely, we treat both problems in a unified way, namely as a conditional generative task. Specifically, for the case of medical image segmentation, we generate a segmentation mask given the input image, whereas for future segmentation we generate the next segmentation mask given the previous ones. Since diffusion models have demonstrated great capabilities in efficiently generating highly detailed and high-resolution samples (Rombach et al., 2022), we opt to use them as our generative model. In particular, we utilize the discrete diffusion framework of Austin et al. (2021), which dovetails perfectly with the discrete nature of the predicted segmentation masks.

The overall organization of the contributions of this chapter are the following: After reviewing relevant literature in Section 4.2, we present our method in Section 4.3, where we propose to model uncertainty of predictions using discrete diffusion models in semantic segmentation. Moreover, for the case of future forecasting, we introduce an autoregressive diffusion framework, that utilizes segmentation generations as inputs to predict further into the future. Then, in Section 4.4, we evaluate our model on a Lung Cancer Medical Imaging Dataset (LIDC) (Armato III et al., 2011) and on two future prediction tasks. The first is a car intersection simulator where we want to predict the path of the cars in the intersections, and the second is the future segmentation task in the Cityscapes dataset (Cordts et al., 2016). From our experimental evaluation, we show that our model consistently outperforms an equivalent deterministic model in all tasks. Moreover, for the two real-world tasks, we demonstrate that our generative framework outperforms existing VAE-based methods on LIDC while performing on par with the state-of-the-art on Cityscapes’ future prediction.

This chapter is organized as follows:

- In Section 4.2, we review the related work on non-deterministic segmentation, diffusion models, and future segmentation.
- In Section 4.3, we present our method for handling ambiguous segmentation using discrete diffusion models. We review diffusion models and introduce the discrete diffusion framework. Then, we present the specific changes we made to adapt it to the segmentation task. Finally, we present a toy experiment to illustrate the need for modeling uncertainty in segmentation.
- In Section 4.4, we evaluate our method on three different tasks: medical image

segmentation, trajectory prediction on a car intersection simulator, and future segmentation on the Cityscapes dataset.

4.2 Related Work

Non-deterministic segmentation As precursors of ambiguous segmentation, [Kohl et al. \(2018\)](#) proposed the Probabilistic U-Net and later its hierarchical version ([Kohl et al., 2019](#)). It sparked further works from [Baumgartner et al. \(2019\)](#); [Bhat et al. \(2022\)](#), among others. These works are all centered on the use of variants of a conditional variational auto-encoder to model the ambiguity. In contrast, we propose to use diffusion models, similar to the recent work by [Rahman et al. \(2023\)](#). We differ in that we use discrete diffusion, and extend the study to future segmentation prediction.

Diffusion and Discrete Diffusion Our approach towards handling ambiguity is to use generative models and be able to sample possible outcomes. In this chapter, we investigate Diffusion Probabilistic Models (DPM), which are generative models first formalized by [Sohl-Dickstein et al. \(2015\)](#) and later successfully applied to high-quality image synthesis by [Ho et al. \(2020\)](#). We more specifically base ourselves on Discrete Denoising Diffusion Probabilistic Models (D3PMs) introduced by [Austin et al. \(2021\)](#). We found D3PMs to offer substantial flexibility regarding the modeling of the corruption process, and to be particularly well suited to the discrete nature of the segmentation task.

Diffusion in Segmentation Since diffusion gained strong popularity, numerous works have tried applying DPMs to segmentation. Most of them used it in the context of medical image segmentation ([Wu et al., 2022, 2023](#); [Wolleb et al., 2022b,a](#); [Pinaya et al., 2022](#)), but contrary to us none of these aimed to solve the inherent ambiguity. Only the previously mentioned [Rahman et al. \(2023\)](#) is leveraging diffusion for handling the ambiguity of medical images.

Few works have applied it to other domains: [Amit et al. \(2021\)](#) applies DPMs to the interactive segmentation task, which is a restricted two-class problem. [Chen et al. \(2022a\)](#) uses another form of discrete DPMs ([Chen et al., 2022b](#)) to perform panoptic segmentation of full images and videos. Similarly to us, they introduce an autoregressive scheme but do not tackle any ambiguous tasks

Future segmentation On the future segmentation forecasting task, different approaches exist. Direct semantic forecasting introduced by [Luc et al. \(2018\)](#) directly predicts the future segmentation map from past ones ([Bhattacharyya et al., 2018](#); [Rochan et al., 2018](#); [Chen and Han, 2019](#)). In this study, we follow this methodology

for future prediction tasks, primarily for the sake of simplicity. In contrast, feature-level forecasting predicts future intermediate features from past ones as done by [Luc et al. \(2018\)](#); [Saric et al. \(2020\)](#); [Chiu et al. \(2020\)](#) and [\(Lin et al., 2021\)](#) which is the best-performing method on this task.

4.3 Method

4.3.1 Diffusion Models

Given samples $\mathbf{x}_0 \sim q(\mathbf{x}_0)$ from a data distribution, diffusion models ([Sohl-Dickstein et al., 2015](#); [Ho et al., 2020](#)) are characterized by a forward Markov process $q(\mathbf{x}_{1:T}|\mathbf{x}_0)$ - that destroys the data \mathbf{x}_0 through the successive addition of noise creating a sequence of increasingly corrupted latent variables $\mathbf{x}_1, \dots, \mathbf{x}_T$ - and by the corresponding reverse process $p_\theta(\mathbf{x}_{0:T})$ which is trained to gradually remove the noise. For the case of continuous data, the forward process usually adds Gaussian noise, whereas for discrete data any Markov transition matrix can be used as a corruption.

During training, the goal is to learn the reverse process such that at inference the model can generate data by gradually denoising a random input. The usual diffusion optimization process consists of minimizing a variational upper bound of the negative log-likelihood as follows

$$L_{vb} = \mathbb{E}_{q(\mathbf{x}_0)} \left[D_{KL} [q(\mathbf{x}_T | \mathbf{x}_0) \| p(\mathbf{x}_T)] + \sum_{t=2}^T \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)} [D_{KL} [q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0) \| p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)]] - \mathbb{E}_{q(\mathbf{x}_1|\mathbf{x}_0)} [\log p_\theta(\mathbf{x}_0 | \mathbf{x}_1)] \right]. \quad (4.1)$$

4.3.2 Discrete Diffusion

As previously mentioned, for the case of discrete random variables, where $\mathbf{x}_0, \dots, \mathbf{x}_T$ are one-hot vectors, arbitrary Markov transition matrices can be employed to define the forward process. In particular, following [Austin et al. \(2021\)](#), at the t -th forward step a transition probability matrix \mathbf{Q}_t is used such that: $q(\mathbf{x}_t | \mathbf{x}_{t-1}) = \mathbf{x}_{t-1} \mathbf{Q}_t$. Note that the transitions are applied independently to each input dimension (e.g. pixel).

In order to use the matrices \mathbf{Q}_t for a discrete diffusion model they need to have the following two properties. First, we want $\bar{\mathbf{Q}}_T = \mathbf{Q}_1 \mathbf{Q}_2 \dots \mathbf{Q}_T$ to converge to a stationary distribution in order to be able to sample \mathbf{x}_T to start the denoising process. Second,

we want to be able to efficiently compute $\bar{\mathbf{Q}}_t$ which enables efficient sampling of \mathbf{x}_t from the t -step marginal $q(\mathbf{x}_t|\mathbf{x}_0) = \mathbf{x}_0\bar{\mathbf{Q}}_t$ for any t . This permits efficient training as we can optimize Equation (4.1) one term at a time with SGD without having to compute the full forward process for every gradient step. While Austin et al. (2021) proposes various alternatives for \mathbf{Q}_t that satisfy these conditions, we found uniform transition matrices to work best, *i.e.* matrices with uniform transition probabilities written as $\mathbf{Q}_t = (1 - \beta_t)\mathbf{I} + \beta_t\mathbf{1}\mathbf{1}^T/K$, where $\mathbf{1}$ is a column vector of all ones and K the number of possible states.

To learn the reverse process, we train a model to predict \mathbf{x}_0 given \mathbf{x}_t , denoted as $\tilde{p}_\theta(\tilde{\mathbf{x}}_0|\mathbf{x}_t)$, where θ are the model parameters. Subsequently, to get the transition probabilities $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$, we marginalise over \mathbf{x}_0 as follows:

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \sum_{\tilde{\mathbf{x}}_0} q(\mathbf{x}_{t-1}|\mathbf{x}_t, \tilde{\mathbf{x}}_0) \tilde{p}_\theta(\tilde{\mathbf{x}}_0|\mathbf{x}_t).$$

where the posterior $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ has a simple analytical expression that can be derived using Bayes' rule and the Markov property.

Finally, we utilize a prediction loss $L_p = \mathbb{E}_{q(\mathbf{x}_0)}\mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)}[-\log \tilde{p}_\theta(\mathbf{x}_0|\mathbf{x}_t)]$ which encourages the correct prediction of \mathbf{x}_0 at each step of the reverse process. Overall, we optimize $L = L_{vb} + \lambda L_p$ as a combination of L_{vb} with this prediction loss L_p scaled by a scalar λ .

4.3.3 Discrete Diffusion Models for Segmentation

In the context of semantic segmentation, every random variable \mathbf{x} corresponds to one pixel in the image. The number of different states K corresponds to the number of segmented classes in the dataset, which is generally a low number below 100, so we do not encounter scaling-related problems.

We adopt the Unet architecture classically used in diffusion, and in particular, we adapt code from Wang (2023). The default Image Diffusion Unet architecture takes as input an image with the current time step and returns a slightly less noisy version of that image. For the Segmentation Diffusion Unet, we instead have a corrupted segmentation map as input and get a denoised version of it as output, *ie.* a prediction $\tilde{\mathbf{x}}_0$ of the clean segmentation. Since segmentation maps are discrete, we first project them into a continuous vector space of dimension E with an embedding layer and convert the Unet output to the probability distribution $\tilde{p}_\theta(\tilde{\mathbf{x}}_0|\mathbf{x}_t)$ with a softmax layer. This distribution is then used to compute $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ and sample from it.

Chapter 4. Handling ambiguous contexts: An application of Discrete Diffusion model to Segmentation

Input Conditioning Our aim is to generate the segmentation map that corresponds to the input frame, not just any random one. Consequently, we must condition the generation process accordingly. To do so, we concatenate the conditioning tensor channel-wise to the embedded noisy segmentation and use this as input to the Unet. Among the different conditioning approaches we tried, we found the simple concatenation to be fast and perform consistently well across all tasks.

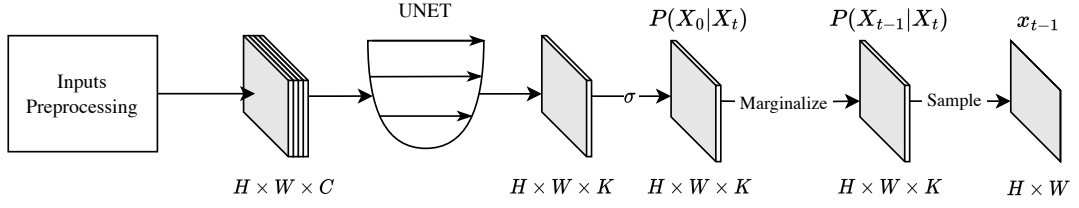
In this chapter, we tackle tumor segmentation and future prediction, for which we use slightly different conditioning tensors. For the medical segmentation task, we directly use the one-channel gray-scale lung scan as conditioning input. For the future segmentation task, we instead use the concatenation of multiple previous segmentations as conditioning. These segmentations are evenly spaced in time and are embedded in the same space as the noisy input \mathbf{x}_t . The rationale for this choice is clarified below.

Time autoregressive Diffusion Conditioning with past segmentation masks, as described above, allows us to forecast the segmentation of a frame occurring at a specific time interval from the current one. To extend our predictive horizon, we propose autoregressively predicting the next segmentation, which involves using the predicted segmentation masks as new inputs to the model. The overall model architectures used are presented in Figure 4.3.1. Using our Segmentation Diffusion model autoregressively constrains the kind of inputs our model can use because the inputs and outputs must be of the same type, *i.e.* segmentation masks. As a result, we rely only on previous segmentations as conditioning inputs for our future segmentation networks.

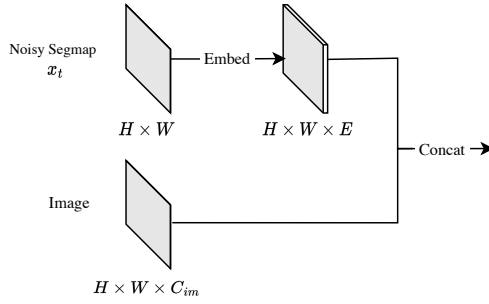
However, we point out that restricting the input exclusively to segmentation masks limits the information available to the model to make its predictions. To illustrate this with a concrete example, consider the scenario where a group of three pedestrians may be represented as a single, undifferentiated blob on the segmentation map. This choice can hinder the model’s ability to discern individual objects or finer details within the scene, thereby affecting the accuracy of its predictions.

4.3.4 Motivating Example: The rectangle world

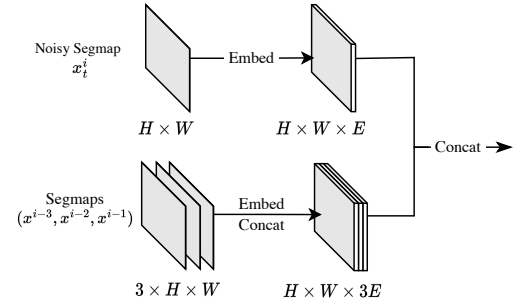
In order to further motivate the need for properly modeling uncertainty in the segmentation task, we create an artificial image segmentation dataset. Each image contains two rectangles randomly filled with one of two colors, with equal probability. However, instead of mapping each color to a single category, it is mapped randomly and equiprobably to two categories. In each image, there are exactly four equally probable correct predictions since we use two rectangles per image, as shown in Figure 4.3.2a. We additionally add noise to the input image by flipping each pixel to a random color with 40% probability.



(a) Overall architecture of Discrete Diffusion Models used for segmentation.

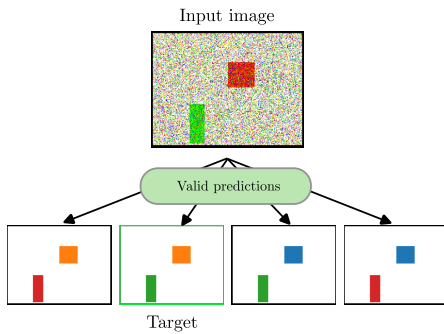


(b) Input preprocessing block used for the medical segmentation task. Conditioning inputs are one-channel images representing lung scans.

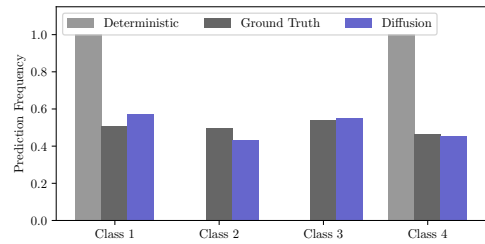


(c) Input preprocessing block used for the future prediction task. Conditioning inputs are segmentation maps that have to be embedded.

Figure 4.3.1: Architectural parts of Discrete Diffusion model adapted for segmentation. All architectures start with an input preprocessing block that handles (1) the embedding of segmentations, (2) the concatenation of the embedded noisy segmentation (a) with the conditioning tensor (b and c).



(a) Example data point from the rectangle world dataset. The noisy input image can have one of the 4 possible ground truths but only one is provided in the dataset.



(b) The predicted class frequency from 100 random inputs in the test set. We see that the deterministic model only generates 2 classes.

Figure 4.3.2: Rectangle world experiment results. Figure 4.3.2a shows an example from the rectangle world dataset, while Figure 4.3.2b showcases our results.

Chapter 4. Handling ambiguous contexts: An application of Discrete Diffusion model to Segmentation

We train a deterministic segmentation model and our generative diffusion model on the aforescribed dataset for 100,000 iterations. Both models, as expected, result in very crisp segmentations. However, as can be observed in Figure 4.3.2b the deterministic model completely misses half of the categories and only ever predicts two. In contrast, the generative model captures the uncertainty of the dataset and generates all possible classes even though the rectangles are still crisp and contain only a single category. Further details and qualitative results of this experiment are provided in Appendix 4.E.

4.4 Experiments

Building on the insights of the motivating example presented above, we conduct more extensive experiments using both simulated and real-world datasets.

For each task, we train a diffusion model and perform inference using only 10 diffusion steps, since we noticed that performing more steps was not bringing significant quality improvements while noticeably impacting generation time. We also train a deterministic model in addition to the diffusion model. This model has the exact same architecture as the diffusion model, except it does not receive the ‘current’ noisy segmentation as input. This model naturally generates a single output, which corresponds to the maximum likelihood prediction for a given input.

The use of a deterministic model is generally ill-suited for ambiguous tasks, and even more for autoregressive future prediction. Firstly, taking the argmax of the segmentation distribution may result in the selection of only a portion of an object’s pixels, for example when two locations are equally likely. This kind of error may be further amplified when using this segmentation to autoregressively predict future ones, resulting in a sequence of progressive deterioration. Secondly, even if the autoregressive generation process leads to a valid segmentation sequence, it only has the capability to generate a single one. However, despite these potential issues, we have found that generated sequences are generally of good quality, and we have therefore also reported ‘deterministic’ results for all experiments.

4.4.1 Lung Cancer Dataset

Dataset

The first ambiguous task we aim to tackle is the segmentation of tumors from the medical image dataset LIDC (Armato III et al., 2011). This dataset contains volumetric

lung CT scans, and we have adopted a pre-processing methodology identical to the one described in [Kohl et al. \(2018\)](#) so we can make meaningful comparisons to their results.

In particular, this pre-processing involves the transformation of the original 3D scans into 2D image crops. These are then re-sampled to 180x180 pixels and centered around the scan abnormality location. We omit the inclusion of small abnormalities measuring less than 3mm in this process, as they are often considered clinically insignificant. This results in 8843 images for training, 1993 for validation, and 1980 for testing.

Four expert radiologists have individually annotated each scan for the presence of abnormalities. For a given scan image, the tumor annotations can vary significantly among the radiologists, as we can see on Figure 4.4.1. By design, the training of conventional deterministic models cannot capture this variability. Segmentation diffusion models fit perfectly into this context and allow sampling from the learned distribution.



Figure 4.4.1: Two training examples from the LIDC dataset. The leftmost column is the lung scans, while the four next columns are the lesion masks as segmented by four different radiologists. These examples illustrate the substantial variability inherent in the dataset.

Results

As mentioned previously, we train both a diffusion model and a deterministic model on this task. During training, we sample at random one of the four available ground truths to perform the optimization step. To evaluate the performance of these models,

Chapter 4. Handling ambiguous contexts: An application of Discrete Diffusion model to Segmentation

we employ the Hungarian-matching-based metric proposed by [Kohl et al. \(2019\)](#). This consists in sampling multiple segmentation masks from the model and using linear assignment to match each mask with the ground truth it has the highest IoU with. In this case, we sample 16 segmentations. Note that we assign the same number of segmentations to each ground truth annotation. For more details regarding the metric computation, see Appendix 4.B.2.

Hungarian Matched IoU (%)	Full Test Set	Subset B
H. P. Unet (Kohl et al., 2019)	53	47
Deterministic Model	43.8	47.5
Diffusion Model	54	54.9

Table 4.4.1: Hungarian Matched IoU (%) for the Hierarchical Probabilistic Unet, a deterministic model, and our proposed diffusion model, all computed using 16 samples. The diffusion model is on-par on the full test set, and noticeably better when looking at subset B. Subset B comprises only the scans where all 4 radiologists found a lesion.

We report the results in Table 4.4.1 for the 2 models, alongside the results of the Hierarchical Probabilistic Unet (HPU) as reported in [Kohl et al. \(2019\)](#). We also show example generations in Figure 4.4.2. Our results are on the full test set as well as on subset B, a subset comprising all scans where all four experts agree that there is some lesion. Examining the results on subset B is interesting as they are more related to the model’s proficiency in capturing shape variations. Indeed, an incorrect prediction, even if small, of a lesion where none exists results in a zero IoU score. This edge case has a strong influence on the overall HMIOU score. When looking at subset B, the presence of a lesion is generally clear, so most model generations should contain a non-empty segmentation and the metric is mostly influenced by the model’s ability to accurately outline lesion shapes and variations.

With a 1% improvement, the diffusion model is marginally better than the original HPU paper on the full test set. However, on subset B, the diffusion model performs substantially better than HPU, which indicates our model’s superior ability to capture shape variations. Interestingly, on subset B the simple deterministic Unet model already performs as well as the HPU. This suggests that the HPU does not generate enough variations that properly cover the predictions of the physicians.

4.4.2 Car Intersection Simulator

In order to clearly showcase the ability of these diffusion models to handle complex distributions with multiple modes, we design a dataset where each frame has a distinct

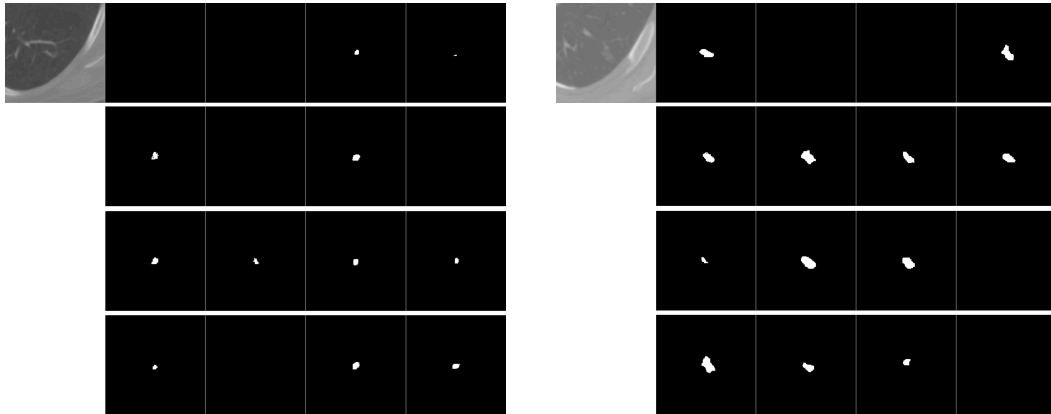


Figure 4.4.2: Qualitative results for two different lung scans on which we sampled the diffusion model 12 times. The top row has the input and the four ground truths. The three bottom rows show samples. More examples can be found in Section 4.B.3.

and limited set of potential future scenarios. Precisely, this dataset is constructed such that the uncertainty between two successive frames can be represented as a Markov process. Because this transition is Markovian, we train the models only using one previous segmentation as conditioning input. Note that we perform autoregressive segmentation prediction, but that we evaluate the overall performance of the model using a trajectory prediction metric, which allows to better quantify how often we are able to sample the correct scenario.

To establish a strong baseline, we train a Transformer encoder model alongside deterministic and diffusion models. Since the segmentation space is too large to be handled by the transformer model directly (a sequence length of 65k), we use a VQVAE (Van Den Oord et al., 2017) to encode the segmentations into a latent space where the transformer generates its predictions. For more details about the architecture, please refer to Appendix 4.C.2.

Simulator and Dataset

The dataset is created using a custom bird’s-eye-view car driving simulator that generates both an image and the corresponding segmentation map. In this simulator, the road layouts feature a single intersection, which can be either a roundabout or a three/four-way crossing. Each simulation features a random number of cars between 2 and 5. Each car belongs to one of four classes and moves along the roads with the same constant speed, kept identical across simulations. Additionally, its direction is clearly identifiable from the segmentation mask. Therefore, given the current segmentation, all potential routes a car can take are unambiguously predetermined.

Chapter 4. Handling ambiguous contexts: An application of Discrete Diffusion model to Segmentation

The intersection type determines the number of potential future routes a car may have, with a maximum of four different possibilities. For this reason, we use a single previous timestep as conditioning for all networks.

The final dataset consists of 4,992,000 training and 1,000 validation examples generated by running the simulator for 20 steps and capturing the segmentation map of each frame. Thus, a training example consists of 20 successive segmentation maps of dimension 256x256.

Results and Discussion

As this is a future prediction task, we use the autoregressive scheme during inference to predict the segmentation of the next 20 frames. From the sampled sequence, we map out the overall trajectory of each car using a simple tracking algorithm and compute the Final Distance Error (FDE) between the last predicted position and the ground truth. For evaluation, we repeat this sampling scheme 10 times to get 10 distinct trajectories per car, from which we only select the trajectory exhibiting the lowest FDE. In Table 4.4.2, we report the mean lowest FDE across 10 samples. We also compute the miss rate, defined as the fraction of trajectories whose FDE exceeds two, which implies the trajectory diverges significantly from the actual route. Please refer to Appendix 4.C.4 for the description of our trajectory extraction procedure and to Appendix 4.C.3 for more details regarding the impact of the number of samples on the miss rate. The inference time per sample, recorded in milliseconds, is measured on an Nvidia V100 GPU.

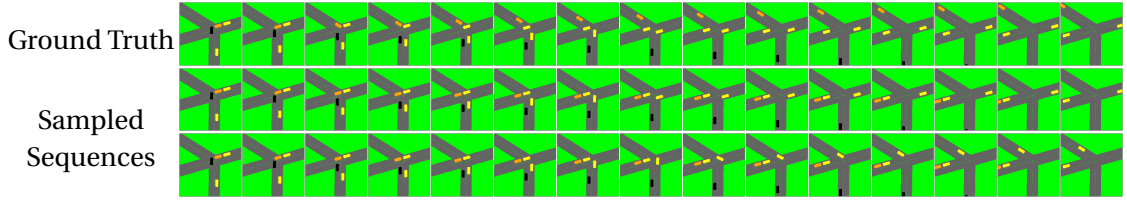
	FDE	Miss Rate	TIME (ms/spl)
Deterministic Model (36M)	68	52.1 %	70
Transformer Encoder (50M)	11.9	19.2 %	3270
Diffusion Model (36M)	11.4	16.1 %	580

Table 4.4.2: Results for trajectory predictions on the car simulator dataset, using the trajectory with the lowest FDE among 10 samples. Both generative models solve the task, and the diffusion model performs best. The modest performance obtained by the deterministic model confirms the inherent ambiguity of the dataset. The high sampling time of the transformer encoder suggests that these models scale poorly to higher dimensionality. Lower is better in all columns.

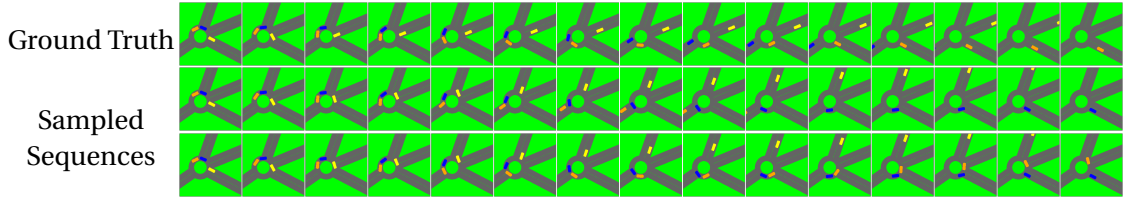
Table 4.4.2 shows that the diffusion model is performing best, and the miss rate more specifically indicates that this model is most often predicting the correct trajectory compared to the transformer baseline. Additionally, contrary to the transformer model

which samples each pixel sequentially and has to work in a learned latent space, the diffusion model works in the image space and has an adjustable number of steps. As specified earlier, we only use ten steps in all experiments, which makes generation with diffusion about five times faster than with the transformer.

Interestingly, the deterministic model has a 52 % miss rate, which indicates that the one trajectory predicted by this model is correct about half the time. While confirming a degree of ambiguity in the dataset, this miss rate shows that the deterministic model is still able to complete the task at least half of the time, and not degenerate the predicted segmentation as the autoregressive process goes on.



(a) Qualitative results for a four-way intersection.



(b) Qualitative results for a four-way roundabout.

Figure 4.4.3: Qualitative results for sampling the diffusion model on the car intersection simulator. The top row is the ground truth, the two bottom rows are sampled sequences. We show only the fifteen first frames due to space constraints. The leftmost frame is the one used as the first input to the model.

4.4.3 Cityscapes

Finally, we aim to tackle the task of segmenting first-person view road scenes. The goal of this task is to predict the segmentation results of unobserved future frames. Short-term future can generally be treated as a deterministic task, relying on the current state of the scene and motion models to estimate what will happen in the next few frames. In contrast, the mid-term future is often a complex prediction problem. As we look further into the future, the task becomes more uncertain and the scene may diverge into different distinct scenarios depending on the behavior of other road users. For instance, a vehicle ahead may change lanes or a pedestrian may start crossing the road.

Chapter 4. Handling ambiguous contexts: An application of Discrete Diffusion model to Segmentation

When we consider the potential combinations of all feasible behaviors for each agent present in the scene, it can yield dozens of distinct future outcomes. The ability to generate multiple samples of possible future segmentations allows to cover many of the different scenarios, and to address critical questions such as: ‘*Is there a scenario in which the child crosses the road ?*’. From this perspective, we believe it is reasonable to compare segmentation results obtained from existing methodologies with the best sample we can generate through our diffusion-based approach.

Dataset

We conducted our experiments with the Cityscapes Dataset (Cordts et al., 2016), which was collected specifically for video-based segmentation research. It comprises a total of 5,000 sequences, with 2,975 sequences designated for model training, 500 for validation, and 1525 for testing. A sequence is made of 30 image frames, among which the 20th frame is manually annotated for semantic segmentation.

Following Lin et al. (2021), we perform short-term ($t + 3$) and mid-term ($t + 9$) prediction, and additionally report next-frame prediction ($t + 1$). To perform the short-term prediction, we try two configurations: we use the subsampled sequence $\{S_{t-6}, S_{t-3}, S_t\}$ as input and do one autoregressive (AR) step similarly to Lin et al. (2021); we also show results using $\{S_{t-2}, S_{t-1}, S_t\}$ with 3 AR steps. To perform the mid-term prediction, we input the sequence $\{S_{t-6}, S_{t-3}, S_t\}$ and do 3 AR steps.

In order to extract the segmentation masks used as conditioning inputs, we run a pretrained segmentation network on previous timesteps. Specifically, we run the transformer-based Segmenter-B model from Strudel et al. (2021), pretrained on ImageNet, and fine-tuned on Cityscapes.

Results and Discussion

In Table 4.4.3, we employ a simple visual code to depict different prediction configurations. Each circle symbolizes one frame, and we arrange them linearly to represent the consecutive frames. The blue circles denote frames used as input, and the red ones are the predicted frames. When performing multiple autoregressive steps, we use one line for each step and stack them vertically.

For the diffusion model, we present the results of using 1, 10, and 100 samples. When sampling multiple future segmentations, we calculate the mIoU based on the sampled segmentation that best matches the ground truth, as determined by the IoU metric. As motivated above, we argue that when the objective is to be able to account for all

possible scenarios in order to capture the true one, it is fair to compare with the ‘best’ of multiple diffusion samples.




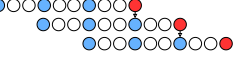
mIoU (%)				
	Short Term		Mid Term	
Saric et al. (2020)	N/A	69.6	N/A	57.9
Sović et al. (2022)	N/A	70.2	N/A	58.5
Lin et al. (2021)	N/A	71.1	N/A	60.3
Deterministic Model	79.5	71.5	72.2	57.6
Diffusion Model (1 spl)	78.8	70.8	71.4	56.6
Diffusion Model (10 spl)	79.1	71.3	72.4	58.0
Diffusion Model (100 spl)	79.3	71.9	73.1	59.1

Table 4.4.3: Results for Cityscapes future segmentation tasks. In the header, each circle represents a timestep, blue ones are used as input, and red ones are predicted. The diffusion model with multiple samples is competitive with current state-of-the-art networks. The deterministic model falls most of the time between 1-sample and 100-sample evaluation, confirming the ambiguity of the task. Results are in terms of mIoU percentage, where higher is better.

While the current best baseline ([Lin et al., 2021](#)) performance is strong, we outperform it in the short-term prediction when using multiple samples, and we come close in the mid-term prediction. Note that this baseline model relies on predicting intermediate image features, while we only use previous segmentation maps. We conjecture that our lower results in the mid-term context compared to the deterministic model from [Lin et al. \(2021\)](#) are relatively imputable to the information loss of using segmentation maps as inputs. As their feature prediction strategy is orthogonal to our approach, we believe we could benefit from it and achieve more competitive results.

Additionally, it is interesting to note that the results of the deterministic model are almost always higher than 1-sample diffusion but lower than 100-sample diffusion. This observation aligns with our expectations, given that the deterministic model is specifically trained to output the most likely outcome, while the diffusion model stochastically samples from possible outcomes. Consequently, it is reasonable to anticipate slightly lower performance on average for the 1-sample diffusion results, as they only represent a single stochastic sample. However, if the task indeed contains ambiguity and the diffusion model can effectively sample from the different modes of the distribution, we would expect improved results on average when considering the best of 100 samples, as it is likely that at least one of them will more closely match the ground truth. And we indeed see that as we predict further in the future and uncertainty increases, the performance gap between the deterministic model and the

Chapter 4. Handling ambiguous contexts: An application of Discrete Diffusion model to Segmentation

100-sample diffusion model also widens.

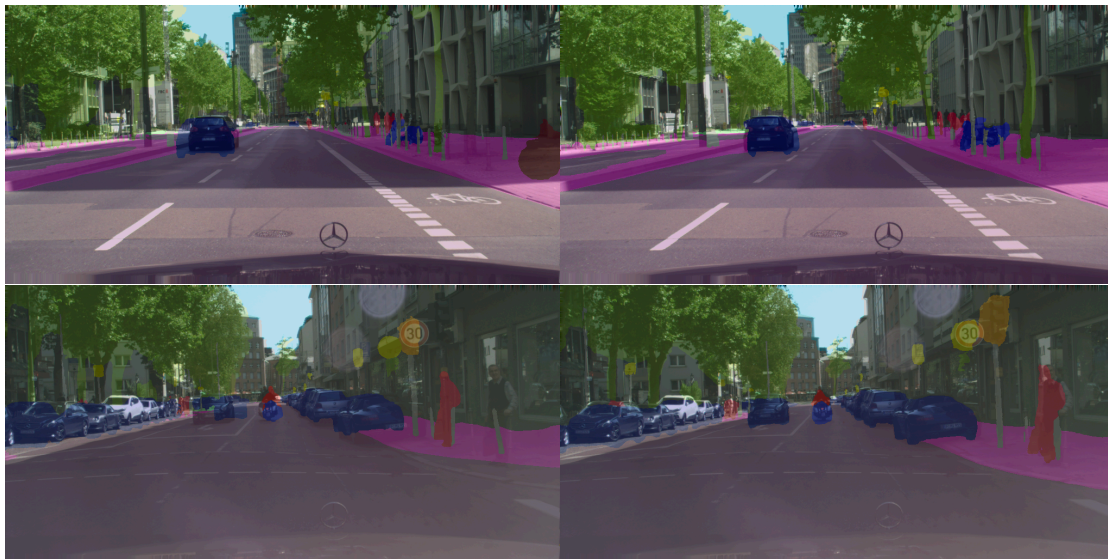


Figure 4.4.4: Qualitative results for two Cityscapes mid-term future segmentation scenarios at $T + 9$. The left column shows what would happen if we were to simply copy the last segmentation map, and the right column shows one sample from our diffusion model.

4.5 Conclusion

This chapter delved into the application of discrete diffusion models to address the challenges posed by non-deterministic and high-uncertainty situations in different segmentation tasks. This study holds significance for anticipatory networks, given their operation in the realm of future segmentation, and we presented a novel approach to perform long-term prediction that leverages discrete diffusion models within an autoregressive framework, using only previous segmentations as input. Additionally, this exploration also covered medical image segmentation and demonstrated the effectiveness of our approach in modeling uncertain segmentation contexts.

Our experiments show that these discrete models offer a promising avenue for performing segmentation in such ambiguous contexts. However, there is still much space for improvement. Specifically, one relevant future direction for our research is to find ways to force diversity in the sampling process, which could involve a form of importance sampling. Currently, the only way at our disposal to cover the whole space of the distribution is through repetitive sampling. This is particularly inefficient, especially for the autoregressive scheme proposed for future prediction. Another interesting direction is assessing whether the computation of the Variational Bound L_{vb} can help to address safety critical questions. Indeed, computing a reliable estimate of this bound may provide a reliable indicator regarding the probability of a given situation happening. However, adapting this approach to make sense in the context of segmentation maps remains an open question.

Appendix - Chapter 4

4.A Model Architecture details

For all tasks, the network architecture is a U-Net from [Wang \(2023\)](#), both for the deterministic and diffusion model, and the only difference is the input processing block. The segmentations are embedded in a 32 dimensional space learnt during training in all experiments. We use 5 contractions (down) and 5 expansions (up) layers, the initial dimension is 32 and the dimension multipliers are (1, 2, 2, 4, 8, 16).

4.B LIDC experiment details

4.B.1 Training details

As mentioned in Section 4.4.1, the pre-processing follows [Kohl et al. \(2018\)](#). In addition, the data preparation pipeline for training consists in: randomly selecting one of the 4 ground truths, resizing image and label to 128x128 pixels using nearest-neighbor interpolation, randomly applying horizontal flip, and normalizing the images with a mean of 0.41 and a standard deviation of 0.21.

We train both models with batch-size 64 for 10,000 epochs, using a prediction loss weight $\lambda = 1e^{-3}$ for the diffusion model. We use AdamW with a learning rate of $1e^{-3}$ and weight decay of $1e^{-4}$, and a polynomial learning rate scheduler.

4.B.2 Metric

As described in Section 4.4.1, we evaluate the performance using the Hungarian-matching based metric proposed by [Kohl et al. \(2019\)](#).

The idea behind this metric is to measure the alignment between the model’s predictions’ distribution and the experts’ predictions’ distribution. To do so, we determine the best matching between the sampled segmentations and the ground truth annotations using linear assignment on the IoU scores.

In practice, we first sample a number of segmentation masks that is a multiple of the number of ground truths. In the case of LIDC, there are 4 ground truths and we sample 16 segmentations. Note that sampling more segmentations does not increase the score but reduces its variance. Then, we compute the IoU between every pair of samples and labels. We use these scores to perform linear assignment, assigning 4

samples to each of the 4 ground truth labels. Finally, we average the IoU of the all the pairs that have been assigned together. Figure 4.B.1 visually explains this process when using 4 samples.

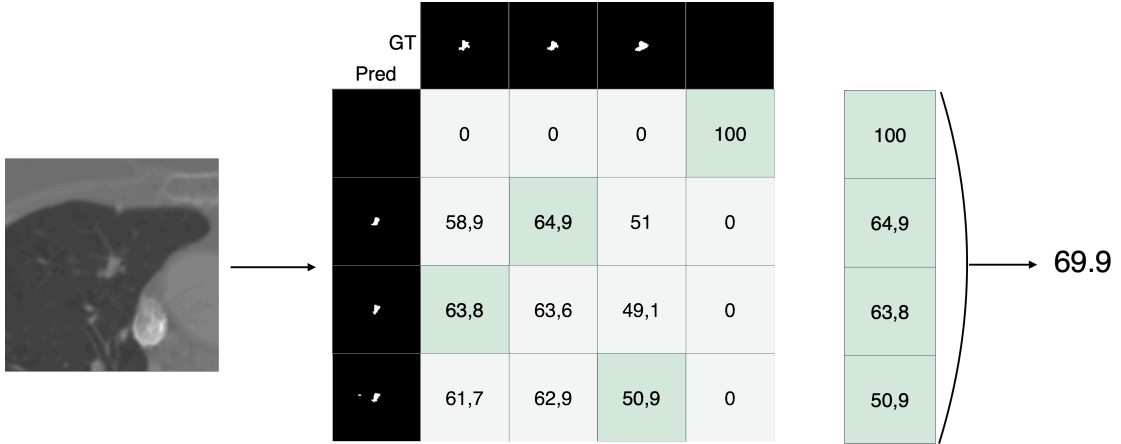


Figure 4.B.1: Visual explanation of HMIoU. We sample 4 segmentation masks (Pred), compute their IoU with each of the ground truth (GT) labels, find the best linear assignment, and average the IoU scores of the assigned pairs.

4.B.3 Qualitative results

On Figure 4.B.2, we show more qualitative results of LIDC segmentation samples. Samples have been vertically aligned under the ground truth they were matched with by the HMIoU computation. The examples on the left are showing failure cases where the shapes are only partly correct and their frequency is not matching the labels' frequency. The examples on the right shows success cases where complex shapes variation are properly captured.

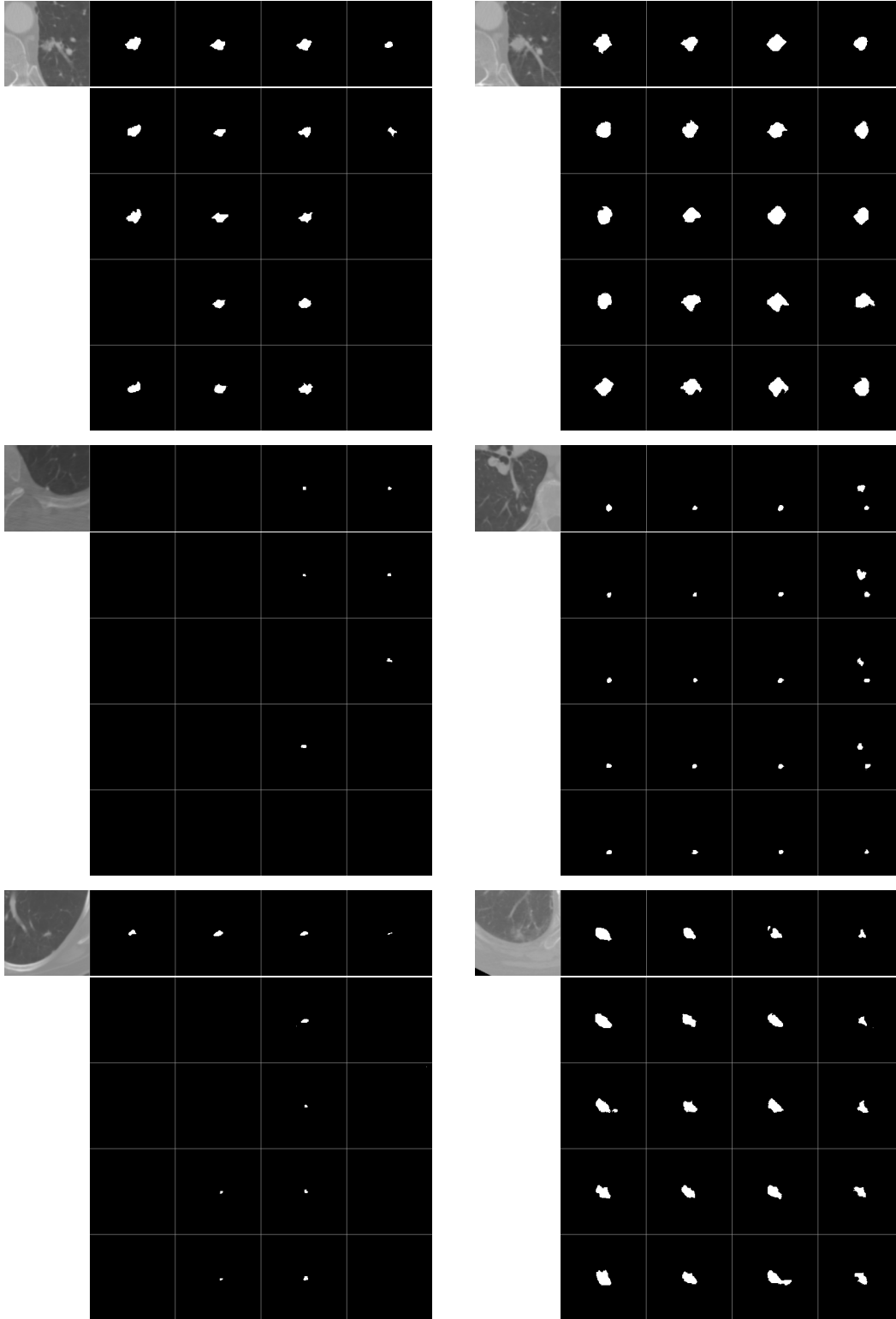


Figure 4.B.2: Qualitative results for 6 different lung scans on which we sampled the diffusion model 16 times. For each example, the top row has the input scan and the four ground truths. The four bottom rows show samples. The examples on the left are failure cases, the examples on the right are well capturing shape variations.

4.C Car simulator experiment details

4.C.1 Training details

We train all models for 1 epoch and there are no data augmentation.

The diffusion model use batch-size 25, using a prediction loss weight $\lambda = 1e^{-3}$. We use AdamW with a learning rate of $1e^{-3}$ and weight decay of $1e^{-4}$, and a polynomial learning rate scheduler.

The deterministic model uses batch-size 40. We use AdamW with a learning rate of $3e^{-4}$ and weight decay of $1e^{-4}$, and a polynomial learning rate scheduler.

The Transformer-Encoder model uses batch-size 40. We use AdamW with a learning rate of $1e^{-3}$ and a polynomial learning rate scheduler.

4.C.2 Transformer-Encoder architecture

As generative baseline for this new task, we use a transformer encoder architecture with 16 encoder layers, and feature dimension 512. Since the $256 \times 256 = 65536$ pixel space is too large to process, we use a VQ-VAE to embed the input segmentations into a latent space using 256 codes. Given the 256 latent codes from the input segmentation, the transformer is sequentially generating the 256 codes of the next frame's segmentation.

VQVAE Architecture The VQVAE consists of an Encoder, a Decoder and a Quantizer.

The VQVAE encoder consists of:

- An embedding layer mapping the segmentation classes to a 16-channels continuous space.
- A convolutional stem (in_channels=16, out_channels=128, kernel_size=2, stride=2)
- A sequence of 5 ConvNext blocks with 128 channels, interlaced with 3 averaging pooling layers (kernel_size=2).
- A linear projection layer (in_channels=128, out_channels=32)

The VQVAE quantizer is an EMA Vector Quantizer with a dictionary of 512 vector codes of dimension 32. The EMA quantizer uses exponential moving averages to update the embedding vectors instead of using an auxiliary loss.

Chapter 4. Handling ambiguous contexts: An application of Discrete Diffusion model to Segmentation

The VQVAE decoder consists of:

- A linear projection layer (in_channels=32, out_channels=128)
- A sequence of 5 ConvNext blocks with 128 channels, interlaced with 3 bilinear upsampling layers (scale_factor=2).
- An output layer mapping the output back to the 16-channels input embedding space.

This VQVAE is trained over 100,000 samples using a batch-size of 10. We use AdamW with a learning rate of $1e^{-3}$ and weight decay of $1e^{-4}$, and a polynomial learning rate scheduler.

4.C.3 Impact of the number of samples on the miss rate

The Figure 4.C.1 shows the evolution of the miss rate, which measures how frequently the correct trajectory is absent among the sampled trajectories, with respect to the number of samples used to compute the miss rate. Each sample allows to get one new possible trajectory per car in the scenario, extracted from the segmentation sequence using our trajectory extraction procedure. Some of these trajectories may be very similar to each other, but Figure 4.C.1 shows that the more we sample, the more likely we are to generate the correct trajectory. The notable reduction in the miss rate with each additional sample indicates that the models are indeed able to cover a substantial portion of the trajectory space.

4.C.4 Trajectory extraction and comparison

To extract car trajectories from a sequence of segmentation maps, we implement the following procedure:

For all frames:

- Identify all connected components and discard those that are too small,
- Determine the center of mass for these components, which will represent the car locations.

For each pair of consecutive frames:

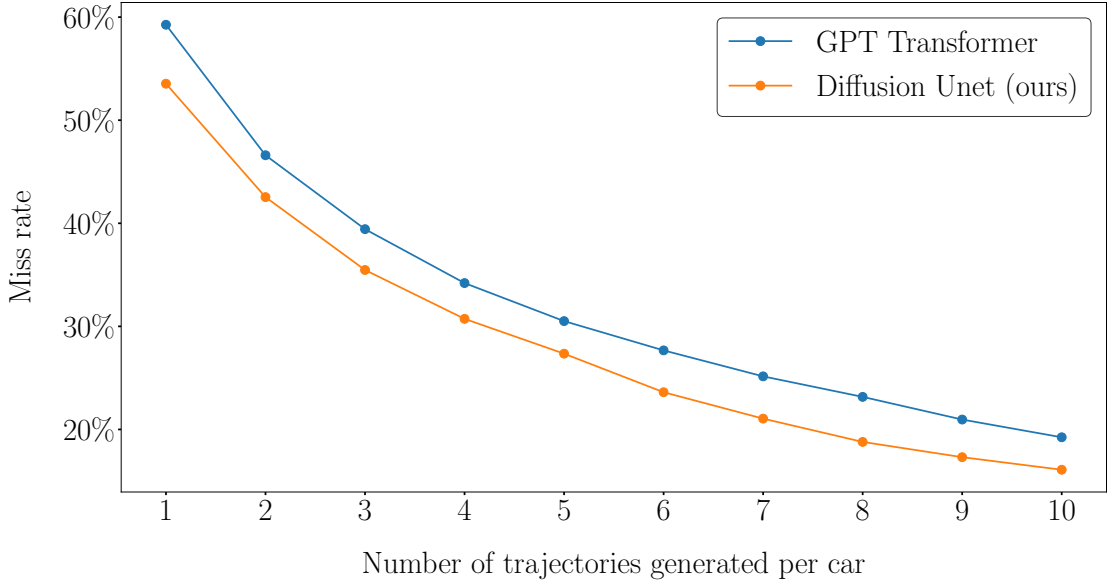


Figure 4.C.1: Evolution of the miss rate with the number of samples used during evaluation, both for the diffusion model (orange) and the Transformer-Encoder model (blue). Performing multiple samplings clearly allows to cover more scenarios and generate the real trajectory more frequently.

- Calculate the squared distance matrix between car locations in the first frame and those in the subsequent frame,
- Compute the optimal pairing of car locations from the first frame with those from the second, utilizing linear sum assignment,
- Remove from the pairing all pairs of cars that are too far from each other,
- For car locations that remained not paired, consider them as entering or exiting the scenario.

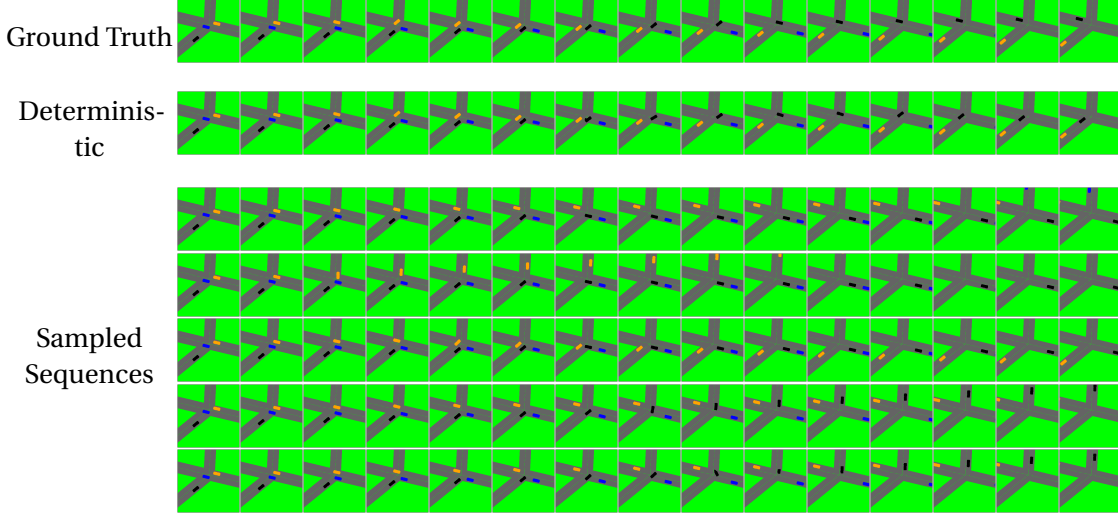
By keeping track of all paired car locations while processing all consecutive segmentations, we can derive the complete trajectory.

In our car intersection simulator experiments, we enhance robustness by excluding trajectories of cars that don't begin at the start of the simulation, as all simulated cars are present in the segmentation right from the beginning of the simulation.

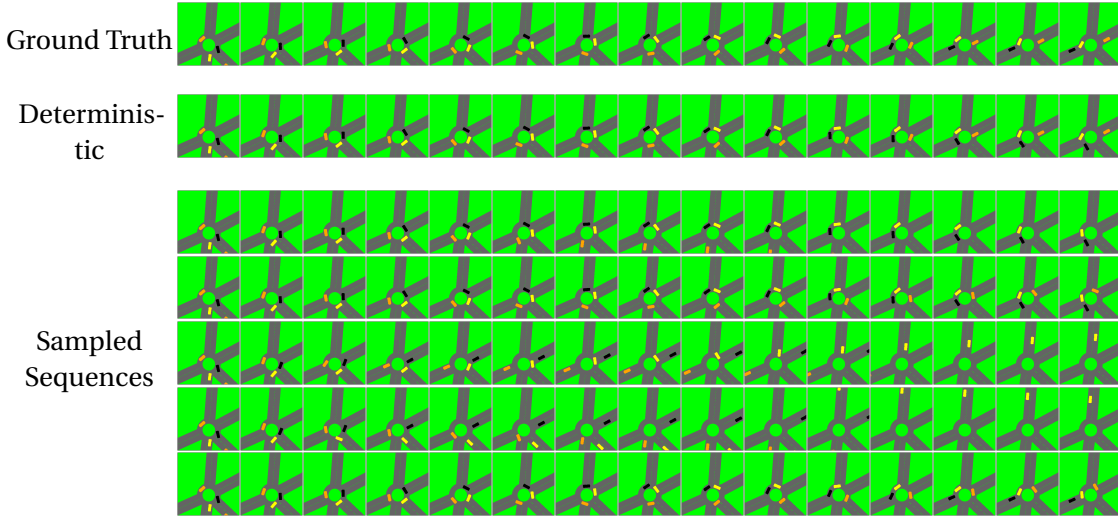
To compare trajectory extracted from the ground truth sequence and a sampled sequence, we use the fact both sequences share the same initial frame. Hence, we simply align trajectories based on the cars' initial positions.

4.C.5 Qualitative results

On Figure 4.C.2b, we show more qualitative results for intersections with multiple possible scenarios.



(a) Qualitative results for a four-way intersection.



(b) Qualitative results for a four-way roundabout.

Figure 4.C.2: Qualitative results on the car intersection simulator. From top: The first row is the ground truth, the second row is the result from the deterministic model, the five next rows are sampled sequences. We show only the fifteen first frames due to space constraints. The leftmost frame is the one used as first input to the model.

4.D Cityscapes experiment details

4.D.1 Training details

On the Cityscapes experiments, we use 3 previous segmentations all embedded in 32 dimensions, the conditioning tensor is therefore 96 dimensional. The 3 previous segmentation have a constant time spacing between each of them. However, during training, we have varied this time spacing for each training sample.

More specifically, during training we randomly sample a spacing $\tau \in \{1, 2, 3\}$ and we use the segmentations $(S_{t-3\tau}, S_{t-2\tau}, S_{t-\tau})$ as input to the models. This allows us to use such a trained model with any small constant spacing τ in inference, and in fact all the results reported for the deterministic and diffusion model in Tables 4.D.1 and 4.4.3 are computed using a single model.

We train the diffusion model for 1000 epochs with a batch-size 3 and a prediction loss weight $\lambda = 0.1$. We use AdamW with a learning rate of $3e^{-4}$, weight decay of $1e^{-4}$ and a polynomial learning rate scheduler.

We train the deterministic model for 1000 epochs with a batch-size 1. We use AdamW with a learning rate of $1e^{-4}$, weight decay of $1e^{-4}$ and a polynomial learning rate scheduler.

4.D.2 Moving Object Results

In Table 4.D.1, we report results for Cityscapes computed on Moving Objects only. The patterns we observe are similar to those from Table 4.4.3, with the deterministic model performances being always between the 1-sample and 100-samples diffusion model results.

The diffusion model using multi-samples suffers from a much smaller performance drop compared to all other models, including Lin et al. (2021). This highlights the ability of the model to capture different scenarios.

4.D.3 Qualitative Results

We show more samples of our diffusion model for future segmentation on Cityscapes in Figure 4.D.1.

Chapter 4. Handling ambiguous contexts: An application of Discrete Diffusion model to Segmentation

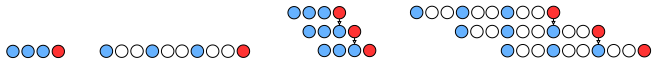
mIoU (%)				
		Short Term		Mid Term
Saric et al. (2020)	N/A	67.7	N/A	54.6
Sović et al. (2022)	N/A	69.0	N/A	55.9
Lin et al. (2021)	N/A	69.2	N/A	56.7
Deterministic Model	79.7	70.5	71.3	53.6
Diffusion Model (1 spl)	79.0	69.9	70.0	52.5
Diffusion Model (10 spl)	79.5	70.9	71.9	54.5
Diffusion Model (100 spl)	80.0	71.5	72.7	56.0

Table 4.D.1: Results for Cityscapes future segmentation tasks on **Moving Objects**. In the header, each circle represents a time step, blue ones are used as input and red ones are predicted. The diffusion model with multiple samples is competitive with the current state-of-the-art network. The deterministic model most of the times falls between 1-sample and 100-samples evaluation, confirming the ambiguity of the task. Results are in terms of mIoU percentage, where higher is better.

4.E Motivating Example Details

The dataset consists of images of size 240×320 pixels, each containing two rectangles filled in either green or red. Since it is an artificial dataset, we have a random seed for training and a different random seed for evaluation. To make the problem slightly more challenging, we randomly swap 40% of the pixels with random colors as can be seen clearly in Figure 4.E.1.

We train both models for 100,000 iterations with a batch size of 16. The Unet has 2.8M parameters and is the same for both the diffusion model and the deterministic one, with the exception of the first layer that embeds the segmentation mask or the image respectively.

In addition to the results shown in Section 4.3.4, we also measure the ability of our generative models to create all possible classes given a single input image. In Figure 4.E.2 we show the frequency with which each class was predicted for each rectangle when generating 100 images from a single input. We observe that all classes are predicted with probability close to 50%, same as they appear in the dataset.



Figure 4.D.1: Qualitative results for Cityscapes short-term ($T + 3$) and mid-term ($T + 9$) future segmentation scenarios. The left column shows what would happen if we were to simply copy the last segmentation map, the next columns on the right shows samples from our diffusion model.

Chapter 4. Handling ambiguous contexts: An application of Discrete Diffusion model to Segmentation

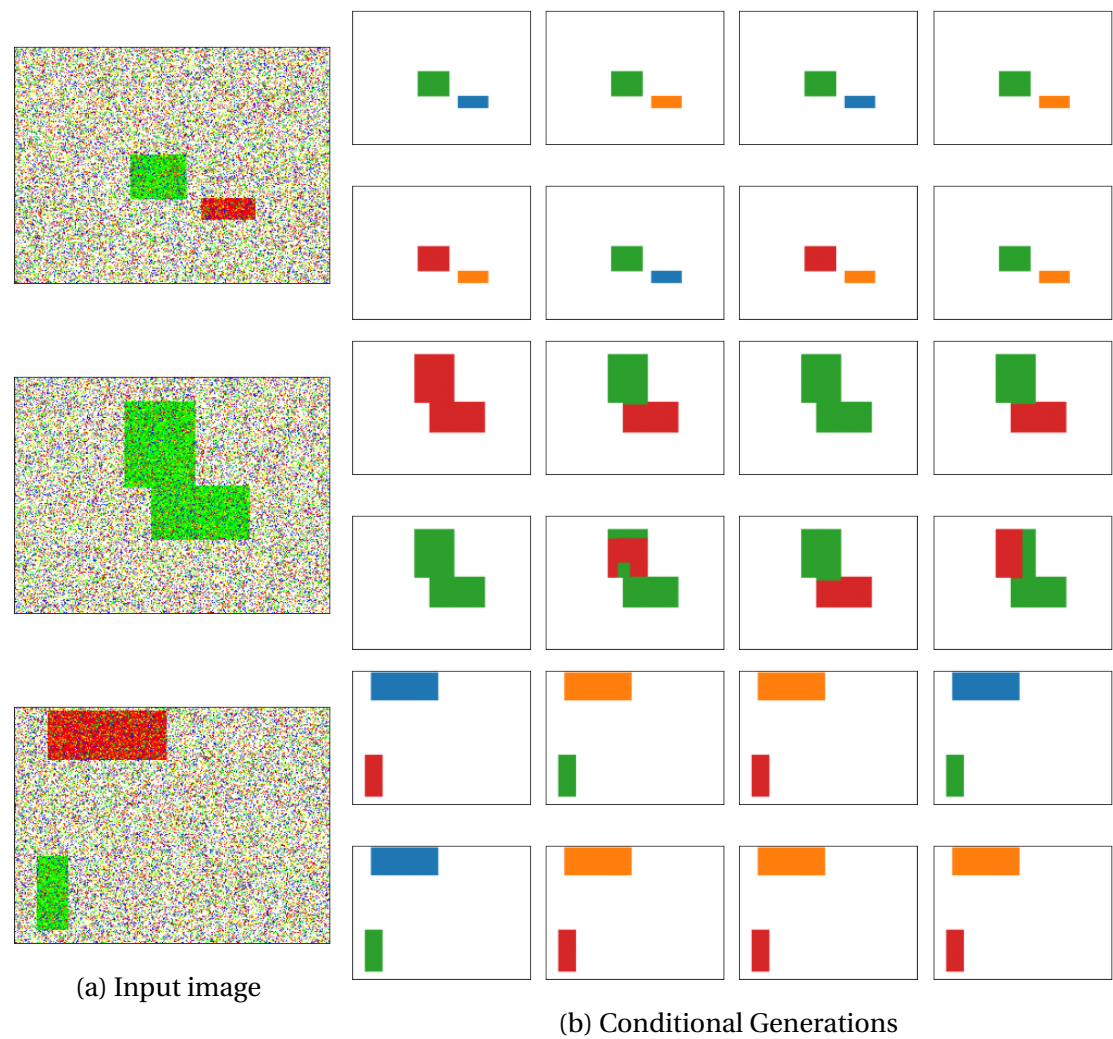


Figure 4.E.1: Example input images and 8 generations from the diffusion model. We observe that all classes have been generated for the input rectangles.

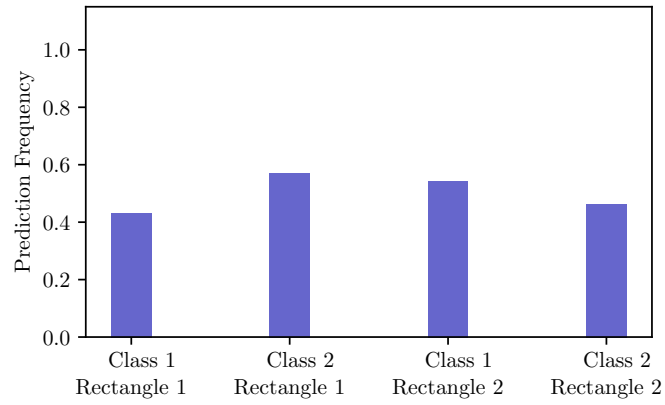


Figure 4.E.2: We show the predicted class frequency from 100 generated samples from our diffusion model conditioned on the same input image. All categories are generated with equal probability.

Conclusion

Summary

In this doctoral study, we explored what we term ‘anticipatory’ segmentation networks. These networks take into account their latency and do not directly predict the input frames’ segmentation but instead the segmentation of a future frame. We have looked at various aspects of these networks.

In Chapter 1, we have explained the need for real-time segmentation networks to be trained with anticipation in mind. We have introduced the LA-mIoU, a segmentation metric taking into account the network latency. This metric has a practical significance and, although it was specifically studied within the context of segmentation, defining a similar latency-aware metric for any other task, such as LA-accuracy or LA-mAP, is straightforward. However, computing it can only be performed on video datasets with a high sampling frequency. As the dataset’s sampling rate decreases, the metric becomes unable to distinguish highly efficient networks. Finally, we have also proposed to use past frames in a simple way and have seen the positive impact on future predictions.

In Chapter 2, we aimed to propose a way to improve real-time anticipatory networks. At the time of this research, Transformers had not yet reached Computer Vision, and we focused on accelerating convolutions, the core component of convolutional networks. We introduced our channel-wise masked convolution, the key idea of which centered on leveraging the temporal coherence of video frames so that we recompute only a subset of all features for each video frame. This technique is specifically designed for the future-prediction task and in that context we have seen that we could get up to 20% speed-up with little to no drop in mIoU.

After focusing on speeding up convolutional segmentation networks, we looked at accelerating transformer-based ones in Chapter 3. We leveraged the partition of images into patches to do early-stopping of patches processing. As our task is segmentation, we have to retain all patches for the final prediction, but we have shown that not all of

Conclusion

them require the same amount of processing. We have confirmed the effectiveness of using the entropy of intermediate predictions as a criterion to stop the processing of already confidently predicted patches. Our method allows reaching consequent speed-ups with little mIoU drop. Moreover, a nice feature of our training scheme is that it lets a practitioner change the network latency by controlling the ratio of patches paused without having to retrain the whole model. Recently, real-time transformer architectures were introduced for segmentation, and we expect they could benefit from our patch-pausing approach. For such real-time video settings, another interesting future work direction could involve investigating whether comparing patches across time steps may enhance efficiency by skipping the processing of similar ones.

Recently, given the impressive achievements of diffusion networks in image generation, we decided to explore their application in the field of semantic segmentation. Through our experiments, we have not found diffusion models to produce more accurate segmentation maps than existing models. However, we observed that these generative models are particularly relevant for handling another aspect of anticipatory networks - ambiguity. We presented these findings in Chapter 4. Indeed, we noticed that we could harness their strong generative capabilities as a way to produce multiple possible segmentations that closely follow the distribution of the training data. In particular, we have adapted a discrete variant of diffusion to the segmentation domain, and we have demonstrated its clear superiority over previous models on a ‘multi-groundtruth’ medical imaging dataset. We also tackled the generation of future predictions, a task inherently holding uncertainty. To generate long-term predictions, we have integrated discrete diffusion within an autoregressive framework, demonstrating its proficiency in capturing potential future scenarios. In particular, we developed a synthetic car intersection simulator and showed that our method was able to generate most of the possible trajectories in this environment. As our sampling method does not favor diversity, multiple samplings are required to generate all possible outcomes, and reducing this need is a promising research direction. On the Cityscapes dataset, the results of experiments were slightly less successful, largely due to the lack of any additional information other than the semantic maps as conditional input. Therefore, another interesting extension to this study could be to implement panoptic segmentation within the same setup. This has the potential to enhance the prediction quality by incorporating additional instance information.

Overall, this thesis work revolved around addressing the various challenges associated with real-time semantic segmentation. It encompassed multiple aspects, including the development of predictive networks, enhancing their computational efficiency, and accommodating the inherent uncertainty in prediction outcomes.

Final remarks

The landscape of semantic segmentation has undergone significant transformation from the onset of this doctoral work, driven by rapid advances in hardware, software, and computer vision research and development. This dynamic and intellectually stimulating field is seeing marked benefits from new, high-performance hardware, making real-time capabilities more within reach for existing networks. Simultaneously, the emergence of large-scale models has pushed the mIoU to new heights. Given the trend of ever-increasing model size, developing methods to improve efficiency is all the more important. Several contributions made in this thesis address this need and therefore shall remain particularly relevant in the future.

Separately, the newly introduced 'segment anything' models - SAM and SEEM - present new avenues for future exploration, especially regarding their adaptation to real-time full-scene semantic segmentation. These models uniquely unite text and images by leveraging pretrained language models, allowing the vision model to harness the world knowledge captured within the language model. The growing popularity and impressive performance of such multimodal models, especially on zero-shot tasks, encourages me to posit that the future of computer vision, including segmentation, leans towards a multimodal approach.

Bibliography

- Amit, T., Shaharbany, T., Nachmani, E., and Wolf, L. (2021). Segdiff: Image segmentation with diffusion probabilistic models. *arXiv preprint arXiv:2112.00390*.
- Armato III, S. G., McLennan, G., Bidaut, L., McNitt-Gray, M. F., Meyer, C. R., Reeves, A. P., Zhao, B., Aberle, D. R., Henschke, C. I., Hoffman, E. A., Kazerooni, E. A., MacMahon, H., van Beek, E. J. R., Yankelevitz, D., Biancardi, A. M., Bland, P. H., Brown, M. S., Engelmann, R. M., Laderach, G. E., Max, D., Pais, R. C., Qing, D. P.-Y., Roberts, R. Y., Smith, A. R., Starkey, A., Batra, P., Caligiuri, P., Farooqi, A., Gladish, G. W., Jude, C. M., Munden, R. F., Petkovska, I., Quint, L. E., Schwartz, L. H., Sundaram, B., Dodd, L. E., Fenimore, C., Gur, D., Petrick, N., Freymann, J., Kirby, J., Hughes, B., Vande Casteele, A., Gupta, S., Sallam, M., Heath, M. D., Kuhn, M. H., Dharaiya, E., Burns, R., Fryd, D. S., Salganicoff, M., Anand, V., Shreter, U., Vastagh, S., Croft, B. Y., and Clarke, L. P. (2011). The lung image database consortium (lidc) and image database resource initiative (idri): A completed reference database of lung nodules on ct scans. *Medical Physics*, 38(2):915–931.
- Austin, J., Johnson, D. D., Ho, J., Tarlow, D., and Van Den Berg, R. (2021). Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems*, 34:17981–17993.
- Badrinarayanan, V., Kendall, A., and Cipolla, R. (2017). Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495.
- Baumgartner, C. F., Tezcan, K. C., Chaitanya, K., Hötter, A. M., Muehlematter, U. J., Schawkat, K., Becker, A. S., Donati, O., and Konukoglu, E. (2019). Phiseg: Capturing uncertainty in medical image segmentation. In *Medical Image Computing and Computer Assisted Intervention–MICCAI 2019: 22nd International Conference, Shenzhen, China, October 13–17, 2019, Proceedings, Part II 22*, pages 119–127. Springer.
- Bhat, I., Pluim, J. P., and Kuijff, H. J. (2022). Generalized probabilistic u-net for medical

Bibliography

- image segmentation. In *International Workshop on Uncertainty for Safe Utilization of Machine Learning in Medical Imaging*, pages 113–124. Springer.
- Bhattacharyya, A., Fritz, M., and Schiele, B. (2018). Bayesian prediction of future street scenes using synthetic likelihoods. *arXiv preprint arXiv:1810.00746*.
- Bolya, D., Fu, C.-Y., Dai, X., Zhang, P., Feichtenhofer, C., and Hoffman, J. (2022). Token merging: Your vit but faster. *arXiv preprint arXiv:2210.09461*.
- Chao, P., Kao, C.-Y., Ruan, Y.-S., Huang, C.-H., and Lin, Y.-L. (2019). Hardnet: A low memory traffic network. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3552–3561.
- Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. (2014). Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*.
- Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. (2017a). Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848.
- Chen, L.-C., Papandreou, G., Schroff, F., and Adam, H. (2017b). Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*.
- Chen, L.-C., Zhu, Y., Papandreou, G., Schroff, F., and Adam, H. (2018). Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 801–818.
- Chen, T., Li, L., Saxena, S., Hinton, G., and Fleet, D. J. (2022a). A generalist framework for panoptic segmentation of images and videos. *arXiv preprint arXiv:2210.06366*.
- Chen, T., Zhang, R., and Hinton, G. (2022b). Analog bits: Generating discrete data using diffusion models with self-conditioning. *arXiv preprint arXiv:2208.04202*.
- Chen, W., Gong, X., Liu, X., Zhang, Q., Li, Y., and Wang, Z. (2019). Fasterseg: Searching for faster real-time semantic segmentation. *arXiv preprint arXiv:1912.10917*.
- Chen, X. and Han, Y. (2019). Multi-timescale context encoding for scene parsing prediction. In *2019 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1624–1629. IEEE.
- Chen, X., Wang, X., Changpinyo, S., Piergiovanni, A., Padlewski, P., Salz, D., Goodman, S., Grycner, A., Mustafa, B., Beyer, L., et al. (2022c). Pali: A jointly-scaled multilingual language-image model. *arXiv preprint arXiv:2209.06794*.

- Cheng, B., Misra, I., Schwing, A. G., Kirillov, A., and Girdhar, R. (2022). Masked-attention mask transformer for universal image segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1290–1299.
- Chiu, H.-k., Adeli, E., and Niebles, J. C. (2020). Segmenting the future. *IEEE Robotics and Automation Letters*, 5(3):4202–4209.
- Choi, S., Kim, J. T., and Choo, J. (2020). 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*, pages 9373–9383.
- Chu, X., Tian, Z., Wang, Y., Zhang, B., Ren, H., Wei, X., Xia, H., and Shen, C. (2021). Twins: Revisiting the design of spatial attention in vision transformers. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 9355–9366. Curran Associates, Inc.
- Contributors, M. (2020). MMSegmentation: Openmmlab semantic segmentation toolbox and benchmark. <https://github.com/open-mmlab/mmdetection>.
- Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., and Schiele, B. (2016). The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223.
- Couprie, C., Luc, P., and Verbeek, J. (2018). Joint future semantic and instance segmentation prediction. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pages 0–0.
- Courdier, E. and Fleuret, F. (2020). Real-time segmentation networks should be latency aware. In *Proceedings of the Asian Conference on Computer Vision*.
- Courdier, E. and Fleuret, F. (2022). Borrowing from yourself: Faster future video segmentation with partial channel update. In *2022 26th International Conference on Pattern Recognition (ICPR)*, pages 1–8. IEEE.
- Courdier, E., Katharopoulos, A., and Fleuret, F. (2023). Segmenting the unknown: Discrete diffusion models for non-deterministic segmentation. Submitted to the *International Conference on Learning Representations (ICLR)*.
- Courdier, E., Sivaprasad, P. T., and Fleuret, F. (2022). Paumer: Patch pausing transformer for semantic segmentation. In *33th British Machine Vision Conference 2022, London, UK, 21 - 24 November 2022*.

Bibliography

- Ding, H., Jiang, X., Shuai, B., Qun Liu, A., and Wang, G. (2018). Context contrasted feature and gated multi-scale aggregation for scene segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2393–2402.
- Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*.
- Dosovitskiy, A., Fischer, P., Ilg, E., Hausser, P., Hazirbas, C., Golkov, V., Van Der Smagt, P., Cremers, D., and Brox, T. (2015). FlowNet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2758–2766.
- Ethayarajh, K., Choi, Y., and Swayamdipta, S. (2022). Understanding dataset difficulty with \mathcal{V} -usable information. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S., editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 5988–6008. PMLR.
- Fan, M., Lai, S., Huang, J., Wei, X., Chai, Z., Luo, J., and Wei, X. (2021). Rethinking bisenet for real-time semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9716–9725.
- Figurnov, M., Collins, M. D., Zhu, Y., Zhang, L., Huang, J., Vetrov, D., and Salakhutdinov, R. (2017). Spatially adaptive computation time for residual networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1039–1048.
- Fu, J., Liu, J., Tian, H., Li, Y., Bao, Y., Fang, Z., and Lu, H. (2019). Dual attention network for scene segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3146–3154.
- Gadde, R., Jampani, V., and Gehler, P. V. (2017). Semantic video cnns through representation warping. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4453–4462.
- Gao, R. (2021). Rethink dilated convolution for real-time semantic segmentation. *arXiv preprint arXiv:2111.09957*.
- Graves, A. (2016). Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*.

- Habibian, A., Abati, D., Cohen, T., and Bejnordi, B. E. (2021). Skip-convolutions for efficient video processing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Han, Y., Huang, G., Song, S., Yang, L., Wang, H., and Wang, Y. (2021). Dynamic neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1.
- He, J., Deng, Z., and Qiao, Y. (2019). Dynamic multi-scale filters for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3562–3572.
- He, K., Zhang, X., Ren, S., and Sun, J. (2015a). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.
- He, K., Zhang, X., Ren, S., and Sun, J. (2015b). Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916.
- He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.
- He, W., Wu, M., Liang, M., and Lam, S.-K. (2021). Cap: Context-aware pruning for semantic segmentation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 960–969.
- Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.
- Ho, J., Jain, A., and Abbeel, P. (2020). Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851.
- Hong, Y., Pan, H., Sun, W., Jia, Y., et al. (2021). Deep dual-resolution networks for real-time and accurate semantic segmentation of road scenes. *arXiv preprint arXiv:2101.06085*.
- Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., et al. (2019). Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1314–1324.
- Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv:1704.04861 [cs]*. arXiv: 1704.04861.

Bibliography

- Hu, J.-F., Sun, J., Lin, Z., Lai, J.-H., Zeng, W., and Zheng, W.-S. (2021). Apanet: Auto-path aggregation for future instance segmentation prediction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3386–3403.
- Hu, P., Caba, F., Wang, O., Lin, Z., Sclaroff, S., and Perazzi, F. (2020). Temporally distributed networks for fast video semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8818–8827.
- Huang, Z., Wang, X., Huang, L., Huang, C., Wei, Y., and Liu, W. (2019). Ccnet: Criss-cross attention for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 603–612.
- Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. (2016). Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size. *arXiv preprint arXiv:1602.07360*.
- Jang, E., Gu, S., and Poole, B. (2016). Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.
- Jin, X., Li, X., Xiao, H., Shen, X., Lin, Z., Yang, J., Chen, Y., Dong, J., Liu, L., Jie, Z., et al. (2017a). Video scene parsing with predictive feature learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5580–5588.
- Jin, X., Xiao, H., Shen, X., Yang, J., Lin, Z., Chen, Y., Jie, Z., Feng, J., and Yan, S. (2017b). Predicting scene parsing and motion dynamics in the future. In *Advances in Neural Information Processing Systems*, pages 6915–6924.
- Kaya, Y., Hong, S., and Dumitras, T. (2019). Shallow-deep networks: Understanding and mitigating network overthinking. In *International conference on machine learning*, pages 3301–3310. PMLR.
- Khan, S., Naseer, M., Hayat, M., Zamir, S. W., Khan, F. S., and Shah, M. (2021). Transformers in vision: A survey. *ACM Comput. Surv.*
- Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A. C., Lo, W.-Y., Dollár, P., and Girshick, R. (2023). Segment anything. *arXiv:2304.02643*.
- Kohl, S., Romera-Paredes, B., Meyer, C., De Fauw, J., Ledsam, J. R., Maier-Hein, K., Eslami, S., Jimenez Rezende, D., and Ronneberger, O. (2018). A probabilistic u-net for segmentation of ambiguous images. *Advances in neural information processing systems*, 31.

- Kohl, S. A., Romera-Paredes, B., Maier-Hein, K. H., Rezende, D. J., Eslami, S., Kohli, P., Zisserman, A., and Ronneberger, O. (2019). A hierarchical probabilistic u-net for modeling multi-scale ambiguities. *arXiv preprint arXiv:1905.13077*.
- Kong, Z., Dong, P., Ma, X., Meng, X., Niu, W., Sun, M., Shen, X., Yuan, G., Ren, B., Tang, H., et al. (2022). Spvit: Enabling faster vision transformers via latency-aware soft token pruning. In *European Conference on Computer Vision*, pages 620–640. Springer.
- Lavin, A. and Gray, S. (2016). Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4013–4021.
- Li, H., Xiong, P., Fan, H., and Sun, J. (2019a). Dfanet: Deep feature aggregation for real-time semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9522–9531.
- Li, X., Liu, Z., Luo, P., Change Loy, C., and Tang, X. (2017). 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.
- Li, X., You, A., Zhu, Z., Zhao, H., Yang, M., Yang, K., Tan, S., and Tong, Y. (2020). Semantic flow for fast and accurate scene parsing. In *European Conference on Computer Vision*, pages 775–793. Springer.
- Li, Y., Chen, X., Zhu, Z., Xie, L., Huang, G., Du, D., and Wang, X. (2019b). Attention-guided unified network for panoptic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7026–7035.
- Li, Y., Shi, J., and Lin, D. (2018). Low-latency video semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5997–6005.
- Liang, Y., GE, C., Tong, Z., Song, Y., Wang, J., and Xie, P. (2022a). EVit: Expediting vision transformers via token reorganizations. In *International Conference on Learning Representations*.
- Liang, Y., Ge, C., Tong, Z., Song, Y., Wang, J., and Xie, P. (2022b). Not all patches are what you need: Expediting vision transformers via token reorganizations. *arXiv preprint arXiv:2202.07800*.
- Lin, Z., Sun, J., Hu, J.-F., Yu, Q., Lai, J.-H., and Zheng, W.-S. (2021). Predictive feature learning for future segmentation prediction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7365–7374.

Bibliography

- Liu, C., Chen, L.-C., Schroff, F., Adam, H., Hua, W., Yuille, A. L., and Fei-Fei, L. (2019a). Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 82–92.
- Liu, Y., Chen, K., Liu, C., Qin, Z., Luo, Z., and Wang, J. (2019b). Structured knowledge distillation for semantic segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2604–2613.
- Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.
- Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440.
- Luc, P., Couprie, C., Lecun, Y., and Verbeek, J. (2018). Predicting future instance segmentation by forecasting convolutional features. In *Proceedings of the european conference on computer vision (ECCV)*, pages 584–599.
- Luc, P., Neverova, N., Couprie, C., Verbeek, J., and LeCun, Y. (2017). Predicting deeper into the future of semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 648–657.
- Marin, D., Chang, J.-H. R., Ranjan, A., Prabhu, A., Rastegari, M., and Tuzel, O. (2022). Token pooling in vision transformers.
- Nekrasov, V., Chen, H., Shen, C., and Reid, I. (2020). Architecture search of dynamic cells for semantic video segmentation. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 1970–1979.
- Nilsson, D. and Sminchisescu, C. (2018). Semantic video segmentation by gated recurrent flow propagation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6819–6828.
- Nirkin, Y., Wolf, L., and Hassner, T. (2021). Hyperseg: Patch-wise hypernetwork for real-time semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4061–4070.
- Orsic, M., Kreso, I., Bevandic, P., and Segvic, S. (2019). 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*, pages 12607–12616.

- Pan, B., fan Jiang, Y., Panda, R., Wang, Z., Feris, R. S., and Oliva, A. (2021). Ia-red2: Interpretability-aware redundancy reduction for vision transformers. In *NeurIPS*.
- Paszke, A., Chaurasia, A., Kim, S., and Culurciello, E. (2016). Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147*.
- Patraucean, V., Handa, A., and Cipolla, R. (2015). Spatio-temporal video autoencoder with differentiable memory. *arXiv preprint arXiv:1511.06309*.
- Peng, C., Zhang, X., Yu, G., Luo, G., and Sun, J. (2017). Large kernel matters—improve semantic segmentation by global convolutional network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4353–4361.
- Pinaya, W. H., Graham, M. S., Gray, R., Da Costa, P. F., Tudosiu, P.-D., Wright, P., Mah, Y. H., MacKinnon, A. D., Teo, J. T., Jager, R., et al. (2022). Fast unsupervised brain anomaly detection and segmentation with diffusion models. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 705–714. Springer.
- Raghu, M., Unterthiner, T., Kornblith, S., Zhang, C., and Dosovitskiy, A. (2021). Do vision transformers see like convolutional neural networks? In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*.
- Rahman, A., Valanarasu, J. M. J., Hacihaliloglu, I., and Patel, V. M. (2023). Ambiguous medical image segmentation using diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11536–11546.
- Rao, Y., Zhao, W., Liu, B., Lu, J., Zhou, J., and Hsieh, C.-J. (2021). Dynamicvit: Efficient vision transformers with dynamic token sparsification. In *Advances in Neural Information Processing Systems (NeurIPS)*.
- Rochan, M. et al. (2018). Future semantic segmentation with convolutional lstm. *arXiv preprint arXiv:1807.07946*.
- Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. (2022). High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695.
- Romera, E., Alvarez, J. M., Bergasa, L. M., and Arroyo, R. (2017). Erfnet: Efficient residual factorized convnet for real-time semantic segmentation. *IEEE Transactions on Intelligent Transportation Systems*, 19(1):263–272.

Bibliography

- Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer.
- Saemann, T., Amende, K., Milz, S., and Gross, H.-M. (2019). Leverage temporal consistency for robust semantic video segmentation. In *ICML 2019 Workshop on Uncertainty and Robustness in Deep Learning*, volume 2.
- Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520.
- Šarić, J., Oršić, M., Antunović, T., Vražić, S., and Šegvić, S. (2019). Single level feature-to-feature forecasting with deformable convolutions. In *Pattern Recognition: 41st DAGM German Conference, DAGM GCPR 2019, Dortmund, Germany, September 10–13, 2019, Proceedings 41*, pages 189–202. Springer.
- Saric, J., Orsic, M., Antunovic, T., Vrazic, S., and Segvic, S. (2020). Warp to the future: Joint forecasting of features and feature motion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10648–10657.
- Shelhamer, E., Rakelly, K., Hoffman, J., and Darrell, T. (2016). Clockwork Convnets for Video Semantic Segmentation. *arXiv:1608.03609 [cs]*. arXiv: 1608.03609.
- Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. (2015). Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. PMLR.
- Sović, I., Šarić, J., and Šegvić, S. (2022). Dense semantic forecasting with multi-level feature warping. *Applied Sciences*, 13(1):400.
- Steiner, A., Kolesnikov, A., , Zhai, X., Wightman, R., Uszkoreit, J., and Beyer, L. (2021). How to train your vit? data, augmentation, and regularization in vision transformers. *arXiv preprint arXiv:2106.10270*.
- Strudel, R., Garcia, R., Laptev, I., and Schmid, C. (2021). Segmenter: Transformer for semantic segmentation. *arXiv preprint arXiv:2105.05633*.
- Sun, D., Yang, X., Liu, M.-Y., and Kautz, J. (2018). Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8934–8943.
- Sun, J., Xie, J., Hu, J.-F., Lin, Z., Lai, J., Zeng, W., and Zheng, W.-s. (2019). Predicting future instance segmentation with contextual pyramid convlstm. In *Proceedings of the 27th acm international conference on multimedia*, pages 2043–2051.

- Takikawa, T., Acuna, D., Jampani, V., and Fidler, S. (2019). Gated-scnn: Gated shape cnns for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5229–5238.
- Tang, M., Perazzi, F., Djelouah, A., Ben Ayed, I., Schroers, C., and Boykov, Y. (2018). On regularized losses for weakly-supervised cnn segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 507–522.
- Tao, A., Sapra, K., and Catanzaro, B. (2020). Hierarchical multi-scale attention for semantic segmentation. *arXiv preprint arXiv:2005.10821*.
- Tay, Y., Dehghani, M., Bahri, D., and Metzler, D. (2020). Efficient transformers: A survey. *arXiv preprint arXiv:2009.06732*.
- Teerapittayanon, S., McDanel, B., and Kung, H. T. (2016). Branchynet: Fast inference via early exiting from deep neural networks. *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469.
- Terwilliger, A., Brazil, G., and Liu, X. (2019). Recurrent flow-guided semantic forecasting. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1703–1712. IEEE.
- Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., and Jegou, H. (2021). Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, volume 139, pages 10347–10357.
- Valada, A., Mohan, R., and Burgard, W. (2019). Self-supervised model adaptation for multimodal semantic segmentation. *International Journal of Computer Vision*, pages 1–47.
- Van Den Oord, A., Vinyals, O., et al. (2017). Neural discrete representation learning. *Advances in neural information processing systems*, 30.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.
- Verelst, T. and Tuytelaars, T. (2020). Dynamic convolutions: Exploiting spatial sparsity for faster inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2320–2329.
- Vora, S., Mahjourian, R., Pirk, S., and Angelova, A. (2018). Future semantic segmentation using 3d structure.

Bibliography

- Wang, J., Gou, C., Wu, Q., Feng, H., Han, J., Ding, E., and Wang, J. (2022). Rtformer: Efficient design for real-time semantic segmentation with transformer. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A., editors, *Advances in Neural Information Processing Systems*, volume 35, pages 7423–7436. Curran Associates, Inc.
- Wang, P. (2023). *Denoising Diffusion Probabilistic Model*, in Pytorch. <https://github.com/lucidrains/denoising-diffusion-pytorch>.
- Wang, W., Xie, E., Li, X., Fan, D.-P., Song, K., Liang, D., Lu, T., Luo, P., and Shao, L. (2021). 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.
- Wolleb, J., Bieder, F., Sandkühler, R., and Cattin, P. C. (2022a). Diffusion models for medical anomaly detection. In *International Conference on Medical image computing and computer-assisted intervention*, pages 35–45. Springer.
- Wolleb, J., Sandkühler, R., Bieder, F., Valmaggia, P., and Cattin, P. C. (2022b). Diffusion models for implicit image segmentation ensembles. In *International Conference on Medical Imaging with Deep Learning*, pages 1336–1348. PMLR.
- Wu, H., Zhang, J., Huang, K., Liang, K., and Yu, Y. (2019). Fastfcn: Rethinking dilated convolution in the backbone for semantic segmentation. *arXiv preprint arXiv:1903.11816*.
- Wu, J., Fang, H., Zhang, Y., Yang, Y., and Xu, Y. (2022). Medsegdiff: Medical image segmentation with diffusion probabilistic model. *arXiv preprint arXiv:2211.00611*.
- Wu, J., Fu, R., Fang, H., Zhang, Y., and Xu, Y. (2023). Medsegdiff-v2: Diffusion based medical image segmentation with transformer. *arXiv preprint arXiv:2301.11798*.
- Wu, S., Wu, T., Lin, F., Tian, S., and Guo, G. (2021). Fully transformer networks for semantic image segmentation. *arXiv preprint arXiv:2106.04108*.
- Xiao, T., Liu, Y., Zhou, B., Jiang, Y., and Sun, J. (2018). Unified perceptual parsing for scene understanding. In *European Conference on Computer Vision*. Springer.
- Xie, E., Wang, W., Yu, Z., Anandkumar, A., Alvarez, J. M., and Luo, P. (2021). Segformer: Simple and efficient design for semantic segmentation with transformers. *arXiv preprint arXiv:2105.15203*.
- Xie, J., Shuai, B., Hu, J.-F., Lin, J., and Zheng, W.-S. (2018). Improving fast segmentation with teacher-student learning. *arXiv preprint arXiv:1810.08476*.

- Xin, J., Tang, R., Lee, J., Yu, Y., and Lin, J. (2020). 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. Association for Computational Linguistics.
- Xu, J., De Mello, S., Liu, S., Byeon, W., Breuel, T., Kautz, J., and Wang, X. (2022). Groupvit: Semantic segmentation emerges from text supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18134–18144.
- Xu, W., Xu, Y., Chang, T., and Tu, Z. (2021). Co-scale conv-attentional image transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9981–9990.
- Yan, H., Zhang, C., and Wu, M. (2022). Lawin transformer: Improving semantic segmentation transformer with multi-scale representations via large window attention. *arXiv preprint arXiv:2201.01615*.
- Yin, H., Vahdat, A., Alvarez, J., Mallya, A., Kautz, J., and Molchanov, P. (2022). A-ViT: Adaptive tokens for efficient vision transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.
- Yu, C., Gao, C., Wang, J., Yu, G., Shen, C., and Sang, N. (2020). Bisenet v2: Bilateral network with guided aggregation for real-time semantic segmentation. *arXiv preprint arXiv:2004.02147*.
- Yu, C., Wang, J., Peng, C., Gao, C., Yu, G., and Sang, N. (2018a). BiSeNet: Bilateral Segmentation Network for Real-time Semantic Segmentation. *arXiv:1808.00897 [cs]*. arXiv: 1808.00897.
- Yu, F., Koltun, V., and Funkhouser, T. (2017). Dilated residual networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 472–480.
- Yu, J. and Huang, T. S. (2019). Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1803–1811.
- Yu, J., Wang, Z., Vasudevan, V., Yeung, L., Seyedhosseini, M., and Wu, Y. (2022). Coca: Contrastive captioners are image-text foundation models. *arXiv preprint arXiv:2205.01917*.
- Yu, J., Yang, L., Xu, N., Yang, J., and Huang, T. (2018b). Slimmable neural networks.

Bibliography

- Yuan, Y., Chen, X., Chen, X., and Wang, J. (2021). Segmentation transformer: Object-contextual representations for semantic segmentation. In *European Conference on Computer Vision (ECCV)*, volume 1.
- Yuan, Y., Chen, X., and Wang, J. (2019). Object-contextual representations for semantic segmentation. *arXiv preprint arXiv:1909.11065*.
- Zhang, H., Wu, C., Zhang, Z., Zhu, Y., Zhang, Z., Lin, H., Sun, Y., He, T., Mueller, J., Manmatha, R., et al. (2020a). Resnest: Split-attention networks. *arXiv preprint arXiv:2004.08955*.
- Zhang, X., Xu, H., Mo, H., Tan, J., Yang, C., and Ren, W. (2020b). Dcnas: Densely connected neural architecture search for semantic image segmentation. *arXiv preprint arXiv:2003.11883*.
- Zhang, X., Zhou, X., Lin, M., and Sun, J. (2018a). Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856.
- Zhang, Z., Zhang, X., Peng, C., Xue, X., and Sun, J. (2018b). Exfuse: Enhancing feature fusion for semantic segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 269–284.
- Zhao, H., Qi, X., Shen, X., Shi, J., and Jia, J. (2017). ICNet for Real-Time Semantic Segmentation on High-Resolution Images. *arXiv:1704.08545 [cs]*. arXiv: 1704.08545.
- Zhao, H., Shi, J., Qi, X., Wang, X., and Jia, J. (2016). Pyramid Scene Parsing Network (PSPNet). *arXiv:1612.01105 [cs]*. arXiv: 1612.01105.
- Zheng, S., Lu, J., Zhao, H., Zhu, X., Luo, Z., Wang, Y., Fu, Y., Feng, J., Xiang, T., Torr, P. H., et al. (2021). 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.
- Zhou, B., Zhao, H., Puig, X., Fidler, S., Barriuso, A., and Torralba, A. (2017). Scene parsing through ade20k dataset. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 633–641.
- Zhou, W., Xu, C., Ge, T., McAuley, J., Xu, K., and Wei, F. (2020). Bert loses patience: Fast and robust inference with early exit. *Advances in Neural Information Processing Systems*, 33:18330–18341.
- Zhu, R., Zhang, S., Wang, X., Wen, L., Shi, H., Bo, L., and Mei, T. (2019a). Scratchdet: Training single-shot object detectors from scratch. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2268–2277.

- Zhu, X., Xiong, Y., Dai, J., Yuan, L., and Wei, Y. (2017). Deep feature flow for video recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2349–2358.
- Zhu, Y., Sapra, K., Reda, F. A., Shih, K. J., Newsam, S., Tao, A., and Catanzaro, B. (2019b). Improving semantic segmentation via video propagation and label relaxation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8856–8865.
- Zou, X., Yang, J., Zhang, H., Li, F., Li, L., Gao, J., and Lee, Y. J. (2023). Segment everything everywhere all at once. *arXiv preprint arXiv:2304.06718*.

Evann Courdier

+33-676167931 | evann.courdier@idiap.ch | [Webpage](#) | [Google Scholar](#) | [LinkedIn](#) | [Github](#)

RESEARCH INTERESTS

Deep Learning, Computer Vision, and Natural Language Processing

EDUCATION

EPFL - École Polytechnique Fédérale de Lausanne	2018-Present
PhD, Information and Communication	GPA - 5.5 / 6
ENS Cachan	2016-2017
MSc, MVA: Mathematics, Computer Vision and Machine learning	GPA - 16.5 / 20
Mines ParisTech	2012-2016
MSc, General Engineering and Computer Science	GPA - 3.4 / 4
Lycée Sainte Geneviève	2010-2012
Intensive undergraduate program to prepare for national competitive examinations	

EXPERIENCE

Apple	U.S.
<i>Research Intern - AI & ML team</i>	<i>January 2023-July 2023</i>
<ul style="list-style-type: none">Worked on using generative models to tackle segmentation in ambiguous situations.	
Idiap Research Institute	Switzerland
<i>Graduate Research Assistant</i>	<i>March 2019-March 2024</i>
<ul style="list-style-type: none">Speeding up real-time semantic segmentation networks for low resource devicesReducing computation in transformer networks	
EPFL	Switzerland
<i>NLP Research Assistant</i>	<i>October 2017-May 2018</i>
<ul style="list-style-type: none">Worked on multi-objective sentence embedding	
LIP6 Sorbonne	France
<i>NLP Research Assistant</i>	<i>March 2017-July 2017</i>
<ul style="list-style-type: none">Developed a probabilistic ML model for co-reference resolution	
Theodo	France
<i>Full-Stack Development Intern</i>	<i>March 2016-August 2016</i>
<ul style="list-style-type: none">Improved 50 websites of BNP bank in a micro-service architecture	
Shenzhen Institute of Advanced Technology	China
<i>Data Visualisation Intern</i>	<i>May 2015-September 2015</i>
<ul style="list-style-type: none">Worked on interactive visualisations of sensor data	
Imperial College London - DSI	England
<i>Data Visualisation Intern</i>	<i>May 2014-September 2014</i>
<ul style="list-style-type: none">Developed a distributed visualization framework overlaying d3.js for a 64-screens data observatory	

PEER-REVIEWED PUBLICATIONS

- Courdier, E., Katharopoulos, A., and Fleuret, F. **Segmenting the unknown: Discrete diffusion models for non-deterministic segmentation** *Submitted to ICLR, 2023.*
- Courdier, E., Sivaprasad, P., and Fleuret, F. **PAUMER: Patch Pausing Transformer for Semantic Segmentation.** *BMVC, 2022.*
- Courdier, E., and Fleuret, F. **Borrowing from yourself: Faster future video segmentation with partial channel update.** *ICPR, 2022.*
- Courdier, E., and Fleuret, F. **Real-Time Segmentation Networks should be Latency Aware.** *ACCV, 2020.*

PYTORCH TUTORIALS

I presented PyTorch tutorials at various conferences online and in person:

- **AMLD 2021** - 6h - 10+ attendees
- **TheWebConf 2021** - 4.5h - 30+ attendees
- **DL4SCI 2020** - 1.5h - 500+ attendees - [video]
- **AMLD 2020** - 3h - 100+ attendees
- **AMLD 2019** - 3h - 100+ attendees

SELECTED PERSONAL WEB PROJECTS

<u>Notebook Progress Tracker</u> <i>Allow to follow student progress during labs using Jupyter notebook</i>	[Flask, React, D3, SQL]
<u>Piano booking EPFL</u> <i>Interface to book a piano room in EPFL</i>	[Vue, GoogleSheet API]
<u>Queueing System for Zoom</u> <i>Platform to queue students and match them with an assistant when available</i>	[Flask, Vue, Peerjs, SQL]
<u>London Metro Traffic Visualization</u> <i>A nice dynamic visualization of London metro data with d3.js</i>	[D3]

TECHNICAL SKILLS

Programming: Python, Web (HTML/CSS/JS), C++, SQL, Shell Scripting
Frameworks: PyTorch, React, Vue, D3, Flask, LaTeX
Operating Systems: Linux, Microsoft Windows, MacOS

PERSONAL SKILLS

- **Languages:** French (*Native*), English (*Fluent - TOEIC:955*), Russian (*B2*), German & Chinese (*Basics*).
- **Other Hobbies:** Karate (*2nd dan*), Climbing, Piano, Meditation, Coding.

MISCELLANEOUS

- Recipient of the EPFL PhD Fellowship
- Recipient of the EPFL Distinguished Service Award 2020, 2021, 2022
- Secured rank 35 (out of 3.5K+ candidates) in the Entrance Examination for Mines ParisTech