Thèse n° 10 055

Confidence Matters: Applications to Semantic Segmentation

Présentée le 29 mars 2023

Faculté des sciences et techniques de l'ingénieur
Laboratoire de l'IDIAP
Programme doctoral en génie électrique

pour l'obtention du grade de Docteur ès Sciences

par

Prabhu Teja SIVAPRASAD

Acceptée sur proposition du jury

Prof. P. Frossard, président du jury
Prof. F. Fleuret, Prof. A. M. Alahi, directeurs de thèse
Prof. M. Cord, rapporteur
Prof. A. Lucchi, rapporteur
Prof. A. Zamir, rapporteur

École
polytechnique
fédérale
de Lausanne

2023

Acknowledgements

As I look back to the last four years of my life, several people have made this journey what it is: *memorable*.

To my advisor, Prof. François Fleuret for being a great helpful advisor and boss. Discussions with him were enlightening and incisive, and I always came out with better clarity. He was kind and gave me the freedom to pursue directions that I found interesting, and motivated me when I faced failures. I aspire to be the *geek* he describes himself as.

To my co-advisor, Prof. Alahi, for all the support at the important stages of this thesis.

To my jury members, Profs. Frossard, Cord, Lucchi, and Zamir: for their astute comments on the thesis manuscript, and for an engaging thesis defense.

To my co-authors Evann, Florian, Thijs, and Martin, for the stimulating discussions that made this thesis possible! I also thank members of the machine learning group: Angelos, Suraj, Arnaud, Kyle, and Sepehr for making it a stimulating and fun group.

To friends at Martigny made my stay in a new environment very comfortable: DJ, Skanda, Pavan, Shantipriya, Apoorv, Angel, Ketan, Julian, Lakmal, Anjith, Anshul, Neha, Tilak, Nithesh, Fabio, Andrei, Hande and many many others. Special thanks go to Ravi for being a fantastic *colocotaire* and a dear friend. Idiap lunch group and badminton group have been tremendously important to me. The second half of my PhD was spent with great office mates Esau, Jakub, Skanda, and Ravi.

Idiap has been a great place to work and have fun, thanks to the support of the staff. I'm particularly thankful to the secretariat Laura, Sylvie, and Nadine for making the transition and stay in a foreign land exceptionally smooth.

To people from a previous life: Amit Kale, Amit Vaze, and Uday Kurkure for inspiring me to push for PhD even while working, Pavan, RK, Nagesh, and TPT gang for being good friends. To Anoop sir, for igniting my interest in research during my Master's. To Balu sir, whose infused love for mathematics and learning in me.

The greatest support came from my family, Manu, and my in-laws. They stand by me through very uncertain times. To Manu's love, the courage and support she gave me through all the tumultuous moments.

To Armasuisse for funding this thesis through grant 050-38.

Martigny, March 8, 2023 Teja

Abstract

The successes of deep learning for semantic segmentation can in be, in part, attributed to its scale: a notion that encapsulates the largeness of these computational architectures and the labeled datasets they are trained on. These resource requirements hinder the applicability of segmentation networks to scenarios where labeled data is expensive, or deployment conditions do not allow for large networks. This dissertation aims at assuaging these problems by (a) transferring the knowledge of trained networks to new domains without the need for labeled data, (b) improving the computational efficiency of segmentation transformers by a differential allocation of computation to input regions.

The first part of this dissertation focuses on reducing the amount of labeled data needed to train these models by transferring knowledge from existing datasets and bridging the domain gap between them. We tackle model adaptation, a problem where we adapt a source data trained segmentation network with only unlabeled data from the target domain by improving the network's *confidence* of predictions. Next, we study test-time adaptation, where the goal is to adapt to a plausible domain shift with access to only a batch of samples at inference time. To do so, we train the network to be *confident* and stable to input perturbations. Experimental results show that methods that improve parameter or input perturbation robustness largely compensate for the absence of source data in the adaptation process.

The second part of this dissertation is on the computational requirements of deep networks. We first present a method for patch pausing to improve the inference efficiency of segmentation transformers. Here, we stop processing input patches deemed to have been processed enough to produce an accurate segmentation. This determination is done by computing the network's *confidence* of segmentation at intermediate layers. We then focus on compute-aware evaluation methods for deep learning, focusing on optimizers. We argue that a fair assessment must include not only the performance obtained but also the cost of finding the hyperparameter configurations that result in that performance. An optimization algorithm that achieves good performance with relatively little tuning effort and computational cost is more valuable in practice than one that performs better, albeit only with more tuning. We conclude that, under our experimental setup, Adam is the most practical choice.

Keywords: semantic segmentation, model adaptation, test-time adaptation, efficient transformers, patch pausing, performance evaluation.

Résumé

Les succès de l'apprentissage profond dans la segmentation sémantique peuvent être en partie attribués à sa taille : une notion qui inclut l'étendue de ces architectures computationnelles et des ensembles de données étiquetées sur lesquels elles sont entraînées. Ces exigences en termes de ressources limitent l'applicabilité des réseaux de segmentation dans des situations où les données étiquetées sont coûteuses ou où les conditions de déploiement ne permettent pas l'utilisation de grands réseaux. Cette thèse vise à résoudre ces problèmes en (a) transférant les connaissances des réseaux entraînés à de nouveaux domaines sans avoir besoin de données étiquetées, (b) améliorant l'efficacité computationnelle des transformeurs de segmentation en allouant de manière différentielle le calcul à certaines régions des données d'entrée.

La première partie de cette thèse se concentre sur la réduction de la quantité de données étiquetées nécessaires à l'entraînement de ces modèles en transférant des connaissances à partir de jeux de données existants, en comblant le fossé entre les domaines. Nous abordons l'adaptation des modèles, un problème où nous adaptons un réseau de segmentation entraîné sur des données source avec uniquement des données non étiquetées du domaine cible en améliorant la *confiance* du réseau dans ses prédictions. Ensuite, nous étudions l'adaptation au moment du test, où l'objectif est d'adapter un modèle à un changement de domaine plausible en n'ayant accès qu'à un lot d'échantillons au moment de l'inférence. Pour ce faire, nous entraînons le réseau à être *confiant* et stable face aux perturbations des données d'entrée. Les résultats expérimentaux montrent que les méthodes qui améliorent la robustesse aux perturbations des paramètres ou des entrées compensent largement l'absence de données sources.

La seconde partie de cette thèse porte sur les besoins computationnels. Nous présentons tout d'abord une méthode de pause de patches pour améliorer l'efficacité des transformeurs de segmentation. Ici, nous arrêtons de traiter les patches d'entrée qui ont déjà été suffisamment traités pour produire une segmentation précise. Cette détermination se fait en calculant la *confiance* du réseau en la segmentation produite dans les couches intermédiaires. Nous nous concentrons ensuite sur les méthodes d'évaluation pour l'apprentissage profond qui tiennent compte des besoins en calcul, en particulier dans le cas des optimiseurs. Nous soutenons qu'une évaluation juste doit inclure non seulement la performance atteinte mais aussi le coût de recherche des hyperparamètres qui permettent d'atteindre cette performance. Un algorithme d'optimisation qui obtient de bonnes performances avec un effort de réglage et un coût de calcul relativement faible a plus de valeur qu'un algorithme qui obtient de meilleures performances, mais au prix de plus de réglage. Nous concluons que, dans notre configuration expérimentale, Adam est le choix le plus pratique.

Keywords : segmentation sémantique, adaptation de modèle, adaptation en temps de test, transformeurs efficaces, pause de patches, évaluation des performances.

Contents

1 Introduction

Neural networks have long held great promise in their theoretical capabilities and have shown practical utility for the past decade, starting with the first application to large-scale classification (Ciresan et al., 2012; Krizhevsky et al., 2012), and have grown to revolutionize various other domains, including language processing, and speech analysis (Zhang et al., 2022a). These networks, trained on large datasets like the ImageNet (Deng et al., 2009), learn transferable feature representations that significantly improve the performance of other recognition tasks (Sharif Razavian et al., 2014). Recent empirical discoveries show that large networks and large datasets are primal to performance improvements (Zhai et al., 2022; Kaplan et al., 2020), and this is evidenced in Figure 1.1, where the recent trends show a nearly monotonic increase in the dataset sizes and the model sizes that have resulted in the performance improvements. Improvements in architecture have gone hand-in-hand with improvements to hardware like the Graphics Processing Units (GPU), and Tensor Processing Units (TPU), and these hardware improvements have, in turn, influenced the flavor of architectural improvements (Hooker, 2021).

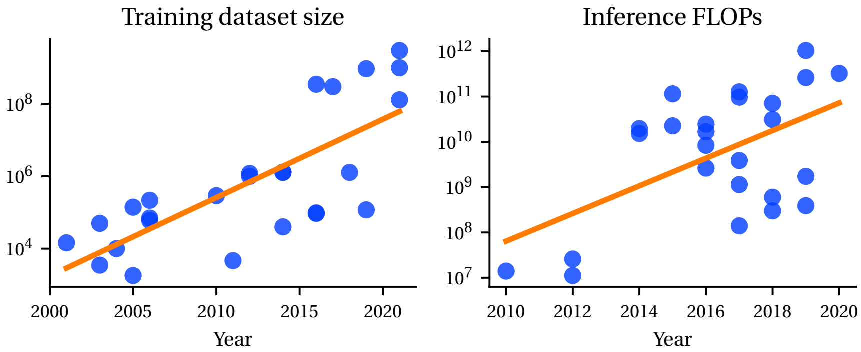


Figure 1.1: Dataset and model size trends for computer vision publications over the years. Datasets and model sizes have grown about four orders of magnitude since 2014. The performance improvements for vision problems are indubitably attributable to the large increase in dataset sizes and model improvements. Data from Sevilla et al. (2022).

These powerful feature extractors trained on ImageNet have been repurposed successfully for several downstream vision tasks, including the one of interest in this thesis – semantic segmentation. Segmentation refers to splitting the image into coherent parts. Several definitions of coherence have been studied, like color and spatial coherence (Achanta et al., 2012), and texture (Malik et al., 2001). In this thesis, we focus on one type where these parts are consistent with object categories present in the image or the semantics of the image; hence termed *Semantic Segmentation*. In the rest of the thesis, we use segmentation and semantic segmentation interchangeably.

1.1 On data requirements to train segmentation networks

Segmentation networks are trained using a standard supervised training learning: given a pixel-wise labeled training dataset, the network's parameters are modified such that the network can predict the labels of the training set as well as possible. These large computation machines require vast amounts of data to be trained adequately. A few large densely labeled datasets such as Cityscapes (Cordts et al., 2016), or Berkeley Deep Drive (BDD) (Yu et al., 2020) exist and generally cover a small geographic region; Cityscapes was collected in Germany and Switzerland, and BDD in San Francisco area. Networks trained on these datasets and deployed in other places or lighting conditions perform poorer than expected (Chen et al., 2017; Dai and Van Gool, 2018), as it violates the key assumption of test and train data being sampled from the same distribution (Recht et al., 2019; Hendrycks and Dietterich, 2019; Jo and Bengio, 2017).

This difference between train (source domain) and test (target domain) is termed *domain shift*, and the way to tackle it is dependent on the available information. See Table 1.1 for a concise representation.

- When the domain shift between train and test is known, and it is possible to collect labeled data from the target domain, transfer learning is performed. *Transfer learning*, or *fine-tuning*, has become the de facto standard in language processing, where a large model is trained on a large corpus possibly unrelated to the current problem, and a small labeled set is used to tailor the network the current problem (Ruder et al., 2019).

- When labeled data from the target domain is costly to obtain, *Unsupervised Domain Adaptation* (UDA) techniques are commonly used. UDA methods use labeled source data and unlabeled target data for the adaptation process and have been prominent in semantic segmentation (Toldo et al., 2020).

- UDA methods, while being appropriate for cases where obtaining labeled target data is expensive, require source data for the adaptation process. This can be an unreasonable expectation when source data cannot be shared, for instance, for legal reasons, like in medical images. *Model adaptation* (MA) adapts a source-trained model using unlabeled target data and is a relevant problem for these practical reasons. We study this in Chapter 2.

Table 1.1: We show the various scenarios proposed to handle the domain change problem. $\mathscr{S} = \{(x_i, y_i)\}_{i=1}^{N_s}$ $\mathscr{T} = \{(x_i, y_i)\}_{i=1}^{N_t}$ are the labeled source and target data, and $\mathscr{S}_x$ and $\mathscr{T}_x/\mathscr{T}_{x'}$ are the unlabeled data from the source, and target domains, $\mathscr{L}(\cdot)$ denotes a generic applicable loss function i.e., $\mathscr{L}$that takes both data and labels can be construed to be a supervised loss, and one that takes only data is an unsupervised loss. A testing process with '-' denotes that the testing procedure does not have steps other than the forward propagation of the data sample. In this thesis, we focus on the last two rows, i.e., model adaptation, and test time adaptation. A similar table has been presented elsewhere (Wang et al., 2021a; Teja and Fleuret, 2021a).

Strategy	Train payload	Test data	Training process	Testing process
Baseline	$\mathscr{S}$	$\mathscr{T}_x$	$\mathscr{L}(\mathscr{S})$	-
Transfer learning	$\mathscr{S}$ & $\mathscr{T}$	$\mathscr{T}_x$	$\mathscr{L}(\mathscr{S}) + \mathscr{L}(\mathscr{T})$	-
Fine-tuning	$\mathscr{T}$ & $f(\cdot, \theta_S)$	$\mathscr{T}_x$	$\mathscr{L}(\mathscr{T})$	-
Unsupervised Domain Adaptation	$\mathscr{S}$ & $\mathscr{T}_{x'}$	$\mathscr{T}_x$	$\mathscr{L}(\mathscr{S}) + \mathscr{L}(\mathscr{T}_{x'})$	-
Model Adaptation (Chapter 2)	$\mathscr{T}_{x'}$ & $f(\cdot, \theta_S)$	$\mathscr{T}_x$	$\mathscr{L}(\mathscr{T}_x)$	-
Test time adaptation (Chapter 3)	$f(\cdot, \theta_S)$	$\mathscr{T}_x$	-	$\mathscr{L}(\mathscr{T}_x)$

- The above scenarios require prior knowledge of the domain shift that will occur at test time and require data from the target domain at the train time. Some domain changes occur gradually due to changing underlying systems and cannot be premeditated, making data curation an impractical endeavor. For this reason, *test-time adaptation* (TTA) has become an important problem, where the goal is to tackle plausible domain shifts at test-time with access to a single batch of samples. We study this in Chapter 3.

Model adaptation (MA) and test-time adaptation (TTA) are two allied forms of source-free adaptation i.e., they do not use source data during the adaptation process (Fang et al., 2022). They differ in two key ways: 1) MA aims at generalizing to the target domain by learning from target domain training data, whereas TTA aims at performing better on the specific target set like transduction (Vapnik, 1998) and, 2) MA can use multi-epoch training, whereas TTA predicts the labels for the current test batch of samples, and generally does not revisit past test samples.

1.2 On computational requirements for segmentation networks

The largeness of the segmentation architectures translates to large training and inference resources; training resources are both computational and data, whereas inference resources are primarily computational, including memory. These requirements of these architectures prohibit their usability in several scenarios e.g. mobile devices, and low-power devices. In this thesis, we propose a method for improving the efficiency of segmentation networks at inference. This is orthogonal to techniques for faster training or reducing the amount of labeled training data.

The problem of efficient inference has been dealt with in several ways. One way has been to improve the efficiency of the architectural components or to simplify the architecture. Simplifications to the convolution layers include separable convolutions (Chollet, 2017) and rank-one decomposition (Jin et al., 2014). These have further led to architectural simplifications like the MobileNet family (Howard et al., 2017; Sandler et al., 2018; Howard et al., 2019). These improvements to classification networks have seeped into the segmentation applications (Hao et al., 2020, Section 3.1.7). Subsequent to the transformer (Vaswani et al., 2017) revolution, several efficient models have been proposed (Tay et al., 2022).

Other avenues for improving the efficiency of neural networks are network pruning and quantization. Pruning refers to removing network elements like weights or even layers. A network can be pruned post-training, though recent papers have found more utility in pruning through the training process (Kuzmin et al., 2019). Similarly, quantization of the network parameters and activation has been used to lower the memory footprint of networks at inference. Training methods for low bit widths have been investigated too (Wu et al., 2020).

The third avenue, and the one we pursue in this thesis, is reducing the amount of data that is processed by a neural network. To motivate this, consider the image classification task of dogs *vs.* cats. It is possible to classify an image as that of a dog with only the input regions that belong to the dog while refraining from processing the rest of the image. This reduction in the amount of input data processed results in better efficiency. This line of thinking is harder to realize using convolutional networks on modern GPUs but is suitable for transformers that treat input images as a sequence of patches, and has resulted in several works that investigated either dropping or combining patches (Meng et al., 2022; Yin et al., 2022; Bolya et al., 2023). We extend this to the problem of semantic segmentation.

1.3 Thesis Contributions and Outline

The four following chapters of this thesis are organized into two parts: the first part contains the work on source-free methods of dealing with domain shifts based on Teja and Fleuret (2021b,a), and the second part deals with computational constraints based on Courdier et al. (2022); Teja et al. (2020). The chapters can be read in any order and contain the relevant prior work.

Part 1: Dealing with domain changes

As seen in Table 1.1, traditional methods for Unsupervised Domain Adaptation (UDA) targeting semantic segmentation exploit information common to the source and target domains, using both labeled source data and unlabeled target data. We introduce the problem of model adaptation to semantic segmentation, where a network trained on the source data is adapted to the target domain using unlabeled target data. To tackle this problem, we propose a method that reduces the uncertainty of predictions on the target domain data. We accomplish this in two ways: minimizing the entropy of the predicted posterior, and maximizing the noise robustness of the feature representation. We show the efficacy of our method on the transfer of segmentation from computer-generated images to real-world driving images, and transfer between data collected in different cities, and surprisingly reach performance comparable with that of the methods that have access to source data. We present this in Chapter 2.

Model adaptation seeks to generalize a model to the target domain. We study a more specific case of this problem named test-time adaptation: we handle domain shift at inference time, i.e., we do not change the training process (remains training on source domain data) but quickly adapt the model at test time to handle any domain shift in the test samples. For this, we enforce consistency of predictions of data sampled in the vicinity of the test sample on the image manifold. We show experiments on tackling image corruptions and domain shifts and find that small network updates at inference time significantly improve performance on out-of-domain test data. We present this in Chapter 3.

Part 2: Dealing with computational needs

In Chapter 4, we study the problem of improving the efficiency of segmentation transformers by using disparate amounts of computation for different parts of the image. Our method, PAUMER, accomplishes this by pausing computation for patches that are deemed not to need any more computation before the final decoder. We use the entropy of predictions computed from intermediate activations of those patches as the pausing criterion and find this aligns well with the semantics of the image. Our method has a unique advantage: a single network trained with the proposed training strategy can be effortlessly adapted at inference to various run-time requirements by modulating its pausing parameters.

We then focus on the meta-problem of evaluation methods in deep learning and concentrate on optimizers. The efficacy of optimizers is often studied under near-optimal problem-specific hyperparameters, and finding these settings may be prohibitively costly for practitioners. In Chapter 5, we argue that a fair assessment of optimizers' performance must take the computational cost of hyperparameter tuning into account, i.e., how easy it is to find good hyperparameter configurations using an automatic hyperparameter search. Evaluating a variety of optimizers on an extensive set of standard datasets and architectures, our results indicate that Adam is the most practical solution, particularly in low-budget scenarios.

We conclude the thesis in Chapter 6 with unanswered questions from our research and some future directions.

Dealing with domain changes Part I

2 Uncertainty Reduction for Model Adaptation in Semantic Segmentation

This chapter is based on

> Teja, P. and Fleuret, F. (2021b). Uncertainty reduction for model adaptation in semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9613–9623.

2.1 Introduction

Domain changes between training and deployment are inevitable in several real-world problems and tackling this with fine-tuning requires producing annotated segmentation datasets for the deployment domain. Producing new segmentation datasets is very laborious and expensive; Cordts et al. (2016) reports that each image required about 90 minutes for labeling and verification. Thus, owing to the large costs, both time and economic, creating annotated datasets for each scenario is impractical, motivating the re-use or transfer of knowledge from available images to the requisite application.

The problem of transfer for segmentation has been predominantly investigated in literature (see Section 2.4) in two settings: adapting a model trained on synthetically generated images to natural images, and adapting a model to cities different from the ones it has been trained for. We, too, adopt these settings for study in this chapter. The synthetic-to-real images adaptation has attracted a lot of attention as the cost of generating segmentation ground truth for graphically rendered frames like GTA (Richter et al., 2016), or Synthia (Ros et al., 2016) is substantially lower. Richter et al. (2016) labeled 24,966 frames at an average of 7 seconds per frame, a significant drop from 90 minutes taken for Cityscapes. However, due to the nature of their generation, these synthetic images have a significant domain gap to the natural images, which results in a substantial drop in the performance of networks trained on synthetic data when used on natural images. Similarly, given the existing real-world datasets like Cityscapes, it is paramount that the knowledge learned on these datasets is effectively transferred to different scenarios without providing annotated data for each scenario.

A large portion of the methods dedicated to tackling this problem, like some reviewed in Section 2.4, require the labeled source data to be available along with the unlabeled target data for the adaptation process. We, in this chapter, focus on the problem where the source data itself is unavailable, but the source-trained classifier is. This is similar to life-long learning (Silver et al., 2013), where the goal is to adapt to several tasks over several domains, and the only information payload carried over is the model itself. Differing from that, we are not concerned with preserving the performance of the source task. The current problem of source data-less transfer is pertinent when data-sharing restrictions exist on the source data; a common way to circumvent this is to share the trained classifier from which the input data itself cannot be reconstructed. A classical application is in medical image processing, where patient data cannot be freely shared due to privacy concerns, but a trained model can be. Another relevant application where we envisage such a setting is in search-and-rescue operations, where data is collected on a mobile drone, and the segmentor network is adapted based on only the data collected without needing to access the original labeled dataset. Thus, the problem of domain adaptation in the absence of source data termed *model adaptation* (Chidlovskii et al., 2016) is of practical significance.

In this chapter, we study the problem of *model adaptation* for semantic segmentation. To the best of our knowledge, ours was the first work to do so. In the absence of source-labeled data that previous works have effectively exploited, we enforce auxiliary properties desirable in a system, namely confident predictions for the target data, and noise resilience, thereby increasing the stability of classification to parameter choices. To this end, we propose a method that uses feature corruption (Chidlovskii et al., 2016; Maaten et al., 2013; Ouali et al., 2020), and entropy regularization (Grandvalet and Bengio, 2005; Vu et al., 2019). We find that having access only to the source classifier, along with unlabeled target data, can result in performance comparable to the case where source data is also available.

2.2 Handling the absence of labeled source data

2.2.1 A toy problem

To motivate our method, we consider the ideal case depicted on Figure 2.1. Let us consider a simple case of binary classification of $Y = \{0, 1\}$ for a scalar input feature $x \in \mathbb{R}$. The probability of classification is defined using a sigmoid function of the input x and a threshold t

$$p(Y = 1|X = x; t) = \frac{1}{1 + e^{-(x-t)}} \tag{2.1}$$

For illustration in Figure 2.1, we show the class conditional distributions, though we do not use class information. With only the information of the feature distribution $\mu_X(x)$, our goal is to reason about scenarios that are likely to generalize better. If the labels are available, traditional wisdom tells us that Figure 2.1c is likely the ideal scenario to attain in terms of generalization (Schapire et al., 1998). We make it a little more concrete here, in the case where

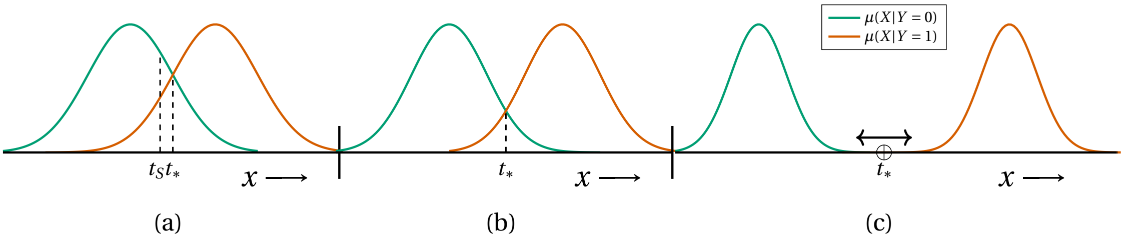


Figure 2.1: Given only the feature distributions in 2.1a to 2.1c in a two-class scenario, which one generalizes the best? In 2.1a, we show the distribution of features extracted by source trained network and the corresponding source threshold on the target data. Tuning the threshold to t_* from t_S is expected to result in better generalization. If we can modify the feature extractor itself, reducing the uncertainty of classification over the domain gives us 2.1b. This can be achieved by penalizing the entropy of predictions of each of the data points. We argue that while entropy is seemingly sufficient, we need to reduce the uncertainty in the network's predictions over a wide range of parameter choices to obtain better separation of the data like in 2.1c, and thereby better generalization. Details in Section 2.2.1.

the labels are not available.

In Figure 2.1a, the feature x is given by a source-trained feature extractor, and t_S is the corresponding learned threshold. It is apparent that such a threshold is likely ill-suited for the target domain, as it places the decision boundary in a high-density region of the feature space, contradicting the traditional cluster & continuity assumptions (Chapelle and Zien, 2005; Lee, 2013; Chapelle et al., 2010). However, if we change the threshold to t_*, we expect better generalization performance. Note that this is also the classifier for which the entropy of output probability predictions over the distribution $\mu_X(x)$ is the lowest. We mathematically define this idea and show numerical simulations for these cases in Appendix 2.B.

We would like to modify the feature extractor so that the class conditional distributions overlap less than in Figure 2.1a. This can be achieved by imposing an entropy penalty on output posterior predictions and using that penalty to train the feature extractor too. We show this in Figure 2.1b. As the overlap of the class-conditional data distributions decreases, we expect the generalization performance to improve.

The ideal scenario is shown in Figure 2.1c, where numerous thresholds can separate the two classes, and we choose one that results in the least uncertain predictions of target data. One can draw parallels to max-margin methods like the SVM, where one is interested in finding a separator that is optimally distant from all classes. In a nutshell, we see that a classifier with stable predictions for a range of parameter choices is likely to generalize better. This is, of course, in our context where we do not have access to labeled training data. To accomplish this, we need to go beyond entropy quantification of the predicted labels; we enforce stability of predictions over noisy features $x \pm \epsilon$.

2.2.2 Proposed method

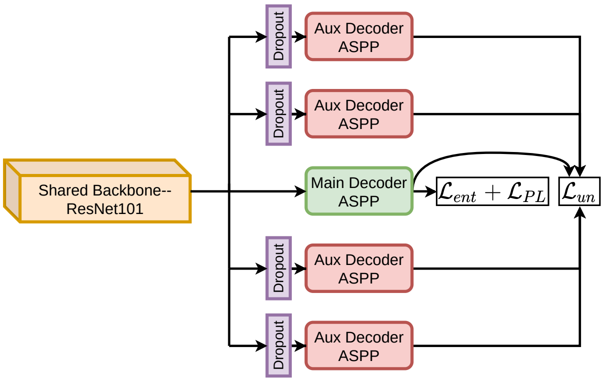


Figure 2.2: An illustration of the proposed method. The main decoder is trained with Equations (2.3) and (2.5), whereas the backbone feature extractor is trained with a combination of Equations (2.2), (2.3) and (2.5). At test time, we discard the auxiliary branches; thus, there is no additional computation at inference.

We, first, formally define the problem. Let $\mathcal{X} \in \mathbb{R}^D$ be the input, and $\mathcal{Y} \in \{1 \dots K\}$ the labels. Let $\mathcal{S}$ with density $\mu_S(\boldsymbol{x})$ be the source domain and $\mathcal{T}$ with density $\mu_T(\boldsymbol{x})$ be the target one, where we do not have access to the labels of $\mathcal{T}$ while training. Let $X_S = \{(\boldsymbol{x}_i, y_i)\ i = 1 \dots N_s, \boldsymbol{x}_i \sim \mu_S(\boldsymbol{x}), y_i \sim p_S(y)\}$ be the source data, and $X_T = \{(\mathbf{x}_i,)\ i = 1 \dots N_t, \boldsymbol{x}_i \sim \mu_T(\boldsymbol{x})\}$ be the target data. Let $p_T(y)$ be the target domain label prior that is unknown to us. Here $\mu_S \neq \mu_T$. We do not have direct access to X_S, but have access to a network trained on X_S. Let us denote that network by $f(\boldsymbol{x}, \boldsymbol{\theta_S}) \equiv f_S(\boldsymbol{x})$ where $\boldsymbol{\theta_S}$ denotes the parameters of the network trained on the source data. Let g and h denote the feature extractor and classifier respectively, whose composition is f i.e., $f = h \circ g$. Let also $\boldsymbol{\theta}_g$ and $\boldsymbol{\theta}_h$ be their corresponding parameters. In our case, the network g refers to the ResNet-101 backbone, and h refers to the ASPP (Chen et al., 2018) decoder, which we describe in Section 2.3.3. For convenience, h also subsumes the softmax layer, and thus $f(\boldsymbol{x}; \boldsymbol{\theta}) = p(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{\theta})$, where $\boldsymbol{y}$ is a vector of probabilities from which the predicted label is sampled. In summarizing, we have f_S and X_T, and our goal is to modify f_S such that its performance is improved on $\mathcal{T}$ data.

Optimizing the feature extractor to generate robust features

The likelihood of predictions $\boldsymbol{y}$ for an input of $\boldsymbol{x} \in \mathcal{T}$ is computed by $f(\boldsymbol{x}; \boldsymbol{\theta})$, on which an entropy penalty can be imposed (Grandvalet and Bengio, 2005). However, with a trivial application of this, for the illustration in Figure 2.1b, the network can learn to separate the distribution by placing the threshold t_* at any arbitrary point and shearing the feature distributions around it, thereby being stable to the choice of the threshold instead of attaining separation of features as in Figure 2.1c. For simple problems like the one in Figure 2.1, it can be achieved by enforcing stability to input perturbations. However, in deep networks, the network can learn to denoise the inputs in the initial few layers of processing. Using stronger augmentations like the ones in Chen et al. (2020) is ill-suited for segmentation, as classification networks are expected to be invariant to such noise, whereas segmentation networks are expected to be equivariant. This can be remedied by adding structured noise to inputs such that the layout of the input objects is preserved (for example, modifying the colors of objects). We achieve this by adding noise to the feature representation using dropout, similar to Ouali et al. (2020), that acts as a structured noise in the input space.

Let $\hat{\boldsymbol{y}}^i = f(\boldsymbol{x}; \hat{\boldsymbol{\theta}}_i)$ be the output of the network using the i^{th} instantiation of dropout, and $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$ the output for the network without dropout. In such a case, we propose to compute the *uncertainty loss* as

$$\mathscr{L}_{un} = \frac{1}{N} \sum_{i=1}^{N} \left(\hat{\boldsymbol{y}}^i - \boldsymbol{y} \right)^2 \tag{2.2}$$

To implement Equation (2.2), we introduce dropout only between g and h. In the context of Bayesian neural networks, such a method has been termed *LastLayer-Dropout* elsewhere (Ovadia et al., 2019). Instead of using a single branch that predicts the output with dropout weights, we use multiple decoders $\hat{h}$ that take in dropped-out features predicted by g. Thus $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta}) = h(g(\boldsymbol{x}; \boldsymbol{\theta}_g); \boldsymbol{\theta}_h)$ and $\hat{\boldsymbol{y}} = f(\boldsymbol{x}; \hat{\boldsymbol{\theta}}) = \hat{h}(g(\boldsymbol{x}; \boldsymbol{\theta}_g); \hat{\boldsymbol{\theta}}_h)$. Thus, each auxiliary decoder $\hat{h}$ sees only partial feature tensor and is required to come as close to reconstructing the original label tensor. We, experimentally, find that freezing the main decoders' weights $\boldsymbol{\theta}_h$ while training the auxiliary decoders' parameters $\hat{\boldsymbol{\theta}}_h$ results in better performance. Our full method is shown in Figure 2.2.

The use of multiple auxiliary decoders has been proposed in Meyerson and Miikkulainen (2018) for defining pseudo-tasks for deep multitask learning and in Ouali et al. (2020) for semi-supervised learning. Meyerson and Miikkulainen (2018), however, use ground-truth labels to train each of the auxiliary classifiers. We note the similarities to several previous works that focus on generating robust representations using various kinds of feature corruptions (Chen et al., 2014; Globerson and Roweis, 2006) that are class agnostic, and ones that are class specific like guided cutout (DeVries and Taylor, 2017). However, we experimentally find that advanced forms of feature noising (like class-dependent noising or targeted cutout) do not work as well as our method. We hypothesize that it is due to the unreliable predictions of the source-trained network on target images.

The proposed uncertainty loss in Equation (2.2) improves the noise resilience of the network. In addition, we use an entropy regularizer that minimizes the entropy of the network's predictions, which results in better empirical performance.

$$\mathscr{L}_{ent} = \mathbb{H}\left[f(\boldsymbol{x}; \boldsymbol{\theta}) \right] \tag{2.3}$$

where, $\mathbb{H}[]$ is the entropy of the probability distribution over all K classes for an input $\boldsymbol{x}$. We hypothesize that this is because the dropout noise modifies the decision boundary given by the optimizer itself. Thus, it is beneficial to explicitly regularize the predictions given that estimate of weights. We find that we do not need a diversity enforcing loss, as specified in (Krause et al., 2010; Liang et al., 2020), to prevent degeneracies.

Regularizing using the source trained model

We note that the losses in Equations (2.2) and (2.3) do not use the information in the source-trained classifier, but enforce certain properties to be satisfied by the network on the target domain. In Figure 2.1c, an interchanged labeling i.e., where class-0 is predicted class-1 and vice-versa, results in the same loss value for Equations (2.2) and (2.3). To avoid such issues and to infuse plausible class structure to the data, we use pseudo-labeling.

Pseudo-labeling or self-training has been a mainstay method in semi-supervised problems before the deep learning era. However, owing to the availability of large-scale datasets, it has been used to great success (Arazo et al., 2020; Lee, 2013; Xie et al., 2019) for several classification problems. Traditional methods use the class with the highest predicted probability as the ground truth for each unlabeled sample. However, in the case of a domain change, the accuracy of such predicted pseudo-labels is low. So we use the following modification to the standard definition

$$\boldsymbol{y}_{PL} = \begin{cases} \arg\max f(\boldsymbol{x},\boldsymbol{\theta}) & \text{if} \max(f(\boldsymbol{x},\boldsymbol{\theta})) \geq \tau \\ \text{IGNORE} & \text{otherwise} \end{cases} \tag{2.4}$$

i.e., we only consider as pseudo-labels the samples that are at least τ confident. The samples with the *IGNORE* label do not contribute to the loss. However, choosing τ is a non-trivial task, as too low a threshold will result in wrong labels, and too high a threshold will result in no target data being bootstrapped for training. In this work we adopt the strategy of class balanced thresholding (Zou et al., 2018), where τ is varied per class, such that a certain proportion of points per class are always selected. We define the pseudo-labeling loss to be the cross-entropy loss with the pseudo labels as defined in Equation (2.4)

$$\mathscr{L}_{PL} = -\mathbb{1}_{\boldsymbol{y}_{PL}}^{T} \log(\boldsymbol{y}) \tag{2.5}$$

where $\mathbb{1}_{\boldsymbol{y}_{PL}}$ is one-hot encoded vector of $\boldsymbol{y}_{PL}$, and log is applied element-wise.

Thus, the overall loss function being optimized is the combination of Equations (2.2), (2.3) and (2.5):

$$\mathscr{L} = \mathscr{L}_{PL} + \lambda_{ent}\mathscr{L}_{ent} + \lambda_{un}\mathscr{L}_{un} \tag{2.6}$$

where $\lambda_{ent}, \lambda_{un}$ are the weights of the individual loss terms.

Our work is connected to recent work in interesting ways: if Equation (2.2) is construed to be a form of self-supervision, our method can be interpreted as a form of *test-time training* (Sun et al., 2020). *Test-time training* proposes to use an auxiliary task at test time that helps combat domain shift from the training set. We differ in that we do not update the network at test time but do so when given target domain data. Similarly, our work, conceptually, uses self-supervision for domain adaptation, similar to (Sun et al., 2019), but doesn't need access to source labeled data. Additionally, our method can be interpreted as making the network a *bit Bayesian* (Kristiadi et al., 2020), where instead of placing Gaussian posterior on the weights of the penultimate layer's weights, we use dropout distribution. As previously mentioned, our method has similarities to pseudo-tasks in multitask learning (Meyerson and Miikkulainen, 2018), which uses labeled data for training.

2.3 Experiments

2.3.1 A toy problem

To elucidate the utility of each of the terms in Equation (2.6), we use a toy problem as shown in Figure 2.3. In Figure 2.3 a & b we show the source and target datasets; we use a rotated version of the source data as the target data. In our case, we do not use the labels of the target data. We train a small two-layer neural network with batch norm and ReLU activations. We provide the exact architecture in Table 2.A.1 in the appendix. We take the outputs before the last classifier layer as the features of the network and use the techniques that we described in Section 2.2.2 to train the network, except we do not use pseudo labeling for this problem. We use a feature dimensionality of 2 and plot the target features in Figure 2.3c with the source classifier. It is very apparent that the source network extracts features that do not transfer well. In Figure 2.3d, with a simple entropy regularization on the target data, we see that the performance improves tremendously. However, some blue points are very close to the separating line. To remedy this, our proposed feature noise decoder (detailed in Section 2.2.2) pushes the points away from the separating line. This can be interpreted as a form of increasing the stability of the classification and thereby reducing uncertainty, which we hypothesize to be the key to better generalization.

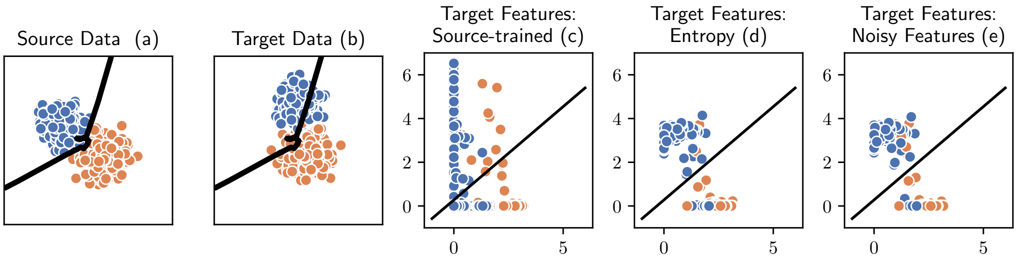


Figure 2.3: A toy example in $\mathbb{R}^2$ to illustrate our algorithm. The target data is unlabeled, however, we shade it for illustration. The thick line in all the figures is the separator. In (a) we show the source data and the classifier trained on that, and in (b) the unmodified classifier on the target data. In (c) we visualize the features extracted by the source-trained classifier on target data. With a simple entropy penalty on the target data, we get substantially better performance (d). With the additional uncertainty loss, we see that the features extracted are pulled further away from the separating line in (e). Details in Section 2.3.1.

2.3.2 Datasets and Evaluation

We demonstrate the efficacy of our method on the standard domain adaptation tasks of GTA (Richter et al., 2016)→ Cityscapes (Cordts et al., 2016) (*GTA-CS*) and Synthia (Ros et al., 2016)→ Cityscapes (*SYN-CS*) and Cityscapes→NTHU Crosscity (Chen et al., 2017) (*CS-CC*), a standard test setting used in several previous works (Section 2.4). Cityscapes consists 2975 annotated images, each of size 2048 × 1024, that act as our training set. It has 500 images as the validation set, which we use to benchmark our method. It consists of 19 semantic classes for segmentation. The GTA dataset consists of 24966 frames, of size 1914 × 1052 grabbed from the famous game Grand Theft Auto. The ground truth is generated by the game renderer itself. It shares the same 19 semantic classes as Cityscapes. Synthia has 9400 images of size 1280 × 760 synthetic images, and shares 16 classes with Cityscapes. For our method, we use a network trained on Synthia or GTA, and adapt it to Cityscapes using the 2975 training images without their ground-truth labels. Crosscity dataset has been recorded in four cities: Rome, Rio, Taipei, and Tokyo with each image of resolution 2048×1024. Following (Chen et al., 2019a), we use their experimental setup of 3200 unlabeled images as target training data, and 100 labeled images as target test data. This adaptation task has 13 shared classes. We use the pre-trained models provided by Chen et al. (2019a) for the source-trained model.

To evaluate our method, we use Intersection-over-Union (IoU) for each class and its average mean-Intersection-over-Union (mIoU) over all classes. We report the metrics for all the 19 classes of *GTA-CS* adaptation, the 16 common classes for the *SYN-CS* experiments, and the 13 common classes in the *CS-CC* experiments. In accordance with some recent papers, we also report a mIoU* comparing only 13 classes for the Synthia to Cityscapes adaptation task.

2.3.3 Implementation details

To facilitate a fair comparison with relevant works, we use a DeepLab V2 network (Chen et al., 2018) with a ResNet-101 backbone (He et al., 2016a). Using the notation defined in Section 2.2, g is the ResNet-101-based feature extractor, and h is the Atrous Spatial Pyramidal Pooling (ASPP) decoder. The ResNet 101 backbone is pre-trained on ImageNet. ASPP (Chen et al., 2018) is a multiscale decoder used to aggregate multiscale information for segmentation. It has 4 parallel atrous convolutions of various rates, which capture long-range contextual information from extracted features. We train the model on the source domain with stochastic gradient descent with a learning rate of 2.5×10^{-4} with a weight decay of 0.0005, momentum of 0.9. We use a poly learning rate decay scheduler with a power of 0.9. For the adaptation experiments, we use a lower learning rate of 5×10^{-5}, with other parameters staying the same, with no learning rate decay. We also use a 10× the learning rate for ASPP decoders (Chen et al., 2018) in our experiments. For all our experiments, we use $\lambda_{\text{ent}} = 1.0$, $\lambda_{un} = 0.1$. We use the defaults prescribed by Zou et al. (2019) to extract class-balanced pseudo labels. For the *CS-CC* experiments, we run the adaptation for only 2 epochs, and for *GTA-CS*, *SYN-CS* for 6 epochs.

Table 2.1: For all the experiments in Tables 2.1 to 2.3 we compare our proposed method with methods that use source data for adaptation. We find that our method is comparable, and in some cases better than the methods that use the source data for adaptation. In underline, we compare our results to the source trained classifier, and with a **bold** the best performance over all methods. We omit the underline if our proposed method, or the source classifier outperforms the methods that use the source data for adaptation. Here, we show results of GTA5 → Cityscapes (*GTA-CS*) domain adaptation.

Uses source data	Method	Road	Sidewalk	Building	Wall	Fence	Pole	Tra. light	Tra. Sign	Veg.	Terrain	Sky	Person	Rider	Car	Truck	Bus	Train	Motorbike	Bike	mIoU
Yes	AdaptSegNet (Tsai et al., 2018)	86.5	36.0	79.9	23.4	23.3	23.9	**35.2**	14.8	83.4	33.3	75.6	58.5	27.6	73.7	32.5	35.4	3.9	30.1	28.1	42.4
	AdvEnt (Vu et al., 2019)	89.4	33.1	81.0	26.6	**26.8**	27.2	33.5	**24.7**	83.9	**36.7**	78.8	**58.7**	**30.5**	**84.8**	**38.5**	**44.5**	1.7	**31.6**	32.4	45.5
	FCAN (Zhang et al., 2018)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	46.6
	CBST (Zou et al., 2018)	91.8	53.5	80.5	32.7	21.0	34.0	28.9	20.4	83.9	34.2	80.9	53.1	24.0	82.7	30.3	35.9	16.0	25.9	**42.8**	45.9
	MRKLD (Zou et al., 2019)	91.0	**55.4**	80.0	**33.7**	21.4	**37.3**	32.9	24.5	**85.0**	34.1	80.8	57.7	24.6	84.1	27.8	30.1	**26.9**	26.0	42.3	**47.1**
	LRENT (Zou et al., 2019)	91.8	53.5	80.5	32.7	21.0	34.0	29.0	20.3	83.9	34.2	80.9	53.1	23.9	82.7	30.2	35.6	16.3	25.9	**42.8**	45.9
	Source	71.3	19.2	69.1	18.4	10.0	35.7	27.3	6.8	79.6	24.8	72.1	57.6	19.5	55.5	15.5	15.1	11.7	21.1	12.0	33.8
No	Our method	**92.3**	55.2	**81.6**	30.8	18.8	**37.1**	17.7	12.1	84.2	35.9	**83.8**	57.7	24.1	81.7	27.5	44.3	6.9	24.1	40.4	45.1

Table 2.2: Results of Synthia → Cityscapes (*SYN-CS*) domain adaptation. See Table 2.1 for details.

Uses source data	Method	Road	Sidewalk	Building	Wall*	Fence*	Pole*	Tra. Light	Tra. sign	Veg.	Sky	Person	Rider	Car	Bus	Motorbike	Bike	mIoU	mIoU*
Yes	AdaptSegNet Tsai et al. (2018)	84.3	**42.7**	77.5	–	–	–	4.7	7.0	77.9	82.5	54.3	21.0	72.3	32.2	18.9	32.3	–	46.7
	AdvEnt Vu et al. (2019)	**85.6**	42.2	**79.7**	8.7	0.4	25.9	5.4	8.1	80.4	**84.1**	57.9	23.8	73.3	**36.4**	14.2	33.0	41.2	48.0
	CBST (Zou et al., 2018)	68.0	29.9	76.3	10.8	1.4	33.9	22.8	29.5	77.6	78.3	60.6	28.3	81.6	23.5	18.8	39.8	42.6	48.9
	MRKLD (Zou et al., 2019)	67.7	32.2	73.9	10.7	1.6	**37.4**	22.2	31.2	80.8	80.5	**60.8**	**29.1**	**82.8**	25.0	19.4	**45.3**	**43.8**	**50.1**
	LRENT (Zou et al., 2019)	65.6	30.3	74.6	13.8	1.5	35.8	**23.1**	29.1	77.0	77.5	60.1	28.5	82.2	22.6	**20.1**	41.9	42.7	48.7
	Source	64.3	21.3	73.1	2.4	1.1	31.4	7.0	27.7	63.1	67.6	42.2	19.9	73.1	15.3	10.5	38.9	34.9	40.3
No	Our method	59.3	24.6	77.0	**14.0**	**1.8**	31.5	18.3	**32.0**	**83.1**	80.4	46.3	17.8	76.7	17.0	18.5	34.6	39.6	45.0

Table 2.3: Results of Cityscapes → Cross-City (*CS-CC*) experiments. See Table 2.1 for details.

City	Uses source data	Method	Road	Sidewalk	Building	Tra. Light	Tra. Sign	Veg.	Sky	Person	Rider	Car	Bus	Motorbike	Bike	mIoU
Rome	Yes	Cross city (Chen et al., 2017)	79.5	29.3	84.5	0.0	22.2	80.6	82.8	29.5	13.0	71.7	37.5	25.9	1.0	42.9
		MaxSquare (Chen et al., 2019a)	82.9	32.6	86.7	**20.7**	**41.6**	85.0	**93.0**	47.2	**22.5**	**82.2**	53.8	**50.5**	9.9	**54.5**
	No	Source	85.0	34.7	86.4	17.5	39.0	84.9	85.4	43.8	15.5	81.8	46.3	38.4	4.8	51.0
		Our Method	**86.2**	**39.1**	**87.6**	14.3	37.8	**85.5**	88.5	**49.9**	21.9	81.6	**56.3**	40.4	**10.4**	53.8
Rio	Yes	Cross city	74.2	43.9	79.0	2.4	7.5	77.8	69.5	39.3	10.3	67.9	41.2	27.9	10.9	42.5
		MaxSquare	76.9	48.8	**85.2**	13.8	18.9	**81.7**	**88.1**	**54.9**	**34.0**	76.8	39.8	**44.1**	**29.7**	53.3
	No	Source	74.2	42.2	84.0	12.1	20.4	78.3	87.9	50.1	25.6	76.6	40.0	27.6	17.0	48.9
		Our Method	**82.8**	**57.0**	84.8	**17.4**	**24.0**	80.5	86.0	54.2	27.7	**78.2**	**43.8**	38.3	21.5	**53.5**
Tokyo	Yes	Cross city	83.4	35.4	72.8	12.3	12.7	77.4	64.3	42.7	21.5	64.1	**20.8**	8.9	40.3	42.8
		MaxSquare	81.2	30.1	77.0	12.3	**27.3**	**82.8**	**89.5**	**58.2**	**32.7**	**71.5**	5.5	**37.4**	**48.9**	**50.5**
	No	Source	81.4	28.4	**78.1**	**14.5**	19.6	81.4	86.5	51.9	22.0	70.4	18.2	22.3	46.4	47.8
		Our Method	**87.1**	**38.3**	77.2	13.7	24.4	82.6	86.9	54.1	28.0	69.6	18.5	19.2	48.0	49.8
Taipei	Yes	Cross city	78.6	28.6	80.0	13.1	7.6	68.2	82.1	16.8	9.4	60.4	34.0	26.5	9.9	39.6
		MaxSquare	80.7	32.5	85.5	**32.7**	15.1	78.1	**91.3**	**32.9**	7.6	69.5	44.8	52.4	**34.9**	**50.6**
	No	Source	82.6	33.0	**86.3**	16.0	**16.5**	**78.3**	83.3	26.5	8.4	70.7	36.1	47.9	15.7	46.3
		Our Method	**86.4**	**34.6**	84.6	22.4	9.9	76.2	88.3	32.8	**15.1**	**74.8**	**45.8**	**53.3**	26.7	50.1

2.3.4 Results

We show the performance of our proposed method on the tasks of *GTA-CS* adaptation in Table 2.1, *SYN-CS* adaptation in Table 2.2, and *CS-CC* adaptation in Table 2.3. We obtain at-par or better results than some of the classic works on unsupervised domain adaptation (with source data). This work is, however, not a claim that source data is not necessary for domain adaptation. The utility of source data has been exploited effectively by recent methods through style transfer techniques (Yang and Soatto, 2020; Kim and Byun, 2020), thus, they obtain higher performance than us. Our proposed method improves substantially on the larger background classes, and we hypothesize this is because the source classifier can make reliable predictions for these classes. The perceptual domain gap also seems to influence the transfer performance, as we get comparable performance for *GTA-CS* and *CS-CC* experiments, compared to the results we obtain for *SYN-CS* experiments.

Importance of loss terms

To correctly attribute performance to each of the terms in Equation (2.6), we ablate over the loss terms in Table 2.4. Broadly summarizing, we find that the first choice of pseudo-labeling results in a substantial improvement in performance over the source classifier. With each term, we see that the performance increases, however, as described in Section 2.3.4, our method also suffers higher variance. Thus, for a new use case, one might expect a small but consistent improvement with pseudo-labeling, and other loss terms are useful if the method can be tuned carefully. We show various additional ablations in Appendix 2.D and some qualitative results in Appendix 2.E.

Table 2.4: Importance of each of the loss terms proposed. $\mathscr{L}_{DT}$ refers to training with loss in Equation (2.6) without freezing the decoders, and the last column $\mathscr{L}$ shows the performance on freezing the main decoder. We see that each of the loss terms gives a consistent improvement over the previous loss values.

Loss function	$\mathscr{L}_{PL}$	$\mathscr{L}_{ent}$	$\mathscr{L}_{ent} + \mathscr{L}_{PL}$	$\mathscr{L}_{un} + \mathscr{L}_{PL}$	$\mathscr{L}_{DT}$	$\mathscr{L}$
	Eq 2.5	Eq 2.3	Eq 2.3 + Eq 2.5	Eq 2.2 + Eq 2.5	Eq 2.6	Eq 2.6
% mIoU	42.24	19.85	42.39	42.72	44.52	45.07

Variance analysis

To obtain a better estimate of performance, we run some of the baseline methods that we use in Table 2.1 with five random seeds and show the performance. We use the publicly available codes from the authors. The codes were executed for a maximum of 72 hours, a limitation imposed by our computation resources. For each method, we change only the random seed for each run and leave the rest of the hyperparameters to the default values set by the authors in their codes. In Table 2.5, we show various statistics computed over the obtained runs. We

Table 2.5: Variance of the methods examined for the GTA→ Cityscapes. We show the mean, standard deviation, minimum and reproduced (maximum) performance obtained over five runs with different random seeds, and the official reported metrics from the paper. For ADVENT, we show two rows to indicate the two testing strategies in their code: The first one is after 90K iterations, and the second is the best-attained performance. We see that the common strategy is to report the best-obtained result.

Method	Performance estimate	Min	Reproduced Results	Previously Reported
AdaptSegnet (Tsai et al., 2018)	39.68 ± 1.49	37.70	42.20	42.40
ADVENT (Vu et al., 2019) (90K)	41.57 ± 0.73	40.73	42.73	43.80
ADVENT (Vu et al., 2019) (Best)	42.56 ± 0.64	41.60	42.39	
CBST (Zou et al., 2018)	44.04 ± 0.88	42.80	45.03	45.90
Proposed Method	42.44 ± 2.18	39.71	45.06	–

Table 2.6: Variance of proposed method for the Cityscapes→NTHU Crosscity adaptation.

	Rome	Rio	Tokyo	Taipei
Performance estimate	53.2± 0.8	52.37± 1.08	49.2± 0.71	49.48±0.76

see that the common trend in publications on UDA is to report the best-obtained performance. While it is a pragmatic choice to use the best-obtained model as benchmarked on a validation set, for deployments, it induces a systemic bias in the assessment of the true performance of the system. Thus, we believe that a better characterization of the system's performance is through computing the average performance and the standard deviation, in addition to the best performance obtained.

In Table 2.5, we find that merely changing the random seed can have a noticeable effect on the performance of some standard systems, an observation made before (Madhyastha and Jain, 2019; Bengio, 2012). The discrepancy between the maximum results in Table 2.5 and the reported numbers can be attributed to hardware and software discrepancies, or budget used. Vu et al. (2019) also remark that one needs to run the experiments a few times to reach comparable performance[1]. Keeping in line with the standards of the domain, we report the best-obtained performance in Tables 2.1 to 2.3, and show results of variance analysis in Tables 2.5 and 2.6. Examining the mean and standard deviation obtained for our *GTA-CS* and *CS-CC* experiments, we find that while our method achieves higher maximum performance, it has a higher variance compared to the methods that use source data for the adaptation process. We hypothesize that the proxy tasks used cannot act as suitable replacements for

[1] https://github.com/valeoai/ADVENT#training

labeled data, in controlling and guiding the optimization process. We leave this analysis to future work.

2.4 Related Work

Semantic Segmentation

Deep learning had its success in semantic segmentation with fully convolutional networks (Long et al., 2015), which converted the full-connected layers to convolutional layers. Following this, various networks that made several architectural changes to improve accuracy metrics (Yu et al., 2017; Zhao et al., 2017; Chen et al., 2018), and computational requirements (Zhao et al., 2018; Paszke et al., 2016; Romera et al., 2017; Orsic et al., 2019) have been proposed. To remedy the data hungriness of these networks, domain adaptation, specifically unsupervised domain adaptation, has been an oft-studied problem recently

Unsupervised Domain Adaptation

Based on the seminal work of adversarial domain adaptation (Ganin et al., 2016) that uses a discriminator network to align features from both domains, several methods have been proposed for segmentation on similar lines. Methods have been proposed that align intermediate feature spaces (Hoffman et al., 2016; Sankaranarayanan et al., 2018; Du et al., 2019; Wang et al., 2020), output space (Tsai et al., 2018, 2019).

Another family of methods uses pseudo-labeling to enrich the target domain training using either hard labels (Zou et al., 2018; Dai et al., 2019; Zhang et al., 2019) or soft labels (Zou et al., 2019). Chang et al. (2019) use a *per-domain* autoencoder to separate domain idiosyncrasies from features relevant to cross-domain segmentation. Vu et al. (2019); Chen et al. (2019a) propose an entropy-based objective, and an adversarial alignment of the entropy maps of the two domains. This method has been extended to multi-domain adaptation (Saporta et al., 2021).

A host of techniques includes style translation as a part of their network that is trained along with the segmentation network (Hoffman et al., 2018; Li et al., 2019; Chen et al., 2019b; Murez et al., 2018), or separately (Yang and Soatto, 2020; Kim and Byun, 2020).

Model Adaptation

Most of the previously mentioned methods need the explicit availability of source domain data during adaptation too and have made tremendous strides in improving the segmentation performance in that case. A few recent papers tackle model adaptation for classification problems (Liang et al., 2020; Li et al., 2020; Chidlovskii et al., 2016). (Kundu et al., 2020) proposes source-free domain adaptation in the case where label knowledge of the target domain is not available, and shows their efficiency on a set of classification problems with varying levels of label overlap. As we argued in Section 2.1, to the best of our knowledge, this problem has not been tackled in the context of semantic segmentation. Though the task of segmentation can be construed to be one that of pixel-wise prediction, we emphasize that the techniques cannot be interchangeably used. As the resource requirements of segmentation networks are substantially higher than that of classification networks, methods proposed for classification that use conditioned generation of target domain data (Li et al., 2020), or use memory-intensive methods for reliable estimation of pseudo-labels (Kim et al., 2020; Liang et al., 2020) are impractical.

Self-supervision for segmentation

Self-supervised learning exploits the structure in data by defining a *pretext task* so that the network learns a good semantic representation of the input image. Examples include rotation prediction (Gidaris et al., 2018), context prediction (Doersch et al., 2015), jig-saw puzzle solving (Noroozi and Favaro, 2016),contrastive learningChen et al. (2020); He et al. (2020). Self-supervised learning for unsupervised domain adaptation has been proposed to train the feature extractor (Sun et al., 2019). However, the applications of self-supervised learning to segmentation have been limited, as the invariances enforced differ widely between classification and segmentation problems. Some attempts to learn segmentation networks include using very strong perturbation techniques like CutMix (French et al., 2019), using consistency regularization through feature noising (Ouali et al., 2020), by using clustering for pseudo-labels (Larsson et al., 2019). As described in Section 2.2.2, we note similarities of our method to tasks proposed in (Ouali et al., 2020).

Multitask learning and robust classification

The use of multiple tasks to enrich the representations learned by a network has been used quite often in computer vision (Zhang et al., 2014; Bilen and Vedaldi, 2016). These ideas have been adapted to single-task learning by devising *pseudo-tasks* (Meyerson and Miikkulainen, 2018). These pseudo-tasks train the shared network structure to learn the task in multiple ways and can be interpreted as using self-supervision. Similarly, increasing the robustness of classification through feature noising has been studied extensively (Globerson and Roweis, 2006; Chechik et al., 2008; Vincent et al., 2010; Chidlovskii et al., 2016; Maaten et al., 2013). Work on learning robust networks through pseudo-ensembling by reducing the variance when dropout is used has been proposed (Bachman et al., 2014). Ideas of large-margin learning have been extended to deep learning (Elsayed et al., 2018).

Uncertainty modeling for neural networks

Our proposed method in Section 2.2.2 is very similar to the uncertainty modeling for neural networks using Monte-Carlo (MC) dropout. MC-dropout (Kendall and Gal, 2017; Gal and Ghahramani, 2016) uses dropout at test-time to sample various outputs and average them for predicted posterior. Bayesian treatments of neural networks have been important owing to their ability to predict uncertainty better and dealing with miscalibration problems (Kristiadi et al., 2020). Such modeling has been used for medical imaging to estimate the confidence of lesion segmentation (Nair et al., 2020), and for autonomous driving to estimate the uncertainty of steering wheel angle prediction (Loquercio et al., 2020).

2.5 Conclusion

In this chapter, we focused on the problem of domain adaptation for semantic segmentation in the absence of source data. In the absence of labels to guide the optimization, we propose a method that reduces the uncertainty of predictions on the target domain data, which can also be interpreted as increasing the stability of the feature extractor. We obtain performance comparable to methods that use source data on the standard benchmark of tasks for semantic segmentation transfer.

Appendix - Chapter 2

Appendix 2.A Toy example's network architecture

For the toy experiment presented in Section 2.3.1, we use a network from Kristiadi et al. (2020). The architectural details of the network are given in Table 2.A.1.

Table 2.A.1: Architecture of the network used for the toy experiment in Figure 2.3

Layer name	**Description**
Feature Extractor	Linear 2×20 BatchNorm ReLU Linear 20×2 ReLU
Classifier	Softmax(2)

Appendix 2.B Entropy measurements

In Figure 2.1 and in Section 2.2.1, we glossed over the details of the toy problem. We provide the details here.

Referring to Figure 2.1, we are trying to reason scenarios that, in the absence of labeled target data, can be expected to result in good target performance. Our argument is that reducing the uncertainty of predictions on the target domain is an effective strategy to improve performance on the target domain. In order to do so, we concoct toy scenarios and analyze their uncertainties. For this illustration, we use entropy as a measure of uncertainty.

Let us consider a two-class classification problem as shown in Figure 2.1. Let X be the feature random variable, and $Y \in \{0,1\}$ be the labels. Let us assume that the class conditional distributions be normally distributed i.e., $X|Y = k \sim \mathcal{N}(\mu_k, \sigma^2)$. Thus $\mu_X(x) = \frac{1}{2}\left(\mathcal{N}(x;\mu_0,\sigma^2) + \mathcal{N}(x;\mu_1,\sigma^2)\right)$, assuming uniform prior on Y. Let the threshold random variable T with a density $\mu_T(t)$. This distribution is determined by a learning algorithm. Let $\hat{Y}$ be the random variable denoting the predictions whose probability is computed using a sigmoid on the feature and a threshold as

$$\rho(x;t) \equiv p(\hat{Y}=1|X=x;T=t) = \frac{1}{1+e^{-(x-t)}}. \tag{2.7}$$

The entropy of the above-defined categorical distribution is given by

$$\mathbb{H}(\hat{Y}|X=x,T=t) = -\big(\rho(x;t)\log(\rho(x;t)) + (1-\rho(x;t))\log(1-\rho(x;t))\big) \tag{2.8}$$

The entropy of classification over the entire domain X can be computed as

$$\mathbb{H}(\hat{Y}|T=t) = \int -\big(\rho(x;t)\log(\rho(x;t)) + (1-\rho(x;t))\log(1-\rho(x;t))\big)\,\mu_X(x)dx \tag{2.9}$$

The above-defined marginal entropy is defined for a specific choice of the threshold t. To see how the feature distribution itself influences the generalization, the entropy over all possible thresholds is computed. The *total entropy* over all the thresholds possible is computed by integrating over the entire range of t.

$$\mathbb{H}(\hat{Y}) = \int\int -\big(\rho(x;t)\log(\rho(x;t)) + (1-\rho(x;t))\log(1-\rho(x;t))\big)\,\mu_X(x)\mu_T(t)dxdt \tag{2.10}$$

We emphasize that t is the set of thresholds that can be generated by a learning algorithm. However, we simplify it by using the domain X for this discussion.

We would like to train networks that result in low overall entropy in Equation (2.10). This coincides with our argument that the features that can be separated by several choices of threshold are likelier to generalize better. To visualize this, we run a few simulations that compute the marginal and total entropy for various settings of feature distributions. In Figure 2.B.1, we show the data distribution $\mu_X(x)$ for various μ_0, μ_1 values. Variance σ^2 is fixed to 1. In orange, we show marginalized entropy (Equation (2.9)). We scale it by a constant for plotting purposes.

It is very apparent that the higher is $|\mu_1 - \mu_2|$, the lower the overall entropy. Given that the two Gaussians are the class conditional distributions, the Bayes optimal decision boundary can be computed to be $\frac{\mu_1+\mu_2}{2}$, coincides with the point where Equation (2.9) is minimized, for all scenarios except for the one with high overlap in Figure 2.B.1(a). Extending it, the lower the Bayes risk, the better the generalization. Thus, the least entropy separation is also the best separator we can achieve. We skip an important detail here: by extending the threshold search to a value that is on the extremities of the X-axis, we can get a threshold that results in an overall entropy that is very low. However, such a separator is practically useless. In our method, we avoid this by using pseudo-labeling and initializing the network with the source-trained weights.

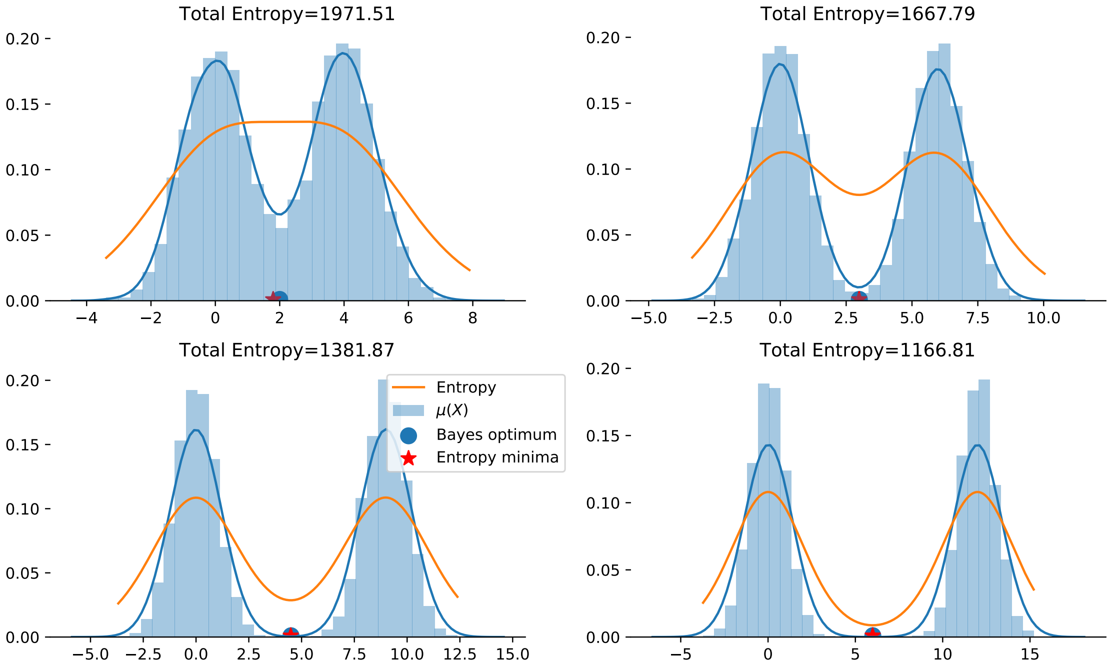


Figure 2.B.1: Illustration of the effect of the separation of the Gaussians on the entropy of the classification.

We can see that for this simple scenario minimizing the entropy is a fairly good strategy in the absence of labels to find near the optimal decision boundary.

Appendix 2.C Using squared error instead of entropy loss in Equation (2.2)

In Section 2.2.2, we proposed the uncertainty loss, and we use a squared error form instead of the standard entropy-based uncertainty quantification. Here we provide a plausible explanation that follows the argument in Chen et al. (2019a). Experimental evidence is provided in (Ouali et al., 2020).

The traditional entropy regularizer has been used extensively in various applications like semi-supervised learning (Grandvalet and Bengio, 2005), domain adaptation (Vu et al., 2019). However, the gradient of the entropy penalty with respect to the softmax output is not well-behaved for the probability is close to 1. We refer the reader to (Chen et al., 2019a, Figure 1) for an illustration of this. However, squared loss and entropy loss can be viewed as a special case of f-divergence.

Let P and Q be two distributions with PDFs p and q their density functions. Then their f-divergence is defined as

$$D_f(P||Q) = \int_{\mathbb{R}} f\left(\frac{p(x)}{q(x)}\right) q(x)dx \tag{2.11}$$

We get KL-divergence by using $f(t) = t\log t$. Instead, using $f(t) = (t-1)^2$ gives the Max-squared loss formulation in (Chen et al., 2019a). They find that using this instead of $f(t) = t\log t$ results in better behavior for optimization. We find a similar empirical result that using the squared error form for Equation (2.2) results in better results than using posterior entropy.

Appendix 2.D Experimental ablations

2.D.1 Sensitivity to auxiliary decoders

In the numbers reported in Tables 2.1 and 2.2, we use a dropout ratio of $p = 0.5$. We show an ablation on the dropout values in Table 2.D.1. Intuitively, this value indicates the level of noise resiliency that we expect in the network; a too-low value has very little use as it is equivalent to having a single decoder without dropout, and too high a value destroys too much information fed to the decoders. Thus an intermediate value like 0.5 is likely to be more appropriate for our use. We find that our experiments support this notion in Table 2.D.1. For this comparison, we use 5 auxiliary decoders. However, our proposed method is quite robust to this choice.

Table 2.D.1: Optimal feature dropout proportions for GTA → Cityscapes experiment.

Dropout (p)	0.2	0.5	0.8
Performance (mIoU) %	44.44	45.07	44.14

2.D.2 Number of auxiliary decoders

The auxiliary decoders play an important role in the level of feature stability induced. Indeed Gal and Ghahramani (2016) show that the larger the number of samples drawn from the dropout distribution, the better the approximation. In Table 2.D.2, we see that the number of decoders plays an important role, but the method is fairly stable in its performance over a range of decoder counts.

Table 2.D.2: Performance variation to the number of auxiliary decoders for GTA → Cityscapes experiment.

# Decoders	1	3	4	5
Performance (mIoU)	43.5	44.20	44.67	45.07

2.D.3 Using single dropout decoder

In our method, we propose the use of multiple decoders instead of one, and we show the effect of using a single decoder to decode the noisy features in Table 2.D.3. While it conceptually seems that using a single decoder should suffice, we see that using multiple auxiliary decoders is helpful. Using multiple decoders forces the feature extractor to be more robust, whereas using a single decoder forces the decoder to be more stable. However, the ASPP decoder is one layer deep, its representation capacity is limited, and thus it is unable to do so, as evidenced in Table 2.D.3.

Table 2.D.3: Utility of using multiple decoders

Method	mIoU %
Single decoder with dropout	43.29
Multiple auxiliary decoders	45.07

Appendix 2.E Qualitative results for Cityscapes to Cross City Adaptation

In Figures 2.E.1 to 2.E.4 we show some qualitative improvements from our results for the *CS*-*CC* experiments.

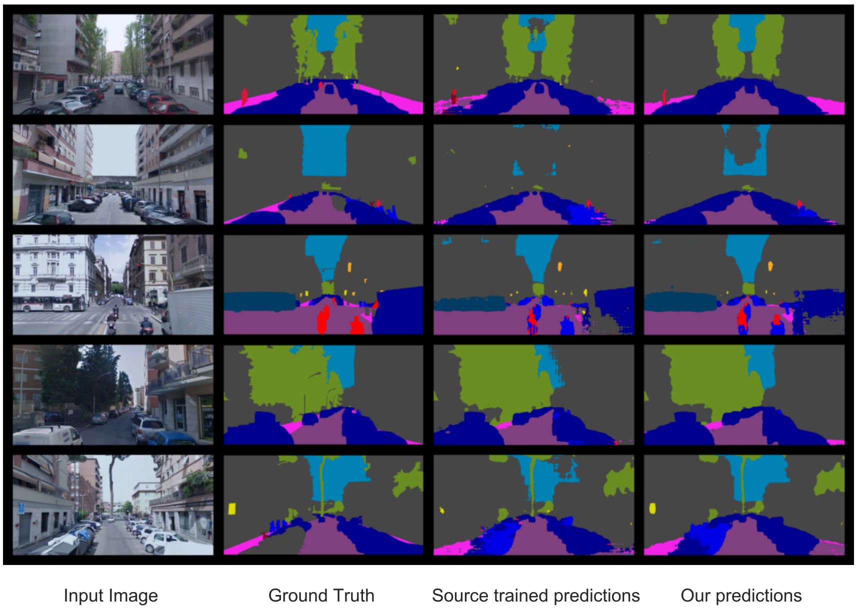


Figure 2.E.1: Best five case results of adaptation for the Cityscapes to Rome (Cross City). The first column is the test image, the second column is the ground truth, the third and fourth columns are the source-trained model, and the results of our proposed method.

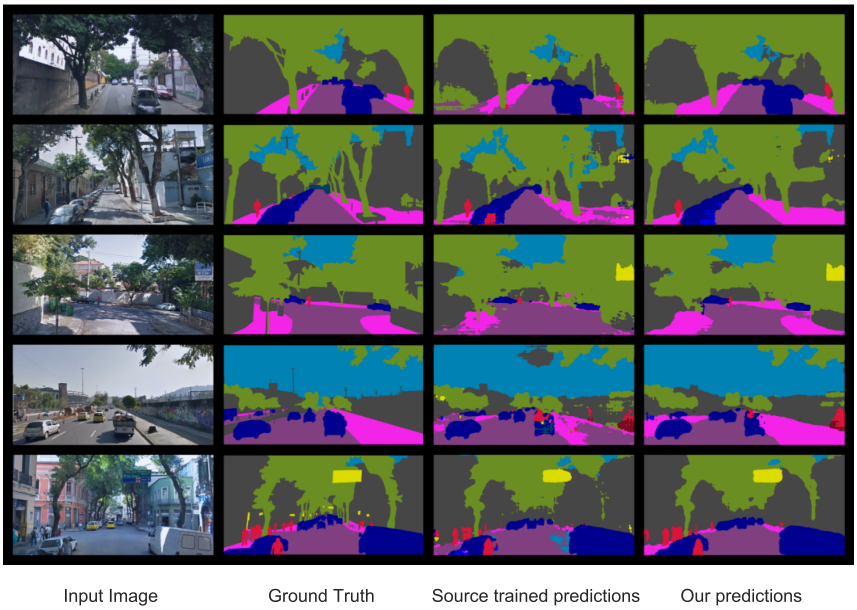


Figure 2.E.2: Best five results of adaptation for the Cityscapes to Rio (Cross City). The first column is the test image and the second column is the ground truth, the third and fourth columns are the source-trained model, and the results of our proposed method.

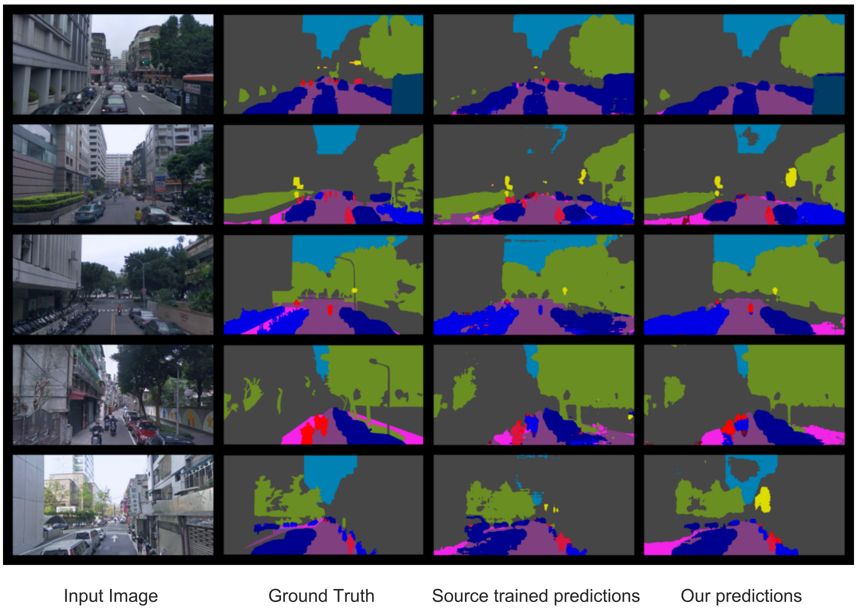


Figure 2.E.3: Best five results of adaptation for the Cityscapes to Taipei (Cross City). The first column is the test image and the second column is the ground truth, the third and fourth columns are the source-trained model, and the results of our proposed method.

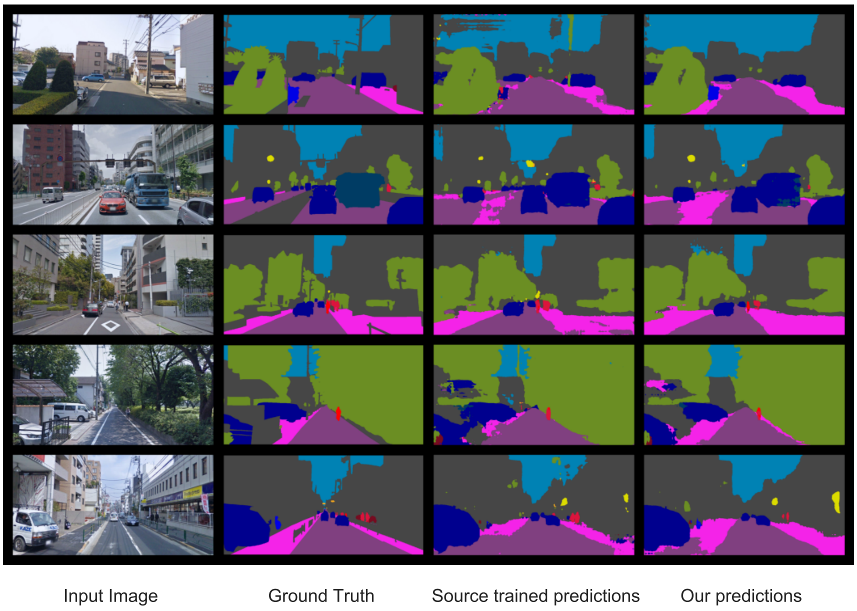


Figure 2.E.4: Best five results of adaptation for the Cityscapes to Tokyo (Cross City). The first column is the test image and the second column is the ground truth, the third and fourth columns are the source-trained model, and the results of our proposed method.

2.E.1 Failure cases

In Figures 2.E.5 to 2.E.8, we show the five images that have least improved over the adaptation process.

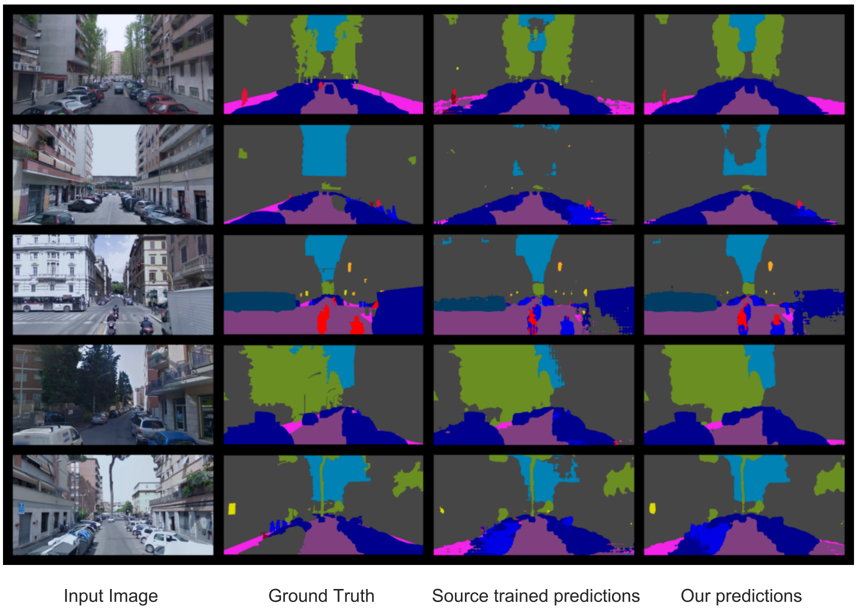


Figure 2.E.5: Worst five results of adaptation for the Cityscapes to Rome (Cross City). The first column is the test image and the second column is the ground truth, the third and fourth columns are the source-trained model, and the results of our proposed method.

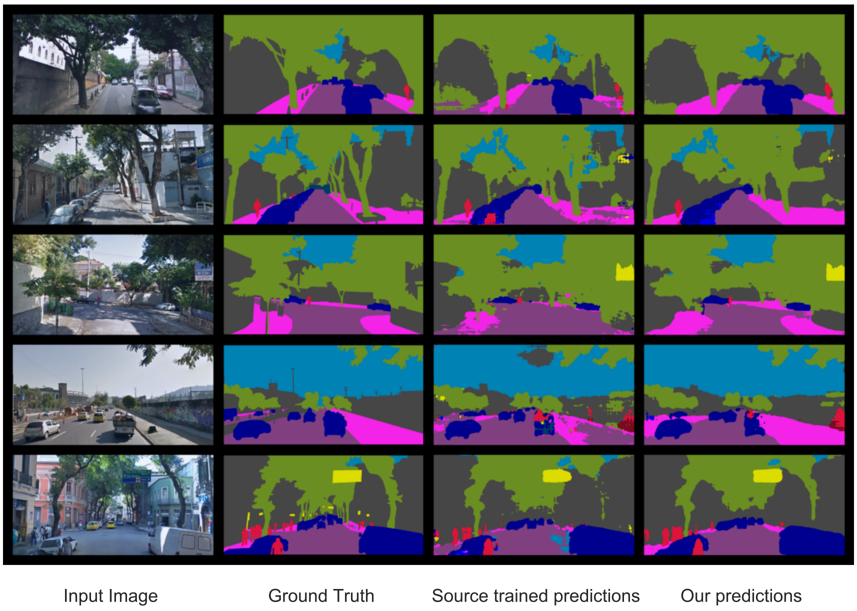


Figure 2.E.6: Worst five results adaptation for the Cityscapes to Rio (Cross City). The first column is the test image and the second column is the ground truth, the third and fourth columns are the source-trained model, and the results of our proposed method.

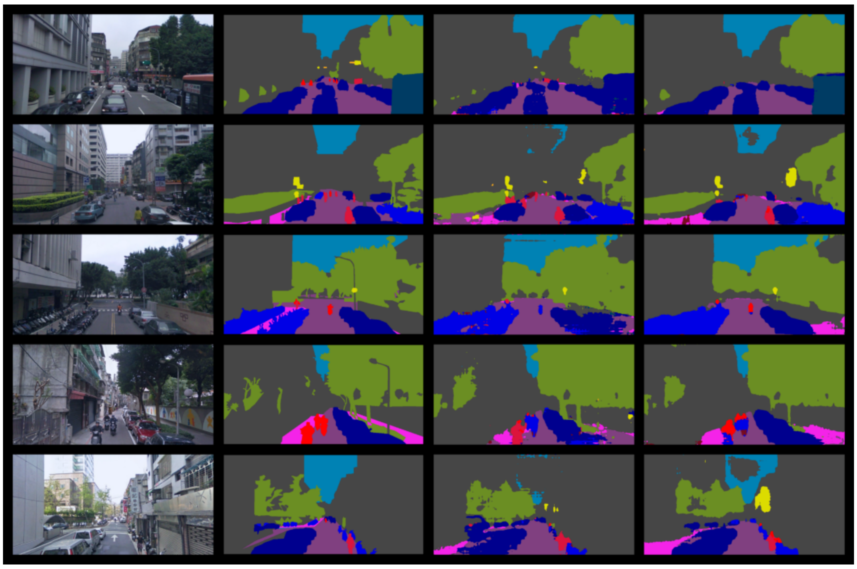


Figure 2.E.7: Worst five results adaptation for the Cityscapes to Taipei (Cross City). The first column is the test image and the second column is the ground truth, the third and fourth columns are the source-trained model, and the results of our proposed method.

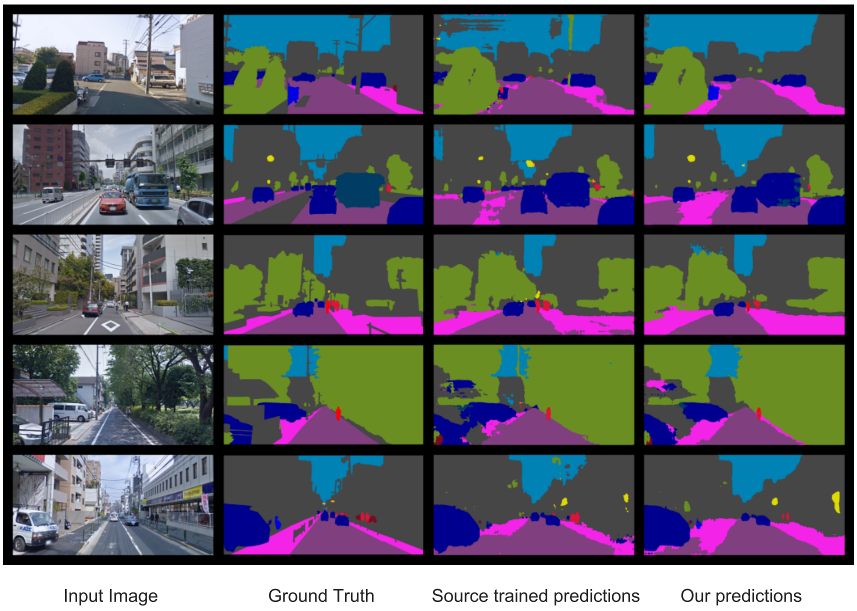


Figure 2.E.8: Worst five results adaptation for the Cityscapes to Tokyo (Cross City). The first column is the test image and the second column is the ground truth, the third and fourth columns are the source-trained model, and the results of our proposed method.

3 Test time Adaptation through Perturbation Robustness

This chapter is based on

> Teja, P. and Fleuret, F. (2021a). Test time adaptation through perturbation robustness. In *NeurIPS 2021 Workshop on Distribution Shifts: Connecting Methods and Applications*.

3.1 Introduction

The most oft-studied scenarios, like the one in Chapter 2, require prior knowledge of the domain shift that will occur at test time and require data from the target domain at the train time. For some problems in sensor data (Vergara et al., 2012), autonomous driving (Bobu et al., 2018), the domain changes gradually and is linked to a physical process (say, change of weather or time of day). In addition to the gradualness, these changes cannot always be anticipated, and thus it is not possible to curate data and train networks for them. This necessitates training strategies that do not need test domain data at train time.

In this chapter, we tackle the problem of handling domain shift at test time, a problem previously termed test time training (Sun et al., 2020), or test time adaptation (Wang et al., 2021a). Differing from model adaptation, the model is adjusted to (possibly) different domains as the data arrives during inference. Owing to the nature of the problem, test-time adaptation necessitates simpler, lighter methods compared to its train-time adaptation counterparts.

Several works have shown that one of the ways to improve generalization performance is to train the network with various input data augmentations (Hendrycks et al., 2020; Cubuk et al., 2019b). Networks are trained with multiple such augmentations for each training sample with the original label. This is equivalent to training the network with original images and adding a consistency regularizer that penalizes different outputs for augmentations (Leen, 1994). In this chapter, we propose a method for test-time adaptation in Section 3.3. Here, we show that enforcing consistency of predictions over various augmentations of the test samples improves the model's performance on corrupted data (CIFAR-10-C/100-C (Hendrycks and Dietterich, 2019), CIFAR-10-$\overline{\text{C}}$ (Mintun et al., 2021)), domain shifts (VisDA-C (Peng et al., 2018)), and on semantic segmentation drifts (*GTA-CS* and Cityscapes→Cityscapes-C). We present preliminary results of an allied method in Appendix 3.F.

3.2 Related work

Data augmentations

Successes of deep learning can be partly attributed to training recipes like data augmentation (Krizhevsky et al., 2012), among others. There have been several attempts at designing these augmentation strategies in the literature. Ratner et al. (2017); Cubuk et al. (2019a) propose to learn data augmentation per task using reinforcement learning. Recently, randomized augmentation strategies like RandAugment (Cubuk et al., 2019b) and AugMix (Hendrycks et al., 2020) have been shown to improve generalization performance, as well as calibration. Data augmentations are the cornerstone of modern self-supervised learning methods (Chen et al. (2020), and survey blog post (Weng, 2021)). A comprehensive survey has been presented in Shorten and Khoshgoftaar (2019).

Consistency losses

Consistency losses were found to help semi-supervised learning problems (Rasmus et al., 2015; Tarvainen and Valpola, 2017; Bachman et al., 2014; Sajjadi et al., 2016). The outputs of the augmented data have been compared through JS-divergence in the current work and in Hendrycks et al. (2020), mean-squared error (Sajjadi et al., 2016; Laine and Aila, 2017), cross-entropy loss (Miyato et al., 2018). Consistency losses over augmented inputs have been the mainstay in self-supervised learning literature like InfoNCE loss (Chen et al., 2020) and cross-correlation (Zbontar et al., 2021).

Transfer learning

In addition to standard techniques like adversarial adaptation (Ganin et al., 2016), data augmentation-based techniques have also been proposed to deal with domain transfer. Volpi et al. (2018) propose that adversarial augmentation leads to better generalization for image domains that are similar to the source domain. Pretraining with auxiliary tasks and using consistency losses was found useful for domain adaptation tasks (Mishra et al., 2021). Huang et al. (2018) use an image translation network for data augmentation for domain adaptation. Augmented inputs have been used to predict multiple outputs, and fused by uncertainty weighting in Yeo et al. (2021). The problem closest to ours is model adaptation (Chidlovskii et al., 2016), where several recent works have shown the utility of pseudo-labeling, and entropy reduction on the unlabeled target set (Teja and Fleuret, 2021b; Liang et al., 2020). In addition, methods that use GANs for image synthesis of target data also exist (Li et al., 2020).

Test time adaptation

Sun et al. (2020) proposed to adapt networks at test time by training networks for the main task and a pretext task like rotation prediction at train time. At test time, they fine-tune the pretext task network to adapt to distribution changes on-the-fly. This method modifies the training procedure to be able to adapt on-the-fly. Recent works (Nado et al., 2021; Schneider et al., 2020) found that adapting batch normalization statistics to the test data is a good baseline for dealing with corruptions at test time. Tent (Wang et al., 2021a) takes this a step further and fine-tunes batch normalization's affine parameters with an entropy penalty. Important questions about the sufficiency of normalization statistics are left unanswered in these works. Our work differs from the existing works in test time adaptation, as we propose increasing the robustness of the network to test samples. It has been shown (Novak et al., 2018) that trained neural networks are robust to input perturbations sampled in the vicinity of the train data; we extend this notion to test time adaptation, by enforcing this robustness as a proxy for improving performance on the test data.

3.3 Perturbation Robustness (PEST)

3.3.1 Method for classification problems

Let $p_{\cdot} \equiv p(y|\cdot) = f(\cdot, \theta)$ be the trained model parameterized by θ on the source training set i.e., $f(\cdot, \theta)$ subsumes the softmax layer. Let $\boldsymbol{x}$ be a test sample and $\tilde{\boldsymbol{x}} = \mathscr{A}(\boldsymbol{x}; \zeta)$ be an output of data augmentation method with hyperparameters $\zeta \sim p_Z(\zeta)$, with p_Z being the density of the augmentation hyperparameters. For a test sample $\boldsymbol{x}$, we sample K augmentations $\tilde{\boldsymbol{x}}_k = \mathscr{A}(\boldsymbol{x}; \zeta_k)$ for $k \in [K]$ where $\zeta_k \sim p_Z(\zeta)$, compute the output probabilities as $p_{\boldsymbol{x}} = f(\boldsymbol{x}, \theta)$, $p_{\tilde{\boldsymbol{x}}_k} = f(\tilde{\boldsymbol{x}}_k, \theta) \quad \forall k$, for the original input, and the augmentations. Over these, we propose to use a consistency loss as in Equation (3.1)

$$\mathscr{L}_{cons}(p_{\boldsymbol{x}}, p_{\tilde{\boldsymbol{x}}_k}) = \frac{D_{KL}(p_{\boldsymbol{x}} \parallel \bar{p}) + \sum_{k=1}^{K} D_{KL}(p_{\tilde{\boldsymbol{x}}_k} \parallel \bar{p})}{K+1}, \tag{3.1}$$

where

$$\bar{p} = \frac{p_{\boldsymbol{x}} + \sum_k p_{\tilde{\boldsymbol{x}}_k}}{K+1} \tag{3.2}$$

is the average posterior density of predictions. Here

$$D_{KL}(p \parallel q) = \sum_i p^i \log\left(\frac{p^i}{q^i}\right)$$

denotes the KL-divergence of the two distributions p and q, where p^i denotes the index i. We note that this loss was originally proposed by Hendrycks et al. (2020) for training with augmentations, and refer the readers to it for experiments about the specific form of Equation (3.1). Mean-squared error over the posterior predictions (Tarvainen and Valpola, 2017; Berthelot et al., 2019) has also been successful in semi-supervised learning. The details of augmentation methods ($\mathscr{A}(\cdot;\cdot)$) used are presented in Appendix 3.A.

In addition to the consistency loss in Equation (3.1), we use an entropy penalty (Wang et al., 2021a; Vu et al., 2019; Grandvalet and Bengio, 2005).

$$\mathscr{L}_{ent}(p_{\boldsymbol{x}}, p_{\tilde{\boldsymbol{x}}_k}, p_{\tilde{\boldsymbol{x}}_2}) = \frac{1}{K+1}\left(\mathbb{H}[p_{\boldsymbol{x}}] + \sum_{k=1}^{K} \mathbb{H}[p_{\tilde{\boldsymbol{x}}_k}]\right) \tag{3.3}$$

Here

$$\mathbb{H}[p] = -\sum_i p^i \log(p^i)$$

is the entropy of predictions p, and the index i is over the number of classes. Contrary to (Wang et al., 2021a), we tune all the parameters of the networks considered. The overall loss is given by the sum of Equation (3.1) and Equation (3.3)

$$\mathscr{L}_{\text{pest}} = \mathscr{L}_{cons} + \mathscr{L}_{ent} \tag{3.4}$$

Our loss function in Equation (3.4) enforces that the network predicts the same output for each augmentation, in addition to being as confident as possible for each of them. We note that Zhang et al. (2022b) proposes a nearly identical method in a concurrent work, and we prove the equivalence in Appendix 3.E. In essence, the total loss in Equation (3.4) is the entropy of the averaged posterior probabilities of the augmented inputs. Rewriting Equation (3.4)

$$\mathscr{L}_{\text{pest}} = \mathbb{H}\left[\frac{\sum_{k=0}^{K} p_{\tilde{\boldsymbol{x}}_k}}{K}\right] \tag{3.5}$$

3.3.2 Method for segmentation

The augmentations $\mathscr{A}$ we use for classification have a restriction that they are label preserving i.e., classification networks are invariant to these transformations. Consequently, Equation (3.4) is trivially applicable when the family of augmentations satisfies label conservation. However, segmentation networks are equivariant to input transformations that modify the input space. To elucidate this idea further, let an augmentation rotate the input by ϕ. For classification, the network is invariant to this and predicts from the same set of labels. For segmentation, the output is also rotated by the same ϕ. To avoid this issue, we restrict our set of augmentation to only the ones that do not modify the input spatial structure like the color space transforms: color jitter and random grayscaling. With this restriction to the augmentation set, Equation (3.4) is applicable to the task of segmentation.

The overall algorithm for PEST for segmentation is shown in Algorithm 1. Here, *episodic* in

Procedure 1 Test Time Perturbation Robustness

```
Input: $\mathcal{T}_x$, source model $f(\boldsymbol{x}, \theta)$
for each batch $\boldsymbol{x}_{\mathcal{B}} \in \mathcal{T}_x$ do
  – Sample $\tilde{\boldsymbol{x}}_k^{\mathcal{B}} = \mathcal{A}(\boldsymbol{x}_{\mathcal{B}}; \zeta_k)$ with $\zeta_k \sim p_Z(\zeta)$ {In Section 3.3}
  if problem is segmentation then
    – Compute aligned outputs $p_{\tilde{\boldsymbol{x}}_k} = \mathcal{A}^{-1}(f(\mathcal{A}(x, \zeta_k), \theta))\ \forall k$
  end if
  – Compute loss $\mathcal{L}$ as in Equation (3.4)
  – Compute updated network weights $\theta^*$ and $\theta \leftarrow \theta^*$
  – Predict $y = \mathrm{argmax}_c\, f(\boldsymbol{x}, \theta)$
  if episodic is True then
      $\theta^* \leftarrow \theta$ {Reset weights to initial state}
  end if
    $\theta \leftarrow \theta^*$
end for
Return: $\{y\}\ \forall \boldsymbol{x}_{\mathcal{B}} \in \mathcal{T}_x$
```

Line 10 refers to resetting the weights to a previous state, possibly source weights. This was empirically shown to be effective for some classification problems (Wang et al., 2021a; Zhang et al., 2022b), and we use that for our experiments on semantic segmentation.

3.4 Experiments

3.4.1 Classification tasks

We show the efficacy of our method on the standard benchmarks used in Tent (Wang et al., 2021a) for ease of comparison: corrupted CIFAR-10 and CIFAR-100, named CIFAR-10-C and CIFAR-100-C respectively (Hendrycks and Dietterich, 2019), and VisDA-C (Peng et al., 2018). CIFAR-10-C and CIFAR-100-C consist 15 corruption types that have been algorithmically generated to benchmark robustness algorithms. VisDA-C dataset is a large-scale dataset that provides training and testing data with 12 classes (real-world objects). Training data consists of rendered 3D models using varying view-point and lighting conditions and the test data is real images cropped from YouTube videos.

For each experiment, we sample two augmentations using either RandAugment (Cubuk et al., 2019b) or AugMix (Hendrycks et al., 2020), and minimize the loss in Equation (3.4) using SGD optimizer with a learning rate of 10^{-4}, momentum 0.9, and weight decay of 5×10^{-4} for 5 iterations. For all the results presented, *Unadapted* refers to the performance of the model trained on the source data of that task. Additional details of the augmentation methods and datasets are presented in Appendix 3.A and Appendix 3.B respectively. We show our main results in this section with AugMix, and present the comparisons to RandAugment in Figure 3.C.1, and other ablations in Appendix 3.D.

Table 3.4.1: Results of VisDA-C, CIFAR-10-C and CIFAR-100-C datasets. For the CIFAR-C datasets, we present the % accuracy averaged over all corruptions here.

Dataset	Method	Accuracy (%)
VisDA-C	Unadapted	44.1
	Tent (Wang et al., 2021a)	60.9
	Proposed	67.2
CIFAR-10-C	Unadapted	56.5
	Norm (Nado et al., 2021)	79.4
	Tent (Wang et al., 2021a)	81.4
	Proposed	81.2
CIFAR-100-C	Unadapted	53.2
	Norm (Nado et al., 2021)	61.7
	Tent (Wang et al., 2021a)	64.5
	Proposed	65.7

Domain Adaptation

We use a ResNet-50 (He et al., 2016b) network that has been pre-trained on Imagenet, and use a test time batch size of 64. With the results presented in Table 3.4.1, we see the proposed method beats the existing methods by a significant margin of $\sim 6.2\%$. We find that lower intensity of augmentations i.e.,, $m = 1$, $n = 1$ for Randaugment, and $\alpha = 1$, depth $= 3$, severity $= 2$, width $= 1$ for AugMix result in the best performance. We also found that our method is relatively stable to small changes to augmentation parameters used.

Corruption (CIFAR-10-C and CIFAR-100-C)

We use Wide ResNet (Zagoruyko and Komodakis, 2016) pre-trained models from the RobustBench code repository (Croce et al., 2020). For CIFAR-10-C, we use a WRN-28-10, and for CIFAR-100-C we use WRN-40-2. These networks have been trained on the training set of CIFAR 10/100. We use the augmentation hyperparameters as in domain adaptation experiments, with the batch size changed to 200. In Table 3.4.1 and Appendix 3.C, we see that the performance of our proposed method is comparable to Tent on CIFAR-10-C, and beats Tent by a margin of $\sim 1\%$ for CIFAR-100-C.

On the dependence of augmentations and corruptions

Here we study the dependence of the performance on the exact set of augmentations used in AugMix or RandAugment in the above results. Mintun et al. (2021) proposes an alternative set of corruptions termed CIFAR-10-$\overline{\text{C}}$ that they estimate to be widely different from the

Table 3.4.2: Results on CIFAR-10-$\overline{\text{C}}$. Our method can deal with corruption types that are widely different from the augmentations in AugMix.

Dataset	Method	Accuracy
CIFAR-10-$\overline{\text{C}}$	Baseline	58.9
	Proposed	75.64

augmentations used in AugMix. We show results for our proposed method in Table 3.4.2. We see that our method gives substantial improvements over the baseline source model, and therefore the utility of the augmentations is to sample from the image manifold than to impart specific invariance to those augmentations.

3.4.2 Dealing with image corruptions in semantic segmentation

We adopt the experimental setup in Chapter 2 and use GTA to Cityscapes (*GTA-CS*) (Richter et al., 2016; Cordts et al., 2016) adaptation problem and add the Cityscapes to Cityscapes-C adaptation (Kamann and Rother, 2020). Cityscapes-C uses the identical corruptions as CIFAR-C datasets for which we showed results for classification experiments. For the *GTA-CS* experiment, we use the same network (DeeplabV2) as Chapter 2, and for Cityscapes→ Cityscapes, we use SwiftNet architecture with a ResNet-18 backbone (Orsic et al., 2019). We use the source-trained models from Zou et al. (2019) and Courdier and Fleuret (2020), respectively. To evaluate performance, we use the mean intersection-over-union (mIoU) metric.

For adaptation, we use SGD optimizer learning rate of 10^{-4} with a momentum of 0.9 and weight decay of 0.0005. We use a batch size of 1, and use 2 augmentations due to a memory constraint posed by our computational infrastructure.

Adapting batchnorm

To handle the small batch size problem, we use the method proposed in (Schneider et al., 2020). We denote $[\mu_S, \sigma_S^2]$ to be the normalization statistics computed on source data. If $[\mu_t, \sigma_t^2]$ are the normalization statistics computed from a given batch of test samples, the final normalization statistics $[\mu_{\text{eff}}, \sigma_{\text{eff}}^2]$ used in our method are

$$\begin{aligned}\mu_{\text{eff}} &= \alpha\mu_S + (1-\alpha)\mu_t \\ \sigma_{\text{eff}}^2 &= \alpha\sigma_S^2 + (1-\alpha)\sigma_t^2.\end{aligned} \tag{3.6}$$

We use a relative weight of $\alpha = \frac{15}{16}$ for the test data. We used episodic resetting as it gave better results. We use Equation (3.6) for the "norm" results in Figure 3.4.1. We also combine it with our proposed method in both Figure 3.4.1 and Table 3.4.3. We see that our method always improves compared to the baseline, and outperforms Schneider et al. (2020) on most

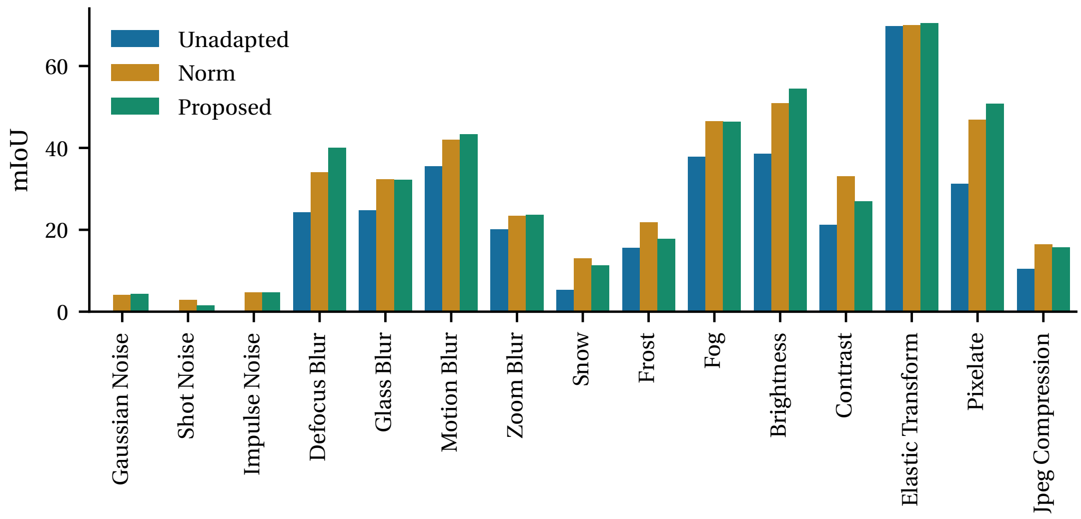


Figure 3.4.1: Cityscapes to Cityscapes-C adaptation on SwiftNet. We see that the in the presence of noise, the source-trained network fails (< 1%) and hence the invisible bar. We see that the proposed method improves the performance on the target domain on nearly all corruptions.

Table 3.4.3: Results on *GTA-CS*. Our method consistently improves over the baseline case. Our method makes the largest improvements on classes that already perform reasonably well, and worsens on classes with lower unadapted performance.

Method	Road	Sidewalk	Building	Wall	Fence	Pole	Tra. light	Tra. Sign	Veg.	Terrain	Sky	Person	Rider	Car	Truck	Bus	Train	Motorbike	Bike	mIoU
Unadapted	71.62	18.70	67.88	16.83	9.84	34.42	27.41	6.16	78.15	21.51	67.06	57.16	20.49	58.91	16.39	12.23	6.71	20.24	12.62	32.86
Proposed	85.17	20.19	76.50	15.88	7.81	32.25	21.71	4.41	81.43	31.30	72.29	57.37	21.91	81.52	26.13	23.69	5.46	21.80	12.61	36.81

corruptions in Figure 3.4.1. We do not report Tent here as we found that it was worsening the performance compared to the unadapted baseline. Our results in Table 3.4.3 and Table 2.1 are not trivially comparable, as the experimental settings are widely different. Several targeted improvements to test-time adaptation for semantic segmentation have been proposed in the recent past, and they achieve better performance than the one presented here (Paul et al., 2022; Wang et al., 2022a; Bahmani et al., 2022).

3.5 Discussion

Our experiments show that a consistency regularization on the input space (Equation (3.1)) improves performance on the test data. Interestingly, learning invariance to general synthetically created deformations leads to better performance even on data from different domains, as shown in Table 3.4.1. However, the specific characteristics of the augmentations that result in better generalization are unclear.

Previous studies found that networks with flat minima in the weight space generalized better (Chaudhari et al., 2017; Keskar et al., 2016). Huang et al. (2020) consider neural networks with flat minima analogous to wide-margin classifiers. Wide margin, in addition to being interpreted as stability to perturbation of parameters, can also be seen as being robust to input perturbations (Bousquet and Elisseeff, 2002; Elsayed et al., 2018). While these prior works partially explain our proposed method, they do not explain how solely searching for a flat region (in the input space) corresponds to a point in the weight space that generalizes.

Works on generalization prediction (Deng et al., 2022; Zhu et al., 2021; Ng et al., 2022) show that agreement of predictions between samples and their augmentations and performance on those samples are linearly related i.e., how the prediction of augmentations matches with each other is also indicative of the network's ability to classify that sample reliably. We use a variation of this to enforce the consistency of augmentations. Similarly, ensuring consistency of outputs to augmentations has been famous in the NLP domain (Ribeiro et al., 2020). In this work, we reverse the process; we do not estimate the performance of a network by measuring the consistency, but we use consistency as a loss function to improve the performance.

We find that the efficacy of our method, as well as that of Tent (Wang et al., 2021a), is dependent on the batch size used (Section 3.D.3), which is a departure from any normal testing scenario, where each sample is labeled independently. The use of a convex combination of source and target statisticsEquation (3.6) (Zhang et al., 2022b; Schneider et al., 2020) partially alleviates the problem. However, the overall performance is still lower than when using a larger batch size or when we can use the entire source statistics as in Chapter 2.

Appendix - Chapter 3

Appendix 3.A Augmentation methods

In this section we describe the augmentation methods RandAugment (Cubuk et al., 2019b), and AUGMIX (Hendrycks et al., 2020) used widely in the main text.

3.A.1 RandAugment

Methods like AutoAugment (Cubuk et al., 2019a) use complex machinery like reinforcement learning to deduce the optimal augmentation strategy for a problem. RandAugment proposes to do the opposite: it randomly samples augmentations from a predefined list of augmentations and composes them functionally to output the augmentation for each data point. The algorithm is shown in Algorithm 2.

Procedure 2 RandAugment

```
Input: $\mathcal{O} \leftarrow$ {Identity, AutoContrast, Equalize, Rotate, Solarize, Color, Posterize, Contrast,
Brightness, Sharpness, ShearX, ShearY, TranslateX, TranslateY}, $\boldsymbol{x}$, $m$, $n$
{$m$ is the maximum intensity of augmentations, $n$ is the number of augmentations}
for i in 1 ... $n$ do
    local_intensity $\leftarrow$ randint(1, $m$)
    sample $\leftarrow$ random_choice($\mathcal{O}$)
    $\boldsymbol{x} \leftarrow$ sample(local_intensity)($\boldsymbol{x}$)
end for
Return: $\boldsymbol{x}$
```

3.A.2 AUGMIX

AUGMIX is a data augmentation technique that has been shown to improve model robustness. Unlike RandAugment, where sampled augmentations are composed, AugMix mixes the results of chains of augmentations in convex combinations. Increasing diversity by composing augmentations can generate a sample that is off the data manifold and the authors argue that their proposed way of combining generates realistic transformations. The specific algorithm is given in Algorithm 3.

Procedure 3 AUGMIX. Adapted from Hendrycks et al. (2020)

```
Input: $\mathcal{O}$ similar to Algorithm 2, $k$, $\alpha$
$\boldsymbol{x}_{aug} \leftarrow$ zeros_like($\boldsymbol{x}$)
$(w_1, w_2, \dots w_k) \sim$ Dirichlet$(\alpha, \alpha, \dots, \alpha)$
for i in $1 \dots k$ do
    sample$_1$, sample$_2$, sample$_3$ $\leftarrow$ random_choice($\mathcal{O}$)
    sample$_{1,2}$ $\leftarrow$ sample$_1 \circ$ sample$_2$, sample$_{1,2,3}$ $\leftarrow$ sample$_1 \circ$ sample$_2 \circ$ sample$_3$
    op $\leftarrow$ random_choice({sample$_1$, sample$_{1,2}$, sample$_{1,2,3}$})
    $\boldsymbol{x}_{aug} += w_i \cdot op(\boldsymbol{x})$
end for
sample $m \sim$ Beta$(\alpha, \alpha)$
$\boldsymbol{x}_{\text{augmix}} = m\boldsymbol{x} + (1-m)\boldsymbol{x}_{aug}$
Return: $\boldsymbol{x}_{\text{augmix}}$
```

Appendix 3.B Datasets

3.B.1 Corruption datasets – CIFAR-10-C, CIFAR-100-C

To benchmark the model robustness, Hendrycks and Dietterich (2019) proposed corruption (along with perturbation) datasets. In this work, we consider the datasets that were derived from the commonly used CIFAR-10 and CIFAR-100 datasets, named CIFAR-10C and CIFAR100-C respectively. They consist 15 corruption types applied to the test data of the original datasets. Each of the 15 corruptions has 5 levels of severity, and we present our results on the maximum level of corruption. The corruptions can be grouped into four main categories – noise, blur, weather, and digital. A summary is presented in Table 3.B.1.

Table 3.B.1: Summary of corruptions in CIFAR-10-C and CIFAR-100-C used in this work.

Noise	**Blur**	**Weather**	**Digital**
Gaussian noise	Defocus blur	Snow	Brightness
Shot noise	Frosted Glass blur	Frost	Contrast
Impulse noise	Motion blur	Fog	Elastic transformation
	Zoom blur	Spatter	Pixelation
			JPEG compression

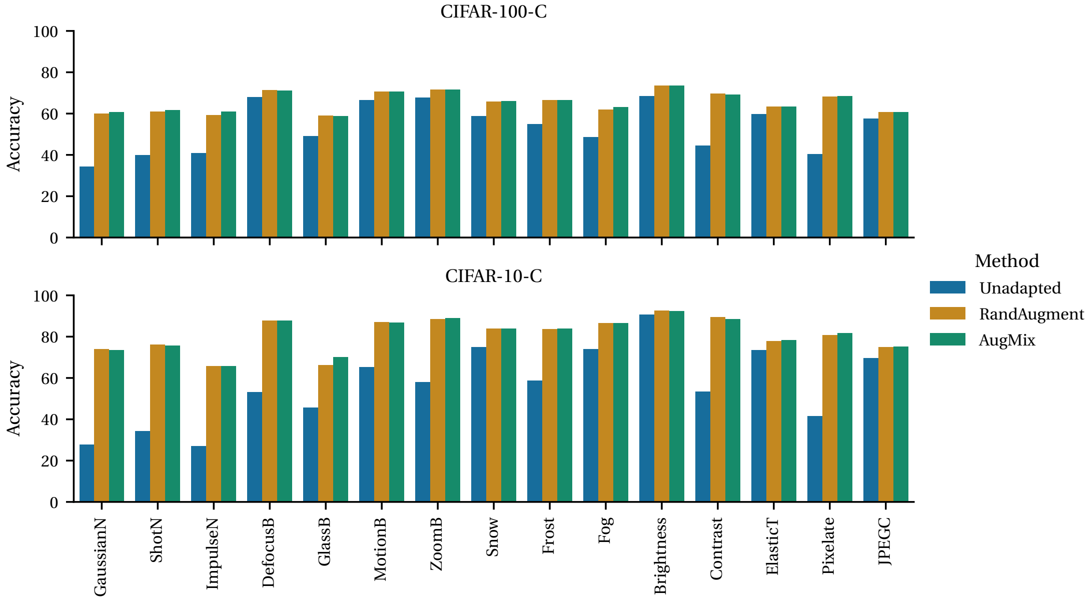


Figure 3.C.1: Performance comparisons for CIFAR-10-C and CIFAR-100-C datasets for unadapted, Tent, Augmix, Randaugment variants of our method. Our proposed method, independent of the augmentation strategy used, greatly improves over the baseline. It is also on par or slightly better than Tent for all corruption categories.

3.B.2 VisDA-C

VisDA-C (Peng et al., 2018) is a large-scale dataset to benchmark unsupervised domain adaptation methods. It consists of three domains of 12 classes (object categories). The source (training) data consists of renderings of 3D models from various viewpoints and lighting conditions. The validation domain consists of real images cropped from MS-COCO. The target (test) data also consists of real images cropped from YouTube bounding box datasets. All three splits contain the classes: aeroplane, bicycle, bus, car, horse, knife, motorbike, person, plant, skateboard, train, and truck.

Appendix 3.C Detailed results of CIFAR datasets

We show the detailed per corruption results of our CIFAR-C experiments in Figure 3.C.1. We find that our method improves drastically when the baseline performance is low.

Appendix 3.D Ablations

In this section, we provide ablations of three important hyperparameters of our algorithm, number of SGD steps, learning rate, and testing batch size. We use the VisDA-C experimental setup and show our results for RandAugment setup with default hyperparameters. The

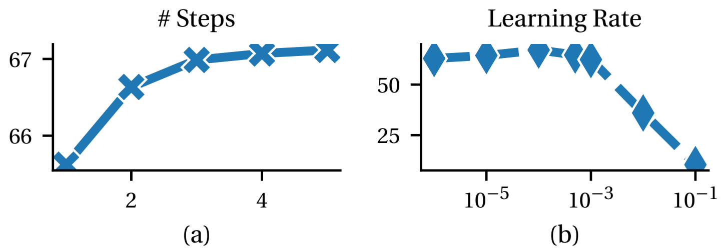


Figure 3.D.1: Ablation experiments. With an unadapted performance of 44.1%, we show the importance of the factors. We see an asymptotically improving performance with the number of SGD steps used. In (b), our method is relatively stable over a wide range of learning rates. Our method is dependent on the batch size as in (c). The necessity of a larger batch size can constrain our method's direct applicability. This method

other hyperparameters remain the defaults set in the main chapter unless they are being ablated. The three ablations refer to Figure 3.D.1. For all the ablations in this section, we show performance as % accuracy on the test set.

3.D.1 Number of SGD update steps

The loss function in Equation (3.4) is minimized over several SGD steps, and we plot the evolution of performance with the number of steps. We see an asymptotic rise in the performance with the number of steps, and we use 5 steps for all other steps. However, this increases the run-time of our algorithm at test time, and thus we recommend a number dependent on the running time requirements.

3.D.2 Learning rate

We change our learning rate on a logarithmic scale from 10^{-6} to 10^{-1}, and we see that our method is reasonably robust to the choice between 10^{-6} to 10^{-3}, with the best learning rate around 10^{-4}.

3.D.3 Batch size

We find that our method is stable for a range of batch sizes but does have a lower threshold of around 16. We hypothesize this is due to training the normalization layers in the networks used. Our limitation is the hardware available to run larger batch sizes.

Table 3.D.1: Ablations of the loss terms in Equation (3.4).

Unadapted	$\mathscr{L}_{cons}$	$\mathscr{L}$
44.08	63.41	67.1

3.D.4 Ablation of terms in Equation 3.4

Our loss function in Equation (3.4) is a summation of two terms. Here we present the results obtained by considering each of the terms. We see that in Table 3.D.1, the usage of Equation (3.1) leads to a performance higher than that of Tent, and the combination of Equations (3.1) and (3.3) leads to better results, albeit at a higher run-time requirement.

3.D.5 Augmentation parameters

We vary the augmentation hyperparameters for both RandAugment, and AugMix in Figures 3.D.2 and 3.D.3. For this, we show results on CIFAR-10-C for each of the corruptions. We show results with only consistency loss (Equation (3.1)), and width and brightness of each circle represent the overall final accuracy. We find that for nearly all the corruptions, a lower augmentation results in better performance than those with a higher augmentation. We hypothesize this is because lower intensity augmentations result in samples closer to the input sample and thus result in samples from the image manifold.

Appendix 3.E Equivalence to MEMO

MEMO (Zhang et al., 2022b) is another test-time adaptation method proposed concurrently to our work (Teja and Fleuret, 2021a). Their method takes a test sample $\boldsymbol{x}$ and augments it to generate B augmentations. The overall loss optimized is the entropy of the average posterior probability.

$$\bar{p}(y|x) := \frac{1}{B}\sum_{i=1}^{B} f(\tilde{\boldsymbol{x}}_i, \theta) \tag{3.7}$$

following the notation introduced previously.

Our proposed loss function in Equation (3.4) can be expanded to

$$\mathscr{L} = \frac{D_{KL}(p_{\boldsymbol{x}} \parallel \bar{p}) + D_{KL}(p_{\tilde{\boldsymbol{x}}_1} \parallel \bar{p}) + D_{KL}(p_{\tilde{\boldsymbol{x}}_2} \parallel \bar{p})}{3} + \frac{1}{3}\left(\mathbb{H}[p_{\boldsymbol{x}}] + \mathbb{H}[p_{\tilde{\boldsymbol{x}}_1}] + \mathbb{H}[p_{\tilde{\boldsymbol{x}}_2}]\right) \tag{3.8}$$

KL Divergence between two distributions can be written as

$$D_{KL}(P \parallel Q) = \mathbb{H}[P, Q] - \mathbb{H}[P] \tag{3.9}$$

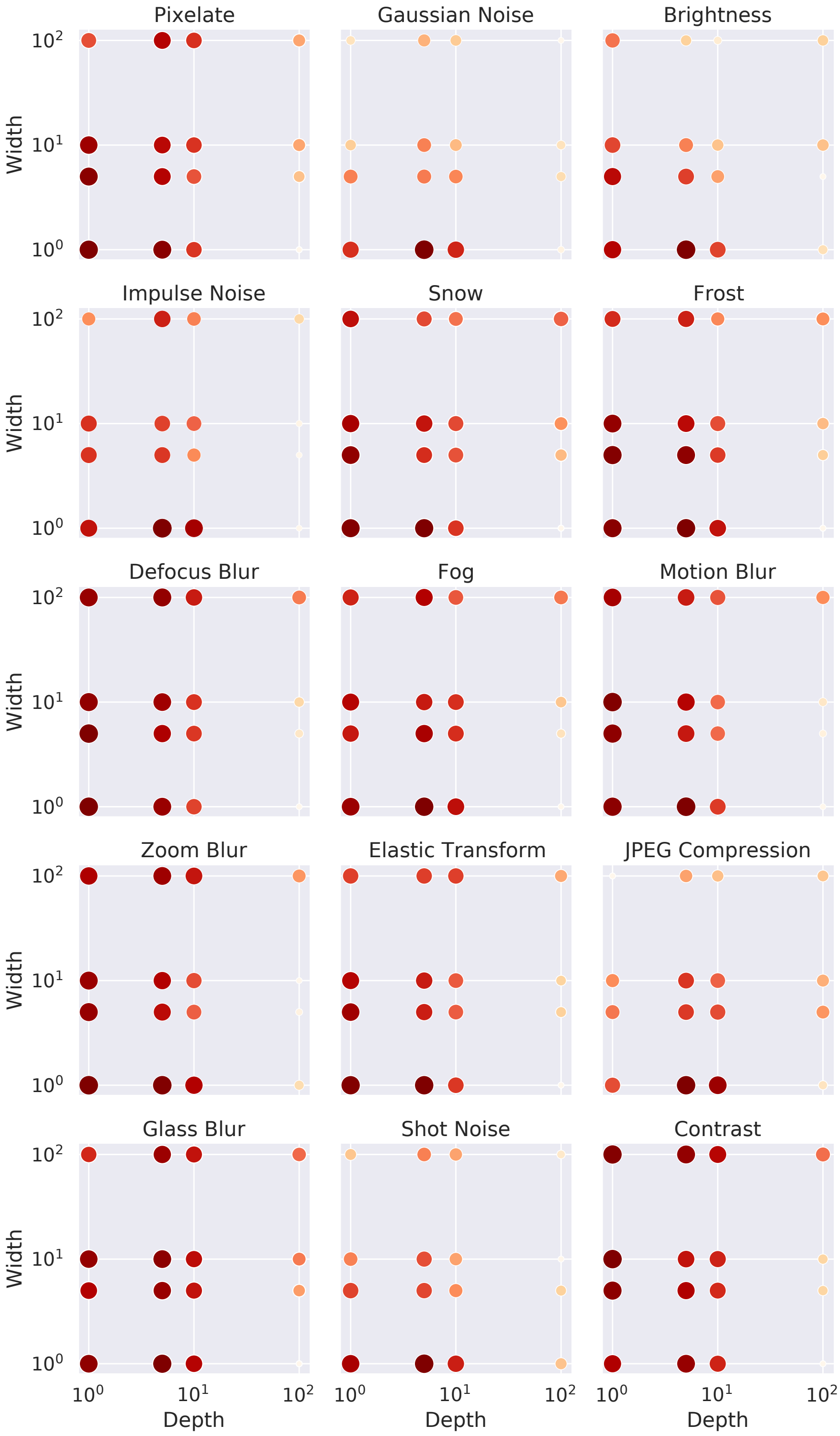


Figure 3.D.2: AugMix hyperparameters for CIFAR-10-C. Bigger and darker are better. We see that lesser augmentations result in better results.

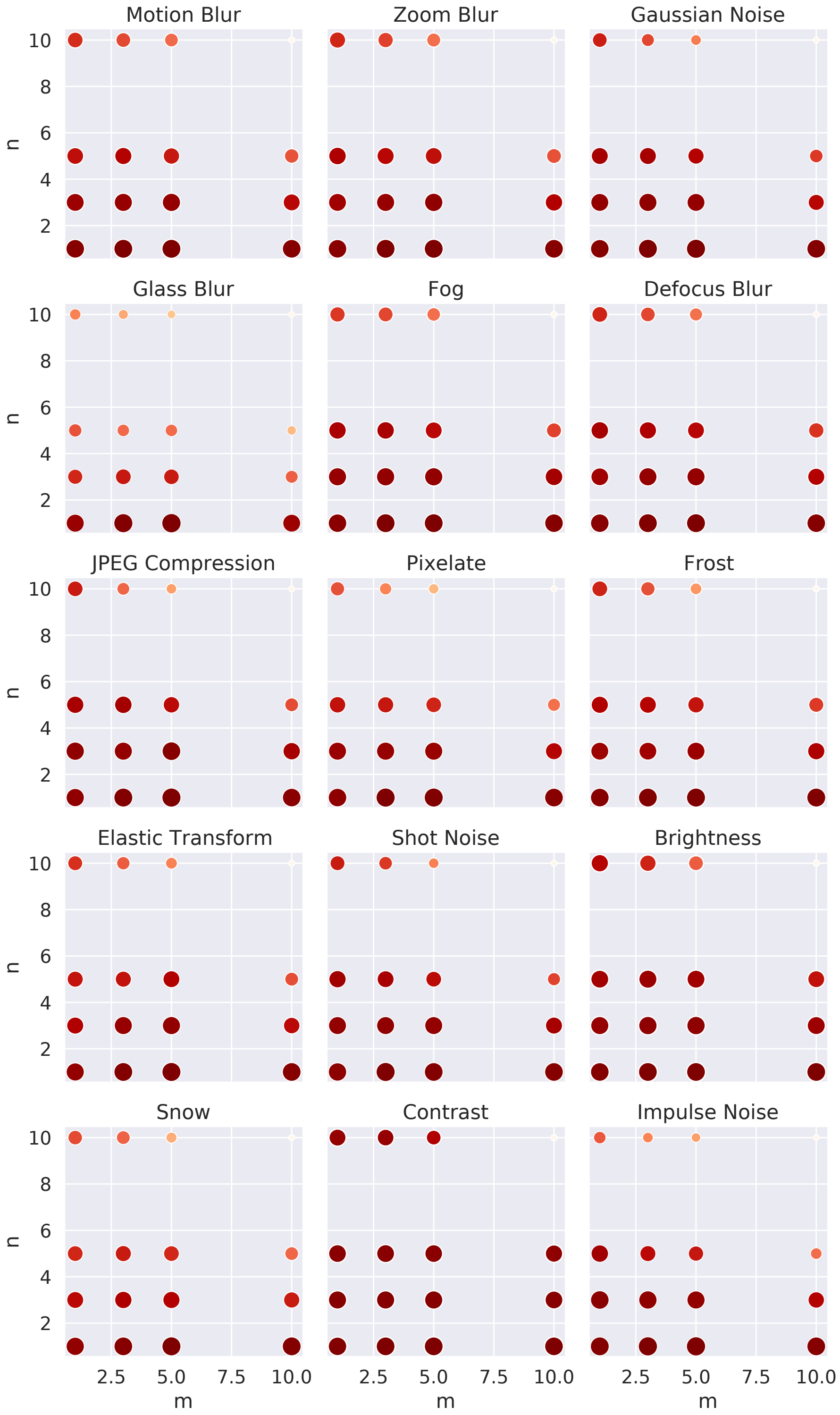


Figure 3.D.3: RandAugment hyperparameters for CIFAR-10-C. Bigger and darker are better. We see that lesser augmentations result in better results.

where $\mathbb{H}[P,Q]$ is the cross entropy between P and Q defined as

$$\mathbb{H}[P,Q] = -\sum_i P_i \log Q_i \tag{3.10}$$

Expanding the first three terms using the definition of KL-divergence above, we see that the entropy terms cancel. This gives us only the cross entropy terms.

$$\mathscr{L} = \frac{\mathbb{H}[p_{\boldsymbol{x}}, \bar{p}] + \mathbb{H}[p_{\tilde{\boldsymbol{x}}_1}, \bar{p}] + \mathbb{H}[p_{\tilde{\boldsymbol{x}}_2}, \bar{p}]}{3} \tag{3.11}$$

$$= \mathbb{H}[\frac{p_{\boldsymbol{x}} + p_{\tilde{\boldsymbol{x}}_1} + p_{\tilde{\boldsymbol{x}}_2}}{3}, \bar{p}] \tag{3.12}$$

From Equation (3.2), we see that this simplifies to

$$\mathscr{K} = \mathbb{H}[\bar{p}, \bar{p}] = \mathbb{H}[\bar{p}] \qquad \text{entropy of } \bar{p} \tag{3.13}$$

Thus, our loss function is almost identical to MEMO's, with one minor difference. We use both the original input and the augmentations to compute the average $\bar{p}$. However, preliminary experiments without the original and instead using multiple augmentations revealed that the exclusion has little influence on the performance.

Appendix 3.F Consistent Pseudo-labeling

3.F.1 Proposed method

Pseudo-labeling (Lee, 2013; Rusak et al., 2022) and entropy minimization are equivalent under the condition that the pseudo-labels are updated each iteration of training to be the soft labels predicted by the network. To see this, consider the cross entropy loss with which pseudo-label based training is done.

$$\mathscr{L}_{\text{ce}} = -\langle \boldsymbol{y}, \log p \rangle \tag{3.14}$$

$$\text{When } \boldsymbol{y} = p$$

$$= -\langle p, \log p \rangle = \mathscr{L}_{\text{ent}} \tag{3.15}$$

where $\boldsymbol{y}$ is the reference label vector, and p is the probability vector for classification.

In Section 3.3, we proposed a loss function for enforcing output consistency for augmented inputs. In this section, we propose a loss that enforces consistency of soft pseudo-labels. Borrowing the notation from Section 3.3: $\boldsymbol{x}$ is a test sample, and y be its unknown ground-truth label and $\tilde{\boldsymbol{x}} \sim \mathscr{A}(\tilde{\boldsymbol{x}}|\boldsymbol{x})$ be an output of data augmentation method $\mathscr{A}(\cdot|\cdot)$ with input $\boldsymbol{x}$, and we sample K of them $\tilde{\boldsymbol{x}}_1, \tilde{\boldsymbol{x}}_2, \ldots \tilde{\boldsymbol{x}}_K$.

For classification problems, the augmentation function $\mathscr{A}$ generates label preserving augmentations. Consider the probability of common label distribution given the augmented inputs.

$$p(y|\tilde{\boldsymbol{x}}_1, \tilde{\boldsymbol{x}}_2, \ldots \tilde{\boldsymbol{x}}_K) = \frac{\prod_{k=1}^{K} p(y|\tilde{\boldsymbol{x}}_k) p(\tilde{\boldsymbol{x}}_k)}{p(\tilde{\boldsymbol{x}}_1, \tilde{\boldsymbol{x}}_2, \ldots \tilde{\boldsymbol{x}}_K) p(y)} \tag{3.16}$$

$$\propto \prod_{k=1}^{K} p(y|\tilde{\boldsymbol{x}}_k) \tag{3.17}$$

Thus, our loss function, modeled after the entropy based pseudo labeling in Equation (3.14), is the entropy of the random variable with density $p(y|\tilde{\boldsymbol{x}}_1, \tilde{\boldsymbol{x}}_2, \ldots \tilde{\boldsymbol{x}}_K)$. On the right side only the numerator has trainable parameters.

$$\mathscr{L}_{\text{cons-pl}} = -\mathbb{H}[Y|X_1, X_2, \ldots X_K] \tag{3.18}$$

In practice, implementing Equation (3.18) entails Hadamard product of the predictions for each of the K augmentations, and renormalizing it to obtain the distribution $\mu(Y|X_1, X_2, \ldots X_K)$. In practice, the multiplication of probability vectors results in floating-point errors. Hence, we compute $\mu(Y|X_1, X_2, \ldots X_K)$ as the softmax over the sum of logits. If $g(\cdot)$ refers to the neural network function that produces the logits, Equation (3.18) is implemented as

$$\mathscr{L}_{\text{cons-pl}} = -\mathbb{H}[\text{Softmax}(\sum_{k=1}^{K} g(\tilde{\boldsymbol{x}}_k))]. \tag{3.19}$$

Here

$$\mathbb{H}[p] = -\sum_k p^k \log(p^k)$$

is the entropy of predictions p, and the index k is over the number of classes.

In comparison, from Appendix 3.E, we see that the loss function from Section 3.3 is

$$\mathscr{L}_{\text{pest}} = -\mathbb{H}\left[\frac{1}{K}\sum_{k=1}^{K} \text{Softmax}(g(\tilde{\boldsymbol{x}}_k))\right]. \tag{3.20}$$

3.F.2 Results

Here, we show the results of consistent pseudo-labeling on the CIFAR-C datasets. All the experimental settings are identical to the ones presented in Section 3.4. For each of the corruptions in CIFAR-C datasets, we compare the performance of PEST with Robust-PL in Figure 3.F.1. Consistent PL almost underperforms PEST across all corruption categories. This is at odds with the observations in Rusak et al. (2022), which suggest pseudo-labeling as the

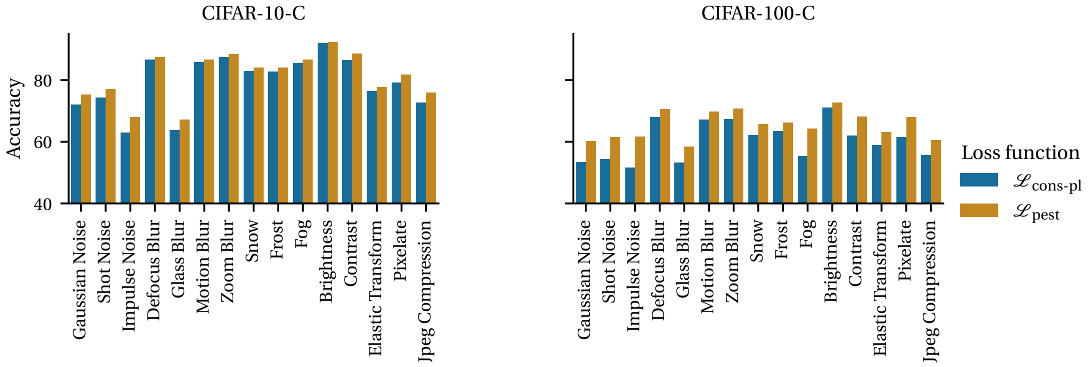


Figure 3.F.1: Comparison of Robust PL and PEST. It is apparent that across all corruptions Consistent PL is at most equal to PEST in performance.

best alternative. However, Rusak et al. (2022) propose the use of robust losses for pseudo-labeling instead of the standard cross entropy loss that we reinterpret as an entropy loss in Equation (3.14).

Dealing with computational needs Part II

4 PAUMER: Patch Pausing Transformer for Semantic Segmentation

In Part I, we presented work on source data-free adaptation techniques that aim at reducing the amount of labeled data required for real-world problems by effectively reusing information gained from the source domain to the target–inference domain. In this chapter, we focus on the problem of efficient inference for segmentation transformers.

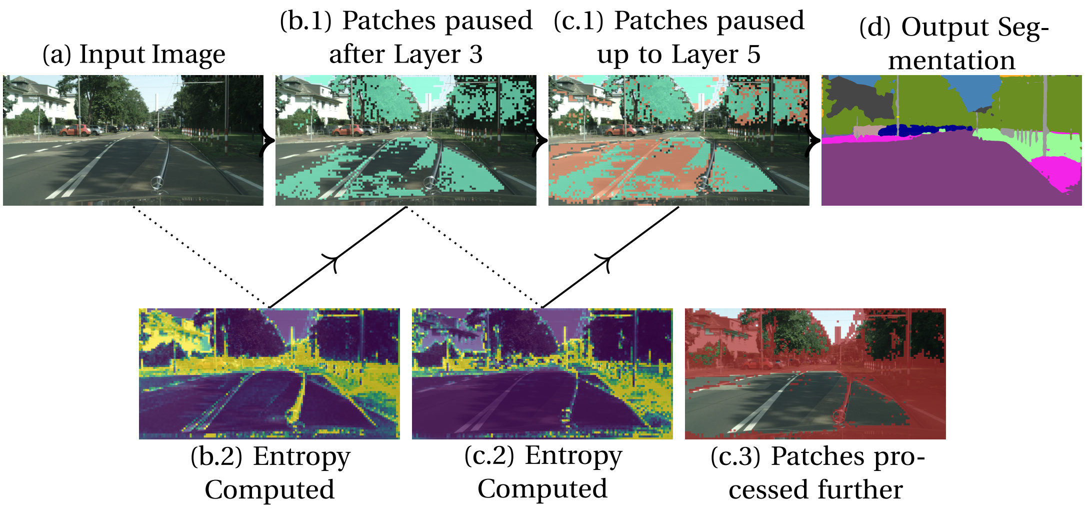
This chapter is based on

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

4.1 Introduction

Vision transformers (Dosovitskiy et al., 2021; Steiner et al., 2021) (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). While their genesis was for image classification, recent methods have adapted transformer architectures to various computer vision tasks (Khan et al., 2021; Wang et al., 2021b; Chu et al., 2021), and specifically to semantic segmentation (Zheng et al., 2021; Strudel et al., 2021; Xie et al., 2021).

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 faster and process more data (Tay et al., 2022). One way to achieve this is to reduce the number of processed patches. Some works use multiscale approaches (Wang et al., 2021b; 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., 2022). For example, it is possible to classify an image as that


(a) Input Image

(b.1) Patches paused after Layer 3

(c.1) Patches paused up to Layer 5

(d) Output Segmentation

(b.2) Entropy Computed

(c.2) Entropy Computed

(c.3) Patches processed further

Figure 4.1.1: Illustration of our proposed method. Our method progressively stops processing patches after they reach a low enough prediction entropy. The first column (a) shows the input image. The second column (b.1) shows the patches that are stopped from being processed after the third transformer layer in green. The third column shows additional patches 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 are used to decide which patches to pause (Section 4.2.1). 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 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 4.2.

of a dog with only the patches that belong to the dog while refraining from processing the rest of the patches.

In this work, we are interested in patch-dropping in the context of semantic segmentation. Unlike image classification, it is impossible 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 particular layer signifies that its representation will not be updated by any subsequent encoder layer, does not contribute to the feature computation of other patches, and is fed directly to the decoder. Consider segmenting natural road scenes from Cityscapes (Cordts et al., 2016) in Figure 4.1.1, 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 and thus improve 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 (Zhou et al., 2017). Our method pushes the pareto front of the speed-accuracy trade-offs, and we find that we can operate at a 50% higher throughput with a drop in mean intersection of union (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.

4.2 Patch-pausing transformer for Semantic Segmentation

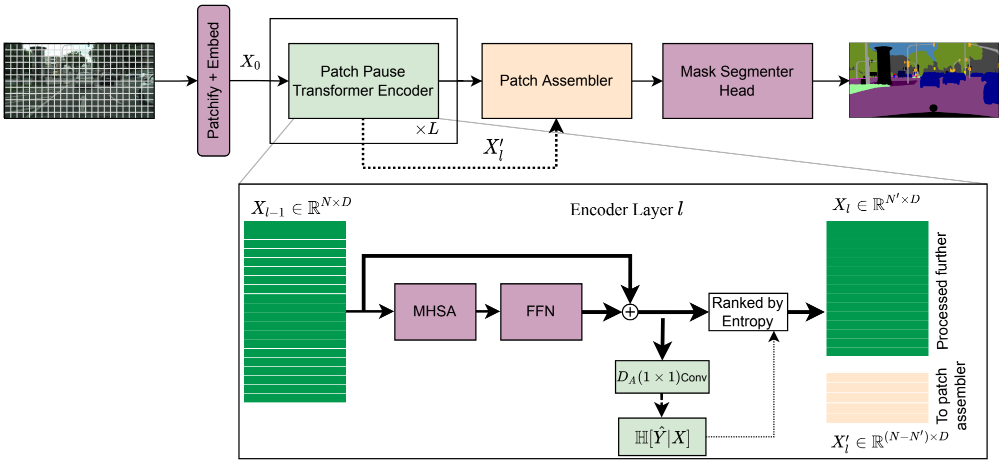


Figure 4.2.1: Schematic of our proposed method. We modify Segmenter (Strudel et al., 2021) to enable the pausing of patches, and feeding them directly to the decoder. Our proposed PAUMER encoder adds a simple auxiliary decoder (a 1 × 1 convolution), and uses the predicted posterior entropy $\mathbb{H}[\hat{Y}|X]$ 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. Some parts of the input need 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 inputs' spatial location. Removal of patches from computation does not require 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 (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 Appendix 4.C

4.2.1 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 can verify the correctness of the network's predictions. We could decode after each layer $l \in \{1 \ldots L\}$, and stop processing 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 4.2.2 the entropy of predictions computed after every second layer in a segmenter. Specifically, we use a Segmenter with ViT-Ti backbone pre-trained 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 a higher overlap of correct and incorrect predictions' entropy in the initial layers. 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.

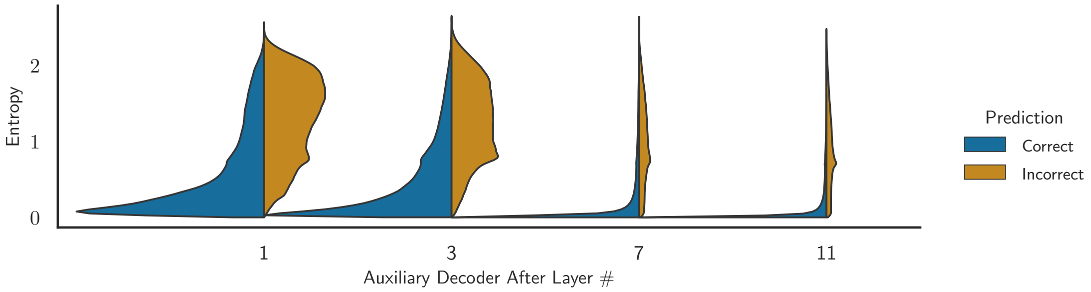


Figure 4.2.2: Violin plot of entropy of predictions at intermediate layers. In this figure, we plot the entropy distribution of the auxiliary predictions for 10% of images in the Cityscapes validation set. The x-axis marks after which layer the prediction was made. For each layer, the entropy distribution 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)

4.2.2 Training PAUMER- One training for many pause configurations

We base our network on Segmenter's architecture (Strudel et al., 2021) detailed in Appendix 4.C. A pause configuration (or configuration) refers to the proportion of patches paused at each layer of the network. An obvious method is to train and test the same patch-pausing configuration that satisfies our run-time requirements. Any changes to the run-time requirements require retraining the network. For this, 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 $\tau_l \sim \mathscr{U}[\tau_l^{\text{lo}}, \tau_l^{\text{hi}}]$, where $\mathscr{U}$ refers to a uniform distribution over the parameters. We compactly represent this in Python's dictionary notation as $\{l : \tau_l\}$; here $\tau_l = 0$ implies that no pausing is done at layer l, and we ignore these layers in our notation. To facilitate the patch pausing, we employ a single auxiliary decoder D_A, parametrized by a 1×1 convolution, after the operations of layer l (see Figure 4.2.1).

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

$$\mathscr{L}_{\text{main}} = \text{CE}(\boldsymbol{y}, \hat{\boldsymbol{y}}) \tag{4.1}$$

$$\mathscr{L}_{\text{aux}}^{l} = \text{CE}(\boldsymbol{y}, \hat{\boldsymbol{y}}^{\boldsymbol{l}}) \tag{4.2}$$

where $\boldsymbol{y}$ is the ground truth, and $\hat{\boldsymbol{y}}$ refers to the logits predicted by the main decoder (mask transformer), and $\hat{\boldsymbol{y}}^{\boldsymbol{l}}$ is output at the l^{th} layer using the auxiliary decoder. The total loss used to train is a combination of losses in Equations (4.1) and (4.2).

$$\mathscr{L}_{\text{total}} = \mathscr{L}_{\text{main}} + \lambda \mathscr{L}_{\text{aux}}^{i} \tag{4.3}$$

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

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

$$H_l := \mathbb{H}[\hat{Y}_l | X_l] = \mathbb{H}[\sigma(D_A(X_l))], \tag{4.4}$$

where σ is the softmax function applied to each pixel independently. With this entropy, we pause the computation of 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 4.1.1). 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 require padding to be processed efficiently on a GPU, adding a substantial overhead. Additionally, pausing a fixed amount of tokens allows for a deterministic computation time.

To reassemble in the original order the patches that have been fully processed and the ones that have been paused, we use a patch assembler module (see Appendix 4.A for PyTorch-like code). It takes X_L and X'_l, and reassembles them into the original shape of X_0. 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 reassembled output is finally fed into the decoder (mask segmenter head) to compute the segmentation map.

Inference

Our training procedure of randomized pause configurations gives us the advantage of choosing a pausing configuration that is informed by the run-time requirements i.e., mIoU and the number of images processed per second. This configuration can have multiple pause locations, each with different pause proportions. We show some results for configurations listed in Table 4.4.1. 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.

4.3 Related Work

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

Segmentation using Transformers

Transformers that were originally proposed for language processing tasks (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 impact the downstream segmentation performance (Wang et al., 2021b; 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, the number of floating point operations per second (FLOPs), or images processed per second.

Network modifications for efficiency

Several components of the transformer architecture have been improved by approximations and simplifications to the attention mechanism, reducing the computation in the MLP block (Tay et al., 2022). In the convolutional networks family, architectural modifications like MobileNet (Howard et al., 2017, 2019), Efficient Nets (Tan and Le, 2019), SwiftNets (Orsic et al., 2019) have been proposed with the explicit goal of improving performance throughputs.

Network pruning, where network weights are removed to reduce the operations by a network has been studied extensively (Kuzmin et al., 2019). Lower precision training and inference are used to reduce the memory footprint of the network (Dettmers et al., 2022; Wu et al., 2020; Micikevicius et al., 2018). These avenues improve network efficiency through design and modification, which can additionally influence the training dynamics. However, faster training is orthogonal to our work, and we are interested in faster inference methods.

Token sparsification methods

Orthogonal to the architectural improvements, recent work has focused on reducing the data processed, and our proposed method is a form of input-dependent reduction. Graves (2016) proposed Adaptive Computation Time (ACT), where the network's amount of processing for each input to an RNN is decided by determining a halting distribution. It was adapted to residual networks by Figurnov et al. (2017), which dynamically chooses 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 indicates 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 tokens for a given image and do not provide the flexibility to vary the number of patches dropped, as our method does.

Early-exit methods

Dynamic neural networks (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. 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) use 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 which predict the same class. DeeBERT (Xin et al., 2020) proposes a two-stage training, where the auxiliary decoders are trained after the main networks are trained and frozen. Li et al. (2017) proposes 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 minimal 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 examine the patch-pausing problem of semantic segmentation. Also, the randomized training presented in Section 4.2.2 has not been used in this context, though similar ideas to reduce network width were studied in slimmable networks (Yu et al., 2019).

4.4 Experiments

4.4.1 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 $25,574$ images 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:

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

We examined an additional simple baseline of random pausing (without training) and found the results not competitive enough to warrant reporting here. Also, some methods in Section 4.3 can drop 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.

Table 4.4.1: Table of configurations. Each column represents a pause configuration, e.g. the first column represents the configuration $\{3 : 0.2\}$ implying pausing 20% of tokens after layer 3, using the notation introduced in Section 4.2.2. Each configuration here corresponds to a marker in Figures 4.4.1a, 4.4.1b and 4.K.1.

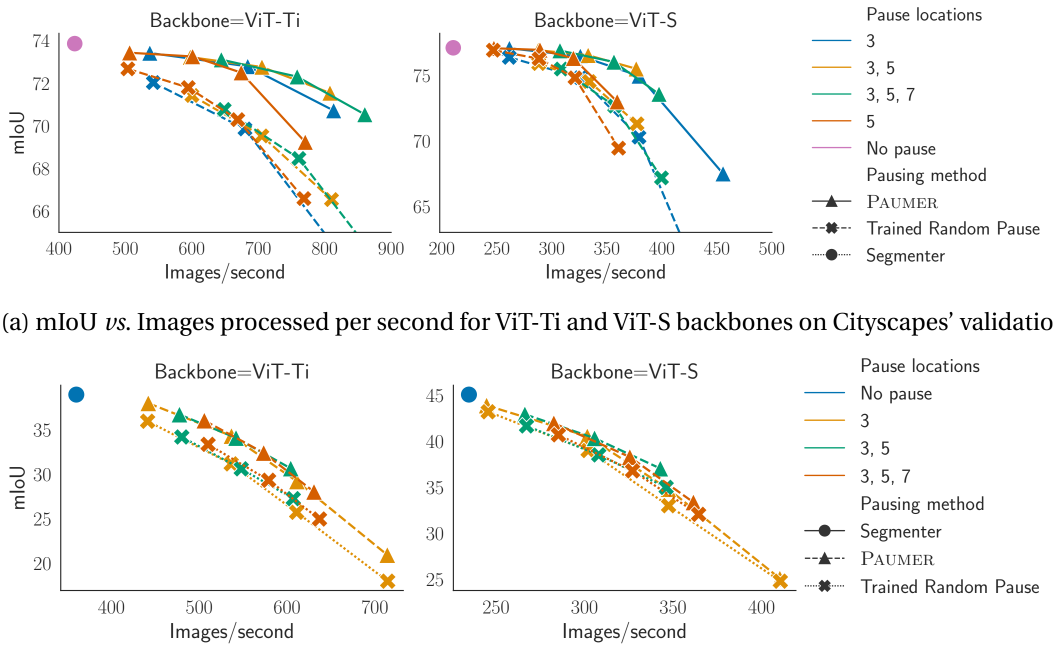
Pause Layer	Pause configurations												
3	0.2	0.4	0.6					0.2	0.3	0.4	0.2	0.3	0.4
5				0.2	0.4	0.6	0.8	0.2	0.3	0.4	0.2	0.3	0.4
7											0.2	0.3	0.4

Training hyperparameters and Inference Configurations:

For our main experiments, we use Segmenter with ViT-Ti and ViT-S backbones (details in Appendix 4.B). During training, we follow the procedure in Section 4.2.2 for every network and dataset, and in particular, we pause a random amount of tokens $\tau_l \sim \mathcal{U}[0.2, 0.8]$ at a random layer $l \in \{3,4,5,6,7,8,9\}$. We initialize the model for our training with pretrained segmenter weights, as we find this results in better performance, and train the model for $80,000$ steps for Cityscapes and $160,000$ steps for ADE20K. The auxiliary loss-weight λ is set to 0.1. Rest of the hyperparameters as kept the same as in Strudel et al. (2021). We implement our method using mmsegmentation (MMSegmentation Contributors, 2020) and use their pretrained models whenever available. At inference, we test the networks with the pause configurations in Table 4.4.1. This list of pause configurations is not exhaustive and does not hold any specific importance, but have been chosen to show the efficiency of our method in trading off mIoU for higher IMPS.

Choosing pause configurations:

Determining the appropriateness of a pausing configuration incurs little additional cost, as it only requires inference with a validation set for each configuration of interest (see Figures 4.4.1a and 4.4.1b). 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 4.H.1 and 4.H.2) by only timing them, and then evaluating the mIoU of the ones that meet the 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 4.H.1), and we



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

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

Figure 4.4.1: Results on Cityscapes and ADE20K for our proposed method PAUMER. Each marker is a configuration from Table 4.4.1. 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.

pick the first layer and proportion. We fix this layer and ratio, then sweep through pausing configurations at a second layer (Figure 4.H.2). We repeat this procedure until adding more layers is no longer beneficial.

4.4.2 Results

Performance analysis

In Figure 4.4.1a, 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 leftmost point corresponds to the original Segmenter model, the solid line to our proposed pausing strategy, and the dashed line to random pausing with training. For both ViT-Ti and ViT-S backbones, a 50% increase of IMPS can be achieved with a 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 4.4.1b. For the ViT-Ti backbone, we see that for a 50% increase in

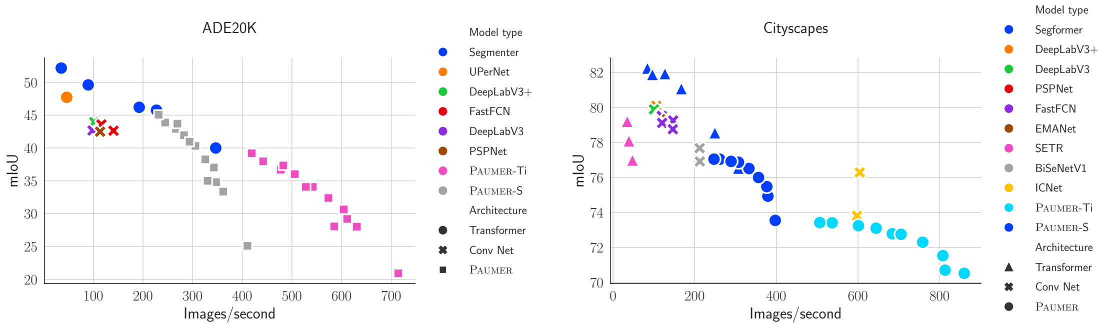


Figure 4.4.2: Performance comparison for ADE20K and Cityscapes. We compare to pretrained models available in mmsegmentation (MMSegmentation Contributors, 2020) for each of these datasets. Architectures devised for speed or accuracy outperform us on those criteria, but PAUMER has the unique advantage that we can trade off one for the other using a tunable hyperparameter. We use different colors for different architectures and different shapes for architectural families.

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 classified correctly.

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 4.4.2, we compare our method to a broad array of architectures for which pretrained models are available in mmsegmentation (MMSegmentation Contributors, 2020). We include both convolution-based architectures, and transformer-based ones. In the transformer family, available networks have focussed on improving the performance, and thus are slower, but more accurate than the PAUMER family of models. CNN-based ones 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.

4.5 Discussion

The improvement in images processed per second is obtained by reducing the number of patches processed. This might not necessarily hold 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.

In addition to architecture dependence, patch-pausing's performance may depend on the dataset. 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.

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

4.6 Conclusion

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

Appendix - Chapter 4

Appendix 4.A Pseudocode for Patch Pauser and Assembler

In Algorithm 4, we present pseudocode for both the patch-pausing mechanism and patch assembler. The code is for illustrative purposes.

Appendix 4.B Backbone details

This chapter uses 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 4.B.1, we describe the main architecture details of the ViT backbones.

Table 4.B.1: ViT architectural details used

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

Appendix 4.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 is then transformed using a linear embedding layer $E : \mathbb{R}^{3P^2} \to \mathbb{R}^D$, giving a feature representation $X_0 \in \mathbb{R}^{N\times D}$, as shown in Figure 4.2.1.

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

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

Procedure 4 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)

    topk_inds = entropy.topk(to_process_count)## topk is faster than sorting on GPU.
    kept_tokens = tokens[:, topk_inds]

    keep_indices.append(topk_inds)
    paused_tokens.append(tokens) ## This is $X_l'$

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

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

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

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

    return X_L
```

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

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

Appendix 4.D Patch-pausing's limitations

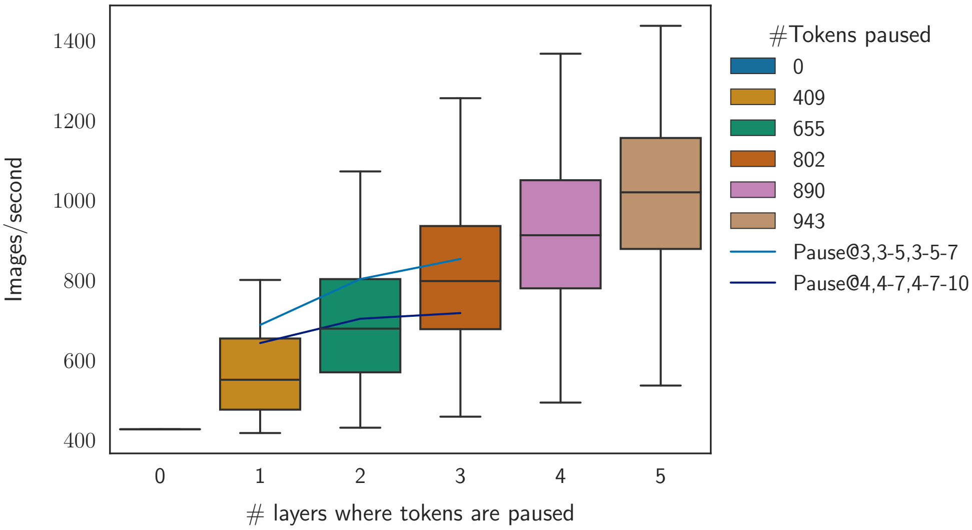


Figure 4.D.1: Evolution of the throughput with the number of layers using patch-pausing.

In Figure 4.4.1b, we investigated pausing up to three times in the network. It is tempting to pause at more layers, but this entails two additional costs: first to compute the entropy and rank the patches, and then the patch assembly as detailed in Appendix 4.A. These costs are (ideally) offset by the reduction in the 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 $\tau = 0.4$.

In Figure 4.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 indicate 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 4.D.1 and holds even as the number of layers to pause 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 4.D.1. We see that pausing more has benefits in the number of images processed, but 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 4.4.1a and 4.4.1b). Thus, the primary limitation is posed by the drop in mIoU rather than throughput.

Appendix 4.E Influence of the training pause ratio $\tau_l - \tau_h$

In Figure 4.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 4.2.2. We see that the results for various train pause ranges are 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), and when the performance difference is higher (≈ 1%). More aggressive pausing at training seems beneficial (0 – 0.9 performs the best). However, as a middle ground to the multiple configurations investigated, we use 0.2 – 0.8 for all our experiments.

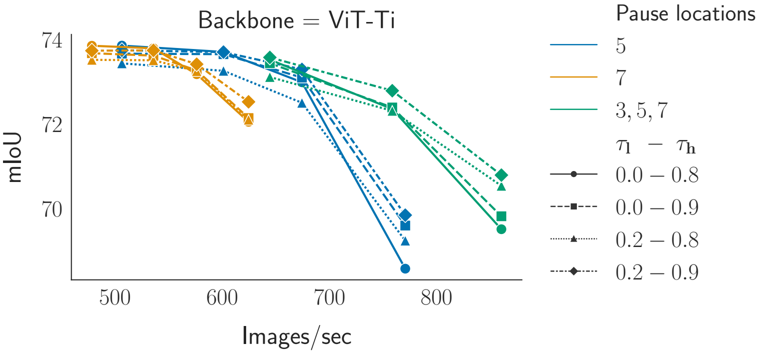


Figure 4.E.1: mIoU *vs.* throughput for different values of the range of the pause ratio $\tau_l - \tau_h$ introduced in Section 4.2.2.

Appendix 4.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 4.4.1a. This illustrates that we can choose a pause configuration that fits our run-time requirements (IMPS), and that it works with the performance specified here.

Table 4.F.1: Trading off mIoU for speed. In Section 4.4.2, 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.

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

Appendix 4.G Influence of the auxiliary loss weight λ

In Figure 4.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 4.2.2. 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 a lesser or a larger number of tokens.

Appendix 4.H Interplay of Pause location and Pausing proportion τ

We showed the results for some configurations in Table 4.4.1. In this section, we study the interplay between pause location and pause proportion τ. We show a sweep over pause configurations in the Figure 4.H.1. For each layer, we choose 20 pause proportions in $(0, 1)$. It is apparent that dropping at a later layer results in a lesser drop in mIoU but also does not result in a large gain in IMPS. Thus depending on the desired run-time, one can choose a

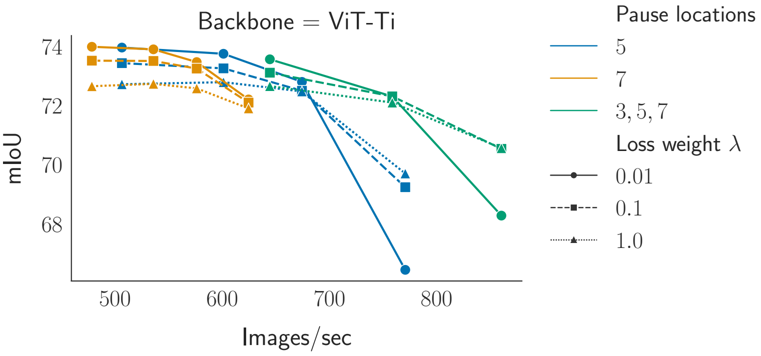


Figure 4.G.1: mIoU *vs.* throughput for different values of the auxiliary loss weight λ introduced in Section 4.2.2. Lower values of λ lead to better performance when pausing few tokens but worse performance when pausing more, and conversely for higher values of λ. Our chosen value of $\lambda = 0.1$ is a trade-off that can also be modified depending on the use case.

pause location and pause proportion that gives the required performance.

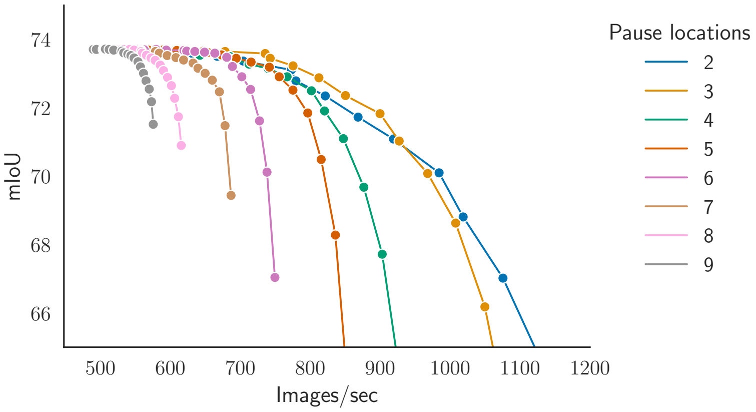


Figure 4.H.1: Pausing once at various layers for various pause proportions for ViT-Ti on Cityscapes. We see that the highest gains in IMPS are achieved by dropping in earlier layers.

To examine this further, we plot the performance of pausing twice in Figure 4.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 4.H.2. We also focus on the first pausing layer, 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 and that pausing small proportions at these layers leads to a slightly higher drop in mIoU than pausing a higher amount in later layers.

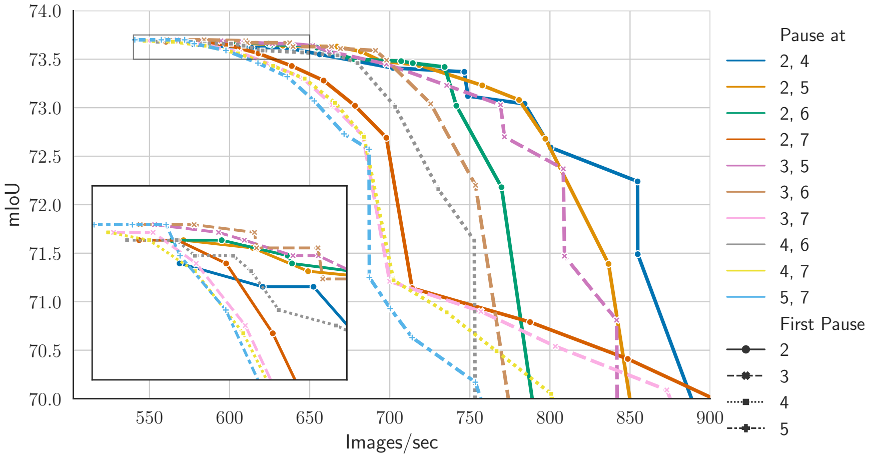


Figure 4.H.2: Pareto front of pausing at two layers for ViT-Ti on Cityscapes.

Appendix 4.I Using Early Exit at test time

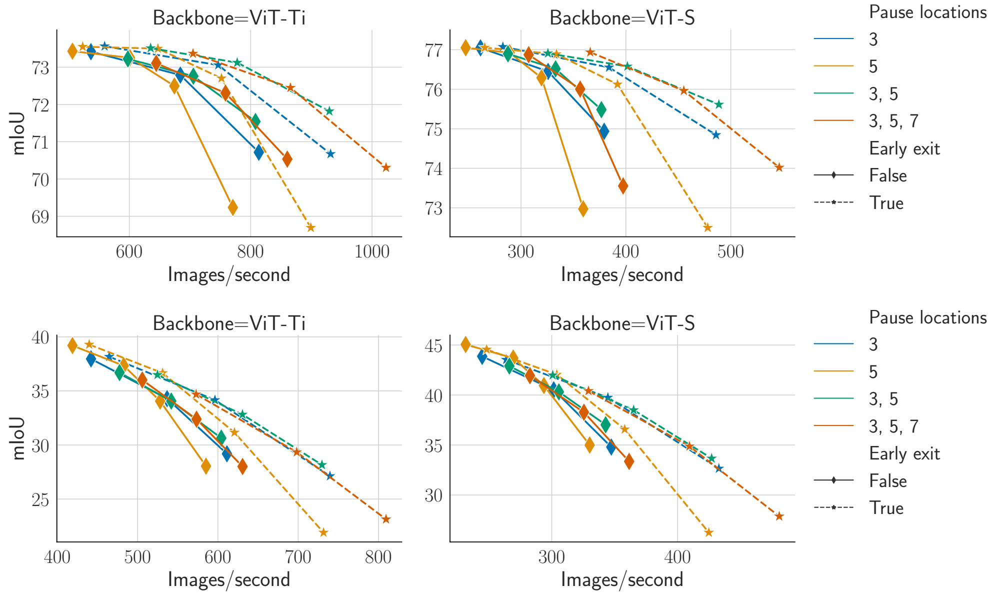


Figure 4.I.1: Comparison when segmenting with and without early exit at test time on CityScapes and ADE20K.

In Figure 4.I.1, we study the use of early exit on a trained PAUMER. Early exit (see works in Section 4.3) refers to stopping the processing of 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 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 4.4.1a, a network with early exit runs at higher throughput with fewer FLOPs by design, but it may run with a lower mIoU. On Figure 4.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 may be needed for different datasets.

Appendix 4.J Comparing SETR, Segmenter, EarlyExit using Segmenter

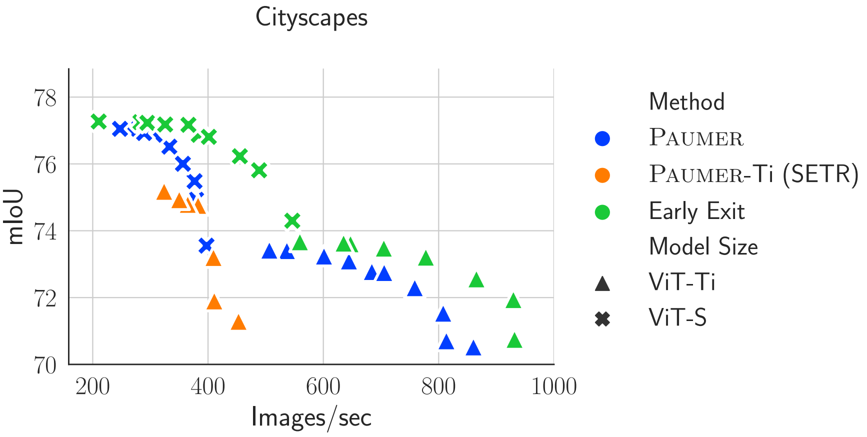


Figure 4.J.1: We compare PAUMER on Segmenter, on SETR, and using an early exit variant of PAUMER on Segmenter. Early exit fares better than PAUMER on Cityscapes. This may be attributable to the dataset, as Cityscapes might not need a more complex mask decoder for accuracy.

In Appendix 4.I we showed the results of early-exit. Here we examine the best performances obtained for each throughput across pause configurations. We estimate this by computing the skyline queries. We see that early exit performs consistently better on Cityscapes.

Additionally, we implement our patch-pausing strategy, PAUMER, on the network architecture SETR (Zheng et al., 2021). SETR's performance drops off more rapidly than Segmenter-based patch-pausing. Note that we adapt SETR's PUP decoder to use it with a ViT-Tiny backbone. In particular, we reduce the number of channels of the decoder to 192, the number of convolutions in the decoder from 4 to 2, and the upscale factor from 2 to 4.

Appendix 4.K Importance of task-specific pretraining

In Figure 4.K.1, we study the importance of initialization. We compare ViT-Ti pretrained on Cityscapes (task-specific), and pretrained on ImageNet (generic) and study their impact on performance. When using a generic pretrained model, our training with PAUMER is increased to 160K iterations instead of 80K. It is apparent that using a task-specific pretrained model

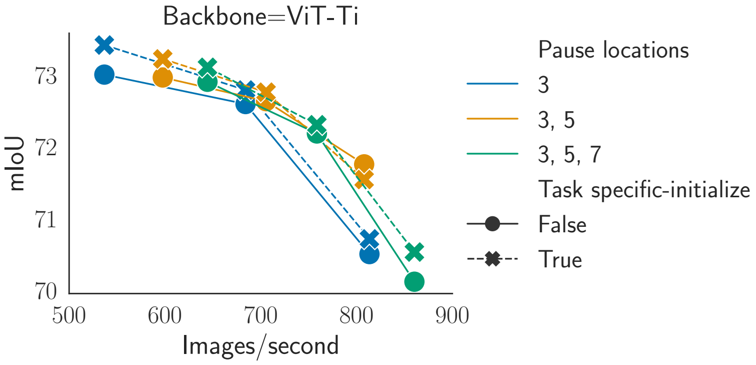


Figure 4.K.1: Importance of initialization for PAUMER. Task-specific initialization benefits performances. We train ViT-Ti PAUMER: solid lines are from ImageNet pretrained backbones, dashed lines are from Cityscapes pretrained Segmenters. Each marker is a configuration from Table 4.4.1.

brings a relatively consistent benefit in this context.

Appendix 4.L Entropy as a measure of patch-pausing

In Section 4.2.1, we argued that entropy is a reasonable indicator of completion of processing. For that, we used the illustration in Figure 4.2.2 to show the increase in separation of entropy histograms for pixels predicted correctly and incorrectly. We expand that in Figure 4.L.1, to analyze that to each class individually. The larger separation in entropy in the first few layers of the network in prevalent in large classes like road, building, vegetation, car. As seen in Figure 4.1.1 too, these larger classes are paused to gain IMPS. Smaller, rarer classes like train, motorcycle, rider are tougher to learn and are unlikely to be paused (as evidenced by their higher entropy).

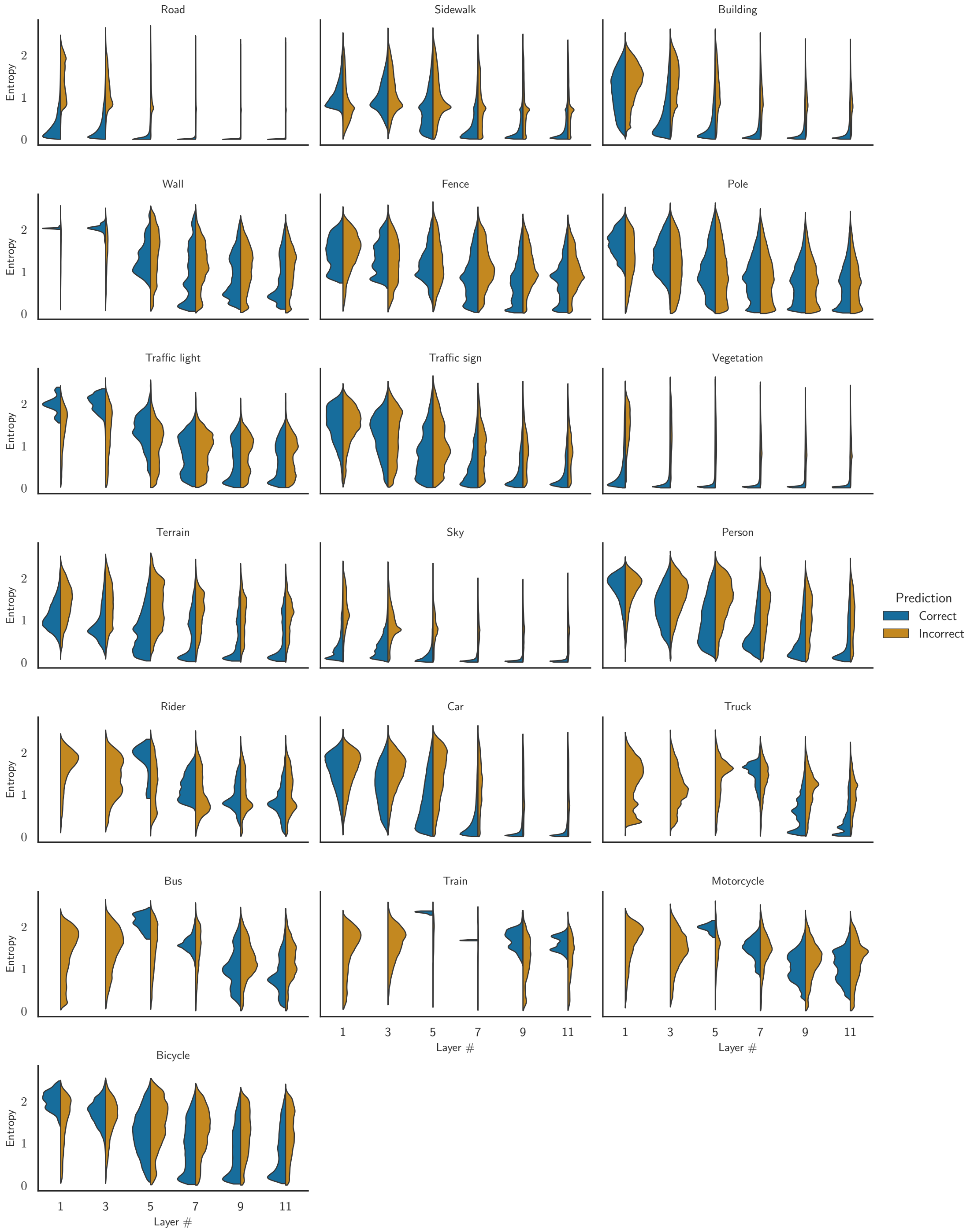


Figure 4.L.1: Entropy per layer for each class of Cityscapes. Continues Figure 4.2.2.

5 Optimizer Benchmarking Needs to Account for Hyperparameter Tuning

We presented work on improving the run-time computational efficiency of segmentation transformers. In this chapter, we turn our focus onto the omnipresent problem of evaluation methods that are computation-aware. We present work on benchmarking strategies for deep learning with a focus on optimizers used in the field. The arguments presented are broadly applicable to the evaluation of deep learning models, and several concurrent works have studied that problem (Dodge et al., 2019; Musgrave et al., 2020). This chapter is based on

> Teja, P., Mai, F., Vogels, T., Jaggi, M., and Fleuret, F. (2020). Optimizer benchmarking needs to account for hyperparameter tuning. In III, H. D. and Singh, A., editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 9036–9045. PMLR.

5.1 Introduction

With the ubiquity of deep learning in various applications, a multitude of first-order stochastic optimizers (Robbins and Monro, 1951) has been in vogue. They have varying algorithmic components like momentum (Sutskever et al., 2013) and adaptive learning rates (Tieleman and Hinton, 2012; Duchi et al., 2011; Kingma and Ba, 2015). As the field grows with newer variants being proposed, the standard method to benchmark the performance of these optimizers has been to compare the best possible generalization performance. While it is certainly an important characteristic to be taken into account, we argue that in practice, an even more important characteristic is the performance achievable with available resources. A similar view of performance measurement has been recently argued for in the deep learning community owing to the strong debate on sustainable and GreenAI (Strubell et al., 2019; Schwartz et al., 2019).

The performance of optimizers strongly depends on the choice of hyperparameter values such as the learning rate. In the machine learning research community, the sensitivity of models to hyperparameters has been of great debate recently, where in multiple cases, reported

Table 5.1.1: Experimental settings shown in the original papers of popular optimizers. The large differences in test problems and tuning methods make them difficult to compare. γ denotes learning rate, μ denotes momentum, λ is the weight decay coefficient.

Method	Datasets	Network architecture	Parameter tuning methods
SGD with momentum (Sutskever et al., 2013)	Artificial datasets MNIST	Fully-connected LSTM	$\mu = 0.9$ for first 1000 updates then $\mu \in \{0, 0.9, 0.98, 0.995\}$. other schedules for μ are used & $log_{10}(\gamma) \in \{-3, -4, -5, -6\}$
Adagrad (Duchi et al., 2011)	ImageNet ranking Reuters RCV1 MNIST KDD Census	Single layer Handcrafted features Histogram features	Perfomance on dev-set
Adam (Kingma and Ba, 2015)	IMDb MNIST CIFAR 10	Logistic regression Multi-layer perceptron Convolutional network	$\beta_1 \in \{0, 0.9\}$ $\beta_2 \in \{0.99, 0.999, 0.9999\}$ $log_{10}(\gamma) \in \{-5, -4, -3, -2, -1\}$
AdamW (Loshchilov and Hutter, 2019)	CIFAR 10 ImageNet 32×32	ResNet CNN	$log_2(\gamma) \in \{-11, -10 \cdots -1, 0\}$ $log_2(\lambda) \in log_2(10^{-3}) + \{-5, -4, \ldots, 4\}$

model advances did not stand the test of time because they could be explained by better hyperparameter tuning (Lucic et al., 2018; Melis et al., 2018; Henderson et al., 2018; Dodge et al., 2019). This has led to calls for using automatic hyperparameter optimization methods (HPO) with a fixed budget for a fairer comparison of models (Sculley et al., 2018; Hutter et al.,

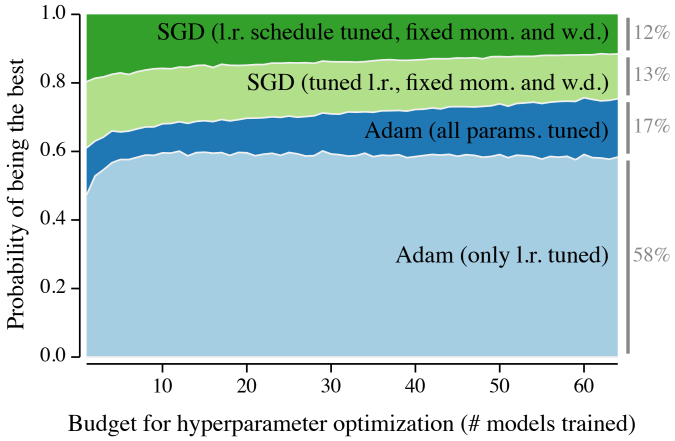


Figure 5.1.1: Hyperparameter optimization budget affects the performance of optimizers. We show the probability of finding a hyperparameter configuration for an optimizer that performs the best at a given search budget on any task (sampled from our benchmark). This is encoded by the height of the respective area in the chart. Generally, we see that tuning more hyperparameters becomes more useful with higher budgets. On our 9 diverse tasks that include vision problems, natural language processing, regression, and classification, tuning only the learning rate for Adam is the most reliable option, even at large budgets.

2019; Eggensperger et al., 2019). This eliminates biases introduced by humans through manual tuning. For industrial applications, automated machine learning (AutoML, Hutter et al., 2019), which has automatic hyperparameter optimization as one of its key concepts, is becoming increasingly more important. In all these cases, an optimization algorithm that achieves good performances with relatively little tuning effort is arguably substantially more useful than an optimization algorithm that achieves top performance but reaches it only with a lot of careful tuning effort. Hence, we advocate that benchmarking the performance obtained by an optimizer must not only avoid manual tuning as much as possible but also has to account for the cost of tuning its hyperparameters to obtain that performance.

Works that propose optimization techniques show their performance on various tasks as depicted in Table 5.1.1. It is apparent that the experimental settings, as well as the network architectures tested, widely vary, hindering a fair comparison. The introduction of benchmarking suites like DeepOBS (Schneider et al., 2019) have standardized the architectures tested on. However, this does not fix the problem of selecting the hyperparameters fairly. Indeed, recent papers studying optimizer performances may employ grid search to select the best values, but the search spaces are still selected on a per-dataset basis, introducing significant human biasSchneider et al. (2019); Wilson et al. (2017); Shah et al. (2018); Choi et al. (2019). Moreover, as only the best-obtained performance is reported, it is unclear how a lower search budget would impact the results. This leads us to the question: how easy is an optimizer to use, i.e., how quickly can an automatic search method find a set of hyperparameters for that optimizer that result in satisfactory performance?

In this work, we introduce a simple benchmarking procedure for optimizers that addresses the discussed issues. By evaluating on a wide range of 9 diverse tasks, we contribute to the debate of adaptive *vs.* non-adaptive optimizers (Wilson et al., 2017; Shah et al., 2018; Chen and Gu, 2018; Choi et al., 2019). To reach a fair comparison, we experiment with several SGD variants that are often used in practice to reach good performance. Although a well-tuned SGD variant is able to reach top performance in some cases, our overall results clearly favor Adam (Kingma and Ba, 2015), as shown in Figure 5.1.1.

5.2 The Need to Incorporate Hyperparameter Optimization into Benchmarking

The problem of optimizer benchmarking is two-fold as it needs to take into account

1. how difficult it is to find a good hyperparameter configuration for the optimizer,
2. the absolute performance of the optimizer.

To see why both are needed, consider Figure 5.2.1, which shows the loss of four different optimizers as a function of their only hyperparameter θ (by assumption). If we only consider requirement #1, optimizer C would be considered the best, since every hyperparameter value is the optimum. However, its absolute performance is poor, making it of low practical value. Moreover, due to the same shape, optimizers A and B would be considered equally good, although optimizer A clearly outperforms B. On the other hand, if we only consider requirement #2, optimizers B and D would be considered equally good, although optimizer D's optimum is harder to find.

As we discuss in Section 5.3, no existing work on optimizer benchmarking takes both requirements into account. Here we present a formulation that does so in Algorithm 5.

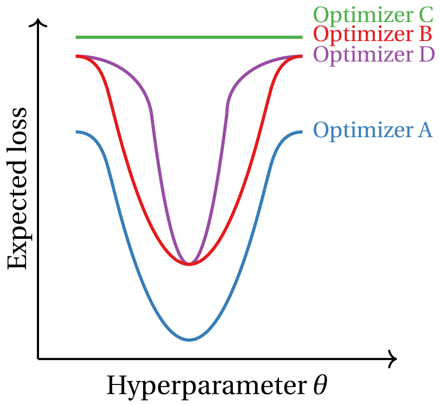


Figure 5.2.1: Illustration. It is important to consider both the absolute performance of optimizers as well as the tuning effort to get to good performances.

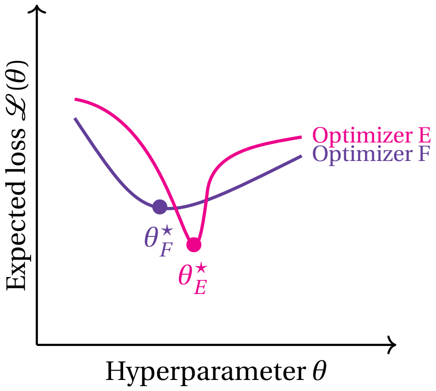


Figure 5.2.2: Illustration. While optimizer E can achieve the best performance after careful tuning, optimizer F is likely to provide better performance under a constrained HPO budget.

We have already established that fairly comparing optimizers needs to account for how easy it is to find good hyperparameter values. When proposing new optimization methods, algorithm designers often only specify the permissible set of values the hyperparameters can take, and informally provide some intuition of good values. For example, for Adam, Kingma and Ba (2015) bound β_1, β_2 to $[0, 1)$ and specify that they should be close to 1. These are valuable information for users of their algorithm, but they do not allow to formally incorporate that information into a benchmarking procedure. Instead, we argue that we need to redefine what constitutes an optimizer so that prior knowledge over reasonable hyperparameter values is included.

Definition. *An optimizer is a pair $\mathcal{M} = (\mathcal{U}_\Theta, p_\Theta)$, which applies its update rule $\mathcal{U}(S_t; \Theta)$ at each step t depending on its current state S_t. It is parameterized through N hyperparameters $\Theta = (\theta_1, \ldots, \theta_N)$ with respective permissible values $\theta_i \in H_i \ \forall i$, and $p_\Theta : (\Theta \to \mathbb{R})$ defines a probability distribution over the hyperparameters.*

In the example above, we could describe the intuition that β_1, β_2 should be close to 1 by the random variables $\hat{\beta}_1 = 1 - 10^{c_1}, \hat{\beta}_2 = 1 - 10^{c_2}$, where $c_1, c_2 \sim U(-10, -1)$.

Procedure 5 Benchmark with ‘expected quality at budget’

```
Input: Optimizer $O$, cross-task hyperparameter prior $\Theta_O$, task $T$, tuning budget $B$
Initialization: Pre-compute a library of size $\gg B$ with validation losses achieved on task $T$
with optimizer $O$ using hyper-parameters sampled from $\Theta_O$.
Initialize $list \leftarrow []$.
for $R$ repetitions do
    Simulate hyperparameter search with budget $B$:
    – $S \leftarrow$ sample $B$ elements from library.
    – $list \leftarrow [\textsc{best}(S), \ldots list]$.
end for
return $\textsc{mean}(list)$, or other statistics
```

Let $\mathscr{L}(\Theta_1)$ refer to the performance (say, test loss) of $\mathscr{M}$ with the specific hyperparameter choice Θ_1.

Let us assume that there are two optimizers E & F, both with a single hyperparameter θ, but no prior knowledge of particularly good values, i.e., the prior is a uniform distribution over the permissible range. Let their loss surface be $\mathscr{L}_E$ and $\mathscr{L}_F$, respectively. As Figure 5.2.2 shows, the minimum of $\mathscr{L}_E$ is lower than that of $\mathscr{L}_F$ (denoted by $\theta_E^\star$ and $\theta_F^\star$) i.e., $\mathscr{L}_E(\theta_E^\star) < \mathscr{L}_F(\theta_F^\star)$. However, the minimum of $\mathscr{L}_E$ is much sharper than that of $\mathscr{L}_F$, and in most regions of the parameter space F performs much better than E. This makes it easier to find configurations that perform well. This makes optimizer-F an attractive option when we have no prior knowledge of the good parameter settings. Previous benchmarking strategies compare only $\theta_E^\star$ and $\theta_F^\star$. It is obvious that in practice, optimizer-F may be an attractive option, as it gives ‘good-enough’ performance without the need for a larger tuning budget.

We incorporate the relevant characteristics of the hyperparameter optimization surface described above into benchmarking through Algorithm 5. In the proposed protocol, we use Random Search (Bergstra and Bengio, 2012) with the optimizers’ prior distribution to search the hyperparameter space. The quality of the optimizers can then be assessed by inspecting the maximum performance attained after k trials of random search. However, due to the stochasticity involved in random search, we would usually have to repeat the process many times to obtain a reliable estimate of the distribution of performances after budget k. We instead use the bootstrap method (Tibshirani and Efron, 1993) that re-samples from the empirical distribution (termed *library* in Algorithm 5). When we need the mean and variance of the best-attained. performance after budget k, we use the method proposed by Dodge et al. (2019) to compute them exactly in closed form. We provide the details of the computation in Appendix 5.D.

Our evaluation protocol has distinct advantages over previous benchmarking methods that tried to incorporate automatic hyperparameter optimization methods. First, our evaluation protocol is entirely free of arbitrary human choices that bias benchmarking: The only free parameters of random search itself are the search space priors, which we view as part of the optimizer. Secondly, since we measure and report the performance of Random Search with low budgets, we implicitly characterize the loss surface of the hyperparameters: In terms of Figure 5.2.2, optimizer-F with its wide minimum will show good performance with low budgets, whereas optimizer-E can be expected to show better performance with high budgets. Such characterizations would not be possible if one only considered the performance after exhausting the full budget. Finally, our evaluation protocol allows practitioners to choose the right optimizer for their budget scenarios.

Discussion of alternative choices In theory, our general methodology could also be applied with a different hyperparameter optimization technique that makes use of prior distributions, e.g., drawing the set of initial observations in Bayesian methods. However, those usually have additional hyperparameters, which can act as potential sources of bias. Moreover, the bootstrap method is not applicable when the hyperparameter trials are drawn dependently, and repeating the hyperparameter optimization many times is practically infeasible.

In our protocol, we consider the number of random search trials as the unit of budget, and not computation time. This is done so as to not violate the independence assumption in the method by Dodge et al. (2019) in Appendix 5.D. We, empirically, show in Appendix 5.F that the conclusions of this work are still valid when time is used as the unit of budget as well.

5.3 Related Work

Benchmarking of optimizers is a relatively unstudied subject in literature. Schneider et al. (2019) recently released a benchmark suite for optimizers that evaluates their peak performance and speed, and the performance measure is assessed as the sensitivity of the performance to changes in the learning rate. Our work primarily takes its genesis from the study by Wilson et al. (2017) that finds SGD-based methods as easy to tune as adaptive gradient methods. They perform grid search on manually chosen grids for various problems and conclude that both SGD and Adam require similar grid search effort. However, their study lacks a clear definition of what it means to be tunable (easy-to-use) and tunes the algorithms on manually selected, dataset-dependent grid values. The study by Shah et al. (2018) applies a similar methodology and comes to similar conclusions regarding performance. Since both studies only consider the best parameter configuration, their approaches cannot quantify the efforts expended to find the hyperparameter configuration that gives the best setting; they would be unable to identify the difference between optimizer among B and D in Figure 5.2.1. In contrast, the methodology in our study is able to distinguish all the cases depicted in Figure 5.2.1.

There exist few works that have tried to quantify the impact of hyperparameter setting in ML algorithms. investigate tunability formally. For decision tree models, Mantovani et al. (2018) count the number of times the tuned hyperparameter values are (statistically significantly) better than the default values. Probst et al. (2019) define the tunability of an ML algorithm as the performance difference between a reference configuration (e.g., the default hyperparameters of the algorithm) and the best possible configuration on each dataset. This metric is comparable across ML algorithms, but it disregards entirely the absolute performance of ML algorithms; thereby being unable to differentiate between optimizers B and D in Figure 5.2.1.

In a concurrent study, Choi et al. (2019) shows that there exists a hierarchy among optimizers such that some can be viewed as specific cases of others and thus, the general optimizer should never under-perform the special case (with appropriate hyperparameter settings). Like in our study, they suggest that the performance comparison of optimizers is strongly predicated on the hyperparameter tuning protocol. However, their focus is on the best possible performance achievable by an optimizer and does not take into account the tuning process. Also, the presence of a hierarchy of optimizers does not indicate how easy it is to arrive at the hyperparameter settings that help improve the performance of the more *general* optimizer. Moreover, while the authors claim their search protocol to be relevant for practitioners, the search spaces are manually chosen *per dataset*, constituting a significant departure from a realistic AutoML scenario considered in our work. Since the focus is only on the best attainable performance, it construed as being benchmarking theoretically infinite budget scenarios.

In recent work, Dodge et al. (2019) proposes to use the performance on the validation set along with the test set performance. They note that the performance conclusions reached by previously established NLP models differ widely from the published works when an additional hyperparameter tuning budget is considered. They recommend a checklist to report for scientific publications that includes details of compute infrastructure, runtime, and more importantly, the hyperparameter settings used to arrive at those results like bounds for each hyperparameter, HPO budget, and tuning protocols. They recommend using expected validation performance at a given HPO budget as a metric, along with the test performance.

There has been recent interest in optimizers that are provably robust to hyperparameter choices, termed the aProx family (Asi and Duchi, 2019a,b). Asi and Duchi experimentally find that, training a Residual network (He et al., 2016b) on CIFAR-10, SGD converges only for a small range of initial learning rate choices, whereas Adam exhibits better robustness to learning rate choices; their findings are in line with our experiments that it is indeed easier to find good hyperparameter configurations for Adam.

Metz et al. (2020) propose a large range of tasks and propose to collate hyperparameter configurations over those. They show that the optimizer settings thus collated, which are problem agnostic like us, generalize well to unseen tasks too.

5.4 Optimizers and Their Hyperparameters

In Section 5.2, we argued that an optimizer is a combination of an update equation and the probabilistic prior on the search space of the hyperparameter values. Since we are considering a setup akin to AutoML with as little human intervention as possible, these priors have to be independent of the dataset. As we view the hyperparameter priors as a part of the optimizer itself, we argue that they should be prescribed by algorithm designers themselves in the future. However, in the absence of such prescriptions for optimizers like Adam and SGD, we provide a simple method to estimate suitable priors in Section 5.4.2.

5.4.1 Parameters of the Optimizers

To compare the tunability of adaptive gradient methods to non-adaptive methods, we chose the most commonly used optimizers from both the strata; SGD and SGD with momentum for non-adaptive methods, and Adagrad and Adam for adaptive gradient methods. Since adaptive gradient methods are said to work well with their default hyperparameter values already, we additionally employ a default version of Adam where we only tune the initial learning rate and set the other hyperparameters to the values recommended in the original paper (Kingma and Ba, 2015) (termed Adam-LR). Such a scheme has been used by Schneider et al. too. A similar argument can be made for SGD with momentum (termed SGD-M): thus, we experiment with a fixed momentum value of 0.9 (termed SGD-M^C), which we found to be the most common momentum value to lead to good performance during the calibration phase.

In addition to standard parameters in all optimizers, we consider weight decay with SGD too. SGD with weight decay can be considered an optimizer with two steps where the first step is to scale current weights with the decay value, followed by a normal descent step (Loshchilov and Hutter, 2019). Therefore, we conduct two additional experiments for SGD with weight-decay: one where we tune weight-decay along with momentum (termed SGD-MW), and one where we fix it to 10^{-5} (termed SGD-M^{C}W^C) along with the momentum being fixed to 0.9, which again is the value for weight decay we found to be the best during calibration. We incorporate a “Poly” learning rate decay scheduler ($\gamma_t = \gamma_0 \times (1 - \frac{t}{T})^p$) (Liu et al., 2015) for SGD-M^{C}W^C (termed SGD-M^{C}D). This adds only one tunable hyperparameter (exponent p). We also experimented with Adam with a learning rate decay scheduler (termed Adam-W^{C}D), but reserve this discussion for the Appendix 5.B, as it did not yield sizeable improvements over Adam-LR or Adam in the problems tested. The full list of optimizers we consider is provided in Table 5.5.2, out of which we discuss Adam-LR, Adam, SGD-M^{C}W^C, SGD-MW, and SGD-M^{C}D in the main chapter. The rest are presented in Appendix 5.B.

Manually defining a specific number of epochs can be biased towards one optimizer, as one optimizer may reach good performance in the early epochs of a single run, and another may reach higher peaks more slowly. In order to alleviate this, it would be possible to add the number of training epochs as an additional hyperparameter to be searched. Since this would incur an even higher computational cost, we instead use the validation set performance as a stopping criterion. Thus, we stop training when the validation loss plateaus for more than 2 epochs or if the number of epochs exceeds the predetermined maximum number set in DeepOBS.

5.4.2 Calibration of Hyperparameter Prior Distributions

As mentioned previously, we use random search for optimizing the hyperparameters, which requires distributions of random variables to sample from. Choosing poor distributions to sample from impacts the performance, resulting in unfair comparisons, and may break requisite properties (e.g. learning rate is non-negative). For some parameters listed in Table 5.4.1, obvious bounds exist due to their mathematical properties or have been prescribed by the optimizer designers themselves. For example, Kingma and Ba (2015) bound β_1, β_2 to $[0, 1)$ and specify that they are close to 1. In the absence of such prior knowledge, we devise a simple method to determine the priors.

We use Random Search on a large range of admissible values on each task specified in DeepOBS to obtain an initial set of results. We then retain the hyperparameters which resulted in performance within 20% of the best result obtained. For each of the hyperparameters in this set, we fit the distributions in the third column of Table 5.4.1 using maximum likelihood estimation. Several recent works argue that there exists a complex interplay between the hyperparameters (Smith et al., 2018; Shallue et al., 2019), but we did not find modeling these to be helpful (Appendix 5.H). Instead, we make a simplifying assumption that all the hyperparameters can be sampled independently of each other. We argue that these distributions are appropriate; the only condition on the learning rate is non-negativity that is inherent to the log-normal distribution, momentum is non-negative with a usual upper bound of 1, βs in Adam have been prescribed to be less than 1 but close to it, ϵ is used to avoid division by zero and thus is a small positive value close to 0. We did not include p of the learning rate decay schedule in the calibration step due to computational constraints and chose a fixed plausible range such that the value used by Liu et al. (2015) is included. We report the parameters of the distributions obtained after the fitting in Table 5.4.1. The calibration step is not included in computing the final performance scores, as the calibrated priors are re-usable across tasks and datasets.

5.5 Experiments and Results

To assess the performance of optimizers for the training of deep neural networks, we benchmark using the open-source suite DeepOBS (Schneider et al., 2019). The architectures and datasets we experiment with are given in Table 5.5.1. We refer the reader to Schneider et al. (2019) for specific details of the architectures. To obtain a better balance between vision and NLP applications, we added an LSTM network with the task of sentiment classification in the IMDB dataset (Maas et al., 2011), details of which are provided in Appendix 5.A.

We aim to answer two main questions with our experiments: First, we look at the performance of various optimizers examined. Related to this, we investigate what effect the number of hyperparameters being tuned has on the performance at various budgets (Section 5.5.1). Second, we consider a problem typically faced in an AutoML scenario: If no knowledge is available a priori of the problem at hand, but only the tuning budget, which optimizer should

Table 5.4.1: Optimizers evaluated. For each hyperparameter, we calibrated a 'prior distribution' to give good results across tasks (Section 5.4.2). $\mathcal{U}[a,b]$ is the continuous uniform distribution on $[a,b]$. Log-uniform(a, b) is a distribution whose logarithm is $\mathcal{U}[a,b]$. Log-normal(μ,σ) is a distribution whose logarithm is $\mathcal{N}(\mu,\sigma^2)$

Optimizer	Tunable parameters	Cross-task prior
SGD	Learning rate	Log-normal(-2.09, 1.312)
	Momentum	$\mathcal{U}[0,1]$
	Weight decay	Log-uniform(-5, -1)
	Poly decay (p)	$\mathcal{U}[0.5,5]$
Adagrad	Learning rate	Log-normal(-2.004, 1.20)
Adam	Learning rate	Log-normal(-2.69, 1.42)
	β_1, β_2	1 - Log-uniform(-5, -1)
	ϵ	Log-uniform(-8, 0)

Table 5.5.1: Models and datasets used. We use the DeepOBS benchmark set (Schneider et al., 2019). Details are provided in Appendix 5.A.

Architecture	Datasets
Convolutional net	FMNIST, CIFAR10/100
Variational autoencoder	FMNIST, MNIST
Wide residual network	SVHN
Character RNN	Tolstoi's War and Peace
Quadratic function	Artificial dataset
LSTM	IMDB

we use (Section 5.5.2)?

5.5.1 When to Tune More Hyperparameters

To answer the question, at which budget tuning more hyperparameters is preferable, we compare Adam-LR to Adam, and SGD-MW to SGD-M^{C}W^C (Table 5.5.2). To this end, we show performance for increasing budgets K in Figure 5.5.2. Plots for the other optimizers and budgets are given in Figure 5.B.1.

On all classification tasks, Adam-LR and SGD-M^{C}W^C obtain higher performances on average than Adam and SGD-MW, respectively, till the budget of 16. Moreover, the first quartile is often substantially lower for the optimizers with many hyperparameters. For higher budgets, both outperform their counterparts on CIFAR-100 and FMNIST on average and in the second quartile, and Adam outperforms Adam-LR on IMDB as well. However, even for the largest budgets, Adam's first quartile is far lower than Adam-LR's.

On the regression tasks, tuning more hyperparameters only helps for SGD-MW on MNIST-VAE. In all other cases, tuning additional hyperparameters degrades the performance for small budgets and achieves similar performance at high budgets.

5.5.2 Summarizing across datasets

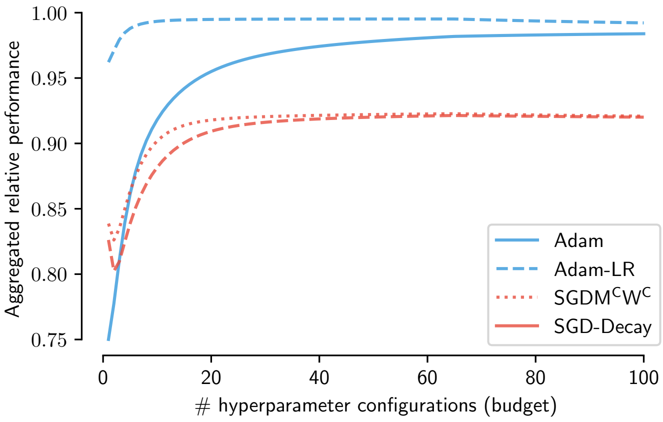


Figure 5.5.1: Aggregated relative performance of various optimizers across datasets

We, now, turn to the question of how our choice of optimizer would change in a setting where nothing is known about the problem, à la AutoML. AutoML setup. To this end, we summarize performances across all datasets. First, for a given budget, we compute the probability that an optimizer outperforms the others on a randomly chosen task. In Figure 5.1.1, we compare

Table 5.5.2: Optimizers and tunable parameters. SGD(γ, μ, λ) is SGD with γ learning rate, μ momentum, λ weight decay coefficient. Adagrad(γ) is Adagrad with γ learning rate, Adam($\gamma, \beta_1, \beta_2, \epsilon$) is Adam with learning rate γ,

Optimizer label	Tunable parameters
SGD-LR	SGD(γ, μ=0, λ=0)
SGD-M	SGD(γ, μ, λ=0)
SGD-M^C	SGD(γ, μ=0.9, λ=0)
SGD-M^{C}W^C	SGD(γ, μ=0.9, λ=10^{-5})
SGD-M^{C}D	SGD(γ, μ=0.9, λ=10^{-5}) + Poly Decay(p)
SGD-MW	SGD(γ, μ, λ)
Adagrad	Adagrad(γ)
Adam-LR	Adam(γ, β_1=0.9, β_2=0.999, ϵ=10^{-8})
Adam	Adam($\gamma, \beta_1, \beta_2, \epsilon$)
Adam-W^{C}D	Adam-LR + Poly Decay(p)

Adam, Adam-LR, SGD-M^{C}W^C, and SGD-M^{C}D, because we found them to yield the overall best results. First, the results reflect the findings from Section 5.5.1 in that tuning more hyperparameters (Adam) becomes a better relative option the more budget is available. However, throughout all tuning budget scenarios, Adam-LR remains by far the most probable to yield the best results.

Figure 5.1.1 shows that Adam-LR is the most likely to get the best results. However, it does not show the margin by which the SGD variants underperform. To address this issue, we compute summary statistics for an optimizer o's performance after k iterations in the following way:

$$S(o,k) = \frac{1}{|\mathscr{P}|} \sum_{p \in \mathscr{P}} \frac{o(k,p)}{\max_{o' \in \mathscr{O}} o'(k,p)},$$

where $o(k,p)$ denotes the expected performance of optimizer $o \in \mathscr{O}$ on test problem $p \in \mathscr{P}$ after k iterations of hyperparameter search. In other words, we compute the average relative performance of an optimizer to the best performance of any optimizer on the respective task, at budget k.

The results are in Figure 5.5.1 which show that Adam-LR performs very close to the best optimizer for all budgets. In the early stages of HPO, the SGD variants perform 20% worse than Adam-LR. This gap narrows to 10% as tuning budgets increase, but the flatness of the curves for high budgets suggests that they are unlikely to improve further with higher budgets. Adam on the other hand steadily improves relative to Adam-LR, and only leaves a 2-3% gap at high budgets.

5.6 Discussion

The key results of our experiments are two-fold. First, they support the hypothesis that adaptive gradient methods are easier to tune than non-adaptive methods: In a setting with a low budget for hyperparameter tuning, tuning only Adam's learning rate is likely to be a very good choice; it doesn't guarantee the best possible performance, but it is evidently the easiest to find well-performing hyperparameter configurations for. While SGD (variants) yields the best performance in some cases, its best configuration is tedious to find, and Adam often performs very close to it. Hence, in terms of Figure 5.2.2, SGD seems to be a hyperparameter surface with narrow minima, akin to optimizer E, whereas the minima of Adam are relatively wide, akin to optimizer F. We investigate the empirical hyperparameter surfaces in Appendix 5.G to confirm our hypothesis. We, thus, state that the substantial value of the adaptive gradient methods, specifically Adam, is its amenability to hyperparameter search. This is in contrast to the findings of Wilson et al. (2017) who observe no advantage in tunabilty for adaptive gradient methods, and thus deem them to be of 'marginal value'. This discrepancy is explained by the fact that our evaluation protocol is almost entirely free of possible human bias: In contrast to them, we do not only avoid manually tuning the hyperparameters through the use of automatic hyperparameter optimization, we also automatically determine the HPO's own

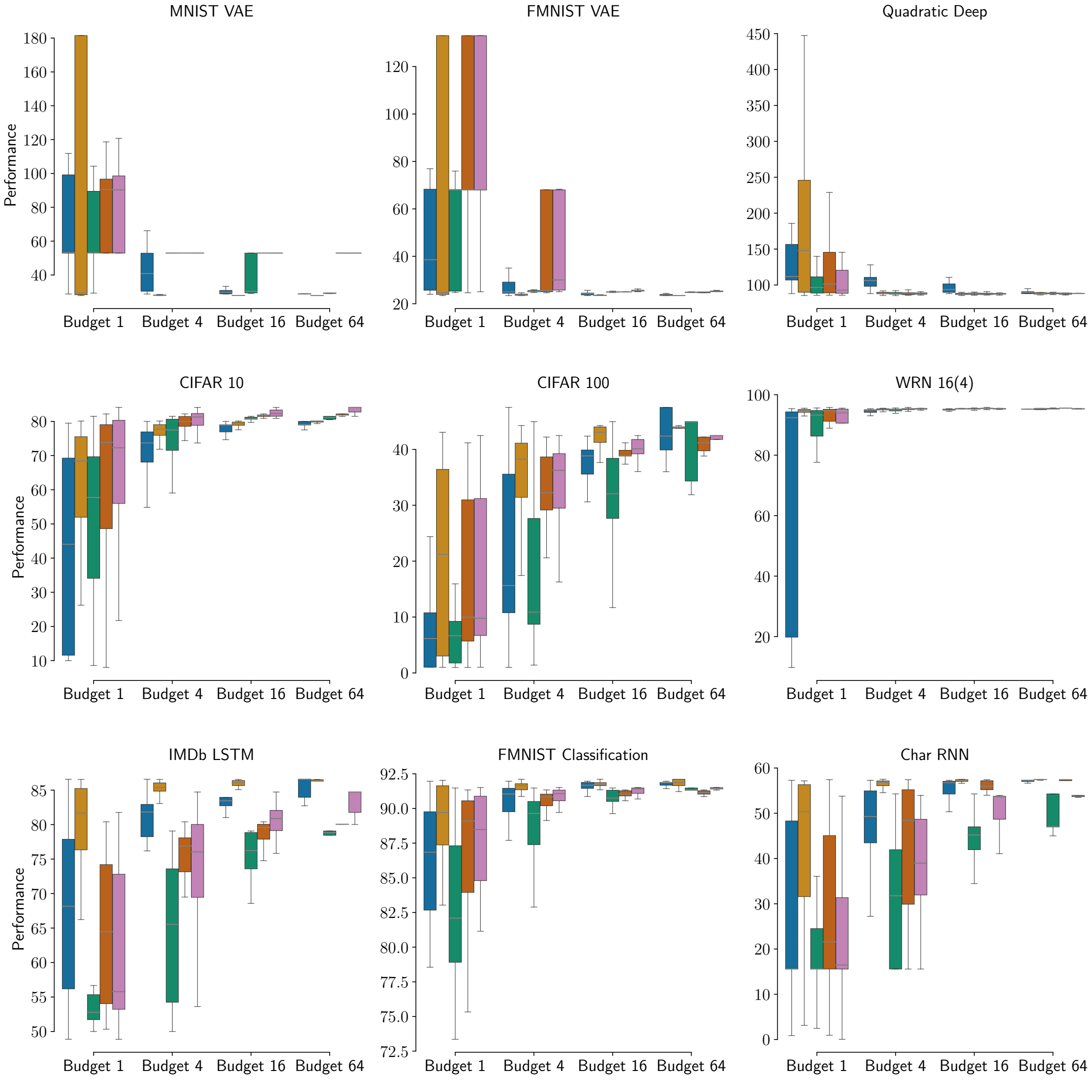


Figure 5.5.2: Performance of **Adam-LR**, **Adam**, **SGD-M^{C}W^C**, **SGD-MW**, **SGD-M^{C}D** at various hyperparameter search budgets. Image is best viewed in color. Some plots have been truncated to increase readability.

hyperparameters by estimating the distributions over search spaces.

Secondly, we find that tuning optimizers' hyperparameters apart from the learning rate becomes more useful as the available tuning budget increases. In particular, we find that Adam approaches the performance of Adam-LR for large budgets. This is, of course, an expected result, and in line with recent work by Choi et al. (2019), who argue that, with sufficient hyperparameter tuning, a more general optimizer (Adam) should never under-perform any particular instantiation thereof (Adam-LR). Choi et al. (2019) claim that this point is already reached in 'realistic' experiments. However, in their experiments, Choi et al. (2019) tune the search spaces for each problem they consider, thereby assuming apriori knowledge of what constitutes meaningful hyperparameter settings for that specific problem. Our results, which are obtained with a protocol that is arguably less driven by human bias, tell a different story: Even with a relatively large tuning budget, tuning only the learning rate of Adam is arguably the safer choice, as it achieves good results with high probability, whereas tuning all hyperparameters can also result in a better performance albeit with high variance. These observations suggest that optimizers with many tunable hyperparameters have a hyperparameter surface that is less smooth, and that is why fixing e.g. the momentum and weight decay parameters to prescribed 'recipe' values is beneficial in low-resource scenarios. Our observations are supported by further experimental evidence in Schmidt et al. (2021), which finds that Adam remains a practical choice across problems; it is not always the best performing optimizer, but is a safe, well-performing choice.

Our study is certainly not exhaustive: We do not study the effect of a different HPO like a Bayesian HPO on the results, due to the prohibitively high computational cost it incurs. By choosing uni-variate distribution families for the hyperparameters to estimate the priors, we do not account for complex relationships between parameters that might exist. We explore this in Appendix 5.H where we use the notion of 'effective learning rate' (Shallue et al., 2019), and we find that it helps improve the performance in the lower budgets of hyperparameter optimization. We attribute this to the fact that SGDElrW is effective at exploiting historically successful (γ, μ) pairs. However, the literature does not provide methods to incorporate these into a probabilistic model that incorporates the causal relationships between them.

In the future, we suggest that optimizer designers not only study the efficacy and convergence properties but also provide priors to sample hyperparameters. Our study demonstrates this as a key component in determining an optimizer's practical value.

5.7 Conclusion

We propose to include the process of hyperparameter optimization in optimizer benchmarking. In addition to showing peak performance, this showcases the optimizer's ease-of-use in practical scenarios. We hope that this work encourages other researchers to conduct future studies on the performance of optimizers from a more holistic perspective, where the cost of the hyperparameter search is included.

Appendix - Chapter 5

Appendix 5.A Architectures of the Models Used in Experiments

Along with the architectures examined by Schneider et al. (2019), we experiment with an additional network and dataset. We included an additional network into our experimental setup, as DeepOBS does not contain a word level LSTM model. Our model uses a 32-dimensional word embedding table and a single layer LSTM with memory cell size 128, the exact architecture is given in Table 5.A.1. We experiment with the IMDB sentiment classification dataset (Maas et al., 2011). The dataset contains 50,000 movie reviews collected from movie rating website IMDB. The training set has 25,000 reviews, each labeled as positive or negative. The rest 25,000 form the test set. We split 20% of the training set to use as the development set. We refer the readers to DeepOBS (Schneider et al., 2019) for the exact details of the other architectures used in this work.

Table 5.A.1: Architecture of the LSTM network used for IMDb experiments

Layer name	Description
Emb	$\begin{bmatrix}\text{Embedding Layer}\\ \text{Vocabulary of 10000}\\ \text{Embedding dimension: 32}\end{bmatrix}$
LSTM_1	$\begin{bmatrix}\text{LSTM}\\ \text{Input size: 32}\\ \text{Hidden dimension: 128}\end{bmatrix}$
FC Layer	Linear$(128 \rightarrow 2)$
Classifier	Softmax(2)

Appendix 5.B Performance Analysis

We show the full performance plots of all variants of SGD, Adam, and Adagrad we experimented with in Figure 5.B.1.

Appendix 5.C How Likely Are We to Find Good Configurations?

In Figure 5.1.1 we showed the chance of finding the optimal hyperparameter setting for some optimizers considered, in a problem-agnostic setting. Here we delve into the case where we present similar plots for each of the problems considered in Section 5.5.

A natural question that arises is: Given a budget K, what is the best optimizer one can pick? In other words, for a given budget what is the probability of each optimizer finding the best configuration? We answer this with a simple procedure. We repeat the runs of HPO for a budget K, and collect the optimizer that gave the best result in each of those runs. Using the classical definition of probability, we compute the required quantity. We plot the computed probability in Figure 5.C.1. It is very evident for nearly all budgets, Adam-LR is always the best option for 4 of the problems. SGD variants emerge to be better options for CIFAR-100 and Char-RNN at later stages of HPO. For some problems like VAEs, LSTM, it is very obvious that Adam-LR is nearly always the best choice. This further strengthens our hypothesis that adaptive gradient methods are more tunable, especially in constrained HPO budget scenarios.

Appendix 5.D Computing the Expected Maximum of Random Samples

The following is a constructive proof of how to compute the expected value that the bootstrap method converges to in the limit of infinite re-sampling. It is a paraphrase of Dodge et al. (2019, Section 3.1), but due to inaccuracies in Equation (5.1) in their paper, we repeat it here for clarity.

Let $x_1, x_2 \dots x_N \sim \mathscr{X}$ be N independently sampled values. Let the random variable $\mathbf{Y}$ be the maximum of a random subset of size S from $x_1, x_2 \dots x_N$ where $S \leq N$. For representational convenience, let them be the first S samples. So, $\mathbf{Y} = \max\{x_1, \dots, x_S\}$. We are interested in computing $\mathbb{E}[\mathbf{Y}]$. This can be computed as

$$\mathbb{E}[\mathbf{Y}] = \sum_y y \cdot P(\mathbf{Y} = y)$$

for discrete $\mathbf{Y}$, with $P(\mathbf{Y} = y)$ be the probability mass function of $\mathbf{Y}$. We can write

$$P(\mathbf{Y} = y) = P(\mathbf{Y} \leq y) - P(\mathbf{Y} < y)$$

As $x_i \ \forall i$ are iid sampled,

$$\begin{aligned} P(\mathbf{Y} \le y) &= P(\max_{i=1\dots S} x_i \le y) \\ &= \prod_{i=1}^{S} P(x_i \le y) \\ &= P(X \le y)^S \end{aligned}$$

$P(X \le y)$ can be empirically estimated from data as the sum of normalized impulses.

$$P(X \le y) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}_{x_i \le y} \tag{5.1}$$

Thus,

$$\mathbb{E}[\mathbf{Y}] = \sum_y y(P(X \le y)^S - P(X < y)^S) \tag{5.2}$$

A very similar equation can be derived to compute the variance too. Variance is defined as $Var(\mathbf{Y}) = \mathbb{E}[Y^2] - \mathbb{E}[Y]^2$. The second operand is given by Equation (5.2). The first operand (for discrete distributions) can be computed as

$$\mathbb{E}[\mathbf{Y}^2] = \sum_y y^2(P(X \le y)^S - P(X < y)^S) \tag{5.3}$$

Given the iterates (not incumbents) of Random Search, the expected performance at a given budget can be estimated by Equation (5.2) and the variance can be computed by Equation (5.3).

Appendix 5.E Aggregating the Performance of Incumbents

In Algorithm 5, we propose returning all the incumbents of the HPO algorithm. Here we propose the use of an aggregation function that helps create a comparable scalar that can be used in a benchmarking software like DeepOBS to rank the performance of optimizers that takes into cognizance the ease-of-use aspect too.

5.E.1 Aggregating Function

For the aggregation function discussed, we propose a simple convex combination of the incumbent performances and term it ω-tunability. If $\mathcal{L}_t$ be the incumbent performance at budget t, we define ω-tunability as

$$\omega\text{-tunability} = \sum_{t=1}^{T} \omega_t \mathscr{L}_t$$

where $w_t > 0 \; \forall$ t and $\sum_t w_t = 1$

By appropriately choosing the weights $\{\omega_t\}$, we can interpolate between our two notions of tunability in Section 5.2. In the extreme case where we are only interested in the peak performance of the optimizer, we can set $\omega_T = 1$ and set the other weights to zero. In the opposite extreme case where we are interested in the "one-shot tunability" of the optimizer, we can set $\omega_1 = 1$. In general, we can answer the question of "How well does the optimizer perform with a budget of K runs?" by setting $\omega_i = \mathbf{1}_{\mathrm{i=K}}$. Figure 5.5.2 and Figure 5.B.1 can also be computed as $\omega_i = \mathbf{1}_{\mathrm{i=K}}$.

While the above weighting scheme is intuitive, merely computing the performance after expending HPO budget of K does not consider the performance obtained after the previous $K-1$ iterations i.e., we would like to differentiate the cases where a requisite performance is attained by tuning an optimizer for K iterations and another for K_1 iterations, where $K_1 \gg K$. Therefore, we employ a weighting scheme as follows: By setting $\omega_i \propto (T-i)$, our first one puts more emphasis on the earlier stages of the hyperparameter tuning process. We term this weighting scheme *Cumulative Performance-Early*(*CPE*). In contrast, the second weighting scheme, *Cumulative Performance-Late* (*CPL*) puts more emphasis on late stages of tuning, and thus on obtaining a better performance at a higher tuning cost: $\omega_i \propto i$. The results of the various optimizers' *CPE* is shown in Table 5.E.1. It is very evident that Adam-LR fares the best across tasks. Even when it is not the best performing one, it is quite competitive. Thus, our observation that Adam-LR is the easiest-to-tune i.e., it doesn't guarantee the best performance, but it gives very competitive performances early in the HPO search phase, holds true.

For a benchmarking software package like DeepOBS, we suggest the use of *CPE* to rank optimizers, as it places focus on ease-of-tuning. This supplements the existing peak performance metric reported previously.

Optimizer	FMNIST(%)↑	CIFAR 10(%)↑	CIFAR 100(%)↑	IMDb(%)↑	WRN 16(4)(%)↑	Char-RNN(%)↑	MNIST-VAE↓	FMNIST-VAE↓	Quadratic Deep↓
Adam-LR	**91.6**	78.8	**42.0**	85.9	**95.3**	56.9	28.9	**24.3**	89.9
Adam	91.3	77.3	38.1	83.4	94.5	54.2	33.1	25.7	95.4
SGD-M^{C}W^C	90.8	81.0	38.8	78.7	**95.3**	53.9	54.0	27.9	**87.4**
SGD-MW	90.5	79.6	33.2	75.2	95.0	44.4	35.2	26.5	87.5
SGD-M^{C}D	91.1	**82.1**	39.2	80.5	95.2	49.6	54.3	29.8	87.5
Adagrad	91.3	76.6	29.8	84.4	95.0	55.6	30.7	25.9	90.6
Adam-W^{C}D	**91.6**	79.4	35.1	**86.0**	95.1	**57.4**	**28.6**	**24.3**	92.8
SGD-LR	90.4	76.9	30.6	68.1	94.7	39.9	53.4	26.2	89.3
SGD-M	90.5	77.8	39.8	73.8	94.9	50.7	37.1	26.3	88.2
SGD-M^C	90.7	78.8	**42.0**	79.0	95.0	55.5	54.1	28.5	88.1

Table 5.E.1: *CPE* for the various optimizers experimented. It is evident that Adam-LR is the most competitive across tasks.

Appendix 5.F Results for Computation Time Budgets

Using number of hyperparameter configuration trials as budget unit does not account for the possibility that optimizers may require different amounts of computation time to finish a trial. While the cost for one update step is approximately the same for all optimizers, some require more update steps than others before reaching convergence.

To verify that our results and conclusions are not affected by our choice of budget unit, we simulate the results we would have obtained with a computation time budget in the following way. For a given test problem (e.g., CIFAR-10), we compute the minimum number of update steps any optimizer has required to finish 100 trials, and consider this number to be the maximum computation budget. We split this budget into 100 intervals of equal size. Using the bootstrapTibshirani and Efron (1993), we then simulate 1,000 HPO runs, and save the best performance achieved at each interval. Note that sometimes an optimizer can complete multiple hyperparameter trials in one interval, and sometimes a single trial may take longer than one interval. Finally, we average the results from all 1,000 HPO runs and compute the same summary across datasets as in Section 5.5.2.

Figure 5.F.1 shows that the conclusions do not change when using computation time as budget unit. In fact, the graphs show almost the exact same pattern as in Figure 5.5.1, where number of hyperparameter trials is the budget unit.

Appendix 5.G Plotting Hyperparameter Surfaces

In Section 5.2, we hypothesize that the performance as a function of the hyperparameter, e.g., learning rate, of an optimizer that performs well with few trials has a wider extremum compared to an optimizer that only performs well with more trials.

In Figure 5.G.1, we show a scatter plot of the loss/accuracy surfaces of SGD-M^{C}W^C and Adam-LR as a function of the learning rate, which is their only tunable hyperparameter. The plots confirm the expected behavior. On MNIST VAE, FMNIST VAE, and Tolstoi-Char-RNN, Adam-LR reaches performances close to the optimum on a wider range of learning rates than SGD-M^{C}W^C does, resulting in substantially better expected performances at small budgets ($k = 1, 4$) as opposed to SGD-M^{C}W^C, even though their extrema are relatively close to each other. On CIFAR10, the width of the maximum is similar, leading to comparable performances at low budgets. However, the maximum for SGD-M^{C}W^C is slightly higher, leading to better performance than Adam-LR at high budgets.

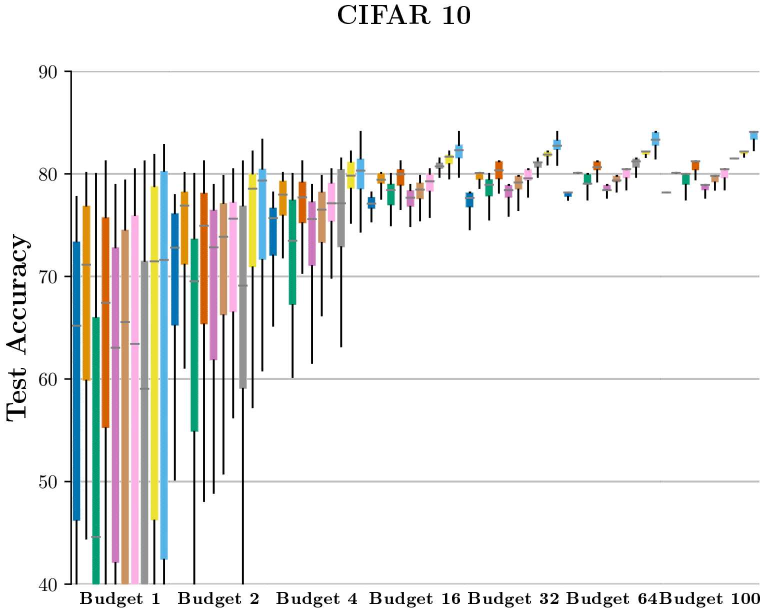
Appendix 5.H Interplay between momentum and learning rate

We ran an additional experiment using ‘effective learning rate’ (Shallue et al., 2019) that combines learning rate γ, and momentum μ of SGD to compute the effective learning rate γ^{eff}.

Intuitively, γ^{eff}quantifies the contribution of a given minibatch to the overall training. This is defined as

$$\gamma^{\text{eff}} = \frac{\gamma}{1-\mu}$$

We designed a variant of SGD-MW, called SGD-LR$_{\text{eff}}$, where we sampled γ and γ^{eff}independently from lognormal priors calibrated as usual, and compute the momentum(μ) as $\mu = \max(0, (1 - \frac{\gamma}{\gamma^{\text{eff}}}))$, hence accounting for the interplay between learning rate and momentum. We plot the performance comparisons between SGD-MW and SGD-LR$_{\text{eff}}$ in Figure 5.H.1, and provide a plot of the aggregated relative performance in Figure 5.H.2. The results show that indeed SGD-LR$_{\text{eff}}$ improves over SGD-MW in the low-budget regime, particularly on classification tasks. We attribute this to the fact that SGD-LR$_{\text{eff}}$ is effective at exploiting historically successful (γ, μ) pairs. For large budgets, however, SGD-LR$_{\text{eff}}$ performs increasingly worse than SGD-MW, which can be explained by the fact that SGD-MW has a higher chance of exploring new configurations due to the independence assumption. Despite the improvement in low-budget regimes, SGD variants, including the new SGD-LR$_{\text{eff}}$ variant, remain substantially below Adam-LR in all budget scenarios. Hence, our conclusion remains the same.



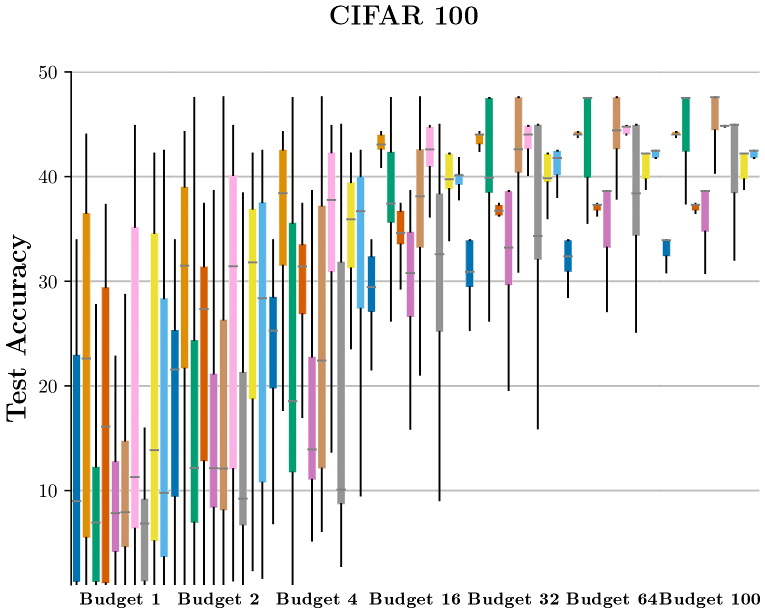


Figure 5.B.1: **Adagrad**, **Adam-LR**, **Adam**, **Adam-W^{C}D**, **SGD-LR**, **SGD-M**, **SGD-M^C**, **SGD-MW**, **SGD-M^{C}W^C**, and **SGD-M^{C}D**

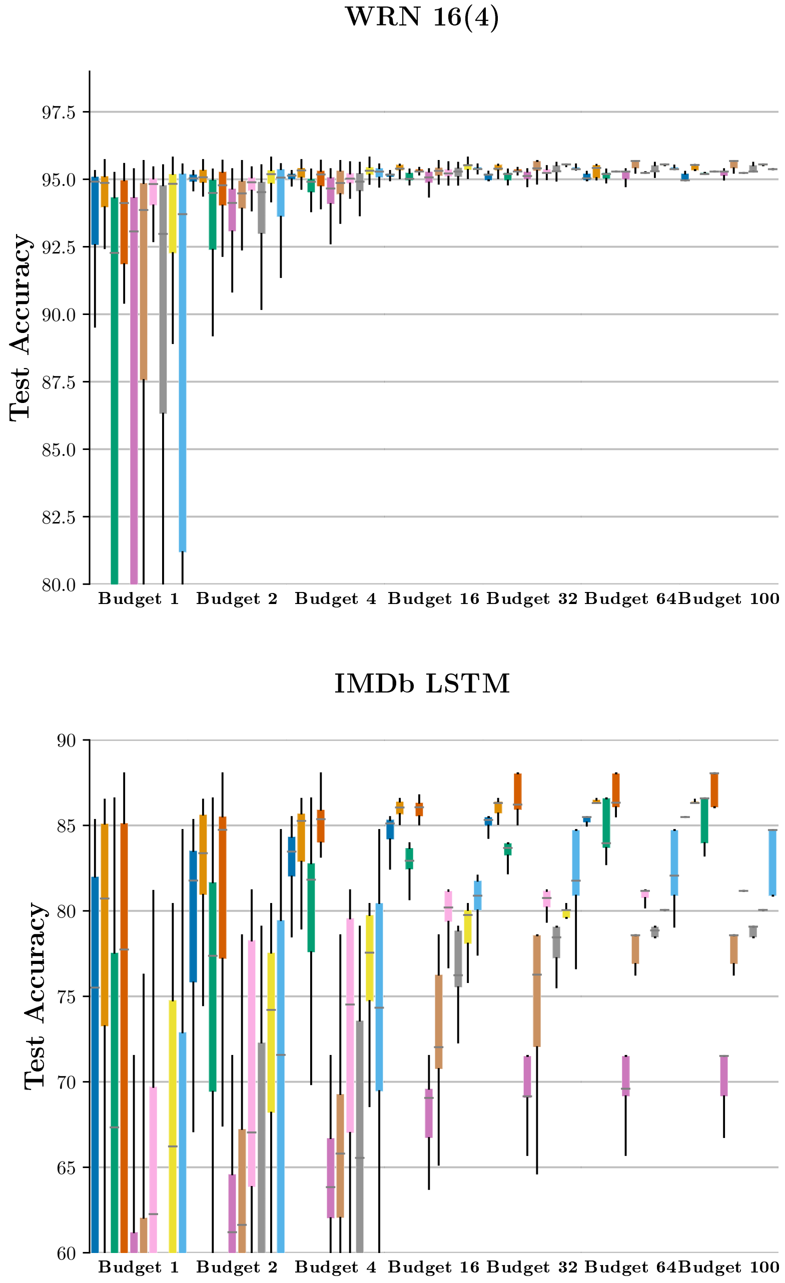
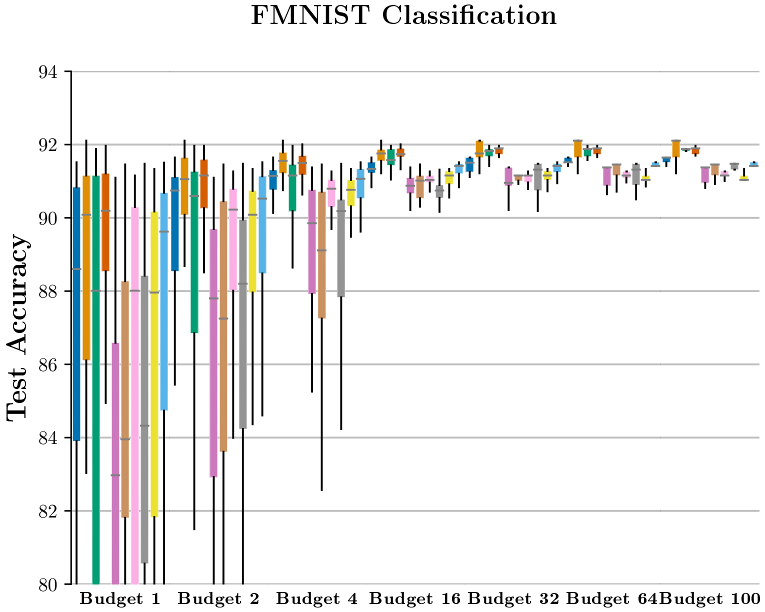


Figure 5.B.1: **Adagrad**, **Adam-LR**, **Adam**, **Adam-W^{C}D**, **SGD-LR**, **SGD-M**, **SGD-M^C**, **SGD-MW**, **SGD-M^{C}W^C**, and **SGD-M^{C}D**



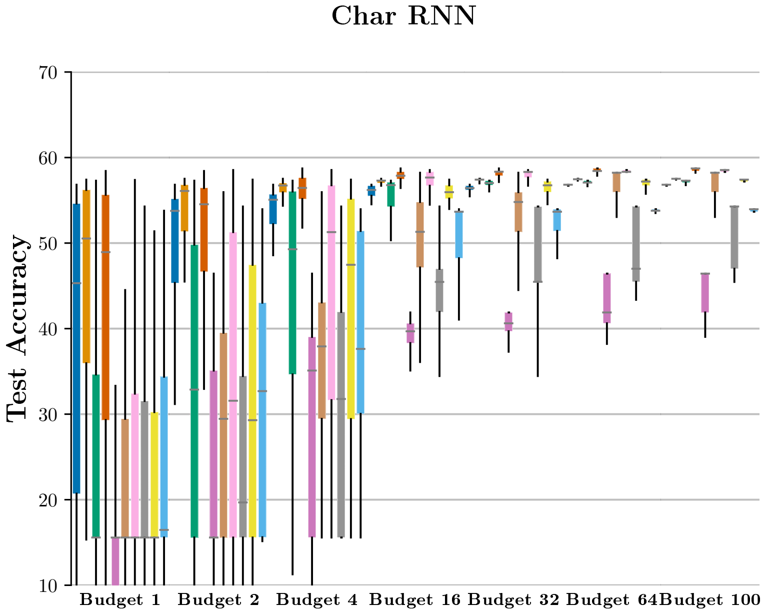
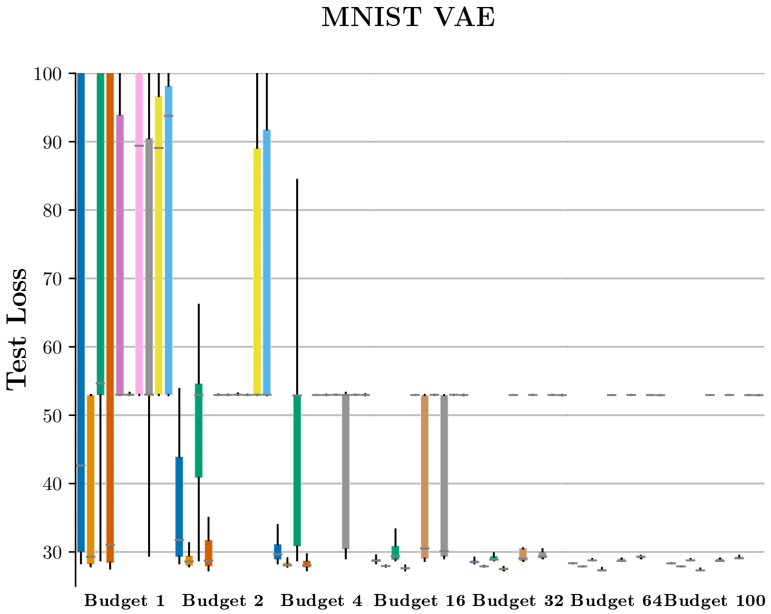


Figure 5.B.1: **Adagrad**, **Adam-LR**, **Adam**, **Adam-W^{C}D**, **SGD-LR**, **SGD-M**, **SGD-M^C**, **SGD-MW**, **SGD-M^{C}W^C**, and **SGD-M^{C}D**



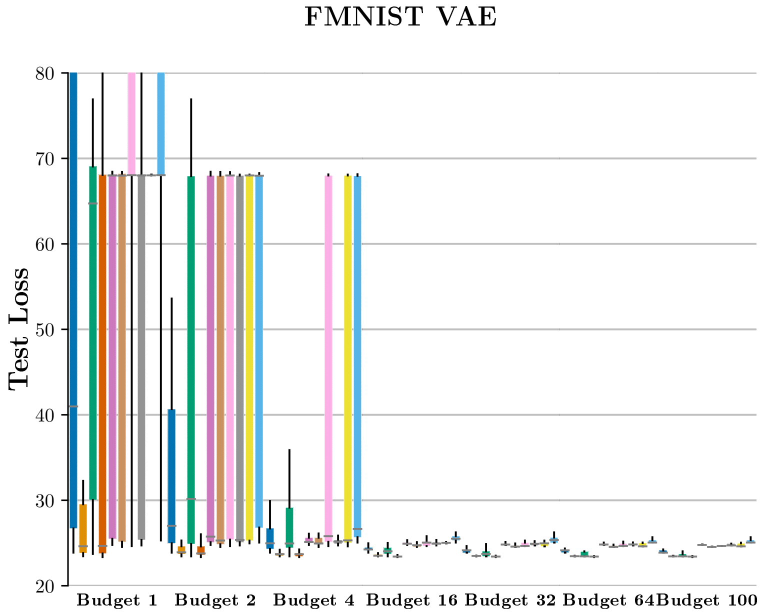


Figure 5.B.1: **Adagrad**, **Adam-LR**, **Adam**, **Adam-W^{C}D**, **SGD-LR**, **SGD-M**, **SGD-M^C**, **SGD-MW**, **SGD-M^{C}W^C**, and **SGD-M^{C}D**

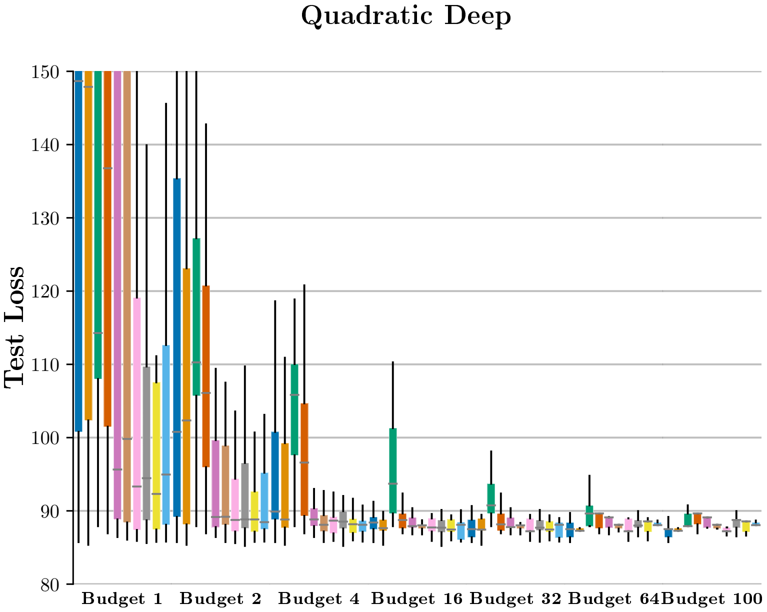


Figure 5.B.1: We show the performance of **Adagrad**, **Adam-LR**, **Adam**, **Adam-W^{C}D**, **SGD-LR**, **SGD-M**, **SGD-M^C**, **SGD-MW**, **SGD-M^{C}W^C**, and **SGD-M^{C}D** over all the experiments. We plot the on the x-axis the number of the hyperparameter configuration searches, on the y-axis the appropriate performance.

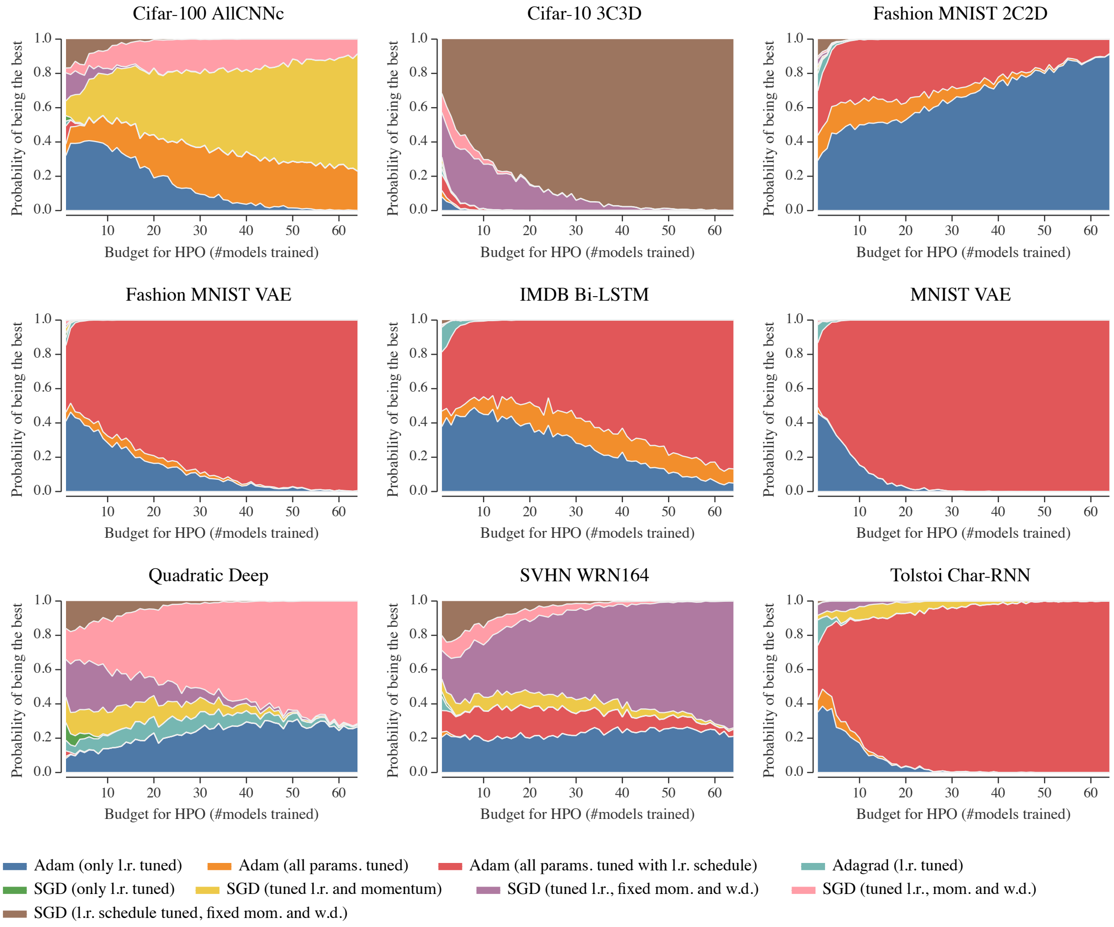


Figure 5.C.1: Which optimizer for which budget? Given a tuning budget K (x-axis), the stacked area plots above show how likely each optimizer (colored bands) is to yield the best result after K steps of hyperparameter optimization. For example, for the IMDB LSTM problem, for a small budget, Adam-LR is the best choice (with ~ 0.8 probability), whereas for a larger search budget > 50, tuning the other parameters of 'Adam' is likely to pay off.

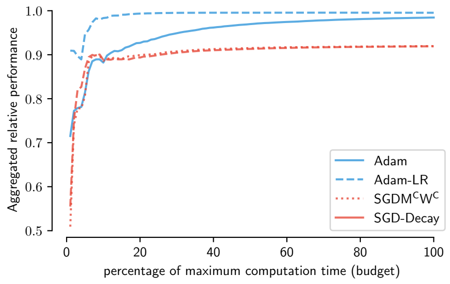


Figure 5.F.1: Aggregated relative performance of each optimizer across datasets.

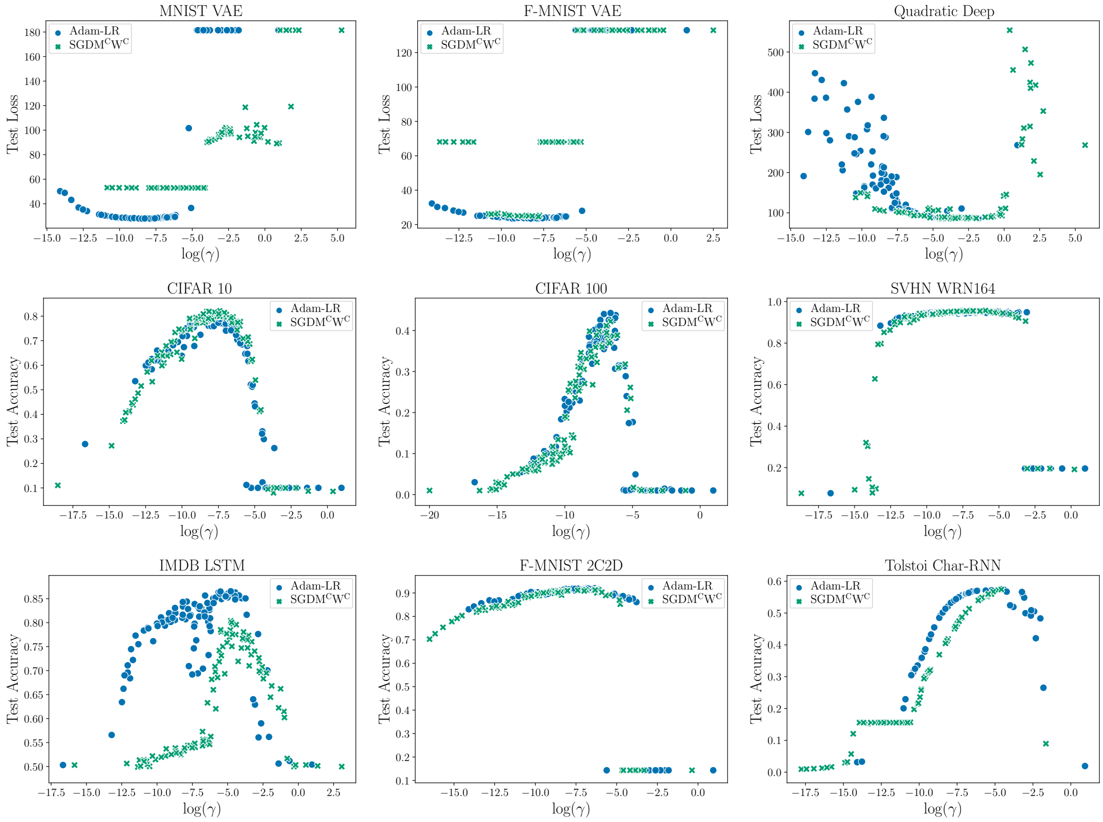


Figure 5.G.1: Scatter plot of performance of Adam-LR and SGD-M^{C}W^C by learning rate value. For a better visibility, we shift the learning rate values of SGD-M^{C}W^C in such a way that the minima of both optimizers are at the same position on the x-axis.

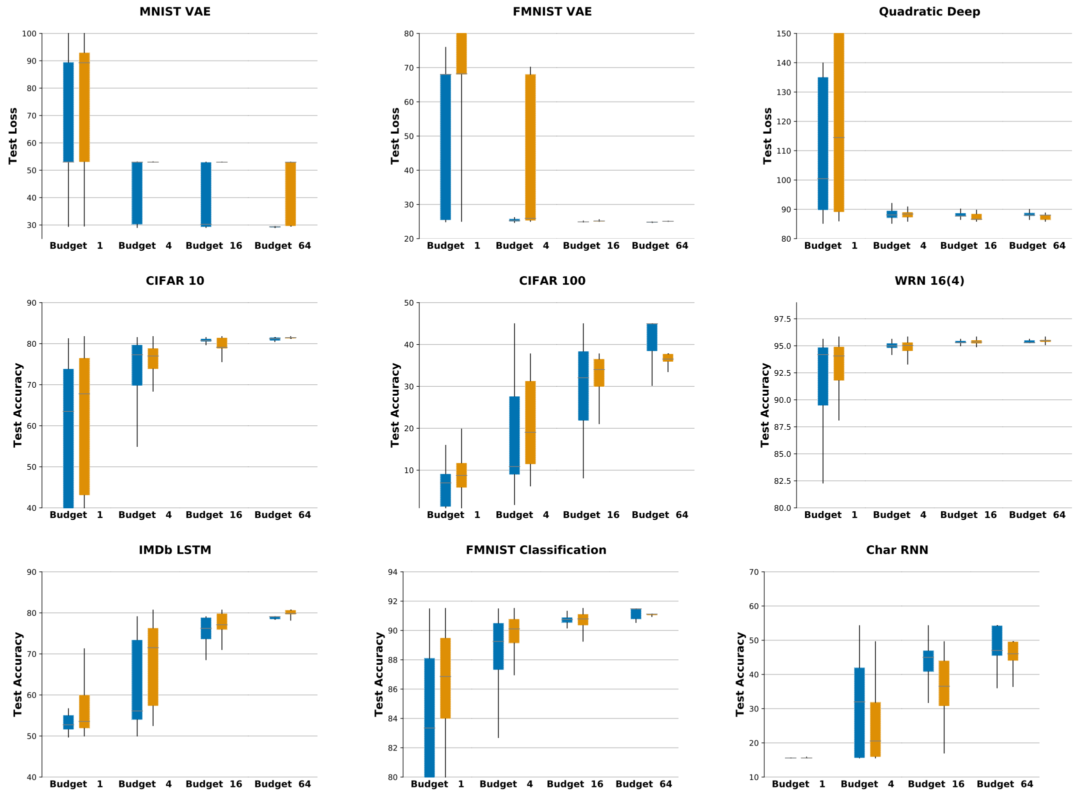


Figure 5.H.1: Performance of **SGD-MW**, **SGD-LR$_{\text{eff}}$**, at various hyperparameter search budgets. Image is best viewed in color. Some plots have been truncated to increase readability.

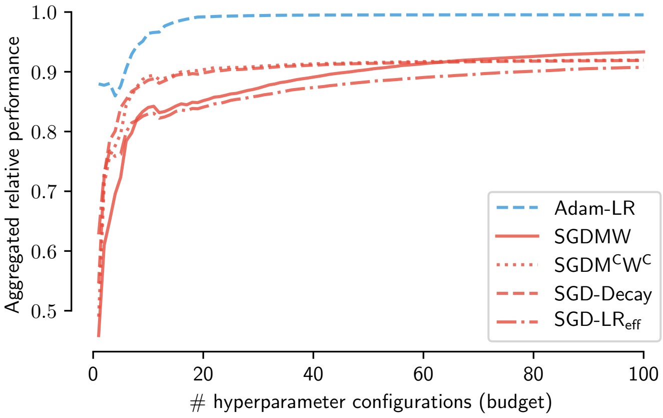


Figure 5.H.2: Aggregated relative performance of SGD-LR$_{\text{eff}}$ compared to other optimizers.

Concluding Remarks Part III

6 Concluding remarks

6.1 Summary

In this thesis, we investigated various problems under the umbrella of efficient deep learning, focusing on semantic segmentation. Despite the decade-long progress in deep learning for computer vision, resource appetite plagues the field. In this work, we broke this appetite down into two major components: (a) labeled data availability from the test domain, which hinders the applicability of already available and trained models due to domain shift, and (b) compute resource availability which hinders ready deployment onto low-power devices, and also a resource-aware benchmarking of progress in deep learning.

We proposed the problem of model adaptation in semantic segmentation, a problem also termed source-free adaptation in literature. Unsupervised domain adaptation techniques address an important problem of transferring knowledge from one domain to another without needing labels but require the source dataset for this process. This can be impractical as semantic segmentation datasets are generally too large to lug around. We proposed a modification in which, instead of the source dataset, a source-trained model is used for adaptation along with the unlabeled target domain dataset. We use the confidence of predictions and noise-robustness of features as important properties that correlate with the ability to make correct predictions.

We extended those observations to an allied problem: test-time adaptation. Most real-world problems do not come with strict prior knowledge of the domain changes in the test data. Therefore, we shift the adaptation to inference time; the network is adapted to tackle possible domain shifts in a given sample or batch of test samples. The previous observations of ‘good properties’ like noise-robustness were reused. In lieu of feature robustness, we ensured that the network’s predictions were invariant to input transformations. This simple loss function was shown to improve performance in the presence of domain changes and image corruptions.

In the second part of the thesis, we turned our focus to the computational resources needed for semantic segmentation. While previous works focussed on designing architectures for run-time efficiency, we took an alternative route where we modulated the amount of input data to be processed by a transformer. This is based on the observation that different parts of an image have varying degrees of difficulty of segmentation and that computation can be heterogeneously distributed according to the difficulty of segmenting an input patch. We experimentally showed that this distribution of computation could be achieved by using posterior entropy to pause patches i.e., we do not update the representations of patches for which the network is confident enough to segment. Our method has an additional advantage in that the number of paused patches is a tunable hyperparameter, and we proposed a training strategy that enables tuning this hyperparameter at inference without retraining.

Our final contribution was a benchmarking strategy that is aware of the computation cost for optimizers used in modern deep learning. Previous methods for benchmarking primarily examined the best achievable performance by a method with no consideration for the effort expended for tuning the hyperparameters of that method. We argued that a method that achieves good performance with little tuning is practically more relevant than one that reaches a higher performance albeit with a lot of tuning. Consequently, we found that adaptive gradient optimizers like Adam are easier to tune than SGD. Tuning only the learning rate of Adam is likely to be a very good choice; it doesn't guarantee the highest performance, but it is the easiest to find well-performing hyperparameter configurations for. We applied these arguments to optimizers for deep learning and have been applied to model evaluation elsewhere (Dodge et al., 2019).

6.2 Future Work

We discuss the unanswered questions specific to the techniques presented in the thesis, and the extensions and alternative avenues.

6.2.1 On adaptation in Part I

Effect of architecture on its adaptability:
For the methods for adaptation in Chapters 2 and 3, the architectures used were treated as black boxes. For example, we used the DeepLabV2 model with ResNet backbone for adaptation experiments and did not consider the specific architectural components of that model in the adaptation. The relationship between adaptability and architectural components is not fully understood. Studies show that normalization layers are important for domain adaptation (Nado et al., 2021; Rusak et al., 2022), but they have only been demonstrated empirically. We found, anecdotally, that a network without normalization layers is not amenable to adaptation, but the theoretical backing for these remains to be provided. Similarly, transformer-based models (Xie et al., 2021; Bai et al., 2021) were shown to have substantially better corruption robustness (Kamann and Rother, 2020) compared to convolutional networks. A study apportioning the value of various kinds of layers and architectures to adaptability and domain robustness will be of great value to the field.

Effect of training tricks on adaptability:
Several papers have studied the source model attributes and training recipes that influence the transfer performance for fine-tuning (Neyshabur et al., 2020; Kornblith et al., 2019; Pinto et al., 2021). In Chapter 3, we found that some models are more adaptable than others. The only discernable difference is the set of augmentations used in training on the source data. This presents an important unexplored part of the work: do some training recipes create more adaptation suitable networks than others? This goes hand-in-hand with the point above about architectural dependence.

Leveraging large pretrained models:
The goal in Part I was to adapt a trained network using an unlabeled target domain dataset. Recent work in large-scale training methods for language (Brown et al., 2020), vision (Abnar et al., 2022; He et al., 2022) has been shown to generalize very well to downstream tasks with little to no data. Some preliminary studies have shown the utility of fine-tuning these large models (Kumar et al., 2022). CLIP-like models have also explored joint representation learning of text and images (Radford et al., 2021) and have been successful at segmentation with a text prompt (Lüddecke and Ecker, 2022; Wang et al., 2022b). Given the powerful feature extraction capabilities learned by these encoders, investigating their transfer performance would be valuable.

6.2.2 On the efficiency of transformers in Chapter 4

Combining patch pausing with complementary methods:
Recent studies on improving the efficiency of transformers have focussed on patch merging (Bolya et al., 2023, ToMe) where redundant computation is reduced by merging patches that are deemed to be similar. This line of thinking can be applied to PAUMER too, in which for the patches we determine to need more processing, merging the redundant computations can result in a boost in the throughput of segmentation transformers.

Image-level patch pausing:
Our method uses the design choice of a fixed proportion of patches due to its GPU cache friendliness. On other devices like CPUs that are not constrained by the batch level parallelism like the GPU (Kurtz et al., 2020), higher throughput may be achieved by an image-dependent pause proportion, the criterion for which needs additional investigation.

Drop-in replacements for attention components:
On the efficiency of transformers front, implementation improvements like the FlashAttention (Dao et al., 2022) have reduced the computational complexity of self-attention and are faster and use lower memory than their traditional implementations in PyTorch. A trivial drop-in replacement with FlashAttention in Chapter 4 can yield additional throughput yields to the proposed PAUMER.

6.2.3 On fair evaluation methods

Effect of hyperparameter search method:
Our work in Chapter 5 proposes that the performance of a network or optimizers depends on the hyperparameter search method itself, an aspect that is unexplored in our work. We used Random Search for our experiments based on the computational benefits it offers. Using Bayesian, multi-fidelity search methods (Klein et al., 2020) or human-like search (Xiong et al., 2020) methods may shed additional light on the effect of the hyperparameter search method on the conclusions reached.

Dependence of hyperparameters:
We made a simplifying assumption that hyperparameters are independent of each other, an assumption that is known to be suboptimal (Loshchilov and Hutter, 2019). Without these assumptions, more sample-efficient calibration methods are required to compute the prior distributions. We leave the study of hyperparameter dependence to future work.

6.3 Final remarks

As deep models become increasingly integrated into our lives, it is imperative that we understand and mitigate their failure modes on data with characteristics that were not encountered during training, like domain shifts studied in this thesis. In particular, tackling these shifts as they are encountered is ever so important.

The widely famous GPT-3 model (Brown et al., 2020) consumes the energy equivalent to powering a 60W bulb for about four minutes to generate one page of text[1]. It is not an exaggeration that improving the efficiency of these models is the need of the hour, a problem we tackled for vision models. The final training cost of the final deployed model was set to be about 4-5 million dollars, trained for nearly 20 years on a commercial GPU. Its performance is astonishing, but at what cost? A true reporting of this would require incorporating the cost of hyperparameter optimization (and failed experiments) into the overall costs to paint a holistic picture of the effort required to reach the high performance it boasts of.

The successes of deep learning in the very recent past, whether in text processing, image analysis, or joint understanding, are taking these modern networks beyond the confines of a laboratory to the world at large. At this seemingly crucial juncture, the ponderings of this thesis are vital for these advances to cement their utility in the real world.

[1] This is a back-of-the-envelope calculation based on the power usage given in https://www.freeingenergy.com/what-is-a-kilowatt-hour/.

Bibliography

Abnar, S., Dehghani, M., Neyshabur, B., and Sedghi, H. (2022). Exploring the limits of large scale pre-training. In *International Conference on Learning Representations*.

Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., and Süsstrunk, S. (2012). Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11):2274–2282.

Arazo, E., Ortego, D., Albert, P., O'Connor, N. E., and McGuinness, K. (2020). Pseudo-labeling and confirmation bias in deep semi-supervised learning. In *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE.

Asi, H. and Duchi, J. C. (2019a). The importance of better models in stochastic optimization. *Proceedings of the National Academy of Sciences*.

Asi, H. and Duchi, J. C. (2019b). Stochastic (approximate) proximal point methods: Convergence, optimality, and adaptivity. *SIAM Journal on Optimization*, 29(3):2257–2290.

Bachman, P., Alsharif, O., and Precup, D. (2014). Learning with pseudo-ensembles. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 3365–3373, Cambridge, MA, USA. MIT Press.

Bahmani, S., Hahn, O., Zamfir, E., Araslanov, N., Cremers, D., and Roth, S. (2022). Semantic self-adaptation: Enhancing generalization with a single sample. *ArXiv*, abs/2208.05788.

Bai, Y., Mei, J., Yuille, A., and Xie, C. (2021). Are transformers more robust than cnns? In *Thirty-Fifth Conference on Neural Information Processing Systems*.

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305.

Berthelot, D., Carlini, N., Goodfellow, I., Papernot, N., Oliver, A., and Raffel, C. A. (2019). Mixmatch: A holistic approach to semi-supervised learning. *Advances in Neural Information Processing Systems*, 32.

Bilen, H. and Vedaldi, A. (2016). Integrated perception with recurrent multi-task neural networks. In *Advances in neural information processing systems*, pages 235–243.

Bobu, A., Tzeng, E., Hoffman, J., and Darrell, T. (2018). Adapting to continuously shifting domains.

Bolya, D., Fu, C.-Y., Dai, X., Zhang, P., Feichtenhofer, C., and Hoffman, J. (2023). Token merging: Your ViT but faster. In *International Conference on Learning Representations*.

Bousquet, O. and Elisseeff, A. (2002). Stability and generalization. *The Journal of Machine Learning Research*, 2:499–526.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Chang, W., Wang, H., Peng, W., and Chiu, W. (2019). All about structure: Adapting structural information across domains for boosting semantic segmentation. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1900–1909.

Chapelle, O., Schlkopf, B., and Zien, A. (2010). *Semi-Supervised Learning*. The MIT Press, 1st edition.

Chapelle, O. and Zien, A. (2005). Semi-supervised classification by low density separation. In *AISTATS*, volume 2005, pages 57–64. Citeseer.

Chaudhari, P., Choromańska, A., Soatto, S., LeCun, Y., Baldassi, C., Borgs, C., Chayes, J., Sagun, L., and Zecchina, R. (2017). Entropy-sgd: Biasing gradient descent into wide valleys. *ArXiv*, abs/1611.01838.

Chechik, G., Heitz, G., Elidan, G., Abbeel, P., and Koller, D. (2008). Max-margin classification of data with absent features. *Journal of Machine Learning Research*, 9(Jan):1–21.

Chen, J. and Gu, Q. (2018). Closing the generalization gap of adaptive gradient methods in training deep neural networks. *CoRR*, abs/1806.06763.

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. (2018). Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Trans. on Pattern Analysis and Machine Intelligence (TPAMI)*, 40:834–848.

Chen, M., Weinberger, K., Sha, F., and Bengio, Y. (2014). Marginalized denoising auto-encoders for nonlinear representations. In *International Conference on Machine Learning*, pages 1476–1484.

Chen, M., Xue, H., and Cai, D. (2019a). Domain adaptation for semantic segmentation with maximum squares loss. In *Proc. of International Conference on Computer Vision (ICCV)*.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020). A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709*.

Chen, Y.-C., Lin, Y.-Y., Yang, M.-H., and Huang, J.-B. (2019b). Crdoco: Pixel-level domain transfer with cross-domain consistency. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Chen, Y.-H., Chen, W.-Y., Chen, Y.-T., Tsai, B.-C., Frank Wang, Y.-C., and Sun, M. (2017). No more discrimination: Cross city adaptation of road scene segmenters. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1992–2001.

Chidlovskii, B., Clinchant, S., and Csurka, G. (2016). Domain adaptation in the absence of source domain data. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 451–460.

Choi, D., Shallue, C. J., Nado, Z., Lee, J., Maddison, C. J., and Dahl, G. E. (2019). On empirical comparisons of optimizers for deep learning. *arXiv preprint arXiv:1910.05446*.

Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258.

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 Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*.

Ciresan, D. C., Meier, U., and Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. In *IEEE Conference on Computer Vision and Pattern Recognition CVPR 2012*. Long preprint arXiv:1202.2745v1 [cs.CV].

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 *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3213–3223.

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

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

Croce, F., Andriushchenko, M., Sehwag, V., Debenedetti, E., Flammarion, N., Chiang, M., Mittal, P., and Hein, M. (2020). Robustbench: a standardized adversarial robustness benchmark. *arXiv preprint arXiv:2010.09670*.

Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., and Le, Q. V. (2019a). Autoaugment: Learning augmentation strategies from data. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 113–123, Los Alamitos, CA, USA. IEEE Computer Society.

Cubuk, E. D., Zoph, B., Shlens, J., and Le, Q. V. (2019b). Randaugment: Practical data augmentation with no separate search. *CoRR*, abs/1909.13719.

Dai, D., Sakaridis, C., Hecker, S., and Van Gool, L. (2019). Curriculum model adaptation with synthetic and real data for semantic foggy scene understanding. *International Journal of Computer Vision (IJCV)*, pages 1–23.

Dai, D. and Van Gool, L. (2018). Dark model adaptation: Semantic image segmentation from daytime to nighttime. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 3819–3824. IEEE.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. (2022). FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*.

Deng, J., Dong, W., Socher, R., Li, L., Li, K., and Li, F. (2009). Imagenet: A large-scale hierarchical image database. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255.

Deng, W., Gould, S., and Zheng, L. (2022). On the strong correlation between model invariance and generalization. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K., editors, *Advances in Neural Information Processing Systems*.

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. (2022). GPT3.int8(): 8-bit matrix multiplication for transformers at scale. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K., editors, *Advances in Neural Information Processing Systems*.

DeVries, T. and Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*.

Dodge, J., Gururangan, S., Card, D., Schwartz, R., and Smith, N. A. (2019). Show your work: Improved reporting of experimental results. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2185–2194.

Doersch, C., Gupta, A., and Efros, A. A. (2015). Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE international conference on computer vision*, pages 1422–1430.

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*.

Du, L., Tan, J., Yang, H., Feng, J., Xue, X., Zheng, Q., Ye, X., and Zhang, X. (2019). SSF-DAN: separated semantic feature based domain adaptation network for semantic segmentation. In *Proc. of International Conference on Computer Vision (ICCV)*.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

Eggensperger, K., Lindauer, M., and Hutter, F. (2019). Pitfalls and best practices in algorithm configuration. *Journal of Artificial Intelligence Research*, 64:861–893.

Elsayed, G., Krishnan, D., Mobahi, H., Regan, K., and Bengio, S. (2018). Large margin deep networks for classification. In *Advances in neural information processing systems*, pages 842–852.

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.

Fang, Y., Yap, P.-T., Lin, W., Zhu, H., and Liu, M. (2022). Source-free unsupervised domain adaptation: A survey. *arXiv preprint arXiv:2301.00265*.

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.

French, G., Aila, T., Laine, S., Mackiewicz, M., and Finlayson, G. (2019). Semi-supervised semantic segmentation needs strong, high-dimensional perturbations. *arXiv preprint arXiv:1906.01916*.

Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059.

Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., Marchand, M., and Lempitsky, V. (2016). Domain-adversarial training of neural networks. *Journal of Machine Learning Research (JMLR)*, 17(1):2096–2030.

Gidaris, S., Singh, P., and Komodakis, N. (2018). Unsupervised representation learning by predicting image rotations. In *International Conference on Learning Representations*.

Globerson, A. and Roweis, S. (2006). Nightmare at test time: robust learning by feature deletion. In *Proceedings of the 23rd international conference on Machine learning*, pages 353–360.

Grandvalet, Y. and Bengio, Y. (2005). Semi-supervised learning by entropy minimization. In *Actes de CAP 05, Conférence francophone sur l'apprentissage automatique*, pages 281–296.

Graves, A. (2016). Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*.

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.

Hao, S., Zhou, Y., and Guo, Y. (2020). A brief survey on semantic segmentation with deep learning. *Neurocomputing*, 406:302–321.

He, K., Chen, X., Xie, S., Li, Y., Dollár, P., and Girshick, R. (2022). Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16000–16009.

He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. (2020). Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9729–9738.

He, K., Zhang, X., Ren, S., and Sun, J. (2016a). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

He, K., Zhang, X., Ren, S., and Sun, J. (2016b). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2018). Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Hendrycks, D. and Dietterich, T. (2019). Benchmarking neural network robustness to common corruptions and perturbations. *Proceedings of the International Conference on Learning Representations*.

Hendrycks, D., Mu, N., Cubuk, E. D., Zoph, B., Gilmer, J., and Lakshminarayanan, B. (2020). AugMix: A simple data processing method to improve robustness and uncertainty. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Hoffman, J., Tzeng, E., Park, T., Zhu, J.-Y., Isola, P., Saenko, K., Efros, A., and Darrell, T. (2018). Cycada: Cycle-consistent adversarial domain adaptation. In *Proc. of the International Conference on Machine Learning (ICML)*.

Hoffman, J., Wang, D., Yu, F., and Darrell, T. (2016). FCNs in the wild: Pixel-level adversarial and constraint-based adaptation. *arXiv preprint arXiv:1612.02649*.

Hooker, S. (2021). The hardware lottery. *Commun. ACM*, 64(12):58–65.

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., and Adam, H. (2019). Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

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*, abs/1704.04861.

Huang, S.-W., Lin, C.-T., Chen, S.-P., Wu, Y.-Y., Hsu, P.-H., and Lai, S.-H. (2018). Auggan: Cross domain adaptation with gan-based data augmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Huang, W. R., Emam, Z., Goldblum, M., Fowl, L., Terry, J. K., Huang, F., and Goldstein, T. (2020). Understanding generalization through visualizations. In Zosa Forde, J., Ruiz, F., Pradier, M. F., and Schein, A., editors, *Proceedings on "I Can't Believe It's Not Better!" at NeurIPS Workshops*, volume 137 of *Proceedings of Machine Learning Research*, pages 87–97. PMLR.

Hutter, F., Kotthoff, L., and Vanschoren, J., editors (2019). *Hyperparameter Optimization*, pages 3–33. Springer International Publishing, Cham.

Jang, E., Gu, S., and Poole, B. (2016). Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Jin, J., Dundar, A., and Culurciello, E. (2014). Flattened convolutional neural networks for feedforward acceleration. *arXiv preprint arXiv:1412.5474*.

Jo, J. and Bengio, Y. (2017). Measuring the tendency of cnns to learn surface statistical regularities. *arXiv preprint arXiv:1711.11561*.

Kamann, C. and Rother, C. (2020). Benchmarking the robustness of semantic segmentation models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8828–8838.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. (2020). Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

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.

Kendall, A. and Gal, Y. (2017). What uncertainties do we need in bayesian deep learning for computer vision? In *Advances in neural information processing systems*, pages 5574–5584.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. (2016). On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*.

Khan, S., Naseer, M., Hayat, M., Zamir, S. W., Khan, F. S., and Shah, M. (2021). Transformers in vision: A survey. *ACM Comput. Surv.*

Kim, M. and Byun, H. (2020). Learning texture invariant representation for domain adaptation of semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12975–12984.

Kim, Y., Hong, S., Cho, D., Park, H., and Panda, P. (2020). Domain adaptation without source data. *arXiv preprint arXiv:2007.01524*.

Kingma, D. and Ba, J. (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

Klein, A., Tiao, L. C., Lienart, T., Archambeau, C., and Seeger, M. (2020). Model-based asynchronous hyperparameter and neural architecture search. *arXiv preprint arXiv:2003.10865*.

Kornblith, S., Shlens, J., and Le, Q. V. (2019). Do better imagenet models transfer better? In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2661–2671.

Krause, A., Perona, P., and Gomes, R. G. (2010). Discriminative clustering by regularized information maximization. In *Advances in neural information processing systems*, pages 775–783.

Kristiadi, A., Hein, M., and Hennig, P. (2020). Being bayesian, even just a bit, fixes overconfidence in relu networks. *arXiv preprint arXiv:2002.10118*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Neural Information Processing Systems (NeurIPS)*, pages 1106–1114.

Kumar, A., Raghunathan, A., Jones, R. M., Ma, T., and Liang, P. (2022). Fine-tuning can distort pretrained features and underperform out-of-distribution. In *International Conference on Learning Representations*.

Kundu, J. N., Venkat, N., Babu, R. V., et al. (2020). Universal source-free domain adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4544–4553.

Kurtz, M., Kopinsky, J., Gelashvili, R., Matveev, A., Carr, J., Goin, M., Leiserson, W., Moore, S., Nell, B., Shavit, N., and Alistarh, D. (2020). Inducing and exploiting activation sparsity for fast inference on deep neural networks. In III, H. D. and Singh, A., editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5533–5543, Virtual. PMLR.

Kuzmin, A., Nagel, M., Pitre, S., Pendyam, S., Blankevoort, T., and Welling, M. (2019). Taxonomy and evaluation of structured compression of convolutional neural networks. *arXiv preprint arXiv:1912.09802*.

Laine, S. and Aila, T. (2017). Temporal ensembling for semi-supervised learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Larsson, M., Stenborg, E., Toft, C., Hammarstrand, L., Sattler, T., and Kahl, F. (2019). Fine-grained segmentation networks: Self-supervised segmentation for improved long-term visual localization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 31–41.

Lavin, A. and Gray, S. (2016). Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Lee, D.-H. (2013). Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on challenges in representation learning, ICML*, volume 3.

Leen, T. K. (1994). From data distributions to regularization in invariant learning. In *NIPS*, pages 223–230.

Li, R., Jiao, Q., Cao, W., Wong, H.-S., and Wu, S. (2020). Model adaptation: Unsupervised domain adaptation without source data. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9638–9647.

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, Y., Yuan, L., and Vasconcelos, N. (2019). Bidirectional learning for domain adaptation of semantic segmentation. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Liang, J., Hu, D., and Feng, J. (2020). Do we really need to access the source data? source hypothesis transfer for unsupervised domain adaptation. In *International Conference on Machine Learning (ICML)*, pages xx–xx.

Liang, Y., GE, C., Tong, Z., Song, Y., Wang, J., and Xie, P. (2022). EVit: Expediting vision transformers via token reorganizations. In *International Conference on Learning Representations*.

Liu, W., Rabinovich, A., and Berg, A. C. (2015). Parsenet: Looking wider to see better. *CoRR*, abs/1506.04579.

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 *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440.

Loquercio, A., Segu, M., and Scaramuzza, D. (2020). A general framework for uncertainty estimation in deep learning. *IEEE Robotics and Automation Letters*, 5(2):3153–3160.

Loshchilov, I. and Hutter, F. (2019). Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Lucic, M., Kurach, K., Michalski, M., Gelly, S., and Bousquet, O. (2018). Are gans created equal? a large-scale study. In *Advances in neural information processing systems*, pages 700–709.

Lüddecke, T. and Ecker, A. (2022). Image segmentation using text and image prompts. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7086–7096.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. (2011). Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.

Maaten, L., Chen, M., Tyree, S., and Weinberger, K. (2013). Learning with marginalized corrupted features. In *International Conference on Machine Learning*, pages 410–418.

Madhyastha, P. and Jain, R. (2019). On model stability as a function of random seed. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 929–939, Hong Kong, China. Association for Computational Linguistics.

Malik, J., Belongie, S., Leung, T., and Shi, J. (2001). Contour and texture analysis for image segmentation. *International journal of computer vision*, 43(1):7–27.

Mantovani, R. G., Horváth, T., Cerri, R., Junior, S. B., Vanschoren, J., de Carvalho, A. C. P. d., and Ferreira, L. (2018). An empirical study on hyperparameter tuning of decision trees. *arXiv preprint arXiv:1812.02207*.

Marin, D., Chang, J.-H. R., Ranjan, A., Prabhu, A., Rastegari, M., and Tuzel, O. (2022). Token pooling in vision transformers.

Melis, G., Dyer, C., and Blunsom, P. (2018). On the state of the art of evaluation in neural language models. In *International Conference on Learning Representations*.

Meng, L., Li, H., Chen, B.-C., Lan, S., Wu, Z., Jiang, Y.-G., and Lim, S.-N. (2022). Adavit: Adaptive vision transformers for efficient image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12309–12318.

Metz, L., Maheswaranathan, N., Sun, R., Freeman, C. D., Poole, B., and Sohl-Dickstein, J. (2020). Using a thousand optimization tasks to learn hyperparameter search strategies. *arXiv preprint arXiv:2002.11887*.

Meyerson, E. and Miikkulainen, R. (2018). Pseudo-task augmentation: From deep multitask learning to intratask sharing—and back. In *International Conference on Machine Learning*, pages 3511–3520.

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., and Wu, H. (2018). Mixed precision training. In *International Conference on Learning Representations*.

Mintun, E., Kirillov, A., and Xie, S. (2021). On interaction between augmentations and corruptions in natural corruption robustness. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*.

Mishra, S., Saenko, K., and Saligrama, V. (2021). Surprisingly simple semi-supervised domain adaptation with pretraining and consistency. *CoRR*, abs/2101.12727.

Miyato, T., Maeda, S.-i., Koyama, M., and Ishii, S. (2018). Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):1979–1993.

MMSegmentation Contributors (2020). MMSegmentation: Openmmlab semantic segmentation toolbox and benchmark. https://github.com/open-mmlab/mmsegmentation.

Murez, Z., Kolouri, S., Kriegman, D. J., Ramamoorthi, R., and Kim, K. (2018). Image to image translation for domain adaptation. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Musgrave, K., Belongie, S., and Lim, S.-N. (2020). A metric learning reality check. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXV 16*, pages 681–699. Springer.

Nado, Z., Padhy, S., Sculley, D., D'Amour, A., Lakshminarayanan, B., and Snoek, J. (2021). Evaluating prediction-time batch normalization for robustness under covariate shift.

Nair, T., Precup, D., Arnold, D. L., and Arbel, T. (2020). Exploring uncertainty measures in deep networks for multiple sclerosis lesion detection and segmentation. *Medical image analysis*, 59:101557.

Neyshabur, B., Sedghi, H., and Zhang, C. (2020). What is being transferred in transfer learning? *Advances in neural information processing systems*, 33:512–523.

Ng, N., Cho, K., Hulkund, N., and Ghassemi, M. (2022). Predicting out-of-domain generalization with local manifold smoothness. *arXiv preprint arXiv:2207.02093*.

Noroozi, M. and Favaro, P. (2016). Unsupervised learning of visual representations by solving jigsaw puzzles. In *European Conference on Computer Vision*, pages 69–84. Springer.

Novak, R., Bahri, Y., Abolafia, D. A., Pennington, J., and Sohl-Dickstein, J. (2018). Sensitivity and generalization in neural networks: an empirical study. In *International Conference on Learning Representations*.

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. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12599–12608.

Ouali, Y., Hudelot, C., and Tami, M. (2020). Semi-supervised semantic segmentation with cross-consistency training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12674–12684.

Ovadia, Y., Fertig, E., Ren, J., Nado, Z., Sculley, D., Nowozin, S., Dillon, J., Lakshminarayanan, B., and Snoek, J. (2019). Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In *Advances in Neural Information Processing Systems*, pages 13991–14002.

Pan, B., Panda, R., Jiang, Y., Wang, Z., Feris, R., and Oliva, A. (2021). Ia-red^2: Interpretability-aware redundancy reduction for vision transformers. volume 34, pages 24898–24911.

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*.

Paul, S., Khurana, A., and Aggarwal, G. (2022). What can we do with just the model? a simple knowledge extraction framework. In *International Conference on Machine Learning. Principles of Distribution Shift Workshop*. https://icml.cc/virtual/2022/workshop/13465#wse-detail-19760.

Peng, X., Usman, B., Kaushik, N., Wang, D., Hoffman, J., and Saenko, K. (2018). Visda: A synthetic-to-real benchmark for visual domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 2021–2026.

Pinto, F., Torr, P., and Dokania, P. K. (2021). Are vision transformers always more robust than convolutional neural networks? In *NeurIPS 2021 Workshop on Distribution Shifts: Connecting Methods and Applications*.

Probst, P., Boulesteix, A.-L., and Bischl, B. (2019). Tunability: Importance of hyperparameters of machine learning algorithms. *Journal of Machine Learning Research*, 20(53):1–32.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al. (2021). Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR.

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*.

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)*.

Rasmus, A., Valpola, H., Honkala, M., Berglund, M., and Raiko, T. (2015). Semi-supervised learning with ladder networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, page 3546–3554, Cambridge, MA, USA. MIT Press.

Ratner, A. J., Ehrenberg, H. R., Hussain, Z., Dunnmon, J., and Ré, C. (2017). Learning to compose domain-specific transformations for data augmentation. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 3239–3249, Red Hook, NY, USA. Curran Associates Inc.

Recht, B., Roelofs, R., Schmidt, L., and Shankar, V. (2019). Do ImageNet classifiers generalize to ImageNet? volume 97 of *Proceedings of Machine Learning Research*, pages 5389–5400, Long Beach, California, USA. PMLR.

Ribeiro, M. T., Wu, T., Guestrin, C., and Singh, S. (2020). Beyond accuracy: Behavioral testing of nlp models with checklist. In *Association for Computational Linguistics (ACL)*.

Richter, S. R., Vineet, V., Roth, S., and Koltun, V. (2016). Playing for data: Ground truth from computer games. In Leibe, B., Matas, J., Sebe, N., and Welling, M., editors, *Proc. of European Conference on Computer Vision (ECCV)*, volume 9906, pages 102–118. Springer International Publishing.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407.

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.

Ros, G., Sellart, L., Materzynska, J., Vazquez, D., and Lopez, A. M. (2016). The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3234–3243.

Ruder, S., Peters, M. E., Swayamdipta, S., and Wolf, T. (2019). Transfer learning in natural language processing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials*, pages 15–18, Minneapolis, Minnesota. Association for Computational Linguistics.

Rusak, E., Schneider, S., Pachitariu, G., Eck, L., Gehler, P. V., Bringmann, O., Brendel, W., and Bethge, M. (2022). If your data distribution shifts, use self-learning. *Transactions of Machine Learning Research*.

Sajjadi, M., Javanmardi, M., and Tasdizen, T. (2016). Regularization with stochastic transformations and perturbations for deep semi-supervised learning. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 1171–1179, Red Hook, NY, USA. Curran Associates Inc.

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 (CVPR)*.

Sankaranarayanan, S., Balaji, Y., Jain, A., Nam Lim, S., and Chellappa, R. (2018). Learning from synthetic data: Addressing domain shift for semantic segmentation. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3752–3761.

Saporta, A., Vu, T.-H., Cord, M., and Pérez, P. (2021). Multi-target adversarial frameworks for domain adaptation in semantic segmentation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9072–9081.

Schapire, R. E., Freund, Y., Bartlett, P., Lee, W. S., et al. (1998). Boosting the margin: A new explanation for the effectiveness of voting methods. *The annals of statistics*, 26(5):1651–1686.

Schmidt, R. M., Schneider, F., and Hennig, P. (2021). Descending through a crowded valley - benchmarking deep learning optimizers. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 9367–9376. PMLR.

Schneider, F., Balles, L., and Hennig, P. (2019). DeepOBS: A deep learning optimizer benchmark suite. In *International Conference on Learning Representations*.

Schneider, S., Rusak, E., Eck, L., Bringmann, O., Brendel, W., and Bethge, M. (2020). Removing covariate shift improves robustness against common corruptions. *CoRR*, abs/2006.16971.

Schwartz, R., Dodge, J., Smith, N. A., and Etzioni, O. (2019). Green AI. *CoRR*, abs/1907.10597.

Sculley, D., Snoek, J., Wiltschko, A. B., and Rahimi, A. (2018). Winner's curse? on pace, progress, and empirical rigor. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*.

Sevilla, J., Heim, L., Ho, A., Besiroglu, T., Hobbhahn, M., and Villalobos, P. (2022). Compute trends across three eras of machine learning. *arXiv preprint arXiv:2202.05924*.

Shah, V., Kyrillidis, A., and Sanghavi, S. (2018). Minimum norm solutions do not always generalize well for over-parameterized problems. *arXiv preprint arXiv:1811.07055*.

Shallue, C. J., Lee, J., Antognini, J., Sohl-Dickstein, J., Frostig, R., and Dahl, G. E. (2019). Measuring the effects of data parallelism on neural network training. *Journal of Machine Learning Research*, 20(112):1–49.

Sharif Razavian, A., Azizpour, H., Sullivan, J., and Carlsson, S. (2014). Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 806–813.

Shorten, C. and Khoshgoftaar, T. (2019). A survey on image data augmentation for deep learning. *Journal of Big Data*, 6:1–48.

Silver, D. L., Yang, Q., and Li, L. (2013). Lifelong machine learning systems: Beyond learning algorithms. In *2013 AAAI spring symposium series*.

Smith, S. L., Kindermans, P.-J., and Le, Q. V. (2018). Don’t decay the learning rate, increase the batch size. In *International Conference on Learning Representations*.

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*.

Strubell, E., Ganesh, A., and McCallum, A. (2019). Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy. Association for Computational Linguistics.

Strudel, R., Garcia, R., Laptev, I., and Schmid, C. (2021). Segmenter: Transformer for semantic segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7262–7272.

Sun, Y., Tzeng, E., Darrell, T., and Efros, A. A. (2019). Unsupervised domain adaptation through self-supervision. *arXiv preprint arXiv:1909.11825*.

Sun, Y., Wang, X., Liu, Z., Miller, J., Efros, A., and Hardt, M. (2020). Test-time training with self-supervision for generalization under distribution shifts. In III, H. D. and Singh, A., editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 9229–9248. PMLR.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In Dasgupta, S. and McAllester, D., editors, *Proceedings of the 30th International Conference on Machine Learning*, number 3 in Proceedings of Machine Learning Research, pages 1139–1147, Atlanta, Georgia, USA. PMLR.

Tan, M. and Le, Q. V. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. *ArXiv*, abs/1905.11946.

Tarvainen, A. and Valpola, H. (2017). Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS’17, page 1195–1204, Red Hook, NY, USA. Curran Associates Inc.

Tay, Y., Dehghani, M., Bahri, D., and Metzler, D. (2022). Efficient transformers: A survey. *ACM Comput. Surv.*, 55(6).

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.

Teja, P. and Fleuret, F. (2021a). Test time adaptation through perturbation robustness. In *NeurIPS 2021 Workshop on Distribution Shifts: Connecting Methods and Applications*.

Teja, P. and Fleuret, F. (2021b). Uncertainty reduction for model adaptation in semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9613–9623.

Teja, P., Mai, F., Vogels, T., Jaggi, M., and Fleuret, F. (2020). Optimizer benchmarking needs to account for hyperparameter tuning. In III, H. D. and Singh, A., editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 9036–9045. PMLR.

Tibshirani, R. J. and Efron, B. (1993). An introduction to the bootstrap. *Monographs on statistics and applied probability*, 57:1–436.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning.

Toldo, M., Maracani, A., Michieli, U., and Zanuttigh, P. (2020). Unsupervised domain adaptation in semantic segmentation: a review. *arXiv preprint arXiv:2005.10876*.

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.

Tsai, Y.-H., Hung, W.-C., Schulter, S., Sohn, K., Yang, M.-H., and Chandraker, M. (2018). Learning to adapt structured output space for semantic segmentation. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7472–7481.

Tsai, Y.-H., Sohn, K., Schulter, S., and Chandraker, M. (2019). Domain adaptation for structured output via discriminative patch representations. In *Proc. of International Conference on Computer Vision (ICCV)*, pages 1456–1465.

Vapnik, V. N. (1998). *Statistical learning theory*. Adaptive and Cognitive Dynamic Systems: Signal Processing, Learning, Commun ications and Control. John Wiley & Sons, Nashville, TN.

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.

Vergara, A., Vembu, S., Ayhan, T., Ryan, M. A., Homer, M. L., and Huerta, R. (2012). Chemical gas sensor drift compensation using classifier ensembles. *Sensors and Actuators B: Chemical*, 166-167:320–329.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.-A., and Bottou, L. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12).

Volpi, R., Morerio, P., Savarese, S., and Murino, V. (2018). Adversarial feature augmentation for unsupervised domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5495–5504.

Vu, T.-H., Jain, H., Bucher, M., Cord, M., and Pérez, P. (2019). Advent: Adversarial entropy minimization for domain adaptation in semantic segmentation. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2517–2526.

Wang, D., Shelhamer, E., Liu, S., Olshausen, B., and Darrell, T. (2021a). Tent: Fully test-time adaptation by entropy minimization. In *International Conference on Learning Representations.*

Wang, H., Shen, T., Zhang, W., Duan, L., and Mei, T. (2020). Classes matter: A fine-grained adversarial approach to cross-domain semantic segmentation. *arXiv preprint arXiv:2007.09222.*

Wang, Q., Fink, O., Van Gool, L., and Dai, D. (2022a). Continual test-time domain adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7201–7211.

Wang, W., Xie, E., Li, X., Fan, D.-P., Song, K., Liang, D., Lu, T., Luo, P., and Shao, L. (2021b). 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.

Wang, Z., Lu, Y., Li, Q., Tao, X., Guo, Y., Gong, M., and Liu, T. (2022b). Cris: Clip-driven referring image segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition.*

Weng, L. (2021). Contrastive representation learning. *lilianweng.github.io/lil-log.*

Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., and Recht, B. (2017). The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, pages 4148–4158.

Wu, H., Judd, P., Zhang, X., Isaev, M., and Micikevicius, P. (2020). Integer quantization for deep learning inference: Principles and empirical evaluation. *arXiv preprint arXiv:2004.09602.*

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. *Advances in Neural Information Processing Systems*, 34.

Xie, Q., Hovy, E., Luong, M.-T., and Le, Q. V. (2019). Self-training with noisy student improves imagenet classification. *arXiv preprint arXiv:1911.04252.*

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.

Xiong, Y., Liu, X., Lan, L.-C., You, Y., Si, S., and Hsieh, C.-J. (2020). How much progress have we made in neural network training? a new evaluation protocol for benchmarking optimizers. *arXiv preprint arXiv:2010.09889*.

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.

Yang, Y. and Soatto, S. (2020). FDA: fourier domain adaptation for semantic segmentation. *arXiv preprint arXiv:2004.05498*.

Yeo, T., Kar, O. F., and Zamir, A. (2021). Robustness via cross-domain ensembles. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12189–12199.

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, F., Chen, H., Wang, X., Xian, W., Chen, Y., Liu, F., Madhavan, V., and Darrell, T. (2020). Bdd100k: A diverse driving dataset for heterogeneous multitask learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2636–2645.

Yu, F., Koltun, V., and Funkhouser, T. A. (2017). Dilated residual networks. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 636–644.

Yu, J., Yang, L., Xu, N., Yang, J., and Huang, T. (2019). Slimmable neural networks. In *International Conference on Learning Representations*.

Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. In Richard C. Wilson, E. R. H. and Smith, W. A. P., editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 87.1–87.12. BMVA Press.

Zbontar, J., Jing, L., Misra, I., LeCun, Y., and Deny, S. (2021). Barlow twins: Self-supervised learning via redundancy reduction. *arXiv preprint arXiv:2103.03230*.

Zhai, X., Kolesnikov, A., Houlsby, N., and Beyer, L. (2022). Scaling vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12104–12113.

Zhang, D., Maslej, N., Brynjolfsson, E., Etchemendy, J., Lyons, T., Manyika, J., Ngo, H., Niebles, J. C., Sellitto, M., Sakhaee, E., Shoham, Y., Clark, J., and Perrault, R. (2022a). The AI Index 2022 annual report.

Zhang, M. M., Levine, S., and Finn, C. (2022b). MEMO: Test time robustness via adaptation and augmentation. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K., editors, *Advances in Neural Information Processing Systems*.

Zhang, Y., David, P., Foroosh, H., and Gong, B. (2019). A curriculum domain adaptation approach to the semantic segmentation of urban scenes. *IEEE Trans. on Pattern Analysis and Machine Intelligence (TPAMI)*.

Zhang, Y., Qiu, Z., Yao, T., Liu, D., and Mei, T. (2018). Fully convolutional adaptation networks for semantic segmentation. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6810–6818.

Zhang, Z., Luo, P., Loy, C. C., and Tang, X. (2014). Facial landmark detection by deep multi-task learning. In *European conference on computer vision*, pages 94–108. Springer.

Zhao, H., Qi, X., Shen, X., Shi, J., and Jia, J. (2018). Icnet for real-time semantic segmentation on high-resolution images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 405–420.

Zhao, H., Shi, J., Qi, X., Wang, X., and Jia, J. (2017). Pyramid scene parsing network. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2881–2890.

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 *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 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, S., An, B., and Huang, F. (2021). Understanding the generalization benefit of model invariance from a data perspective. *Advances in Neural Information Processing Systems*, 34:4328–4341.

Zou, Y., Yu, Z., Liu, X., Kumar, B. V., and Wang, J. (2019). Confidence regularized self-training. In *Proc. of International Conference on Computer Vision (ICCV)*, pages 5982–5991.

Zou, Y., Yu, Z., Vijaya Kumar, B., and Wang, J. (2018). Unsupervised domain adaptation for semantic segmentation via class-balanced self-training. In *Proc. of European Conference on Computer Vision (ECCV)*, pages 289–305.

Prabhu Teja Sivaprasad

Education

2018-2023 **Doctor of Philosophy (PhD) in (Electrical Engineering)**, *Idiap Research Institute, École polytechnique fédérale de Lausanne (EPFL)*, Switzerland, Advisor: Dr. François Fleuret.

2015 **Master of Science (Electronics and Communication Engineering)**, *International Institute of Information Technology (IIIT-H)*, Hyderabad, India, Advisor: Dr. Anoop M Namboodiri.

2010 **Bachelor of Technology (Electronics and Communication Engineering)**, *Vellore Institute of Technology (VIT)*, Vellore, India, Advisor: R Prakash.

Experience

09/2021-02/2022 **Applied Sciencist Intern**, *AWS*, **Amazon Development Center Germany GmbH**.
Project on large scale optimization.

11/2018- Now **Research Assistant**, *Machine Learning group*, **Idiap Research Institute, Switzerland**.

- Method for adapting networks to domain shifts at inference time using augmentation robustness. Presented at NeurIPS 2021 Workshop on Distribution Shifts.
- Method for unsupervised domain adaptation for semantic segmentation. Specifically the case of source data-less domain adaptation using uncertainty quantification. Published CVPR 2021.
- Critical study of the practices of benchmarking of optimizers. Defined the notion of tunability. Large scale experimentation revealed that Adam optimizer is the most tunable of the considered list. Published at ICML 2020.
- Teaching Assistant (TA) for the course EE-559 on Deep Learning ($\sim$ 400 students) taught by Dr François Fleuret at EPFL for the spring semesters of 2020, 2021, 2022. My tasks are to hold tutorial sessions after each lecture, and to design and evaluate course projects.

4/2017-10/2018 **Research Scientist**, *Self Serviced Performance Advertising*, **Amazon Development Center India**.

- Built NLP models for auto-moderation of advertisements on Amazon site using word embeddings, sentence embeddings, cross-lingual transfer
- Productionised models for scoring millions of ads with low latency constraints.

07/2014-02/2017 **Research Engineer**, *Imaging and Computer Vision group*, **Siemens Healthineers Pvt Ltd, Bangalore**.

- Segmentation of human vertebra in Computed Tomography images: Active Shape models, Machine Learning (Random Forest) based boundary detection and Laplacian Mesh deformation
- Non-linear optimization for parameter estimation of Magnetic Resonance Imaging using Levenberg-Marquardt and Nelder-Mead (Simplex) method.
- Deep neural networks for organ detection and segmentation in Computed Tomography images.

02/2014-06/2014 **Intern**, *Imaging and Computer Vision group*, **Siemens Corporate Research and Technology, Bangalore**.

- Texture analysis for faulty steel plate detection from camera feeds.
- Large scale random forests for handling large number of medical volumes.

Publications

- *Courdier E*, Prabhu Teja*, Fleuret F* **Paumer: Patch Pausing Transformer for Semantic Segmentation** *33rd British Machine Vision Conference, 2022.*
- *Prabhu Teja, Fleuret, F* **Test time Adaptation through Perturbation Robustness** *NeurIPS 2021 Workshop on Distribution Shifts.* [PDF]
- *Prabhu Teja, Fleuret, F* **Uncertainty Reduction for Model Adaptation in Semantic Segmentation** *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2021.* Webpage at `https://git.io/JthPp`.
- *Prabhu Teja*, Mai, F.*, Vogels, T., Jaggi, M. and Fleuret, F* **Optimizer Benchmarking Needs to Account for Hyperparameter Tuning** *In Proceedings of the 37th International Conference on Machine Learning (ICML), 2020.* Webpage at `https://git.io/J0qV9`.
- *Prabhu Teja, Namboodiri, A* **A Ballistic Stroke Representation of Online Handwriting for Recognition.** *International Conference on Document Analysis & Recognition–2013* [PDF].

Select Projects

Machine Learning models for Ad-moderation on Amazon platform.

Designed sentence embeddings based models for advertisements to check for highly objectionable content as defined by policy management. Lead the effort on the integration between the machine learning platform and the software pipelines. Worked on transferring the knowledge base built for English to low-resource marketplaces through transfer learning.

Multi-organ detection and segmentation using deep Convolutional networks.

Designed a CNN that outputs proposal heat-maps of each organ (kidney, liver, spleen). Preliminary results show that volume proposals thus found have a high true positive rate and a low false positive rate.

Segmentation of Vertebra from CT Images.

Deviced a method that uses shape priors, and used Laplacian Mesh Deformation w ith constraints derived from the image to fit the mesh to the vertebra. Developed a prototype in MevisLab for ready deployment.

Large Scale Random Forests.

Commonly available implementations of Random Forests require all features available in memory. Built a new random forest model implementation that has on the fly feature computation, efficient data structures and storage for the weak-learners and is parallelisable. Testing it on large amounts of data (∼50GB) showed its effectiveness in real-world scenarios

Programming skills

Languages	Python, C++
Libraries	PyTorch ecosystem, Scientific Python ecosystem, LaTeX, Eigen

Professional Activities

Reviewer for ICML 2023, NeurIPS 2022, ICLR 2022, IEEE Transactions on Multimedia.