Real-time Domain Adaptation in Semantic Segmentation: an in-depth examination of discriminators and optimizations

Benfenati Luca
Politecnico di Torino
s286582@studenti.polito.it

Scarano Nicola
Politecnico di Torino
s287908@studenti.polito.it

Vacca Alessio
Politecnico di Torino
s288240@studenti.polito.it

***Abstract*—One of the main challenges in semantic segmentation is retrieving a large and high-quality labeled dataset to train and test your model. As a matter of fact, real world datasets have to be manually annotated, thus undermining the entire development of convolutional neural networks for this task. In this context, it is necessary to exploit synthetic datasets to train our model, which are automatically annotated. In this paper, we address the domain shift problem proposing a domain adaptation technique based on unsupervised adversarial learning, paired with a real-time network architecture. Our solution tries to both narrow down the gap between the two domains and speed up the inference model in a self-driving car scenario. Different architectures for the discriminator are explored and a thorough analysis and comparison among different compression and optimization techniques (such as pruning, dropout, weight sharing) has been carried out to find the best model.**

I. Introduction

Semantic segmentation's goal is to assign a category label to each pixel of an image; the development journey of this specific task has evolved through:

- image classification, which identifies *what* is present in the image;
- object recognition and detection, which identifies *what* is present in the image and *where*, using a bounding box;
- semantic segmentation, which identifies *what* is present in the image and *where*.

Therefore, in order to address this detailed task, we need to provide to our model the ground-truth class for each pixel for each image used for training. Whilst it is well known that when trained on a very large amount of data, complex deep neural networks are able to perform greatly over a broad range of computer vision tasks, large labeled datasets are often prohibitively expensive and time-consuming to retrieve, especially if we are counting on manually annotating the images to obtain the ground-truths. The annotation problem reveals to be one the bottleneck of training on large data.

In this context, the use of synthetic datasets combined with **domain adaptation** techniques tries to address this limit. The use of synthetic datasets in the training phase strongly reduces the great computational effort required by manual annotation: synthetically generated images are already provided with ground-truth labels. However, directly applying a model trained on a large-scale labeled source domain to another sparsely unlabeled target domain has been proved to be ineffective. A performance drop is experienced due to the presence of a change in the data distribution between the two domains, i.e. *domain shift*. Domain adaptation techniques are introduced to deal with this domain gap, aiming to learn a model from a labeled source domain that can generalize well to a different (but related) unlabeled target domain. The objective is to train the model in two different steps:

1) the model is trained on a synthetic dataset, for which ground truth are already available;
2) the model is adapted to match the gap with a real dataset.

The last aspect to be considered is the inference speed. An autonomous driving application as the one we are considering has an high demand for efficient inference speed, in order to be deployed in **real-time** scenarios. In this context, the key challenge is how to greatly improve the model efficiency, reducing the huge computational costs, while trying to keep the same segmentation accuracy level.

Our contribution consists of an unsupervised domain adaptation strategy based on adversarial learning to adapt a model trained on synthetic data to real world data. The network proposed is formed by two modules: a Convolutional Neural Network (CNN) for real-time semantic segmentation, BiSeNet [1], and a discriminator network. Since we are considering an autonomous-driving application, it is important to have a fast-inference model that conforms to different urban scenarios and different weather or lighting conditions. To meet these requirements, a thorough analysis of different discriminators has been carried out, in order to find the best suited for the task. Finally, different kind of optimizations techniques are deployed to speed up and, hopefully, even increase the performances of our models.

II. Related Works

Although in recent years several studies on both semantic segmentation networks and domain adaptation techniques have been carried out by researchers, only a few of them consider the real-time perspective and optimize networks for autonomous driving application.

A. Semantic Segmentation

Modern segmentation methods are predominantly based on the recent developments for deep neural network architectures. An exhaustive overview of the available deep-learning-based semantic segmentation methods has been carried out by Hao et al. in [2]. As already briefly stated in the introduction, semantic segmentation's aim is to partition the image into mutually exclusive subsets, in which each subset represents a meaningful region of the original image. Using deep neural network, through the learning process, we are able to gradually close the inconsistency gap between high-level semantics, or global information, and low-level features, or local information. Segmentation methods are grouped according to the supervision level: full supervised, weak supervised and semi-supervised. Among supervised methods, it is worth mentioning:

- *Context-based methods*, that leverages context information, first contemplated by Lucchi et al, [3], then applied in DilateNet by Yu et al., [4];
- *GAN-based methods*: Generative Adversarial Networks achieved great performances in many application and were first explored in [5] by Luc et al., proposing an architecture based on a segmentation network and an adversarial network.
- *Real-time methods*, introduced to solve the huge computational cost which hinders the application in some real-time segmentation contexts.

Among the different real-time networks, we consider BiSeNet (Bilateral Segmentation Network) [1], one of the most efficient and popular solution. BiSeNet achieves real-time inference speed and excellent performance combining information from two distinct "paths" of the network:

- Spatial Path, to preserve the spatial size of the original input image and encode affluent spatial information;
- Context Path, to provide sufficient receptive fields.

B. Domain adaptation

As already explained in the introduction, domain adaptation techniques are necessary when dealing with the domain shift between the synthetic dataset used for training and the real-world dataset on which we want to deploy our model. A complete overview of single-source deep unsupervised visual domain adaptation has been carried out by Zhao et al. in [6]. Domain adaptation is a specialized form of transfer learning whose goal is to learn a model from a labeled source domain that can generalize well to a different, but related, unlabeled target domain.

For our experiments, we started from an adversarial discriminative model, based on the single-level unsupervised domain adaptation strategy proposed in [7]. The adversarial learning technique consists in learning to adapt the output space features of two different datasets, training the network with one of this and at the same time feeding it also with the images of the second (target). Then a discriminator will try to distinguish the two outputs of the segmentation network and the result will be used to update the weights of the latter. In practice, the discriminator tries to identify to which dataset the images given in input belong, while instead the segmentation network is trained both trying to make the prediction as close as accurate and to get the discriminator wrong.

III. Methods

In this section we describe the adversarial training procedure we have developed, then the network architecture which was implemented and, finally, the optimization techniques adopted.

A. Training procedure

Our domain adaptation algorithm is based on the method proposed in [7]. As already anticipated in the section II-B, we have two modules: a segmentation network G and the discriminator D. Two sets of images $\{\mathcal{I}_S\}, \{\mathcal{I}_T\} \in \mathbb{R}^{H\times W\times C}$ from source and target domains, respectively IDDA and CamVid (further discussed in section IV-A). We first forward the source image I_s to the segmentation network for optimizing G. Then we predict the segmentation softmax output P_t for the target image I_t (without annotations). Since our goal is to make segmentation predictions P of source and target images (i.e., P_s and P_t) close to each other, we use these two predictions as the input to the discriminator D to distinguish whether the input is from the source or target domain. With an adversarial loss on the target prediction, the network propagates gradients from D to G, which would encourage G to generate similar segmentation distributions in the target domain to the source prediction. The adaptation task can now be formulated considering two different loss functions, one for each module. Thus:

$$\mathcal{L}(I_s, I_t) = \mathcal{L}_{seg}(I_s) + \lambda_{adv}\mathcal{L}_{adv}(I_t) \tag{1}$$

where $\mathcal{L}_{seg}$ is the segmentation loss for the source domain, $\mathcal{L}_{adv}$ is the adversarial loss that adapts predicted segmentations of target images to the distribution of source predictions, and λ_{adv} is the weight used to balance the two losses.

1) Discriminator training: For the discriminator training, given the segmentation softmax output $P = G(I) \in \mathbb{R}^{H\times W\times C}$, we forward P to a specific discriminator D using a binary cross-entropy loss with logits L_d (`BCEwithLogitLoss`) for the two classes (i.e., source and target). This loss combines a Sigmoid layer with the `BCELoss`, thus obtaining a more numerically stable result. The loss can be written as:

$$\begin{aligned}\mathcal{L}_d(P) = -\sum_{h,w}(1-z)\log(D(P)^{(h,w,0)}) \\ + z\log(D(P)^{(h,w,1)})\end{aligned} \tag{2}$$

where $z = 0$ if the sample is drawn from the target domain, and $z = 1$ for the sample from the source domain.

2) Segmentation training: For the segmentation training we define the loss as the dice loss for images from the source domain:

$$\mathcal{L}_{dice}(P_s, G_s) = 1 - 2\frac{\sum_i^N p_i g_i}{\sum_i^N (p_i + g_i)} \tag{3}$$

where p_i corresponds to the prediction of the i_{th} pixel of P and g_i is the corresponding ground-truth. For the training with the source domain, following the advice in [1], we add two specific auxiliary loss functions to supervise the output of the Context Path. Thus, we can write the segmentation loss as:

$$\mathcal{L}_{seg}(I_s) = \mathcal{L}_{dice}(P_s, G_s) + \alpha \sum_{i=2}^{K} \mathcal{L}_{dice}(P_{s_i}, G_s) \quad (4)$$

where α is used to balance the weight of the principal loss and auxiliary losses, which is equal to 1 in our case, and K, the index for losses, is set to 3, thus having 2 additional losses.

Then, when training the segmentation model with the target domain, we forward the images in this domain to G and obtain the prediciton $P_t = G(I_t)$. To make the distribution of P_t closer to P_s we use an adversarial loss L_{adv} as following:

$$L_{adv}(I_t) = -\sum_{h,w} \log(D(P_t)^{(h,w,1)}) \quad (5)$$

As already stated in the introduction, this loss is designed to train the segmentation network and "fool" the discriminator by maximizing the probability of the target prediction being considered as the source prediction.

B. Network architecture

The proposed model consists of two parts:

1) a semantic segmentation model to predict output results,
2) a discriminator to distinguish whether the input is from the source or target segmentation output

and the general scheme is shown in Figure 1.

1) Semantic Segmentation network: with an adversarial loss, the proposed segmentation model aims to fool the discriminator, with the goal of generating similar distributions in the output space for either source or target images. BiSeNet is implemented as segmentation network. The Spatial Path contains three layers and each layer contains a convolution with stride 2, followed by batch normalization layer and ReLU activation layer. The Context Path deploys a lightweight model to quickly downsample the feature map and obtain a large receptive field, then a global average pooling is set to extract the maximum one. After that, the up-sampled output features of global pooling and the features of the lightweight model are combined. An *Attention refinement module* (ARM) is used to refine the features of each stage: ARM employs global average pooling, as well as convolution and sigmoid activation layers to compute an attention vector to guide the feature learning. Finally, a *Feature fusion module* (FFM) is deployed to combine the features of the two paths: Spatial Path's features encode mostly richly detailed information, while Context Path's mainly context ones. The output features are concatenated, then batch normalization is used to balance the scales of the features. After two activation layers, ReLU and sigmoid, average pooling is used on the concatenated features to compute a weight vector, which is used to re-weight the features.

2) Discriminator networks: We implement three different discriminators, starting from the one proposed by [7] to achieve lighter model, searching for faster solutions, in terms of both training and evaluating time. The proposed discriminator architectures are outlined below.

Fully-convolutional discriminator (FCD): the first discriminator (AdaptSegNet [7]) consists of 5 fully convolutional layers with kernel 4 × 4 and a stride of 2. Each convolutional layer is followed by a Leaky ReLU (Rectified Linear Unit, thus having a small slope for negative values instead of a flat slope) with 20° angle for the negative slope, except for the last one.

Tucker-factorization discriminator (TFD): the second discriminator exploits the advantages of rank factorization and, in particular, of *tucker decomposition* applied on the tensors of the kernels of the convolutional layers. Tucker decomposition is a high-order generalization of singular value decomposition (SVD) and principal component analysis (PCA), and it often achieves orders-of-magnitude higher data compression ratio than matrix compression algorithms. In our case, the Tucker method is used to decompose the convolution kernel into a small core tensor with key information and two factor matrices reflecting the linear relationship in the third dimension and fourth dimension of the convolution kernel respectively [8]. The objective of this decomposition is to generate a separable convolutional layer that replaces the original with a series of small efficient ones. Figure 2 shows the basic idea on how Tucker decomposition approximates a large-size tensor with a small-size tensor [9]. In particular we can see how a d-way tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times ... \times I_d}$ can be compressed by Tucker decomposition into a core tensor $\mathcal{G} \in \mathbb{R}^{\mathbb{R}_1 \times \mathbb{R}_2 \times ... \times \mathbb{R}_d}$ and d factor matrices $\{\mathcal{A}_k \in \mathbb{R}^{I_k \times R_k}\}_{k=1}^{d}$ as follows:

$$\mathcal{X} \approx \mathcal{G} \times_1 \mathcal{A}_1 \times_2 \mathcal{A}_2 \times ... \times_d \mathcal{A}_d \quad (6)$$

where all columns are orthonormal inside each factor matrix $\mathcal{A}_k$. We call $R = [R_1, R_2, ..., R_d]$ as the multilinear rank of X . When $R_k \ll I_k$, $\mathcal{X}$ can be represented with the above Tucker format at a much lower cost. Once all factor matrices are obtained, the core tensor can be computed as:

$$\mathcal{G} \approx \mathcal{X} \times_1 \mathcal{A}_1^T \times_2 \mathcal{A}_2^T \times ... \times_d \mathcal{A}_d^T \quad (7)$$

For the implementation we exploit the python library Tensorly-torch [10] that provides high-level tested API to use tensor decomposition methods in neural deep networks. These tensor methods allow us to leverage and preserve the structure, for both individual layers and whole network in order to obtain models that are potentially lighter, more robust and more capable to generalize.

Depth-Wise Discriminator (DWD): for the third discriminator, we implemented a custom version of convolution, in particular the depth-wise separable convolution proposed in [11]. The objective of this different operation is to reduce the number of parameters of the model, thus reducing the chance of over-fitting, the training time and the model size. The depth-wise separable convolution deals not just with the spatial dimensions of an image and kernel (i.e. width and height), but also with the depth dimension (i.e. the number of channels). Each convolutional layer of this type splits a kernel into 2 separate kernels that perform two convolutions:

- a depth-wise convolution, or a convolution without changing the depth, with kernel 4×4 and stride 2;

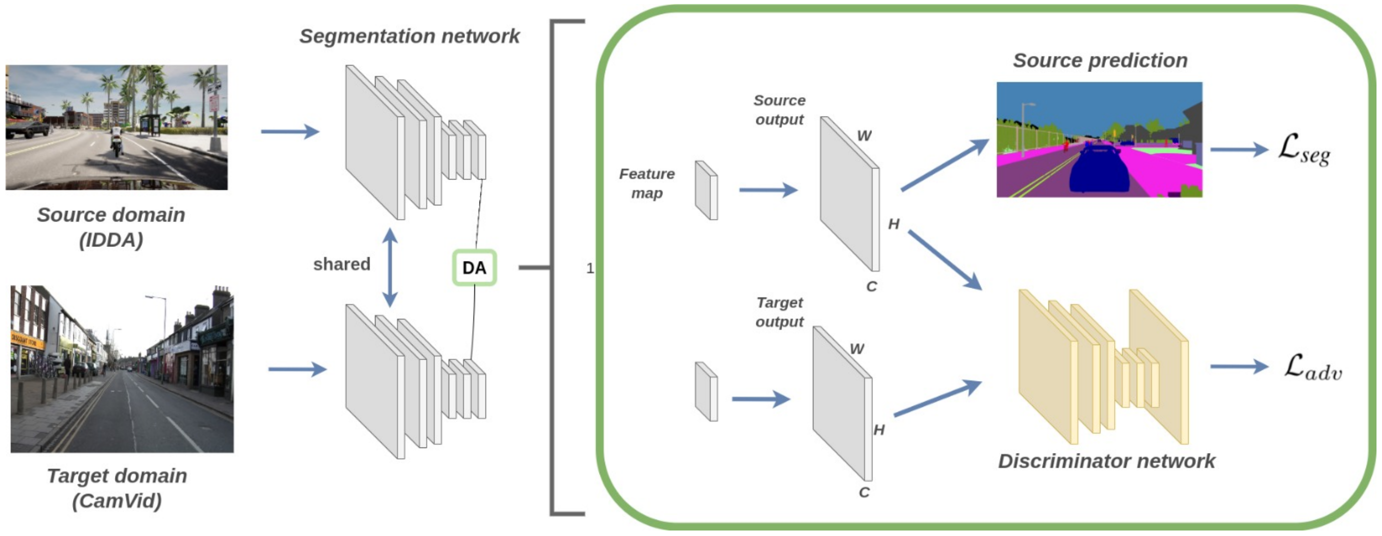

Fig. 1. Algorithm overview

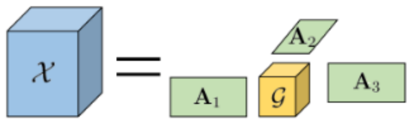
$\mathcal{X}$ = $\mathbf{A}_1$ $\mathcal{G}$ $\mathbf{A}_2$ $\mathbf{A}_3$

Fig. 2. Tucker decomposition

- and a point-wise convolution to increase the number of channels of each image, with kernel 1×1. This kernel iterates through every single point and, thus, has a depth equal to the number of channels of the input image.

The main advantage is that in the depth-wise separable convolution, we only transform the image once. Then, we take the transformed image and simply elongate it to the different channels. Without having to transform the image iteratively, we are able to save up on computational power.

The architecture of our discriminator consists of 5 of this depth-wise convolutional layers, each one of them followed by a Leaky ReLU with 20° angle for the negative slope.

C. Optimization

Since we are dealing with a real-time inference scenario, the significant amount of computing, memory, and power may become a bottleneck. One of the methods for tackling this energy efficiency is enabling inference efficiency and training efficiency. In order to lighten and speed up our network, different optimization techniques are explored and tested. Our objective is to obtain a model which is faster to train and to evaluate, with the same accuracy of the non-optimized model or, ideally, even improved.

1) Pruning: pruning is one of the methods for inference to efficiently produce models smaller in size, more memory-efficient, more power-efficient and faster at inference with minimal loss in accuracy. Among the different type of pruning techniques, we explored **weight pruning** ([12]) on convolutional layers. This pruning method is based on the magnitude of the weight: during the procedure we remove those weights that are ”low” in magnitude, i.e. we set to 0 all weights below a certain threshold. This type of pruning does not reduce the number of parameters, but rather should speed up the training and the inference because our network is able to pass along zeros faster. Weight pruning is studied on two different dimensions:

- the ”where” dimension: both on single modules and on the global model, both on the segmentation and the discriminator network
- the ”when” dimension: both during training and post training; for what concerns during training technique we study the behavior with a fixed and with an increasing sparsity (i.e. amount of pruned weights).

2) Dropout: dropout is a regularization method that approximates training a large number of neural networks with different architectures in parallel. During training, some number of layer outputs are randomly ignored or “dropped out”. This has the effect of making the layer look-like and be treated-like a layer with a different number of nodes and connectivity to the prior layer. In effect, each update to a layer during training is performed with a different “view” of the configured layer.

IV. Experiments

In this section, we present the experiments carried out to validate the proposed domain adaptation method for semantic segmentation. Firstly, we train and test our segmentation model BiseNet on the CamVid dataset, in order to obtain our first result which is used as ”upper-bound” for the second part. Therefore, our adversarial training procedure is deployed, training the model on IDDA and testing it on the CamVid dataset. A thorough description of the configuration details is carried out, along with the adversarial training procedure and the optimization schedule.

A. Datasets

1) CamVid: (Cambridge-driving Labeled Video Database) [13] is a collection of videos with object class semantic labels, filmed from the perspective of an automobile driving in some real-world scenarios. The database used consists of 22 minutes, 14 seconds of footage, captured at 960x720 pixel frames at 30 frames per second. The frames were hand-labeled to identify 32 classes of interest, but for our purpose we consider only 12 of them (Bicyclist, Building, Car, Column Pole, Fence, Pedestrian, Road, Sidewalk, Sky, Sign Symbol, Tree, and the Background class). Four HD-video sequences are included: three of them are shot at daytime, one was shot at dusk, so objects are significantly darker and grainier. The environment is mixed urban and residential. The subset of frames we considered consists of 468 images for the training set, and 234 images for the test set, considering images from all of the four sequences.

2) IDDA: (ItalDesign DAtaset) [14] is a large scale, synthetic dataset for semantic segmentation in an autonomous-driving scenario, generated with CARLA [15]. This dataset provides more than one million of labeled images as the sum of 105 different scenarios over three axes of variability: 5 viewpoints, given by different type of car on which the virtual camera is put, 7 towns, with different characteristics, and 3 weather conditions. The ground-truth labels are automatically produced by the 3D graphic engine that generates the images: 27 semantic classes are produced. Once again, for our purpose we consider only the 12 classes already used in CamVid. Translation, grouping and color encoding actions are required to deal with the label mismatch between the two datasets.

Data augmentation techniques are explored to both enhance the size and the quality of our training dataset and reduce the risk of overfitting. A further analysis of the techniques which have been tested is reported in the Appendix section A. From the simulations, we found the best configuration to be Horizontal Flip with 50% of probability combined with Gaussian Blur with $\sigma \in [1, 2]$, applied to both datasets, i.e. source and target.

B. Configuration details

1) Semantic segmentation network: for the training and testing of our semantic segmentation network (BiseNet) on CamVid, **transfer learning** is exploited. The Context Path is initialized with the values of a pre-trained *ResNet*, a Residual Neural Network, which introduces the "identity shortcut connection" to address the vanishing gradient problem during the back-propagation step. In our network, both *ResNet-18* and *ResNet-101* are explored and studied: their weights, obtained from the training on ImageNet, [16], are the starting weights of our Context Path. We refer to *ResNet* as the backbone of our segmentation network.

Three gradient-based optimizers for the losses are considered: *RMSprop*, *Adam* and *SGD*. Preliminary tests (Appendix section B) have shown that *SGD* is the best-performing one. As [1] suggests, we use mini-batch stochastic gradient descent with batch size 4, momentum 0.9 and weight decay $1e^4$. For each loss, we apply the poly-learning rate strategy in which the initial rate is multiplied by $(1 - \frac{\text{iter}}{\text{max}_{\text{iter}}})^{0.9}$. The initial learning rate is $2.5e^{-2}$.

2) Discriminator network: for training the discriminator, we use the Adam optimizer, as suggested in [7], with the learning rate as 10^4 and the same polynomial decay as the segmentation network. The momentum is set as 0.9 and 0.99, which is the default value.

3) Adversarial training procedure: we finally train our network, implementing the **unsupervised domain adaptation** method explained in section III. To train the proposed adaptation model, we jointly train the segmentation network and discriminators in one stage. In each training batch, we forward the source image I_s to optimize the segmentation network for L_{seg} in eq. 4 and generate the output P_s. For the target image I_t, we obtain the segmentation output P_t, and pass it along with P_s to the discriminator for optimizing L_d in eq. 2. In addition, we compute the adversarial loss L_{adv} in eq. 5 for the target prediction P_t. During this procedure, it is essential to balance the contribution of the segmentation and adversarial losses (eq. 1): as [7] found, a smaller λ_{adv} may not facilitate the training process significantly, while a larger value may propagate incorrect gradients to the network. We empirically choose λ_{adv} as 0.001.

C. Optimization schedule

Our proposed optimization techniques are applied on the best-performing model, considering the three different discriminators. To establish which weights to prune, we decide to use the L1 norm, also known as Manhattan Distance or Taxicab norm, which is the sum of the magnitudes of the weights, (i.e. the sum of absolute difference of the components of the weight vectors). Four different experiments are carried out to find the best-suited model:

- two experiments on during-training pruning (DTP): in this case the discriminator model has been pruned globally at each epoch, but considering firstly a fixed sparsity of 0.2, and then a sparsity that increments with the epochs with a rate given by $i_{\text{epoch}} * 0.005$.
- two experiments on post-training pruning (PTP): in this case the segmentation model has been pruned both on single modules and on the overall model. In this case different levels of sparsity were considered in order to find the higher value for which the model could be pruned without compromising the accuracy (and the mIou). Specifically, sparsity levels are taken from the interval $[0, 0.85]$ with step 0.05;

Then, we study the behavior of dropout only applied to the best discriminator network found. Specifically, we add dropout after every convolution and activation in the forward step, as suggested in [17], with probability of an element to be zeroed $p = 0.01$.

D. Metrics

Two different metrics are used to evaluate our results:

TABLE I
PERFORMANCE OF DISCRIMINATORS FOR DOMAIN ADAPTATION

		Accuracy	mIou	training time
seg	**Upper Bound**	0.882	0.675	10:20
adv	**BiseNet+FCD**	**0.689**	**0.337**	25:30
	BiseNet+TFD	0.664	0.334	28:35
	BiseNet+DWD	0.618	0,297	**18:55**

- pixel accuracy:

$$acc = \frac{\text{number of correctly-classified pixels}}{\text{total predictions}}$$

- intersection over Union (IoU):

$$mIoU = \frac{\text{target} \cap \text{prediction}}{\text{target} \cup \text{prediction}}$$

which quantifies the percent overlap between the target mask and our prediction output.

V. Results

In this section we present some of the experimental results of our simulations. Firstly, we found the best configuration of number of epochs and backbone network to train BiseNet in the semantic segmentation task: this result represents our upper bound and has been obtained training BiseNet for 100 epochs with *ResNet-101* as backbone. The unsupervised domain adaptation results are obtained training the architecture showed in Figure 1 with the same configuration for the segmentation network. Table I shows these results for the different discriminators. As expected, domain adaptation performances worsen a lot compared to the upper bound: this is explained considering the domain gap between the synthetic dataset (IDDA), on which our model is trained, and the real-world dataset (CamVid), on which our model is tested. Considering accuracy and mIoU performances, our best model is BiseNet+FCD, while considering training time the best model is BiseNet+DWD. It is worth noticing that, even if BiseNet+TFD showed us results comparable to our best model in terms of accuracy and mIoU, its deployment was probably compromised by the use of the ”Tensorly”, which is not optimized for our GPU. Finally, since the objective is to obtain a fast, light and high accuracy model, we select BiseNet+FCD (Baseline), since it has shown the best peformances in terms of accuracy and mIoU, and we apply our optimization techniques in order to reduce its training and testing time.

Table II shows the results of the different optimization techniques applied **during-training**. In particular, our discriminator model has been optimized considering dropout, global unstructured pruning and both of them combined. Note that, as explained in the methods section III-C, since we are applying pruning with a sparsity level that increases during training, we show also the initial and final epoch training time to see how our model has lighten with the increase of epochs. Parameters are the ones explained in the Experiments section IV-C. All of the optimization techniques show an improvement in terms both of performances (accuracy and mIoU) and training time.

TABLE II
PERFORMANCE OF DIFFERENT OPTIMIZATION TECHNIQUES

	Accuracy	mIou	initial training time	final training time
Baseline (BiseNet+FCD)	0.689	0.337	25:30	25:30
Fixed Pruning	0.652	0.297	25:30	23:10
Incremental Pruning	0.691	**0.371**	25:30	19:00
Dropout	0.688	0.340	21:30	21:30
Inc Pruning+Dropout	**0.701**	0.341	25:30	**18:00**

In particular, during-training pruning with incremental sparsity combined with dropout with $p = 0.01$ results in a 1.2% increase in accuracy, a 0.4% increase in mIoU and a 28% decrease in training time. However, with pruning-only we are able to achieve a 3.4% increase in mIoU with a 25% decrease in training time. Since we consider mIoU to be the most robust metric for our task, this latter optimized model (BiseNet+FCD optimized with pruning-only) is considered for the next step.

Finally, Figures 3 and 4 show the results of the **post-training** pruning simulations: in particular our segmentation model BiseNet trained with our domain adaptation method (considering the optimized model aforementioned) has been pruned with different sparsity levels, both on single modules and on the overall model. If we consider global pruning, it is immediate to notice that performances of the model do not decrease significantly until almost 40% sparsity is reached: this means that the segmentation model can be pruned greatly post-training, without compromising its accuracy and mIoU. For PTP on single modules the decrease in performances is experienced for lower levels of sparsity.

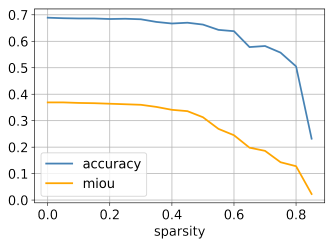

Fig. 3. PTP on modules

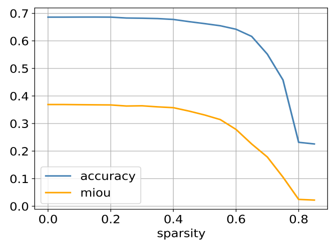

Fig. 4. PTP on global model

Figure 5 gives a visual idea for the performances of our best optimized model, compared to our initial upper bound and with the baseline. It is immediate to notice that, as expected, objects such as column poles and sign symbols are harder to adapt since they easily get merged with the background class. However, we can also highlight that our ”Adapted and Optimized model” is able to outline the shape of object more precisely with respect to the ”Adapted model”.

VI. Conclusions

We explored real-time domain adaptation in semantic segmentation, evaluating different discriminator together with two different optimization techniques. We were able to achieve better performances with respect to our baseline, increasing

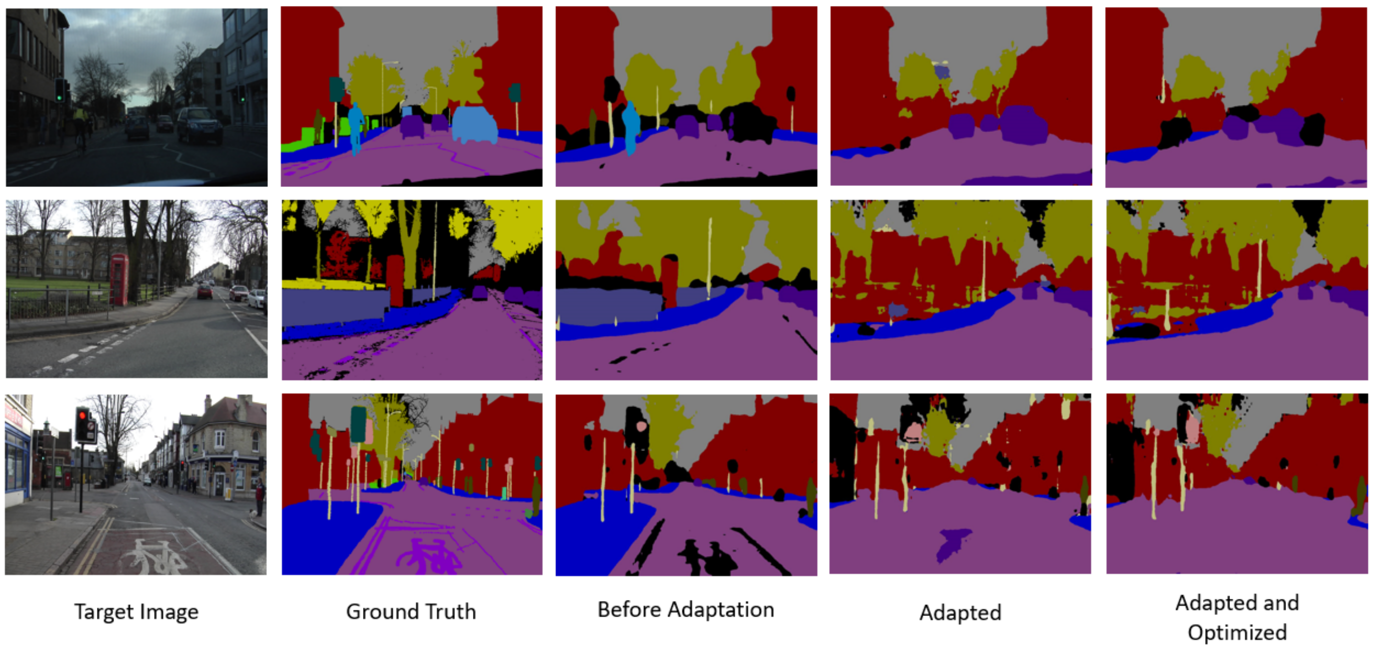

Fig. 5. Examples of results of the adapted model's prediction. Note that "before adaptation is the result of the prediction of our "Upper Bound" (table I), "Adapted" is the result of our "Baseline" (table II), and "Adapted and optimized" is the result of our best adapted model (BiseNet+FCD + DT-pruning + PT-pruning)

both the accuracy and the mIoU percentages of the model, and at the same time reducing its training time.

Future implementations may consider other discriminators or variations of the ones already proposed: for the "tucker-factorization" discriminator, for example, the CP-factorization may be interesting to analyse, which factorizes a tensor into a linear combination of rank one tensors. For what concerns pruning, the "what-to-prune" dimension may be explore: we studied when and where to apply pruning, but due to resource and time limitations, the choice of the model to prune has not been explored. Specifically, it would be interesting to see how pruning during training and dropout in the segmentation model behave. Finally, weight sharing [18] and quantization techniques were partially deployed and briefly tested for our pipeline, but it was not carried out a thorough analysis on them. It would be interesting to see how they behave on these conditions.

References

[1] Yu, Changqian et al. "BiSeNet: Bilateral Segmentation Network for Real-Time Semantic Segmentation." Lecture Notes in Computer Science (2018): 334–349. Crossref. Web.

[2] Hao, Shijie et al. "A Brief Survey on Semantic Segmentation with Deep Learning." Neurocomputing 406 (2020): 302-321.

[3] A. Lucchi, Yunpeng Li, X. Boix, K. Smith and P. Fua, "Are spatial and global constraints really necessary for segmentation?," 2011 International Conference on Computer Vision, 2011, pp. 9-16.

[4] Yu, Fisher, and Vladlen Koltun. "Multi-scale context aggregation by dilated convolutions." arXiv preprint arXiv:1511.07122 (2015).

[5] Luc, Pauline, et al. "Semantic segmentation using adversarial networks." arXiv preprint arXiv:1611.08408 (2016).

[6] S. Zhao et al., "A Review of Single-Source Deep Unsupervised Visual Domain Adaptation," in IEEE Transactions on Neural Networks and Learning Systems

[7] Tsai, Yi-Hsuan et al. "Learning to Adapt Structured Output Space for Semantic Segmentation." 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (2018)

[8] Song D., Zhang P., and Li F. "Speeding Up Deep Convolutional Neural Networks Based on Tucker-CP Decomposition". 5th International Conference on Machine Learning Technologies (2020) DOI:https://doi.org/10.1145/3409073.3409094

[9] Zhang K., Zhang X and Zhang Z., "Tucker Tensor Decomposition on FPGA", Department of Electrical & Computer Engineering, University of California, Santa Barbara, CA 93106 (2019)

[10] Jean Kossaifi and Yannis Panagakis and Anima Anandkumar and Maja Pantic, TensorLy: Tensor Learning in Python, Journal of Machine Learning Research (JMLR), vol. 20, num.26, 2019.

[11] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions", 2017, 1610.02357, arXiv

[12] Bondarenko, Andrey Borisov, Arkady Aleksejeva, Ludmila. (2015). Neurons vs Weights Pruning in Artificial Neural Networks. Environment. Technology. Resources. Proceedings of the International Scientific and Practical Conference. 3. 22.

[13] Gabriel, Jourdan and Fauqueur, Julien and Cipolla, Roberto. (2009). Semantic object classes in video: A high-definition ground truth database. Pattern Recognition Letters. 30. 88-97. 10.1016/j.patrec.2008.04.005.

[14] Alberti, Emanuele and Tavera, Antonio and Masone, Carlo and Caputo, Barbara. (2020). IDDA: A Large-Scale Multi-Domain Dataset for Autonomous Driving. IEEE Robotics and Automation Letters. PP. 1-1. 10.1109/LRA.2020.3009075.

[15] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in Proceedings of the 1st Annual Conference on Robot Learning, 2017, pp. 1–16

[16] J. Deng, W. Dong, R. Socher, L. -J. Li, Kai Li and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," 2009 IEEE Conference on Computer Vision and Pattern Recognition, 2009, pp. 248-255, doi:10.1109/CVPR.2009.5206848.

[17] Park, S., Kwak, N. (2016). Analysis on the Dropout Effect in Convolutional Neural Networks. ACCV.

[18] Han, Song et al. "Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding." arXiv: Computer Vision and Pattern Recognition (2016)

TABLE III
PERFORMANCE OF DIFFERENT DATA AUGMENTATION TECHNIQUES

	Image	Pixel					
	HorizontalFlip	GaussianBlur	LinearContrast	AddGaussianNoise	Multiply	Accuracy	MIou
Baseline	x	x	x	x	x	0.875	0.648
Aug 1	0.5	x	x	x	x	0.881	0.668
Aug 2	0.5	[0,3]	x	x	x	0.877	0.666
Aug 3	**0.5**	**[1, 2]**	x	x	x	**0.882**	**0.675**
Aug 4	x	[0,3]	[0,75,1.5]	[0,0.05*255], 0.5	[0.8,1.2], 0.5	0.876	0.624
Aug 5	0.5	[1,2]	[0.75,1.5]	[0,0.05*255], 0.5	[0.8,1.2], 0.5	0.881	0.656

APPENDIX

In this brief appendix, we show some simulation results on those parameters or configuration details that were not fine-tuned or suggested by previous works or researches. In particular, in order to propose a more complete overview on the configuration choices taken, we decide to report results on data augmentation technique and the optimizer deployed.

A. Study of the data augmentation techniques

The table III shows the different configurations tested. Note that the baseline is obtained training BiseNet with backbone *ResNet-101* for 100 epochs on CamVid. The other results that consider data augmentation are obtained with the same training procedure.

B. Study of optimizers

RMSprop and *Adam* (Adaptive Moment Estimation) compute adaptive learning rates for each parameter. Whilst *RMSprop* scales the gradients by the inverse of a moving average, *Adam* compute those moving averages considering both gradients and squared gradients. Due to resource and time limitations, we decide to select the best optimizer for the task training BiseNet with backbone *ResNet-18*, for 50 epochs. Results are showed in table below IV.

TABLE IV
PERFORMANCE OF DIFFERENT OPTIMIZERS

	Accuracy	mIou
SGD	**0.850**	**0.546**
RMSprop	0.801	0.472
ADAM	0.733	0.401