Real-time Domain Adaptation in Semantic Segmentation

Dragos-Constantin Buhnila, Fabio Gigante, Jad Kharrat
s331394, s328890, s324896
*

Abstract

Semantic segmentation is a fundamental task in computer vision, essential for real-time applications like autonomous driving and surveillance. However, domain shifts between training and deployment environments can severely worsen model performance. This project explores real-time domain adaptation in semantic segmentation networks, focusing on PIDNet trained on the LoveDA dataset. Despite implementing adversarial training and image-to-image translation techniques, these approaches failed to significantly improve model performance on the target domain. In contrast, data augmentation strategies proved effective, leading to a measurable improvement in mean Intersection over Union (mIoU) when applied during training. These results may highlight the challenges in applying advanced domain adaptation methods to real-time systems and seem to suggest that simple data augmentation may offer a more practical solution in certain scenarios.

1. Introduction

In real-time applications, the demand for accurate and efficient segmentation models is paramount. However, a significant challenge arises when these models are deployed in environments different from the ones they were trained on, leading to a phenomenon known as domain shift. This discrepancy between the source (training) domain and the target (deployment) domain often results in a notable decline in model performance.

Domain adaptation aims to bridge this gap by enhancing the model's ability to generalize across different domains. In this project, we focus on unsupervised domain adaptation for semantic segmentation, using PIDNet, a state-of-the-art real-time segmentation network, trained on the LoveDA dataset, which comprises of satellite images of Chinese urban and rural landscapes. Our objective is to evaluate the impact of domain adaptation techniques, such as adversarial training and image-to-image translation, on the performance of semantic segmentation models.

Despite the promise of these advanced techniques, our experiments reveal that they do not yield significant performance improvements in our setup. Conversely, simpler methods like data augmentation demonstrate a notable increase in segmentation accuracy. The cause for these results may lay either in the real-time model's inability to perform DA, or in the domain gap not matching that which the techniques implemented are good at bridging.

2. Related Work

Semantic segmentation has evolved significantly with the advent of deep learning, becoming a cornerstone of various computer vision applications. Early reviews, such as [5], provide comprehensive insights into the advancements and challenges in this field. There we can find DeepLabV2 [1] listed as a relatively efficient and relatively precise model: it's distinguishing features are the use of atrous convolutions, atrous pyramid pooling, and CRFs. We use DeepLab to set a base in order to do a comparison with a real-time model.

Real-time semantic segmentation has emerged as a critical area, driven by the need for fast and efficient models. BiSeNet [11] proposed a bilateral structure to optimize the balance between accuracy and speed. PIDNet [2] tried to improve on this approach through a 3-branch structure that borrows PID structure from control theory. STDC [4] too started from BiSeNet, but built a brand new backbone more specific to segmentation tasks, while also merging the two branches to avoid redundancy; also, like PIDNet, it also focuses on boundaries, at least when training.

Domain adaptation seeks to address the performance drop observed when models trained in one domain are applied to another. Reviews like [12] discuss various strategies for unsupervised domain adaptation, highlighting the importance of bridging the gap between source and target domains. The LoveDA dataset [3] serves as a valuable resource for evaluating domain adaptation techniques, particularly in the context of remote sensing.

Several approaches have been proposed to tackle do-

*GitHub: github.com/dragosbuhnila/Real-time-Domain-Adaptation-in-Semantic-Segmentation

main shifts. Adversarial methods, as outlined in [9], utilize a game-theoretic approach to align feature distributions across domains. Image-to-image translation techniques, such as DACS [8], create synthetic target-like images from source domain data to enhance model generalization.

3. Method

3.1. Introduction

In the following, an overview of the 3 models and of the 2 DA techniques used ensues. For a more detailed explanation please consider looking at the respective original papers.

3.1.1 Base Models

The main models we used are the DeepLabV2 and PIDNet, which are non real-time and real-time respectively. Subsequently we introduced another real-time model called STDC.

3.1.2 DA Techniques

The main interest of the study is evaluating the performance of these DA techniques on real-time networks, but we hypotesize that causes for suboptimal results could be coming from the dataset used instead: Domain Adaption often tackles adapting synthetic to real-world domains, and the two techniques we covered are no exception, but our dataset is characterized by a different kind of gap, so results are not guaranteed.

AdaptSegNet: The first domain adaptation technique uses an adversarial approach aimed at adapting masks predictions directly, instead of modifying feature representation. It hosts a discriminator which we train to improve, while at the same time training our mask prediction to be hard to discriminate between source and target. As this training is extremely competitive it's beneficial to implement a balance between discriminator and adversarial training in a way that one module does not overpower the other one.

DACS: This other technique refines the image mixing paradigm ([10]), by mixing source and target images, and the source GT masks and target pseudo masks. The model is thus optimized on these mixed images, focusing on more confident pixels or masks.

3.2. Detailed Architecture Comparison: DeepLabv2 vs. PIDNet

In this section, we delve into the architectural details of DeepLabv2 and PIDNet to provide a comprehensive comparison, emphasizing their design philosophies, operational differences, and suitability for various semantic segmentation tasks.

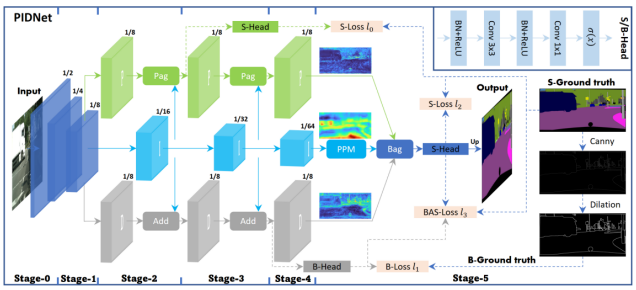

Figure 1. "PIDNet Architecture" [2]

3.2.1 DeepLabv2 Architecture

DeepLabv2 is a pioneering model in semantic segmentation, known for its ability to capture contextual information at multiple scales. Its architecture includes several key innovations, it leverages **Atrous Convolution** to expand the receptive field without losing resolution, enhancing contextual understanding. It employs a **ResNet-101 backbone**, which benefits from residual learning for deep network training. **Multi-Scale Image Processing**, facilitated by the ASPP module, improves segmentation of objects at different sizes. Finally, **Fully Connected CRFs** refine segmentation by improving boundary localization through spatial modeling.

3.2.2 PIDNet Architecture

PIDNet, designed for real-time applications, emphasizes efficiency without compromising too much on accuracy. Its architecture is influenced by control theory, particularly Proportional-Integral-Derivative (PID) controllers, which guide its design to balance responsiveness and stability:

- **Parallel Branches for Multi-Resolution Features**: PIDNet employs three parallel branches to parse detailed, context and boundary information, respectively, and employs boundary attention to guide the fusion of detailed and context branches. The Integrative Branch contains local and global contextual informations and serves as a backup for the other branches, so a flow of information is needed.
- **I and D Fusion**: Simple element-wise addition.
- **P and I Fusion**: Uses a Pixel-attention-guided (Pag) fusion module for selective feature incorporation.
- **Context Aggregation**: Employs a Parallel Aggregation Pyramid Pooling Module (PAPPM) for efficient multi-scale context capture.
- **Bag**: To guide the fusion of detailed (P) and context (I) representations a Boundary-attention-guided (Bag)

fusion module is used. Which force the model to trust the detailed branch more along the boundary region and utilize the context features to fill other areas.

3.2.3 Comparative Analysis

The performance of DeepLabv2 and PIDNet was compared to evaluate the trade-offs between accuracy and inference speed in classic versus real-time segmentation networks. This comparison was essential for understanding the impact of architectural choices on segmentation performance.

3.2.4 Training Setup

To compare Deeplab and PIDNET we used a fixed batch size of 24, with a learning rate of $2e-3$ and a no step-down policy. We decided to resize each input image to a size of $256x256$, so a version of the image with a $\frac{1}{16}$ of the original resolution for memory constraints of our environment.
We trained the models on 20 epochs with a shuffled DataLoader with no augmentations. For the loss computations we ignored the results of the pixel in which the gt mask was undefined, directly optimizing for 7 classes (indexes $0-6$).

3.2.5 Loss Functions

For DeepLab a single loss function was used, which was the Cross Entropy between our output and the ground truth mask. For PIDNet as many as four loss functions were used, each one with a weight assigned:

- **Semantic losses** l_0, l_2 We use two semantic losses, based on corss entropy: a "low-level" one is calculated immediately after the first Pag module, for better optimization of the whole network, and a "high-level" one is calculated after the Bag module, to optimize the actual prediction.
- **Boundary Loss** l_1 is a loss that uses a binary cross entropy weigthed for focusing on boundaries, calculated between the model Boundary prediction and the Boundary ground truth of the image (obtained using Canny+Dilation on the original Image).
- **Boundary Awareness Loss** l_3 is a loss that uses the output of the boundary head to coordinate the segmentation performed by the main head.

$$l_3 = -\sum_{i,c}\{1 : b_i > t\}\,(s_{i,c}\log\hat{s}_{i,c})$$

where t refers to a predefined threshold, and b_i, $s_{i,c}$, and $\hat{s}_{i,c}$ denote the output of the boundary head, the segmentation ground-truth, and the predicted result of the i-th pixel for class c, respectively.

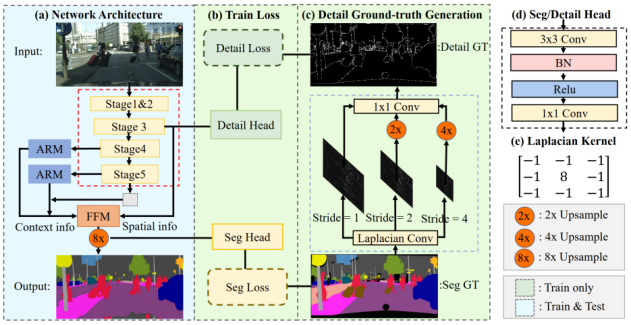

Figure 2. Structure of the BiSeNet inspired STDC implementation

3.3. STDC

In order to better understand the relationship between the DA techniques, the Dataset, and PIDNet, we decided to try and gather results on another real-time network too, which is STDC (see figure 2 for the structure).

3.3.1 STDC Architecture Key Points

Short-Term Dense Concatenate Module The main module, used throughout most layers, which focuses on extracting features relevant for semantic segmentation, by using scalable receptive field and multi-scale information, and keeping a lower number of channels in deeper layer to avoid redundancy. The different levels of the module are fused right before providing the output.

Decoder The lower level stages of the model, containing spatial information, are fused with decoded features from the higher, context, layers, through the same Feature Fusion Module that BiSeNet uses. The decoded features take features from increasingly smaller stages, up until a global average pool, and merges them bottom-up using upsampling of the smaller layer and ARMing (Attention Refinement Module) of the bigger layer.

Losses: Segmentation and Detail Losses are computed from a couple of different heads (one may also decide to also add others at every stage): the segmentation head, placed after the FFM, uses Cross Entropy, while the detail head, coming out from at the last spatial layer (stage 3 in the figure), uses a combination of Binary Cross Entropy, and Dice, as the latter is insensitive to the number of foreground/background pixels make it resilient to class imbalance. The boundary ground truth is extracted thanks to Laplacian convolution with different strides.

$$L_{\text{dice}}(p_d, g_d) = 1 - \frac{2\sum_i p_d^i g_d^i + \epsilon}{\sum_i (p_d^i)^2 + \sum_i (g_d^i)^2 + \epsilon} \qquad (1)$$

where i denotes the i-th pixel, and ϵ is a Laplace smoothing term to avoid division by zero, set to 1.

3.4. Domain Adaptation Techniques

3.4.1 Adversarial Training

We implemented an **adversarial training strategy** for semantic segmentation following the approach in [9]. We used a **Single level Adversarial Training**, meaning that we only considered the output of the last layer of the segmentation model during the training of our **discriminator**. The key idea is to use a **discriminator network** to guide the segmentation model toward generating **source outputs** that resemble those of the **target domain**.
Segmentation model and Discriminator are playing against each other:

$$\max_{\mathbf{D}} \min_{\mathbf{G}} \mathcal{L}(I_s, I_t).$$

The ultimate goal is to **minimize the segmentation loss** in **G** for source images (based on the segmentation model output), while **maximizing the probability** of target predictions being considered as source predictions (based on the discriminator output).

Discriminator Structure The discriminator, D, is a fully convolutional network that takes as input the **prediction** of the segmentation model and predicts whether the segmentation map is from the **source or target domain**. It is structured as follows: A series of **five convolutional layers** with **stride 2**, **kernel size of 4**, and **number of channels** respectively $\{64, 128, 256, 512, 1\}$ for downsampling; **Activation: LeakyReLU** with a negative slope of **0.2** except for the last layer.

No batch-normalization layer is used.

Training Strategy We use a two-step adversarial training process.

1. **Segmentation Model Update:** Trained with a **segmentation loss** (L_{seg}) using labeled source domain images. (Here we include all the losses that the segmentation model normally uses for a segmentation task). An **adversarial loss** (L_{adv}) is added to encourage **domain-invariant feature learning**. That is a **binary cross entropy loss** on the output of the discriminator feeded with the target domain inputs and a **tensor of ones** of the same size. (This is because the discriminator is trained to classify source domain inputs with a tensor of 1). The segmentation model is updated to **fool the discriminator** by minimizing:

$$L_{\text{seg}} + \lambda_{\text{adv}} L_{\text{adv}} \tag{2}$$

2. **Discriminator Update:** The discriminator is trained using a **binary cross-entropy loss** to correctly classify segmentation maps as **source or target domain**. The strategy is, as mentioned before, to use a **binary cross entropy loss** between the output of the discriminator feeded with **source/target domain inputs** against a **tensor of zeros/ones** of the same size. In this way the discriminator is trained to correctly tell the two domain outputs apart. This is called the **Discriminative loss**.

When **adversarial loss** and **segmentation loss** are backpropagated the **discriminator is not updated**, and vice versa, when the **discriminative loss** is backpropagated the **segmentation model is not modified**.

3.4.2 Image-to-Image Translation

To improve domain adaptation, we incorporated **image-to-image translation** using a self-training approach inspired by [8]. This method aims to reduce the domain gap by generating more diverse and realistic target-like images from the source domain.

Method Overview: We employed **Domain Adaptive Contrastive Self-training (DACS)**, a method that enhances adaptation through **mixing source and target domain images**.

DACS leverages a technique inspired by ClassMix [10], which mixes images and their corresponding labels from different domains to create augmented training samples. The process is guided by the semantic labels of the source domain, ensuring that the mixed image retains meaningful semantic structure. The mixed image is then used as input to the segmentation model, while the mixed labels serve as the supervision signal.

The process consists of the following steps:

Image Selection: A source image x_s with its corresponding label y_s and a target image x_t (without labels) are randomly selected. Then we predict the mask for the target image, thus creating a pseudo-label y_t.

Class-wise Mixing: A subset of classes from the source label y_s is chosen, and the remaining regions in x_s are replaced with the same regions from x_t, for both the image and the mask.

Self-training: The resulting mixed image x_{mix} and its corresponding mixed label y_{mix} are used for training the segmentation model. Pixels where the model's predicted class has a confidence score above a certain threshold (0.968) have a high weight during cross entropy loss calculation. This ensures that only high-confidence predictions are used, reducing noise in the training process. For each mixed image, we also keep training the model on the source image, meaning that we propagate two losses for each batch.

4. Experiments

4.1. Dataset

For our purpose we used the Land-cover Domain Adaptive semantic segmentation (LoveDA) dataset. ”Compared to the existing datasets, the LoveDA dataset encompasses two domains (urban and rural), which brings considerable challenges due to the: 1) multi-scale objects; 2) complex background samples; and 3) inconsistent class distributions. The LoveDA dataset is suitable for both land-cover semantic segmentation and unsupervised domain adaptation (UDA) tasks [1].” [3]

LoveDA contains 5987 HSR images with 166768 annotated objects from Nanjing, Changzhou and Wuhan cities. There are 2713 images for urban scenes and 3274 images for rural scenes. The images are from 18 spatially independent areas, covering 18 administrative districts in different cities.

We use the Urban domain as Source and the Rural domain as Target. The public dataset is divided in three splits (Train, Validation and Test), of whom Test lacks ground truth masks. We thus decided to use the ‘Validation‘ split for testing (i.e. for evaluation), meaning that we had to split the training set to get some validation samples (we used a 8:2 ratio for that, and randomized picking through a seed). One may find the actual files here, referring to ”Train.zip” and ”Val.zip”.

The image resolution is 1024x1024 but we generally decided to use a resized version of 512x512 for memory constraints.

4.1.1 Source Domain

Our Source Domain is composed of 1156 annotated images, representing urban scenes from four different regions.

4.1.2 Target Domain

Our Target Domain is composed of 992 annotated images, representing rural scenes from two different regions. During Unsupervised Domain adaptation we used rural scenes from the Train dataset.

As we can see in Fig. 3 between the two domains there is a strongly imbalanced representation of certain categories, a shift in pixel intensity and pixel-wise size imbalance between classes. For example most of the buildings have relatively small scales in the rural areas, representing the “long tail” phenomenon. However, the buildings in the urban scenes have a larger size variance. This kind of imbalance is present even in other classes. This require the model to have multi-scale capture capabilities and robustness against pixel intensity shift. The inconsistent class distributions further increase the difficulty of generalization for a model.

4.1.3 Boundaries, Class Imbalance, and Other Issues

We developed our own dataloader instead of the one provided by the author, in which we included, together with the images and GT masks (in case of source domain), boundaries, extracted using cv2.Canny, like the authors of PIDNet do.

Statistics about the distribution of classes were extracted too, as the LoveDA author’s github wasn’t extensive enough about precise values. These distributions have been used in certain configurations to account for class imbalance during training.

The adversarial approach isn’t probably able to deal with such an issue by itself, so it may have proven useful to apply class weighting, but we only did it only for DACS’s Mix Loss.

For DACS, we hypotesized the i2i method to able to deal with class imbalance, at least a little, but to help it more, we tried the insertion of logic that favored Buildings and Roads being taken mostly from the source domain, in different ways, but none proved useful.

A minor issue is the unclassified class, for which the model is prevented to learn from but still makes prediction for. Since all unclassified pixels are black (all zeroes in RGB), a logic to prevent this can be added in the various models before the final head for correctness.

4.2. Evaluation Metrics

To evaluate the performance of the semantic segmentation models, the following metrics were used:

Mean Intersection over Union (mIoU): It measures the overlap between the predicted segmentation and the ground truth across all classes, providing a comprehensive view of model performance. Interpolation of the predicted mask to the original size (1024x1024) was done before calculating mIoU. Unclassified class ground truths were also ignored when considering union in the predicted mask.

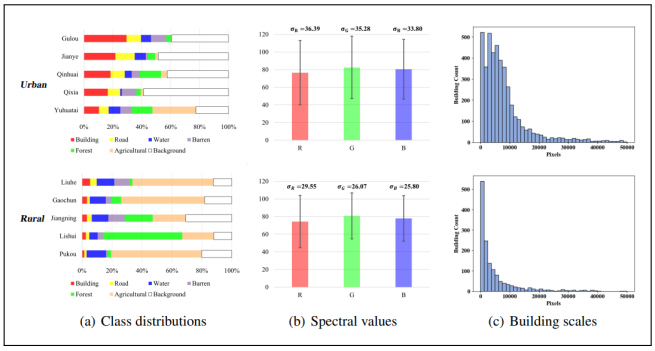

Figure 3. ”Statistics for the urban and rural scenes in a representative city” [3]

Latency: Latency measures the time taken for the model to predict a mask for an image.
FLOPs (Floating Point Operations): FLOPs indicate the computational complexity of the model.
Number of Parameters The number of parameters in the model provides insight into its size and complexity.

4.3. Data Augmentation

4.3.1 Base Techniques

Many augmentation techniques have been tried, including RandomFlip and Rotation, Jitter, GaussianBlur, Brightness, RandomCrops. Results can be better seen in Table 3.

Please observe the gap between domains too, as some augmentations prove beneficial only because they improve the general representation, while others are better at closing the gap between domains. Note that normalization, both using statistics for both domains and only for rural, was also attempted in previous iterations, but yielded suboptimal results to our surprise.

4.3.2 Neural Style Transfer

We decided to use a more refined approach to transfer the looks of Rural on Urban, and thus tried a Pytorch implementation of Neural Style Transfer thought for art styles [6]. Again, the results are shown on Table 3. This specific approach does not seem to work, probably because it is too aggressive, and the final images too blurry, as one may observe from figure 4. Since barren is the only class where we have an improvement, we may hypotesize that the style transfer may be finding style differences in that.

4.4. Models

4.4.1 DeepLabV2

For this model we didn't try an excessive amount of configurations, as it performed well quite easily. That is to say that one may easily improve the results we gathered, which we didn't because of computing limitations and focus on DA, that proved much less trivial than expected.

We started from a version of DeepLab pretrained on ImageNet, and used standard parameters for the SGD and for the step scheduler: LR = 1e-3, Momentum = 0.9, Weight Decay = 5e-5, no step down, BS = 24. The training is performed by a simple cross entropy which ignores the unclassified class.

The results have already been presented in the metrics paragraph, and can be found on Table 1.

One may find the source of the architecture that we used here.

4.4.2 PIDNet

Getting PIDNet to have decent performance has been quite straightforward too, and the settings are the same we used for DeepLabV2 for the comparison, while we later increased batch size and image size to 512x512, as this model is much more lightweight. Looking once more at Table 1 we may immediately spot a notable drop in performance on the lightweight model. The gap was closed quite easily by increasing image size to 512, as one can see from the first row of Table 3.

To improve results on Rural we implemented augmentations, some of which were chosen for trying to mimick the target domain's style: the results are really good as one can see from the table from before, but not necessarily thanks to the augmentations in which we put the most faith on (e.g. Style Transfer).

Although in the more recent versions of their implementation, the authors of PIDNet opt for the use of Online Hard Example Mining and class weighted Cross Entropy, we stick to basic Cross Entropy and Binary Cross Entropy, while keeping the weights of the different losses to 0.4 and 1 for segmentation losses l0 and l2 respectively, 20 for boundary loss l1, and 1 for the main segmentation loss l3, also called Boundary Awareness Loss (see figure 1).

One may find the source of the architecture that we used at the authors' original repo.

4.4.3 STDC

Since the main reason why we implemented this network was to understand better whether the lightweight nature of real-time networks was to blame or not for the poor results in DA, we kept the model simple and didn't bother too much to perfect its performance through hyperparameter tuning.

We'd like to jump to the conclusion of all this, which can be seen in Table 2. We may first of all see that the base gap on Rural for this model is much worse than PIDNet, so the comparison of the DA results will be quite difficult. When running DACS we see the usual small mIoU decrease, as we use EMA, which as we already stated seems to just be slowing down the worsening of the performance. When running Adversarial we again didn't find an improvement.

The model is pretrained by the authors, and we decided to use their best performing one in terms of mIoU, which is 'maxmIOU75'.

Following the authors' repo, we used an improved version compared to the paper, in which we compute not only the segmentation and detail loss, but also segmentation and detail losses on all stages (see figure 2). All losses are weighted the same, while SDG and scheduler settings are the usual. Here we trained on 512x512 images and used batch size of 32.

4.5. Domain Adaptation

4.5.1 Adversarial through AdaptSegNet - PIDNet

The adversarial approach proved to be quite underwhelming, as visible from Table 3. We tried many configurations but were unable to ever improve the results, so we'll just provide the results that we found were best, and which used this configuration: PIDNet LR = $1e-4$, Discriminator LR = $5e-4$ (no step down), Momentum = 0.9, Weight decay =$5e-5$, $\lambda_{\text{seg}} = 1$, $\lambda_{\text{adv}} = 1e-4$, $\lambda_{\text{disc}} = 0.1$, Base model = PIDNet pretrained on Jitter+RotateFlip,

As the discriminator was written from scratch, we decided to initialize its weights using Kaimig. Other initialization methods could have been explored, but we highly doubt the final results would have changed, as the discriminator seems to work properly: by plotting the losses in order to try and improve through more educated guesses, we found, as expected, competition between the discriminator and adversarial loss.

In general, we tried many different hyperparameter configurations and ablations: polynomial LR update, LR values tweaking, loss weights tweaking, switching to Adam optimizer, backpropagating in different stages, using spike and charge (i.e. instead of optimizing both the segmentation and discriminator models at once we do some rounds for each), using or not using a pre-fine-tuned model (i.e. the PIDNet or STDC model fine-tuned on LoveDA).

A more thorough exploration of the hyperparameter space would surely give more insight into what's going on with the poor performance, but our resources don't allow us to.

An important problem that we cited before too and we didn't address was class imbalance. By adding weights to the cross entropies we may have tried to learn better about classes that were rare in the source domain but common in the target one.

4.5.2 STDC on Adversarial

We finally ran STDC on this method too, and although the augmentations still hold rank one, this time the adversarial managed to beat STDC, while pushing Urban mIoU to a record for STDC (as if the Adversarial training was used just as extra epochs for the segmentation training), as visible in table 2.

The configuration for the STDC training on all these methods has been the same we used for the best DACS configuration. We also loaded the augmentation model (Jitter + RotateFlilp) as base model before doing the DA.

4.5.3 DACS

The image to image approach through DACS was more promising, but ended up not improving the Rural results yet again. The configurations tested here often went greatly beyond what the original paper proposed, as the associated repo has been updated and used methods proposed by a subsequent paper called DAFormer [7]. These additions have been: updating the pseudo mask predictor through ema, weighting pixels based on their confidence, through either hard or soft weighting (0/1 or gradual based on confidence threshold). Extra elements that we personally tried to add were class weighting and forcing to pick buildings and roads only from rural.

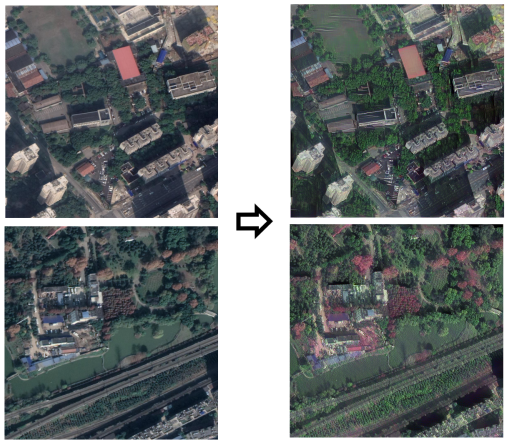

Figure 4. Style Transfer [6]

Table 1. LoveDA Urban Results (DeepLabV2 vs PIDNet - 20 epochs - Resize=256 and BatchSize=24)

Model	mIoU (%)	Latency (s)	FLOPs	Params
DeepLabV2	29.20%	0.0360	95.5×10^9	43.0×10^6
PIDNet	24.98%	0.0129	3.17×10^9	7.7×10^6

At Table 3 one may find the best results that we obtained, which once again, don't surpass the augmentations results. Although the results obtained through EMA seem to be promising, at least when compared with the other DA method, by seeing that mIoU seems to be going down through the epochs, we may hypothesize that we're just slowing down the degeneration of PIDNet thanks to the ema. It may also be interesting to note though that the loss of the mixed images still has a decreasing, although slowly depending on the instance, trend.

The configuration used for the best result was: using Jitter and RotateFlip augmentated PIDNet as base, Lambda Mixed = 18, LR = 1e-6 w/o step down, Confidence Pixel Weighting = Hard, Class Weighting only on Mixed Loss, using EMA with alpha = 0.99. In Table 3 one may also see how much worse results are without EMA, and how not using it doesn't end up with that steady worsening of results with increasing epochs.

Method	mIoU (%)	bg	building	road	water	barren	forest	agricultural	Urban Perf. (%)	diff
STDC - Base	15.43%	30.97%	20.80%	**14.02%**	21.90%	3.45%	11.25%	5.68%	29.23%	13.8%
STDC - Jitter + RotateFlip	**19.08%**	40.96%	**23.73%**	9.64%	**38.58%**	5.43%	10.97%	4.23%	27.30%	8.22%
STDC - Jitter + RotateFlip + Style Transfer	17.62%	40.70%	20.23%	7.91%	37.62%	3.53%	**12.52%**	0.8%	26.07%	8.45%
DACS	17.63%	41.16%	19.96%	9.55%	32.81%	6.34%	9.87%	3.77%	25.54%	**7.91%**
Adversarial	18.06 %	**47.43%**	21.41%	11.60%	27.78%	**9.86%**	13.89%	**6.93%**	**31.17%**	13.11%

Table 2. Comparison of STDC in versions: base, augmentaitons, DACS, and Adversarial Methods. Hyperparameters are the same as the best ones for the respective PIDNet applications

Augmentations	mIoU (%)	bg	building	road	water	barren	forest	agricultural	Urban Perf. (%)	diff w/Rural
None	21.89%	43.84%	33.54%	21.90%	21.88%	4.93%	12.64%	14.46%	31.19%	9.31%
Jitter	23.57%	44.70%	30.44%	24.07%	33.46%	**5.90%**	12.27%	14.20%	33.57%	9.99%
GaussianBlur	23.03%	35.96%	31.81%	22.72%	36.50%	3.60%	11.08%	19.54%	32.64%	9.61%
RandomFlip or Rotate	23.33%	47.09%	26.47%	24.22%	31.03%	5.83%	5.40%	23.24%	35.28%	11.95%
RandomCrop 900–600	25.99%	48.83%	24.89%	27.23%	35.08%	5.18%	6.76%	33.97%	**36.28%**	10.29%
Jitter + RandomFlipOrRotate (0.5)	**27.82%**	48.56%	34.09%	27.60%	**38.60%**	4.37%	**15.58%**	25.94%	34.42%	**6.60%**
Brightness (+/- 0.3)	23.82%	44.58%	30.90%	21.21%	27.45%	4.44%	12.26%	25.89%	31.35%	7.54%
Jitter + RandomFlipOrRotate + RandomCrop 900–600 (0.5, 0.5, 0.5)	**27.49%**	**50.45%**	**35.14%**	**28.10%**	30.83%	4.40%	5.75%	**37.97%**	35.76%	8.27%
Jitter + RandomFlipOrRotate + RandomCrop 900–600 (0.3, 0.3, 0.3)	24.33%	46.97%	20.07%	24.95%	34.20%	4.91%	13.19%	26.03%	34.11%	9.78%
Jitter + RandomCrop 900–600	25.71%	44.77%	25.44%	26.79%	38.32%	4.43%	13.98%	26.20%	33.91%	8.20%
Style Transfer	19.41%	42.54%	15.24%	18.85%	32.86%	**10.98%**	7.43%	14.49%	30.07%	10.66%
Style Transfer + Jitter	18.40%	39.81%	21.52%	18.60%	30.62%	3.89%	6.98%	7.40%	27.79%	9.38%
Style Transfer + Jitter + RandomFlipOrRotate (0.5, 0.5)	21.03%	41.44%	21.43%	24.87%	33.13%	4.26%	13.32%	8.77%	27.99%	6.96%
DACS w/EMA @ epoch 20	25.13%	46.85%	27.38%	24.46%	35.85%	5.62%	15.35%	20.40%	30.86%	5.73
DACS w/EMA @ epoch 11	25.92%	48.01%	30.72%	25.07%	34.09%	5.07%	15.49%	22.99%	31.71%	5.79
DACS w/o EMA @ epoch 20	21.50%	46.66%	26.66%	20.40%	29.69%	4.53%	11.26%	11.33%	28.09%	6.59
DACS w/o EMA @ epoch 15	21.77%	45.73%	23.20%	22.03%	31.31%	4.59%	12.71%	12.87%	28.16%	6.39
Adversarial with PIDNet	23.87%	46.48%	21.04%	17.01%	42.82%	11.25%	16.13%	12.36%	**40.20%**	16.33%

Table 3. Results of various augmentations for PIDNet over 20 epochs. Check the respective paragraphs in the Experiments section for details about hyperparameters and configuration. Note that images are resized to 512x512 for lighter training, but are upscaled again to 1024x1024 when calculating mIoU.

4.5.4 STDC on DACS

We finally ran STDC here too (see Table 2), and saw no improvement with respect to the augmentation results. The configuration for the STDC training on all these methods has been the same we used for the best DACS configuration. We also loaded the augmentation model (Jitter + RotateFlilp) as base model before doing the DA.

5. Conclusions

The domain gap is quite evident, and fixing it through augmentations has generated some results, but DA techniques really didn't. We hypotesized the causes could be either: ineffectiveness of the DA methods on smaller, real-time models; dataset structure problems, as in, the kind of gap between domains isn't really solvable through the analyzed techniques; the hyperparameters and model implementation is not optimal.

For the first, when trying another real-time model (STDC instead of PIDNet), Adversarial was finally able to beat the augmentation on Rural, but Urban grew too, by a lot, so this probably doesn't count as successful Domain Adaptation.

For the second one, we found that there actually was quite the class imbalance, but there were also issues with poor quality of GT labeling, especially when deciding between background, barren, and forest. When trying to address class imbalance on DACS, only really small improvements have been met (less than .5%), but that may prove a good starting point.

For the third one, we describe the configurations tested throughout the experiments paragraph. A relatively good amount of resources have been spent for exploring as many configurations as possible in DACS and AdaptSegnet, but for sure this can be improved further through grid search for example.

In a possible future work we would like to spend more time on alternative real-time model, in order to better assess the cause for these suboptimal results, and put more effort towards finding a better way to deal with the class imbalance problem.

References

[1] Liang-Chieh Chen, George Papandreou, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *TPAMI*, 2018. 1

[2] Xu Jia Cong et al. Pidnet: A real-time semantic segmentation network inspired by pid controllers, 2021. Available as PDF. 1, 2

[3] LoveDA Dataset. Loveda: A remote sensing land-cover dataset for domain adaptive semantic segmentation, 2021. Available as PDF. 1, 5

[4] Mingyuan Fan, Shenqi Lai, Junshi Huang, Xiaoming Wei, Zhenhua Chai, Junfeng Luo, and Xiaolin Wei Meituan. Rethinking bisenet for real-time semantic segmentation. *arXiv preprint arXiv:2104.13188*, 2021. 1

[5] Shijie Hao, Yuan Zhou, and Yanrong Guo. A brief survey on semantic segmentation with deep learning, 2021. Available as PDF. 1

[6] Matthias Bethge Leon A. Gatys, Alexander S. Ecker. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576v2*, 2015. 6, 7

[7] Luc Van Gool Lukas Hoyer, Dengxin Dai. Daformer: Improving network architectures and training strategies for domain-adaptive semantic segmentation. *arXiv preprint arXiv:2111.14887*, 2021. 7

[8] Wilhelm Tranheden, Viktor Olsson, Juliano Pinto, and Lennart Svensson. Dacs: Domain adaptation via cross-domain mixed sampling. *arXiv preprint arXiv:2104.06478*, 2021. 2, 4

[9] Yi-Hsuan Tsai, Wei-Chih Hung, Samuel Schulter, Kihyuk Sohn, Ming-Hsuan Yang, and Manmohan Chandraker. Learning to adapt structured output space for semantic segmentation. In *CVPR*, 2018. 2, 4

[10] Juliano Pinto Lennart Svensson Viktor Olsson, Wilhelm Tranheden. Classmix: Segmentation-based data augmentation for semi-supervised learning. *arXiv preprint arXiv:2007.07936*, 2020. 2, 4

[11] Changqian Yu, Jingbo Wang, Chao Peng, Changxin Gao, Gang Yu, and Nong Sang. Bisenet: Bilateral segmentation network for real-time semantic segmentation. *ECCV*, 2018. 1

[12] Sicheng Zhao, Xiangyu Yue, Shanghang Zhang, Bo Li, Han Zhao, Bichen Wu, Ravi Krishna, Joseph E. Gonzalez, Alberto L. Sangiovanni-Vincentelli, Sanjit A. Seshia, and Kurt Keutzer. A review of single-source deep unsupervised visual domain adaptation. *arXiv preprint arXiv:2007.08702*, 2020. 1