Increasing CNN generalization and robustness by visual domain adaptation

Master Thesis
Master thesis in the field of study “Computational Engineering” by Evgeny Gorelik
(Student ID: 2666468)
Date of submission: 1.8.2022

1. Review: Stefan Roth
2. Review: Mateo Castrillon
Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT

field of study:
Computational Engineering
VISINF
VISINF Lab

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB TU Darmstadt

Hiermit versichere ich, Evgeny Gorelik, die vorliegende Masterarbeit gemäß § 22 Abs. 7 APB der TU Darmstadt ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 1.8.2022

E. Gorelik

E. Gorelik

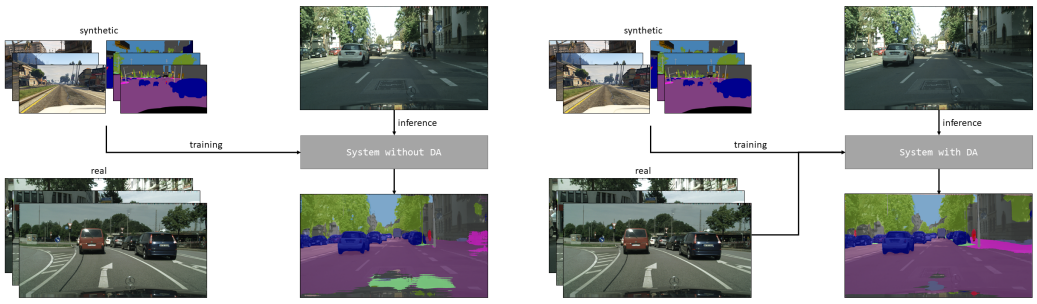
Contents

Abstract

This work proposes a method for *domain adaptation* in *semantic segmentation* by leveraging the predictions of *self-supervised learning* methods for alignment of class representations in feature space. Based on the simple idea of *class-sensitive cropping*, this work aims to improve the class-discriminative abilities of semantic segmentation networks on a domain with unlabeled data, by using knowledge from another similar domain with labeled data. The key idea of the proposed method is the alignment of crops of the same class from two different domains in feature space, while simultaneously increasing the distance of feature representations between crops from different classes.
The method proposed in this work combines approaches from *self-supervised learning* and from *contrastive learning* to achieve alignment of different classes in feature space. With an mIoU of 49.8% on the *GTA→Cityscapes* benchmark for autonomously driving vehicle training, the resulting domain adaptation framework titled Feature Representation Learning (FRL) achieves results, comparable to state-of-the-art domain adaptation frameworks.

1. Introduction

In the past decade, deep learning based systems have shown great success for a huge number of real-world scenarios. With the increase in size of those systems, the need for data, required for training, has also increased drastically. Manual labeling this data is very labour-intensive, especially in the case of *semantic segmentation*. Therefore, research has shifted from the approach of labeling every new dataset by hand to reusing labeled datasets in combintation with unlabeled datasets.
The goal of this newly arised field of *transfer learning* is to transfer knowledge gained from training on datasets with annotations to another similar dataset without annotations. Findings show, that without proper adaptation the trained networks perform poorly, especially if both datasets differ in visual appearance. This difference can be caused by numerous factors, *e.g.* the images were taken using different cameras, during different times of the day or at different locations. In a more drastic setting, training data can come from a simulated environment, while the system is later deployed in the real world. This adaptation setting is called *sim2real*.



(a) Without domain adaptation (b) With domain adaptation

Figure 1.1. Training and Inference difference between methods with and without domain adaptation

This work deals with the problem of sim2real adaptation, with focus on the task of domain adaptation in semantic segmentation of images. Semantic segmentation is used in many applications, ranging from medical imaging to embedding in autonomously driving vehicles. This work will focus on the sim2real setting, in which a neural network is trained on synthetic images of a simulated environment from the perspective of a camera, mounted to a vehicle. The objective of semantic segmentation is to train a neural network, which is tasked with the prediction of a class (*e.g. 'road', 'sidewalk', 'building', etc...*) for every pixel in an image. The network is then adapted to increase its semantic segmentation accuracy on real-world images, recorded from a similar camera perspective. Since the synthetic dataset and the real-world dataset differ visually, this task is categorized as a domain adaptation. The training scheme for networks with and without domain adaptation is shown in Figure 1.1. In contrast to training schemes without domain adaptation, training with domain adaptation uses the real-world data in the training process.
With increasing difference between the visual appearance of categories (*e.g. 'road', 'car'*) in the datasets, the task of domain adaptation becomes more difficult. This leads to a notable discrepancy in performance between system with and without adaptation, where methods without domain adaptation fail to produce good results.
This thesis gives an overview of the research in the field of domain adaptation, discusses weaknesses of existing approaches and provides a new method for improving existing state-of-the-art methods.
The contributions of this work are the following:

1. Exploration of feature representation alignment approaches in combination with self-supervised learning
2. Extension of the domain adaptation framework Self-Supervised Augmentation Consistency (SAC) [5], that uses self-supervised learning, by introducing *feature representation learning* on class-sensitive crops
3. Development of two methods for creating class-sensitive crops by using:
    a) Convolution based label density estimation
    b) Connected Component based instance detection
4. Novel approach for approximating visual similarity between classes

With the introduction of class-sensitive cropping, this work bridges the gap between domain adaptation methods for semantic segmentation and *image classification*, two fields of research, with little common methodology so far. Therefore, this work incentivises followup research into the utilization of other domain adaptation techniques for image

classification in domain adaptation for semantic segmentation.
For a basic understanding of neural networks, optimization and domain adaptation, chapter 2 provides a sparse introduction to those topics. Thereafter, this work is divided into three main parts. In chapter 3 an overview of the current state of research on topics relevant for this thesis is provided. In chapter 4 a novel domain adaptation method for semantic segmentation is presented and described in detail. The experimental details of the method proposed in chapter 4 are presented in chapter 5, followed by an evaluation of the method on state-of-the-art level benchmarks in section 5.2. Finally, a discussion of the results concludes this thesis in chapter 6.

2. Basics

This chapter provides some basic background, required for the understanding of the topics of semantic segmentation and domain adaptation. Since a full scale introduction to deep learning, including all subcategories would exceed the scope of this thesis, the reader is encouraged to read introductory literature if new to the topic [55, 8]. Therefore, this thesis covers only the basic topics, relevant for the understanding of the domain adaptation method proposed in chapter 4.

2.1. Mathematical Notations

In the course of this thesis classical mathematical notations will be used as follows:

- Vectors $\boldsymbol{x} \in \mathbb{R}^n$ are denoted as bold lower case letters and are column vectors per default
- Matrices $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ are denoted as bold upper case letters where $\boldsymbol{A}$ has m rows and n columns
- Scalars s or S are denoted as cursive letters
- $\boldsymbol{x}^T\boldsymbol{y}$ is the dot product between $\boldsymbol{x} \in \mathbb{R}^n$ transposed and $\boldsymbol{y} \in \mathbb{R}^n$
- $\boldsymbol{A}\boldsymbol{x}$ is the matrix vector multiplication between $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{A} \in \mathbb{R}^{m \times n}$
- $||\boldsymbol{x}||$ denotes the euclidean norm of $\boldsymbol{x}$
- $f(\boldsymbol{x})$ denotes a function f applied to $\boldsymbol{x}$
- $f_\theta(\boldsymbol{x})$ denotes a function f, which is parameterized by θ applied to $\boldsymbol{x}$
- $f(\boldsymbol{x}; \theta)$ implies, that the output of the function f for the input $\boldsymbol{x}$ is dependent on θ (Read as: *"f of x given theta"*). Note, that $f_\theta(\boldsymbol{x})$ implies $f(\boldsymbol{x}; \theta)$.

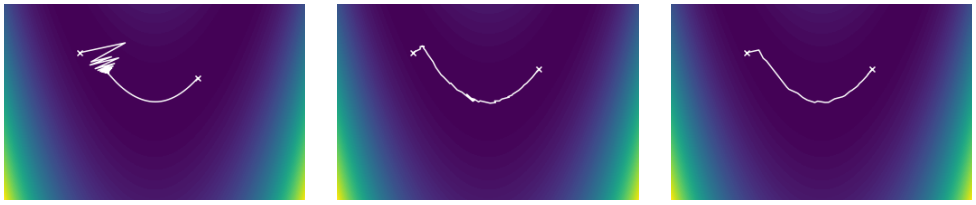
- $p(x|y)$ denotes the conditional probability of x given y

2.2. Optimization

In *optimization*, the goal is to find an optimal state $\theta^\star$ of a system $g_\theta(\cdot)$, so it best approximates a functional mapping $g(\cdot)$. The input and output of g and h can be single valued (*e.g.* in tasks of polynomial approximation or linear regression) or highly multi-dimensional matrices, representing audible or visual content (*e.g.* in tasks of image recognition, object detection or semantic segmentation). There are two possibilities for obtaining an optimal solution:

1. Obtain the solution $\theta^\star$ by analytical solving of the objective function g_θ with regards to θ on a given input $\boldsymbol{x}$
$$g_\theta(\boldsymbol{x}) = h(\boldsymbol{x}) \tag{2.1}$$

2. Obtain the solution $\theta^\star$ by iterative improvements to an initial possibly sub-optimal solution. Here the approaches differ in efficiency, depending on the knowledge of the function g_θ.
If *gradients* are not computable, g_θ is viewed as a black-box and gradient-free optimization approaches (*e.g.* Nelder-Mead Simplex [36], Multi-Directional Search [57]) are required.
If *gradients* are computable, gradient based algorithms (*e.g.* gradient descent, conjugate gradient descent, quasi-newton method, BFGS method) can be used, which exceed the performance of gradient-free methods with regards to computation cost and convergence rate.



(a) Gradient Descent (b) Nelder-Mead (c) Conjugate Gradient

Figure 2.1. Optimization steps of three methods on Rosenbrook function $100 \cdot (p_1 - p_0^2)^2 + (1 - p_0)^2$ with global optimum $p^\star = (1, 1)$

In most cases, Equation 2.1 cannot be solved analytically. In those cases optimization is achieved by iterative (so-called *numerical*) optimization. The main objective of numerical optimization is to find a set of parameters $\theta^\star$ with:

$$\theta^\star = \arg\min_\theta \mathcal{L}(\theta; \mathcal{X}) \tag{2.2}$$

with the *loss function*:

$$\mathcal{L}(\theta; \mathcal{X}) = \sum_{x \in \mathcal{X}} d(g_\theta(\boldsymbol{x}), h(\boldsymbol{x})) \tag{2.3}$$

where $\mathcal{X}$ is the available dataset, containing vectors $\boldsymbol{x}$, for which $h(\boldsymbol{x})$ is either known or computable and $d(\cdot, \cdot)$ is a distance function. Commonly used distance functions are L1 or L2 norms, or any type of symmetric and positive semidefinite functions, for which the *triangle inequality* holds.

For finding the optimal parameterization $\theta^\star$, all numerical optimization methods start with a random guess or a result of previous optimization procedures $\theta^{(0)}$. For neural networks this is called *pretraining*. The guess $\theta^{(i)}$ is iteratively updated by shifting the parameters θ towards a more optimal solution:

$$\theta^{(i+1)} = \theta^{(i)} - \mu \nabla \mathcal{L}(\theta^{(i)}, \mathcal{X}) \tag{2.4}$$

with $\nabla \mathcal{L}(\theta, \mathcal{X})$ being the gradient of the loss function from Equation 2.3 and μ being the step size (*learning rate* in machine learning context) in the opposite direction of the gradient.
By repeatedly applying Equation 2.4, the output of g_θ converges to the output of h. Thus, the parameters θ gradually converge towards $\theta^\star$. Though Equation 2.1 is formally may not achieved, the application of numerical optimization leads to the relaxed formulation

$$g_\theta(\boldsymbol{x}) \approx h(\boldsymbol{x}) \tag{2.5}$$

with varying degrees of approximation quality for different tasks and applications.

2.3. Neural Networks

The functions discussed in section 2.2 were regarded as generic functions with real-valued, possibly high dimensional input $\boldsymbol{x}$, and real-valued, possibly high dimensional output $\boldsymbol{y}$. As universal function approximators [24], neural networks are suited for a wide variety of tasks, commonly subdivided into regression and classification. Originally, *fully connected* neural networks consist of recursive repetition of two operations:

1. Linear multiplication with weights of the network with addition of a bias term:

$$\boldsymbol{o}^{(k)}(\boldsymbol{x}) = \boldsymbol{W}^{(k)} \cdot \boldsymbol{x} + \boldsymbol{b}^{(k)} \tag{2.6}$$

2. *Activation functions* $\sigma(\boldsymbol{x})$ as non-linear parts of the neural networks (*e.g. sigmoid, ReLU*)

As neural networks are a wide-ranging topic, giving an in-depth introduction to neural networks goes beyond the scope of this thesis. If new to the topic, the reader is encouraged to read follow-up literature [9].
Mainly, this work will focus on a subtype of neural networks, namely the Convolutional Neural Network (CNN) [31]. CNNs are the golden standard of network architectures for visual recognition tasks, due to the low number of internal parameters (referred to as *weights*), compared to fully connected networks, and good results on recognition tasks in real-world applications.

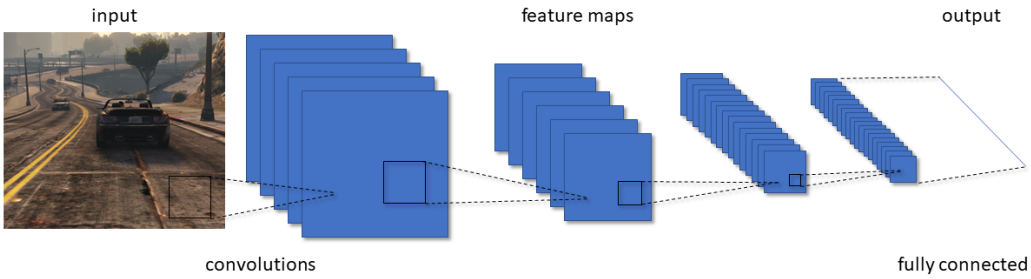
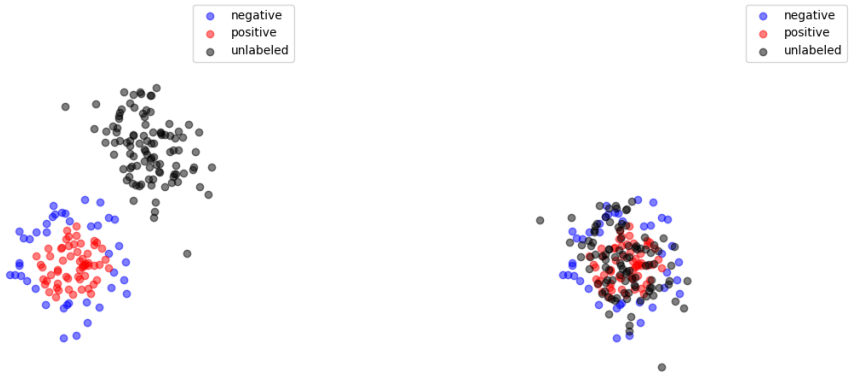


Figure 2.2. Convolutional Neural Network (CNN) architecture (based on [66])

Just like fully connected neural networks, CNNs work by recurrently feeding the output of layers to the input of successive layers. Instead of linear multiplications, as defined in Equation 2.6, CNNs compute convolutions between input matrices $\boldsymbol{X}$ and weight matrices $\boldsymbol{W}^{(k)}$.
A generic CNN architecture is depicted in Figure 2.2. As seen in the visualization, the number of layers increases throughout the network, while their individual height and width decrease. The output of those convolutions is called a *feature map*. These feature maps will play an important role in the method proposed in chapter 4.

2.4. Domain Shift from a Geometric Perspective

In this section a more intuitive perspective on the concept of *domain shift* is presented. Though not directly applicable to real world applications, a geometric view on the problem may aid the understanding of methods discussed in the following chapters.
Consider the following example:
A labeled dataset $\mathcal{X}_S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$ and an unlabeled dataset $\mathcal{X}_T = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_M\}$ are given with the goal to classify the points in $\mathcal{X}_T$. Furthermore, $\mathcal{X}_S$ and $\mathcal{X}_T$ are assumed to be dividable into the binary classes *'positive'* and *'negative'*. Therefore, the label information of $\mathcal{X}_S$ can be used to find the most probable labels for $\mathcal{X}_T$.



(a) Initial State (b) Final State

Figure 2.3. Domain Adaptation by Mean Shift (Example 1)

Figure 2.3a visualizes a possible distribution of $\mathcal{X}_S$ and $\mathcal{X}_T$ in two dimensional space. As seen in the image, the labels y are categorized as *'positive'* around the center of all samples in $\mathcal{X}_S$ and *'negative'* everywhere else. The resulting classification boundary for y would be a circle around the center of $\mathcal{X}_S$. However, when applying the same boundary to $\mathcal{X}_T$, all points would be classified as negative.
Although this could be a plausible outcome, in the setting of domain adaptation, the internal structure of the labeled *source dataset* and the unlabeled *target dataset* are assumed to be similar. This renders the probability of all points in $\mathcal{X}_T$ being assigned the label *'negative'* very unlikely. A visual comparison between $\mathcal{X}_S$ and $\mathcal{X}_T$ indicates, that the points are sampled from a very similar Gaussian distribution with a *shifted* mean.

The canonical method for overlaying $\mathcal{X}_\mathcal{T}$ over $\mathcal{X}_\mathcal{S}$ would be to subtract the target data mean $\boldsymbol{\mu}_\mathcal{T}$ and add the source data mean $\boldsymbol{\mu}_\mathcal{S}$ to every data point $\boldsymbol{z}$.

Algorithm 1: Domain Adaptation by Mean Shift

input : Source data $\mathcal{X}_\mathcal{S} = \{(\boldsymbol{x}, y)_i\}$
Target data $\mathcal{X}_\mathcal{T} = \{\boldsymbol{z}_i\}$
output : Transformed data $\{\overline{\boldsymbol{z}}_i\}$

$\mu_\mathcal{S} \leftarrow mean(\{\boldsymbol{x}_i\})$
$\mu_\mathcal{T} \leftarrow mean(\{\boldsymbol{z}_i\})$
for $i = 1, \ldots, M$ **do**
  $\overline{\boldsymbol{z}}_i \leftarrow \boldsymbol{z}_i - \mu_\mathcal{T} + \mu_\mathcal{S}$
end

A pseudo-code implementation, which utilizes the idea of domain adaptation by mean shift is presented in algorithm 1. The resulting distribution of target data after the transformation is depicted in Figure 2.3b. After the target data is shifted, the classification boundaries of the source data can be applied to it. The classification results of the target data using algorithm 1 are more likely.
Although algorithm 1 manages to align the means of the two distributions, it only works reliably, if both distributions have the same *covariance matrix*.

(a) Initial State (b) State after Mean Shift

Figure 2.4. Domain Adaptation by Mean Shift (Example 2)

An example, in which the covariance matrices $\boldsymbol{\Sigma}_\mathcal{S}$ and $\boldsymbol{\Sigma}_\mathcal{T}$ are not identical is depicted

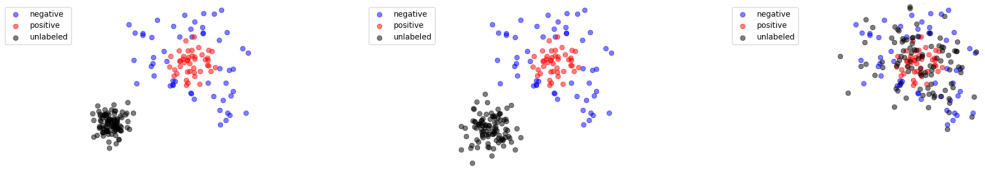
in Figure 2.4a. After applying algorithm 1 to the target data in Figure 2.4a, the target distribution is located within the classification boundary for the label *'positive'* (seen in Figure 2.4b). Using this transformation, almost all target data points would be labeled as *'positive'*.
Since, in domain adaptation this is assumed to be unlikely, the covariance matrix of the source data has to be transferred to the target data.

Algorithm 2: Domain Adaptation by Principal Component Alignment

input : Source data $\mathcal{X}_\mathcal{S} = \{(\boldsymbol{x}, y)_i\}$
Target data $\mathcal{X}_\mathcal{T} = \{\boldsymbol{z}_i\}$
output : Transformed data $\{\overline{\boldsymbol{z}}_i\}$

1 $\boldsymbol{\mu}_\mathcal{S} \leftarrow mean(\{\boldsymbol{x}_i\})$
2 $\boldsymbol{\mu}_\mathcal{T} \leftarrow mean(\{\boldsymbol{z}_i\})$
3 $\boldsymbol{\Sigma}_\mathcal{S} \leftarrow cov(\{\boldsymbol{x}_i\})$
4 $\boldsymbol{\Sigma}_\mathcal{T} \leftarrow cov(\{\boldsymbol{z}_i\})$
5 $\boldsymbol{W}_\mathcal{S}, V_\mathcal{S} \leftarrow eig(\Sigma_\mathcal{S})$
6 $\boldsymbol{D}_\mathcal{S} \leftarrow diag(\boldsymbol{W}_\mathcal{S}^{-\frac{1}{2}})$
7 $\boldsymbol{W}_\mathcal{T}, \boldsymbol{V}_\mathcal{T} \leftarrow eig(\boldsymbol{\Sigma}_\mathcal{T})$
8 $\boldsymbol{D}_\mathcal{T} \leftarrow diag(\boldsymbol{W}_\mathcal{T}^{-\frac{1}{2}})$
9 **for** $i = 1, \ldots, M$ **do**
10 $\quad \overline{\boldsymbol{z}}_i \leftarrow \boldsymbol{z}_i - \boldsymbol{\mu}_\mathcal{T}$
11 $\quad \overline{\boldsymbol{z}}_i \leftarrow D_\mathcal{T} V_\mathcal{T}^T \overline{\boldsymbol{z}}_i$
12 $\quad \overline{\boldsymbol{z}}_i \leftarrow (D_\mathcal{S} V_\mathcal{S}^T)^{-1} \overline{\boldsymbol{z}}_i$
13 $\quad \overline{\boldsymbol{z}}_i \leftarrow \overline{\boldsymbol{z}_i} + \boldsymbol{\mu}_\mathcal{S}$
14 **end**

A pseudo-code implementation for the transfer of a covariance matrix for a distribution onto another distribution is presented in algorithm 2. In the algorithm the eigenvalue decomposition $eig(\cdot)$ of both covariance matrices provides the eigenvectors $\boldsymbol{V}_\mathcal{S}$ and $\boldsymbol{V}_\mathcal{T}$ with their eigenvalues $\boldsymbol{W}_\mathcal{S}$ and $\boldsymbol{W}_\mathcal{T}$ respectively. The inverse square roots of the eigenvalues are then written into diagonal matrices $\boldsymbol{D}_\mathcal{S}$ and $\boldsymbol{D}_\mathcal{T}$ in lines 6 and 8.



(a) Initial State (b) State after whitening (c) State after PC alignment

Figure 2.5. Domain Adaptation by Principal Component Alignment (Example 2)

Figure 2.5c depicts the result, after applying algorithm 2. After transforming the target data, $\mathcal{X}_S$ and $\mathcal{X}_T$ are identically distributed. For a better understanding of the way the algorithm 2 works, Figure 2.5b shows the results for every point z after line 11 in the algorithm. After line 11 the target data is essentially normally distributed with mean 0 and a variance of 1. This step is called data *whitening*. After the target data is whitened, the covariance matrix of the source data can be applied to it in line 12 of the algorithm. Once the data is identically distributed, the classification boundaries of the source data will most likely be correct for the target data too. This simplified example represents an approach for domain adaptation, however, in real-world scenarios, the data distribution is rarely fully captured by a unimodal Gaussian distribution.

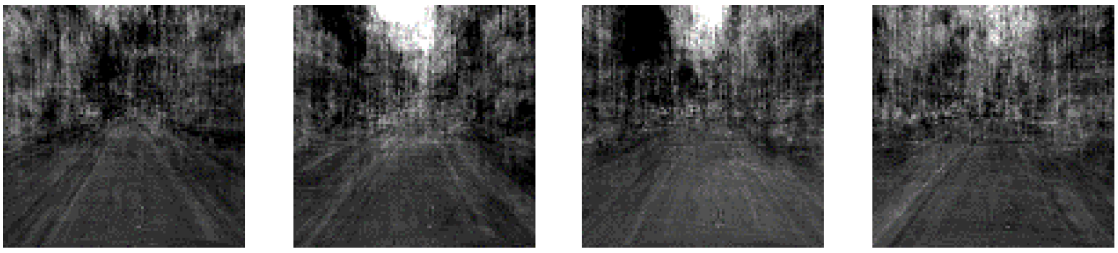


Figure 2.6. Samples from Gaussian Approximation of Cityscapes dataset [14]

As a demonstration, Figure 2.6 shows some samples of a unimodal Gaussian approximation of the Cityscapes dataset [14], which consists of dash-cam recordings in the real world. Though some structural shapes can be seen in the images (*e.g. 'sky', 'road', 'tree'*), the appearance of those shapes lacks fine detail, as it is impossible to capture nuances using a unimodal Gaussian distribution.
Real examples of domain adaptation are rarely as simple as aligning two dimensional

unimodal Gaussian distributions. Therefore, this section is only meant to give the reader an intuitive understanding, which problems can occur in domain adaptation and how to intuitively describe a solution. The proposed method in chapter 4 applies some ideas, described in this section to high dimensional vector spaces for alignment of distributions of so-called *feature representations*.

3. Related Work

This chapter gives an overview of previous works, related to the topics of this thesis. The chapter is divided into four parts, in which first the topics of domain adaptation and semantic segmentation are covered separately, then methods for the combined task of domain adaptation for semantic segmentation are reviewed. Lastly, an overview on the topic of contrastive learning is provided.

3.1. Domain Adaptation

Domain adaptation is a field of research, which promises good performance of a discriminative model on *unlabeled* data, by using knowledge, inferred from a similar *labeled*[1] dataset. The data distribution of the *source domain*, for which labels are available, presumably differs from the distribution of data in the *target domain*. The two distributions have to be related in some form. In use cases, the target and the source data samples share the number of dimensions (*e.g.* the same vector space $\mathbb{R}^{C\times H\times W}$) and there exists a categorical mapping of the content of the source domain to the content of the target domain (*e.g.* the datasets share class labels).
There are three possibilities of how the probability distribution between source and target data can relate. In the case of *prior shift* [30], the probability $p(y)$ of a label differs between source and target domain, and the conditional probability $p(\boldsymbol{x}|y)$ of a sample $\boldsymbol{x}$ given a class y stays the same.

$$\boldsymbol{x} \in X, y \in Y : \begin{array}{l} p_{\mathcal{S}}(y) \neq p_{\mathcal{T}}(y) \\ p_{\mathcal{S}}(\boldsymbol{x}|y) = p_{\mathcal{T}}(\boldsymbol{x}|y) \end{array} \tag{3.1}$$

In other words, prior shift occurs if the relative count of the label y differs between the target and the source domain (*e.g. "more bicycles are seen in source then in target data"*).

[1] In literature, *labeled* data and *annotated* data are equivalent.

In the case of *covariate shift*[30] the prior $p(\boldsymbol{x})$ of a sample $\boldsymbol{x}$ differs and the probability $p(y|\boldsymbol{x})$ of the label y given a sample $\boldsymbol{x}$ stays the same.

$$\boldsymbol{x} \in X, y \in Y: \begin{array}{l} p_{\mathcal{S}}(\boldsymbol{x}) \neq p_{\mathcal{T}}(\boldsymbol{x}) \\ p_{\mathcal{S}}(y|\boldsymbol{x}) = p_{\mathcal{T}}(y|\boldsymbol{x}) \end{array} \tag{3.2}$$

Essentially, covariate shift states that a data point from the source dataset is unlikely to be sampled under the distribution of the target dataset (*e.g. "an image from the Cityscapes dataset is unlikely under the distribution of the GTA dataset"*).
Lastly, in the case of *concept shift*[30], the data distributions $p_{\mathcal{S}}(\boldsymbol{x})$ and $p_{\mathcal{T}}(\boldsymbol{x})$ stay the same, but the conditional probabilities $p_{\mathcal{S}}(y|\boldsymbol{x})$ and $p_{\mathcal{T}}(y|\boldsymbol{x})$ differ.

$$\boldsymbol{x} \in X, y \in Y: \begin{array}{l} p_{\mathcal{S}}(\boldsymbol{x}) = p_{\mathcal{T}}(\boldsymbol{x}) \\ p_{\mathcal{S}}(y|\boldsymbol{x}) \neq p_{\mathcal{T}}(y|\boldsymbol{x}) \end{array} \tag{3.3}$$

Concept shift is inherently difficult, since it implies that the identical image could be labeled differently in source and target domain. This renders unsupervised domain adaptation without any annotations in the target domain very ineffective, since the labeling of data changes from source to target domain.
The methodology, discussed in this thesis assumes covariate shift between the data of source and target domain, as it is most common. In section A.2 the topic of prior shift will be explored on the datasets, used for evaluation in this thesis.
Domain adaptation also differs in complexity with regards to the set of labels $\mathcal{K}$ for source and target data. Here, it is possible to subdivide into two types of adaptation [19]:

***Closed set* domain adaptation** The mapping of labels from source to target domain is bijective, so every class in the source domain has a respective class in the target domain ($\mathcal{K}_{\mathcal{S}} = \mathcal{K}_{\mathcal{T}}$).

***Open set* domain adaptation** The source and target domain share a set of labels, but also have a set of private labels. The mapping of labels between source and target domain is then injective. In other words, the set of labels in the source data is assumed to be a subset of labels in the target data ($\mathcal{K}_{\mathcal{S}} \subset \mathcal{K}_{\mathcal{T}}$).

The problem of open set domain adaptation is inherently more complicated than its closed set counterpart. Research in this area is sparse. The most common solutions classify, whether the class is known or unknown [40, 50]. In semantic segmentation (described in section 3.2), open set domain adaptation solutions have also been recently proposed, which either use an adaptive threshold on prediction confidence [15], or extend the label space by using a simplex-based approach [22]. For completeness, the domain adaptation

settings *partial* domain adaptation, *open-partial* domain adaptation and *boundless* domain adaptation [56] are mentioned but not further discussed. This work will also focus on closed set domain adaptation. For the following sections, the equality between the source label set $\mathcal{K}_{\mathcal{S}}$ and the target label set $\mathcal{K}_{\mathcal{T}}$ is assumed.
In general, domain adaptation is applicable, when the distribution of target data can be transferred into the distribution of source data. This requires the internal structure of the distributions for every class to be similar in some sense. The goal of domain adaptation is to infer information, acquired from learning on the source domain to the target domain.

3.2. Semantic Segmentation

As the main focus of this thesis, the topic of *semantic segmentation* is presented in this section. In the classical setting of semantic segmentation a set of images $\mathcal{X} = \{X_1, \dots X_N \in \mathbb{R}^{C \times H \times W}\}$ is given, where N is the number of images, C is the number of channels per image (*e.g.* three channels for RGB images) and H and W are the height and width of the images. Note, that the image size is fixed for all images in the set and images may have to be resized accordingly.

(a) Image Input (b) ResNet [23] Output (c) YOLO [44] Output (d) DeepLab Output

Figure 3.1. Inference Tasks on Visual Input

With regards to task complexity, the task of semantic segmentation (Figure 3.1d) exceeds the complexity of image recognition (Figure 3.1b) and object detection (Figure 3.1c). Notably, scene understanding and object detection can mostly be inferred from semantic segmentation[2].
In semantic segmentation, the goal is to determine a pixel-wise prediction of labels $\boldsymbol{Y}^{H \times W}$, where each entry represents the according class for the pixel, for every image $\boldsymbol{X}^{C \times H \times W} \in \mathcal{X}$.

[2] Object detection requires instance recognition which, combined with semantic segmentation is the task of panoptic segmentation [29].

(a) Image (b) Label

Figure 3.2. Example from GTA [47] dataset

Figure 3.2 shows an example of an image with its according label map from the GTA [47] dataset. In a classical semantic segmentation setting, these labels are mostly given. The training objective for a network $g(\boldsymbol{X}) \rightarrow \boldsymbol{Y}$ is to correctly predict the class of each pixel in the image. For the simplified objective of image classification, the goal is to predict a class for the whole image by passing $\boldsymbol{X}$ through a network $g(\cdot)$:

$$\boldsymbol{X} \in \mathbb{R}^{C \times H \times W}, \boldsymbol{y} \in \mathbb{R}^{K} : g(\boldsymbol{X}) \rightarrow \boldsymbol{y} \tag{3.4}$$

with K being the number of classes. The output of the network f is a score vector $\boldsymbol{y}$, which can be transformed into a probability distribution over the classes by applying the *softmax* [9, p.236] function:

$$\sigma(\boldsymbol{y})_i = \frac{\exp(y_i)}{\sum_{j=0}^{K} \exp(y_j)} \tag{3.5}$$

The prediction for image $\boldsymbol{X}$ as produced by the network f has the class index $k = \arg\max \boldsymbol{y}$ with the prediction certainty $\max \sigma(\boldsymbol{y})$.
In semantic segmentation the problem is extended, so each pixel has a score per class. This results in the following output of network f:

$$\boldsymbol{X} \in \mathbb{R}^{C \times H \times W}, \boldsymbol{Y} \in \mathbb{R}^{K \times H \times W} : f(\boldsymbol{X}) \rightarrow \boldsymbol{Y} \tag{3.6}$$

with the label prediction being the $\arg\max \boldsymbol{Y}$ along the first dimension.
For solving the task of pixel-wise label prediction (referred to as semantic segmentation), a set of architectures have been introduced, most prominently Fully Convolutional Network (FCN) [35] or DeepLab [12] architectures with ResNet [23] or VGG [28] networks as feature extractors.

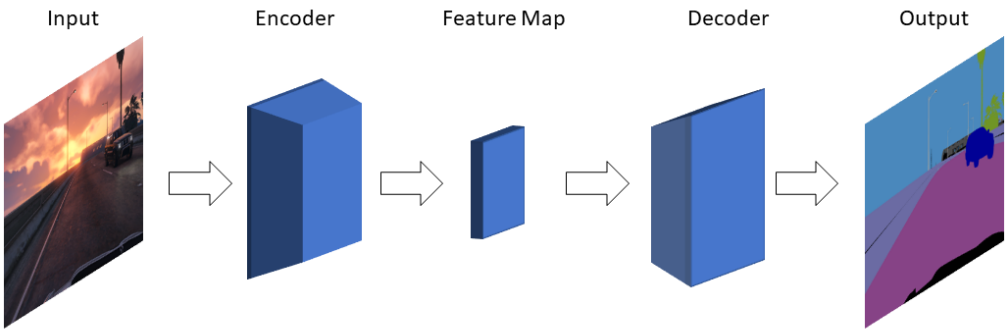


Figure 3.3. High-Level Semantic Segmentation Pipeline

Figure 3.3 shows the forward propagation of an image through the network. A semantic segmentation network consists of the following high-level parts:

1. A *feature encoder* which transforms the images from their raw format $\mathbb{R}^{C\times H\times W}$ into feature space $\mathbb{R}^{C'\times H'\times W'}$
2. A *decoder* which maps the feature representation of the image to a prediction tensor $\boldsymbol{Y}$ as defined in Equation 3.6

Inclusion of the feature encoder and decoder in Equation 3.6 results in the formulation:

$$\boldsymbol{X} \in \mathbb{R}^{C\times H\times W}, \boldsymbol{Y} \in \mathbb{R}^{K\times H\times W} : g(f(\boldsymbol{X})) \rightarrow \boldsymbol{Y} \tag{3.7}$$

with the feature extractor f and the classifier g.
Commonly, the spatial dimensions of the features produced by $f(\boldsymbol{X})$ are smaller then the initial height H and width W of the input images $\boldsymbol{X}$ ($H' \ll H, W' \ll W$) and the channel depth of the features is greater then the number of channels C of the input images($C' \gg C$). Also, for the sake of parallelization, commonly multiple images are passed to the network at once, resulting in an image batch $\boldsymbol{X} \in \mathbb{R}^{B\times C\times H\times W}$.
The objective of semantic segmentation networks is conventionally the minimization of the *cross-entropy* [9, p.235]:

$$\mathcal{L}_{seg}(\boldsymbol{X}, \hat{\boldsymbol{Y}}) = - \sum_{(h,w)\in[1,H]\times[1,W]} \sum_{k=1}^{K} \hat{Y}^{(h,w,k)} \log g(f(\boldsymbol{X}))^{(h,w,k)} \tag{3.8}$$

where $\hat{\boldsymbol{Y}}$ is the true label map (also called *ground truth*) for the input image $\boldsymbol{X}$ and the loss is calculated for every pixel in the image. The entropy is minimal if the network

prediction $g_\sigma(f(\boldsymbol{X})) = \arg\max_k g(f(\boldsymbol{X}))$ corresponds to the ground truth $\hat{\boldsymbol{Y}}$ for every pixel in the image.

3.3. Domain Adaptation for Semantic Segmentation

Reviewing the network architecture presented in Figure 3.3, domain adaptation for semantic segmentation can be achieved by adaptations at three different levels [56]:

- Input level
- Feature level
- Output level

All adaptation methods work by inserting domain adaptation modules and altering data at the appropriate point in the architecture pipeline.

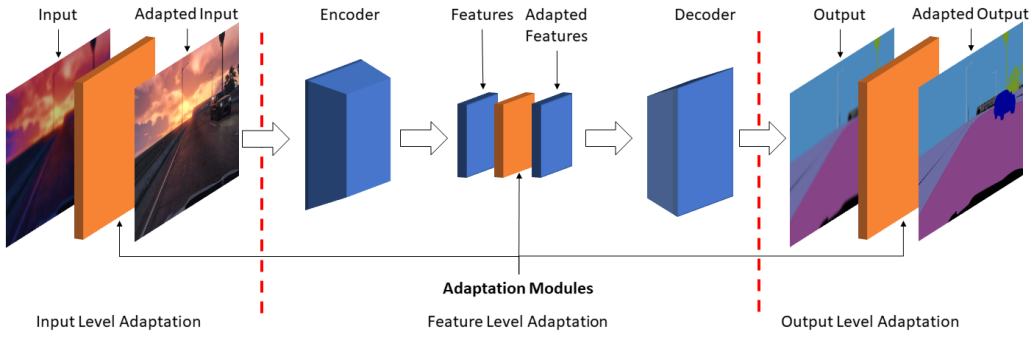


Figure 3.4. Domain adaptation for Semantic Segmentation (based on [56])

Figure 3.4 depicts, where the modules would be placed in the architecture shown in Figure 3.3. In the following, the three adaptation methods are described in detail.

3.3.1. Input Level Adaptation

Input level adaptation refers to the alignment of the visual appearance of images between source and target data. Visual alignment of images from different domains is referred to as *style transfer*. In style transfer, the goal is to infer visual characteristics, specific

to a certain domain A onto the content of an image from domain B with minimal loss in the images structural information. Here, many approaches have been explored, like the utilization of *cycle consistency* in Cycle-GANs [68]. In cycle consistency, images are transferred from domain A to domain B, and the result is then transferred back to domain A. The difference between the original image and the image after two transfers can be minimized, resulting in cycle consistency. Bidirectional learning has also been applied to semantic segmentation [34] with image-translation being a key part of the algorithm, yielding a good level of success. More recently, Cai *et al.* introduced a new framework called Frequency Domain Image Translation (FDIT) [11], which yields good results for pixel accurate transfer of style information.
The biggest advantage of input level adaptation by style transfer compared to the other adaptation methods is, that, if the visual cues of the source domain are transferred perfectly to the content of the target domain, the data distributions between source and target data become independent and identically distributed (iid.). This results in the simplified problem of *semi-supervised learning* [61]. In semi-supervised learning, the data is partially unlabeled but drawn from the same distribution. Without the domain gap, training a neural network becomes easier, since training on source data only provides more accurate results.

(a) Source for Content (b) Target for Style (c) Source in Target Style

Figure 3.5. Frequency Domain Adaptation (FDA) [65] artifacts

Two big pitfalls in domain adaptation by style transfer, on the other hand, are either the relaxation of pixel-accuracy after the transferring images, which is the problem of existing GANs [13], conditional adversarial nets [27] and autoencoder networks [41], or the presence of artifacts in the results, visible in the form of dark color patches in the results of Fourier based domain adaptation [65] (Figure 3.5) and color jittering in the results of Cycle-GAN [68] (Figure 3.6).

(a) Real Cityscapes Image (b) Fake Cityscapes Image

Figure 3.6. Image Translation using Cycle-GAN [68]

Notably, the work of Cai *et al.* [11] promises compelling results with little artifacts and high pixel accuracy by focusing on the transfer of low frequency information from the source domain and keeping the high frequency information of the target images unchanged. Though it is not part of this thesis, domain adaptation using FDIT [11] could also result in good adaptation quality.

3.3.2. Output Level Adaptation

In output level adaptation, the goal is to gradually improve the quality of the networks predictions on target data. One method to achieve this is the adaptation of entropy in the predictions for source and target data, as proposed by Vu *et al.* [62]. Entropy, in this context, relates to the pixel-wise prediction certainty over the labels, where low entropy scores are equivalent to high certainty predictions. Training the network to produce low entropy results on target data increases the prediction certainty, thereby increasing the prediction quality.
Another method for output level adaptation is called *self-supervised learning*, in which the goal is to use the predictions of the network on target data as pseudo-labels [69] which

gradually improve over time.
Commonly, thresholds on confidence of the networks predictions are used for pseudo-labels in order to guarantee, that wrongly classified pixels do not jeopardize the training process. Over time, as the quality of the predictions on target data increases, the number of pixels crossing the threshold in prediction certainty also increases. Since the network is more likely to predict easy examples with high certainty, especially in the beginning of the training, those easy examples will be the main focus of the adaptation in the beginning, and towards the end of the training the network will move to more complex examples. This type of training, where easy examples are dealt with first in order to guide solutions for complex examples, is referred to as *curriculum learning*.
The pitfall of output level adaptation is the dependence on correct pseudo-labels. If the pseudo-labels contain incorrectly classified pixels, the network will be encouraged to misclassify those pixels and all similar pixels throughout the training.

3.3.3. Feature Level Adaptation

The last adaptation method is targeting the feature representation layer, the output of the feature encoder in Figure 3.3. The feature representation of a network is a high dimensional mapping of the image into *feature space* usually obtained by successive application of convolutions, activation functions (*e.g.* ReLU [2], Sigmoid [8, p.10]) and pooling [8, p.39], as well as Batch [26] and Instance Normalization [59]. As a more abstract form of domain adaptation, feature level adaptation is difficult to conceptualize intuitively. Therefore, section 2.4 gives a geometrically intuitive explanation of domain shift in abstract spaces.
GAN architectures can also aid the alignment of feature representations between source and target data. Multiple publications [32, 51] use the feature representation to simultaneously create label maps and reconstruct the original input image. The pixel-level distance between the reconstructed image and the original image is included in the total loss for both, source and target data, while the cross-entropy is calculated on source data only.
This work falls into this category of adaptation. It leverages an output level adaptation framework to create pseudo-labels, and uses these pseudo-labels to create crops, which are aligned in feature space. A detailed description of the proposed method is presented in section 4.2. Notably, during the creation of this thesis, the SePiCo framework [64] was committed for review. SePiCo is conceptually very similar to the approach, proposed in this thesis. It uses crops of classes to align feature representations of objects *contrastive learning* (described in section 3.4).

WANG *et al.* proposed to use different feature alignment strategies for different classes, where features of target data for so-called *stuff* classes (*e.g.'sky', 'tree'*) are aligned towards a *centriod* of feature representations for source data and features for so-called *instance* classes (*e.g. 'car', 'person'*) are aligned towards the closest *instance* feature representations for source data [63].
Feature level adaptation is often combined with output level adaptation, since different classes have different feature representations requiring self-supervised learning for creation of pseudo-labels for target data.
Like output level adaptation, feature level adaptation is highly susceptible to wrongly classified pixels in the pseudo-labels. Furthermore, as a highly abstract form of adaptation, the training of feature level adaptation is inherently difficult. In theory, every sample $\boldsymbol{X}$ has a feature representation $f(\boldsymbol{X})$, for which the classification $g(f(\boldsymbol{X}))$ is optimal. In practice, the optimal feature representation is unknown, which renders distinguishing between 'correct' and 'incorrect' feature representations very difficult. In contrast, visual comparison of images in input level adaptation and pseudo-labels for output level adaptation provide a direct quality assessment of the adaptation.
In the following section an approach for clustering features in feature space is described. The benefit of clustering features is that the distance of a feature representation to its according cluster indicates the quality of the feature representation.

3.4. Contrastive Learning

The goal of *contrastive learning* is to increase the margin between the feature representation of different classes and to decrease the distance between feature representations of the same class. Similarity, in this context can be interpreted in many ways, such as structural similarity (*e.g.* texture, shape, size), visual similarity (*e.g.* color, brightness, gloss), spatial similarity (*e.g.* location) or semantic similarity (*e.g.* objects belong to the same class). Whereas structural, visual and spatial similarities can be extracted from the images directly, semantic similarities require further resources like label-maps or bounding boxes for objects. Works as TextonBoost [54] have explored extraction of similarities from images. In recent years the idea of using label information for the extraction of semantic similarities emerged [25, 4].

Like many other developments in semantic segmentation, contrastive learning is based on adaptation methods for image classification. Especially in the context of face recognition [52], the importance of accurately differentiating between different individuals is crucial.

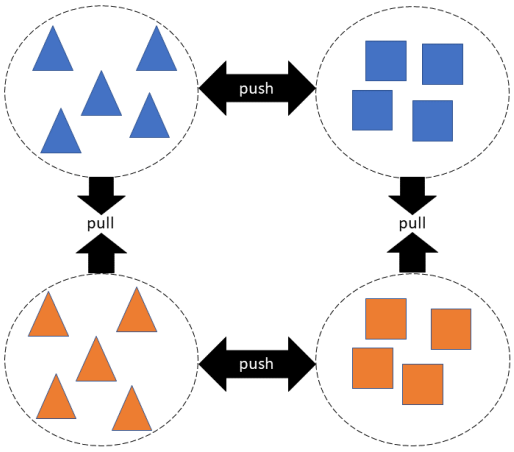


Figure 3.7. Schematic Visualization of Contrastive Learning

As depicted in Figure 3.7, the main idea of contrastive learning is to push apart representations of objects from different categories (visualized by △ and □) and pull together representations of objects from the same category, even if they are from different domains (visualized in orange and blue). In order to achieve this contrastive representation of faces, SCHROFF *et. al.* introduced a *triplet loss* using an anchor $\boldsymbol{a}$, a positive example from the same individual $\boldsymbol{p}$ and a negative example from a different individual $\boldsymbol{n}$. For random samples $\boldsymbol{p}$ and $\boldsymbol{n}$ the optimization objective is the minimization of [52]:

$$\mathcal{L}_{triplet}(\boldsymbol{a}, \boldsymbol{p}, \boldsymbol{n}) = \|\boldsymbol{a} - \boldsymbol{p}\| - \|\boldsymbol{a} - \boldsymbol{n}\| \tag{3.9}$$

In Facenet [52], the anchor $\boldsymbol{a}$ and the positive example $\boldsymbol{p}$ are feature representations of different images of the same face and the negative example $\boldsymbol{n}$ is the feature representation of an image of a different face.
For domain adaptation in the image classification context, triplet loss has been used to align feature representations of images belonging to the same class for intra and inter domain samples [16, 6]. The challenging part of aligning feature representations of images belonging to the same class from source and target domain, is the absence of labels for target data. Existing approaches [16, 6] bypass this problem by using the network to create pseudo-labels for target data, which guides the selection of positive and negative samples for a given anchor. In order to avoid the utilization of wrongly classified labels as pseudo-labels for target data, only pseudo-labels which surpass a certain confidence

threshold are used.

Algorithm 3: Domain Adaptation using Triplet Loss from Equation 3.9

input : Source Data: $\mathcal{X}_\mathcal{S} = \{(\boldsymbol{X}_\mathcal{S}, y)^i\}$
Target Data: $\mathcal{X}_\mathcal{T} = \{\boldsymbol{X}_\mathcal{T}^i\}$
Model $g(f(\boldsymbol{X}))$
Threshold t

output : Trained network $g(f(\cdot))$

Train network on source data $\mathcal{S}$
forall $\boldsymbol{X}_\mathcal{T}^i \in \mathcal{X}_\mathcal{T}$ **do**
  $l_i, c_i \leftarrow g(f(\boldsymbol{X}_\mathcal{T}^i))$
  if $c_i > t$ **then**
    Sample $(\boldsymbol{P}, p) \in \mathcal{X}_\mathcal{S}$ with $p = l_i$
    Sample $(\boldsymbol{N}, n) \in \mathcal{X}_\mathcal{S}$ with $n \neq l_i$
    Minimize $\mathcal{L}_{triplet}(f(\boldsymbol{X}_\mathcal{T}^i), f(\boldsymbol{P}), f(\boldsymbol{N}))$
  end
end

The pseudo-code in algorithm 3 shows a generalized version of domain adaptation using the triplet loss function, where c_i represents the confidence of a prediction using the classifier g, and f is the feature extractor as defined in Equation 3.7.
Though, for image classification this approach is feasible, for semantic segmentation the fact, that an image contains a multitude of classes at different locations, constraints this simple solution. Hence, aligning the feature representation of full images does not promise good results. Previous works introduce attention modules for feature extraction in order to differentiate between the classes [4], or require the feature-map to obtain the dimensionality of the input image (conventionally $\mathbb{R}^{C \times H \times W}$) in order to use given ground-truth labels setting constraints for the utilized network.
In object detection, Rezaeianaran *et al.* [45] have proposed a method, which utilizes contrastive learning. In their approach, visually similar regions are clustered in both domains. Then, a domain discriminator is used to align those clusters between the domains. The idea for the approach, presented in section 4.2 is partially inspired by the approach of Rezaeianaran *et al.*.

3.5. Analysing State-of-the-Art Methods

As the baseline for this thesis the implementation[1] from the publication "Self-Supervised Augmentation Consistency for Adapting Semantic Segmentation" [5] by ARASLANOV *et al.* is chosen. Categorically it belongs to the methods of output adaptation (introduced in section 3.3). The idea is to enforce prediction consistency on images with the usage of *data-augmentation* (*e.g.* photometric transformations, flipping, cropping). Pseudo-labels for the target data are created using *multi-scale fused* predictions of a slow evolving *teacher network* (also referred to as *momentum network*). These pseudo-labels are used to guide the training of the *student network* by computing the cross entropy (Equation 3.8) between the prediction of the teacher and the student network.

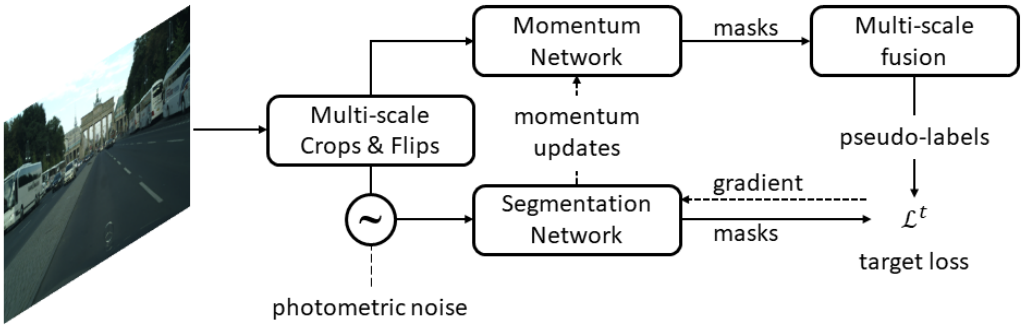


Figure 3.8. Self-Supervised Augmentation Consistency (SAC) framework [5]

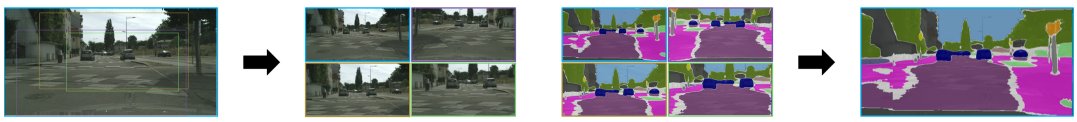
Figure 3.8 shows the Self-Supervised Augmentation Consistency (SAC) framework as proposed by ARASLANOV *et al.*. The training procedure consists of the following steps:

1. Train a segmentation network on the source data while using Adaptive Batch Normalization (ABN) [33] on target data
2. After a number of epochs (≈100) introduce a separate momentum network, which uses multi-scale fusion for creation of pseudo-labels and is updated after a given number of iterations to provide better generalization

[1] Code available at `https://github.com/visinf/da-sac`

3. Train the segmentation network iteratively on batches from source (using existing labels) and target data (using pseudo-labels)

The objective of ABN is to minimize the cross entropy loss on source data. On target data, all but the batch normalization layers are frozen in the backward pass allowing statistical meta-data of the target data to be included in the training. Running ABN for approximately 100 epochs produces a good segmentation model on source data. With an accuracy of ≈36% on target data, the model serves as a good pretrained baseline for the start of SAC training.
The framework contains multiple approaches for dealing with a variety of problems. The pretrained ABN model is used to go over the target data and create pseudo-labels for every image. Using these pseudo-labels, a list is created, that enlists, which labels are seen in which image. This list is used to produce an adapted sampling probability for every target image. This sampling procedure is called *importance sampling*. With importance sampling, target images, which contain rare classes (*e.g. 'bike', 'traffic light'*) are sampled with a higher probability.
Another approach, utilized by the framework, is a *moving threshold* for each class which aims to improve the quality of pseudo-labels for rare classes. The threshold of a class being assigned to a pixel in pseudo-labels changes throughout training for each class independently. Classes which appear less frequently have a lower threshold. All pixels, for which no class surpasses the threshold are assigned no label and excluded from the loss function. In previous works [34], the computation of pseudo-labels was taken offline. The training had to be interrupted and pseudo-labels recomputed which was necessary for determining the thresholds of classes for pseudo-label creation. The SAC framework has the computation of those thresholds included in its online training.



(a) Target Image to Batch (b) Fused Prediction

Figure 3.9. Multi-Scale Fusion

For the improvement of accuracy of pseudo-labels, the SAC framework proposes *multi-scale fusion*. In multi-scale fusion, an image from target data is given to the pseudo-label creating momentum network in multiple variations, in form of a full image, an image crop or a flipped image crop. The affine transformations, used to create those crops, as well

as the inversion of those transformations are tracked. This means, that every cropping operation can be inverted and the location of the cropped region in the original image is known. The predictions are fused in the overlapping regions. The variations are scaled to the same size and stacked into a batch, before being given to the network. This results in thresholded pseudo-labels $\overline{\boldsymbol{Y}}$ with a higher number of correctly classified pixels, than pseudo-labels, created without fusion of multiple predictions.
A student model $g(f(\cdot))$ is given the input image with photometric pertubations and distortions and the cross entropy (Equation 3.8):

$$\mathcal{L}_{pseudo} = \sum_{n=1}^{N} \mathcal{L}_{seg}(\boldsymbol{X}_n, \overline{\boldsymbol{Y}}_n) \tag{3.10}$$

between the pseudo-labels $\overline{\boldsymbol{Y}}$ and the prediction $g(f(\boldsymbol{X}))$ of the student model is calculated.
An iteration in SAC training consists of the following steps:

1. Load image and label pairs $(\boldsymbol{X}_{\mathcal{S}}, \boldsymbol{Y}_{\mathcal{S}})$ from source data
2. Calculate $\mathcal{L}_{seg}(\boldsymbol{X}_{\mathcal{S}}, \boldsymbol{Y}_{\mathcal{S}})$ as defined in Equation 3.8
3. Load image $\boldsymbol{X}_{\mathcal{T}}$ from target data and create pseudo-labels $\overline{\boldsymbol{Y}}$ using multi-scale label voting on teacher model
4. Calculate $\mathcal{L}_{pseudo}$ for the output of the student network on the created pseudo-labels
5. Propagate the loss backwards trough the network and create appropriate gradients for the parameters of the student network
6. Take a step, according to gradient descent optimization

This training procedure will be the basis for the adaptation method proposed throughout the next sections.

4. Proposed Method

This chapter presents the main approach for domain adaptation in semantic segmentation of this work. The chapter firstly provides a motivation, from which the main method of this thesis is derived. Thereafter, an overview of the proposed method is given in sections 4.2 to 4.7.

4.1. Motivation

Many solutions for the task of domain adaptation for semantic segmentation have been implemented in the last years. A common problem of the methods discussed in section 3.3 is the confusion of similar looking classes.

(a) Ground Truth (b) SAC Inference (c) FS Inference

Figure 4.1. Comparison between inference from SAC [5] and Fully Supervised (FS) training on GTA←Cityscapes

As shown in Figure 4.1, the SAC model trained on GTA data [47] confuses the class *'road'* for the class *'sidewalk'* in some regions of the image whereas the model trained directly on Cityscapes data [14] has less problems differentiating between the two classes.
This thesis proposes a framework for improving the class-discriminative abilities of CNNs for domain adaptation applied to semantic segmentation tasks. The proposed method, titled Feature Representation Learning (FRL), aims to improve the prediction of output

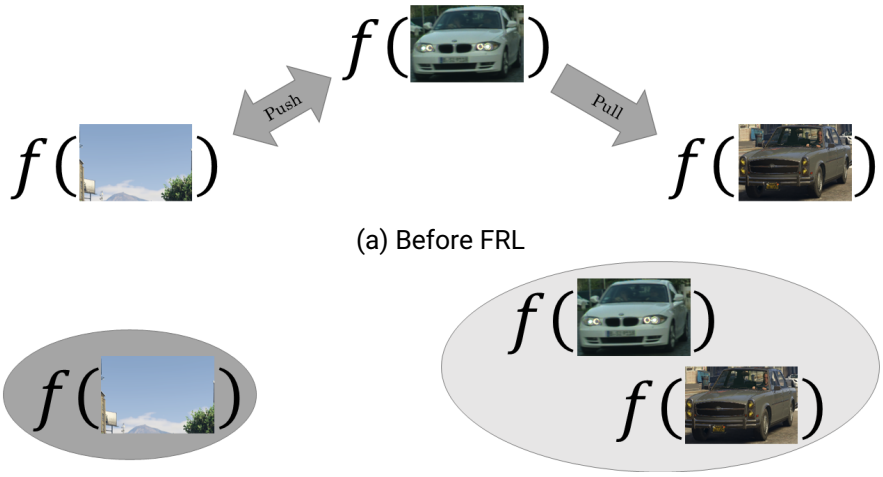
level adaptation architectures like SAC. By leveraging their pseudo-label creation process for the extraction of *class-sensitive crops*, the feature representations of classes for both domains are clustered in feature space.

4.2. Feature Representation Learning

Although more commonly seen in image recognition [20, 58], feature representation alignment also finds its application in semantic segmentation [25, 4]. Distinguishing between important and unimportant components of the feature representation for every class is crucial for the success of feature level adaptation methods in semantic segmentation. In image classification feature level adaptation is commonly used [16, 6, 20, 58] due to the spatially unbound nature of labels for the task (*e.g.* numbers, letters, scene descriptions). Since the predictions of semantic segmentation and object detection tasks are spatially bound (*e.g. "pixel at location* (x, y) *belongs to class 'car'... "*), aligning feature distributions of classes becomes more difficult.
Methods for feature alignment have been proposed in the past. SAITO *et al.* [49] encourage the model to produce class-discriminative features by adversarially freezing the feature encoder and decoder on the same input. This results in a class-discriminative clustering in feature space.
This work introduces a novel method for *feature representation alignment* by sampling from single class regions and aligning the crops in feature space, without training the decoder afterwards.

An intuitive schematic overview of feature representation learning is depicted in Figure 4.2, where $f(\cdot)$ are the feature representations of the cropped images created by the network. The goal is to cluster the feature representations of crops of the same class spatially so the classifier is incentivized to assign the same class to other image patches containing this class.
In contrast to the original implementation of the triplet loss, where the anchor $\boldsymbol{a}$ was stationary, the idea in this work is to reverse the role of anchor and the examples, where the anchor (referred to as *prototype*) $\boldsymbol{a}$ is moved towards the positive example and away from negative examples.
FRL requires a single assumption:

Assumption 1 *Let* $\boldsymbol{a}_1$*,* $\boldsymbol{a}_2$*,* $\boldsymbol{n}$ *and* $\boldsymbol{p}$ *be four normalized vectors with* $\|\boldsymbol{a}_1 - \boldsymbol{p}\| < \|\boldsymbol{a}_1 - \boldsymbol{n}\|$ *and* $\|\boldsymbol{a}_2 - \boldsymbol{p}\| > \|\boldsymbol{a}_2 - \boldsymbol{n}\|$*. Let* $h(\cdot)$ *be a loss function, that judges the quality of those vectors.*



(a) Before FRL

(b) After FRL

Figure 4.2. Feature Representation Learning (FRL)

Then, $h(\boldsymbol{a}_1) < h(\boldsymbol{a}_2)$ *holds.*

In other words, assumption 1 states, that between two vectors $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$, the one closer to the positive sample $\boldsymbol{p}$ produces a better result with regards to the loss $h(\cdot)$. Using this assumption, FRL can be formalized as follows.
Let $g(f(\cdot))$ be a network trained on source data $(\mathcal{X}_\mathcal{S}, \mathcal{Y}_\mathcal{S})$ only (ABN extension as used in section 3.5 is possible) and $\mathcal{X}_\mathcal{T}$ be the unlabeled target data. The loss function for feature representation learning is formulated as:

$$(\boldsymbol{X}_\mathcal{S}, \boldsymbol{Y}_\mathcal{S}) \in (\mathcal{X}_\mathcal{S}, \mathcal{Y}_\mathcal{S}), \boldsymbol{X}_\mathcal{T} \in \mathcal{X}_\mathcal{T} :$$
$$\mathcal{L}_{FRL} = \sum_{i=0}^{K} \sum_{\substack{j=0 \\ j \neq i}}^{L} \mathcal{L}_{triplet}(f(\mathcal{I}(\boldsymbol{X}_\mathcal{T}, \overline{\boldsymbol{Y}}, i)), f(\mathcal{I}(\boldsymbol{X}_\mathcal{S}, \boldsymbol{Y}_\mathcal{S}, i)), f(\mathcal{I}(\boldsymbol{X}_\mathcal{S}, \boldsymbol{Y}_\mathcal{S}, j))) \quad (4.1)$$

where $\mathcal{I}(\boldsymbol{X}, \boldsymbol{Y}, k)$ is a *class-sensitive crop* (explained in section 4.4) of class k taken from image $\boldsymbol{X}$ using segmentation map $\boldsymbol{Y}$ and $\mathcal{L}_{triplet}$ can be used as defined in Equation 3.9. $\overline{\boldsymbol{Y}}$ is the pseudo-label created from the output of $g(f(\cdot))$.
The feature representation loss $\mathcal{L}_{FRL}$ is included in the training in order to increase feature representation distance of crops from different classes and to decrease feature

representation distance of crops from the same class. Instead of the triplet loss (defined in Equation 3.9) the InfoNCE [1] loss function:

$$\mathcal{L}_{NCE}(\boldsymbol{a}, \boldsymbol{p}, \boldsymbol{n}) = -\log \frac{\exp \boldsymbol{a} \cdot \boldsymbol{p}}{\exp \boldsymbol{a} \cdot \boldsymbol{p} + \exp \boldsymbol{a} \cdot \boldsymbol{n}} \tag{4.2}$$

can be used, as seen in [64, 25]. Another valid choice as distance function between feature vectors is the cosine-similarity as used by Alonso *et al.* [4]. In principal any distance function $d(\cdot, \cdot)$ between two vectors can be used as a contrastive loss function, as long as it is symmetrical, positive definite and the triangular inequality holds.

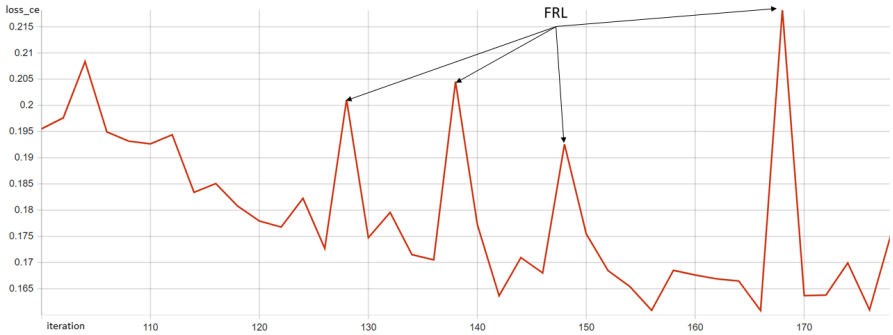


Figure 4.3. Cross-Entropy Loss after Feature alignment epoch on source data

There are two possible ways to include the feature loss (Equation 4.1) in the training:

1. Minimize cross entropy on pseudo-labels (Equation 3.10) and feature loss in alternation
2. Minimize cross entropy on pseudo-labels and feature loss simultaneously

Intuition might lead to the suggestion of training for feature representation learning and self-supervised learning in an alternating fashion, since a different feature representations lead to a different pseudo-labels. Therefore, the initial proposal for FRL was to iterate between cross-entropy loss and feature loss minimization.
However, since both training steps are antagonistic, running FRL iterations separate from cross-entropy minimization leads to spikes in the cross-entropy loss after every FRL epoch, as depicted in Figure 4.3. Though the cross-entropy values return to levels from before FRL, the convergence speed of the training is strongly reduced by these spikes.
Therefore, a fitting alternative is to minimize the cross-entropy on pseudo-labels and the feature loss simultaneously. Combined, the target loss is formulated as:

$$\mathcal{L}_{\mathcal{T}} = \lambda_{pseudo}\mathcal{L}_{pseudo} + \lambda_{FRL}\mathcal{L}_{FRL} \tag{4.3}$$

with λ_{pseudo} and λ_{FRL} being the contributions to the target loss. The full procedure is depicted in Figure 4.4.

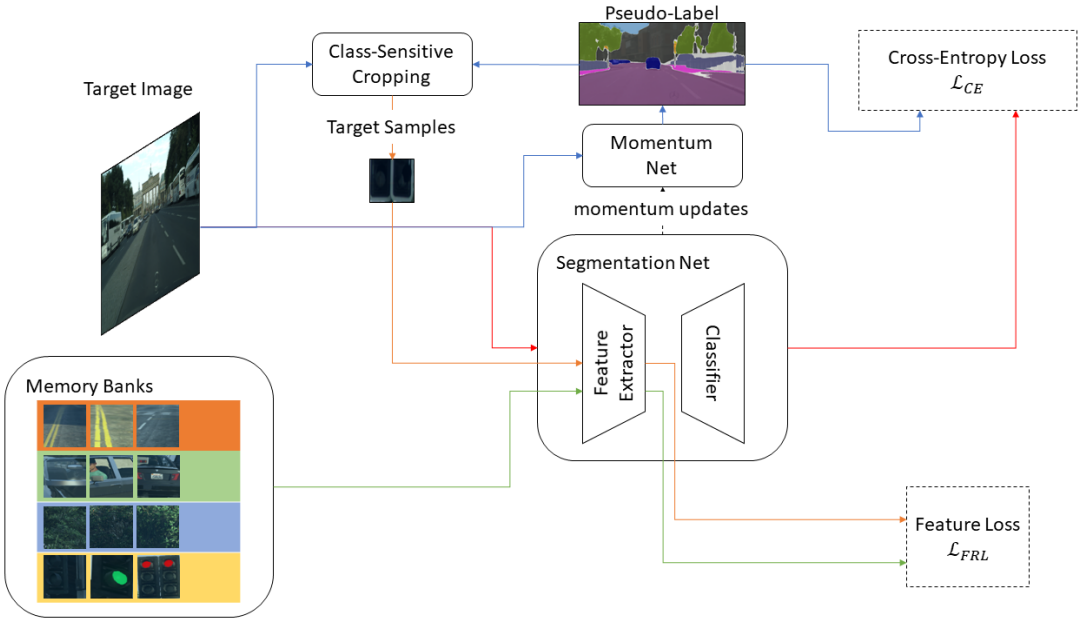


Figure 4.4. **Feature Representation Learning** target step

The cropping algorithms in section 4.4 use the pseudo-labels generated by the momentum network for sampling crops containing mostly a unique class. The crops providing positive and negative samples for Equation 4.3 are stored in memory banks (described in section 4.3). This assures, that for every class in a target image, a positive example is available. The feature representation distances of crops from the same class is minimized. Simultaneously, the feature representation distances of crops from different classes are maximized, aiding the class-discriminative abilities of the network on images from the target domain.

```
def frl(x, y, model, bank_features):
    '''
        args:
            x: images [C,H,W]
            y: labels/pseudo-labels [H,W]
            model: segmentation architecture
            bank_features:  feature representations
                            of memory banks
        return:
            loss: contrastive feature loss
    '''
    assert contains_labels(y)
    crops, valid_indicator = csc(x, y) # class-sensitive crop
    crop_features = model.features(crops[valid_indicator>0])
    loss = contrastive_loss(crop_features, bank_features)
    return loss
```

Listing 4.1 Feature representation learning

A torch-like implementation of FRL is shown in Listing 4.1. Since crops of some classes are not available in every input image and some classes contain too few pixels in the image to be useful for feature alignment, the `valid_indicator` keeps track of the indices of valid crops.

4.3. Memory Banks

The FRL loss, as defined in Equation 4.1, requires source data $(\boldsymbol{X}_{\mathcal{S}}, \boldsymbol{Y}_{\mathcal{S}})$ and target data $\boldsymbol{X}_{\mathcal{T}}$. Since rare classes (*e.g. 'train', 'bike'*) have a low probability of being seen in a random image, the feature loss can have difficulties finding positive examples for these classes. In other words, it is very unlikely that a rare class is seen in a random sample from source data and a random sample from target data.
In order to overcome this problem Alonso *et al.* [4] use banks to store features of all classes. These *feature banks* assure, that every feature representation prototype $\boldsymbol{a}$ has a positive example $\boldsymbol{p}$ it can tend towards and a negative example $\boldsymbol{n}$ it can repel from.
In the implementation presented in this thesis, the banks store the RGB values of crops containing the according classes instead of the feature representations themselves. The feature representations of the crops in the memory banks are recalculated every iteration.

The feature representations of the crops change with the progression of training as the network evolves. As seen in Figure 4.5, the distance of the feature representation for

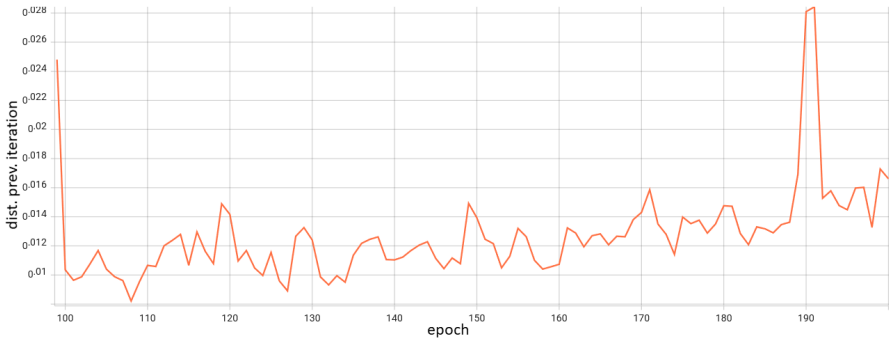


Figure 4.5. Shift of feature representations

the same crop does not stay constant. The changes of the feature representations are tracked by the mean distance to their feature representations in previous iterations. From this graph it becomes obvious, that the feature representations have to be updated every iteration, since a feature representation of a class might change over time.
Furthermore, it is statistically more advisable to exchange the entries of those memory banks. Fully filling the memory banks (*e.g* 4 slots per class $\times$ 19 classes results in 76 slots) at the beginning of every epoch ends up being very time consuming. Therefore, in FRL the banks are filled completely in the first FRL epoch and after that only individual slots are exchanged with new crops. This decreases the runtime of each epoch while simultaneously providing dynamical memory banks.
The memory banks are a key component of FRL, since they provide positive and negative class samples throughout the epoch. Therefore, it is important, that the feature representations of those crops are mapped to the correct label. This is ensured by a *feature quality filter* [4] which propagates the image through the network and compares the output to the ground truth. If the labels are incorrect, the feature representation leading up to this misclassification is also assumed to be incorrect.
Notably, since no labels are available for target data, a feature quality filter is only useful for source data. As the memory banks are filled with source data crops, the feature quality filter ensures that the feature representations of those crops are optimal.

4.4. Class-Sensitive Cropping

A main problem of domain-discriminative feature alignment [20, 58] is the class-invariant sampling of images from source and target domain. Given the example of digit recognition between the datasets SVHN [37] and MNIST [31], methods like RevGrad [20] or ADDA [58] would commonly sample different digits when aligning features using a domain-discriminator. This is inherently problematic, since the features of images with different labels are aligned in feature space (*e.g. "The feature representation of the digit 4 from MNIST is aligned with the feature representation of digit 6 from SVHN"*). This results in sub-optimal class-discriminative abilities of the trained network, since class boundaries are not inferred into the target domain. Optimally, when sampling from source and target domain the images should be conditioned with their labels. However, a basic assumption of domain adaptation tasks is that no labels for target data are available.
Works on domain adaptation using triplet loss [6, 16] overcome this hurdle by using the pre-trained network to create pseudo-labels with confidence thresholds on the outputs. This avoids the utilization of uncertain predictions for target data. For RevGrad and ADDA these approaches would assumably further increase the accuracy on target data.
This work proposes to use pseudo-labels as an orientation, in which region a class is located in the image. The feature representation of those guided crops are then used as input for the contrastive loss function. In the following two possible methods for crop extraction are presented.

4.4.1. Connected-Components based Cropping

The first method for creating class-sensitive crops uses connected-components in the segmentation labels to extract regions containing mostly the target class. The creation of crops using this method can be divided into four steps:

1. Create a binary label mask
2. Calculate connected components on binary label mask
3. Sort connected components by number of associated pixels
4. Crop around connected components with most associated pixels

(a) Image and Label (b) Label Mask (c) CC (d) Final Crops

Figure 4.6. Connected-Component (CC) based cropping

A visual demonstration of connected-components based cropping is depicted in Figure 4.6

The steps are now described in detail. Firstly, the binary label mask $\boldsymbol{M}$ of a target class k containing the entries:

$$M_{ij} = \begin{cases} 1 & , Y_{ij} = k \\ 0 & , Y_{ij} \neq k \end{cases} \tag{4.4}$$

is created. For source data the segmentation map $\boldsymbol{Y}$ is directly taken from the data. For target data the segmentation map $\overline{\boldsymbol{Y}}$ is acquired from the output $g(f(\boldsymbol{x}))$ of the network with applied thresholds. This mask serves as an orientation on where the class is located in the given image.
Then a connected component algorithm [3] is used to identify, which pixels belong to which pixel-clusters. This results in a tensor $\boldsymbol{P}$ with:

$$P_{ij} = \begin{cases} z & , M_{ij} \text{ connected to pixels from cluster } z \\ 0 & , else \end{cases} \tag{4.5}$$

Details on implementations of CC algorithms will not be discussed in this thesis. The field of CC algorithms spans a wide variety of implementations ranging from intuitive clustering by incremental cluster counting [18] to highly optimized CUDA implementations, as used for the implementation of this work [3]. Notably, many GPU implementations of CC algorithms work by iteratively clustering pixels with a fixed number of iterations. Hence, the number of iterations influences the size of the crops used for FRL.
After the connected components are computed, they are sorted by the number of pixels belonging to the same cluster. This serves as a quality assessment of the content of the

crops. If a cluster has many pixels assigned to it, the quality of the cluster is assumed to be better due to the high resolution of the crop.
Finally, all crops are resized to the same predefined crop size. Resizing the crops is necessary, since in CNNs, the size of the input image relates to the size of the feature representations. Hence, in order to create comparable feature representations, all crops must be equivalently shaped, resulting in an equal number of dimensions in their feature representation. Results of crops using connected-components can be seen in Figure 4.7.

Figure 4.7. CC crop results

Notably, this last step of resizing pixel clusters to the same size might alter textures of classes, since big parts of the image are shrunk down to a fixed size. Therefore, subsection 4.4.2 presents an elegant alternative, that assures, that the crops contain just a single class, and the feature representations have equivalent dimensionality without the need for downsizing the image.

```
def csc_cc(x, y):
    '''
        args:
            x: image [C,H,W]
            y: label/pseudo-label [H,W]
        return:
            crops: class-sensitive crops
            valid_indicator: indicator for usability of crops
    '''
    crops = torch.zeros(NUM_CLASSES, 3, CROP_SIZE, CROP_SIZE)
    valid_indicator = torch.zeros(NUM_CLASSES)
    for c in range(NUM_CLASSES):
        label_mask = torch.zeros_like(y)
        label_mask[y == c] = 1
        valid_indicator[c] = (label_mask.sum() > 0)
        if not valid_indicator[c]:
            continue
        cc = connected_components(label_mask)
        class_clusters = cc[cc>0]
        ind, count = class_clusters.unique(return_counts=True)
        box_x, box_y = get_crop_box(cc, ind[count.argmax()])
        crops[c] = resize(x[box_y1[0]:box_y[1],box_x[0]:box_x[1]], CROP_SIZE)
    return crops, valid_indicator
```

Listing 4.2 Connected Component based cropping

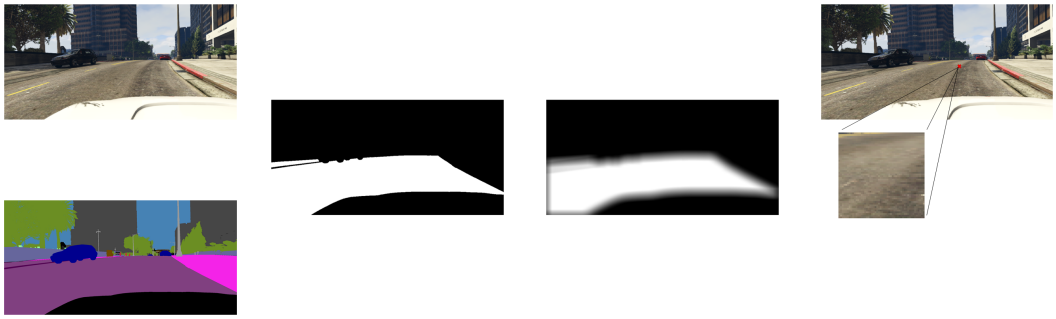
A torch-like [42] implementation of connected component base class-sensitive cropping is shown in Listing 4.2. The `connected_components` function[1] returns a cluster map, where each pixel is assigned the index of the cluster it belongs to, or zero, if the pixel belongs to a different class. The variables `box_x` and `box_y` contain the corner locations of the rectangular crop. For classes, which have many pixels assigned (*e.g. 'road'*), these boxes can exceed the crop size, while for long tailed classes (*e.g. 'traffic light'*), these boxes are usually smaller. Since for CNNs the dimensionality of the feature representations strongly depends on the size of the input image, the crops are all resized to a fixed size `CROP_SIZE`.

4.4.2. Label-Density based Cropping

The second method for the generation of class sensitive crops uses convolutions to create a density distribution of the label across the image. This so-called *label-density* refers to

[1] Kornia Implementation [46]

the number of pixels, which have the same label assigned, in every region of the image.



(a) Image and Label (b) Label Mask (c) Label Density (d) Final Crop

Figure 4.8. Label-Density based cropping

In Figure 4.8 the process for extraction of class-sensitive crops using label-density is depicted. The creation of crops can be divided into three steps:

1. Create a binary label mask
2. Calculate label density on binary label mask
3. Crop around point with maximal label-density

As for the method described in subsection 4.4.1, first a mask for the class is created as defined in Equation 4.4. The label density is approximated by the number of pixels, which are set to 1 in the label mask surrounding every pixel. It is sufficient to analyze the region of the crop size around a pixel, since pixels further away will not be part of the crop. Therefore, the label density of a class can be approximated by the number of pixels in a region belonging to the specified class. The label density for every pixel can be calculated by a convolution between a matrix, which contains ones in all entries

$$\boldsymbol{B} = \begin{pmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{pmatrix} \tag{4.6}$$

and the label mask $\boldsymbol{M}$ from Equation 4.4. Accordingly, the density $\boldsymbol{D}$ is defined as the convolution between $\boldsymbol{M}$ and $\boldsymbol{B}$:

$$\boldsymbol{D} = \text{conv}(\boldsymbol{M}, \boldsymbol{B}) \tag{4.7}$$

This is mathematically equivalent to applying a box filter to the label mask.
In order to preserve the dimensionality of the image in the label density $\boldsymbol{D}$, the convolution is padded with zeros of half the kernel size of the box filter. The kernel size of the box filter is also made uneven (*e.g.* 51×51), so the center of the convolution box has always a unique pixel location in the image.
The resulting label distribution $\boldsymbol{D}$ can be used as weight for sampling crops containing a class from the image. More efficiently, this work proposes to simply take the arg max of the distribution as the center of the crop. This makes the algorithm deterministic, so it always returns the same crops for a given label mask.

Figure 4.9. Label-Density Crops

Examples of crops are shown in Figure 4.9. Compared to crops seen in Figure 4.7 some differences are visible. Connected components based crops include a larger part of the objects whereas label-density based crops often show just a partial crop of the object (*e.g. "a wheel of a motorcycle"*). Label-density based crops can return multiple unconnected instances in the same crop as seen in column 2 of row 8 of Figure 4.9. Also notably, not all crops contain only a single class. This is the case for both, CC and label-density based cropping. Mechanisms, which target this problem are introduced in section 4.6.

```
def csc_conv(x, y):
    '''
        args:
            x: image [C,H,W]
            y: label/pseudo-label [H,W]
        return:
            crops: class-sensitive crops
            valid_indicator: indicator for usability of crops
    '''
    crops = torch.zeros(NUM_CLASSES, 3, CROP_SIZE, CROP_SIZE)
    valid_indicator = torch.zeros(NUM_CLASSES)
    for c in range(NUM_CLASSES):
        label_mask = torch.zeros_like(y)
        label_mask[y == c] = 1
        valid_indicator[c] = (label_mask.sum() > 0)
        if not valid_indicator[c]:
            continue
        label_density = conv(label_mask, uneven(CROP_SIZE))
        i = label_density.argmax(dim=1)
        j = label_density.argmax(dim=2)
        crops[c] = x[:, i-CROP_SIZE//2:i+CROP_SIZE//2, \
                        j-CROP_SIZE//2:j+CROP_SIZE//2]
    return crops, valid_indicator
```

Listing 4.3 Label density based cropping

In Listing 4.3 a torch-like [42] implementation of label density based cropping is presented. The `conv` operation returns the result of a convolution between the label mask and a bias-less 1 matrix of size `uneven(CROP_SIZE)`. As in Listing 4.2 the `valid_indicator` tracks available classes in the image.

4.5. Similarity based Loss Weight

Feature representation alignment with class-sensitive cropping is attained by direct comparison of features from the same and different classes as positive and negative examples. Since some classes are more similar in appearance than others (*e.g. 'road'* and *'sidewalk'*, *'bicycle'* and *'motorcycle'*), it makes sense to assign higher loss values to feature similarity of similar classes in order to induce a larger margin between the boundaries of those classes in feature space. Therefore, a score for similarity measurement is necessary, represented

as a matrix $\boldsymbol{W}$ with the entries:

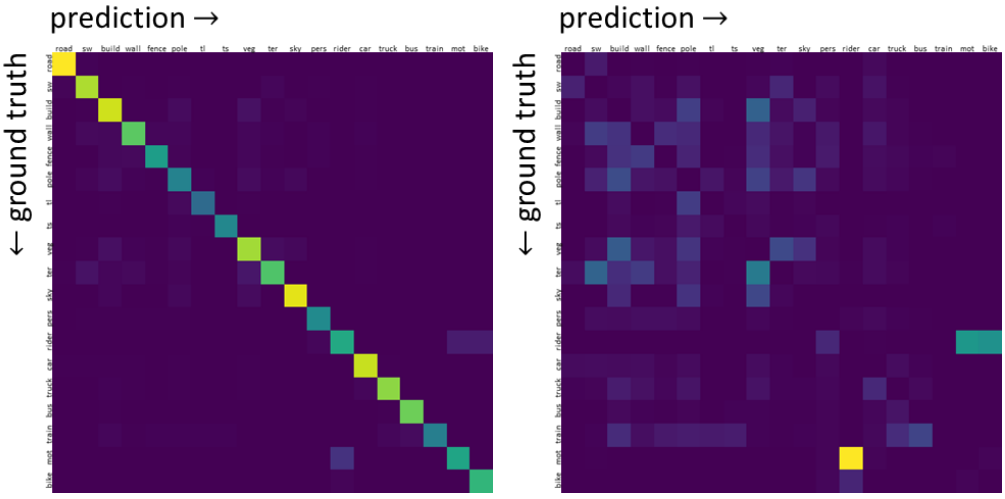
$$W_{ij} = d(i, j) \tag{4.8}$$

where i and j are two classes and $d(\cdot, \cdot)$ is a distance measure between classes. There are many possibilities to define the similarity of classes $d(\cdot, \cdot)$. Semantic grouping (*e.g."trucks and cars are vehicles"*) could be a reasonable possibility, since classes which belong to the same parent class tend to have similar visual appearance. However, this grouping would have to be done manually, since the network is only presented all classes as a set, without any hierarchical structure. Furthermore, some of the classes within these hierarchical structure might still be hard to distinguish, while others can be distinguished more easily. Therefore, a similarity measure between every individual class and all other classes is necessary.
The approach presented in this thesis uses information of the networks precision on the source domain to induce knowledge about class-similarity in the target domain. As the SAC framework proposes, the first step of domain adaptation using FRL is training a network that generates reliable predictions on the source data. In the SAC framework, this network is then used to calculate pseudo-labels for all images in the target domains training dataset and to store the content of the the images in a list (*e.g. "images containing 'car' are the following..."*). For FRL, this trained network is also used to calculate a *confusion matrix* for data in the source domain. Since predictions of neural networks are not perfect, the amount of pixels, for which the network has confused a class for another class, is a good indicator on similarity between those two classes.

In Figure 4.10, an example of a confusion matrix is shown. Along the diagonal the predicted labels of the model correspond to the true label. The entry W_{ij} in the off-diagonal contains the number of pixels, for which the model predicted the label j instead of the correct label i[2]. The result is a distribution of wrongly predicted labels.
A confusion matrix is the best indicator for classes, which are similar from the perspective of the network. By weighing the FRL loss function with the confusion rate of pairs of classes the network can focus on classes which it finds hard to distinguish. With similarity based loss weights, Equation 4.1 can be extended:

$$\begin{aligned}
&(\boldsymbol{X}_{\mathcal{S}}, \boldsymbol{Y}_{\mathcal{S}}) \in (\mathcal{X}_{\mathcal{S}}, \mathcal{Y}_{\mathcal{S}}), \boldsymbol{X}_{\mathcal{T}} \in \mathcal{X}_{\mathcal{T}} : \\
&\mathcal{L}_{FRL} = \sum_{i=0}^{K} \sum_{\substack{j=0 \\ j \neq i}}^{K} \boldsymbol{W}_{ij} \mathcal{L}_{triplet}(F(\mathcal{I}(\boldsymbol{X}_{\mathcal{T}}, \overline{\boldsymbol{Y}}, i)), F(\mathcal{I}(\boldsymbol{X}_{\mathcal{S}}, \boldsymbol{Y}_{\mathcal{S}}, i)), F(\mathcal{I}(\boldsymbol{X}_{\mathcal{S}}, \boldsymbol{Y}_{\mathcal{S}}, j)))
\end{aligned} \tag{4.9}$$

[2]In Figure 4.10b the diagonal is blanked for better visualisation of the entries next to the diagonal.



(a) Original CM (b) CM with blanked diagonals

Figure 4.10. Confusion Matrix (CM) after baseline training

where $\boldsymbol{W}_{ij}$ is the similarity between class i and class j. Since a confusion matrix is not necessarily symmetrical (as seen in Figure 4.10b), it is not inherently a distance measure. In order to make the confusion matrix symmetric, the following formula can be used:

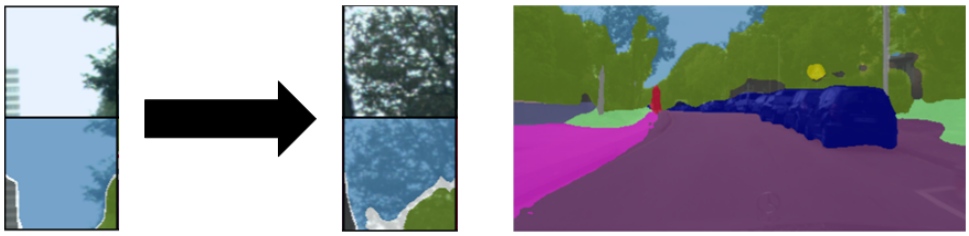
$$\boldsymbol{W}_{sym} = \frac{1}{2}(\boldsymbol{W} + \boldsymbol{W}^T) \tag{4.10}$$

Since the amount of pixels assigned to a class differs between classes, it is important to normalize each entry of the confusion matrix by the number of pixels involved in its calculation.

4.6. Feature Filtering

As mentioned above, class-sensitive crops do not necessarily contain only a single class. Some pixels may belong to a different class, than the class of interest (*e.g. "motorcycle in crop of rider"*). This results in the unwanted alignment of different classes in feature space.

As seen in Figure 4.11, the feature representation of the class *'tree'* may merge with the feature representation of *'sky'* on the border between those classes. In the following two



(a) Cropping Bias Development (b) Cropping Bias Result

Figure 4.11. FRL misclassification by reinforcement

possibilities for filtering the relevant feature dimensions for class-conditional crops are proposed.
As mentioned in section 4.2, a problem of rectangular crops is the presence of pixels with different class labels in crops. Figure 4.11b illustrates, how this problem might cascade into a misclassification of pixels over time. If λ_{FRL} is chosen too large, the network will be encouraged to assign all pixels in a crop to the same class. In order to avoid this problem, pseudo-labels can be used as a mask in the loss function.
Let $\boldsymbol{M}$ be a binary label mask as defined in Equation 4.4, which indicates, whether a pixel in the image belongs to class k. Two possibilities for masking out irrelevant pixels are plausible:

1. Input level masking
2. Feature level masking

In input level masking, the RGB values of the crop are modified, before being passed through the network. For an image $\boldsymbol{X}$, the perturbed image $\overline{\boldsymbol{X}}$ contains the values:

$$\overline{X}_{ij} = \begin{cases} X_{ij} & , M_{ij} = 1 \\ 0 & , else \end{cases} \tag{4.11}$$

Figure 4.12. Example of Class-Mix [38]

These methods of class-sensitive input level perturbations have been used for data augmentation in Class-Mix [38], where all pixels containing a class (*e.g.'road', 'car'*) were layered on top of other images. Examples for images using the Class-Mix augmentation are presented in Figure 4.12.

(a) Original Crop (b) Crop (Gaussian) (c) Crop (Constant)

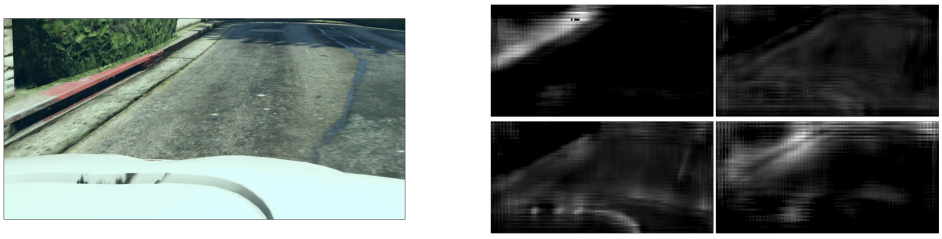
Figure 4.13. Input Level Perturbation

Figure 4.13 depicts an example of input level perturbation. Pixels, which are not assigned to the class *'road'* get replaced by either a constant value (*e.g.* zero) or the according pixel value of the crop after applying a Gaussian filter. In expectation, the absence of content

in the black parts of the image avoids the induction of biases by focusing on the relevant visual cues in the crop.
In feature level masking, $\boldsymbol{M}$ is applied to the layers in the feature representation itself. Thus, the perturbed feature representation $\overline{\boldsymbol{F}}$ of an image $\boldsymbol{X}$ is defined as:

$$\overline{\boldsymbol{F}}_l = \boldsymbol{F}_l \cdot \text{scale}(\boldsymbol{M}) \tag{4.12}$$

where $\boldsymbol{F}_l$ is the l-th layer of the feature representation and scale$(\cdot)$ is a function, that scales $\boldsymbol{M}$ to the same spatial dimensions as $\boldsymbol{F}_l$.
In the following, the feature representations using both masking methods are compared visually.

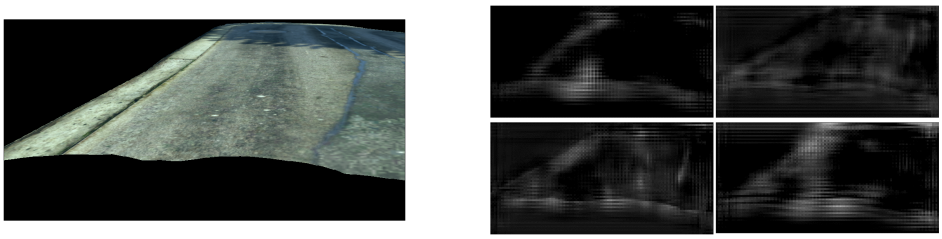


(a) Input Image (b) Features

Figure 4.14. Input image with according feature representation

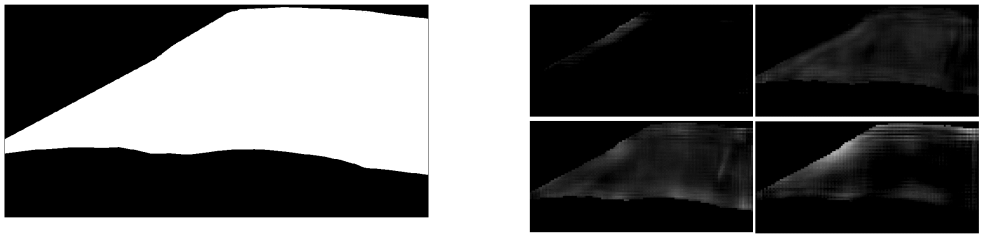
In Figure 4.14, an input image with a subset of feature representation layers is shown (only 4 out of 1024 layers are shown). Notably, the feature representations have non-zero values in in the feature representations, where the pixel labels differ from the class of interest *'road'*. Though the class *'road'* is strongly represented in the image, including dimensions of the feature representation for which the label is not *'road'* in the loss might lead to a decrease of FRL performance.

Figure 4.15a depicts the image from Figure 4.14a with all irrelevant pixels set to black. The resulting feature representation of the same feature layers as in Figure 4.14b is shown in Figure 4.15b. Notably, more zero valued entries are visible in the feature representation of the perturbed image, especially in the regions, where the original image is masked. However, the feature representation also changes in the regions where the pixels are labeled as *'road'*. Essentially, the network treats Figure 4.14a and Figure 4.15a as two different images with little similarity in their feature representation. Though not directly a part of this thesis, subsection A.1.2 discusses, how the feature representations of Figure 4.14a and Figure 4.15a could be aligned better.



(a) Perturbed Image (b) Features

Figure 4.15. Perturbed image with according feature representation



(a) Label Mask (b) Masked Features

Figure 4.16. Original features after applying label mask

Figure 4.16b depicts the feature representation of the feature layers in Figure 4.14b after applying a downscaled version of label mask Figure 4.16a. This results in hard cuts between dimensions assigned the label *'road'* and dimensions assigned a different label. Though both approaches can be used to solve the problem depicted Figure 4.11, due to the absence of *feature robustness* training (described in subsection A.1.2) in FRL training, in this thesis, only feature level masking is evaluated. Small scale tests suggest, that FRL using feature level masking produces better results.

4.7. Entropy Minimization

The concept of entropy was already mentioned with the definition of cross-entropy in Equation 3.8. Next to being a conventional semantic segmentation training objective, cross-entropy can be used as a measure of certainty for the networks prediction. The prediction certainty for every pixel can directly be extracted as the maximum of the softmax layer of a network. Per definition, the softmax layer returns a probability distribution, in which the sum over all classes for a single pixel is equal to 1.

(a) Input Image (b) Prediction Overlay (c) Prediction Entropy

Figure 4.17. Entropy in prediction of a network

In Figure 4.17 the entropy for the prediction of a network for an input image is depicted. In general the entropy is higher along the border of a class (*e.g.* edge between *'tree'* and *'sky'*) and lower for continuous regions of a class. The entropy at a pixel location (h, w) is defined as [62]:

$$E(\boldsymbol{X})^{(h,w)} = \frac{-1}{\log(K)} \sum_{(k=1)}^{K} \sigma(g(f(\boldsymbol{X})))^{(h,w,k)} \log \sigma(g(f(\boldsymbol{X})))^{(h,w,k)} \tag{4.13}$$

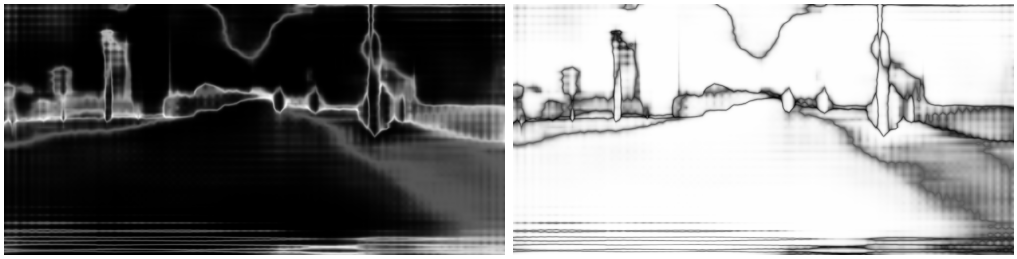
where $\sigma(\cdot)$ is the softmax function (Equation 3.5) along the class dimension.
Vu *et al.* [62] used the concept of entropy with adverserial training, where a discriminator learned to differentiate between entropy maps (example shown in Figure 4.17c) from source and target data. In general, a low entropy in the prediction of the network is desirable and used in semantic segmentation frameworks as in the work of Alonso *et al.* [4]. Thus, the loss from Equation 4.3 can be extended to account for the entropy:

$$\mathcal{L}_{\mathcal{T}} = \lambda_{pseudo}\mathcal{L}_{pseudo} + \lambda_{FRL}\mathcal{L}_{FRL} + \lambda_{ent}\mathcal{L}_{ent} \tag{4.14}$$

with $\mathcal{L}_{ent}$ being defined as:

$$\mathcal{L}_{ent} = \sum_{\boldsymbol{X} \in (\mathcal{X}_{\mathcal{S}} \cup \mathcal{X}_{\mathcal{T}})} \sum_{h=1}^{H} \sum_{w=1}^{W} E(\boldsymbol{X})^{(h,w)} \tag{4.15}$$

As proposed by Alonso *et al.* [4], the entropy term is added as a regularization to the target loss.



(a) Output Entropy (b) max of softmax

Figure 4.18. Entropy and Max. Prob.

As seen in Figure 4.18 for most cases a small entropy corresponds a large max of the softmax output. Basically, Equation 4.13 can be seen as a reduction of the softmax layer, which follows the minimization objective of the optimizer.

5. Experiments

This chapter gives an overview of the practical part of the thesis, in which the method, proposed in chapter 4, is evaluated experimentally. The implementational details, including the training setup and procedure, are specified in section 5.1. Thereafter, FRL is evaluated in section 5.2

5.1. Implementation

The implementation of domain adaptation using feature representation learning is based on the implementation of domain adaptation using self-supervised augmentation consistency. As a segmentation architecture DeepLabv2 [12] is used with VGG16 [28] as backbone. For hyperparameter tuning and proof of concept a MobileNetv2 [10] version of the framework was also implemented.
The experiments follow the convention of training on simulated synthetic data (Grand Theft Auto (GTA) game-play) with label annotations and recorded real-world data (Cityscapes) without label annotations. As a common domain adaptation task, GTA→Cityscapes allows comparison with most domain adaptation methods for semantic segmentation in literature.

5.1.1. Training

Just like in SAC, training commences with cross entropy minimization on source data and Adaptive Batch Normalization (ABN) [33] on target data for approximately 100 epochs. The best performing model is then chosen for combined FRL and SAC training. The confusion matrix, as described in section 4.5 is calculated offline after ABN training. At the same time sample weights for importance sampling in SAC are computed.
A FRL epoch consists of following steps:

1. Update current samples in the bank (section 4.3)
2. Iterate through image and label pairs $(\boldsymbol{X}_{\mathcal{S}}, \boldsymbol{Y}_{\mathcal{S}})$ from source data and images $\boldsymbol{X}_{\mathcal{T}}$ from target data:
   a) Calculate cross entropy on $(\boldsymbol{X}_{\mathcal{S}}, \boldsymbol{Y}_{\mathcal{S}})$
   b) Create pseudo-labels $\overline{\boldsymbol{Y}}_{\mathcal{T}}$ for $\boldsymbol{X}_{\mathcal{T}}$ using the teacher network
   c) Calculate cross entropy $\mathcal{L}_{pseudo}$ on $(\boldsymbol{X}_{\mathcal{T}}, \overline{\boldsymbol{Y}}_{\mathcal{T}})$ with photometric perturbations for the student network
   d) Extract class-sensitive crops $\mathcal{I}(\boldsymbol{X}, \overline{Y}, k)$ for all classes k in the image
   e) Calculate feature representations of extracted crops and of crops in the bank
   f) Calculate $\mathcal{L}_{FRL}$ (Equation 4.9) using calculated feature representations
   g) Calculate $\mathcal{L}_{ent}$ on the output of the student network for $\boldsymbol{X}_{\mathcal{T}}$
   h) Accumulate $\mathcal{L}_{pseudo}$, $\mathcal{L}_{FRL}$ and $\mathcal{L}_{ent}$ with appropriate coefficients for backward pass

In step 2e) only gradients for the crops of the target image are computed. Feature representations of crops from the bank are regarded as constants.
In order to assure the quality of pseudo-labels in the first 10 epochs after ABN training no FRL loss is calculated. Like in the SAC framework, a validation step is inserted after every second epoch and the best performing models on the second validation dataset are stored.

5.1.2. Setup

For training, the GTA dataset, consisting of 10 000 annotated images was divided into a *training set* containing 9000 images and a *validation set* containing 1000 images. The Cityscapes dataset, which consists of 3975 annotated images and 1525 unlabeled images, was divided into a training set with 2975 images and two validation sets with 500 images each. The unlabeled Cityscapes test set was not used in the training procedure.
The implementation of SAC was used as the base training procedure with DeepLabv2 as the segmentation architecture and VGG16 as the backbone.
As an extension, FRL was implemented as a subclass of SAC. During ABN training and warm-up phase (in total $\approx$ 110 epochs) no FRL was used. For pseudo-label generation the momentum network of SAC was used (described in section 3.5). The class-sensitive crops

were produced using the original images, which the momentum net uses to generate its pseudo-labels.
For the training a *NVIDIA GeForce RTX 3090* was used. Though the implementation of SAC is able to run on multiple GPUs, this functionality has not been transferred to FRL. Thus, the utilization of multiple GPUs was not possible.
Since the model had to fit on a single GPU and the maximal available capacity of 24GB was not enough for ResNet101, which requires 4 Titan-X GPUs with 12GB each, an evaluation of FRL using ResNet101 as a backbone was not possible.
For FRL the bank size 4 was chosen, and the crop size was set to 100×100 pixels. For SAC the parameters, given by the original implementation were left unchanged.

5.2. Evaluation

In order to reasonably compare the baseline framework using Self-Supervised Augmentation Consistency (SAC) presented in section 3.5 and its extension using Feature Representation Learning (FRL) presented in section 4.2 both domain adaptation approaches were evaluated extensively. This section presents the evaluation process and the according results.
The performance of a semantic segmentation model is measured using the *Intersection over Union (IoU)* metric [56]:

$$\text{IoU}_k = \frac{TP_k}{FP_k + FN_k + TP_k} \tag{5.1}$$

for each class k. In Equation 5.1, TP_k represents the number of true positives (the number of pixels correctly assigned to class k), FP_k represents the number of false positives (the number of pixels assigned to class k, although they belong to a different class) and FN_k represents the number of false negatives (the number of pixels assigned to a different class although they should have been assigned to class k).
As a unified score the mean Intersection over Union (mIoU) [56]:

$$\text{mIoU} = \sum_{k=1}^{K} \frac{\text{IoU}_k}{K} \tag{5.2}$$

is used.

Method	road	sidew	build	wall	fence	pole	light	sign	veg	terr	sky	pers	ride	car	truck	bus	train	moto	bicy	IoU
ABN	79.0	31.8	77.2	22.2	23.2	33.5	26.8	18.6	75.8	16.4	75.2	57.3	17.3	81.2	22.2	24.6	10.8	**12.7**	0.8	37.2
SAC	84.9	**45.1**	86.2	33.0	29.7	37.7	**45.2**	35.9	85.4	32.1	**87.2**	67.8	29.3	86.0	35.2	47.8	**18.1**	10.7	26.7	48.6
FRL	**85.3**	44.8	**86.3**	**35.6**	**31.7**	**37.9**	43.9	**44.2**	**85.7**	**35.8**	86.3	**68.6**	**31.3**	**86.9**	**39.4**	**54.8**	04.4	12.0	**31.1**	**49.8**
FS	*96.1*	*74.9*	*87.2*	*31.5*	*35.4*	*45.3*	*48.6*	*60.6*	*88.5*	*50.0*	*89.1*	*67.7*	*42.1*	*87.1*	*27.6*	*18.6*	*29.3*	*32.7*	*62.7*	*56.6*

Table 5.1. Per-class IoU (%) of GTA→Cityscapes adaptation, evaluated on the Cityscapes validation set (VGG16 architecture)

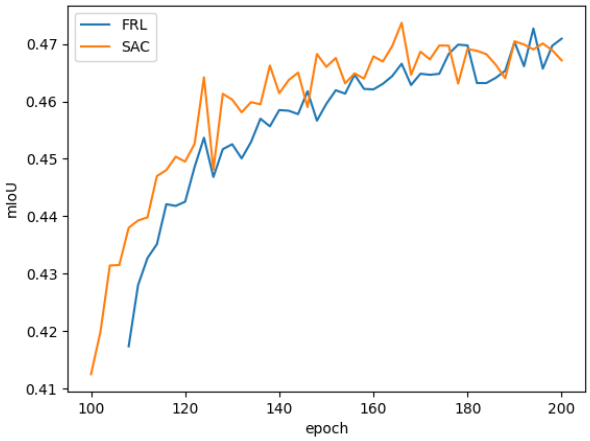


Figure 5.1. mIoU of SAC and FRL during training

In Table 5.1 the mIoU of adaptation methods is compared to the mIoU of Fully Supervised (FS) training using Cityscapes labels[1]. The results show, that SAC improves the overall classification results of ABN by about 10% with an mIoU of 48.6%[2].
FRL achieves an mIoU of 49.8% in its best runs, slightly outperforming SAC.

Figure 5.1[3] displays the evaluation of SAC and FRL on the second validation set during training. After SAC reaches its maximal mIoU around epoch 160, the improvement of SAC stagnates strongly. In contrast FRL still increases its mIoU until training is terminated in epoch 200.

[1] FS training was not optimized. Optimized training on Cityscapes reaches up to 86.93 (mIoU). [Status 18.06.2022]
[2] The original publication on SAC [5] states, that an mIoU of 49.9 has been achieved. These measures have not been reproduced experimentally.
[3] The warmup phase of FRL is not shown. Hence, training starts at epoch 108.

Crops	CM	$\mathcal{L}_{ent}$	mIoU
conv			48.4
cc			47.6
conv	✓		**49.8**
cc	✓		48.3
conv	✓	✓	48.9
cc	✓	✓	47.0

Table 5.2. Training results with different setups

Table 5.2 depicts an ablation study with different parts of the training turned off. The usage of the confusion matrix (section 4.5) increases the mIoU, but the minimization of the entropy loss $\mathcal{L}_{ent}$ seems to decrease the mIoU.

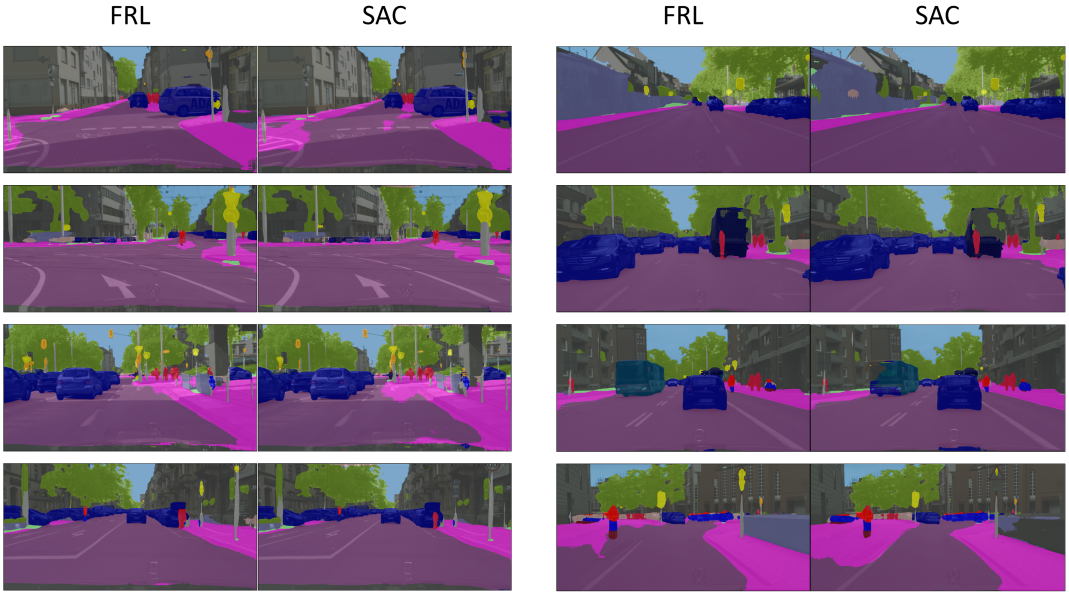


Figure 5.2. Examples for comparison between SAC and FRL

As seen in Figure 5.2, in some cases FRL is able to improve prediction accuracy. Especially the recognition of *'road'* is more accurate after FRL training. The recognition of classes *'bus'*, *'truck'* and *'traffic sign'* has also improved.

5.2.1. Class-Sensitive Cropping

The two cropping method presented in section 4.4 have both benefits and drawbacks. The computation time depends on three parameters:

- The batch size BS
- The size of the crops CS
- The number of iterations for the connected component algorithm CC_ITERATIONS

all of which influence both methods differently.

Specification	conv	cc (500 iter.)	cc (250 iter.)
CS=50, BS=2	**0.543**	2.856	1.430
CS=50, BS=3	**0.871**	3.522	1.784
CS=50, BS=4	**1.233**	4.172	2.110
CS=50, BS=5	**1.560**	4.914	2.485
CS=100, BS=2	2.121	2.999	**1.517**
CS=100, BS=3	3.417	3.503	**1.773**
CS=100, BS=4	4.646	4.173	**2.116**
CS=100, BS=5	5.834	4.851	**2.451**

Table 5.3. **Class-sensitive cropping** runtime (sec) with different setups

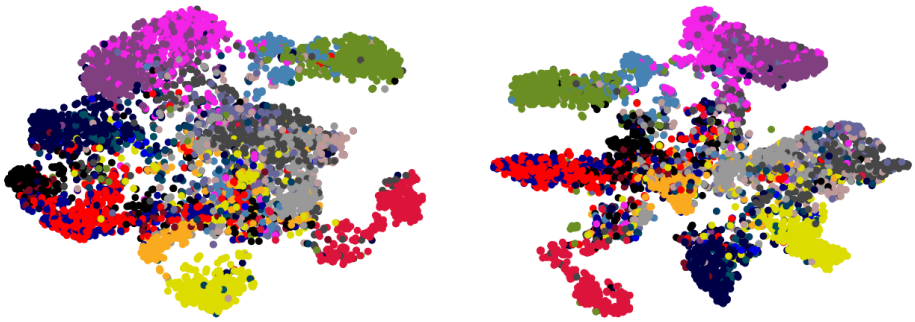
Table 5.3 shows the computation time required for crops given different inputs. Whereas label-density based cropping (conv in Table 5.3) dominates the runtime for smaller crop-sizes of 50×50 pixels with increase in crop-size and in batch-size, its computation time exceeds even the connected-components (cc) method with 500 iterations.
Hence, for larger crop sizes and bigger batch sizes the CC algorithm with 250 iterations is a reasonable choice with regards to the runtime.

Figure 5.3 shows crops produced by label-density based cropping (conv), connected components based cropping with 500 iterations (cc (500)) and connected components based cropping with 250 iterations (cc (250)). Notable differences are the shift in perspective due to the quadratic shape of the final crops (*e.g.* 100×100 pixels), which is caused by reshaping crops with arbitrary aspect ratios to quadratic crops. The downsizing from larger image patches to the given 100×100 sized crops also results in aliasing for some crops, when using CC based cropping. On the other hand crops created by conv occasionally lose their context where *e.g.* the class *'tree'* becomes just a crop of the leaves with no stem excluding important visual cues, relevant for a correct classification.

Figure 5.3. Crops produced by methods from section 4.4

5.2.2. Feature Distance Analysis

The main goal of FRL is the alignment of crops from the same class in feature space. The feature space of VGG16 on 512×1024 pixel input with 3 color channels is 8 388 608 dimensional. For crops of size 50×50 the dimensionality in feature space is reduced to 36 864 dimensions.
Class-sensitive crops essentially reduce the problem of domain adaptation for semantic segmentation to the problem of domain adaptation for image classification. Therefore, quality assessment methods for domain adaptation in image classification can also be used for the analysis of the feature representation after FRL. The most prominent method for visualization of feature representations in domain adaptation is the t-SNE visualization [60].



(a) t-SNE on SAC (b) t-SNE on FRL

Figure 5.4. t-SNE comparison using CS validation set

Figure 5.4 shows the t-SNE visualization on class-sensitive crops. A slight improvement of the feature representations of the individual classes is notable. However, the alignment does not reach the optimal state, where every class is clearly separable from every other class.

6. Discussion

In this thesis, a framework for improving the performance of CNNs on different domains was proposed. The goal of the framework was to align the feature representation of crops containing objects from the same category. Using this alignment in feature space the network was encouraged to learn a compact feature representation for the classes essentially clustering the classes in feature space.
The results presented in section 5.2 show that the approach had partial success. As seen in Figure 5.4 the feature alignment of FRL is slightly improved compared to SAC. However, Table 5.1 shows little improvement of FRL when compared to SAC. This raises the question, whether feature alignment is a usefull approach and what can be done to further improve the results.
The closely related framework SePiCo [64] hints at which flaws the implementation of FRL might have, and what can be done to improve its performance. Using a similar approach as FRL, it achieves a mIoU of 61.0% on GTA→Cityscapes using DeepLabv2 and ResNet101. Notably, SePiCo is presented in three variants. Only one of them (BankCL) uses memory banks as a source for positive and negative examples in contrastive learning. The other two variants (ProtoCL and DistCL) track the distributions of the feature maps for all classes. Still, even the BankCL implementation outperforms SAC using ResNet101 and the DeepLabv2 backbone. A direct comparison of the evaluation between SePiCo and FRL is not possible in this thesis, since FRL was not evaluated on ResNet101. But, since the performance of FRL is comparable to SAC, it is reasonable to assume, that SePiCo outperforms FRL as well.
Implementational differences between SePiCo and FRL are the following:

- Crops in SePiCo are created as random boxes and their content is evaluated after creation (in FRL class-sensitive cropping is already guided by the pseudo-labels)
- SePiCo uses a projection head for further reduction of the dimensionality of features (FRL simply flattens the feature maps into a vector)
- SePiCo does not use entropy minimization from section 4.7 as a regularization

These differences could be partially responsible for the differences in performance between the two architectures. However, one conceptual detail has the potential to drastically improve the quality of the classification when used with FRL.

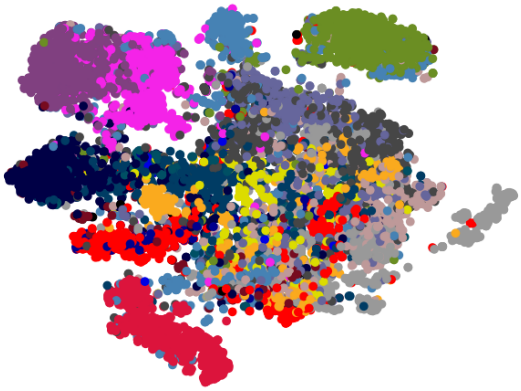


Figure 6.1. t-SNE of ABN on GTA validation

As mentioned in section 3.5, SAC uses a two stage training, where a baseline, that performs well on source data is established first using ABN. Figure 6.1 depicts the t-SNE visualization after ABN for the GTA training data, which was used as the source domain for training. For FRL the first stage of the training was left unaltered for a direct comparison between SAC and FRL.
For many classes the t-SNE visualisation shows no indication of alignment after ABN training. This means, that for numerous crops assumption 1 in section 4.2 does not hold, essentially worsening the quality of the model. Including feature alignment from the beginning of the training would therefore be the first step for improving FRL at its current state.
Naturally, feature level adaptation as an abstract form of adaptation is less intuitive as it's counterparts input and output level adaptation. In input level adaptation the success of style transfer methods can be judged directly by visual comparison, and in output level adaptation the quality of the pseudo-labels can be compared in benchmarking scenarios, where target labels are available. In contrast, feature level adaptation has little to no way for instant quality assessment since the parameterization of the neural network is very dynamical and the optimal parameterization is unknown.
This work has explored a variety of topics with regards to feature level adaptation. Subsections A.1.1 and A.1.2 propose two more topics, related to the field of feature level

adaptation.

6.1. Conclusion

In this work, the current state of research on the topic of domain adaptation for semantic segmentation was investigated. The evaluation of the state-of-the-art method SAC on the sim2real domain adaptation task on imaging datasets for autonomous driving (GTA→Cityscapes) showed, that a problem of output level adaptation methods is the misclassification of similar classes.
In order to address this problem, this work has presented a novel approach for domain adaptation. By using pseudo-label guided class-sensitive cropping, the source and target domains were aligned in feature space. It utilized the confusion matrix for classes in the source domain to infer similarity information between classes in the target domain, thereby further increasing the feature representation alignment of different classes.
The performance of the proposed method is comparable to modern state-of-the-art methods, reaching an mIoU of 49.8%. A feature analysis in Figure 6.1 suggests, that in theory, FRL has not reached its full potential, since the features are not fully aligned after training. Hence, further implementational variations might result in a better training outcome.
By essentially bridging the gap between semantic segmentation and image classification, this work also encourages further experimentation with feature level adaptation in semantic segmentation, an adaptation method mainly reserved for domain adaptation in image classification.

6.2. Outlook

As domain adaptation will stay an ongoing research topic for the foreseeable future, feature level adaptation in combination with output or input level adaptation will likely be explored in future works.
Especially in the case of semi-supervised domain adaptation where the target data is partially labeled, feature level adaptation has great potential for improving the performance. In these cases, the feature representations essentially have anchor points for the according classes, whereas in unsupervised domain adaptation the feature alignment is fully dependent on the pseudo-labels, created by the teacher network.
In unsupervised domain adaptation the combination of input level adaptation using FDIT

[11] and feature level adaptation might yield interesting results, as FDIT seems to have little artifacts and still seems to preserve most of the content of the target image. Therefore, an interesting idea would be to transfer some images from target to source domain using FDIT and use the pseudo-labels of those transferred images as anchor points for the feature representations of classes.
Whichever ideas might come up in the future, domain adaptation will most likely be important, regardless, whether in automotive research (*e.g.* sim2real), medical imaging or in space travel (*e.g.* domain adaptation from *'earth'* to *'mars'*)..

7. Acknowledgement

I would like to thank M. Pfitzer, M. Castrillon, J. Müller and S. Heinrich from Continental for our weekly meetings and numerous discussions on the topic of this thesis. I am thankful for the generous support throughout the creation of this work, and I am especially thankful for the help with setting up the environment for the experiments. I also would like to thank Prof. S. Roth, without whom this work would not have been possible. I am grateful for the fruitful discussions and his guidance through the topic. Furthermore, I would like to express my gratitude for the possibility to use the equipment at Continental for the evaluation of the experiments. Lastly, I would like to thank everybody, who proof read this work for their time and effort.

Bibliography

[1] Y. Li A. van den Oord and O. Vinyals. “Representation learning with contrastive predictive coding”. In: *CoRR* (2018). DOI: vol.abs/1807.03748.

[2] Abien Fred Agarap. “Deep learning using rectified linear units (relu)”. In: *arXiv preprint arXiv:1803.08375* (2018).

[3] Stefano Allegretti, Federico Bolelli, and Costantino Grana. “Optimized Block-Based Algorithms to Label Connected Components on GPUs”. In: *IEEE Transactions on Parallel and Distributed Systems* (2019).

[4] Inigo Alonso et al. “Semi-supervised semantic segmentation with pixel-level contrastive learning from a class-wise memory bank”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 8219–8228.

[5] Nikita Araslanov and Stefan Roth. “Self-Supervised Augmentation Consistency for Adapting Semantic Segmentation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2021, pp. 15384–15394.

[6] Imad Eddine Ibrahim Bekkouch et al. “Triplet loss network for unsupervised domain adaptation”. In: *Algorithms* 12.5 (2019), p. 96.

[7] Bilel Benjdira et al. “Unsupervised Domain Adaptation Using Generative Adversarial Networks for Semantic Segmentation of Aerial Images”. In: *Remote Sensing* 11.11 (2019). ISSN: 2072-4292. DOI: 10.3390/rs11111369. URL: https://www.mdpi.com/2072-4292/11/11/1369.

[8] Siddhartha Bhattacharyya et al. *Deep Learning: Research and Applications*. Vol. 7. Walter de Gruyter GmbH & Co KG, 2020.

[9] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738.

[10] Catalin Buiu, Vlad-Rares Danaila, and Cristina Nicoleta Raduta. “MobileNetV2 Ensemble for Cervical Precancerous Lesions Classification”. In: *Processes* 8.5 (2020). ISSN: 2227-9717. DOI: 10.3390/pr8050595. URL: https://www.mdpi.com/2227-9717/8/5/595.

[11] Mu Cai et al. “Frequency domain image translation: More photo-realistic, better identity-preserving”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 13930–13940.

[12] Liang-Chieh Chen et al. “DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.4 (2018), pp. 834–848. DOI: 10.1109/TPAMI.2017.2699184.

[13] Yunjey Choi et al. “Stargan v2: Diverse image synthesis for multiple domains”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 8188–8197.

[14] Marius Cordts et al. “The cityscapes dataset for semantic urban scene understanding”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 3213–3223.

[15] Zhiying Cui, Wu Longshi, and Ruixuan Wang. “Open set semantic segmentation with statistical test and adaptive threshold”. In: *2020 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE. 2020, pp. 1–6.

[16] Weijian Deng et al. “Rethinking triplet loss for domain adaptation”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 31.1 (2020), pp. 29–37.

[17] Zhijie Deng, Yucen Luo, and Jun Zhu. “Cluster Alignment With a Teacher for Unsupervised Domain Adaptation”. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 9943–9952. DOI: 10.1109/ICCV.2019.01004.

[18] Luigi Di Stefano and Andrea Bulgarelli. “A simple and efficient connected components labeling algorithm”. In: *Proceedings 10th international conference on image analysis and processing*. IEEE. 1999, pp. 322–327.

[19] Abolfazl Farahani et al. “A brief review of domain adaptation”. In: *Advances in Data Science and Information Engineering* (2021), pp. 877–894.

[20] Yaroslav Ganin and Victor Lempitsky. “Unsupervised domain adaptation by backpropagation”. In: *International conference on machine learning*. PMLR. 2015, pp. 1180–1189.

[21] Ian Goodfellow et al. “Generative adversarial nets”. In: *Advances in neural information processing systems* 27 (2014).

[22] Xiaoqing Guo et al. “SimT: Handling Open-set Noise for Domain Adaptive Semantic Segmentation”. In: *arXiv preprint arXiv:2203.15202* (2022).

[23] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[24] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural networks* 2.5 (1989), pp. 359–366.

[25] Hanzhe Hu, Jinshi Cui, and Liwei Wang. “Region-Aware Contrastive Learning for Semantic Segmentation”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 16291–16301.

[26] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. ICML’15. Lille, France: JMLR.org, 2015, pp. 448–456.

[27] Phillip Isola et al. “Image-to-image translation with conditional adversarial networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1125–1134.

[28] Andrew Zisserman Karen Simonyan. “Very deep convolutional networks for large-scale image recognition”. In: 2015.

[29] Alexander Kirillov et al. “Panoptic segmentation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 9404–9413.

[30] Wouter Kouw and Marco Loog. *An introduction to domain adaptation and transfer learning*. Dec. 2018. DOI: 10.13140/RG.2.2.33906.56004.

[31] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[32] Kyungsu Lee, Haeyun Lee, and Jae Youn Hwang. “Self-Mutating Network for Domain Adaptive Segmentation in Aerial Images”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 7068–7077.

[33] Yanghao Li et al. “Adaptive batch normalization for practical domain adaptation”. In: *Pattern Recognition* 80 (2018), pp. 109–117.

[34] Yunsheng Li, Lu Yuan, and Nuno Vasconcelos. “Bidirectional learning for domain adaptation of semantic segmentation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 6936–6945.

[35] Jonathan Long, Evan Shelhamer, and Trevor Darrell. “Fully convolutional networks for semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440.

[36] John A Nelder and Roger Mead. “A simplex method for function minimization”. In: *The computer journal* 7.4 (1965), pp. 308–313.

[37] Yuval Netzer et al. “Reading Digits in Natural Images with Unsupervised Feature Learning”. In: *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*. 2011. URL: http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf.

[38] Viktor Olsson et al. “Classmix: Segmentation-based data augmentation for semi-supervised learning”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2021, pp. 1369–1378.

[39] Fei Pan et al. “Unsupervised intra-domain adaptation for semantic segmentation through self-supervision”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 3764–3773.

[40] Pau Panareda Busto and Juergen Gall. “Open set domain adaptation”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 754–763.

[41] Taesung Park et al. “Swapping autoencoder for deep image manipulation”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 7198–7211.

[42] Adam Paszke et al. “Pytorch: An imperative style, high-performance deep learning library”. In: *Advances in neural information processing systems* 32 (2019).

[43] Rindra Ramamonjison et al. “Simrod: A simple adaptation method for robust object detection”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 3570–3579.

[44] Joseph Redmon and Ali Farhadi. “Yolov3: An incremental improvement”. In: *arXiv preprint arXiv:1804.02767* (2018).

[45] Farzaneh Rezaeianaran et al. “Seeking similarities over differences: Similarity-based domain alignment for adaptive object detection”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 9204–9213.

[46] E. Riba et al. “Kornia: An Open Source Differentiable Computer Vision Library for PyTorch”. In: *Winter Conference on Applicatons of Computer Vision*. 2020. URL: https://arxiv.org/pdf/1910.02190.pdf.

[47] Stephan R. Richter et al. “Playing for Data: Ground Truth from Computer Games”. In: *European Conference on Computer Vision (ECCV)*. Ed. by Bastian Leibe et al. Vol. 9906. LNCS. Springer International Publishing, 2016, pp. 102–118.

[48] German Ros et al. “The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 3234–3243.

[49] Kuniaki Saito et al. “Adversarial Dropout Regularization”. In: *International Conference on Learning Representations*. 2018.

[50] Kuniaki Saito et al. “Open set domain adaptation by backpropagation”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 153–168.

[51] Swami Sankaranarayanan et al. “Unsupervised domain adaptation for semantic segmentation with gans”. In: *arXiv preprint arXiv:1711.06969* 2.2 (2017), p. 2.

[52] Florian Schroff, Dmitry Kalenichenko, and James Philbin. “Facenet: A unified embedding for face recognition and clustering”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 815–823.

[53] Connor Shorten and Taghi M Khoshgoftaar. “A survey on image data augmentation for deep learning”. In: *Journal of big data* 6.1 (2019), pp. 1–48.

[54] Jamie Shotton et al. “Textonboost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation”. In: *European conference on computer vision*. Springer. 2006, pp. 1–15.

[55] Sandro Skansi. *Introduction to Deep Learning: from logical calculus to artificial intelligence*. Springer, 2018.

[56] Marco Toldo et al. “Unsupervised domain adaptation in semantic segmentation: a review”. In: *Technologies* 8.2 (2020), p. 35.

[57] Virginia Joanne Torczon. “Multidirectional search: A direct search algorithm for parallel machines”. PhD thesis. Rice University, 1989.

[58] Eric Tzeng et al. “Adversarial discriminative domain adaptation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7167–7176.

[59] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. “Instance normalization: The missing ingredient for fast stylization”. In: *arXiv preprint arXiv:1607.08022* (2016).

[60] Laurens Van der Maaten and Geoffrey Hinton. “Visualizing data using t-SNE.” In: *Journal of machine learning research* 9.11 (2008).

[61] Jesper E Van Engelen and Holger H Hoos. “A survey on semi-supervised learning”. In: *Machine Learning* 109.2 (2020), pp. 373–440.

[62] Tuan-Hung Vu et al. “Advent: Adversarial entropy minimization for domain adaptation in semantic segmentation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2517–2526.

[63] Zhonghao Wang et al. “Differential treatment for stuff and things: A simple unsupervised domain adaptation method for semantic segmentation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 12635–12644.

[64] Binhui Xie et al. *SePiCo: Semantic-Guided Pixel Contrast for Domain Adaptive Semantic Segmentation*. 2022. DOI: 10.48550/ARXIV.2204.08808. URL: https://arxiv.org/abs/2204.08808.

[65] Yanchao Yang and Stefano Soatto. “Fda: Fourier domain adaptation for semantic segmentation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 4085–4095.

[66] Quan Zhang. “Convolutional neural networks”. In: *Proceedings of the 3rd International Conference on Electromechanical Control Technology and Transportation*. 2018, pp. 434–439.

[67] Stephan Zheng et al. “Improving the robustness of deep neural networks via stability training”. In: *Proceedings of the ieee conference on computer vision and pattern recognition*. 2016, pp. 4480–4488.

[68] Jun-Yan Zhu et al. “Unpaired image-to-image translation using cycle-consistent adversarial networks”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2223–2232.

[69] Yang Zou et al. “Unsupervised domain adaptation for semantic segmentation via class-balanced self-training”. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 289–305.

A. Appendix

A.1. Feature Alignment

Due to the limited time for the creation of this work, some ideas, related to the methodology of feature alignment were not completely implemented or not tested. Further extensions to FRL are possible, two of which are mentioned in the next two sections.

A.1.1. Domain Discriminative Learning

A conceptual idea, mentioned before, is the utilization of domain adaptation approaches for image classification on crops, produced by the methods in section 4.4. Especially crops produced by the connected component based algorithm provide enough context, to classify the objects in the crops based on the crop only.

Figure A.1. Connected component based cropping examples

The examples in Figure A.1 can easily be assigned a category (*e.g.'road','car'*). Hence, if feature representations of crops from the source and the target domain are hard to distinguish, the network can use its classifier, trained on source data to create segmentation maps for target images.

Methods like RevGrad [20] and ADDA [58] feed the features produced by the feature encoder to a *domain discriminator*, which tries to determine, whether the features stem

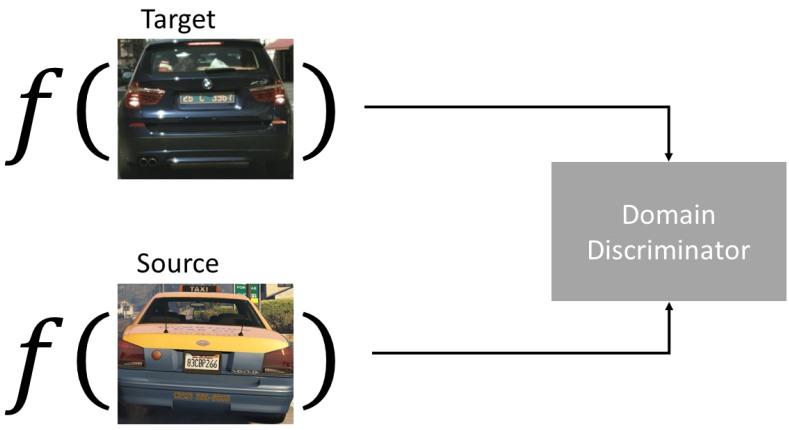


Figure A.2. Domain Discriminative Learning

from an image from the target or an image from the source data. Over time, the discriminator gets better at distinguishing characteristics of source and target domain, forcing the network to improve its alignment of both domains in feature space. These adverserial approaches have proven to be very successful for the task of image classification[1].
Though domain discriminators have been used in semantic segmentation [32, 7, 39, 51, 62], the high complexity of image scenarios is a hindering factor for their effectiveness. Hence, they are often not applied to the features produced by the network, but to entropy [62] of the images or the reconstructions of the target images [32] or the output of the network [7]. Though Pan *et al.* apply a domain discriminator to the feature representations of images [39], their approach could arguably benefit from a lower complexity level of the image.
A conceptual visualization of *domain discriminative learning* is shown in Figure A.2. At first glance, the objective of domain discriminative learning seems similar to the objective of feature representation learning. For both tasks, the goal is the alignment of data distributions in feature space. Domain discriminative learning and feature representation learning present two different training types for the achievement of this goal. While reducing the distance of feature representations and correctly labeling the pixels in the image are minimization objectives for a domain discriminator, the goal is to maximize the total loss. The maximal loss for a domain discriminator means in return, that the feature

[1] RevGrad + CAT [17] ranking third [Status 19.06.2022]

representations between the two domains are indistinguishable (at least for the domain discriminator).
Since the network and the discriminator have opposing objectives, the training is called *adversarial*. This type of training is commonly found in a Generative Adversarial Network (GAN) [21] and has proven to be effective for generative models. In GAN training the goal is the minimization of [21]:

$$\min_F \max_D V(D,F) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\left[\log D(\boldsymbol{x})\right] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[\log(1 - D(F(\boldsymbol{z})))\right] \tag{A.1}$$

with F being a sample generator function, D the discriminator, $p_{\text{data}}(\boldsymbol{x})$ being the distribution of the data and $p_{\boldsymbol{z}}(\boldsymbol{z})$ being a random other distribution.
In the context of domain discrimination the discriminator plays a similar role. When presented two feature representations, its goal is to determine which feature representation comes from which domain.
Notably, a 100% accuracy for the domain discriminator is not desired, since it implies that the feature representations of the two domains have distinguishable characteristics. This antagonistic training style reaches its optimum, if the discriminator can not distinguish between the domains. This state is called the *Nash equilibrium*. For domain discriminative learning this means, that the feature representation is optimal if the discriminator ends up guessing to which domain a feature representation belongs to.
The combination of class sensitive crops and domain discriminative learning could yield good results. Conventional domain discriminators gave the problem, that no class discriminative boundaries are transferred from source to the target domain. Class sensitive crops would solve this problem.

A.1.2. Feature Robustness Training

Another related topic in the field of feature representation learning is the concept of *feature robustness*. This final domain adaptation topic, mentioned in this thesis targets the stability of the feature representation. As the whole approach of feature representation learning depends on a good feature representation of the objects in the class sensitive crops, a stable, consistent and robust representation is beneficial for the overall performance of the approach in section 4.2.
The concept of feature robustness [67] is closely related to the concept of triplet loss and contrastive learning. The idea is to insert multiple versions of the same image with slight perturbations and punish shifts in feature space. Since the images contain the same

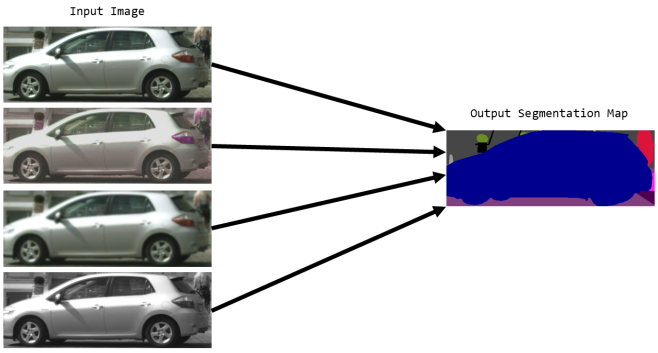


Figure A.3. Idea of data augmentation

semantic content, the output (in form of a segmentation map) of all perturbed images should be identical to the output for the original image.

The approach of output consistency is the key idea for *data augmentation* [53]. As seen in Figure A.3, all input images on the left must result in the same label map on the right. ARASLANOV *et al.* [5] leverage this idea in the SAC architecture by training the student model on images with perturbed image data (so-called *noisy images*), while the teacher creates pseudo-labels on images without *photometric transformations* [53] (*e.g.* Gaussian blur, color jittering, grayscale transformation).
Next to photometric transformations ARASLANOV *et al.* use *geometric transformations* (*e.g.* cropping, flipping) to present multiple views of the image, thereby improving scale and orientation invariance of the network. Numerous other forms of data augmentations have been proposed [43, 38], generally improving the robustness of neural networks, especially when available training data is sparse.

Since simply inserting a feature representation into decoder results in a segmentation map, if all photometric perturbed versions of the image result in the same feature representation, they have to result in the same segmentation map as well. In summary, this is the key idea of *feature robustness* (shown in Figure A.4).
For feature robustness training, only photometric transformations are applicable, since geometric transformations result in different segmentation maps. Since full 1920×1080 images have a very high-dimensional representation in feature space, the network could be fed crops created by methods from section 4.4 instead. The feature representations of these crops, created using the pseudo-labels of the teacher network are assumed to be of better quality, since they have classes assigned in the pseudo-labels.

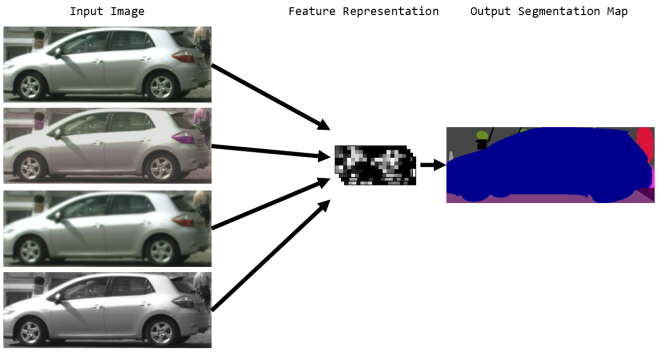


Figure A.4. Idea of feature robustness

For a set of photometric transformations $\mathcal{P}$, feature robustness is improved by minimizing:

$$\sum_{P \in \mathcal{P}} \|F(\boldsymbol{X}) - F(P(\boldsymbol{X}))\| \tag{A.2}$$

where $\boldsymbol{X}$ is a class-sensitive crop.
The feature robustness loss function can be written as:

$$\mathcal{L}_{robust}(\boldsymbol{X}) = \sum_{k=1}^{K} \sum_{P \in \mathcal{P}} \|f(\mathcal{I}_k(\boldsymbol{X})) - f(P(\mathcal{I}_k(\boldsymbol{X})))\| \tag{A.3}$$

The feature robustness loss can simply be added to the target loss from Equation 4.14:

$$\mathcal{L}_{\mathcal{T}} = \lambda_{pseudo}\mathcal{L}_{pseudo} + \lambda_{FRL}\mathcal{L}_{FRL} + \lambda_{ent}\mathcal{L}_{ent} + \lambda_{robust}\mathcal{L}_{robust} \tag{A.4}$$

with λ_{robust} being the contribution of the feature robustness loss.

```python
def robustness_loss(valid_crops, model, alterations):
    '''
        args:
            valid_crops: valid csc crops  [B,C,H,W]
            model: segmentation architecture
            alterations: photometric transformations [list]
        return:
            loss: accumulated loss for all alterations
    '''
    with torch.no_grad():
        original_features = model.features(valid_crops)
    loss = 0
    for alteration in alterations:
        altered_features = model.features(alteration(valid_crops))
        loss += (original_features - altered_features).norm()
    return loss
```

Listing A.1 Feature Robustness

A torch-like [42] implementation of a feature robustness function is shown in Listing A.1. Improving the robustness of features to photometric perturbations will likely result in feature representations which mainly focus on the content in input. This is hugely beneficial for feature alignement and would likely improve the performance of feature level adaptation frameworks like FRL.

A.2. Domain Gap Analysis

A question, rarely asked when benchmarking domain adaptation frameworks is whether the domain adaptation task GTA→Cityscapes has reached its full potential when compared to fully supervised training on Cityscapes.
Though the mIoU difference between FRL, SAC and fully supervised training in Table 5.1 is not very large, in some cases the domain adaptation frameworks consistently misclassify certain pixels whereas the fully supervised training classifies those pixels correctly.

DS	road	sidew	build	wall	fence	pole	light	sign	veg	terr	sky	pers	ride	car	truck	bus	train	moto	bicy
GTA	0.4653	0.0367	0.1363	0.0184	0.0072	0.0122	0.0014	0.0012	0.0729	0.0326	0.1708	0.0013	0.0003	0.0235	0.0128	0.0028	0.0037	0.0004	0.0001
SYNTHIA	0.1909	0.1998	0.3031	0.0028	0.0028	0.0107	0.0004	0.0010	0.1067	0.0000	0.0709	0.0439	0.0049	0.0420	0.0000	0.0158	0.0000	0.0021	0.0023
CS	0.3697	0.0598	0.2270	0.0067	0.0087	0.0126	0.0021	0.0057	0.1614	0.0111	0.0393	0.0123	0.0015	0.0692	0.0027	0.0026	0.0022	0.0010	0.0046

Table A.1. Percentage of classes

A direct pixel-count comparison of the labels between both datasets (in %) is shown in Table A.1. Since, during training, the labels of Cityscapes are not available, the network assumes the prior distribution of GTA data to be correct. This leads to an underrepresentation of heavy tailed classes like *'bike'* in the target data. In evaluation, SAC often misclassifies parts of the bike as a motorcycle. The under-representation of the class *'bike'* in the GTA dataset plays a crucial role in this misclassification.

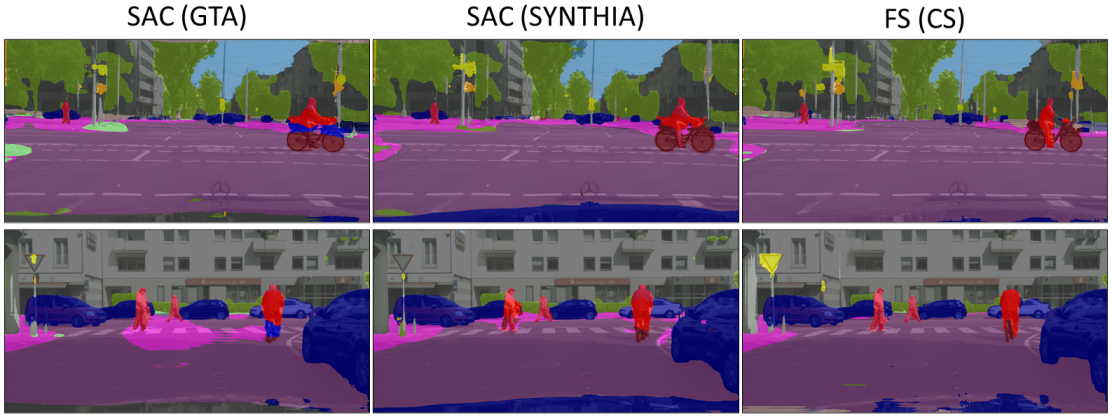


Figure A.5. Domain Gap (Example 1)

A visual comparison between the predictions of SAC, trained on the GTA dataset, the SYNTHIA dataset [48] and Fully Supervised (FS) training on Cityscapes data shows some interesting misclassification examples (Figure A.5). The network trained on GTA misclassifies upper parts of a *'bicycle'* as a *'motorcycle'* (column 1). These misclassification examples can be explained by the underrepresentation of the class *'bicycle'* in Table A.1.

Another notable misclassification example is shown in Figure A.6. Here, *'road'* is confused for *'sidewalk'* by SAC (GTA) if people are crossing the street. SAC (SYNTHIA) has no problems with this specific setting. However, as seen in row 2 of Figure A.6, SAC (SYNTHIA) confuses *'road'* for *'sidewalk'*, whereas SAC (GTA) has no problems, correctly classifying the street.
This implies a domain shift, which is not explained by Table A.1. For domain adaptation methods trained on GTA data, the classification of road and sidewalk is conditionally dependent on the presence of people on the street. In order to analyse this hypothesis, the conditional probability $p(\text{pixels classified as } \textit{'road'} | \text{people on the street detected})$ has to be determined for the source dataset (GTA) and the target dataset (Cityscapes). For

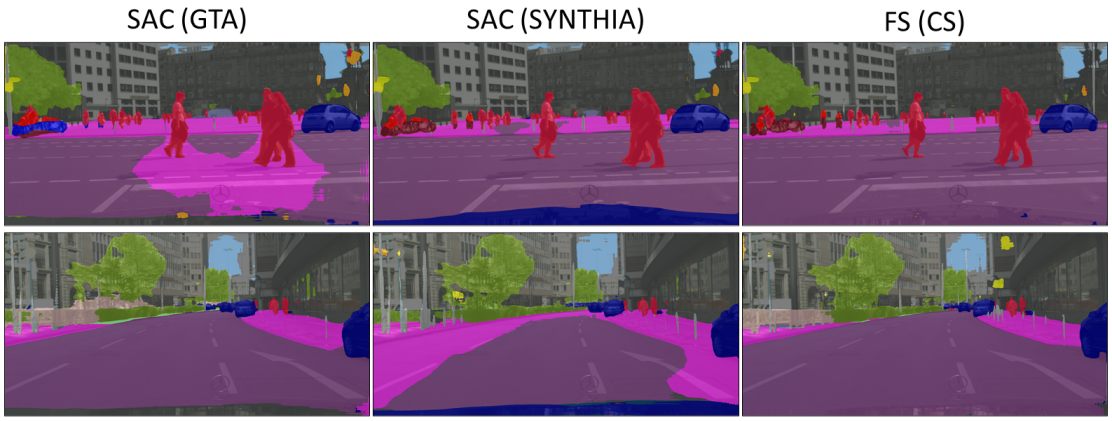


Figure A.6. Domain Gap (Example 2)

this, an analysis of average spatial proximity of the labels *'road'* and *'person'* is necessary. Using this analysis, the question, to what extent conditional probabilities are learned in the source domain should be answered.
The labels *'road'* and *'person'* are in spatial proximity, if there is an edge between the two labels in the segmentation map.

```
def get_connected_labels(label):
    '''
        args:
            label [B,C,H,W]
    '''
    m = torch.nn.MaxPool2d(kernel_size=3, stride=1, padding=1)
    label_max = m(label)    # maxpool
    label_min = -m(-label)  # minpool
    connected_labels = label_max * NUM_CLASSES + label_min
    return connected_labels.flatten().bincount(minlength=NUM_CLASSES**2) \
                           .reshape(NUM_CLASSES, NUM_CLASSES)
```

Listing A.2 Connected Labels

Listing A.2 presents a simple algorithm, for finding connected labels in a segmentation map. It works by applying a *maxpool* and a *minpool* operation to every pixel using a 3×3 kernel. This results in a tuple containing the class with the smallest ordinal value and the biggest ordinal value in the 3×3 proximity of every pixel.
Running this evaluation over all three datasets, it becomes visible, that the number of

edges between the labels *'road'* and *'person'* are rare for the GTA dataset. The Tables A.2, A.3 and A.4 show evaluations of the algorithm for 2000 images from each dataset.

	road	sidewalk	building	wall	fence	pole	traffic light	traffic sign	vegetation	terrain	sky	person	rider	car	truck	bus	train	motorcycle	bicycle
road	370708426	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
sidewalk	4120805	60778834	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
building	227369	395215	152216001	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
wall	274692	425724	521378	18337884	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
fence	67526	56372	721014	172091	4251630	-	-	-	-	-	-	-	-	-	-	-	-	-	-
pole	192949	272312	2844699	134578	54203	6897025	-	-	-	-	-	-	-	-	-	-	-	-	-
traffic light	4587	1854	361387	7088	2363	148290	878865	-	-	-	-	-	-	-	-	-	-	-	-
traffic sign	7887	2709	90563	7418	8187	17253	1555	741096	-	-	-	-	-	-	-	-	-	-	-
vegetation	187260	190495	7048009	611521	405971	1301586	93461	38171	62071705	-	-	-	-	-	-	-	-	-	-
terrain	631749	1346104	468698	274306	120148	307376	12889	6830	1473834	19714508	-	-	-	-	-	-	-	-	-
sky	29031	9632	3396614	148886	261247	2706511	155970	52124	9000066	288053	133228239	-	-	-	-	-	-	-	-
person	**100112**	251386	395355	75428	28723	24858	412	474	90902	30852	7726	2679766	-	-	-	-	-	-	-
rider	14081	2526	8378	2174	577	1194	20	207	7850	3772	2170	1708	238640	-	-	-	-	-	-
car	1338846	175729	436436	127760	45379	63057	1250	4009	187563	63917	20261	32140	4289	19346313	-	-	-	-	-
truck	260922	33780	228522	28583	24774	43878	3229	5738	106608	25848	68593	11222	1494	73095	10276883	-	-	-	-
bus	48321	3933	39334	4513	1897	5992	416	1073	18138	3901	13087	1913	970	11648	2991	3089433	-	-	-
train	19965	2081	19435	3761	9532	11199	1309	2246	3504	1520	16638	236	4	2218	97	118	1067116	-	-
motorcycle	44907	9591	3866	2225	491	340	-	27	2580	1511	80	565	30267	4392	564	344	-	233066	-
bicycle	37265	2110	220	110	133	242	-	-	1410	13439	-	284	7129	10	-	-	-	-	55849

Table A.2. Pixel count for classes in 3×3 neighborhood (2000 GTA labels)

	road	sidewalk	building	wall	fence	pole	traffic light	traffic sign	vegetation	terrain	sky	person	rider	car	truck	bus	train	motorcycle	bicycle
road	183370857	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
sidewalk	3844969	190820276	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
building	175226	2156585	294598699	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
wall	30120	261913	149598	2587900	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
fence	874587	236896	41035	1703	1789252	-	-	-	-	-	-	-	-	-	-	-	-	-	-
pole	815232	1808165	3202331	31465	13789	6840098	-	-	-	-	-	-	-	-	-	-	-	-	-
traffic light	14754	13745	133222	28	87	28406	291865	-	-	-	-	-	-	-	-	-	-	-	-
traffic sign	53693	65767	206523	725	1334	78693	869	736214	-	-	-	-	-	-	-	-	-	-	-
vegetation	2220093	4598731	9710152	88924	57432	915296	25053	70181	96914982	-	-	-	-	-	-	-	-	-	-
terrain	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
sky	5060	191	3583327	23	-	523137	15001	8114	1520754	-	68597909	-	-	-	-	-	-	-	-
person	**5128697**	10138647	3059609	142005	95992	305434	2726	37331	906030	-	24	34536762	-	-	-	-	-	-	-
rider	668875	486249	236446	9668	22819	28587	147	3734	132291	-	1	229711	3306442	-	-	-	-	-	-
car	2615462	886137	347026	33467	95748	237988	1676	16198	831107	-	857	607388	161064	40934191	-	-	-	-	-
truck	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
bus	375867	115939	337087	11486	9132	132180	1965	13010	520879	-	9815	146657	45033	95698	-	14226459	-	-	-
train	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
motorcycle	490837	209167	82604	76	17511	14418	119	1180	70591	-	1	87049	246425	39153	-	3288	-	1626686	-
bicycle	1926539	1337254	141891	4149	12930	32266	51	1261	75468	-	3	132064	475096	34349	-	5449	-	10620	573791

Table A.3. Pixel count for classes in 3×3 neighborhood (2000 SYNTHIA labels)

	road	sidewalk	building	wall	fence	pole	traffic light	traffic sign	vegetation	terrain	sky	person	rider	car	truck	bus	train	motorcycle	bicycle
road	237668420	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
sidewalk	2098845	36683792	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
building	39631	510796	142032635	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
wall	23336	137209	53102	4296663	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
fence	35718	115882	128347	81259	5739617	-	-	-	-	-	-	-	-	-	-	-	-	-	-
pole	112191	481257	2143967	67440	69022	5147545	-	-	-	-	-	-	-	-	-	-	-	-	-
traffic light	100	116	268529	1456	1337	90126	1018255	-	-	-	-	-	-	-	-	-	-	-	-
traffic sign	28980	7911	577106	6239	23066	97077	10670	3032408	-	-	-	-	-	-	-	-	-	-	-
vegetation	90078	216747	4131806	141948	209101	1485069	164751	329055	100903522	-	-	-	-	-	-	-	-	-	-
terrain	346220	341339	51627	29847	37562	94216	938	5121	193875	6808368	-	-	-	-	-	-	-	-	-
sky	139	79	821649	200	2915	285287	26270	24537	1159519	83	23268207	-	-	-	-	-	-	-	-
person	**328513**	369877	775067	45704	52124	103857	1294	7149	172169	20851	2960	6961771	-	-	-	-	-	-	-
rider	22512	15201	72810	4222	5518	6593	272	1377	35122	3012	218	9645	696069	-	-	-	-	-	-
car	1266887	230257	901194	40566	48563	171231	1981	20827	475633	61309	1872	135802	24927	43179989	-	-	-	-	-
truck	20646	4152	27968	616	1667	6146	315	1751	20150	695	2318	7283	888	18073	1067134	-	-	-	-
bus	23811	4575	28144	405	1723	12562	580	2554	39691	1420	1261	8097	1207	15169	558	1357394	-	-	-
train	15658	2805	29663	291	2725	14078	1192	2823	14990	724	2949	11211	438	7249	-	253	1744002	-	-
motorcycle	18019	25243	38700	638	3020	11868	-	246	8934	4013	10	6925	14942	13416	-	203	844	644726	-
bicycle	92495	179696	175299	11468	29654	68289	3	1830	45216	23347	-	58744	97169	48663	642	360	757	10343	2451542

Table A.4. Pixel count for classes in 3×3 neighborhood (2000 CS labels)

Though the pixel distribution over all classes in Table A.1 is a good indicator, for which classes may become underrepresented in the target domain, some special cases require further analysis. The algorithm in Listing A.2 provides a simple method to evaluate, which conditional probabilities could be learned from the source data.
Since no labels of target data are available in unsupervised domain adaptation, a compar-

ison between the datasets using this method is not possible. For semi-supervised domain adaptation, an approach, which includes adaptation for those conditional probabilities could be implemented.

Acronyms

ABN Adaptive Batch Normalization. 30, 31, 35, 55, 56, 58, 64

ADDA Adversarial Discriminative Domain Adaptation. 40, 74

BankCL Bank Contrastive Learning. 63

CAT Cluster Alignment with a Teacher. 75

CC Connected-Component. 41, 42, 45, 60

CM Confusion Matrix. 48, 59

CNN Convolutional Neural Network. 12, 33, 42, 43, 63

CS Cityscapes. 62, 79, 82

CUDA Compute Unified Device Architecture. 41

DeepLab Deep Labeling. 20, 21, 55, 56, 63

DistCL Distribution Contrastive Learning. 63

FCN Fully Convolutional Network. 21

FDA Frequency Domain Adaptation. 24

FDIT Frequency Domain Image Translation. 24, 25, 65, 66

FRL Feature Representation Learning. 5, 33, 34, 35, 36, 38, 39, 41, 47, 49, 51, 52, 55, 56, 57, 58, 59, 61, 62, 63, 64, 65, 74, 79

FS Fully Supervised. 33, 58, 80

GAN Generative Adversarial Network. 24, 25, 26, 76

GB Gigabyte. 57

GPU Graphical Processing Unit. 41, 57

GTA Grand Theft Auto. 5, 19, 21, 33, 55, 56, 58, 63, 64, 65, 79, 80, 82

iid. independent and identically distributed. 24

InfoNCE Information Noise-Contrastive Estimation. 36

IoU Intersection over Union. 57, 58

mIoU mean Intersection over Union. 5, 57, 58, 59, 63, 65, 79

MNIST Modified National Institute of Standards and Technology. 40

PC Principal Component. 16

ProtoCL Prototype Contrastive Learning. 63

ReLU Rectified Linear Unit. 12, 26

ResNet Residual Neural Network. 20, 21, 57, 63

RevGrad Gradient Reversal. 40, 74, 75

RGB Red-Green-Blue. 20, 38, 49

SAC Self-Supervised Augmentation Consistency. 7, 30, 31, 32, 33, 34, 47, 55, 56, 57, 58, 59, 62, 63, 64, 65, 79, 80

SePiCo Semantic-Guided Pixel Contrast. 26, 63

SNE Stochastic Neighbor Embedding. 61, 62, 64

SVHN Street View House Numbers. 40

SYNTHIA SYNTHetic collection of Imagery and Annotations. 80, 82

VGG Visual Geometry Group. 21, 55, 56, 58, 61

YOLO You Only Look Once. 20