University of Stuttgart
Institute for Signal Processing and System Theory
Professor Dr.-Ing. B. Yang

Bachelor Thesis S1419

Input or Output Space Alignment? A Comparative Study for Continual Unsupervised Domain Adaptation in Semantic Segmentation

Angleichen der Pixel- oder Vorhersageräume? Eine Analyse zur Kontinuierlichen Unüberwachten Domänenanpassung für die Semantische Segmentierung

Author: Tim Lindenau

Date of work begin: April 1, 2022
Date of submission: August 21, 2022

Supervisor: Robert Marsden

Keywords: Continual Unsupervised Domain Adaptation, Continual Learning, Unsupervised Domain Adaptation, Semantic Segmentation, Neural Style Transfer, Adversarial Learning

Abstract

Developing fully autonomous driving vehicles is a key ambition of all major automotive producers for the current decade. One of the biggest challenges to achieve this goal is the precise and reliable identification of objects in front and around the vehicle, e.g. streets, cars, and people. Semantic Image Segmentation can provide excellent results for this task but requires a high effort for manually labeling the required training data, i.e. identifying all relevant objects in several thousand images. As the quality of semantic image segmentation decreases significantly when the environment changes (e.g. different weather conditions), this task of labeling data would have to be repeated for many different environments, thus multiplying the manual effort.

The reason for the performance degradation is the distribution shift in the data between the environment for which the segmentation network was trained (labeled data) and the environment it is applied in. Unsupervised domain adaptation (UDA) can solve this problem. Requiring only labeled training data from one source environment and unlabeled data from a different, so-called, target environment, UDA helps to overcome the distribution shift and improves segmentation performance for the target environment. However, for use in autonomous driving, adapting to one target environment is not sufficient. Rather, good performance is required for multiple target environments, for which the unlabeled training data generally becomes available sequentially. Continual learning (CL) allows the segmentation model to adapt to new target environments without forgetting information about previous ones. The combined application of both concepts – UDA and CL – is called continual unsupervised domain adaptation (continual UDA).

In this work, we compare five frameworks for continual UDA. Three of them use style transfer to align data distributions in the input space (CACE, C-WCT^2, C-CMD). The other two align distributions in the output space using adversarial learning (ETM, MuHDi). The first style-transfer-based method is CACE, an already established framework for continual UDA. Keeping the basic structure of CACE, but integrating two different style transfer algorithms (i.e. WCT^2 and CMD) that are expected to overcome some limitations of the CACE style transfer algorithm (e.g. they preserve more details, thus being more photorealistic), we created two new frameworks for style-transfer-based continual UDA. The last two frameworks ETM and MuHDi, which align distributions in output space, mainly diverge in the way they facilitate continual learning.

Our experiments delivered two results. (1) Even though using better style transfer algorithms, the two new frameworks C-WCT^2 and C-CMD did not achieve better performance than the established CACE framework. As both C-WCT^2 and C-CMD additionally suffer from limitations in real-world application, we consider CACE the superior method. (2) Input-space alignment (CACE) delivered superior results than output-space alignment (ETM, MuHDi). In particular, both output-space-aligned frameworks have difficulties with continual learning and are not able to sufficiently prevent forgetting when adapting to new environments.

Kurzzusammenfassung

Die Entwicklung autonom fahrender Fahrzeuge ist eines der Hauptziele aller großen Automobilhersteller. Eine zentrale Herausforderung, um dies zu erreichen, ist die präzise und zuverlässige Erkennung von Objekten vor und um das Fahrzeug, z.B. Straßen, Autos und Menschen. Die semantische Bildsegmentierung liefert gute Ergebnisse für diese Aufgabe, erfordert jedoch einen hohen Aufwand für das Erstellen beschrifteter Trainingsdaten für das Segmentierungsnetzwerk. Da die Qualität der semantischen Bildsegmentierung deutlich abnimmt sobald sich die Umgebung ändert (z.B. unterschiedliche Wetterbedingungen), müssten diese Trainingsdaten prinzipiell für alle möglichen Umgebungen erstellt werden. Praktisch ist das nicht möglich, da es eine Vervielfachung des Aufwandes bedeuten würde.

Der Grund für die Leistungsverschlechterung in einer Umgebung die nicht den Trainingsdaten entspricht, ist eine Verschiebung der zugrundeliegenden Verteilung der Eingangsdaten des Segmentierungsnetzwerkes. Die unüberwachte Domänenanpassung (UDA) hilft dieses Problem zu überwinden. Mit beschrifteten Trainingsdaten für nur eine Umgebung (Quelle) und nicht beschrifteten Daten für eine andere Umgebung (Ziel), ist UDA in der Lage die Verteilungsverschiebung auszugleichen und somit gute Segmentierungsleistungen auf der Zielumgebung zu erzielen. Für den Einsatz im autonomen Fahren ist die Anpassung an nur eine Zielumgebung jedoch nicht ausreichend. Vielmehr ist eine gute Leistung für verschiedene Zielumgebungen erforderlich, für welche die Trainingsdaten in der Regel erst nacheinander verfügbar werden. Kontinuierliches Lernen ermöglicht dem Segmentierungsnetzwerk, sich an neue Zielumgebungen anzupassen, ohne dabei Wissen über frühere Umgebungen zu vergessen. Die kombinierte Anwendung beider Konzepte – UDA und kontinuierliches Lernen – wird als kontinuierliche unüberwachte Domänenanpassung (kontinuierliche UDA) bezeichnet.

In dieser Arbeit werden fünf Architekturen für die kontinuierliche UDA verglichen. Die ersten drei verwenden Methoden des Stiltransfers, um die Verteilungen im Eingangsraum (Pixelraum) des Segmentierungsnetzwerkes anzugleichen (CACE, C-WCT^2, C-CMD). Die beiden anderen gleichen die Verteilungen im Vorhersageraum an (ETM, MuHDi). Die erste auf Stiltransfer basierende Architektur ist CACE, ein bereits etabliertes Framework für kontinuierliche UDA. Indem wir die Grundstruktur von CACE beibehalten aber zwei verschiedene Methoden für den Stiltransfer (WCT^2, CMD) integrieren, die gezielt Schwachpunkte des in CACE gewählten Stiltransferalgorithmus überwinden sollen, entwickelten wir zwei neue Architekturen für die kontinuierliche UDA. Die beiden letztgenannten Architekturen ETM und MuHDi, welche die Verteilungen im Vorhersageraum anpassen, unterscheiden sich hauptsächlich in der Art und Weise, wie sie kontinuierliches Lernen ermöglichen.

Die Experimente lieferten zwei Ergebnisse. (1) Trotz besseren Stiltransferalgorithmen konnten die beiden neuen Frameworks C-WCT^2 und C-CMD keine besseren Ergebnisse als das etablierte CACE Framework erzielen. Da sowohl C-WCT^2 als auch C-CMD zusätzlich unter Einschränkungen leiden die ihre praktische Anwendung erschweren, halten wir

CACE für die überlegene Methode. (2) Distributionen im Eingangsraum anzupassen (CACE) lieferte bessere Ergebnisse als der zweite Ansatz, die Distributionen im Vorhersageraum anzupassen (ETM und MuHDi). Dabei war besonders auffällig, dass ETM und MuHDi unter großen Problemen mit dem kontinuierlichen Lernen litten. So konnten sie das Vergessen bei der Anpassung an neue Umgebungen nicht ausreichend verhindern.

Contents

Summary of Abbreviations

Abbreviation	Meaning
UDA	Unsupervised Domain Adaptation
ML	Machine Learning
NN	Neural Network
CF	Catastrophic Forgetting
CL	Continual Learning
CNN	Convolutional Neural Network
FCNN	Fully Convolutional Neural Network
DNN	Deep Neural Network
CRF	Conditional Random Field
GAN	Generative Adversarial Network
ASPP	Atrous Spatial Pyramid Pooling
NST	Neural Style Transfer
MMD	Maximum Mean Discrepancy
AdaIN	Adaptive Instance Normalization; also a style transfer algorithm
WCT	Whitening and Coloring Transformation; also a style transfer algorithm
CMD	Central Moment Discrepancy; also a style transfer algorithm
PR	Perfect Reconstruction
BCE	Binary Cross Entropy Loss
IoU	Intersection over Union

Summary of Used Symbols

Symbol	Meaning
$x \in \mathbb{R}$	Scalar number
$\underline{x} \in \mathbb{R}^{M}$	Column vector of length M
$\boldsymbol{X} \in \mathbb{R}^{M\times N}$	Matrix or 2nd-order tensor
$\boldsymbol{X} \in \mathbb{R}^{M\times N\times L}$	3rd-order tensor
$\boldsymbol{X}$	General notation for a tensor
$\mathcal{L}$	General symbol for a loss
M	General symbol for a neural network
$f(x)$	Output of a neural network for input x
$I \in \mathbb{R}^{3\times H\times W}$	RGB image
I_c, I_s, I_o	Content, style, and output image
H / W	Height and width of e.g. image, segmentation map, or feature map
$\mathbf{F}^l \in \mathbb{R}^{K_l\times H\times W}$	Feature maps in layer l. If the layer is clear from the context or irrelevant, the layer might be omitted.
$\boldsymbol{F}^l \in \mathbb{R}^{K_l\times HW}$	Vectorized feature maps in layer l. Beware the change of style compared to the non vectorized feature maps.
$\boldsymbol{F}^l_i \in \mathbb{R}^{H\times W}$	i-th feature map in layer l
$\underline{F}^l_i \in \mathbb{R}^{HW}$	Vectorized i-th feature map in layer l
$\boldsymbol{G}^l \in \mathbb{R}^{K_l\times K_l}$	Gram Matrix of the feature maps in layer l
K_l	Number of feature maps in layer l of a CNN
N_l	Number of elements in each feature map of layer l
$\mathcal{F}$	General symbol for a distribution
$\mathcal{D}$	General symbol for a domain, e.g. target domain or source domain
C	Number of classes used for segmentation
$\boldsymbol{S} \in \mathbb{R}^{C\times H\times W}$	Segmentation map
$\boldsymbol{Y} \in \mathbb{R}^{C\times H\times W}$	Soft predictions of a segmentation network (softmax probability)

1. Introduction

1.1. Motivation

Developing fully autonomous driving vehicles is a key ambition of all major automotive producers for the current decade. One of the biggest challenges to achieve this goal is the precise and reliable identification of objects in front and around the vehicle, e.g. streets, cars, and people. This task is of the highest relevance because it drives all further driving decisions like stopping, braking, accelerating, and turning.

Semantic image segmentation can provide excellent results for pixel-wise object identification with high manual effort for image labeling

From a machine learning (ML) perspective, the perception task is solved with semantic image segmentation [1] by assigning each pixel in an image to one of many classes like street or car. This works well and accuracies[1] of around 85% are possible [2].

State-of-the-art approaches for semantic segmentation are based on deep convolutional neural networks (CNN) [3, 4, 5]. The CNN is trained in an end-to-end manner using images of traffic situations in which all relevant objects have been manually identified (fully labeled data).

Many of such images are required to achieve good performance. This poses a problem since providing a set of labeled training data for the network requires high effort. Labeling a single image currently takes on average about 90 minutes [6]. Thus, even for a small dataset of 10,000 images, labeling takes 90,000 minutes, 15,000 hours, or 625 days, i.e. three person-years.

Different environments multiply the required labeling effort for semantic segmentation

Semantic segmentation provides excellent results when the vehicle is in an environment similar to the environment of the training data. However, in reality, this is mostly not the case. A car experiences many different environments (e.g. different times of day, different weather conditions, or different countries) and good segmentation performance is required for all of them.

When the testing/application environment diverges from the one used for training, the accuracy of predictions drops [7]. This is because in general neural networks react sensitively to changes in the distribution of input data, i.e. the data distribution for training is different from the one for testing. A shift in the environment means exactly such a shift to the data distribution, as for example, colors may shift to a different color tone.

Thus, to guarantee good performance for all environments, it would be required to train the neural network with training data from all environments. With the previously described effort

[1]With accuracies we mean the common metric for semantic segmentation mean intersection-over-union.

and complexity for creating labeled images, this is not possible and different solutions must be found that reduce the required amount of data.

Key challenge: Deliver good results for multiple environments when labeled training data is available only for one environment

The key challenge is to provide good results for multiple environments (target environments) when labeled data is only available for one environment (source environment). For the target environments, no training data with labeled images is available, however, non-labeled images exist as these are far easier to obtain. It is further assumed that the unlabeled training data for the different target environments become available sequentially over time. Only the labeled source data is always available.

Solving the challenging task of achieving good performance in multiple different environments requires two foundational concepts: unsupervised domain adaptation and continual learning.

Unsupervised domain adaptation can reduce the distribution discrepancy between the source and the target environment

The first sub-task is to provide good performance in one target environment which is different from the source environment. Unsupervised domain adaptation (UDA) is one effective strategy to achieve good performance for a target environment when only unlabeled data is available for this environment, but labeled data is available for the source environment.

The idea in UDA is that if there is a performance drop for data from a target environment because its distribution is different from that of the source environment, the performance can be improved by minimizing this distribution discrepancy.

Multiple approaches exist to achieve distribution alignment. (1) Aligning distributions in a latent feature space [8], (2) aligning distributions in the output space of the segmentation network [8], or (3) aligning distributions in the input space (pixel space) before the segmentation network [9, 10]. The details of these approaches are described in further chapters.

Continual learning helps to retain knowledge about previous environments when adapting to new ones

While UDA can be efficiently used to improve performance for one target environment, in reality, there are multiple different target environments that all require good segmentation performance. For these different environments, the training data are generally not available at the same time but instead become available sequentially. For example, unlabeled training data for different countries become available over time due to the effort and time required to create them; the same is true for data for winter and summer.

For later application in driving, it is important that the segmentation network preserves information from all previously learned environments because when making predictions, the environment constantly changes and also repeats itself. For example, the day/night shift, winter/summer shift, or vacation trips in different countries are all experienced multiple times.

It is therefore required that the NN can adapt to a new target environment whenever the respective training data becomes available, while simultaneously preserving the learned knowledge about previous environments.

The simplest approach of just using UDA to adapt to new target environments one after another does not work because doing so, the segmentation network cannot preserve information about previous target environments when adapting to the new one. This effect is typically called catastrophic forgetting (CF) [11], which describes that whenever a NN is trained for a sequence of tasks (e.g. sequence of target environments), the performance of previous tasks decreases with every new one.

Continual learning (CL) aims to mitigate the issue of forgetting, when using UDA to sequentially adapt to new target domains, by adding extra measures within each adaptation step. Options to prevent forgetting are: (1) regularization approaches that limit changes done to the network between different target environments [12], (2) dynamically expanding the network with every new target environment [13], or (3) replay-based approaches [14]; in replay, some samples of previous target environments are stored in memory and replayed whenever adapting to a new target environment. Like for UDA, the details will also be explained in later chapters.

The combined application of UDA and CL to achieve good performance under different environments is called continual unsupervised domain adaptation (continual UDA).

1.2. Problem Description

Several approaches for continual UDA exist. This work aims to compare five different frameworks with respect to their capability to sequentially adapt to multiple environments.

In the context of continual UDA, these different environments are typically called domains (e.g. source domain and target domain). Given a labeled source domain and multiple unlabeled target domains, each continual UDA framework is tasked to sequentially adapt to all target domains while preventing catastrophic forgetting. The final evaluation criterion is average segmentation performance in the metric of mean intersection-over-union for all target domains and the source domain. Also analyzing the performance for the source domain increases the challenge of our experimental setup compared to others like [13, 15]. Including the source domain in the analysis makes sense, because generally, it is also a relevant environment for later driving application. For example, the source domain might be images taken in sunny weather, while the target domains treat other weather conditions like fog or rain.

Three of the compared frameworks are based on distribution alignment in input space and a replay mechanism to prevent forgetting. The other two frameworks align distributions in output space and are using different approaches for continual learning.

In the context of semantic segmentation, aligning in **input space** is achieved with so-called style transfer. Given a content image and a style image, the task of style transfer is to generate an output image with the content of the original content image and the style of the style image. Using style transfer to align distributions in input space is possible by transferring a labeled source image into the style of the target domain. By doing so, the content and therefore the labels do not change, but the image distribution shifts closer to that of the target domain. Thus, the style transferred image can be used to train the NN for good performance on the target domain [16].

The first style-transfer-based method is CACE [10], an already established framework for continual UDA. Besides the style transfer algorithm used in CACE, different alternative algorithms exist that address some known weaknesses of the former one. For example, WCT^2 [17], which enhances photorealism, or CMD [18], for which there is theoretical reasoning that it creates more accurate stylizations. Starting from the CACE framework, two additional style-transfer-based frameworks for continual UDA are created. These frameworks implement a structure similar to that of CACE, but diverge in the choice of style transfer algorithm. Inspired by the name of the style transfer algorithm, we call these new frameworks continual WCT^2 (C-WCT^2) and continual CMD (C-CMD).

Aligning distributions in **output space** is typically achieved in an adversarial manner, where a discriminator is trained to differentiate between the source and target domain. The segmentation network instead tries to trick the discriminator by making its predictions for both domains indistinguishable [8].

The two frameworks using output space alignment – ETM [13] and MuHDi [15] – are based on such an adversarial learning scheme. They diverge in the way they aim to prevent catastrophic forgetting: ETM dynamically expands the segmentation network with every target domain, while MuHDi uses advanced regularization techniques.

Figure 1.1 summarizes the key commonalities and differences of all five frameworks. A detailed explanation of the five frameworks – including their mathematical background – will be given in Section 3.

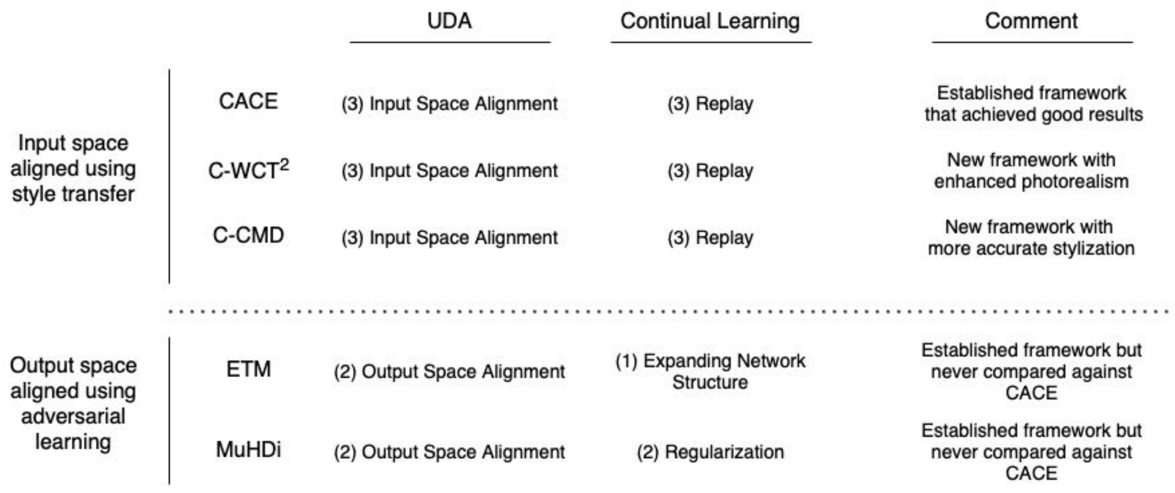
When comparing all five frameworks, we are primarily interested in answering two research questions. First, how does the choice of style transfer algorithm affect performance in continual UDA? Second, what is the superior approach for continual UDA, style-transfer-based input space alignment or adversarial-learning-based output space alignment?

Question 1: How does the choice of style transfer algorithm affect performance in continual UDA?

All style-transfer-based frameworks (CACE, C-WCT^2, C-CMD) implement the same basic structure and diverge only in the style-transfer algorithm used to align distributions. Comparing the performance of CACE with its adaptations that implement WCT^2 and CMD as their style transfer algorithms, we want to answer the question of how the choice of style transfer algorithm affects continual UDA performance. In particular, we are interested if there are performance advantages from the enhanced photorealism of WCT^2 or the theoretically more accurate stylization of CMD.

Question 2: Which approach for continual UDA, style transfer based input space alignment or adversarial output space alignment, is more effective?

Style-transfer-based approaches for an alignment in input space and adversarial-learning-based approaches for an alignment in output space have not often been compared with one another. By comparing them for multiple challenging sequences of domains, we want to identify the superior approach. Two sub-questions are of special interest: 2a) is there a noticeable difference in how well input and output space alignment can adapt to new target domains? 2b) how well do input space alignment and output space alignment perform in the continual setting, i.e. can CF be successfully prevented using both approaches?



		UDA	Continual Learning	Comment
Input space aligned using style transfer	CACE	(3) Input Space Alignment	(3) Replay	Established framework that achieved good results
	C-WCT^2	(3) Input Space Alignment	(3) Replay	New framework with enhanced photorealism
	C-CMD	(3) Input Space Alignment	(3) Replay	New framework with more accurate stylization
Output space aligned using adversarial learning	ETM	(2) Output Space Alignment	(1) Expanding Network Structure	Established framework but never compared against CACE
	MuHDi	(2) Output Space Alignment	(2) Regularization	Established framework but never compared against CACE

Figure 1.1.: Overview of the five frameworks for continual UDA that are compared in this work.

1.3. Course of Action

We begin this work by first laying the necessary **theoretical foundations** with the first part being a revisitation of semantic image segmentation, unsupervised domain adaptation, and continual learning.

The second part focuses on style transfer as it is one effective way for UDA at input level. We explain the basic idea behind style transfer and give a detailed look at the two state-of-the-art algorithms WCT^2 and CMD.

Then we present the **five frameworks for continual UDA**. We explain the structure of each framework as well as its theoretical background.

In the **methods section** we motivate and shortly describe the two experiments that we performed to compare the five different frameworks and to answer the two research questions.

The **first experiment** focuses on style transfer and visually compares the quality of stylization of the three different style transfer algorithms that are used in the three different frameworks for style-transfer-based continual UDA. These three algorithms are WCT^2, CMD, and the style transfer algorithm that is used in CACE.

The **second experiment** then compares the five different frameworks for their performance in continual UDA by collecting the performance values of every framework for three different sequences of domains.

In the **discussion section** we use the results from both experiments to answer the two previously asked research questions.

Finally, we end this work by **concluding** all relevant results.

2. Theoretical Fundamentals

2.1. Semantic Image Segmentation

Semantic image segmentation is the task of assigning a class to every pixel in an image based on its semantic content. This task is one of the most challenging problems in ML but very important for all applications where an advanced scene understanding is required (e.g. autonomous driving).

Before the rise of deep learning, segmentation methods were often based on conditional random fields (CRF). CRFs make use of the property that neighboring pixels in an image are typically in some relationship and model the conditional probability $P(Y|X)$; Y being the predictions and X being the input [19]. Remarkable progress outperforming CRFs has been achieved using deep neural networks (DNN), or more specifically, fully convolutional deep neural networks (FCNN). In their seminal work Long et al. [3] have shown that well-performing segmentation networks can be created from existing convolutional neural networks (CNN) for classification (e.g. VGG [20] or ResNet [21]) by replacing the dense classification layers with upsampling or deconvolution layers[1]. Compared to other methods, these fully convolutional networks bring several advantages. They work for arbitrary input sizes and do not require manual feature extraction but instead allow training just from images and their labels in an end-to-end manner.

Numerous improvements to the architecture of these networks have been proposed since the initial work of [3]. For example, exchanging the pooling operations for atrous/dilated convolutions to avoid the usage of upsampling layers [4], and based on that even a completely new structure for making predictions called atrous spatial pyramid pooling (ASPP). Here, the prediction quality is improved by combining the outputs of multiple atrous convolutions at different rates, thus capturing objects at different scales [5].

Both these improvements have first been proposed by the team behind deeplab, whose DeepLab-v2 [5] network provides state-of-the-art segmentation performance and will be used for all segmentation tasks throughout this work. In short, DeepLab-v2 consists of a ResNet-101 [21] as the feature extractor and an ASPP module as the classifier head. A schematic overview of the structure is given in Figure 2.1.

Unfortunately, the superior performance of the FCNN-based approaches does not come without its cost. Due to their large model capacity, these models require a large amount of annotated data for training. Collecting this data is typically a major problem requiring a lot of effort. Some strategies have been proposed to reduce the number of full annotations required to train the NN. One of the most important ones is unsupervised domain adaptation (UDA) which will be explained in detail in the next section.

[1]Also known as transposed convolution.

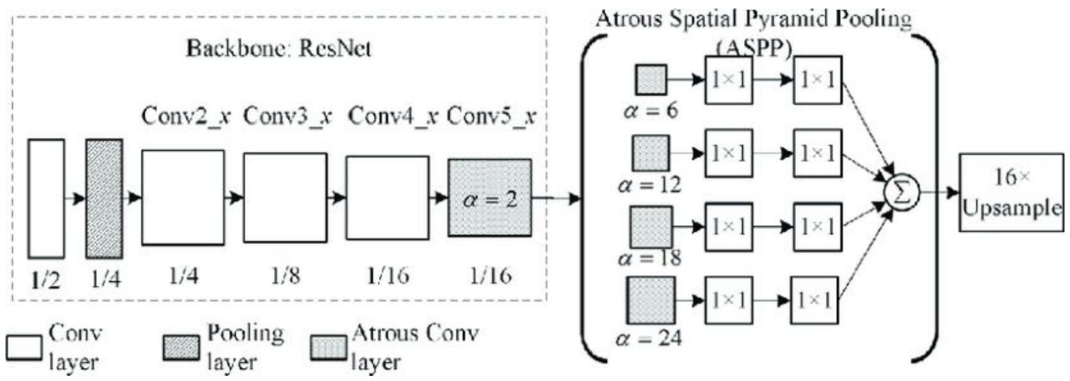


Figure 2.1.: Overview of the architecture of DeepLab-v2 consisting of the ResNet-101 as the feature extractor and an ASPP head for classification. Illustration taken from [22].

2.2. Unsupervised Domain Adaptation

In machine learning it is typically assumed that training and test samples are independent and drawn from the same distribution (independent and identically distributed). This simplification does not always hold, for example, when 'a model is trained in one or more domains (called source domains) and then applied in another different but related domain (called target domain)' [16, p.3]. One practical example is a semantic segmentation model for autonomous driving, trained on images during sunshine but evaluated at night.

Most NNs react sensitively to distribution changes which break the assumption of identically distributed data [7]. Thus, if the distribution of the training data is different from the distribution of the testing data, there is a large drop in performance. It is this drop in performance that makes it practically impossible to apply a model trained on the source domain for data from the target domain.

A simple solution to overcome the drop in performance would be to also collect training data from the target domain. However, especially for the use case in semantic segmentation, doing so is oftentimes not possible due to the high cost of labeling data. Therefore, different unsupervised methods (unsupervised domain adaptation) for improving performance on the target domain have been proposed. Requiring only labeled data from the source domain and unlabeled data from the target domain, these methods improve target domain performance by minimizing the discrepancy between the source and target distribution.

In the following, the three most important types of UDA will be explained briefly.

Adaptation at Input Level: The goal is to find a content preserving transformation at the pixel level that maps source inputs to a given target distribution before training the neural network. This is commonly achieved with style transfer by first style transferring a labeled source image into the style of the target domain before using this image to train the segmentation network. Since the style transfer preserves the content and thus labels of the source image while moving the distribution closer to that of the target domain, it is possible to train the segmentation with the transformed source samples and later apply the network to data from the target domain without a significant loss in performance [16].

Adaptation at Feature Level: A second approach for UDA is to seek for an alignment in the latent feature space by training a feature extractor to discover domain-invariant features. Since these features are similar across domains, even if the classifier is only trained with samples from the source domain, performance on the target should still be good. The task of training the feature extractor to generate domain-invariant features is often achieved in an adversarial manner using Generative Adversarial Networks (GAN) [23]. A discriminator is trained to distinguish different domains just from their features, and the generator (i.e. the feature extractor) tries to trick the discriminator by aligning the features such that no distinction is possible [16]. Although this method has been effective for the task of image classification, it has not been widely used for semantic segmentation. The reason for this lies in the typically much more complex feature space used in segmentation, for which it is difficult to find a suitable feature extractor that only extracts domain-invariant information [8].

Adaptation at Output Level: The previously mentioned problem of dealing with a complex latent space can be avoided when aligning the domains in the much simpler output space. This idea originated in the work of Tsai et al. [8] who argued that using a complex latent space is not necessary because already the output of a segmentation network contains a lot of rich information, both spatially and locally. Even if two images stem from different domains, their segmentation output still shows a lot of similarities (e.g. cars typically drive on streets and people walk on pavements). Therefore, performance on the target domain can be improved if the model is trained to produce similar outputs for both domains. [8] have shown that, for semantic segmentation their method performs well, thus being used in many different applications since [15, 13].

2.3. Continual Learning

After having shown methods to adapt a segmentation network to perform well on one target domain different from the source domain, a second issue arises when considering that in many cases (e.g. autonomous driving) there is not just one target domain but multiple different ones, and good performance is required for all of them.

As already discussed in the introduction, the simplest solution to guarantee good performance on all targets would be to first collect data from all target domains and then simultaneously train the NN on all domains at once. Unfortunately, this approach is not feasible since generally the training data for different domains become available sequentially and storing the data for combined learning is not possible due to memory limitations and privacy restrictions [10, 15].

The solution of using UDA to sequentially adapt to new target domains also does not work because doing so cannot preserve information about previous target domains when adapting to the current one. This property of NNs to forget a lot of information from previous tasks whenever they are trained for a new task is commonly referred to as catastrophic forgetting [11]. To overcome this issue and allow sequential learning, additional measures must be included in the learning stage that limit forgetting.

The field of Continual learning (CL) deals with this challenge of training a NN for a sequence of sequential tasks without forgetting knowledge about proceeding ones. For the problem of classifying images from different domains, CL has already been intensively studied and three approaches have been proven to be effective. First, using regularization to prevent the weights of the network from changing too much between iterations and thus guaranteeing that the network remains similar between different tasks [12]. Second, memory-based approaches in which some information from previous domains is saved and then replayed when training new ones [14]. Lastly, some model-based approaches have been developed in which the model is dynamically expanded with each new domain. The idea here being that the original model learns domain-invariant knowledge, while every extension learns domain-dependent knowledge [24].

Unlike for image classification, the literature on CL in the context of semantic segmentation is still very sparse. [13] uses a model-based approach in which a small target-specific memory is introduced to increase the model capacity for each new domain. [15] uses a regularization-based approach with advanced distillation methods. Finally, a memory-based approach using replay has been investigated in [10, 9]. The specific details of [13, 15, 10] and their integration with UDA will be explained in more detail in Section 3.

2.4. Neural Style Transfer - The Fundamentals

The goal of style transfer is to generate a new image I_o based on two input images I_c and I_s, where the new image has the style of I_s and the content of I_c. Nonparametric methods from the area of texture transfer have been researched for a long time and have shown some good results [25]. However, all of these methods suffer from the fundamental limitation that they only use low-level hand-crafted image features that cannot capture semantic information such as objects and scenery [26]. On the search for a better representation, Gatys et al. [26] have been inspired by CNNs and their ability to extract high-level semantic information from images. Using an optimization-based procedure that simultaneously reduces a style and a content loss based on the features of CNNs, they were able to outperform existing methods and achieve impressive style transfers for a variety of styles and images. On the basis of their work, many new style transfer algorithms using the features of CNNs have been developed. Together these methods form the new class of neural style transfer algorithms (NST).

Due to its importance for the field of NST, the idea behind [26] will be summarized in the next section.

2.4.1. Optimization-Based Style Transfer

CNNs have shown impressive results in computer vision tasks (e.g. image classification and segmentation) because of their ability for hierarchical feature learning. While the first layers of a CNN capture local structures like edges and gradients, with deeper layers the representation becomes more abstract, encoding more complex patterns such as objects and their arrangement [27]. These features can be used to derive an efficient representation of the content and style of an image [26]. They allow for an efficient content representation because capturing content is equivalent to capturing the scene arrangement, omitting fine details. The output of deeper layers of a CNN does exactly this, thus leading to a natural formulation of the content loss between two images $I_\mathrm{c}, I_\mathrm{o}$ as the Euclidean distance between their high-level feature maps

$$\mathcal{L}^l_{\text{content}}(I_\text{c}, I_\text{o}) = \frac{1}{2}\sum_{i=1}^{K_l}\sum_{j=1}^{H\cdot W}\left(F^l_{i,j}[I_\text{c}] - F^l_{i,j}[I_\text{o}]\right)^2, \tag{2.1}$$

where $\underline{F}^l_i[\cdot] \in \mathbb{R}^{HW}$ is defined as the vectorized feature map of the i-th filter in layer l and K_l denotes the number of distinct filters in this layer.

For the style representation it intuitively makes sense that style is not as localized as content and cannot be captured just by comparing individual pixel values. [26] have shown that a good style representation can, however, be found when creating a feature space on top of the CNN space. This space is characterized by the correlation between the different feature maps in the same layer. It can be described using the Gram matrix $\boldsymbol{G}^l \in \mathbb{R}^{K_l \times K_l}$. Each element G^l_{ij} is calculated as the inner product between the vectorized feature maps $\underline{F}^l_i$ and $\underline{F}^l_j$ in layer l

$$G^l_{ij}[\cdot] = \sum_{k=1}^{H\cdot W} F^l_{i,k}[\cdot]F^l_{j,k}[\cdot]. \tag{2.2}$$

Given two images I_s and I_o, the style loss for layer l can be formulated as the Euclidean distance between their Gram matrices

$$\mathcal{L}^l_{\text{style}}(I_\text{s}, I_\text{o}) = \frac{1}{4N_l^2K_l^2}\sum_{i,j}\left(G^l_{ij}[I_\text{s}] - G^l_{ij}[I_\text{o}]\right)^2, \tag{2.3}$$

where $N_l = HW$ is the number of elements in each feature map.

For the complete style loss, multiple layers are combined since different layers carry different style information like stroke and color. This is due to their differences in the receptive field and the abstraction level. Including multiple layers, the total style loss then comes together as a weighted sum of layer-wise losses weighted with the factors w_l

$$\mathcal{L}_{\text{style}}(I_\text{s}, I_\text{o}) = \sum_{l=1}^{L} w_l\,\mathcal{L}^l_{\text{style}}(I_\text{s}, I_\text{o}). \tag{2.4}$$

Using these two losses, transferring the style of an image I_s onto a content image I_c can be formulated as an optimization problem of the output image I_o, where, starting from white noise, I_o is iteratively updated to minimize the weighted sum of content loss and style loss

$$\mathcal{L}_{\text{total}}(I_\text{s}, I_\text{c}, I_\text{o}) = \alpha \cdot \mathcal{L}_{\text{content}}(I_\text{c}, I_\text{o}) + \beta \cdot \mathcal{L}_{\text{style}}(I_\text{s}, I_\text{o}). \tag{2.5}$$

For their implementation, Gatys et al. [26] made use of the VGG-19 [20] network without the dense classification layers as the CNN.[2] The content loss is calculated from the output of layer *conv4_2*. The style loss is calculated from layers *conv1_1*, *conv2_1*, *conv3_1*, *conv4_1* and *conv5_1*, with each layer weighted equally.

[2]Since the task of this CNN is only to extract features, it is often called feature extractor equivalently.

Understanding Neural Style Transfer

While the Gram matrix used by Gatys et al. [26] has shown impressive results for capturing and transferring style, the underlying reason for this has initially been unknown. The first ones to provide a theoretic explanation for this have been Li et al. [28] in their work on 'demystifying neural style transfer'. They have shown that aligning Gram matrices in feature space is equivalent to minimizing the maximum mean discrepancy (MMD) [29] – a common metric to measure the distance between two distributions – of the feature distributions under the use of a second-order polynomial kernel $k(x, y) = (x^T y)^2$.

To understand this statement it must first be defined what a feature distribution is exactly. The layer-wise (non-vectorized) feature maps $\mathbf{F}^l$ themselves are no distribution but just a collection of individual values from different positions (i.e. x, y). Please note that for a better distinction we use upright typesetting to denote the non-vectorized feature maps. The necessary transfer to a distribution is possible by interpreting the respective vectorized feature maps $\boldsymbol{F}^l \in R^{K_l \times HW}$ as a set of $H \cdot W$ samples $\in \mathbb{R}^{K_l}$, all independently drawn from the same multivariate distribution [28]. With this interpretation, the values of the feature maps are just realizations of this underlying distribution, hence the name feature distribution. This also means that the values of $\boldsymbol{F}^l$ (samples) can be used to estimate descriptors of the feature distribution such as mean and variance.

Grounded on the initial result that minimizing the gram-based loss can be understood as a process of distribution alignment, it is natural to ask if other distribution alignment methods can also successfully transfer style. To answer this, [28] minimized the MMD under a choice of different popular kernels:

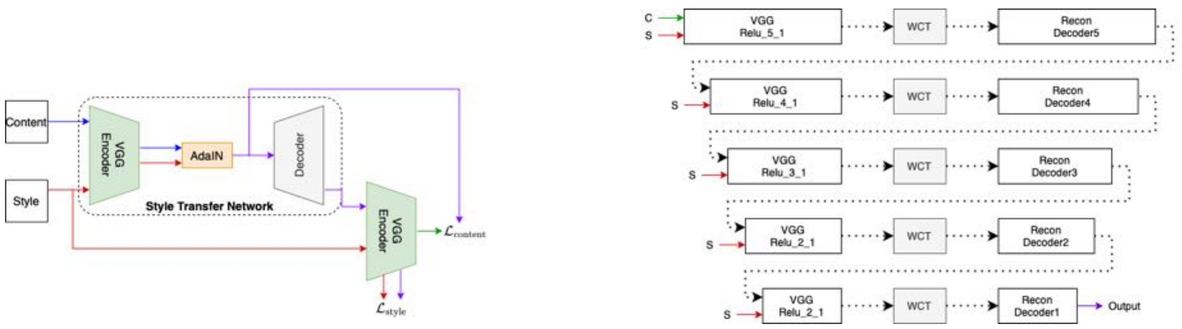
1. Linear kernel: $k(x, y) = x^T y$
2. Polynomial kernel: $k(x, y) = \left(x^T y + c\right)^d$
3. Gaussian kernel: $k(x, y) = \exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right)$

The results were unambiguous with all kernels successfully being able to transfer style, however, each one focusing on different aspects.

Taken together, these results prove that NST can generally be achieved and understood as a process of aligning image feature distributions, where the gram-based loss is one but by no means the only way to achieve this. They also permit for a new interpretation of NST as a domain adaptation problem between content and style image – remember distribution alignment is the core of domain adaptation. Most importantly, however, the results raise the question if there are better alternatives for the gram-based loss which can improve the stylization performance of NST. [18] answers this very question by showing theoretical and practical limitations of the gram-based loss and proposing a new loss function based on higher-order central moments that can overcome these limitations. This new loss will be explained in detail in Section 2.5.1.

2.4.2. Feedforward Style Transfer

Even though the results of optimization-based methods for NST have been impressive, optimization-based methods suffer from one major drawback, their efficiency. Requiring



(a) AdaIN style transfer network (b) WCT multi-level stylization pipeline

Figure 2.2.: (a) Architecture of the AdaIN style transfer network for transferring style (dotted area) and training. The figure is based on [30]. (b) Schematic overview of the multi-level stylization pipeline that is being used in the WCT style transfer network. The figure is based on [31].

hundreds to thousands of iterations to produce good results – which even on modern GPUs can take up to a few minutes – drastically reduces their practical application, especially for time-critical real-time tasks. Therefore, significant efforts were made from the beginning to speed up NST by replacing the slow iterative optimization process with a feedforward structure that requires only one forward pass per image.

AdaIN – Feedforward Style Transfer Using Adaptive Instance Normalization

Early approaches solved the challenge of feedforward style transfer by training one individual network per style, thus trading in generality, meaning the ability to transfer different styles, for speed [32, 33]. The first to successfully solve the question of universal style transfer[3] have been Huang et al. [30]. They have shown that style can be transferred in a single forward pass by matching the channel-wise mean and variance of high-level features in a encoder/decoder architecture.

With the results from Section 2.4.1, this intuitively makes sense, since matching first- and second-order moments is one way to align distributions. Especially assuming a multivariate Gaussian distribution where all feature channels are independent[4], aligning channel-wise mean and variance would even be good enough for a perfect distribution match [34].

To align the mean and variance, [30] introduced the 'Adaptive Instance Normalization' operation (AdaIN) to first normalize each feature map individually (Instance Normalization) and then reapply the mean and variance of the style. Given a content image to be stylized I_c and a style input I_s, the AdaIN operation aligns the channel-wise means and variances of the vectorized feature maps $\boldsymbol{F}[I_\text{c}] \in \mathbb{R}^{K_l \times HW}$ to match $\boldsymbol{F}[I_\text{s}]$. This is achieved by

$$\text{AdaIN}\left(I_\text{c}, I_\text{s}\right) = \sigma\left(\boldsymbol{F}[I_\text{s}]\right) \underbrace{\left(\frac{\boldsymbol{F}[I_\text{c}] - \mu\left(\boldsymbol{F}[I_\text{c}]\right)}{\sigma\left(\boldsymbol{F}[I_\text{c}]\right)}\right)}_{\text{IN}} + \mu\left(\boldsymbol{F}[I_\text{s}]\right),$$

where $\mu(\cdot)$ and $\sigma(\cdot)$ calculate the channel-wise mean and standard deviation.

[3]Style transfers working for an infinite number of arbitrary styles.
[4]This is most likely not given for images.

The full style transfer architecture introduced by [30] consists of a fixed VGG-19 encoder [20] (up to *relu4_1*) as the feature extractor and a symmetric trainable decoder. After encoding both style and content, one AdaIN layer is used to transform the content image feature maps to match those of the style image, before using the decoder to transform back these features to image space.

However, before being able to do so, the decoder must first be trained for image reconstruction. Two different losses are optimized simultaneously to train the decoder [30]. First, a content loss that compares the feature activations of the encoded content image with those of the stylized image after it has been passed through the encoder for a second time. Second, a style loss comparing the feature activations of the style image with those of the re-encoded stylized image. After the training is completed, the decoder is also fixed and arbitrary style transfers can be efficiently achieved in just one single forward pass.

The full AdaIN style transfer architecture for training and application (dotted area) is illustrated in Figure 2.2a.

WCT – Feedforward Style Transfer Using the Whitening and Coloring Transformation

Li et al. [31] came up with a second architecture for universal feedforward NST, which is closely related to that of Huang et al. [30]. The broad structure is the same, consisting of a fixed VGG-19-based encoder and a trainable reconstruction decoder. However, unlike [30], they do not use AdaIN for feature alignment but instead the whitening and coloring transformation (WCT).

The whitening and coloring transformation is a common tool in mathematics to transform a vector of random variables with a known covariance matrix into a random vector with a different specified covariance matrix [35]. For the use case in NST, the goal of WCT is to transform the vectorized feature maps of the content image $\boldsymbol{F}[I_\mathrm{c}] \in \mathbb{R}^{K_l \times HW}$ in such a way that their covariance matrix matches the one of the style image. Two steps are involved to achieve this, i.e. whitening and coloring.

Whitening transform: In the whitening step, the vectorized feature maps of the content image $\boldsymbol{F}[I_\mathrm{c}]$ are transformed to be uncorrelated. This is achieved via

$$\hat{\boldsymbol{F}}[I_\mathrm{c}] = \boldsymbol{E}[I_\mathrm{c}]\,\boldsymbol{D}[I_\mathrm{c}]^{-1/2}\,\boldsymbol{E}[I_\mathrm{c}]^T\;(\boldsymbol{F}[I_\mathrm{c}] - \mu(\boldsymbol{F}[I_\mathrm{c}]))\,, \tag{2.6}$$

where $(\boldsymbol{F}[I_\mathrm{c}] - \mu(\boldsymbol{F}[I_\mathrm{c}]))$ is a centered version of the features subtracting their mean vector, $\boldsymbol{D}[I_\mathrm{c}]$ is a diagonal matrix containing the eigenvalues of the covariance matrix $\boldsymbol{F}[I_\mathrm{c}]\,\boldsymbol{F}[I_\mathrm{c}]^T \in \mathbb{R}^{K_l \times K_l}$, and $\boldsymbol{E}[I_\mathrm{c}]$ is the corresponding matrix of eigenvectors [31].

Coloring transform: In the coloring step, the normalized features $\hat{\boldsymbol{F}}[I_\mathrm{c}]$ are transformed to match the correlations of the style image. This is achieved via

$$\boldsymbol{F}_\mathrm{stylized}[I_\mathrm{c}, I_\mathrm{s}] = \left(\boldsymbol{E}[I_\mathrm{s}]\,\boldsymbol{D}[I_\mathrm{s}]^{-1/2}\,\boldsymbol{E}[I_\mathrm{s}]^T\,\hat{\boldsymbol{F}}[I_\mathrm{c}]\right) + \mu(\boldsymbol{F}[I_\mathrm{s}])\,, \tag{2.7}$$

where the meanings of $\boldsymbol{E}[I_\mathrm{s}]$ and $\boldsymbol{D}[I_\mathrm{s}]$ are the same as before [31].

Transferring style using WCT generally achieves better results compared to AdaIN [31], because WCT is a more powerful method for distribution alignment. This is because in addition to the channel-wise means and variances (diagonal elements of the covariance matrix), WCT also aligns the correlations between feature channels.

In addition to using WCT for distribution alignment, Li et al. [31] also made improvements to the architecture of the style transfer network. As explained in Section 2.4.1, different layers of the encoder capture different characteristics of style. It is therefore desirable to transfer the style not just at one layer – as is the case for the AdaIN architecture – but multiple different ones combined. To this end, Huang et al. [30] proposed a new *multi-level stylization pipeline* as illustrated in Figure 2.2b.

The new style transfer architecture starts with extracting the features at *relu5_1*, applying WCT, and then using a decoder to reconstruct back to image space. The output of this reconstruction is then re-fed into the encoder, this time, extracting the features already at *relu4_1*. After applying WCT for a second time, a second decoder is used to transform the features back to image space. This process is repeated for layers *relu3_1*, *relu2_1*, and *relu1_1*. In following this process, new aspects of the style, from coarse (e.g. colors) to fine (e.g. local details), are iteratively applied to the image, resulting in overall stronger and better stylizations. This improvement in stylization, however, comes at the cost of an increased training complexity, since all five reconstruction decoders must be trained independently.

Finally, both methods (AdaIN, WCT) allow for easy user control over the degree of stylization at runtime. This is achieved by linearly interpolating between the stylized feature maps of the content image and the original ones before decoding

$$\boldsymbol{F}_{\text{final}} = \alpha\, \boldsymbol{F}_{\text{stylized}} + (1-\alpha)\, \boldsymbol{F}_{\text{original}}. \tag{2.8}$$

The mixing coefficient $\alpha \in [0, 1]$ to adjust the strength of stylization can be set by the user.

2.4.3. Class-Conditioning the Style Transfer to Respect Semantic Meaning

All NST algorithms presented so far transfer style image-wide. In many cases, this negatively influences the stylization since some dominant properties of the style can influence the stylization process more than they should. Two examples: (1) when the style image is very dark, the resulting image typically also just turns out uniformly darkened. (2) When there are prominent patterns in the style, those may just be randomly applied to the content image without respecting whether it is semantically logical. As an illustration of the latter, it might happen that the cloudy structure of the sky is projected onto the roof of a house, a transformation that from a logical perspective does not make sense.

To avoid these issues and achieve more realistic stylizations, [10] and [36] proposed to use image-segmentation-maps to only align the style between semantically similar regions of content and style image.

For the algorithm proposed by Gatys et al. [26], the procedure of class-conditioning the style transfer is explained in the following. The same basic procedure is also applicable for feedforward methods and can easily be extended to AdaIN or WCT as shown by [10].

Class-Conditioning for Optimization-Based Style Transfer

Class-conditioning the style transfer requires only one change to the general procedure. The style loss from Equation (2.3) must be conditioned on different classes and calculated individually for all of them. For this, we define $\mathbf{F}^l_c[\cdot] = \mathbf{F}^l[\cdot]\boldsymbol{M}^l_c[\cdot]$ as the (non vectorized) feature maps after masking the original feature maps with the masking matrix $\boldsymbol{M}^l_c$, so that only entries of the class c are included. This means $\mathbf{F}^l_c[\cdot]$ only contains the values for which the pixel at the same position in the original image belongs to class c. Since the image dimensions might be larger than the feature dimensions, the image is downsampled to match the size of the feature map before masking. $\boldsymbol{G}^l_c[\cdot]$ is the respective Gram matrix after masking for class c and by summing over all classes the content-aware style loss for layer l becomes [36]

$$\mathcal{L}^l_{\text{style+}}(I_\text{s}, I_\text{o}) = \frac{1}{C}\sum_{c=1}^{C}\frac{1}{4N^2_{l,c}K^2_l}\sum_{ij}\left(G^l_{c,ij}[I_\text{s}] - G^l_{c,ij}[I_\text{o}]\right)^2. \tag{2.9}$$

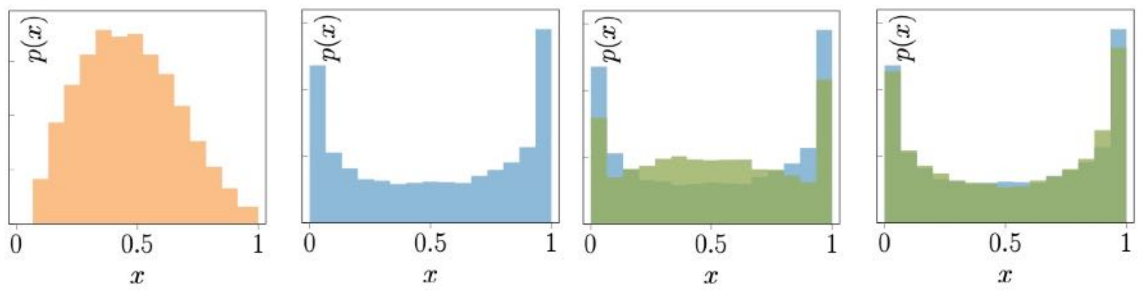
One difficulty that arises with class-conditioning the style transfer is the existence of so-called orphan labels. Orphan labels are all pixels in the content image whose classes are not represented in the style image so that no stylization information is available. To decrease the chance of orphan labels, the number of classes should be kept as small as possible. In case that orphan labels still appear, there are some strategies to deal with them, such as excluding the associated pixels from the optimization problem [17] or optimizing all orphan label pixels against the global style of the style image.

2.5. State-Of-The-Art Methods for Neural Style Transfer

After having provided the basic theory for NST, two improved implementations are presented in this section. An optimization-based algorithm introducing an alternative to the gram-based loss in Section 2.5.1 and a feedforward architecture based on WCT that is focusing on being photorealistic in Section 2.5.2.

2.5.1. CMD – Improving Style Transfer by Taking into Account Higher-Order Moments

As explained in Section 2.4.1, the underlying process of style transfer is to align image distributions in feature space. So far, this has either been achieved by matching second-order statistics in the form of covariance matrices [26, 31] or by aligning first-order moments in the form of channel-wise means and variances [30]. These two approaches suffer from the same limitation [18]. They cannot always reach a perfect alignment between two distributions because they are unable to uniquely identify a distribution [18]. The covariance matrix cannot uniquely identify a distribution since two different distributions can lead to the same matrix [18, 37]. The same is true for mean and variance values, where it is easy to find two different distributions that share these properties, e.g. an exponential distribution with $\lambda = 1$ and normal distribution with $\mu = 1$ and $\sigma^2 = 1$.



(a) Source (b) Target (c) Mean & Variance (d) CMD, $K = 5$

Figure 2.3.: Illustration of the distribution alignment capabilities of the CMD. The source distribution Beta(2, 3) and target distribution Beta(0.5, 0.45) cannot be aligned just by matching the mean and the variance. The CMD can successfully align them using just five moments. With some adaptions, figure taken from [18, p.4].

To improve distribution alignment and achieve a more exact style transfer, Kalischek et al. [18] introduced a new style loss for NST. The introduced loss is based on the central moment discrepancy metric (CMD) [38]. This metric measures the distance between two distributions by taking into account many higher-order moments the former methods omitted. It is therefore able to successfully align distributions for cases the other methods failed at. The superiority of using the central moment discrepancy for aligning distributions is illustrated in Figure 2.3.

In the following we give an introduction into the mathematical background of the central moment discrepancy before explaining its practical application in NST.

Mathematical background of the Central Moment Discrepancy

One common way to measure the distance between two distributions $\mathcal{F}_s$ and $\mathcal{F}_t$ are so-called integral probability metrics (IPM) [39]. Given a class of functions $\mathcal{G} = \{g : \mathcal{X} \to \mathbb{R}\}$, an integral probability metric is defined as

$$d_{\mathcal{G}}(\mathcal{F}_s, \mathcal{F}_t) = \sup_{g \in \mathcal{G}} \left| \mathrm{E}_{\mathcal{F}_s}[g] - \mathrm{E}_{\mathcal{F}_t}[g] \right| . \tag{2.10}$$

The central moment discrepancy [38] is a special IPM focusing on the polynomial function space. Choosing this space makes natural sense since first, the expectations of polynomials are sums of moments and second, moments are known to be robust distribution descriptors.

More specifically, the CMD uses the function space $\mathcal{P}^k$ defined as the class of 'homogeneous[5] polynomials $p : \mathbb{R}^m \to \mathbb{R}$ of degree k' [38, p.5]. To construct this space, a new vector-valued function is introduced that assigns a m-dimensional vector $\underline{x} = (x_1, \ldots, x_m)^T$ to all its monomial values of order k.

$$\begin{aligned} \underline{\nu}^{(k)} : \mathbb{R}^m &\to \mathbb{R}^{\frac{(k+1)^{m-1}}{(m-1)!}} \\ \underline{x} &\to \left(x_1^{r_1} \ldots x_m^{r_m} \right)_{\substack{(r_1,\ldots,r_m)\in\mathbb{N} \\ r_1+\ldots+r_m=k}} \end{aligned} \tag{2.11}$$

[5]Polynomials where all nonzero terms have the same degree, e.g. $x^5 + 2x^3y^2 + 9xy^4$.

For example, $\underline{v}^{(3)}((x_1, x_2)) = \left(x_1^3, x_1^2 x_2, x_1 x_2^2, x_2^3\right)^T$.

With an additional normalized coefficient vector $\underline{w}$ all elements of $\mathcal{P}^k$ can be written in the form of

$$p(\underline{x}) = \left\langle \underline{w}, \underline{v}^{(k)}(\underline{x}) \right\rangle. \tag{2.12}$$

Unfortunately, such an integral probability metric suffers from one major drawback, i.e. mean over-penalization [38]. Mean over-penalization means that mean values overproportionally contribute to the calculation of $d_{\mathcal{P}^k}$, such that even small changes in the mean can lead to large changes in the metric. A new centralized version of the IPM solves this problem [38, p.5,6]

$$d^c_{\mathcal{G}}(\mathcal{F}_\mathrm{s}, \mathcal{F}_\mathrm{t}) = \sup_{g \in \mathcal{G}} \left| \mathrm{E}_{\mathcal{F}_\mathrm{s}}\left[g(\underline{x} - \mathrm{E}_{\mathcal{F}_\mathrm{s}}[\underline{x}])\right] - \mathrm{E}_{\mathcal{F}_\mathrm{t}}\left[g(\underline{x} - \mathrm{E}_{\mathcal{F}_\mathrm{t}}[\underline{x}])\right] \right|. \tag{2.13}$$

Based on this centralized version, the CMD is defined as a weighted sum of all centralized IPMs up to polynomial order k

$$\mathrm{cmd}_k(\mathcal{F}_\mathrm{s}, \mathcal{F}_\mathrm{t}) = a_1 d_{\mathcal{P}^1}(\mathcal{F}_\mathrm{s}, \mathcal{F}_\mathrm{t}) + \sum_{j=2}^{k} a_j d^c_{\mathcal{P}^j}(\mathcal{F}_\mathrm{s}, \mathcal{F}_\mathrm{t}), \tag{2.14}$$

where $a_j \geq 0$ and $d_{\mathcal{P}^1}(\mathcal{F}_\mathrm{s}, \mathcal{F}_\mathrm{t})$ is redefined to $d_{\mathcal{P}^1}(\mathcal{F}_\mathrm{s}, \mathcal{F}_\mathrm{t}) = |\underline{\mu}_t - \underline{\mu}_s|$ because doing so holds more information than $d_{\mathcal{P}^1}(\mathcal{F}_\mathrm{s}, \mathcal{F}_\mathrm{t}) = 0$ [38].

Defining $\underline{c}_1(\mathcal{F}) = \mathrm{E}_{\mathcal{F}}[\underline{x}]$ and $\underline{c}_j(\mathcal{F}) = \mathrm{E}_{\mathcal{F}}\left[\underline{v}^j(\underline{x} - \mathrm{E}_{\mathcal{F}}[\underline{x}])\right]$ for $j \geq 2$, Equation (2.14) can be equivalently formulated as [38, Proof of Theorem 2]

$$\mathrm{cmd}_k(\mathcal{F}_\mathrm{s}, \mathcal{F}_\mathrm{t}) = \sum_{j=1}^{k} a_j \left\| \underline{c}_j(\mathcal{F}_\mathrm{s}) - \underline{c}_j(\mathcal{F}_\mathrm{t}) \right\|_2. \tag{2.15}$$

This formulation describes the difference between the central moments $\underline{c}_j$ of both distributions, hence the name central moment discrepancy.

For $k \to \infty$, i.e. taking all central moments into account, [40, Theorem 1] states that $\mathrm{cmd}_\infty(\mathcal{F}_\mathrm{s}, \mathcal{F}_\mathrm{t}) = 0$ only if $\mathcal{F}_\mathrm{s} = \mathcal{F}_\mathrm{t}$, thus solving the problem of indistinguishability. While this equality does no longer hold for smaller k, [38, Proposition 1] shows that there is a theoretical upper bound for the contribution of the k-th term towards the cmd that is strictly decreasing with the order of k. Therefore, since the contribution of higher moments is continuously decreasing the higher the order, it is reasonable to limit the calculation of cmd_k to only the first m marginal moments without introducing a major error [18]. Furthermore, experiments from [38] have shown that most cross-moments in $\underline{v}^k$ are not needed and that even reducing $\underline{v}$ to omit all cross moments, i.e.

$$\underline{v}^{(k)}(\underline{x}) = \left(x_1^k, \ldots, x_m^k\right)^T,$$

achieves better results than comparable approaches (e.g. gram-based).

Integrating the Central Moment Discrepancy for Neural Style Transfer

As Kalischek et al. [18] have shown, to adapt the central moment discrepancy for usage in NST, we must first wrap every feature activation with a sigmoid function $\sigma(\cdot)$. Doing so guarantees that all values fall in between $[0, 1]$ which is a core requirement of the central moment discrepancy [38]. Then, with a slight stretch of notation[6], the style loss for layer l can be formulated as

$$\mathcal{L}^{l}_{\text{style}}(I_{\text{s}}, I_{\text{o}}) = \text{cmd}_k \left(\tilde{\mathbf{F}}^l[I_{\text{s}}], \tilde{\mathbf{F}}^l[I_{\text{o}}] \right), \tag{2.16}$$

where $\tilde{\mathbf{F}} = \sigma(\mathbf{F})$.

For an efficient calculation of cmd_k, k is typically set to be around five and all cross moments are omitted, thus simplifying the coefficients $\underline{c}_i(\cdot)$ to just expectations over the i-th powers of $(\underline{x} - \underline{\mu}_{\tilde{\mathbf{F}}^l})$, i.e.

$$\underline{c}_i \left(\tilde{\mathbf{F}}^l \right) = \text{E}_{\tilde{\mathbf{F}}^l} \left[\left(\underline{x} - \underline{\mu}_{\tilde{\mathbf{F}}^l} \right)^i \right] \in \mathbb{R}^{K_l}. \tag{2.17}$$

To clarify this notation, $\underline{x} \in \mathbb{R}^{K_l}$ is a sample from $\tilde{\mathbf{F}}^l$ containing the values of all feature channels for one 'pixel position'. $\underline{\mu}_{\tilde{\mathbf{F}}^l}$ is the channel-wise mean calculated over all such samples. The expectation is formed over the i-th power of all samples subtracting their mean [18].

Since using the central moment discrepancy is the main difference of this new NST algorithm from others like the one of Gatys et al. [26], starting from now, the acronym CMD will be reinterpreted to denote this new style transfer algorithm.

Network Architecture

An optimization algorithm based on VGG-19 [20] as the feature extractor forms the foundation of the style transfer architecture introduced by Kalischek et al. [18]. Because the inputs of the style loss are wrapped with the sigmoid function, the non-rectified outputs from layers *conv1_1*, *conv2_1*, *conv3_1*, *conv4_1*, and *conv5_1* are used to calculate $\mathcal{L}^{l}_{\text{style}}$. The content loss is calculated according to Equation (2.3) with the rectified output of layer *relu4_1* as the input. To respect the different number of feature maps in each layer, the style losses of different layers l are scaled by factors of $10^3/K_l^2$, where K_l is the number of filters in layer l. The factor 10^3 is a hyperparameter that [18] fixed to this value.

2.5.2. WCT^2 – Photorealistic Style Transfer Using Wavelet Transforms

So far, all presented style transfer algorithms have shown that they are capable of impressive stylizations. They, however, all struggle with the task of photorealism. With photorealism, we mean the capability of an algorithm to transfer the style between two photos with the constraint that the generated result still looks like a photo.

The different approaches struggle with photorealism because they all use a content representation in high-level CNN feature space. While these features are good for representing general content information, they inevitably lose some low-level information contained in the content image. The loss of information is due to the usage of max-pooling layers in the

[6]$\mathbf{F}$ is not a distribution but the tensor of feature-activations. However, it can be interpreted as a distribution $\mathcal{F}$ by taking all the values at one 'pixel position' as a sample. This is the same process as explained in Section 2.4.1.

encoder [17] and leads to missing fine details, distorted structured, irregular artifacts, and an overall non-photorealistic result [17, 41].

Different approaches have been proposed to improve photorealism. For optimization-based style transfer, Luan et al. [36] proposed an additional regularizer term in pixel space. This regularizer should prevent distortions by forcing the stylized image to be explainable by transformations that are locally affine in the color space of the content image. Their solution was only partially effective. It was effective at preventing distortions but could not restore the loss of fine details and often resulted in inconsistent stylizations [41].

For feedforward structures, a first attempt to improve photorealism was undertaken by Li et al. [41]. They aimed to compensate for the information loss during encoding by replacing the upsampling layers in the decoder with learnable unpooling layers. This certainly helped but could not compensate for the full information loss, therefore requiring a series of post-processing steps (i.e. filtering/smoothing) to achieve photorealism at the cost of runtime and a loss of fine details [17].

The so far best attempt has been undertaken by Yoo et al. [17]. Also focusing on feedforward style transfer, instead of artificially trying to restore some information lost during encoding – the basic idea of [41] – they aimed to directly solve the fundamental problem of information loss itself. By replacing the pooling and unpooling layers with wavelet-based alternatives, they created an encoder/decoder architecture that theoretically allows for perfect reconstruction of the input signal and can achieve photorealism without any required post-processing steps.

In the following two sections we first give an introduction into wavelet pooling and explain why an encoder/decoder structure based on it theoretically allows for perfect signal reconstruction. Then we present the full architecture for NST called WCT^2.

Wavelet Pooling and Perfect Reconstruction

For wavelet pooling two filters are constructed out of the Haar-wavelet [42]; a low pass (L) and a high pass (H)

$$L^T = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\end{bmatrix},\ H^T = \frac{1}{\sqrt{2}}\begin{bmatrix}-1 & 1\end{bmatrix}.$$

Together these filters form four kernels $\left\{\mathrm{LL}^T\ \mathrm{LH}^T\ \mathrm{HL}^T\ \mathrm{HH}^T\right\}$. Convolving the kernels with the input generates four different pooling outputs. This is a difference compare to other pooling operations which typically only have one output. For simplicity, we call the different outputs of wavelet pooling LL, LH, HL, and HH. All outputs have a direct practical representation with LL extracting smooth surfaces and textures, whereas the high-pass filters (LH, HL, and HH) capture edges and edge-like structures [17].

Wavelet pooling is different from other techniques such as max-pooling, as it has a direct inverse function that makes it possible to exactly reconstruct the input signal[7]. In detail, the pooling operation can be reversed by 'performing a component-wise transposed convolution,

[7]It should be noted that there are alternatives to Haar wavelet pooling/unpooling which also have an inverse operation. Haar wavelets are chosen because they split the output into 4 channels, each with a practical representation. This enables us to focus the stylization on different parts, e.g. only stylize the low-pass component [17].

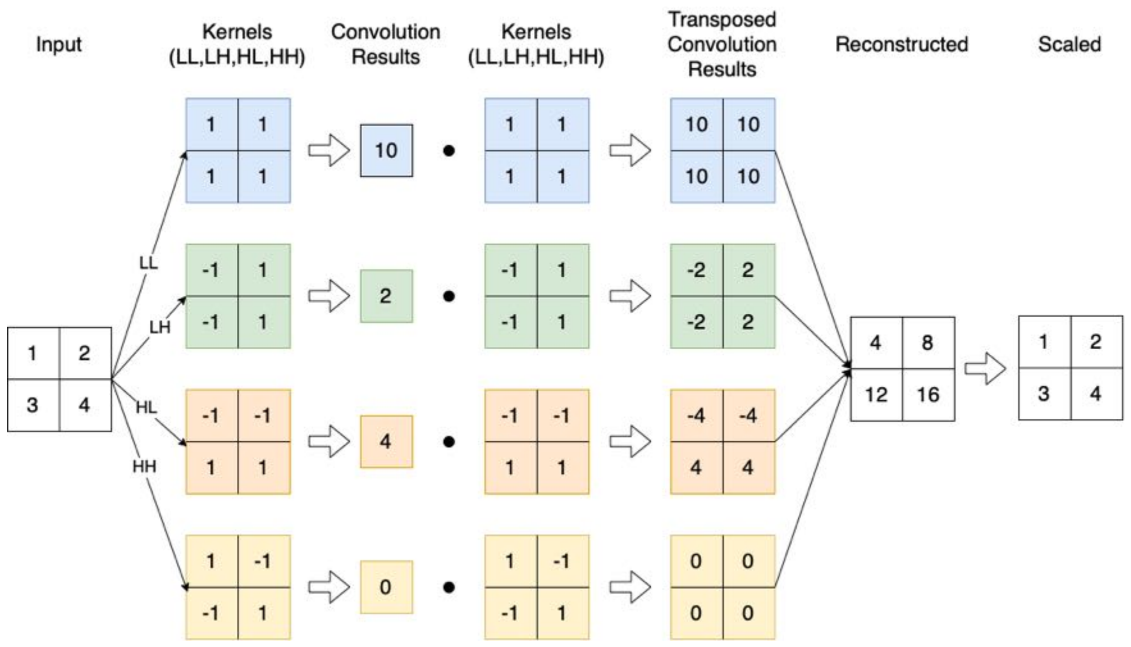


Figure 2.4.: Example of the reconstruction capabilities of wavelet pooling/unpooling. In the pooling step, the four kernels are convoluted with the input. The original input can be recovered by multiplying the results of the convolutions with the respective kernels and then adding all results together.

followed by a summation' [17, p.3]; i.e. wavelet unpooling. The process of wavelet pooling and unpooling is illustrated with an example in Figure 2.4.

The existence of an inverse pooling operation is the key property that allows wavelet-based NNs to perform perfect reconstruction of the input signal. However, the step from the existence of an inverse for the pooling operation to the statement that the full network is able to perform perfect reconstruction is not automatically obvious. The work of Ye et al. [43] provides the missing link, proving that it is possible to obtain a CNN capable of perfect reconstruction (PR) when some preconditions are met. A short introduction to their work is given for completeness. For more detailed information, the interested reader is advised to have a look at the original paper [43].

Frame Theory: The existence of a direct inverse operation for wavelet pooling can be formalized in the scope of frame theory. Given an operator $\mathbf{\Phi} = \begin{bmatrix} \underline{\phi}_1 & \cdots & \underline{\phi}_m \end{bmatrix} \in \mathbb{R}^{n \times m}$, where $\{\underline{\phi}_k\}_{k=1}^m$ are vectors in a Hilbert space H, $\{\underline{\phi}_k\}$ forms a frame if it satisfies the inequality [44]

$$\alpha \|\underline{x}\|^2 \leq \|\mathbf{\Phi}^T \underline{x}\|^2 \leq \beta \|\underline{x}\|^2, \ \forall \underline{x} \in H. \tag{2.18}$$

Here, $\underline{x} \in \mathbb{R}^n$ is a random input signal and $\alpha, \beta > 0$ are the bounds of the frame.

A frame satisfies the PR condition that $\underline{x}$ can be exactly recovered from $\underline{z} = \mathbf{\Phi}\underline{x}$, if there exists a dual frame $\widetilde{\mathbf{\Phi}}$ such that $\widetilde{\mathbf{\Phi}}\mathbf{\Phi} = \mathbb{I}$ [17]; $\mathbb{I}$ being the identity matrix.

In addition to perfect reconstruction, some more frame properties are desirable for usage in NNs [17]. First, to not amplify any noise, it is desirable that a frame does not amplify the power of the input signal. This is achieved when the frame is tight, meaning $\alpha = \beta$ and therefore $\widetilde{\mathbf{\Phi}} = \mathbf{\Phi}$ or equivalently $\mathbf{\Phi}\mathbf{\Phi}^T = \mathbb{I}$. Second, for the usage with parametric models, an ideal frame should provide good energy compaction, i.e. concentrating the energy of the input $\underline{x}$ in as few elements of $\underline{z}$ as possible. Latter is desirable because it helps the model to 'adaptively deal with varying amount of information with a fixed number of parameters' [17, Appendix A].

Haar wavelets form such an ideal frame $\mathbf{\Phi} = \begin{bmatrix} L & H \end{bmatrix}$. It is tight since

$$\mathbf{\Phi}\mathbf{\Phi}^T = LL^T + HH^T = \mathbb{I}, \tag{2.19}$$

and it is compact because compact signal representation is one key property of wavelets [45].

Perfect Reconstruction: Ye et al. [43] have shown that training an encoder-decoder CNN is equivalent to learning some local bases $\mathbf{\Psi}$ (e.g. parameters of the model) with fixed global bases $\mathbf{\Phi}$ (e.g. architecture of the model). More specifically, training is the same as 'finding a multi-layer realization of the convolution framelets [46]' [17, Appendix A]

$$\mathbf{Z} = \mathbf{\Phi}^T \left(\underline{x} \circledast \mathbf{\Psi}\right) \tag{2.20}$$

$$\underline{x} = \left(\widetilde{\mathbf{\Phi}}\mathbf{Z}\right) \circledast \widetilde{\mathbf{\Psi}}, \tag{2.21}$$

where $\mathbf{\Phi}$ and $\widetilde{\mathbf{\Phi}}$ and respectively $\mathbf{\Psi}$ and $\widetilde{\mathbf{\Psi}}$ are frames and their duals.

The global bases $\mathbf{\Phi}, \widetilde{\mathbf{\Phi}}$ are fixed with the network architecture, for example, the choice of pooling layers. The local bases $\mathbf{\Psi}, \widetilde{\mathbf{\Psi}}$ are learned during training. Perfect reconstruction of the input signal is only possible if both frames satisfy the PR criterion

$$\mathbf{\Phi}\widetilde{\mathbf{\Phi}}^T = \mathbb{I}, \ \mathbf{\Psi}\widetilde{\mathbf{\Psi}}^T = \mathbb{I}. \tag{2.22}$$

Therefore, if already the global bases do not satisfy the PR criterion, perfect reconstruction is never achievable. This is, for example, the case for architectures using max-pooling, which does not have an exact inverse. WCT^2, on the other hand, makes use of the reconstruction capabilities of wavelet pooling, thereby allowing the NN to learn local bases for theoretically perfect reconstruction [17].

Network Architecture

An encoder-decoder architecture based on the VGG-19 [20] network up to layer *conv4_1* is once again used as the base architecture of [17]. All max-pooling and unpooling layers are replaced with wavelet-based alternatives. At every wavelet pooling stage, the low-frequency component (LL) is forwarded to the next layer, while the high-frequency components (LH, HL, HH) are directly skipped to the corresponding unpooling layer of the decoder. Stylization is carried out by the whitening and coloring transform. However, unlike *single-stage stylization* as in [30] or *multi-level stylization* as in [31, 17], a new *progressive stylization* architecture is being used. During one forward pass, features are progressively transformed, starting with the LL components after *conv1_1*, up to *conv4_1*.

The progressive stylization approach provides several advantages. Compared to *single-stage stylization*, it leads to better stylization results. Compared to *multi-level stylization*, it simplifies the training and reduces the number of parameters because it does not require to train an individual decoder for each level. Additionally, it prevents the amplification of errors that

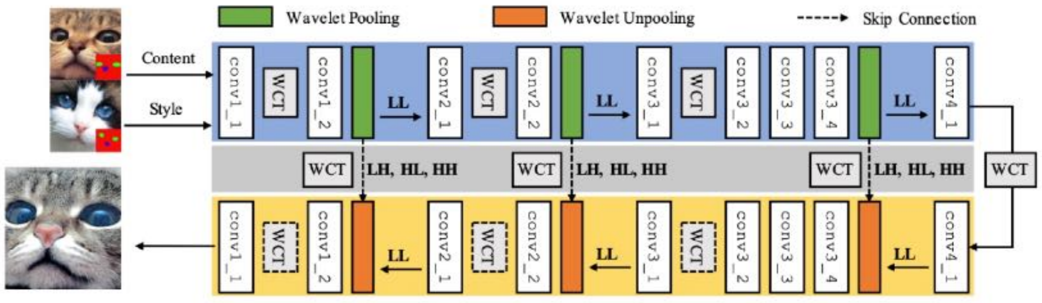


Figure 2.5.: Overview of the progressive stylization architecture used in WCT^2. WCT is performed in the encoder, at the skip connections, and in the decoder. Figure taken from [17, Appendix A.4].

may occur when iteratively encoding and decoding the same image in a multi-level architecture. However, it should still be noted that even though close, a progressive approach cannot create the same level of stylization as a multi-level one. This is because in the progressive approach stylization is performed from fine to coarse, making it possible that initially fine-tuned details become overruled by later coarse transformations [47].

Starting from the base implementation where only the low-frequency LL component is stylized in the encoder, the stylization intensity can be further increased by adding additional WCT stages to the high-frequency skip connections and the decoder.

The complete architecture of WCT^2 with the maximum number of WCT operations is illustrated in Figure 2.5.

3. Five Frameworks for Continual Unsupervised Domain Adaptation in Semantic Image Segmentation

After having provided all the necessary theoretical foundations in Chapter 2, the five different frameworks for continual unsupervised domain adaptation that we are comparing in this work are introduced in this chapter. Three frameworks are using NST to adapt to new domains. They all implement a replay mechanism to prevent forgetting but diverge in the selected algorithm for style transfer. The other two frameworks use adversarial training to align domains in output space. They mainly diverge in the chosen approach to prevent forgetting.

The goal of each framework is to learn a sequence of domains $\{\mathcal{D}_t\}_{t=0}^{T}$, while at the same time retaining knowledge about all proceeding domains (including the source domain $\mathcal{D}_0$). The fully labeled source domain $\mathcal{D}_0$ consists of N^0 images and segmentation maps $\{I_i^0, \boldsymbol{S}_i^0\}_{i=1}^{N^0}$ and is available throughout the whole training. The unlabeled target domains $\{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_T\}$ arrive sequentially forming the target datasets $\{I_j^t\}_{j=1}^{N^t}$, where N^t is again the number of images per target domain t. UDA is used to sequentially adapt the model to each target domain, however, with the limitation that the images of a given target domain $\mathcal{D}^t$ are only accessible when adapting to this domain, and that they cannot be used after that to prevent CF.

The input images I are generally tensors of dimension $I \in \mathbb{R}^{3\times H\times W}$. However, since we only treat them as an abstract input to the segmentation network, the bold typesetting is omitted to improve readability, make the notation consistent with the section about NST, and avoid confusion with the identity matrix. The segmentation maps are one-hot encoded. They are of dimension $\boldsymbol{S} \in \mathbb{R}^{C\times H\times W}$, where C is the number of classes. Finally, we define $\boldsymbol{Y} \in \mathbb{R}^{C\times H\times W}$ as the soft predictions (softmax probability) of the segmentation network.

For a simplification of notation, we use the shorthand formulation I^0 to denote a random image from the source domain and I^t for a random image from the target domain t.

3.0.1. CACE – A Framework for Continual UDA Using Class-Conditioned Style Transfer

The Class-Specific Adaptation to Changing Environments (CACE) [10] framework was developed as an improvement to the older ACE [9] framework, the latter being one of the first proposals for continual UDA for semantic segmentation. The general idea behind ACE is presented first, before explaining the changes introduced by [10] that transformed ACE into CACE.

Style Transfer as a Way to Adapt the Segmentation Network to New Domains

Starting from a segmentation network trained on the source domain, ACE makes use of NST to adapt to new target domains. In more detail, given a target domain $\mathcal{D}_t$, ACE uses the AdaIN style transfer architecture from Section 2.4.2 to transfer images from the source domain $\mathcal{D}_0$ into the style of images from the target. Since images generated this way combine the content and labels of the source domain with the style of the target domain, they can be used to train the segmentation model for good performance on the target domain.

More formally we define $I^{0\to t}$ as a source image stylized in the style of the target domain t. $\boldsymbol{S^0} \in \mathbb{R}^{C\times H\times W}$ is the respective segmentation map and $\boldsymbol{Y^{0\to t}} \in \mathbb{R}^{C\times H\times W}$ are the soft predictions of the segmentation network for input $I^{0\to t}$. Using this notation, the segmentation network is trained by minimizing the cross-entropy loss

$$\mathcal{L}_{\text{seg}}(I^{0\to t}, \boldsymbol{S^0}) = \text{CrossEntropy}(\boldsymbol{Y}^{0\to t}, \boldsymbol{S^0}) = \frac{-1}{CHW}\sum_{c=1}^{C}\sum_{h=1}^{H}\sum_{w=1}^{W} S^0_{chw}\log(Y^{0\to t}_{chw}). \tag{3.1}$$

Using a Style Memory to Prevent Forgetting

ACE implements a replay mechanism (memory-based approach) to prevent forgetting. After training on each target domain $\mathcal{D}_t$, the style characteristics of this domain, i.e. the channel-wise mean and variance, are stored in a small memory. When training for a new target domain, only every second image is transferred into the target style. The other image is either transferred into a random style from a previous target domain to prevent forgetting on this domain, or not stylized at all to prevent forgetting on the source domain.

Class-Conditioning the Style Transfer to Improve Performance

Although ACE has shown some good results, oftentimes its performance was not significantly better than that of simple augmentation techniques like color jittering [10]. This is primarily because the image-wide style transfer used in ACE is not capable of capturing and transferring enough style information for a successful domain adaptation [10]. In more detail, as explained in Section 2.4.3, transferring style image-wide only captures dominant patterns but misses important class-specific differences [10]. CACE solves this issue and improves performance by class-conditioning the style transfer to only align style between semantically similar regions of source and target image. Since no labels are available for the target domains, self-generated labels using the predictions of the segmentation network (pseudo labels) have to be used instead. The memory for replay is kept but the required storage size grows since style information must be stored individually for each class.

Even though class-conditioning the style transfer theoretically allows for the existence of orphan labels disturbing the style transfer, CACE can successfully prevent them by using its memory. To achieve this, CACE already extracts and stores the style of all target images before training on a given target domain. Then, when transferring style during training, CACE is not limited to extracting style from only one target image. Instead, the style information for every class can be sampled from the memory individually.

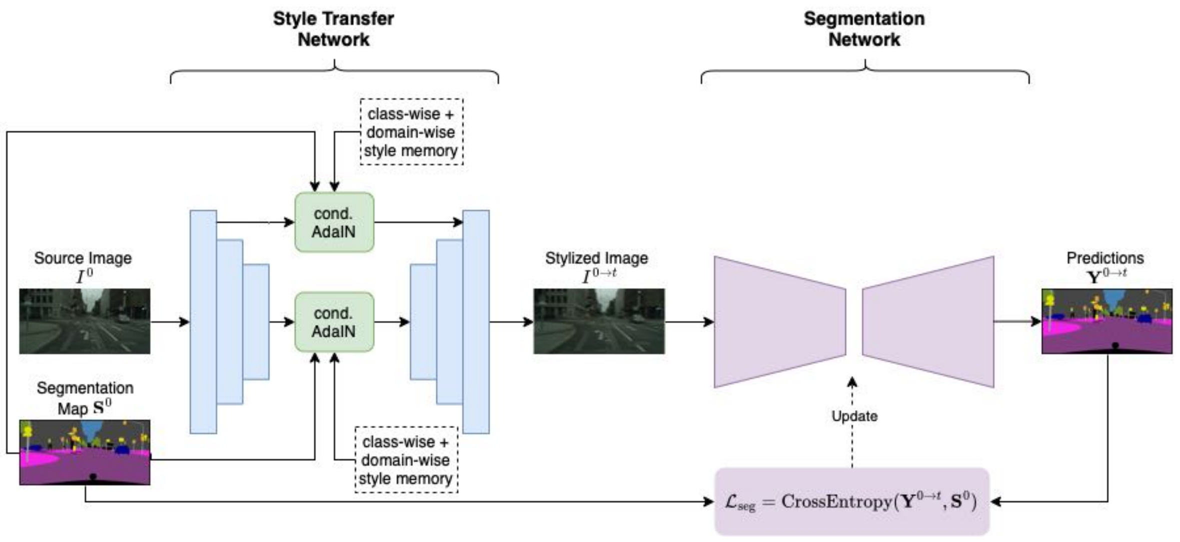

Style Transfer Network
Segmentation Network
class-wise + domain-wise style memory
cond. AdaIN
cond. AdaIN
class-wise + domain-wise style memory
Source Image I^0
Segmentation Map $\mathbf{S}^0$
Stylized Image $I^{0\to t}$
Predictions $\mathbf{Y}^{0\to t}$
Update
$\mathcal{L}_{\text{seg}} = \text{CrossEntropy}(\mathbf{Y}^{0\to t}, \mathbf{S}^0)$

Figure 3.1.: Architecture of the CACE framework used to adapt to new domains. CACE consists of two separate blocks. The style transfer network stylizes source images into the style of a target domain using class-conditioned AdaIN. These stylized images are then used to train the segmentation network by minimizing the cross entropy loss between the soft predictions and the original labels. The figure is based on [10].

In addition to class-conditioning the style transfer, CACE further improves performance by reducing the number of artifacts and increasing the level of detail in the stylized results. This was achieved by adding an early skip connection with an additional AdaIN stage to the encoder-decoder structure used for style transfer.

Network Architecture

The structure of the CACE framework is illustrated in Figure 3.1. It consists of two parts, the style transfer network and the segmentation network. Using the style memory, the style transfer network transforms source images into a given target style. These stylized images are then used to train the segmentation network for good performance on the target domain by minimizing the cross entropy loss between the predictions (softmax probability) and the original labels.

What this schematic overview does not show is the replay mechanism. As previously discussed, the style memory is not only used to stylize source images into the style of the current target domain. It is also used to stylize source images into the style of a previous target domain to prevent forgetting. To prevent forgetting on the source domain, some replayed images are not stylized at all. The style transfer network is skipped for these images and they are directly used to train the segmentation network.

3.0.2. Continual-WCT^2 and Continual-CMD – Two Adaptations to CACE with Improved Style Transfer Algorithms

Marsden et al. [10] have shown that the CACE framework is generally effective for continual UDA. However, CACE only uses a style transfer network based on the AdaIN style transfer network. As previously discussed, transferring style like this suffers from two weaknesses:

- In Section 2.5.1 it has been shown how for some cases AdaIN cannot properly align the feature distributions of style and content image. CMD has been introduced to overcome this limitation and to create more accurate stylizations.
- In Section 2.5.2 it has been shown how AdaIN can struggle with Photorealism, i.e. preventing distortions and preserving fine details. WCT^2 has been introduced to overcome this limitation and to create high-detail photorealistic stylizations.

From these weaknesses in the style transfer algorithm of CACE, it makes sense to ask whether performance can be improved by exchanging the style-transfer-network used in CACE for the supposedly superior methods WCT^2 or CMD.

To evaluate this, we developed two new frameworks for continual UDA that share a structure similar to that of CACE but implement the two alternative algorithms for NST. In accordance with the name of the respective style transfer algorithm, but since they are used for continual unsupervised domain adaptation, we call the resulting frameworks continual WCT^2 (C-WCT^2) and continual CMD (C-CMD).

Integrating a new NST algorithm into CACE is not as simple as just exchanging the style algorithm and keeping everything else fixed. For example, storing the mean and variance values to remember the style of an image is only appropriate when using the AdaIN operation for stylization. It will become clear that to integrate WCT^2 and CMD, some trade-offs had to be made that limit the practical application of the resulting frameworks. Still, since the main objective of our two developed frameworks is to identify the effect of differences in stylization on continual UDA performance, we believe that it is justified to ignore the practical constraints for now and focus only on performance.

The two new frameworks C-WCT^2 and C-CMD are presented in the following.

C-WCT^2 – A CACE-like Framework with Enhanced Photorealism

Two different implementations were developed to integrate the NST algorithm WCT^2 into a CACE-like structure. Both come with their own set of issues that limit practical application.

Implementation 1: A simplified implementation omitting the style memory

To simplify the implementation, the first version completely omits the style memory. Instead of sampling style from a memory, for every source image I^0, a random style image I^t is sampled and its style information is extracted for stylization.

Compared to using a memory this approach is not ideal. For one, it is not runtime efficient because it requires one to newly extract all style information for every source image that becomes stylized. Instead of reading the style from a memory, this is associated with much more computational overhead. Even worse, in order to be able to extract style from an image, our implementation requires storing all the images from previous target domains. This is a break with the original continual UDA assumption that domains arrive sequentially and are no longer available after training. Finally, a third issue connected with omitting the memory is that it makes for the chance of orphan labels which hurt the stylization and consequently continual UDA performance.

Implementation 2: Reintroducing the style memory can prevent orphan labels but the memory requirements are very large

In an attempt to overcome the previous limitations, the second implementation reintroduces the style memory from CACE. As explained with CACE, using such a memory can completely prevent orphan labels since it is possible to sample style information for each class individually. Additionally, the memory increases diversity in the dataset since it is possible to mix the style information of different images by sampling each class style from a different image.

However, to implement such a memory and store stylization information, the WCT operation used in WCT^2 requires one to store a styling matrix instead of just mean and variance values as in CACE. The consequence of this is that the style memory grows many times larger than that of CACE. For example, it can be shown (Appendix A.1) that when using 19 classes, approximately 47 MB must be stored per image. This is to the order of 1000 larger than the memory requirements of CACE and also significantly larger than just saving the raw image itself.

The style memory therefore does not overcome the memory issue of the first implementation. Instead, the memory requirement is even higher. It is so high that even in settings where memory is not strictly restricted it can lead to problems. For example, when performing the experiments, we will later see that the style memory could not be used for one experiment since the required memory would grow upwards of 150 GB.

C-CMD – A CACE-like Framework with More Accurate Stylization

Integrating the NST algorithm CMD with the CACE framework is difficult. CACE uses an efficient feedforward style transfer which can stylize images in around one second. CMD instead is optimization-based and the stylization of just one image typically takes upwards of five minutes.

With this time consumption, it is impossible to stylize one image for every training iteration. To illustrate this, we will later see that the training of the continual UDA frameworks takes around $10,000$ iterations per target domain. With more than five minutes per image, this would mean a runtime of over $50,000$ minutes, 830 hours, or 34 days for just one target domain.

Despite the inefficiency, we still believe that it is worthwhile to evaluate the effect of using CMD on continual UDA performance. Thus, in order to be able to perform the later experiments, we decided to integrate CMD in a trade-off implementation that comes with some major limitations.

Our implementation makes CMD applicable for continual UDA by splitting the training into a two-step process that drastically reduces the number of stylized images required.

Step 1: Pre-stylizing the images to reduce the number of required style transfers

In the first step, for every style image from the target domain, one random image from the source domain is transferred into this style. Since this step happens 'offline' before training the segmentation network, the network cannot be used to generate pseudo labels and real labels have to be used instead.

Step 2: Training the segmentation network using pre-stylized images

In the second step, the previously generated images are used to train the segmentation network. This second step is performed the same way as it is done in CACE. One pre-stylized image is always sampled from the current target domain, while the second image is sampled from a previous domain to prevent forgetting.

Training the segmentation network in this two-step procedure comes with two major drawbacks:

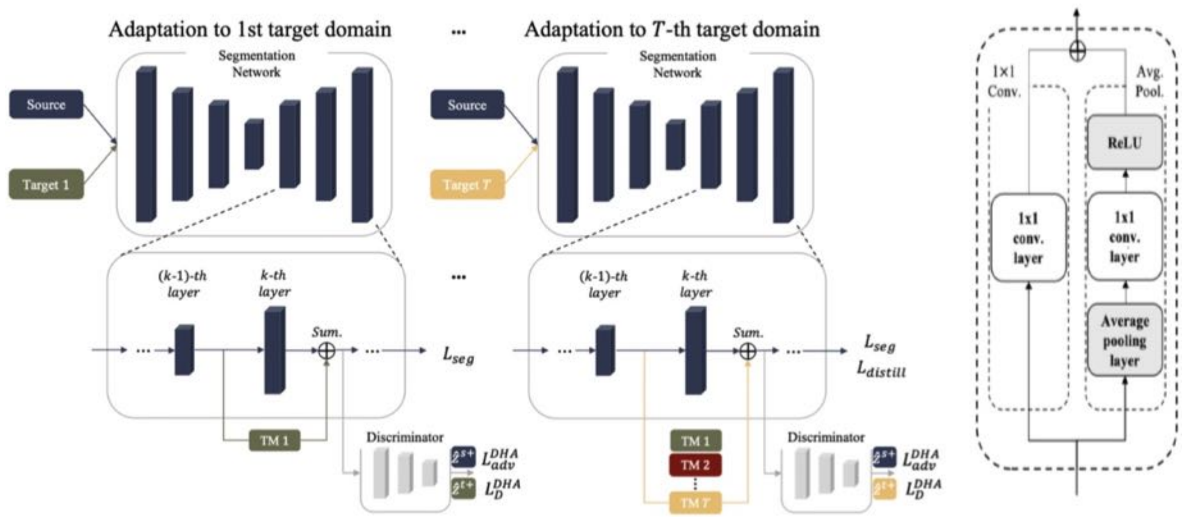
- Limiting the stylization to just one content/style-pair per target image reduces the dataset size for training compared to the former frameworks CACE and C-WCT^2.
- By using real labels instead of pseudo labels, the training is no longer unsupervised and cannot be performed in practice where no labels for the target domains are available.

The first drawback cannot be overcome because it is due to the optimization-based nature of CMD. For the second one, a solution exists. It is quite possible to use the segmentation network already in the first step to generate pseudo labels. However, since 1) implementing this improved version was difficult with our code-base, 2) time was limited, 3) when later performing the experiments we can actually access all labels, and finally, 4) [10] reported that the advantage of using real-labels is not that large, we decided for the presented and simpler implementation.

3.0.3. ETM – Continual UDA Using an Alignment in Output Space and a Target Specific Memory to Prevent Catastrophic Forgetting

A completely different approach than the previous three frameworks has been chosen by Kim et al. [13]. Instead of transferring style, they are making use of an adversarial learning scheme to align the source and target domains in the output space of the segmentation network. The domain alignment is based on the idea of generative adversarial networks (GAN) [23]. The discriminator is tasked to differentiate if a given input, i.e. the predictions of the segmentation network, is from the source or the target domain. The segmentation network takes the role of the generator and tries to trick the discriminator by making its predictions for the source and the target domain indistinguishable.

While an approach like this is not new and has already been successfully used by the likes of AdaptSegNet [8] and Advent [48], Kim et al. [13] added to this with the introduction of a target-specific memory to improve continual learning capabilities.



(a) ETM framework (b) TM structure

Figure 3.2.: (a) Architecture of the ETM framework. For each target domain, a TM module is added. The discriminator tries to distinguish the source and the target domain, while the combination of segmentation network and TM module tries to trick the discriminator by making both domains indistinguishable. For every target domain, the respective TM module is added parallel to the layer of the segmentation network that enters into the discriminator. (b) Each TM module consists of one 1×1 Conv. module to extract localized information and one Avg. pooling module to extract contextual information. Both figures are taken from [13].

A Lightweight Target-Specific Memory to Store Domain Discrepancies

Continually expanding the network capacity has already been successfully used for continual learning in image classification [24]. Kim et al. [13] have been the first to transfer this knowledge into the field of semantic segmentation by introducing a new lightweight target-specific memory module (TM). The idea is that for every new target domain, a new TM module is added to the network, tasked to capture and store the domain discrepancy between the source and one specific target domain. With the domain discrepancy mainly stored in the TM module, the segmentation network should consequently mostly learn domain-invariant knowledge, making it suitable for every target domain and preventing CF.

To effectively store discrepancy information, the TM module is specifically designed for use in semantic segmentation, where both accurate localized and image-wide contextual information are important, and discrepancies must be captured for both. To this end, each module consists of two parts. The first one, a 1×1 convolution layer with a small receptive field to capture localized information. The second one, an average pooling module with a large receptive field to capture context. The full structure of the TM module is illustrated in Figure 3.2b.

The TM modules are included in the network structure by positioning them parallel to the layer behind which the adversarial loss is calculated. They are connected as shown in Figure 3.2a. Therefore, if adversarial losses are calculated at multiple positions, also multiple TM modules have to be added per domain. To capture the domain discrepancy, each TM module is trained for a specific target domain alongside the segmentation network and frozen thereafter. When making predictions, only the memory modules for the particular domain are activated to support the segmentation network.

The full structure of the ETM framework for multiple target domains is illustrated in Figure 3.2a.

Double-Hinged Adversarial Loss – A New Loss Function to Improve Domain Alignment

In typical GAN applications, the role of the generator is to produce outputs as close to the source as possible. The objective of the standard generator loss is therefore always only to bring the target features closer to the source, but never the opposite. Kim et al. [13] argue that for semantic segmentation this process is not ideal because the goal of the segmentation network is not to imitate the source domain but rather to align target and source domain. Since this alignment can be achieved in two ways, either by bringing the target features closer to the source, or vice versa by bringing the source features closer to the target, they argue that an ideal loss should also allow for both. To this end, Kim et al. [13] introduced a new loss function called double-hinged adversarial loss (DHA).

Let $\boldsymbol{z}^0$ and $\boldsymbol{z}^t \in \mathbb{R}^{H \times W}$ be the output values of the discriminator after passing a source image I^0 and target image I^t through the segmentation network. The DHA loss for the segmentation network (Generator) is defined as

$$\mathcal{L}_{\text{adv}}^{\text{DHA}}\left(\boldsymbol{z}^0, \boldsymbol{z}^t\right) = \sum_{h=1}^{H} \sum_{w=1}^{W} \operatorname{ReLU}\left(z_{hw}^0 - z_{hw}^t\right). \tag{3.2}$$

In simple terms, this loss function evaluates how close the source and the target are to each other and prevents changes to the segmentation network as soon as the target features are closer to the source side than the source features themselves.

The role of the discriminator is the same as in traditional GAN, and its loss is

$$\mathcal{L}_{\text{D}}^{\text{DHA}}\left(\boldsymbol{z}^0, \boldsymbol{z}^t\right) = \sum_{h=1}^{H} \sum_{w=1}^{W} \left[\operatorname{ReLU}\left(1 - z_{hw}^0\right) + \operatorname{ReLU}\left(1 + z_{hw}^t\right)\right]. \tag{3.3}$$

Knowledge Distillation to Limit Changes to the Segmentation Network

Finally, even if the TM module captures most domain invariance and is frozen after being trained on one domain, the segmentation network still changes between domains, thus leading to forgetting of old ones. To limit this effect, an additional loss is added to the optimization problem to distill knowledge [49] with the model trained on the previous target domain. This loss limits the changes that are being made to the current model by penalizing predictions that differ from those that the segmentation network would have done before training on the current domain. As before, $\boldsymbol{Y}^0, \boldsymbol{Y}_{\text{old}}^0 \in \mathbb{R}^{C \times H \times W}$ are the soft predictions of the current, respectively old segmentation network for an input image I^0 from the source domain. The distillation loss can be written as

$$\mathcal{L}_{\text{distill}}\left(\boldsymbol{Y}^0, \boldsymbol{Y}_{\text{old}}^0\right) = -\sum_{i=1}^{C \cdot H \cdot W} \hat{y}_{\text{old},i}^0 \log\left(\hat{y}_i^0\right), \tag{3.4}$$

with

$$\hat{y}^0_{\text{old},i} = \frac{\exp\left(Y^0_{\text{old},i}/T'\right)}{\sum_j \exp\left(Y^0_{\text{old},j}/T'\right)},\ \hat{y}^0_i = \frac{\exp\left(Y^0_i/T'\right)}{\sum_j \exp\left(Y^0_j/T'\right)}, \tag{3.5}$$

and the distillation temperature T' to soften the predictions.

Objective Function

To make the final formulation of the objective functions more concise, some extra notation is introduced in the following. Let $f^{[:m]}(x)$ be the output value when the input x is passed from the start to the m-th layer of the segmentation network (without any TM modules) and let $f^{[n:]}(x)$ be the complement, i.e. the output value when x is passed from the $(n+1)$-th layer till the end. Furthermore, if we define

$$f^{[:k]+}(x) = f^{[:k]}(x) + \text{TM}\left(f^{[:(k-1)]}(x)\right) \tag{3.6}$$

as the combined sum of the output of the segmentation network and the TM module at layer k, then the final output after the segmentation network is

$$f^{+}(x) = f^{[k:]}\left(f^{[:k]+}(x)\right). \tag{3.7}$$

With this notation, the training objective for the segmentation network (generator) and the discriminator is as follows:

$$\min\left[\lambda_{\text{seg}} \cdot \mathcal{L}_{\text{seg}}(\boldsymbol{Y}^{0+}, \boldsymbol{S}^0) + \lambda_{\text{adv}} \cdot \mathcal{L}^{\text{DHA}}_{\text{adv}}(z^{0+}, z^{t+}) + \lambda_{\text{distill}} \cdot \mathcal{L}_{\text{distill}}(\boldsymbol{Y}^0, \boldsymbol{Y}^0_{\text{old}})\right] \tag{3.8}$$

and

$$\min\left[\mathcal{L}^{\text{DHA}}_{\text{D}}(z^{0+}, z^{t+})\right], \tag{3.9}$$

with

$$\boldsymbol{Y}^{0+} = f^{+}(I^0),\ \boldsymbol{Y}^0 = f(I^0),\ \boldsymbol{Y}^0_{\text{old}} = f_{\text{old}}(I^0),\ \hat{z}^{0+} = D\left(f^{[:k]+}(I^0)\right),\ \hat{z}^{t+} = D\left(f^{[:k]+}(I^t)\right). \tag{3.10}$$

$\mathcal{L}_{\text{seg}}$ is the cross-entropy loss that trains the model for good segmentation performance using images and the given labels from the source domain.

Network Architecture

The Expanding Target specific Memory (ETM) framework makes use of the AdaptSegNet [8] as the segmentation model. This network is based on DeepLab-v2 [5] but adds a second ASPP classifier at the second to last layer of the ResNet101 [21] feature extractor. Two discriminators are used, the first one working on the output of ASPP (1) and the second one on ASPP (2). Each discriminator consists of five fully convolutional layers as introduced by [8]. For every target domain, two TM modules are added right before the discriminators parallel to ASPP (1) and ASPP (2). Due to the fact that there are two ASPP classifiers and two discriminators, all previously introduced losses must be calculated twice and added together to form the final training objective.

Unlike the NST-based approaches, the segmentation network is never individually trained on the source domain but instead directly begins adapting to the first target domain. Since no previous model is available for comparison, no knowledge distillation is undertaken on this first target domain.

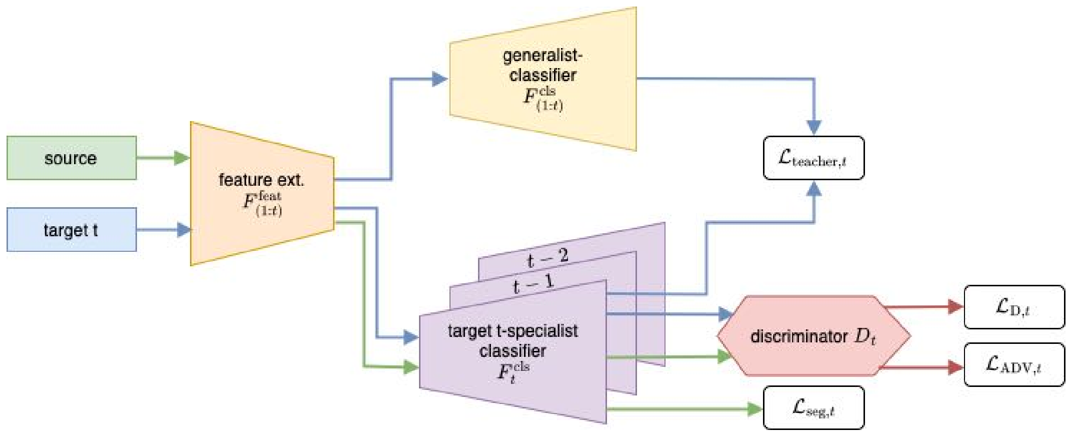

generalist-classifier $F^{\text{cls}}_{(1:t)}$
source
target t
feature ext. $F^{\text{feat}}_{(1:t)}$
$\mathcal{L}_{\text{teacher},t}$
t − 2
t − 1
target t-specialist classifier F^{cls}_{t}
discriminator D_t
$\mathcal{L}_{\text{D},t}$
$\mathcal{L}_{\text{ADV},t}$
$\mathcal{L}_{\text{seg},t}$

Figure 3.3.: Overview of the architecture of MuHDi consisting of one generalist head and many target specialist heads. The different specialist heads are trained for good segmentation performance on the source domain as well as good distribution alignment with one target domain. The knowledge of the different specialist heads is distilled into the generalist head, so this one performs well for all domains. The figure is based on [15].

3.0.4. MuHDi – Continual UDA Using an Alignment in Output Space and Advanced Distillation to Prevent Catastrophic Forgetting

The fifth framework for continual UDA is the one introduced by Saporta et al. [15]. While they are also using an adversarial approach for UDA, they aim to minimize catastrophic forgetting with advanced distillation methods. More precisely, they are performing distillation not only between the current and old model, but also introduced a second distillation between a target-specific specialist classifier head and a domain-independent generalist head.

Let $M_{(0:t)}$ be the continual segmentation model trained from the source domain up to domain t. This model $M_{(0:t)} = [M^{\text{feat}}_{(0:t)}, M^{\text{cls}}_{(0:t)}]$ can be separated into two parts, the feature extractor $M^{\text{feat}}_{(0:t)}$ and the generalist classifier head $M^{\text{cls}}_{(0:t)}$. For every target domain $t \in [1, T]$, there also exists a second specialist classifier head M^{cls}_{t} to form the domain-specific specialist model $M_t = [M^{\text{feat}}_{(0:t)}, M^{\text{cls}}_{t}]$. This specialist model is adversarially trained to perform well on one target domain. The generalist model $M_{(0:t)}$ is not trained on any target domain. Instead, target-specific knowledge is distilled into the generalist head from the specialist. In doing so, the generalist should be able to only learn relevant knowledge about new domains without forgetting much information about previous ones, thus preventing CF. A schematic overview of the multi head architecture is presented in Figure 3.3. A more detailed look at the training objectives of specialist and generalist is given in the following.

A Specialist Head to Adapt to New Target Domains

The training objective of the specialist head is twofold. First, it is trained to handle the adversarial alignment between source and target in the output space. Second, it is trained for good segmentation performance with samples from the source domain. For the first objective, the head is associated with a discriminator D_t tasked to distinguish segmentation predictions from the source and the target domain. To make the following better readable, we slightly adjust the notation compared to before. Let $\boldsymbol{Y}_t[I]$ be the predictions of M_t for input image I, then, following the original GAN publication [23], the discriminator is trained to minimize the binary cross-entropy loss (BCE)

$$\mathcal{L}_{D_t}\left(I^0, I^t\right) = \mathcal{L}_{\text{BCE}}\left(D_t\left(\boldsymbol{Y}_t[I^0]\right), 1\right) + \mathcal{L}_{\text{BCE}}\left(D_t\left(\boldsymbol{Y}_t[I^t]\right), 0\right). \tag{3.11}$$

The specialist model M_t (segmentation network) tries to trick the discriminator. It is trained by minimizing

$$\mathcal{L}_{\text{ADV},t}\left(I^t\right) = \mathcal{L}_{\text{BCE}}\left(D_t\left(\boldsymbol{Y}_t[I^t]\right), 1\right). \tag{3.12}$$

As with all other frameworks, the second objective, i.e. the segmentation performance of the specialist model on the source domain, is optimized using the cross-entropy loss

$$\mathcal{L}_{\text{seg},t}(I^0, \boldsymbol{S}^0) = \text{CrossEntropy}\left(\boldsymbol{Y}_t[I^0], \boldsymbol{S}^0\right). \tag{3.13}$$

A Generalist Head to Prevent Forgetting

The generalist segmentation head $F^{\text{cls}}_{(0:t)}$ is trained to perform well on all domains. It learns knowledge about the current target domain from the t-th specialist head via knowledge distillation. In a teacher-student setting, this transfer of knowledge is achieved by minimizing the Kullback-Leibler (KL) divergence between the predictions of both segmentation heads for the same target sample. To make the formulation of the loss function better readable, $\boldsymbol{Y}_t[I^t][c, h, w]$ are the predictions of the t-th specialist model for input image I^t, indexed at class c and the position h, w. Following this notation, the teacher loss to train the generalist head is

$$\mathcal{L}_{\text{teacher},t}(I^t) = \sum_{c=1}^{C}\sum_{h=1}^{H}\sum_{w=1}^{W} \boldsymbol{Y}_t[I^t][c, h, w] \log \frac{\boldsymbol{Y}_t[I^t][c, h, w]}{\boldsymbol{Y}_{(0:t)}[I^t][c, h, w]}. \tag{3.14}$$

To further prevent catastrophic forgetting, additional distillation losses based on the model trained for the previous target domain $M_{(0:t-1)}$ are added to the training of the generalist model. To guarantee that the current generalist model $M_{(0:t)}$ will also perform well on previous domains, these losses are designed such that the currently trained model $M_{(0:t)}$ maintains similar outputs and intermediate feature activations as the previous model $M_{(0:t-1)}$ when faced with the same image from the source domain[1]. More specifically, the first loss $\mathcal{L}_{\text{KL},(1:t)}$ is based on the KL divergence and measures the distance between the soft predictions of the old and current model. The second loss $\mathcal{L}_{\text{LocalPod},(1:t)}$ measures the distance between intermediary feature activations from different layers of the segmentation network and implements the LocalPod distillation from [49].

Generalist and specialist heads are trained simultaneously. Therefore, combining all losses, the complete objective function for the segmentation network can be written as

$$\begin{aligned}\min\Big[&\lambda_{\text{seg}}\mathcal{L}_{\text{seg},t}(I^0, \boldsymbol{S}^0) + \lambda_{\text{adv}}\mathcal{L}_{\text{ADV},t}(I^t) + \lambda_{\text{teacher}}\mathcal{L}_{\text{teacher},t}(I^t) + \\ &\lambda_{\text{KL}}\mathcal{L}_{\text{KL},(1:t)}(I^0) + \lambda_{\text{LocalPod}}\mathcal{L}_{\text{LocalPod},(1:t)}(I^0)\Big].\end{aligned} \tag{3.15}$$

[1]Ideally one would use samples from all previous target domains, however, since we assume target domains are no longer accessible after training and that target domains are aligned to match the source, we are using the source as a proxy.

Network Architecture

The Multi Head Distillation (MuHDi) framework once again makes use of the AdaptSegNet [8] as the segmentation model. For all except the first target domain, the model is extended with two additional classifiers (ASPP) to create the generalist and specialist head. For the first target domain extending the model is not necessary since for this special case generalist and specialist are the same head. For the adversarial training, two discriminators are added to the model that operate on the outputs of the specialist heads ASPP (1.1) and ASPP (2.1). The discriminators share the same structure as those of ETM.

As with ETM, the model is never trained on the source domain. Instead, training directly starts with the first target domain. Also as before, there is no distillation performed for the first target domain.

4. Methods

Two different experiments were performed in order to compare all five previously described frameworks for continual UDA.

- The first experiment focused only on style transfer and made a visual comparison of how all three NST-based frameworks (CACE, C-WCT^2, and C-CMD) stylize images.
- The second experiment was the actual main experiment. In this experiment all five frameworks for continual UDA were compared by collecting results for different sequences of domains.

In the following, we explain the two experiments and motivate why both are useful for comparing and understanding differences in the performance of the five frameworks for continual UDA.

4.1. First Experiment: Visual Comparison of Style Transfer Algorithms

In the first experiment (Section 5), we took a step back from continual UDA and just compared the three different style transfer algorithms CMD, WCT^2, and the style transfer algorithm used in CACE. The comparison was done qualitatively by analyzing the visual impression of stylized results for all NST algorithms.

Analyzing the difference in stylization is important in order to understand:

- How different variations of the same NST algorithm (e.g. class-conditioned) perform and which one delivers the best result.
- How the different NST algorithms lead to different image stylizations (e.g. sharper edges, artifacts, colors). The choice of style transfer algorithm is the key difference when using style transfer for continual UDA. Therefore, differences in stylization directly lead to differences in continual UDA performance.

For every NST algorithm, we analyzed different variations to see how they lead to the best performance. For example, we analyzed the effect of class-conditioning and orphan labels.

With the best implementation of every NST algorithm identified, we then visually compared the three different NST algorithms in order to identify differences in the way they stylize images.

The insights we gained from this experiment are applied in the discussion section to understand and explain performance differences that exist between all three NST algorithms when used for continual UDA.

4.2. Second Experiment: Comparison of the Five Frameworks for Their Performance in Continual UDA

In order to compare all five frameworks for their performance in continual UDA, in Section 6, we applied them to three different and challenging sequences of domains.

The different sequences that provided different challenges were:

- A fully synthetic sequence consisting of 11 domains that diverge in the atmospheric conditions the images were generated in (e.g. fog, night, rain).
- A sequence consisting of real-world data. There are five domains that once again diverge in the atmospheric conditions.
- A third sequence that introduced two new challenges. First, the adaptation from a fully synthetic source domain to two target domains consisting of real-world data. Second, an additional distribution shift between the source domain and the second target domain since there is a difference in the country; the source domain consists of images taken in a western city, while the second target domain is localized in India.

The different sequences will be explained in detail when performing the experiment.

For each sequence and all frameworks, the task of continual UDA was performed in the following way, as already described in the theory section. Each sequence consists of one source domain and multiple target domains. The data from the source domain is labeled and available throughout the whole training. The unlabeled target domains arrive sequentially and can no longer be accessed after adapting to the given domain. The goal is to sequentially adapt to all target domains while preserving knowledge about all previous domains. All frameworks were evaluated for average segmentation performance across all domains, including the performance on the source domain.

5. First Experiment: Visual Comparison of Style Transfer Algorithms

The choice of style transfer algorithm takes a key part for NST-based continual UDA since it is the one part responsible for domain alignment. A detailed understanding of the different NST algorithms is important because:

- For best performance, the NST algorithms should be in their best possible form. Therefore, it is important to understand how different variations of an algorithm, e.g. class-conditioning, influence stylization performance.
- Different algorithms stylize images in different ways. Therefore, already from the styling performance conclusions can be drawn to later performance in UDA.

To get this understanding, we completed an in-depth analysis of the three NST algorithms CMD, WCT^2, and the algorithm used in CACE (class-conditioned AdaIN with an additional skip connection). First, we analyzed the performance of each algorithm individually for different variations. With the key properties recognized and the best implementation found, we then compared all different algorithms with one another.

The analysis was performed using images from the popular Cityscapes [6] and ACDC [50] datasets. Both datasets contain real-world images of street scenery. For ACDC, the dataset consists of images taken in four different atmospheric conditions; i.e. images taken in fog, at night, in rain, or with snow. The different NST algorithms were tested for the task of transferring an image from Cityscapes into the style of different conditions from the ACDC dataset. All images used for the analysis were generated at a resolution of 512×1024.

The analysis was performed only qualitatively by visually describing the results and pointing out special features. A quantitative evaluation was omitted because up to this point there is no one commonly used metric to evaluate the stylization performance of NST algorithms [18].

We aim to provide an illustrative example image for every point we are making. More stylized images can be found in the Appendix A.3.

5.1. Visual Analysis of the Style Transfer Algorithm CMD

5.1.1. Experimental Setting

Several variations of CMD were implemented, starting with the base implementation (1) that was explained in Section 2.5.1. One major drawback of this implementation is that the style transfer is done image-wide, leading to all the previously discussed problems. To fix this, a second improved version (2) has been implemented that transfers the style class-conditioned. If orphan labels appeared, they were handled by optimizing those pixels against the global style of the style image. Doing so is obviously not ideal since it reintroduces the original problem of dominating style aspects to at least parts of the image. It is especially a problem if there are a lot of classes and the chance of a mismatch between style and content is high. Therefore, a third version (3) has been implemented, where the chance of orphan labels was reduced by merging multiple classes with similar semantic meaning and similar style (e.g. car and truck or wall and house). Information about the different classes and which classes have been merged together is available in the Appendix A.2.

For the experiments, the parameters were set to *content_weight* = 0.1, *style_weight* = 0.9, and *learning_rate* = 0.01.

5.1.2. Results

Comparing the results of (1), (2), and (3) validated the expected disadvantage of non-class-conditioning the style transfer in (1). Often the target style was not reproduced realistically but only its dominant features. For the case of fog, for example, the result typically turned out dull and grayed over the whole image, omitting important characteristics such as a reduced long-distance vision. Images at night also suffered, with the results oftentimes being just uniformly darkened but individual aspects like the sky not stylized enough and remaining in the look of day. As expected, class-conditioning the style transfer in (2) and (3) improved on this and achieved better-stylized results. One example for the effect of class-conditioning can be seen in Figure 5.1a. The image created without class-conditioning (left) does not manage to fully represent the look of night compared to the image with class-conditioning (right).

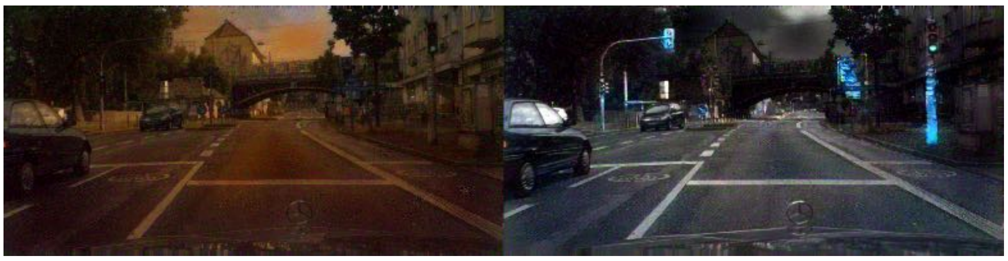
During the experiments it was noticed that orphan labels were a major problem for CMD. As can be seen in Figure 5.1b, CMD did not manage to make the respective pixels look like they are part of the image. Instead, all detail was lost for these parts and they became just greyish blurred blobs that made the image look distorted.

One possible approach to reduce the chance of orphan labels is to reduce the number of classes by merging similar ones. To verify the effectiveness of this operation, we first had to find an example where, due to reducing the number of classes, a previous orphan label became resolved. Our experiments validated the intuition that for these cases, the quality of stylization significantly improves from reducing the number of classes. A prime example of this is Figure 5.1c, where, by merging the classes parking and street, CMD was able to accurately stylize the parking space in the image on the right. Interestingly, even if there were no orphan labels, there was no performance degradation resulting from the reduction of classes. If not, the results of implementation (3) turned out better than those of (2). One explanation for this could be that by reducing the number of classes, the number of pixels per class is increased, thus leading to a more exact estimate of the feature distributions.

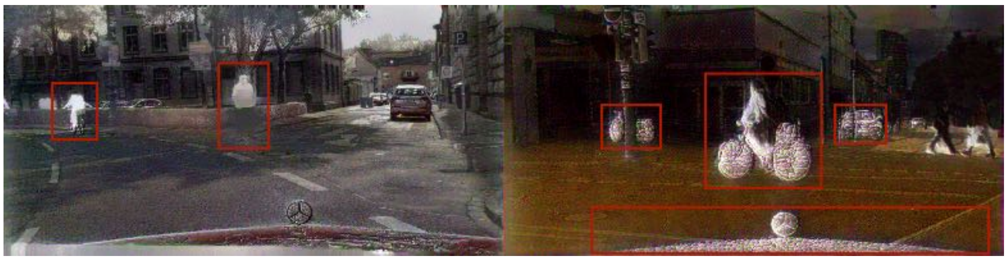
With implementation (3), images generated by CMD were typically well stylized. However, we noted that they were not perfectly photorealistic. The stylized image shown in Figure 5.1d is one example of this. It misses some fine details like the text of the license plates. Additionally, there is just a general feeling to the image that it does not look like a real photograph. It is hard to translate this feeling into words. We believe it has something to do with some edges being too sharp, while other parts are slightly blurred.

We attempted to overcome the former issue by adding an extra photorealism regularizer originally proposed by Luan et al. [36] to the optimization problem. Unfortunately, doing so did not help and the results were unsatisfactory. Photorealism was only improved for very few images and most of the time the photorealism regularizer instead introduced artifacts or caused optimization problems (Figure 5.1e). To make matters worse, the introduction of the photorealism regularizer also further increased the already long runtime of CMD by a few more minutes.

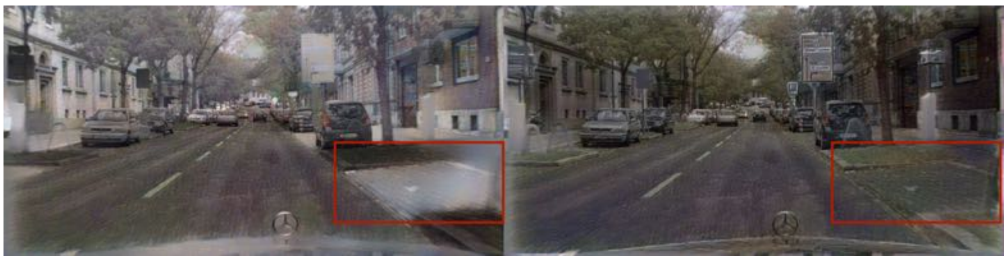
To sum up, with implementation (3) CMD managed to produce well and strongly stylized results. Sometimes, however, the results missed details and sharpness and did not completely turn out to look like a photograph anymore. Also, while the chance of orphan labels could be reduced by merging similar classes, for some images they were still an issue, and since CMD handled them so badly, those images typically turned out looking distorted.



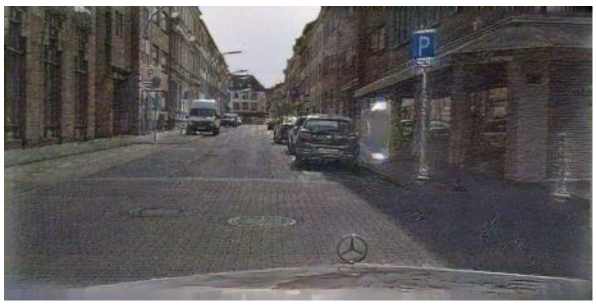
(a) Effect of class-conditioning the style transfer. The image on the left without class-conditioning is not stylized enough to represent the look of night.



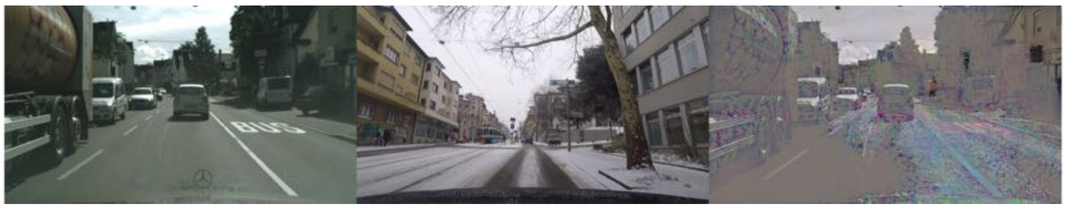
(b) CMD has major issues with the existence of orphan labels. The associated parts lose all their details and become distorted.



(c) Analysis of the effect of reducing the number of classes to prevent orphan labels. The image on the left is created with the full number of classes and the image on the right with a reduced number of classes. As can be seen, the stylization of the parking lot improves when reducing the number of classes.



(d) Some images created by CMD are not photorealistic. They are missing some sharpness as seen in the blurred license plates.



(e) Using the photorealism regularizer negatively influences the stylized result. There are many artifacts in the generated image. From left to right: content image, style image, and stylized result.

Figure 5.1.: Visual analysis of the style transfer capabilities of CMD for different implementations (e.g. effect of class-conditioning).

5.2. Visual Analysis of the Style Transfer Algorithm WCT^2

5.2.1. Experimental Setting

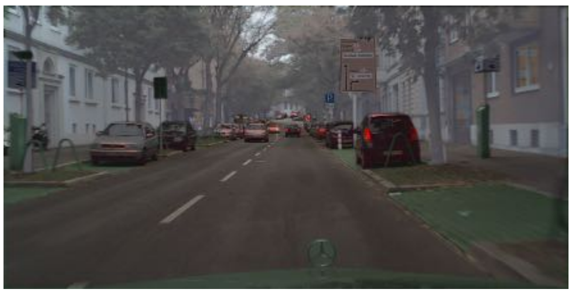
WCT^2 was implemented according to the Section 2.5.2. Three different variations were implemented. The first one (1), once again, transferring style non-class-conditioned, the second one (2) class-conditioned, and the third one (3) class-conditioned with a reduced number of classes. Orphan labels were handled by ignoring the associated elements in the feature maps when performing the style transfer (WCT operation).

To achieve maximum stylization, the mixing factor α was set to 1 and all WCT stages, i.e. encoder, decoder, and all skip connections, were activated.

5.2.2. Results

As expected, results from WCT^2 turned out very photorealistic. The results were very sharp, there were hardly any distortions, and WCT^2 was able to keep a very high level of detail, as shown in Figure 5.3 (right). While (1) suffered from the same limitations as previously discussed, class-conditioning the style transfer in (2) fixed those. As Figure 5.2a shows, orphan labels were still a disturbing factor but WCT^2 managed better with them. The resulting pixels remained sharp and only the color changed, typically to a green tone. Still, reducing the number of classes could improve stylization and led to the best results.

While WCT^2 managed to consistently stylize images well, there was just one common artifact disturbing the result. Often the results struggled from a 'halo effect' around the borders between classes which disturbed realism. Figure 5.2b for instance struggles from a strong halo effect around the borders of trees.


(a) WCT^2 orphan labels

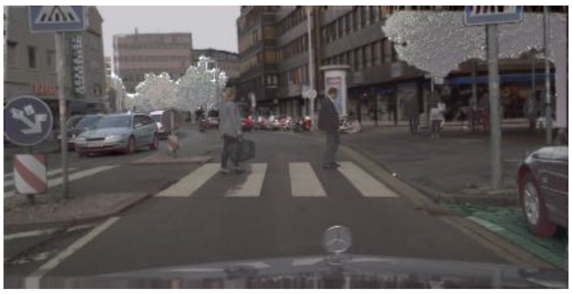

(b) WCT^2 halo effect

Figure 5.2.: (a) Compared to CMD, orphan labels do not affect WCT^2 that much. Affected regions remain sharp and only the color shifts to a green tone. (b) One common artifact with WCT^2 is a halo effect around the borders of objects. In this image, this effect is very pronounced around the trees.

5.3. Visual Analysis of the Style Transfer Algorithm from CACE

5.3.1. Experimental Setting

Finally, since the architecture of the style transfer network used in CACE is already predefined to be class-conditioned, experiments were only performed for this setting. The style transfer network, i.e. class-conditioned AdaIN, was implemented according to Section 3.0.1. As with WCT^2, orphan labels were handled by ignoring the associated elements in the feature maps when performing the style transfer.

5.3.2. Results

The results of the style transfer network from CACE were good and stylized results generally shared some similarities to those of WCT^2. CACE managed to consistently produce well-stylized results which matched expectations. The images were also photorealistic without any notable artifacts and most details preserved.

Still, CACE did not manage to achieve the same level of detail as the photorealism-focused WCT^2 did. One example of this is the brick pavement in Figure 5.3, which has lost some of its sharp edges during stylization.

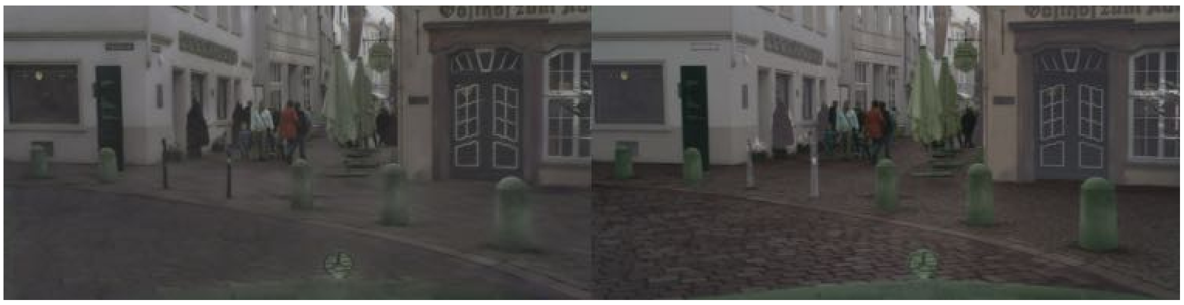


Figure 5.3.: Illustration of the enhanced photorealism achieved by WCT^2 as compared to CACE. WCT^2 (right) manages to preserve more detail in the fine structure of the pavement compared to CACE (left).

5.4. Conclusion

The results from CMD and WCT^2 validated the intuition that for best stylization performance the stylization must be class-conditioned. The results have also shown that orphan labels are an issue and that it is desirable to minimize their number to be as low as possible.

5.4.1. Final Comparison between All Three NST Algorithms

To finish this section, we now make a final comparison between all three different style transfer algorithms. For this purpose, using all three NST algorithms, one random image was

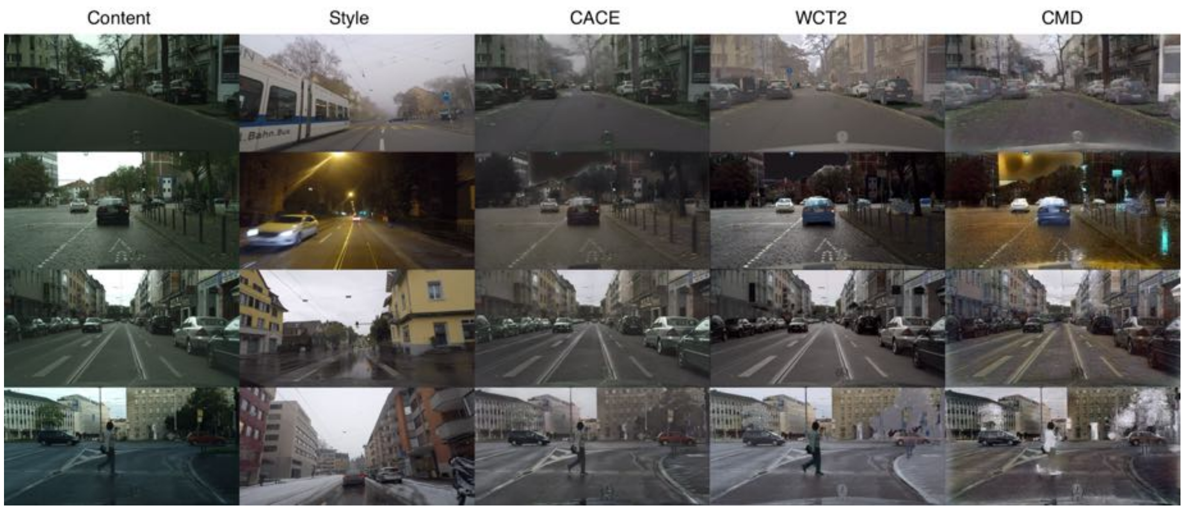


Figure 5.4.: Final comparison of the three style transfer algorithms WCT^2, CMD, and the one from CACE. For four different styles (from top to bottom: fog, night, rain, snow), one image is stylized per algorithm. Please zoom in to see finer details.

stylized for the styles of fog, night, rain, and snow. All methods used the same configuration, i.e. class-conditioned with a reduced number of classes. Figure 5.4 shows all twelve generated images.

In general, all methods were successful at transferring most of the style for all four atmospheric conditions (fog, night, rain, snow). There were, however, some differences depending on the specific environment. These differences are explained below.

Fog: For the style of fog (first row), CACE achieved the best results. Only it managed to reduce the long-distance vision and to blur the trees in the back. The other methods did not do this. WCT^2 just grayed out the whole image and with CMD mostly the individual cars were grayed out.

Night: The style of night (second row) achieved good results across all methods with especially the results of CACE and WCT^2 convincingly looking like night. The result of CMD was also plausible. There was just a bigger emphasis on the yellow color of the lights.

Rain: The style of rain (third row) was difficult. Here, no method managed to let the result look like it was raining. Still, they managed to transfer important characteristics like the darker color of the street.

Snow: For the final style of snow (fourth row), only CMD managed to create snow in the stylized result (trees in the top right corner). WCT^2 at least tried to color the trees but instead just generated the previously mentioned halo effect. CACE did not manage at all to stylize anything in the look of snow. The other characteristics of the style (e.g. colors) were transferred well by every method.

To conclude, the most important result which should be taken away from this comparison is that each method stylizes images differently. WCT^2 is slightly more photorealistic than CACE but also deviates in other respects so that the stylized images do not always share the same look. For CACE and CMD, the difference in stylization is even greater and often the two methods perform the style transfer very differently (e.g. the result at night). It will be interesting to see how these differences will later influence performance in continual UDA.

5.4.2. Runtime Comparison between All Three NST Algorithms

One last difference between the three methods, which has nothing to do with their performance but instead their practical usability, is the question of runtime. CMD is an optimization-based algorithm and required between five and ten minutes to style a single image. The runtime of both WCT^2 and CACE was several times less, with the former requiring only about 2 seconds per image. CACE was even more efficient since it does not require a single value decomposition for stylization. Additionally, CACE only has two AdaIN stages compared to WCT^2 with up to 15 but always at least four WCT stages[1].

[1]The exact number is dependent on the configuration. It is four when stylization is only performed at the encoder, 13 for stylization performed at the encoder and the skip connections, and 15 for stylization performed at the encoder, the skip connections, and the decoder.

6. Second Experiment: Comparison of the Five Frameworks for Performance in Continual UDA

We performed multiple experiments to compare the five frameworks (CACE, C-WCT^2, C-CMD, ETM, and MuHDi) for their performance in continual UDA, using three sequences of domains. The task of continual UDA that all frameworks were evaluated for has already been described in Section 3 and in Section 4.2. The results of the experiments are presented in this chapter.

In the following, we first present the experimental setting used to perform the experiments. We start with an in-detail presentation of the used sequences and datasets. We then explain the metric used for evaluating the performance of each framework. After that, we provide implementation details of all five frameworks and introduce two baselines we regularly refer to when presenting the results. Finally, after the experimental setup has been established, we report the results from all experiments. The results are reported individually for every sequence.

6.1. Experimental Setting

6.1.1. Datasets

Three challenging sequences were used for evaluating the five frameworks. The first two deal with the adaptation to changes in the atmospheric conditions (e.g. weather) and the last sequence with the adaptation from a synthetic source domain to two real-world target domains. The first two sequences of domains have been originally introduced in [10], the third sequence in [13].

The first fully synthetic sequence (SYNTHIA) consists of 11 different domains derived from the 'Old European Town' subset of the SYNTHIA-SEQ dataset [51]. Each domain contains approximately 1,000 images with 760×1280 resolution and is split equally into one training set and one validation set. The first domain (dawn) is treated as the source domain and all others are treated as target domains.

The second sequence (CS/ACDC) is more challenging and consists of five real-world domains. The source domain is made up of the Cityscapes (CS) [6] dataset containing $2,975$ training and 500 validation images with resolutions of 1024×2048. The four unlabeled target domains (fog, night, rain, and snow) originate from the ACDC [50] dataset and consist of approximately 400 training and 100 validation images of resolution 1080×1920 each.

We used 19 classes for the real-world dataset and 13 classes for the synthetic one. Information about the exact classes can be found in the Appendix A.2. For training, the shorter side of each image was rescaled to size 640 for the real-world sequence and 760 for the synthetic one, before it was randomly cropped to size 512×1024. Random horizontal flipping was used for additional data augmentation.

The third sequence (GTA/CS/IDD) starts with the synthetic GTA-5 [52] dataset consisting of $21,400$ train and $3,566$ validation images of resolution 1052×1914. The sequence continues with the first real-world target dataset Cityscapes. It ends with the IDD dataset [53] containing $6,993$ train and 981 validation images of Indian street scenes, each at a resolution of 1080×1920. The difficulties in this third sequence are twofold. First, the domain adaptation from synthetic to real, but also the domain discrepancy between western Cities (GTA, CS) and Indian cities (IDD).

18 classes were being used for this sequence. We did not perform any data augmentation and only rescaled all images to size 512×1024 before training.

6.1.2. Performance Metrics for Semantic Segmentation

The segmentation performance – the final objective that each framework aims to increase – was measured with the mean intersection-over-union (mIoU) metric. This metric was calculated individually per domain by averaging the intersection-over-union[1] (IoU) over all classes. It was also calculated sequence-wide by averaging the domain-wise mIoU values for all domains, including the source (mean mIoU).

For comparing the different frameworks, the sequence-wide performance after adapting to the last target domain (measured in mean mIoU) was used as the final evaluation criterion.

6.1.3. Implementation Details

Parameter Settings and Experimental Setup

For a fair comparison, all frameworks shared the same underlying segmentation network. They were using the established DeepLab-v2 [5] with a ResNet-101 [21] feature extractor. In agreement with general practice, the ResNet-101 feature extractor was pre-trained with weights from training on ImageNet [54] and all batch normalization layers were frozen during training.

Hyperparameter values were set according to established standards. For CACE, C-WCT[2], and C-CMD, we used the same values as presented in the CACE paper [10]. For MuHDi and ETM, we used the hyperparameters from the original implementations [13, 15].

In detail, the following values were selected:

The segmentation network was trained using SGD with a momentum of 0.9 and a weight decay of 5×10^{-4}. For the NST-based approaches, the learning rate was held constant at 2.5×10^{-4}. For MuHDi and ETM, it was continuously decreased with a polynomial decay starting from 2.5×10^{-4}.

[1]Equivalently known as the Jaccard index.

Additional settings required for the adversarial learning frameworks ETM and MuHDi

To allow efficient adversarial training, both MuHDi and ETM require adaptations to the segmentation network. They are adding a second ASPP-classifier head to the output of the second to last layer of the ResNet-101 to create the network structure of AdaptSegNet [8]. Two discriminators are added for every target domain. The discriminators were trained using the Adam optimizer with a learning rate of 1×10^{-4}.

ETM and MuHDi require extra parameters for the weighting between the segmentation losses, discriminator losses, and distillation losses. The chosen values are summarized in Table 6.1.

Table 6.1.: Additional hyperparameter values for ETM and MuHDi.

Hyperparameter	$\lambda_{seg}^{(1)}$	$\lambda_{seg}^{(2)}$	$\lambda_{adv}^{(1)}$	$\lambda_{adv}^{(2)}$	$\lambda_{distill}^{(1)}$	$\lambda_{distill}^{(2)}$	λ_{KL}	$\lambda_{LocalPod}$
ETM	0.1	1	0.0002	0.001	0.02	0.2	-	-
MuHDi	0.1	1	0.0002	0.001	-	-	10^{-5}	10^{-5}

Additional settings for the style transfer networks used in CACE, C-WCT^2, and C-CMD

Both the experiments from the original CACE paper [10] and the visual stylization analysis (Section 5) clearly showed the superiority of class-conditioning the style transfer. Therefore, all experiments were only performed for this setting. Additionally, for all variants of the frameworks where there is a chance or orphan labels, i.e. all variants without a style memory, some experiments were performed using a reduced number of classes for stylization. These experiments are all denoted with the suffix *reduced classes*. Information about which classes were combined to reduce the total number is available in the Appendix A.2.

For the style transfer networks, all three frameworks shared the same VGG-19 [20] encoder that was pre-trained on ImageNet [54] and then frozen. WCT^2 and the style transfer algorithm from CACE additionally require a decoder symmetric to the encoder.

For CACE, this decoder was initially trained for 30,000 iterations on the source and first target domain combined. It was then fine-tuned with an additional 5,000 iterations for each additional target domain.

The decoder for the WCT^2 style transfer algorithm used in C-WCT^2 was pre-trained with images from the MS-COCO dataset [55]. Fine-tuning the decoder on the source and target domains did not show an advantage and was therefore omitted. To achieve maximum stylization, all WCT stages were active and the mixing factor α was set to 1.

C-CMD uses an optimization-based style transfer algorithm (CMD) and therefore does not require a decoder. The weighting factors for content loss and style loss were set to $\lambda_{style} = 0.9$ and $\lambda_{content} = 0.1$. The learning rate was set to 0.01.

Number of Training Iterations Chosen for the Different Sequences

The number of training iterations were also selected according to established values. We mostly used the same number of iterations as chosen in the original implementations of CACE [10], ETM [13], and MuHDi [15]. If no existing reference existed because implementation diverged too much (e.g. C-CMD with its heavily restricted dataset size), we motivated

Table 6.2.: Summary of the number of training iterations for all five tested frameworks and all three sequences of domains.

		CACE	C-WCT^2	C-CMD	ETM	MuHDi
SYNTHIA	Source	50,000	50,000	50,000	not tested	not tested
	Target 1	10,000	10,000	5,000		
	Target 2, ...	10,000	10,000	5,000		
CS/ACDC	Source	100,000	100,000	100,000	100,000	100,000
	Target 1	10,000	10,000	5,000	40,000	40,000
	Target 2, ...	10,000	10,000	5,000	40,000	40,000
GTA/CS/IDD	Source	100,000	100,000	not tested	0	0
	Target 1	10,000	10,000		120,000	120,000
	Target 2	10,000	10,000		60,000	60,000

the chosen number of iterations by performing multiple experiments and selecting the best performing result.

Table 6.2 summarizes the number of iterations used for training on every sequence. Two things to note:

- Compared to CACE and C-WCT^2, the number of iterations on the target domains was halved for C-CMD. This was necessary because the way C-CMD integrates CMD reduces the dataset size to only one stylized image per target domain image. Thus, to prevent overfitting the number of iterations had to be reduced.
- When introducing ETM and MuHDi it had been stated that the adversarial learning approaches are not trained on the source domain at all. Mostly this was still the case as seen for the GTA/CS/IDD sequence, where training started with $120,000$ iterations on the first target domain. However, the size of the target domains of the CS/ACDC sequence is significantly smaller. Therefore, the number of iterations had to be reduced to prevent overfitting. To still guarantee that the segmentation model was being trained enough, it was pre-trained with 100,000 iterations on the source domain. It should be noted that no distillation was performed against this pre-trained model. Instead, the training on the first target domain was performed as usual.

6.1.4. Baselines

Two baselines are reported alongside the results of the five frameworks. The first baseline is the performance of the segmentation network when it was only trained on the source domain. It is denoted as *source-only*. The second baseline is the performance of the segmentation network when it was only trained on the source domain but color jittering had been used for additional data augmentation. This baseline is denoted as *jitter*.

6.2. Results

6.2.1. SYNTHIA

The results for the first sequence SYNTHIA are reported in Table 6.3.

CACE: As expected from [10], CACE performed well for the synthetic SYNTHIA sequence, turning out as the second best method overall. It managed to outperform source-only by a 13.6 % points mean mIoU and the color jitter baseline by 9.5 % points.

C-WCT^2: The C-WCT^2 framework performed even better. With a small lead of 0.3 % points mean mIoU over CACE, it achieved the best result out of all methods. Surprisingly, the best performance was achieved with the C-WCT^2 implementation that samples style from images (denoted as C-WCT^2). The supposedly superior memory-based implementation (denoted as C-WCT^2 style memory) still performed well but could not quite achieve the same performance and turned out slightly below the performance of CACE.

C-CMD: As previously described, some large trade-offs had to be made to integrate the style transfer algorithm CMD for continual learning (i.e. using fewer images but stylized with original not pseudo labels). The results must therefore be taken with caution and must always be interpreted with these trade-offs in mind.

Despite the implementation disadvantages, the performance of the C-CMD framework on SYNTHIA was good. It clearly outperformed source-only and the color jitter baseline could, however, not completely match up to CACE and C-WCT^2, and overall turned out worst out of the three.

ETM & MuHDi: The performance of the adversarial frameworks MuHDi and ETM was not tested for this sequence.

Table 6.3.: Collection of the results for the SYNTHIA sequence. All numbers are the performance after adapting to the final target domain. The best performance is presented in bold.

	Dawn	Fall	Fog	Night	Rainnight	Softrain	Spring	Summer	Sunset	Winter	Winternight	mean mIoU
Source-only	70.5	56.5	58.6	53.7	26.2	37.1	56.6	47.6	66.0	40.0	40.7	50.3
Jitter	70.3	62.4	64.0	62.3	29.2	38.7	57.5	53.8	70.6	40.4	49.7	54.4
CACE	**71.4**	69.0	**68.5**	66.5	49.1	57.9	67.3	69.7	71.3	53.7	59.1	63.9
C-WCT^2	71.0	**69.7**	67.7	**68.6**	43.5	53.1	**68.6**	**71.5**	**71.9**	**60.2**	**60.5**	**64.2**
C-WCT^2 style memory	71.1	69.6	67.6	68.8	42.5	51.4	68.5	71.4	**71.9**	58.2	60.1	63.7
C-CMD	69.1	66.5	66.6	65.8	**51.0**	**59.1**	66.3	68.0	69.3	60.0	57.5	63.3

Table 6.4.: Collection of the results for the Cityscapes/ACDC sequence. All numbers are the performance after adapting to the final target domain. The best performance is presented in bold.

	Cityscapes	Fog	Night	Rain	Snow	mean mIoU
Source-only	**71.5**	52.5	19.0	37.4	41.8	44.4
Jitter	70.7	**62.9**	22.6	45.0	46.6	49.6
CACE	69.1	62.1	25.3	**47.6**	**51.1**	**51.0**
C-WCT^2	69.6	53.4	23.9	42.8	43.9	46.7
C-WCT^2 reduced classes	68.4	57.1	23.4	47.0	44.2	48.0
C-WCT^2 style memory	69.3	60.7	26.1	45.3	48.9	50.1
C-CMD reduced classes	70.6	60.0	**29.3**	41.7	48.1	49.9
MuHDi	62.3	59.5	12.9	45.4	47.8	45.6
ETM	48.5	40.5	13.8	43.4	45.3	38.3

6.2.2. Cityscapes/ACDC

The results for the second sequence CS/ACDC are reported in Table 6.4.

CACE: Once again CACE performed strong, this time achieving the overall best performance for this sequence. It outperformed source-only by 6.6 % points mean mIoU and the color jitter baseline by 1.4 % points. However, compared to the SYNTHIA sequence, the performance gain was significantly smaller because the real-world sequence is more challenging [10] and, therefore, a good domain alignment is harder to reach.

C-WCT^2: The results of C-WCT^2 were not so clear this time. The implementation that samples style from images performed poorly. It performed only slightly better than source-only and was far behind the performance of color jittering let alone CACE. The CS/ACDC sequence has a large number of 19 classes, so the assumption was that orphan labels were an issue for the stylization. The experiments confirmed this, and by reducing the number of classes (denoted as C-WCT^2 reduced classes), the performance could be increased by 1.3 % points mean mIoU. Still, the performance was less than expected and far behind that of CACE. The most considerable performance improvement was achieved when using the style-memory. By doing so, orphan labels could be completely prevented, and the data variety was increased. With this implementation, C-WCT^2 finally managed to achieve good results and to outperform the color jitter baseline. Compared to CACE, however, the performance was still worse by almost 1 percentage point.

C-CMD: The last NST-based approach C-CMD performed all right. With a reduced number of classes for stylization (denoted as C-CMD reduced classes), the process which has already shown favorable for C-WCT^2, the C-CMD framework managed to outperform the color jitter baseline. However, the difference was not particularly large, and averaged over the whole sequence, C-CMD once again turned out as the worst performing NST-based approach.

One observation that stood out when comparing all three frameworks based on NST was that even though CACE was clearly best averaged over all domains, it was not the best in every domain. Instead, the performance of different methods was domain dependent and even the

Table 6.5.: Comparison of the segmentation performance of CACE, ETM, and MuHDi immediately after training on different target domains of the Cityscapes/ACDC sequence. Each row indicates the target domain on which the segmentation network was trained last. The columns indicate the segmentation performance for all domains at that time. The performance of the domain for which the segmentation network was last trained is in bold.

(a) CACE

	CS	Fog	Night	Rain	Snow
Fog	72.4	**61.9**	19.0	37.4	41.8
Night	70.6	57.6	**25.8**	44.8	45.9
Rain	69.1	59.0	27.3	**45.0**	46.1
Snow	70.8	60.8	27.3	46.0	**49.9**

(b) ETM

	CS	Fog	Night	Rain	Snow
Fog	66.4	**54.4**	15.5	39.7	40.0
Night	63.4	48.9	**19.2**	37.9	38.4
Rain	56.5	46.1	17.7	**47.1**	39.7
Snow	48.5	40.5	13.8	43.3	**45.3**

(c) MuHDi

	CS	Fog	Night	Rain	Snow
Fog	67.4	**57.1**	16.8	41.8	42.0
Night	66.8	56.3	**20.1**	41.6	40.1
Rain	66.6	60.8	15.3	**51.1**	47.2
Snow	62.3	59.5	12.9	45.4	**47.8**

overall worst performing framework C-CMD could achieve the best performance for one target domain (night). The difference was not even very close and C-CMD performed 4 % points better than CACE on this domain. In other domains, however, C-CMD performed very poorly. This was, for example, the case for the domain of rain, where C-CMD performed almost 6 % points worse than CACE.

ETM & MuHDi: The focus is now shifted towards the adversarial-learning-based frameworks. Both ETM and MuHDi performed poorly and could not match expectations, with the performance of ETM being particularly low. For no domain did ETM achieve results comparable with that of CACE. For the source and the first target domain, the performance even broke down completely with a difference to CACE larger than 20 % points.

MuHDi performed marginally better and had no performance drops as evident as with ETM. However, mean performance was still poor and ended up in the region of source-only.

Taking a closer look at the results of both ETM and MuHDi, it becomes clear that the main reason for their unsatisfactory performance is catastrophic forgetting. The tables summarized under 6.5 can help better understanding this issue. In each table, the columns represent the performance for one of the five domains (CS, fog, night, rain, snow). Every row indicates the domain up to which the model was trained before evaluating its performance.

What one can see from the tables is that the main problem of ETM and MuHDi was not adapting to new target domains (numbers written in bold). For every target domain, ETM and MuHDi were able to achieve an improvement over source-only immediately after training. Even though, compared to CACE, their performance was worse for three of the four target domains, these differences were not large enough to explain why ETM and MuHDi performed that poorly when looking at the final results after training has been completed for the whole sequence.

Instead, it becomes clear from the tables that ETM and MuHDi have problems preserving knowledge leading to strong forgetting.

For ETM in particular, it is easy to see that performance for each domain steadily decreased the longer it has been since training was performed for one domain; a clear expression of CF. Although not quite as pronounced, the same behavior did also hold for MuHDi, where for the domains Cityscapes (source), night, and rain, the performance substantially decreased after training.

To put the results of the adversarial-based frameworks into perspective, it is good to compare their behavior with that of CACE. Table 6.5a shows that CACE was good at preventing CF, with the performance after adapting to a target domain remaining mostly constant and sometimes even increasing slightly. The same behavior also held for the other NST-based frameworks, which were equally successful at preventing CF.

6.2.3. GTA-5/CS/IDD

For the last sequence GTA/CS/IDD, the results are reported in Table 6.6.

CACE: While CACE has already been tested for the previous two sequences [10], its performance was never evaluated for the synthetic-to-real use case that exists in the GTA/CS/IDD sequence. CACE also performed well for this sequence, outperforming the color jitter baseline for both target domains and achieving the best results out of all methods.

C-WCT^2: Only the implementation that uses a reduced number of classes for stylization could be tested for C-WCT^2. Using the style memory was not possible because with approximately 10,000 target samples, the memory would have grown too large.

The results of C-WCT^2 were also good and it managed to outperform the color jitter baseline for all domains. The performance was, however, behind that of CACE, which we at least partially credit to C-WCT^2 not being able to profit from the style-memory.

C-CMD: C-CMD could not be tested for this last sequence because even generating just one image per target style would be too time-consuming with thousands of target images.

For both tested NST-based frameworks (CACE, C-WCT^2), one observation stood out with this final sequence compared to the previous ones. The performance improvement achieved through UDA was much more dependent on the domain. So did CACE manage to improve over source only by an impressive 11.2 % points on the first target domain, but only achieved a significantly less impressive improvement of 3.4 % points on IDD. The same behavior was also observed for C-WCT^2. A possible explanation for this behavior will later be given in the discussion section.

ETM & MuHDi: Finally, still neither of the adversarial frameworks was able to achieve performance comparable to that of CACE.

This time ETM did a little better. The performance on the source domain was still very low. Performance on Cityscapes was better and the issue of CF was not as pronounced as before. Still, ETM did not come close to matching CACE's performance for this domain. For the last domain, however, ETM performed very well and managed to achieve the best performance out of all frameworks.

Table 6.6.: Collection of the results for the GTA-5/Cityscapes/IDD sequence. All numbers are the performance after adapting to the final target domain. The best performance is presented in bold. To evaluate the degree of forgetting, the performance difference before and after training on IDD is reported in brackets.

	GTA-5	Cityscapes	IDD	mean mIoU
Source-only	76.2	33.8	41.2	50.4
Jitter	75.0	40.6	42.7	52.8
CACE	**76.2**	**45.0** (-0.2)	44.6	**55.3**
C-WCT^2 reduced classes	76.1	44.0 (-1.5)	43.5	54.5
ETM	66.6	38.7 (-2.5)	**45.5**	50.3
MuHDi	71.2	33.6 (-6.0)	40.0	48.3

On the other hand, MuHDi did not show the same improvements and completely failed the task for the last sequence. It did not manage to achieve any improvements but instead turned out below source-only for all three domains. Two aspects played together to create this result. For one, poor UDA performance in adapting to the target domains, with the performance for the last domain (IDD) even turning out below source-only after training on this domain. Secondly, MuHDi suffered from strong forgetting, with the performance for the first target domain (CS) dropping by six percentage points after training on IDD.

7. Discussion

After having collected and presented the results of all five frameworks and the three sequences, these results are now used to answer the two research questions.

1. How does the choice of style transfer algorithm affect performance in continual UDA?
2. Which approach for continual UDA, style-transfer-based input space alignment or adversarial output space alignment, is more effective?

7.1. Question 1: How Does the Choice of Style Transfer Algorithm Affect Performance in Continual UDA?

7.1.1. A Detailed Look at the Effect of Enhancing Photorealism by Using the Style Transfer Algorithm WCT^2

Adapting CACE to integrate the photorealism-focused WCT^2 style transfer algorithm did not yield any benefits. Only for the synthetic SYNTHIA sequence did the C-WCT^2 framework manage to outperform CACE by a small margin. For the more important real-world CS/ACDC sequence, CACE has instead achieved better performance with a larger margin of 0.9% points mean mIoU. Not only was CACE better in average performance, it also performed best for three out of the four target domains. Finally, also for the GTA/CS/IDD sequence CACE outperformed C-WCT^2. The comparison for this sequence was, however, not entirely fair, since C-WCT^2 could not make use of the style memory.

Just from the performance values it can be concluded that integrating WCT^2 offered no advantage over CACE and in case of doubt, C-WCT^2 even performed worse. While this already makes a recommendation for using WCT^2 as the style transfer algorithm unlikely, it is completely precluded by the fact that when using WCT^2, there is no possibility for an efficient style memory. We currently see no possibilities to lower the memory requirements and thus no way for the C-WCT^2 framework to be an attractive alternative for CACE.

Originally, integrating WCT^2 has been performed to evaluate the effect of photorealism on continual UDA performance. It was expected that by enhancing the photorealism, continual UDA performance would also be increased. This was not the case, indicating that there is no advantage from enhancing photorealism over the level achieved by CACE. While this is a first indication, the experimental results are not enough to make a final evaluation on the importance of photorealism. This is because the WCT^2 style transfer network used in C-WCT^2 differs from the one in CACE not only in photorealism, but the entire structure and parameters of the networks are different. As seen in Section 5, this has the effect that both do not always transfer style in the same way, but instead diverge in more aspects than just photorealism. These differences in stylization are, for example, the reason why C-WCT^2 could outperform CACE on the night domain of the CS/ACDC sequence, despite being worse on every other domain.

It is this entanglement of different factors which makes it difficult to directly connect UDA performance and photorealism. To illustrate this, it might be possible that improving photorealism did not have a positive impact and the performance differences between CACE and C-WCT^2 were only due to differences in stylization. It might, however, also be possible that the enhanced photorealism was a relevant factor without which the performance of C-WCT^2 would have been lower.

Since for further research in the field of NST-based continual UDA, the question of the relevance of photorealism is of great importance, we now propose a new experiment that could help better answering this question. The idea is that to be able to uniquely trace back performance changes to differences in photorealism, everything else must be kept fixed. The proposed experiment is now to take the CACE framework and slowly start blurring the stylized images before training the segmentation network. If the performance does not degrade from reasonable amounts of blur, it is also not expected to rise from improvements in photorealism over the level CACE achieves.

We did not have time to perform this experiment but believe that it is an interesting idea to investigate in future work.

7.1.2. A Detailed Look at the Effect of Matching Higher Order Moments by Using the Style Transfer Algorithm CMD

As with WCT^2, adapting CACE to integrate CMD as the style transfer algorithm did not yield any benefits. Instead, the performance of the resulting C-CMD framework was worse for both tested sequences (SYNTHIA, CS/ACDC).

We expected that by using higher-order moments, the stylization and therefore also continual UDA performance would benefit. The experimental results do not allow for a complete evaluation of this statement.

Compared to CACE, the results of C-CMD were significantly worse, suggesting at first glance that there is no advantage to using higher-order moments. However, it must be taken into consideration that the experiments for C-CMD were not carried out under the same conditions as those for CACE. Compared to CACE, using real labels for the stylization is an advantage, but the less diverse dataset and the possibility of orphan labels are disadvantages. It is impossible to estimate the influence of these factors on continual UDA performance. Therefore, any comparison between C-CMD and CACE is difficult, and statements about the role of matching higher-order moments are hard to make.

A second experiment has been performed to better compare the performance differences between both frameworks. In this experiment, comparability was improved by taking the style transfer algorithm from CACE but using it in a setting like C-CMD. We first pre-stylized one image per target style and then used these images to train the segmentation network. The result immediately after adapting to each target domain (UDA performance) of the CS/ACDC sequence are presented in Table 7.1.

Table 7.1.: Improved comparison of the performance of CACE and C-CMD for the CS/ACDC sequence. For a fair comparison, both methods are adapted to be as similar as possible, meaning that they both implement the training procedure of C-CMD. Both first pre-stylize images using real labels. They then use these images to train the segmentation network. The reported results are the performance on every target domain immediately after adapting to this domain.

	Fog	Night	Rain	Snow
CACE	58.0	24.6	46.6	47.2
C-CMD	59.6	29.2	42.4	48.1

The results were inconclusive. For two domains C-CMD performed better, for two domains CACE, and averaged over all domains the performance was similar. The differences between the target domains were, however, very large. So did C-CMD perform better for the domain of night by a large 4.6 % points, while CACE performed better on rain by 4.2 % points.

The results therefore confirmed the expectation that differences in stylization play a large role in domain adaptation. They could, however, not show any advantage of using CMD that holds when averaging over different domains.

7.2. Question 2: Which Approach for Continual UDA, Style-Transfer-Based Input Space Alignment or Adversarial Output Space Alignment, is More Effective?

Comparing the results between the NST-based CACE framework (alignment in input space) and the adversarial-learning-based MuHDi and ETM frameworks (alignment in output space), it is clear that CACE is the superior method. While CACE reached state-of-the-art performance for all sequences, both adversarial-learning-based methods only achieved unsatisfactory results located in the region of source-only or even lower.

To best understand where these performance differences come from, it is useful to split the analysis into two parts:

- CACE, ETM, and MuHDi use different approaches for UDA. While CACE aligns domains in the input space, the other frameworks align domains at the output level. By focusing only on the performance in adapting to new target domains (UDA), we want to see if there are noticeable differences between both approaches.
- In overall continual UDA performance ETM and MuHDi performed very poorly. We have already seen that CF is a major reason for this, so we will focus on this issue in the second part.

7.2.1. Comparing the Performance of Input Space Alignment and Output Space Alignment in Adapting to New Target Domains

The experiments have shown that both input space alignment (CACE) and output space alignment (ETM, MuHDi) is generally effective for adapting to new target domains. This can be seen as directly after adapting to a particular target domain, all three frameworks managed to improve performance over that of source-only most of the time. Nevertheless, there were some clear differences between the UDA performance achieved by CACE and that of ETM and MuHDi.

In general, the UDA performance of CACE was better than that of ETM and MuHDi for most target domains. The tables 6.5 have already been introduced previously. They show the performance for all domains after adapting to a given target domain of the CS/ACDC sequence. Focusing only on the performance for each target domain immediately after the model has been trained for this domain – in our opinion the best measurement for UDA performance – it can be seen that CACE outperformed both MuHDi and ETM for three out of the four target domains.

The results of CACE were also more constant. So did CACE manage a notable increase in performance for all four target domains of this sequence. ETM and MuHDi instead struggled with the domain of night, where they could hardly improve performance over that of source-only. The reason for ETM and MuHDi struggling with the domain of night is not completely clear. One possible explanation is that the assumption of structural similarity in the output space takes a hit for the night domain, as in general, images at night show fewer cars, cyclists, and people.

The third GTA/CS/IDD sequence (Table 6.6) can give some more insight about the capabilities of input space alignment and output space alignment. In [13] the authors undertook a similar experiment to ours and compared the performance of multiple continual UDA frameworks for the GTA/CS/IDD sequence. In particular, they analyzed the performance of the CACE predecessor ACE. In their experiments, ACE completely failed the task and was hardly able to improve performance on the first target domain (CS) and not at all able to improve performance on the second target domain (IDD). From this, they concluded that NST-based approaches are 'unsuitable for the real-world problems with large domain discrepancy' [13, p.6]. Our results for CACE (Table 6.6) refute this claim, as we were able to improve performance for both target domains. Especially for the first target domain Cityscapes, the increase in performance was impressive, showing that an alignment in input space can overcome the domain gap from synthetic to real.

For the second target domain (IDD), the result was not that clear and CACE only achieved a substantially smaller improvement over source-only. When adapting from GTA-5 to IDD, there is a second challenge besides that of synthetic data to real data. While the GTA-5 source domain is situated in a Western city, the IDD domain is situated in India. Many characteristics diverge between these places, such as the architecture or the types of vehicles that are common. We believe that NST-based domain alignment struggles with this type of domain discrepancy because it cannot be simply modeled as a change in style.

While these differences in the environment make domain adaptation difficult for NST-based approaches, it was impressive to see that CACE remained partially effective and managed some performance improvements. Still, for best performance the change in the country must be factored in. We believe that changes in, for example, the architecture can only be factored in when aligning the domains in the output space where the representation is more abstract, hence the better performance of ETM on the IDD domain.

To conclude, both aligning distributions in input space using NST and aligning distributions in output space have been proven to be effective for the task of UDA in semantic segmentation. Overall, the performance of aligning in input space was better and more constant. There is just one use-case where we see an advantage of output-space alignment; output space alignment can remain effective when the domain discrepancy cannot be modeled with a style transfer.

7.2.2. Why Do the Adversarial Approaches Perform so Poorly – Understanding the Issue of Catastrophic Forgetting

In the last section, it was shown how for most cases input space alignment in the form of CACE is more effective for UDA than output space alignment in the form of ETM and MuHDi. While noticeable, the difference was not large enough to explain the very poor performance for continual UDA that was observed for both ETM and MuHDi. Instead, two other factors were responsible that both have something to do with the CL capabilities of the frameworks. These factors are poor performance on the source domain, paired with the inability to effectively prevent CF.

Adversarial-learning-based approaches struggle to achieve the same source domain performance as NST-based ones because they are never trained on the source domain individually[1]. While the segmentation performance on the source domain is always a part of the objective function, it is only one part and weighted against other losses like the one of the discriminator. Classically, adversarial approaches are therefore only used if the performance on the source domain is not of interest, as it is often assumed when adapting from a synthetic source domain to a real-world target domain[2].

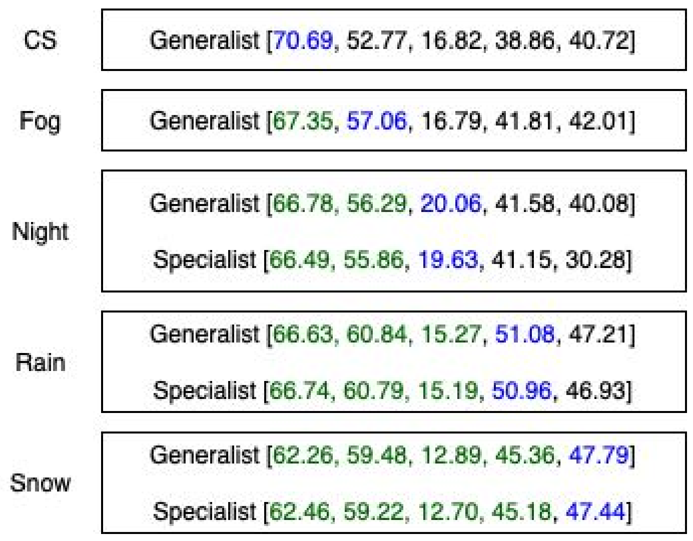
However, even if the performance on the source domain is excluded, MuHDi and ETM still performed worse than CACE for both tested sequences. This is due to the issue of CF, which ETM and MuHDi could not prevent sufficiently. For some combinations, e.g. ETM when applied to the CS/ACDC sequence or MuHDi when applied to the GTA/CS/IDD sequence, the issue of CF was even so bad that it questions the effectiveness of both frameworks for continual learning. To evaluate the effectiveness of both frameworks for preventing forgetting, a detailed analysis is performed in the following.

Analyzing the Issue of CF for the Architecture of MuHDi

In the MuHDi architecture, there is always a specialist and a generalist head trained simultaneously. The specialist is trained for segmentation on the source domain and domain alignment with the target domain. The generalist uses distillation to combine previously learned

[1]For the GTA/CS/IDD sequence, we actually pre-trained the model on the source domain. However, since distillation was never performed against this pre-trained model, strong forgetting was observable when continuing the training on the first target domain. Pre-training therefore did not help to substantially improve performance on the source domain and performance remained poor.

[2]Not in our experimental setting though.



CS	Generalist [70.69, 52.77, 16.82, 38.86, 40.72]
Fog	Generalist [67.35, 57.06, 16.79, 41.81, 42.01]
Night	Generalist [66.78, 56.29, 20.06, 41.58, 40.08]
	Specialist [66.49, 55.86, 19.63, 41.15, 30.28]
Rain	Generalist [66.63, 60.84, 15.27, 51.08, 47.21]
	Specialist [66.74, 60.79, 15.19, 50.96, 46.93]
Snow	Generalist [62.26, 59.48, 12.89, 45.36, 47.79]
	Specialist [62.46, 59.22, 12.70, 45.18, 47.44]

Figure 7.1.: Comparison of the segmentation performance of the generalist head and the specialist head collected over a full training on the CS/ACDC sequence. The blue color denotes the domain that was currently trained on. The green color denotes all previous domains.

knowledge with the knowledge of the current specialist head to learn about new target domains while preventing forgetting.

The effectiveness of MuHDi to prevent CF can therefore be assessed by comparing the performance of the generalist head with that of the specialist head. If the chosen approach is effective, it is expected that the performance for previous target domains is better when evaluated with the generalist head, compared to when it is evaluated using the specialist head trained for a particular target domain. This should hold for all training steps, e.g. adapting to target domain one, adapting to target domain two, and so forth.

Our results for the CS/ACDC sequence, as illustrated in Figure 7.1, show this expected behavior for all domains except the source (Cityscapes). The source domain is a special case because here two effects are working against one another. For one, remembering already learned knowledge in the generalist head, but also training the specialist head for segmentation performance on the source dataset. Therefore, we argue that the source domain should be excluded when evaluating the effectiveness of MuHDi.

Overall, it can be concluded that our results align with those from [15], showing that the structure of MuHDi can help to prevent forgetting. It should, however, be noted that the advantage of the generalist head is oftentimes just marginally. Therefore, if there is a large domain discrepancy and the specialist head evolves a lot during training, the generalist head cannot compensate for this and substantial forgetting can still occur.

The training on the IDD domain of the CS/ACDC sequence is a prime example of this. Tracking the performance over the first 30,000 training iterations (Table 7.2), it can be seen that the performance for IDD is first dropping a lot, before then slowly recovering. The initial large performance drop indicates a large change to the NN (generalist head) in which it initially forgets much of the previously learned knowledge for the current, but also all previous target domains. Over the course of the training it then relearns the knowledge, however, primarily from the IDD dataset. On the other hand, most of the domain-dependent

Table 7.2.: Temporal evolution of the segmentation performance when using MuHDi to adapt to the IDD domain of the GTA/CS/IDD sequence. For different iterations, the performance on IDD as well as CS is presented.

Iteration	0	5,000	10,000	15,000	20,000	25,000	30,000
IDD	41.6	38.5	40.0	37.6	40.8	39.0	40.0
CS	39.6	29.6	34.1	31.2	33.3	33.2	33.9

information from the previous targets cannot be restored just by distilling from the reference model[3], thus leading to large forgetting of mIoU 6.0 % points on the first target domain Cityscapes (Table 6.6).

Analyzing the Issue of CF for the Architecture of ETM

The main idea of the ETM framework is that the segmentation network learns mostly domain invariant information, while the TM modules learn domain-dependent information, i.e. the domain discrepancy between the source domain and the target domain. However, already [13] have acknowledged that this statement is vague and that the real distribution of knowledge most likely diverges from this idealization. Two observations from the experiments indicate that the deviation from the idealization might be larger than originally assumed.

First, it was surprising to see how poor the performance for the source domain was for both tested sequences. If the TM modules capture most of the domain discrepancy towards the source domain, it was expected that the segmentation model without any TM module performs well for the source domain. This was not the case. Instead, source performance was better when using any TM module trained for a particular target domain, than when using no TM module at all (Table 7.3), indicating that the TM module must also contain some segmentation information relevant to the source domain.

In addition, when analyzing the performance for the CS domain of the GTA/CS/IDD sequence after a full training has been completed, it can surprisingly be observed that performance is better when the TM module trained on IDD is used, instead of the original one trained on CS (Table 7.3). Again, this indicates that the TM module trained alongside IDD contains more relevant information for Cityscapes than the CS module itself.

Taken together, these observations cast shade on whether the assumption that knowledge is divided between the segmentation network and the TM modules is correct and thus, whether the structure of the ETM framework is capable of preventing forgetting.

[3]The model which was trained up until the previous target domain and then saved to compare the outputs and feature activations.

Table 7.3.: Segmentation performance of ETM after being fully trained on the GTA/CS/IDD sequence. To evaluate the effectiveness of the TM modules, the performance for each domain using all combinations of TM modules is presented. The rows denote which TM module was being used when evaluating the segmentation performance. The columns denote the different domains performance values were collected for.

	GTA-5	Cityscapes	IDD
No TM	66.6	38.5	44.2
Module CS	68.2	38.7	44.5
Module IDD	70.9	39.5	45.5

8. Conclusion

Two different questions were analyzed in this work. First, by exchanging the style transfer algorithm in the CACE framework with two alternatives, we analyzed how changes to the style transfer algorithm lead to changes in continual UDA performance. Second, we compared style-transfer-based approaches for input-space alignment with adversarial-learning-based approaches for output space alignment to determine which method is superior for the task of continual UDA.

The first result was obtained just from the challenge of integrating the NST algorithms WCT^2 and CMD with the CACE framework. With all the difficulties that arose doing so, it became clear that a holistic alternative for the AdaIN-based style transfer algorithm used in CACE must fulfill more than just good performance in continual UDA. In particular, it must be efficient to implement in a feedforward manner and must allow for a very lightweight memory to store style information for replay.

Even ignoring practical usability, we did not see any performance advantages from using WCT^2 for the style transfer. We expected that from being more photorealistic, the framework using WCT^2 would achieve better performance in continual UDA. This was not the case. Instead, the performance of C-WCT^2 compared to CACE was at most equally good, if not worse. However, since it is difficult to infer directly from these results to the importance of photorealism, we proposed a new study that could help answer this question.

For the effect of matching higher order moments, the results of the experiments were inconclusive. The framework using CMD showed a clear advantage on the night domain but for other domains (e.g. rain) CACE performed much better. It can therefore not be concluded that matching higher order moments is always advantageous, but only that different methods for NST, with their differences in stylization, perform differently depending on the domain.

Finally, the comparison between style-transfer-based domain alignment in input space and adversarial-learning-based domain alignment in output space showed that the former one is the superior approach. It has become especially obvious that adversarial-learning-based approaches (ETM, MuHDi) are not yet able to sufficiently prevent CF and that a lot of research still needs to be done in this direction. In contrast to previous work, we have also shown that style transfer is able to overcome even large domain gaps such as the one between synthetic and real data. Even for the case that the domain gap is only partially in style (IDD), we have shown that NST-based approaches can remain effective to some degree. In such cases, however, the output-space-aligned methods without their limitations to style are in the advantage and can achieve better domain adaptation.

To give a final recommendation for practical application, all experiments have shown that the CACE architecture is currently the best performing one and is generally recommended for use. Not only did it provide best-in-class performance for every sequence, but it also managed to do so in a memory-efficient way suitable for real-world application.

A. Appendix

A.1. Approximation of the Style Memory Size Required by WCT^2

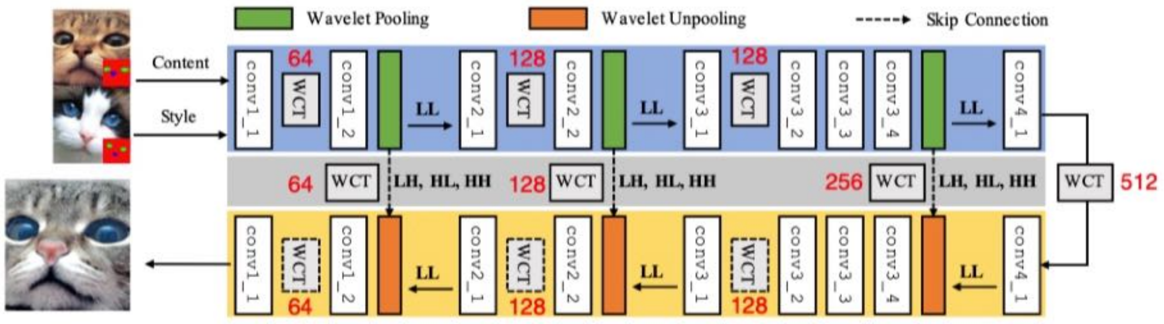


Figure A.1.: The architecture of the WCT^2 style transfer network, where for every WCT layer the number of channels/filters is noted in red. With minor adaptions, figure taken from [17].

Figure A.1 shows the full architecture of WCT^2 with the number of channels entering each WCT-module marked in red. According to Section 2.7, to store the style of an image when using the WCT operation, we must store both its channel-wise mean $\underline{\mu} \in \mathbb{R}^{K_l}$ and a stylization matrix. This matrix has maximum dimensions of $\mathbb{R}^{K_l \times K_l}$ depending on the number of eigenvectors in the covariance matrix. For every WCT module and every class, the style must be stored individually.

For every channel-depth K_l, Table A.1 summarizes the number of occurring WCT operations. It also states the maximum number of elements in the style matrix as well as the mean vector depending on K_l. Finally, in the last row Table A.1 summarizes the maximum number of elements summed over all WCT operations that are required to be stored. Beware that every skip connection adds three WCT modules, one for every component.

Table A.1.: Number of elements WCT^2 requires to store in the style memory.

K_l	Quantity	Elements in Style Matrix	Elements in Mean Vector
64	5	4096	64
128	7	16384	128
256	3	65536	256
512	1	262144	512
$\sum$		593920	2496

As shown in Table A.1, in the worst case 596416 values must be stored for each class. Assuming that each element is stored as a 32-bit floating point number (standard size of PyTorch) and that there are 19 classes, this gives about 45 MB for the stylization information of just one image.

In application, the required memory is typically smaller than this worst-case approximation. First, because the stylization matrix is not necessarily of size $\mathbb{R}^{K_l \times K_l}$ and second, because typically some classes are missing. Nevertheless, the memory requirement remains very high.

For comparison, the AdaIN operation used in CACE only requires to store the channel-wise mean vectors and the channel-wise variance vectors; both $\in R^{K_l}$. Additionally, CACE only performs style transfer at two stages, the first one with a channel depth of 64, and the second one with a depth of 512. Therefore, assuming 19 classes, storing one image takes approximately 48 kB, almost a factor of 1000 less than the memory required of WCT^2.

A.2. Additional Information about the Three Sequences Used for Evaluating the Performance of the Five Frameworks

Table A.2.: All 13 classes of the SYNTHIA sequence.

misc	sky	building	road	sidewalk	fence	vegetation	pole	car	traffic-sign	pedestrian	cyclist	lanemarking

Table A.3.: All 19 classes of the CS/ACDC sequence (top). Combination of different classes when the number was reduced for stylization (bottom).

road	sidewalk	building	wall	fence	trafficlight	pole	trafficsign	vegetation	terrain	sky	person	rider	car	truck	bus	train	motorcycle	bicycle
street		construction			*	poles		vegetation		*	human		vehicle					

Table A.4.: All 18 classes of the GTA/CS/IDD sequence (top). Combination of different classes when the number was reduced for stylization (bottom).

road	sidewalk	building	wall	fence	trafficlight	pole	trafficsign	vegetation	sky	person	rider	car	truck	bus	train	motorcycle	bicycle
street		construction			*	poles		*	*	human		vehicle					

A.3. Additional Stylization Results for the Three NST Algorithms and Different Variations in the Implementation

For each of the three NST algorithms (CMD, WCT^2, and the stylization algorithm from CACE), additional results are presented in the following section. For each algorithm and its different implementations, 16 additional images have been created.

For the following figures, the first column shows four content images to be stylized. The first row shows four style images; one each for the style of fog, night, rain, and snow. All other images are the respective combination of content and style image.

A.3.1. CMD

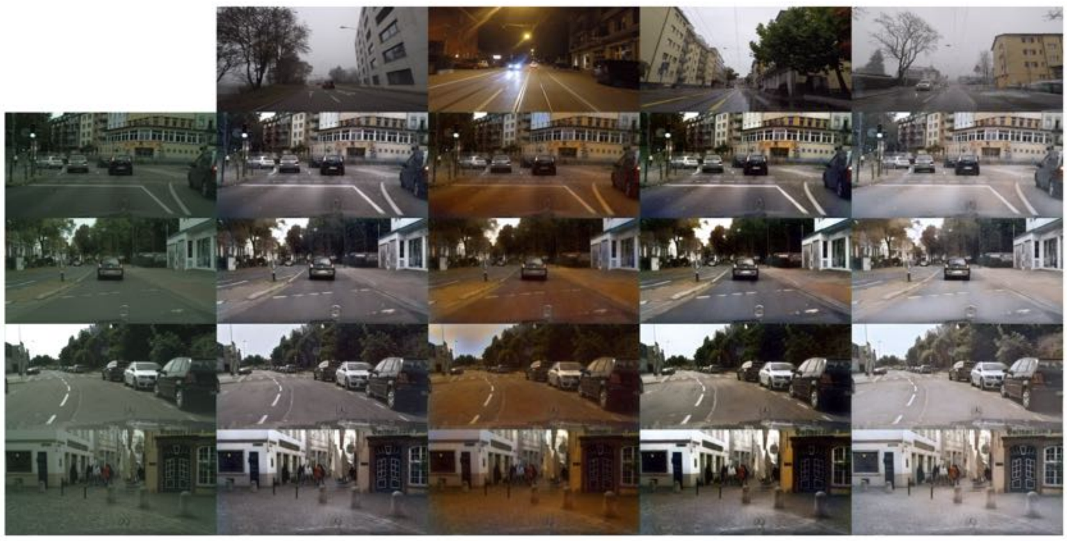


Figure A.2.: Additional stylization results for CMD. All images were generated without class-conditioning.

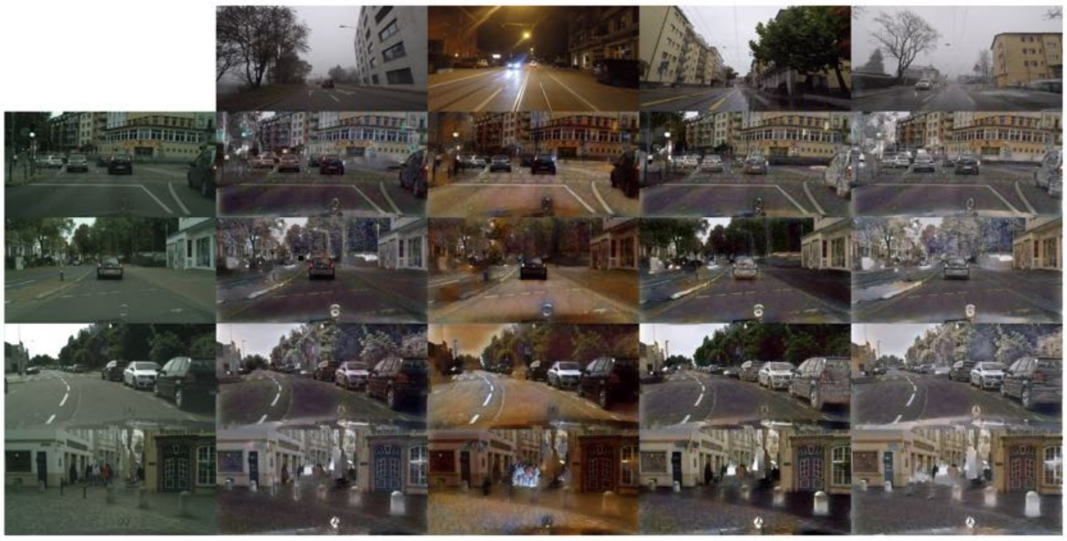


Figure A.3.: Additional stylization results for CMD. All images were generated with class-conditioning.

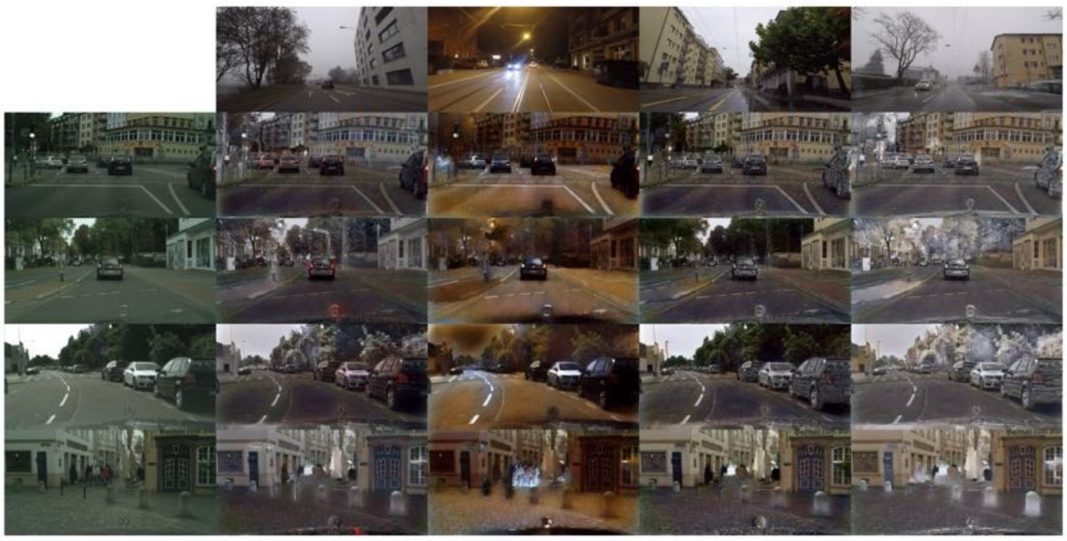


Figure A.4.: Additional stylization results for CMD. All images were generated with class-conditioning and a reduced number of classes.

A.3.2. WCT2

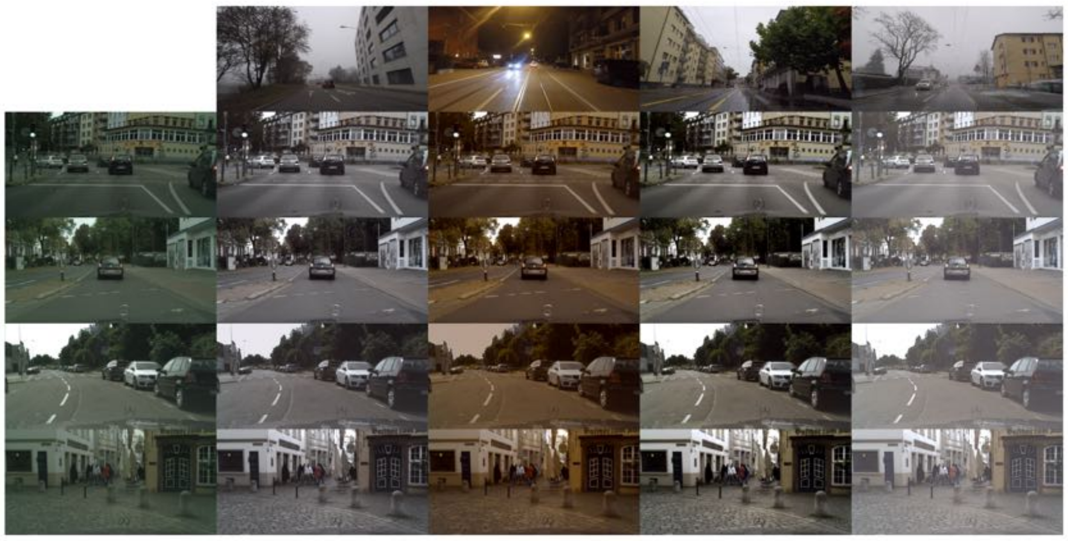


Figure A.5.: Additional stylization results for WCT^2. All images were generated without class-conditioning.

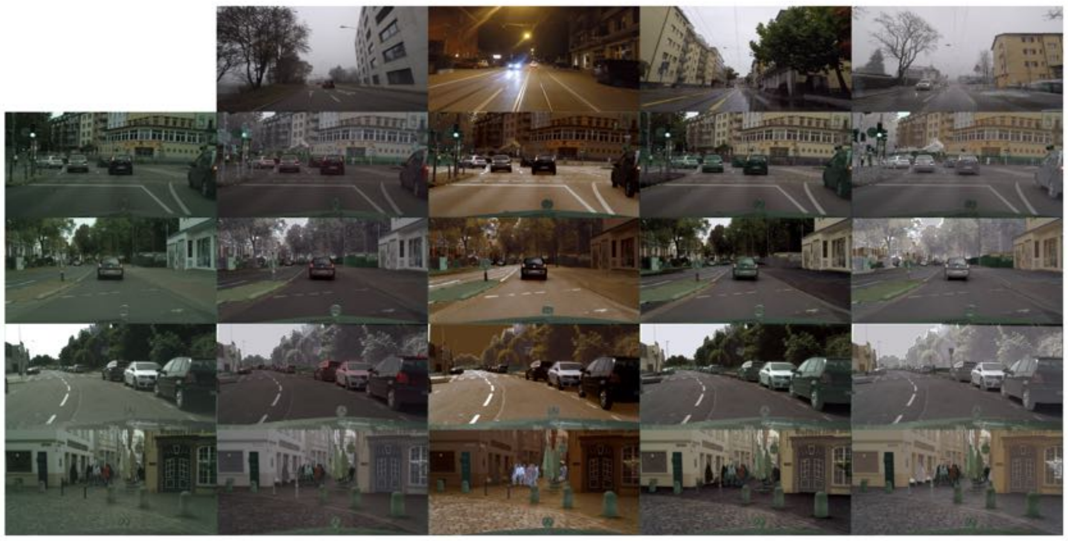


Figure A.6.: Additional stylization results for WCT^2. All images were generated with class-conditioning.

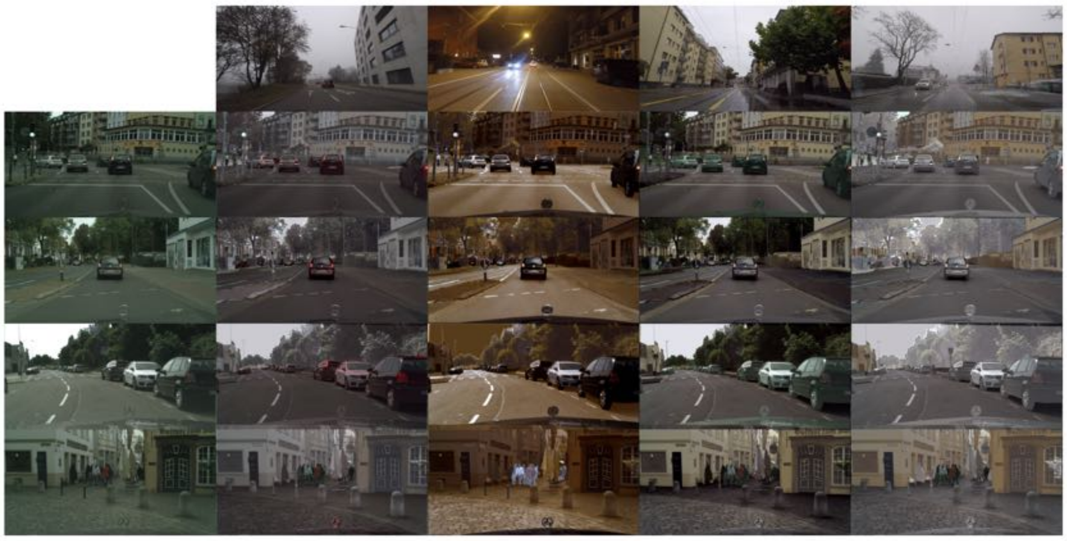


Figure A.7.: Additional stylization results for WCT^2. All images were generated with class-conditioning and a reduced number of classes.

A.3.3. CACE

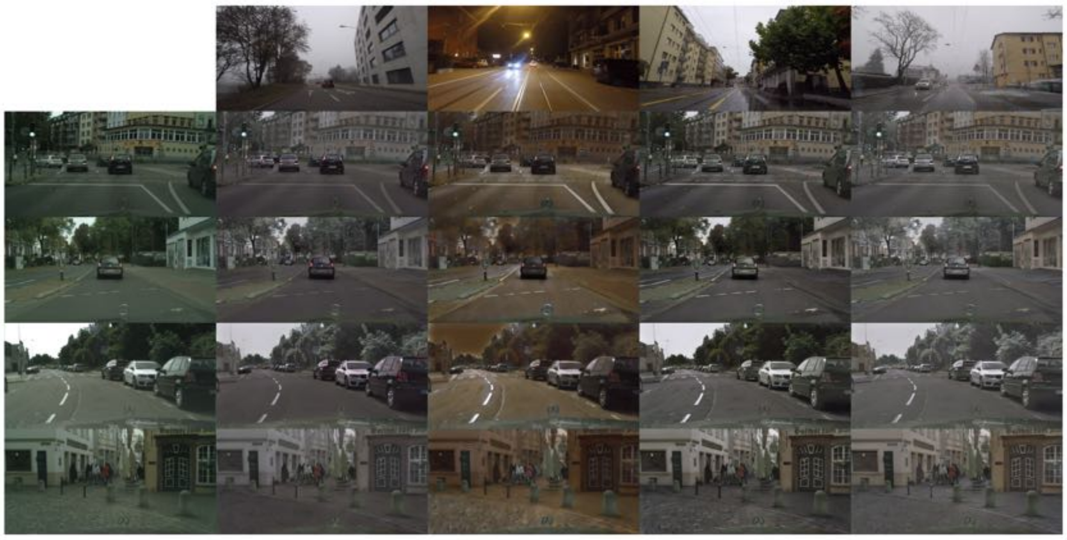


Figure A.8.: Additional stylization results for CACE. All images were generated with class-conditioning.

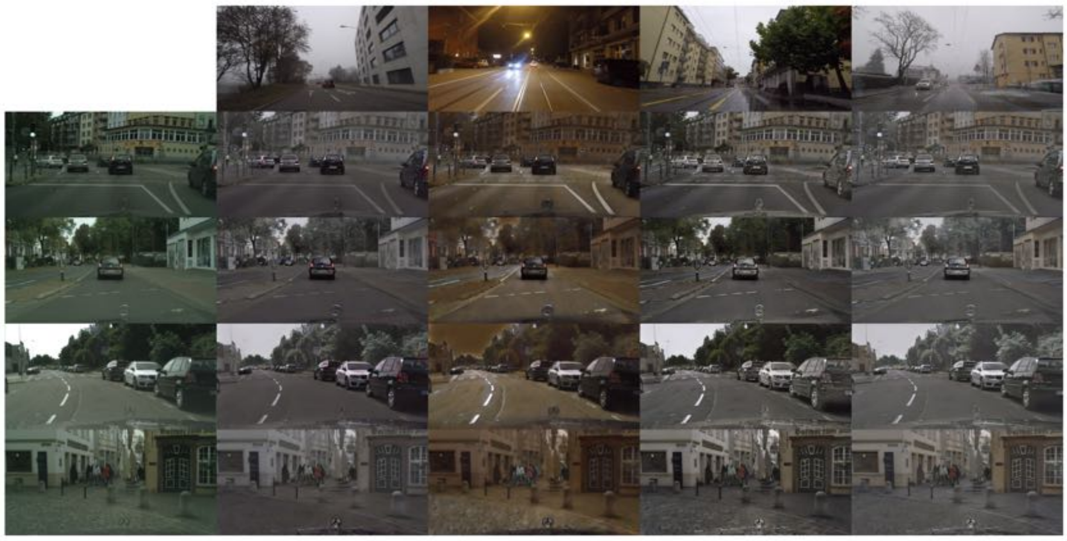


Figure A.9.: Additional stylization results for CACE. All images were generated with class-conditioning and a reduced number of classes.

List of Figures

List of Tables

Bibliography

[1] M. Siam, M. Gamal, M. Abdel-Razek, S. Yogamani, M. Jagersand and H. Zhang, "A comparative study of real-time semantic segmentation for autonomous driving," in Proceedings of the IEEE conference on computer vision and pattern recognition workshops, 2018, pp. 587–597.

[2] Z. Chen, Y. Duan, W. Wang, J. He, T. Lu, J. Dai and Y. Qiao, "Vision Transformer Adapter for Dense Predictions," arXiv preprint arXiv:2205.08534, 2022.

[3] J. Long, E. Shelhamer and T. Darrell, "Fully convolutional networks for semantic segmentation," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 3431–3440.

[4] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected crfs," arXiv preprint arXiv:1412.7062, 2014.

[5] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," IEEE transactions on pattern analysis and machine intelligence, vol. 40, no. 4, pp. 834–848, 2017.

[6] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 3213–3223.

[7] A. Farahani, S. Voghoei, K. Rasheed and H. R. Arabnia, "A brief review of domain adaptation," Advances in data science and information engineering, pp. 877–894, 2021.

[8] Y.-H. Tsai, W.-C. Hung, S. Schulter, K. Sohn, M.-H. Yang and M. Chandraker, "Learning to adapt structured output space for semantic segmentation," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 7472–7481.

[9] Z. Wu, X. Wang, J. E. Gonzalez, T. Goldstein and L. S. Davis, "Ace: Adapting to changing environments for semantic segmentation," in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 2121–2130.

[10] R. A. Marsden, F. Wiewel, M. Döbler, Y. Yang and B. Yang, "Continual Unsupervised Domain Adaptation for Semantic Segmentation using a Class-Specific Transfer," Tech. Rep., 2022.

[11] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," in Psychology of learning and motivation. Elsevier, 1989, vol. 24, pp. 109–165.

[12] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho and A. Grabska-Barwinska, “Overcoming catastrophic forgetting in neural networks,” Proceedings of the national academy of sciences, vol. 114, no. 13, pp. 3521–3526, 2017.

[13] J. Kim, S.-M. Yoo, G.-M. Park and J.-H. Kim, “Continual Unsupervised Domain Adaptation for Semantic Segmentation,” arXiv preprint arXiv:2010.09236, 2020.

[14] T. L. Hayes, K. Kafle, R. Shrestha, M. Acharya and C. Kanan, “Remind your neural network to prevent catastrophic forgetting,” in European Conference on Computer Vision. Springer, 2020, pp. 466–483.

[15] A. Saporta, A. Douillard, T.-H. Vu, P. Pérez and M. Cord, “Multi-Head Distillation for Continual Unsupervised Domain Adaptation in Semantic Segmentation,” arXiv preprint arXiv:2204.11667, 2022.

[16] M. Toldo, A. Maracani, U. Michieli and P. Zanuttigh, “Unsupervised domain adaptation in semantic segmentation: a review,” Technologies, vol. 8, no. 2, p. 35, 2020.

[17] J. Yoo, Y. Uh, S. Chun, B. Kang and J.-W. Ha, “Photorealistic style transfer via wavelet transforms,” in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 9036–9045.

[18] N. Kalischek, J. D. Wegner and K. Schindler, “In the light of feature distributions: moment matching for Neural Style Transfer,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 9382–9391.

[19] X. Liu, Z. Deng and Y. Yang, “Recent progress in semantic image segmentation,” Artificial Intelligence Review, vol. 52, no. 2, pp. 1089–1106, 2019.

[20] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” arXiv preprint arXiv:1409.1556, 2014.

[21] K. He, X. Zhang, S. Ren and J. Sun, “Deep residual learning for image recognition,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[22] J. Huang, L. Guixiong and B. He, “Fast semantic segmentation method for machine vision inspection based on a fewer-parameters atrous convolution neural network,” PloS one, vol. 16, no. 2, p. e0246093, 2021.

[23] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, “Generative adversarial nets,” Advances in neural information processing systems, vol. 27, 2014.

[24] J. Yoon, E. Yang, J. Lee and S. J. Hwang, “Lifelong learning with dynamically expandable networks,” arXiv preprint arXiv:1708.01547, 2017.

[25] Y. Jing, Y. Yang, Z. Feng, J. Ye, Y. Yu and M. Song, “Neural style transfer: A review,” IEEE transactions on visualization and computer graphics, vol. 26, no. 11, pp. 3365–3385, 2019.

[26] L. A. Gatys, A. S. Ecker and M. Bethge, “Image style transfer using convolutional neural networks,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 2414–2423.

[27] A. Khan, A. Sohail, U. Zahoora and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," Artificial intelligence review, vol. 53, no. 8, pp. 5455–5516, 2020.

[28] Y. Li, N. Wang, J. Liu and X. Hou, "Demystifying neural style transfer," arXiv preprint arXiv:1701.01036, 2017.

[29] A. Gretton, D. Sejdinovic, H. Strathmann, S. Balakrishnan, M. Pontil, K. Fukumizu and B. K. Sriperumbudur, "Optimal kernel choice for large-scale two-sample tests," Advances in neural information processing systems, vol. 25, 2012.

[30] X. Huang and S. Belongie, "Arbitrary style transfer in real-time with adaptive instance normalization," in Proceedings of the IEEE international conference on computer vision, 2017, pp. 1501–1510.

[31] Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu and M.-H. Yang, "Universal style transfer via feature transforms," Advances in neural information processing systems, vol. 30, 2017.

[32] D. Ulyanov, V. Lebedev, A. Vedaldi and V. S. Lempitsky, "Texture networks: Feed-forward synthesis of textures and stylized images." in ICML, vol. 1, no. 2, 2016, p. 4.

[33] C. Li and M. Wand, "Precomputed real-time texture synthesis with markovian generative adversarial networks," in European conference on computer vision. Springer, 2016, pp. 702–716.

[34] Y. Zhang, M. Li, R. Li, K. Jia and L. Zhang, "Exact feature distribution matching for arbitrary style transfer and domain generalization," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 8035–8045.

[35] M. Hossain, "Whitening and coloring transforms for multivariate gaussian random variables," Project Rhea, vol. 3, 2016.

[36] F. Luan, S. Paris, E. Shechtman and K. Bala, "Deep photo style transfer," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 4990–4998.

[37] E. Risser, P. Wilmot and C. Barnes, "Stable and controllable neural texture synthesis and style transfer using histogram losses," arXiv preprint arXiv:1701.08893, 2017.

[38] W. Zellinger, B. A. Moser, T. Grubinger, E. Lughofer, T. Natschläger and S. Saminger-Platz, "Robust unsupervised domain adaptation for neural networks via moment alignment," Information Sciences, vol. 483, pp. 174–191, 2019.

[39] A. Müller, "Integral probability metrics and their generating classes of functions," Advances in Applied Probability, vol. 29, no. 2, pp. 429–443, 1997.

[40] W. Zellinger, T. Grubinger, E. Lughofer, T. Natschläger and S. Saminger-Platz, "Central moment discrepancy (cmd) for domain-invariant representation learning," arXiv preprint arXiv:1702.08811, 2017.

[41] Y. Li, M.-Y. Liu, X. Li, M.-H. Yang and J. Kautz, "A closed-form solution to photorealistic image stylization," in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 453–468.

[42] I. Daubechies, Ten lectures on wavelets. SIAM, 1992.

[43] J. C. Ye, Y. Han and E. Cha, "Deep convolutional framelets: A general deep learning framework for inverse problems," SIAM Journal on Imaging Sciences, vol. 11, no. 2, pp. 991–1048, 2018.

[44] R. J. Duffin and A. C. Schaeffer, "A class of nonharmonic Fourier series," Transactions of the American Mathematical Society, vol. 72, no. 2, pp. 341–366, 1952.

[45] T. F. Chan and J. Shen, Image processing and analysis: variational, PDE, wavelet, and stochastic metho SIAM, 2005.

[46] R. Yin, T. Gao, Y. M. Lu and I. Daubechies, "A tale of two bases: Local-nonlocal regularization on image patches with convolution framelets," SIAM Journal on Imaging Sciences, vol. 10, no. 2, pp. 711–750, 2017.

[47] T.-Y. Chiu and D. Gurari, "PhotoWCT2: Compact Autoencoder for Photorealistic Style Transfer Resulting from Blockwise Training and Skip Connections of High-Frequency Residuals," in Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2022, pp. 2868–2877.

[48] T.-H. Vu, H. Jain, M. Bucher, M. Cord and P. Pérez, "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.

[49] Z. Li and D. Hoiem, "Learning without forgetting," IEEE transactions on pattern analysis and machine intelligence, vol. 40, no. 12, pp. 2935–2947, 2017.

[50] C. Sakaridis, D. Dai and L. Van Gool, "ACDC: The adverse conditions dataset with correspondences for semantic driving scene understanding," in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 10 765–10 775.

[51] G. Ros, L. Sellart, J. Materzynska, D. Vazquez and A. M. Lopez, "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.

[52] S. R. Richter, V. Vineet, S. Roth and V. Koltun, "Playing for data: Ground truth from computer games," in European conference on computer vision. Springer, 2016, pp. 102–118.

[53] G. Varma, A. Subramanian, A. Namboodiri, M. Chandraker and C. V. Jawahar, "IDD: A dataset for exploring problems of autonomous navigation in unconstrained environments," in 2019 IEEE Winter Conference on Applications of Computer Vision (WACV). IEEE, 2019, pp. 1743–1751.

[54] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in 2009 IEEE conference on computer vision and pattern recognition. Ieee, 2009, pp. 248–255.

[55] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár and C. L. Zitnick, "Microsoft coco: Common objects in context," in European conference on computer vision. Springer, 2014, pp. 740–755.

Declaration

Herewith, I declare that I have developed and written the enclosed thesis entirely by myself and that I have not used sources or means except those declared.

This thesis has not been submitted to any other authority to achieve an academic grading and has not been published elsewhere.

Stuttgart, August 21, 2022 Tim Lindenau