Domain Adaptation on Semantic Segmentation with Separate Affine Transformation in Batch Normalization

by

Junhao Yan

Thesis submitted to the University of Ottawa
In partial fulfillment of the requirements
For the M.A.Sc. degree in
Electrical and Computer Engineering

Department of Electrical and Computer Engineering
Faculty of Engineering
University of Ottawa

Abstract

Domain adaptation on semantic segmentation generally refers to the procedures for narrowing the distribution gap between source and target data, which is vital for developing the automatic vehicle system. It requires a large amount of data with well-labelled ground truth at the pixel level. Labelling this scale of data is extremely costly due to the lot of human effort required. Also, manually labelling often comes with label noises that are harmful to automatic vehicle system development. In this case, solving the above problem utilizes computer-generated data and ground truth for development. However, a notorious problem exists when a system is trained with synthetic data but deployed in a real-world environment, which results from the distribution (domain) difference between these two kinds of data, and domain adaptation helps solve this issue.

In the thesis, the limitation of conventional batch normalization layer on adversarial learning based domain adaptation methods is mentioned and discussed. From the view of the limitation, we propose replacing the Sharing Affine Transformation with our proposed Separate Affine Transformation (SEAT) to improve the domain adapting performance. The proposed SEAT is simple, easily implemented, and integrated into existing adversarial learning-based unsupervised domain adaptation methods. Also, to further improve the adaptation quality on lower-level features, we introduce multi-level adaptation by adding the lower-level features to the higher-level ones before feeding them to the discriminator, which is different from others by adding extra discriminators. Finally, a simple training strategy, self-training, is adopted to improve the model performance further.

Extensive experiments show that our proposed method is able to get comparable results with other domain adaptation methods with simpler design.

Acknowledgements

First of all, I would like to express my sincere gratitude and deep appreciation to my supervisor, Dr. WonSook Lee. This work could not have been done without her support.

Also, I would like to thank my family and my beloved one. Every time I felt painful and wanted to give up, they kept pushing me to go on and gave me so much courage.

Last but not least, I am also very thankful for all my friends in Ottawa. I cannot go through this without their accompany especially when the Covid-19 pandemic has spread all around the world. People always consider the degree or the knowledge as the biggest gain in the journey of pursuing master degree. For me, friends are my biggest gain. Thank you all!

Dedication

It is dedicated to myself, my family and people who gave me unlimited support in this wonderful journey.

Table of Contents

List of Tables

List of Figures

Abbreviations

AI Artificial Intelligence 1

ASPP Atrous Spatial Pyramid Pooling 22, 25, 44, 46

BN Batch Normalization 4–7, 9, 20, 28, 29, 36

CNN Convolutional Neural Network 6–8, 11, 22, 66, 68

COVID-19 Coronavirus Disease 2019 1

CRF Conditional Random Fields 13

DA Domain Adaptation 4, 5, 15, 16, 18

DL Deep Learning 1, 7, 69

DNN Deep Neural Network 9

GAN Generative Adversarial Network 4, 6, 7, 10

GT Ground Truth 27, 28, 33, 35

ICS Internal Covariate Shift 9

NN Neural Network 6–8, 63–68

RF Receptive Field 22

SEAT Separate Affine Transformation xii, 4–6, 25, 28, 29, 31, 38, 39, 41, 42

SGD Stochastic Gradient Descent 35, 69

UDA Unsupervised Domain Adaptation 6, 7, 18, 19, 35, 36, 38

Chapter 1

Introduction

Artificial Intelligence (AI) was firstly mentioned by Alan Turing in a report in 1969 [72]. In recent years, AI has been widely adopted in many areas of human life and helps improve our quality of life. Generally speaking, we could find AI everywhere, and it becomes inseparable from our life. In the education area, AI tutor allows students to get extra and one-on-one help. There is an idiom in Chinese, teaching students following their aptitude. With the help of AI tutor, students can learn content that suits them best. Robots have become more popular in the manufacturing area as they can work in environments considered dangerous to humans. In the healthcare area, AI help the doctor detect cancers and find the corresponding proper treatment. During the Coronavirus Disease 2019 (COVID-19) pandemic period, it is also used to detect and help stop the spread of the virus, which reduces the pressure on the medical system for many countries. In the transportation area, AI has been utilized in the creation of self-driving vehicles. With AI involved, vehicles enable braking, accelerating, and lane changing according to road situations. Also, in the media area, the AI based recommendation system will recommend content based on our regular use in most music and news platforms.

Deep Learning (DL) must also been mentioned as it is the main direction of AI development in recent years. DL is a subset of machine learning algorithms that progressively extract higher-level features from the raw input by the usage of the neural networks. It has improved state-of-the-art results in many fields, like object detection and recognition,

speech recognition, etc [40].

It is well known that training a sizeable neural network requires a large amount of well-labelled data to prevent the model from overfitting to a particular set of data [23]. Nowadays, most data are manually collected in real world and labeled them by human, which comes with the following limitations. First, uncommon scenes are hard to collect in real life. For example, accident scenes very useful for the development of autonomous driving but hard to collect in real life. Second, collecting and manually annotating a large amount of data with dense pixel-level labels has been extremely costly due to the much human effort required. Third, manual annotation always comes with label noise, like wrong labelling, multi-labelling, etc, where some examples are shown in Figure 1.1. According to the study, there is an average of 3.4% label errors at testing set across ten commonly used datasets [54].

1.1 Motivation

Utilizing synthetic data generated and labelled by the computer is an appealing idea to solve the abovementioned problems. There are several advantages of doing so.

- Without the manual annotation, we could save a lot of human resources like money, time.
- As the synthetic data is generated and labelled by the computer, label noises are perfectly avoided, and we could generate as much data as needed.
- Uncommon scenes that are hard to collect in real life can also be generated by the computer, like crime scenes, accident scenes, etc.

However, there is an not negligible problem existing in using the synthetic data, which is that the distribution between source, with synthetic data, and target, with real-world data, domains are different [27], resulting in a model trained with source domain data performing poorly in target domain data, so-called "domain shift" problem.

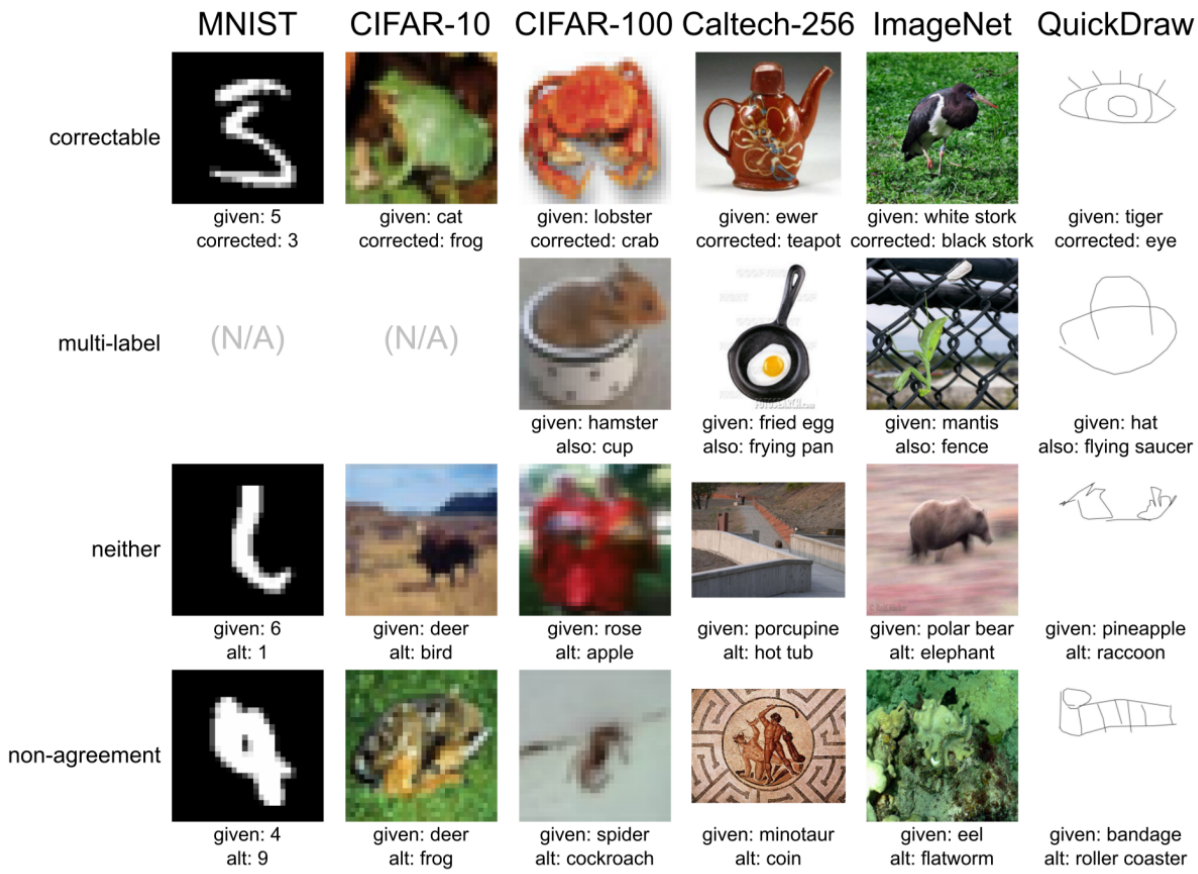


Figure 1.1: Examples of different label noises in 6 commonly used datasets. This figure is collected from [54].

To address the above issue, the Domain Adaptation (DA) was firstly mentioned by Ben-David et al. [5] to reduce the "domain shift" problem. In general, DA can be classified into two categories which are the non-adversarial learning based and adversarial learning based DA while most of adversarial learning based DA methods are built upon on Generative Adversarial Network (GAN). In the thesis, we point out the limitation of the usage of Batch Normalization (BN) in adversarial learning based DA methods, and propose the corresponding solution. More specific, the source domain and target domain have different distributions, researchers still share BN layers [28] between them. It is inappropriate to normalize source domain data with target domain statistics as they have different distributions. Also, during training, the model is trained with source domain data and target domain data alternately with different loss functions. Thus, it causes conflicts at the update of the affine transformation pair in BN layers. More details will be discussed at section 3.3.

Moreover, as mentioned in [71], lower-level features [1] in the neural network are not adapted well. Researchers tend to solve it by directly adding one more discriminator to adapt lower-level features, resulting in more computational resources and more time spent training the network.

All in all, the thesis objective is to improve the performance of adversarial learning based DA methods by solving the problems mentioned above.

We propose to utilize domain separate mean and variance to normalize input data, followed by SEAT to project the normalized source or target domain data to their original representations, instead of using sharing BN layers for source domain data and target domain data. Furthermore, to better adapt lower-level features without adding an extra discriminator, we propose combining the lower-level features with higher-level ones.

[1] Lower-level features represent the output far away from the network. Higher-level features represent the output at the tail of the network. Generally speaking, higher-level features contains more global information than lower-level features because larger area of the image is seen when the network goes deeper.

1.2 System Overview

Figure 1.2 shows the overview of our system, including a segmentation network (**G**) which is in charge of predicting pixel level segmentation, and a discriminator (**D**) which is used to discriminate source and target domain inputs.

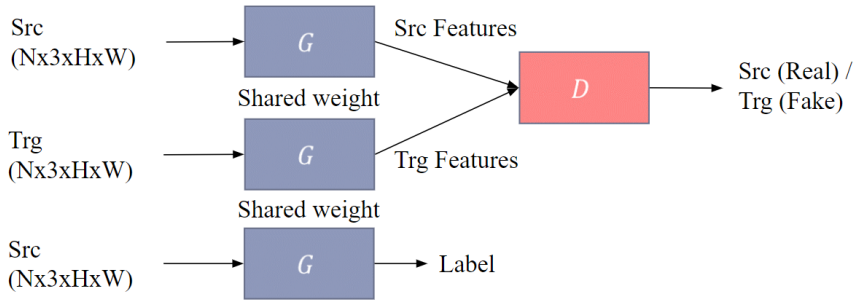


Figure 1.2: **G**, **D** represent segmentation network and discriminator respectively. Three **G** are shared weight, with source (**Src**) and target (**Trg**) domain images as input.

During training, the source domain images and target domain images are fed to the **G** alternatively to output source features and target features. After that, those features are sent to the **D** to distinguish whether the features are from the source domain or target domain. The **G** are trained to fool the **D**, such that it cannot distinguish target features from source features. While the goal of **D** is not to be fooled by the **G**. Two networks are trained alternatively. Besides, when the input of the **G** belongs to source domain, the **G** should also give correct label which is the semantic segmentation map in the thesis.

1.3 Contributions

In the thesis, we indicate the limitation of conventional BN in the adversarial learning based DA methods, and replace it with our proposed SEAT. The SEAT is easy to couple with other DA methods, and faster the convergence speed. Also, we add lower-level features with higher-level features as the final output, which is more effective and computational resource saving comparing with using multiple discriminators.

1.4 Thesis Organization

The structure of the thesis is organized as follows. In Chapter 2, Neural Network (NN), Convolutional Neural Network (CNN) and BN are presented at the very begining, followed by the introduction of GAN and Unsupervised Domain Adaptation (UDA) on semantic segmentation. After that, the model architecture is introduced. Chapter 3 presents the details of SEAT and multi-level adaptation, and how to combine them into the baseline network are also discussed in the chapter, followed by the introduction of loss functions we used to train the network. Next, we illustrate the source, target datasets used in this thesis, and training details at the chapter 4. Moreover, case studies of the SEAT and multi-level adaptation are presented in the chapter. Finally, in chapter 5, the conclusion and future work are presented.

Chapter 2

Literature Review

In this chapter, we first give a brief introduction of NN and CNN. BN is also introduced as our work is based on it. After that, GAN is presented, followed by the introduction of UDA on Semantic Segmentation.

2.1 NN & CNN

NN can be seen as the key component of DL. It's inspired by modeling biological neural systems in human brain [37]. A NN can be seen as a collection of neurons (the Figure 2.1 shows what a neuron look like), where these neurons form a computational graph. In other works, the output of some neurons can become the input of other neurons. The Appendices A.1 presents more details of NN, including the forward pass, backward pass, etc.

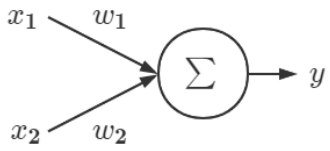


Figure 2.1: A neuron with two inputs x_1, x_2 respectively, where $y = \sum_{i=0}^{n} x_i * w_i$

.

However, utilizing an NN on the image processing tasks is difficult, as the number of trainable parameters increases drastically with an increase in the size of the image. For example, 196,608 (3*128*128*4) trainable parameters are needed for an NN with four neurons when the input image size is 3x128x128. If we enlarge the size of the input image twice, then the number of trainable parameters needs to be four times than the original. Thus, using an NN on the images processing tasks is costly and inefficient, especially for large-scale images.

CNN was specially designed to deal with the handwritten recognition in 1998 [41], while it has been utilized and become popular in other image processing tasks nowadays, like classification, segmentation, etc. Figure 2.2 shows an very famous and classical CNN, named VGG16 [66] on image classification task.

Compared with NN, CNN applies the sharing kernel to extract the relevant features from the input using the convolution operation, which enables it to be less sensitive to the size of the input and to capture the spatial features from an image. Also, NN may make a different prediction on the same object even though only the position of the object has changed. While CNN solves this problem. It retains the features of the image in a similar visual way. When the object is flipped, rotated or changed position, it can also effectively identify the object.

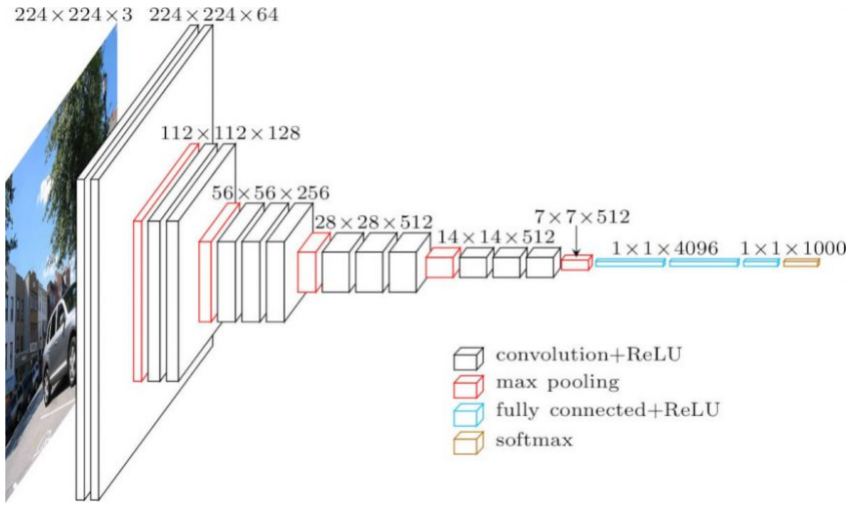


Figure 2.2: The architecture of VGG16, which is collected from the internet.

2.2 Batch Normalization

Training a Deep Neural Network (DNN) is complicated because of the continuous change of the distribution of each layer [28]. Changing the distribution of previous layers requires the current layer to re-adapt the input distribution. A small change to the network parameter amplifies as the network goes deep. This phenomenon is named as Internal Covariate Shift (ICS), leading to slow convergence of a DNN.

So as to tackle the aforementioned problem, BN was proposed by normalizing the input of each layer to zero mean, unit variance and then reprojecting to the original representation by a pair of learnable affine transformation parameters (see algorithm 2.1).

Algorithm 2.1 Batch Normalization Over A Mini-Batch

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$

Input: γ, β: A pair of learnable parameters

Output: y_i

Calculate mini-batch mean: $u_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$

Calculate mini-batch variance: $\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - u_{\mathcal{B}})^2$

Normalize: $\hat{x}_i \leftarrow \frac{x_i - u_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$

Scale and shift: $y_i \leftarrow \gamma\hat{x}_i + \beta$

The advantages of using BN layers in the networks can be listed in follows:

- The network becomes more stable during training. Thus, we could use a larger learning rate and speed up the training.
- Less sensitive to the weight initialization.
- BN adds some noise to the network. It could be regarded as the regularization to some extent.

The Figure 2.3 shows the training accuracy and loss of VGG [66] with and without BN layers. As we can see, the VGG with BN layers converges faster compared with the standard one.

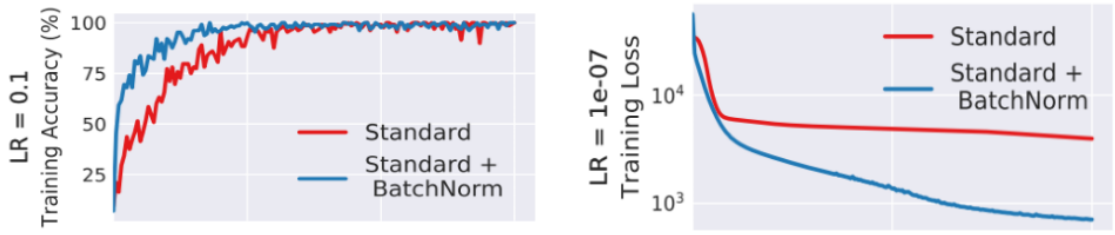


Figure 2.3: Testing accuracy and loss of a network with and without batch normalization layer. the Figure is captured from [65]

2.3 Generative Adversarial Network

GAN [24] was proposed by Goodfellow in 2014. GAN is a min-max game by two networks, named generator **G** and discriminator **D** respectively. Given a noise vector as input, **G** is trained to capture the data distribution and give an output with distribution as close as the given data. **D** is trained to discriminate whether the sample data is drawn from the training data distribution or the generated data distribution. These two networks are trained simultaneously with the equation 2.1.

$$\min_{\mathbf{G}} \max_{\mathbf{D}} \mathbf{V}(\mathbf{D},\mathbf{G}) = \mathbf{E}_{x\sim p_{data}(x)} log(\mathbf{D}(x)) + \mathbf{E}_{x\sim p_{gen}(x)} log(1-\mathbf{D}(\mathbf{G}(x))) \tag{2.1}$$

Until now, there are a lot of impressive improvements being proposed based on the original idea from Goodfellow. For example, GAN is always used in the creation of images, including generating cartoon characters [30], human faces [32], realistic scenes and objects, etc. Compared with the conventional GAN, conditional GAN [51] introduces the label as extra input for the generator and the discriminator so that the generator can give a conditional output. Based on the conditional GAN, pixel2pixel [29, 87] tackles image-to-image translation problems by learning the mapping from input images to output images, including labels to the street scene, aerial to map, day to night, etc. As for text to image, GAN-based methods [16,60,83] demonstrate the usage of GAN to generate realistic images from textual descriptions. There are also GAN aiming to change features on the human face, like IcGAN [57] being able to reconstruct hair colour, style, facial expression, and even

gender for the human face. FaceAging [1] with the ability to change appearance at different ages, from younger to older. Moreover, GAN is also well implemented in super-resolution, where the generator is required to output a high-resolution image given a corresponding low-resolution image. [8, 42, 73], and photo in painting, where the generator, is required to reconstruct the whole image, given an input image with hole [56, 82]. The GAN systems introduced above are mainly focused on the generation in 2-dimensional objects, while 3D-GAN [77], PrGANs [18] can generate 3-dimensional objects by mapping the probability from 2D to 3D. Examples of different GANs are shown at the Figure 2.4.

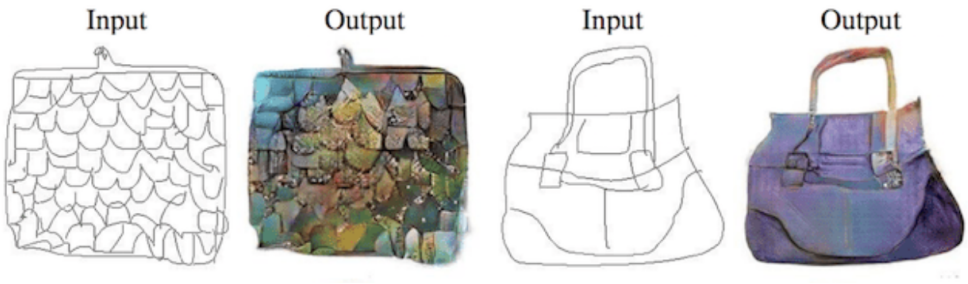
2.4 Unsupervised Domain Adaptation on Semantic Segmentation

In this section, semantic segmentation related literature are introduced at the very beginning, followed by the literature review relevant to the unsupervised domain adaptation on semantic segmentation .

2.4.1 Semantic Segmentation

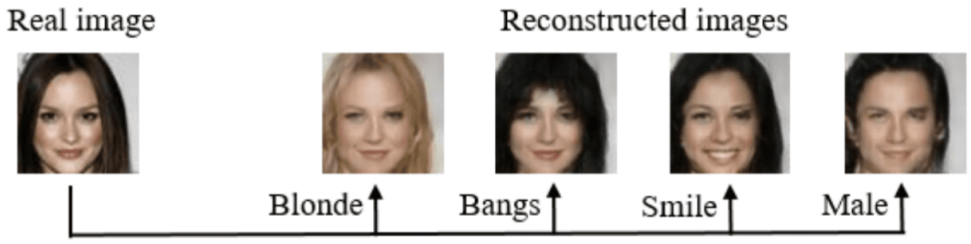
In the last decade, image segmentation has been one of the most challenging tasks in computer vision. Unlike classification and recognition, image segmentation is a pixel-level task required to segment the unknown given image. As the sub-category of image segmentation, semantic segmentation aims at assigning the same labels to those pixels that are semantically relevant and distinct labels while they are not (see Figure 2.5). Compared with traditional image segmentation methods [33, 53], CNN based methods are most used nowadays because of their robustness and capacity of handling complex scenarios. There are many applications about semantic segmentation in our life, like autonomous driving, virtual make-up, virtual try-on, etc. Due to its difficulties and needs in our lives, semantic segmentation is still one of the hottest topics in deep learning.

In 2015, Jonathan Long et al. [47] proposed to adapt three famous networks in classi-

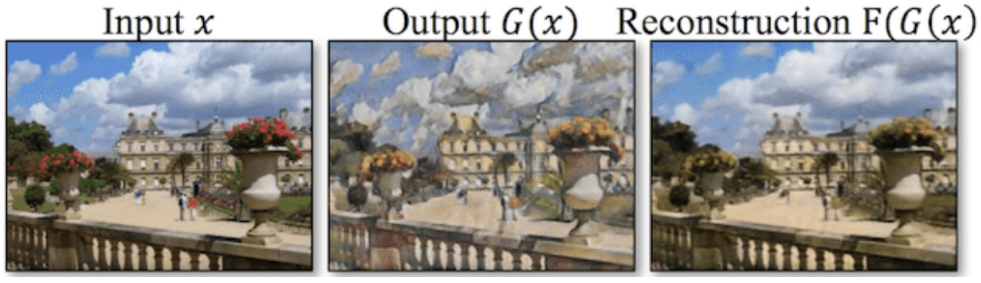


(a) Pixel2pixel [29].

(b) StackGAN [83].



(c) IcGAN [57].



(d) CycleGAN [87].

Figure 2.4: Examples of different GANs.

(a) Original Image (b) Semantic Segmentation Map

Figure 2.5: A sample of semantic segmentation. (a) is the original image, while (b) is the corresponding semantic segmentation map, while the semantic relevant objects are marked with same color.

fication task, which are AlexNet [39], GoogleNet [68], VGG [66] respectively, to semantic segmentation task. Although the performance surpasses the traditional image segmentation methods, there is still much room for improvement. For instance, Jonathan Long et al. [47] directly upsamples the low-resolution output of the network to the desired size, which is unsuitable to the pixel level task [20]. Based on this, SegNet [4] proposes to use an encoder-decoder structure, which is still very common nowadays for pixel-level tasks, to solve the issue as mentioned earlier where the encoder takes charge of producing feature maps. The decoder is responsible for mapping them to pixel-wise predictions. However, it is crucial to incorporate local and global information to achieve good pixel-level accuracy. Thus, Conditional Random Fields (CRF), which can capture the long-range dependencies of pixels, is used as a post-processing step by the DeepLabV1 and V2 [10, 11] to refine the segmentation result. To increase the receptive field without adding more convolutional layers, DeepLabV2 to V3+ [11–13] make use of the dilated convolution to replace the conventional one. Using dilated convolution enables us to reach the same receptive field size with fewer computation resources than the conventional one. Adopting multi scales input to the network is also a good choice to improve the segmentation result [7, 17, 59, 64] as the network could capture different scale objects with different scale input images (small scale images are beneficial to the prediction on the road, tree, etc, while large scale images make for predicting pedestrians, traffic lights, etc.). Last but not least, feature fusion uses the mechanism of combining local features from previous layers with global features

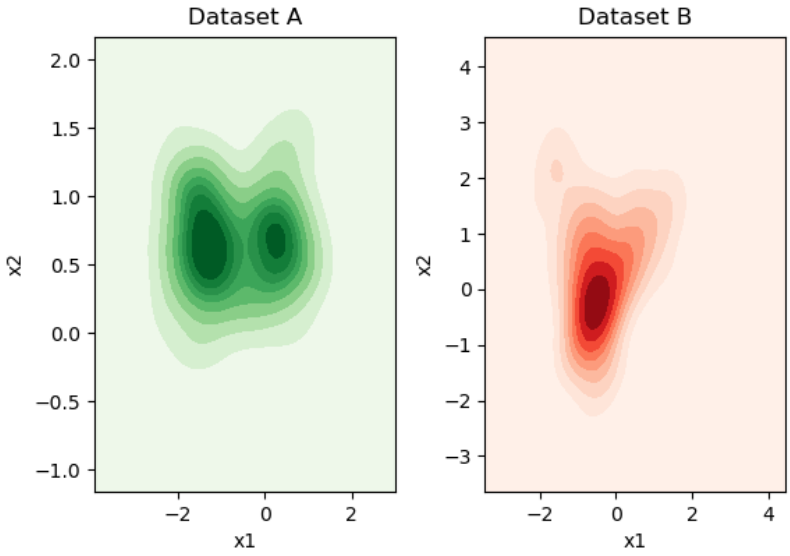


Figure 2.6: Distribution of the dataset A and the dataset B, where both of them are point dataset in two dimensions. The X-axis and the Y-axis are the range of feature x_1 and x_2 respectively. Darker areas indicate that more points are distributed there.

from subsequent layers. It is frequently used by many famous backbones like ResNet [25], UNet [62], HrNet [75], on semantic segmentation.

2.4.2 Unsupervised Domain Adaptation

Machine learning has an assumption that both the training data and the testing data are drawn from the same distribution. But in real world applications, this assumption is easily violated since the training data and the testing data can originate from different distributions. Thus, when a model trained with the data where its distribution cannot reflect the testing data's distribution, the trained model will experience degradation in performance.

We did a small experiment as followed. We manually generated dataset A with specific mean and standard deviation. And then, based on the dataset A, we added a linear transformation to generate dataset B (the distribution of the dataset A and dataset B are shown at the Figure 2.6), where the dataset A and B can be seem as the source and

Train/Test	A (%)	B (%)
A	98.0	76.0
Train/Test	$\hat{A}$ (%)	B (%)
$\hat{A}$	98.0	84.0
Train/Test	A (%)	$\hat{B}$ (%)
A	98.0	84.0

Table 2.1: Classification accuracy on the test set that has the same and different distribution as the train set.

target domain dataset respectively. After that, we trained a linear classifier for binary classification task using the train set of A, and tested it with the test set of B (see Table 2.1 top row). We could observe there is a large degradation on the performance when classifier tested on a test set with different distribution . We also visualize the decision boundary so as to understand how the classifier perform under a disparity domain dataset, which is shown at the Figure 2.7. As shown, the decision boundary is not well suited on the dataset B. In order to decrease the degradation resulting from the domain gap (domain disparity), DA was proposed and aims to solve this type of problem by aligning the discrepancy of domain distributions.

Definitions & Goals

Let $\mathbf{X_s} = \{x_i^s\}_{i=0}^{\mathcal{N}^s}$ be the source domain data, with labels $\mathbf{Y} = \{y_i^s\}_{i=0}^{\mathcal{N}^s}$, and $\mathbf{X_t} = \{x_i^t\}_{i=0}^{\mathcal{N}^t}$ be the target domain data, where $\mathcal{N}^s$, $\mathcal{N}^t$ represent the number of source domain data and target domain data. Assuming the existence of two distinct joint probability distributions $\mathbf{P_S}(x^s, y)$ and $\mathbf{P_T}(x^t, y)$ are related to the source domain and target domain respectively, noted as Ω_s and Ω_t. DA is to find an non-linear transformation $\mathbf{T}$ to transform source domain distribution to target domain distribution ($\mathbf{T}$: $\Omega_s \rightarrow \Omega_t$), or conversely, target domain distribution to source domain distribution ($\mathbf{T}$: $\Omega_t \rightarrow \Omega_s$). And the transformed

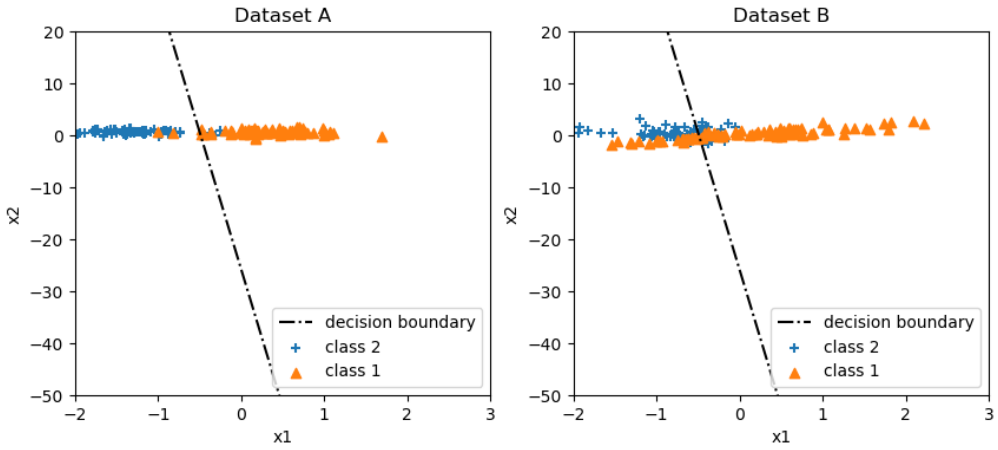


Figure 2.7: Visualization of the dataset A and B with 2 classes. The black dot-dash line is the decision boundary of the classifier which is trained on the dataset A and applied on the dataset B.

source data should satisfy the following equation:

$$\mathbf{P_s}(y|x_s) = \mathbf{P_t}(y|\mathbf{T}(x_s)), \mathbf{T} : \Omega_s \rightarrow \Omega_t \tag{2.2}$$

$$\mathbf{P_t}(y|x_t) = \mathbf{P_s}(y|\mathbf{T}(x_t)), \mathbf{T} : \Omega_t \rightarrow \Omega_s \tag{2.3}$$

The above equation means that the label information is still preserved after transformation. For example, if we rotate the dataset A in a certain degree ($\hat{A} = \mathbf{T}A$) to map the distribution of the dataset B. We train the classifier with the $\hat{A}$ and make prediction on the dataset B. The accuracy increases to 84.0% from 76.0% (see Table 2.1 middle row). The Figure 2.8 show the new decision boundary. The Figure 2.9 also show the transformed dataset B, and the performance is shown at the Tabel 2.1 bottom row. and In practice, the distribution between two domains are more complex than what we showed at the example. Thus, it is not simple to find a proper transformation to decrease the domain gap. However, as more data being available in public, deep learning approaches have started to dominate this area.

According to the standard proposed by the Sicheng Zhao et al [86], the deep domain adaptation can be summarized into categories shown at the table 2.2. As our DA method is under $\mathcal{N}_{\mathcal{TL}} = 0$, $\mathcal{N}_{\mathcal{SL}} = \mathcal{N}_{\mathcal{S}}$, $\mathcal{N}_{\mathcal{T}} = 1$, $\mathcal{N}_{\mathcal{S}} = 1$, the literature review of DA method is also focused within this range.

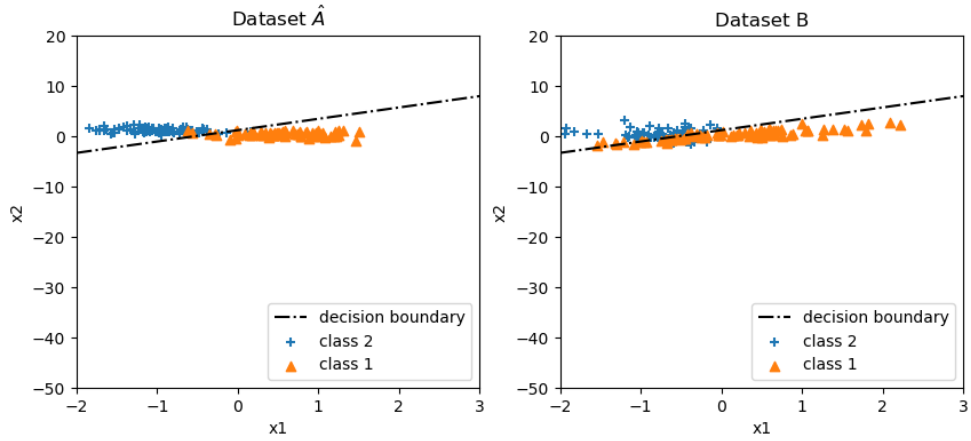


Figure 2.8: Visualization of the dataset $\hat{A}$ and B ($\mathbf{T}$: $\Omega_s \rightarrow \Omega_t$).

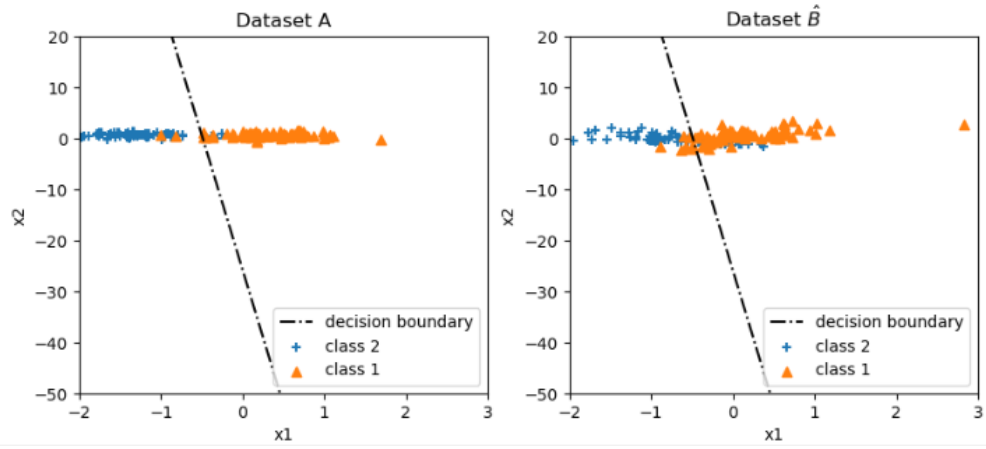


Figure 2.9: Visualization of the dataset A and $\hat{B}$ ($\mathbf{T}$: $\Omega_t \rightarrow \Omega_s$).

Standard	Classification	Definition	Description
target label	unsupervised DA	$\mathcal{N}_{\mathcal{TL}} = 0$	target data is fully unlabeled
	fully supervised DA	$\mathcal{N}_{\mathcal{TL}} = \mathcal{N}_{\mathcal{T}}$	target data is fully labeled
	semi-supervised DA	$0 < \mathcal{N}_{\mathcal{TL}} < \mathcal{N}_{\mathcal{T}}$	target data is partially labeled
source label	strongly-supervised DA	$\mathcal{N}_{\mathcal{SL}} = \mathcal{N}_{\mathcal{S}}$	source data is fully labeled
	weakly-supervised DA	$\mathcal{N}_{\mathcal{SL}} < \mathcal{N}_{\mathcal{T}}$	source data is weakly labeled
#targets	single-target DA	$\mathcal{N}_{\mathcal{T}} = 1$	only one target domain
	multi-target DA	$\mathcal{N}_{\mathcal{T}} > 1$	multiple target domains
#sources	single-source DA	$\mathcal{N}_{\mathcal{S}} = 1$	only one source domain
	multi-source DA	$\mathcal{N}_{\mathcal{S}} > 1$	multiple source domains

Table 2.2: Classifications of Domain Adaptation. $\mathcal{N}_{\mathcal{TL}}$, $\mathcal{N}_{\mathcal{SL}}$, $\mathcal{N}_{\mathcal{T}}$, $\mathcal{N}_{\mathcal{S}}$ denote the number of target label, source label, target data, source data respectively.

Adversarial Learning-Based UDA Methods (With Discriminator Involved)

Adversarial learning is widely used on UDA area [19,26,27,71,76,85]. Basically, adversarial learning based methods consists of two components. One is called the discriminator, which is used to discriminate target domain samples from the source domain samples, and the other one is called the generator (feature extractor), which aims to find a representation that the discriminator cannot distinguish two domains samples. Two networks are trained alternatively until the discriminator can not classify correctly while the generator give correct labels for source domain samples.

At the very beginning, a single discriminator is used at the feature space on classification task [19, 48]. While Yi-Hsuan Tsai et al. [71] proposes to adapt to the structured output space for the semantic segmentation task because the latter task requires more visual cues and details that are needed to adapt. Based on it, more and more researchers have started to make innovations in this area. For example, Judy Hoffma et al. [26] utilize transferred target-style source domain images generated from CycleGan [87] to train the model. Myeongjin Kim et al. [35] diversify the texture of source domain images. Then

the generated images with various textures are used during training to prevent the segmentation model from overfitting to one specific texture. To better align cross-domain features, more constraints are also added during training. For example, Tuan-Hung Vu et al. [74] combines entropy-based loss with adversarial loss to align the domain gap better, while Fei Pan et al. [55], based on the original Adversarial Learning-based UDA methods, proposes to split the target domain images into hard and easy split using an entropy-based ranking function. And then fine-tune the model by adapting the domain from easy split to hard split. To adapt the target domain features towards the most likely space of source domain features, Zhonghao Wang et al. [76] proposes to treat stuff categories (sky, tree, etc) and thing categories (car, traffic sign, etc) with different strategies. Guangrui Li et al. [44] facilitates Content-Consistent Matching to find out synthetic images with similar distribution as real ones in the target domain.

Non-Adversarial Learning-Based UDA Methods (Without Discriminator Involved)

Compared with adversarial training-based methods, non-adversarial training-based methods usually require less computational resources and less effort to balance the segmentation model and the discriminator. Q.Lian et al. [45] combine curriculum domain adaptation [84] with self-training to construct a pyramid curriculum that contains different levels of information about the target domain. G. Kang et al [31], W. Liu [46] proposes to build the pixel-level cycle association and patches-level association respectively between source and target domain data and contrastively strengthen their connections. While Y. Yang et al. [80] describes a Fourier transfer-based unsupervised domain adaptation by swapping the low-frequency spectrum of synthetic images with that of target images.

2.5 Models Architecture

In this section, the architecture of the segmentation network and the discriminator are introduced.

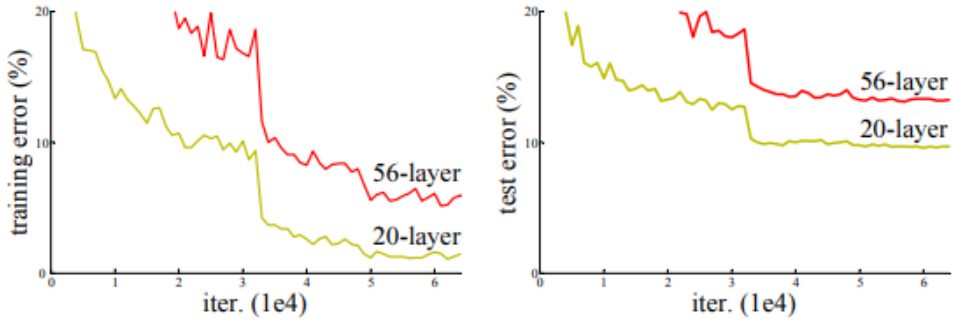


Figure 2.10: Training error (left) and testing error (right) on CIFAR-10 [38]. The figure is captured from [25].

2.5.1 Segmentation Network

The DeepLabV3 is proven to be a very successful network in semantic segmentation task due to the atrous convolutional layer [12]. Thus, DeepLabV3 is combined with the ResNet-101 [25] to form our segmentation network.

ResNet

Deeper networks are always hard to converge because of the notorious problem of vanishing/exploding gradients [6, 21]. This problem is largely addressed by adding the BN [28] layer after each convolution. However, simply using BN is still not enough to let deeper network outperform shallower network. This phenomenon can be found at the figure 2.10. As we can see, both the training error and the test error of a 56-layer network are higher than a 20-layer network.

Based on the hypothesis that multiple nonlinear layers should be capable of approximating the residual functions ($\mathcal{F}(\mathbf{X}) = \mathcal{H}(\mathbf{X}) - \mathbf{X}$) if it could approximate the complex functions ($\mathcal{H}(\mathbf{X})$), Kaiming He et al [25] designs the residual block (see the Figure 2.11) and make a serial of networks in different level of depth based on the special design residual block to solve the problem as mentioned above. Compared with directly learning the complex functions $\mathcal{H}(\mathbf{X})$, learning the residual function $\mathcal{F}(\mathbf{X})$ with the precondition from the identity $\mathbf{X}$ is much easier. the Figure 2.12 show the architecture of ResNet, while ResNet-101 is one of them.

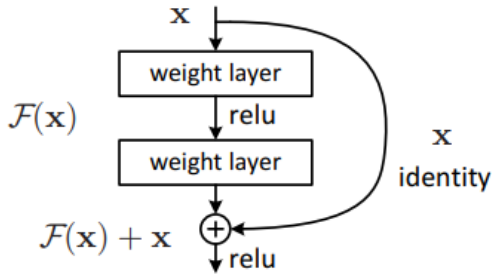


Figure 2.11: Residual block, where $\mathbf{X}$ is the input, $\mathcal{F}$ is the mapping function, which consists of convolutional layers. the Figure is captured from [25].

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 2.12: Architecture of ResNet with differet layers, while the [*] represents one residual block. the Figure is captured from [25].

DeepLabV3

One challenge of using CNN is that the features map becomes smaller while traversing through the convolutional and pooling layers of the network, which results in the loss in information of images and making the object boundries become fuzzy. DeepLabV3 solve this challenge by introducing the Atrous Spatial Pyramid Pooling (ASPP) and the Atrous Convolution (Dilated Convolution). Before going deeper on this two modules, we would like to introduce the Receptive Field (RF) first as it is crucial for the segmentation task.

The receptive field is defined as the size of the region in the input that produces the feature [2]. More specifically, the receptive field is the metric to indicate how large the original image can be seen by a single pixel in the feature map, where the Figure 2.13 gives a more intuitive explanation. In the segmentation task, we need to predict each pixel, which means we want the RF to be large enough to take every crucial information from the input image into account. The equation 2.4 shows how to calculate the RF for a layer, where r, s, k represents the size of RF, stride and kernel respectively.

$$r_{i+1} = s_i \times r_i + (k_i - s_i) \tag{2.4}$$

As we can see, if we want to enlarge the RF, we need to increase either the size of the stride or the size of the kernel size. As known, the larger stride and kernel size lead to losing more information of the input image and more computational cost respectively.

So as to keep large enough RF while using small size of stride and kernel, atrous convolution [11] is proposed to replace the conventional convolution layer (see the Figure 2.14). On top of the features map extracted from the backbone, ASPP, consisting of four parallel atrous convolutions with different atrous rates (1, 6, 12, 18) is applied to get the multi-scale context information. The final model architecture is shown at 2.15.

2.5.2 Discriminator

PatchGAN has achieved great success in the image-to-image translation task [29]. Based on the insight that semantic segmentation and image-to-image translation are pixel-level

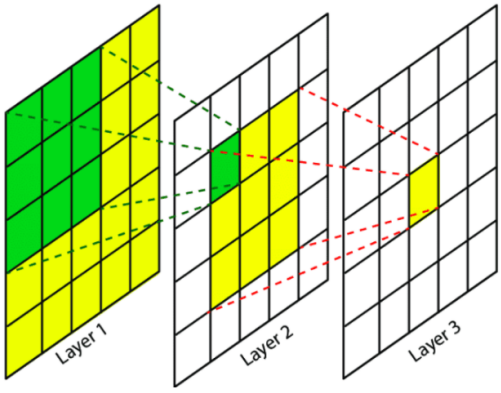


Figure 2.13: The receptive field for green pixel in layer 2 is 3, while the yellow pixel in layer 3 is 5.

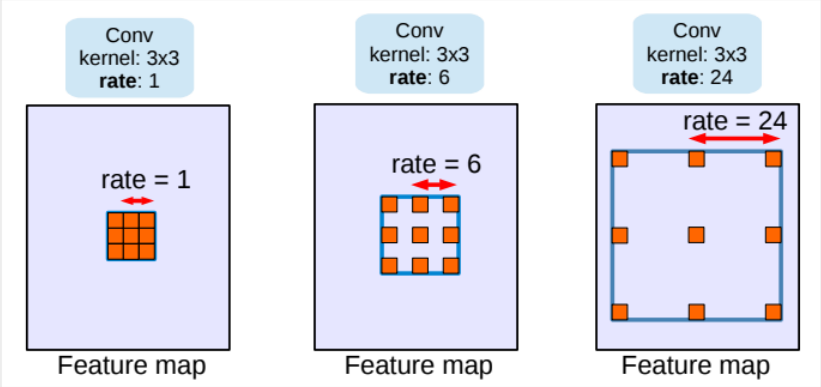


Figure 2.14: Atrous Convolution layer with kernel size of 3x3 and different atrous rates, where rate=1 represents conventional convolution, rate=6,24 represents atrous convolution. Employing large value of atrous rate enlages the receptive field [12].

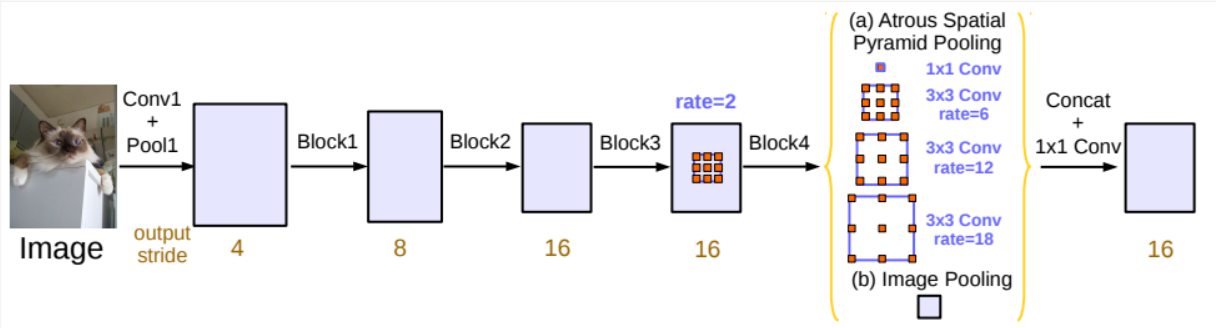
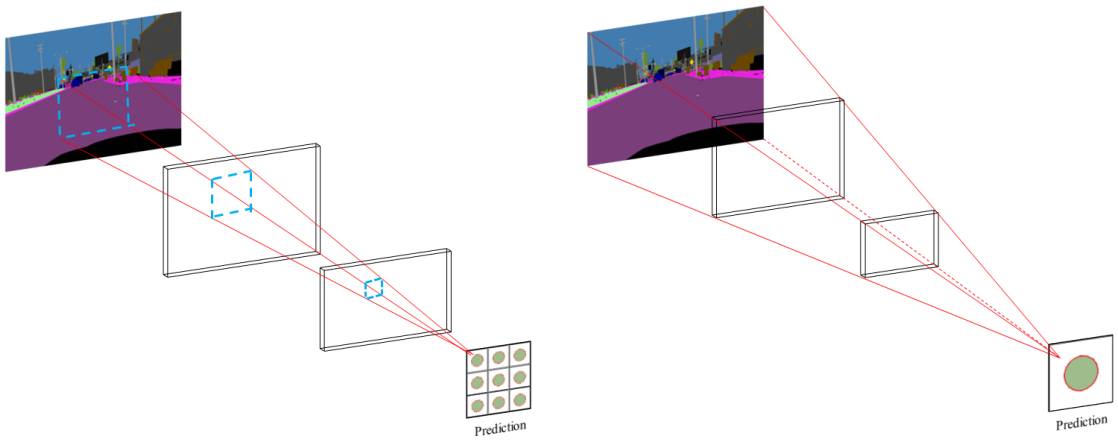


Figure 2.15: Model architecture of DeepLabV3 [12].

tasks, we also choose to use PatchGAN instead of conventional GAN to narrow the domain gap. The only difference between them is that PatchGAN generates an $\mathcal{N} \times \mathcal{N}$ array, where each element in the array is a scaler to classify the corresponding patch as real or fake, instead of directly generating a scaler vector to classify the whole image as real or fake. Figure 2.16 shows how PatchGAN and Conventional GAN look like.



(a) PatchGAN (b) Conventional GAN

Figure 2.16: Comparison of PatchGAN and Concentional GAN. The PatchGAN outputs an array with prediction on each patch, while the Conventional GAN makes prediction on the whole image.

Chapter 3

Method

In this section, we introduce the multi-level adaptation first, followed by objective functions and SEAT. Finally, self-training is also discussed. The overall model architecture of our method is shown at the Figure 3.1, where **G** and **D** represent the segmentation network and the discriminator respectively.

3.1 Multi Level Adaptation

In order to solve poor adaptation quality on lower-level features, Tsai et al [71] propose to add one more discriminator to adapt lower-level features. However, adding one more discriminator requires nearly two times the computational resources during training compared with only one discriminator being used. To improve the adaptation quality while saving more computational resources, we combine the lower-level features with higher-level ones. As shown in the figure 3.1, outputs from layer 3 and layer 4 of the segmentation network are passed through to the ASPP module and then added together with a hyper-parameter α. After that, the combined outputs are sent to the discriminator for discriminating.

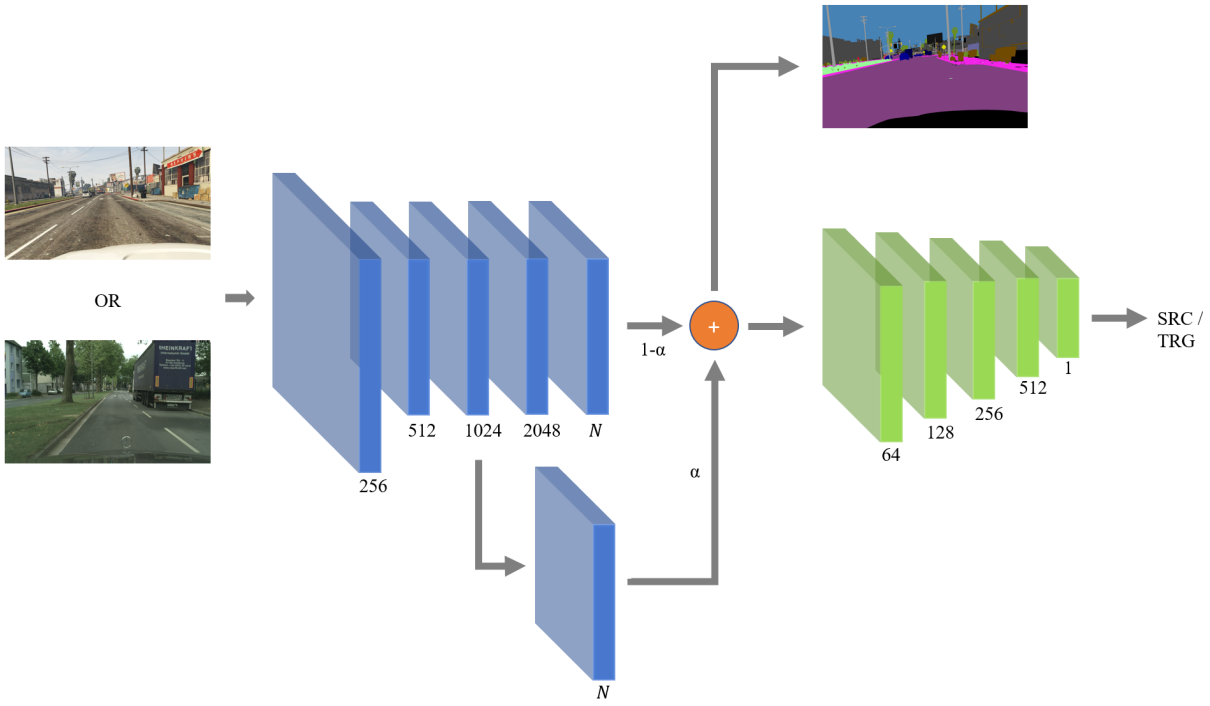


Figure 3.1: The overall model architecture. Blue blocks represent **G** integrating with our proposed SEAT inside the red rectangle. Green blocks represent **D**. Arrows correspond to the flow direction, where orange arrows represent our multi-level adaptation by adding the lower-level features with higher-level ones with parameter α. Numbers represent the size of channels, while $\mathcal{N}$, $\oplus$ denotes the number of classes and the sum operation.

3.2 Objective Functions and Training Schema

The source domain images $x^s \in \mathbb{R}^{3\times\mathcal{H}\times\mathcal{W}}$ and target domain images $x^t \in \mathbb{R}^{3\times\mathcal{H}\times\mathcal{W}}$ are firstly forwarded to the **G** to get lower-level and higher-level features of source domain and target domain respectively, denoted as $\{f^{ls}, f^{hs}, f^{lt}, f^{ht}\} \in \mathbb{R}^{\mathcal{N}\times\mathcal{H}\times\mathcal{W}}$. With these features, the following loss functions are used to train **D** and **G** alternately.

3.2.1 Discriminator Objective Function

Firstly, **D** is trained to discriminate the source domain features and target domain features. The conventional discriminator loss is applied and shown at equation 3.1.

$$\mathcal{L}_{dis} = -\frac{1}{\mathcal{M}} \sum_{i=0}^{\mathcal{M}} \mathcal{L}_{dis}^{src}(\alpha f_i^{ls} + (1-\alpha) f_i^{hs}) + \mathcal{L}_{dis}^{trg}(\alpha f_i^{lt} + (1-\alpha) f_i^{ht}) \tag{3.1}$$

$$\mathcal{L}_{dis}^{src}(x) = (1-y) log(1 - \mathbf{D}(x)) \tag{3.2}$$

$$\mathcal{L}_{dis}^{trg}(x) = y log(\mathbf{D}(x)) \tag{3.3}$$

$y = 1$ if samples are drawn from target domain, $y = 0$ if samples are drawn from source domain. $\mathcal{M}$ represents the number of images in a batch.

3.2.2 Segmentation Network Objective Functions

There are two goals being set for **G**. Firstly, given the target domain input x^t, **G** is required to output features in source style so as to fool **D**. In order to do so, we apply adversarial loss shown at 3.4.

$$\mathcal{L}_{adv} = -\frac{1}{\mathcal{M}} \sum_{i=0}^{\mathcal{M}} log(1 - \mathbf{D}(\alpha f_i^{lt} + (1-\alpha) f_i^{ht})) \tag{3.4}$$

Secondly, the cross-entropy loss is applied to **G** so as to output segmentation map that is similar with the given Ground Truth (GT) labels Y^s, which is shown at the equation 3.5.

$$\mathcal{L}_{seg} = -\frac{1}{\mathcal{M}} \sum_{i=0}^{\mathcal{M}} Y_i^s log(\alpha f_i^{ls} + (1-\alpha) f_i^{hs}) \tag{3.5}$$

The overall loss function of **G** is shown at equation 3.6, where β is set to 0.001 during experiments.

$$\mathcal{L}_{ssn} = \beta \mathcal{L}_{adv} + \mathcal{L}_{seg} \tag{3.6}$$

3.2.3 Training Schema

Too strong or too weak **D** often leads to the collapse of **G** because of the gradient explosion and the gradient vanishing problems. Thus, balancing the **D** and **G** is very important and widely studied by researchers. Among the proposed balancing strategies, training the **D** and **G** alternately is widely used and is shown below.

Algorithm 3.1 Training Schema

Input: Source domain images x^s and GT labels Y^s

Input: Target domain images x^t

Input: Segmentation network **G** and Discriminator **D**

Input: Epoch size E, Batch size B

while epoch $\leq E$ **do**

while $i \leq B$ **do**

Sample source domain images x_i^s, labels Y_i^s and target domain images x_i^t over a batch

$\mathbf{G}(x_i^s) \rightarrow f_i^{ls}, f_i^{hs}$, $\mathbf{G}(x_i^t) \rightarrow f_i^{lt}, f_i^{ht}$

Update **D** by $\mathcal{L}_{dis}(f_i^{ls}, f_i^{hs}, f_i^{lt}, f_i^{ht})$

Update **G** by $\mathcal{L}_{adv}(f_i^{lt}, f_i^{ht})$ and $\mathcal{L}_{seg}(f_i^{ls}, f_i^{hs}, Y_i^s)$

end while

end while

3.3 Separate Affine Transformation

In this section, we will describe why the conventional BN layers degrades the model performance and how BN layers intergrating with SEAT avoids this issue.

First, we want to build up the relation between the Cross-Entropy and the KL-Divergence [15], where KL-Divergence is used to measure the distance between two probability distributions. The definitions for them are shown below, where A and B represent GT's distribution and prediction's distribution here.

$$H(A,B) = -\sum_{x} P_A(x) log(P_B(x)) \tag{3.7}$$

$$KL(A,B) = \sum_{x} P(A)(x) log(P_A(x)) - P_A(x) log(P_B(x)) \tag{3.8}$$

If we substitute equation 3.7 with equation 3.8, we get equation 3.9.

$$H(A,B) = KL(A,B) + H(A,A) \tag{3.9}$$

From equation 3.9, as $H(A,A)$ is a constant. Thus, we could consider that minimizing the Cross-Entropy is equal to minimizing the distance between probability distribution A and B.

Let us go back to the issue brought from the conventional BN. In order to give a straightforward explanation, we draw how affine transformation maps input distribution to output distribution during training at figure 3.2. As we could see from the figure 3.2 (a) and (b), the update of affine transformation pair in conventional BN is conflicting if it is trained simultaneously to align distribution between source's prediction to that of source GT and to align the distribution of target's prediction to that of source's prediction, which leads to negative impact in the domain adaptation.

On the contrary, if we utilize source and target affine transformation (AF_{src} and AF_{trg}) for source domain input and target domain input shown at the figure 3.2 (c), the conflict is avoided. The architecture between conventional BN layers and our proposed one are shown at figure 3.3, and the algorithm 3.2 presents the flow of a BN layer intergrating with SEAT.

3.4 Self Training

After the first stage of training introduced above, we could use the trained model to generate pseudo labels for target domain images given a certain threshold $\psi \in [0,1]$. More

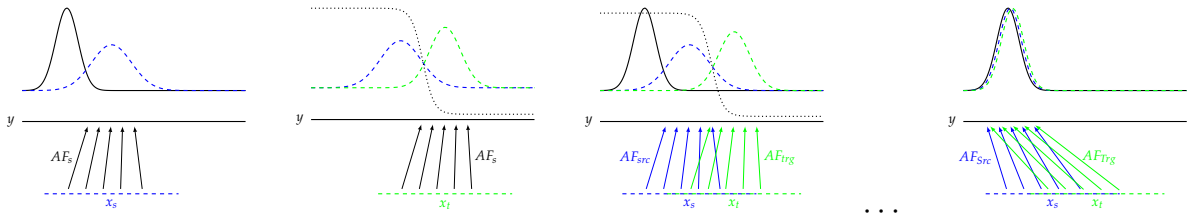


Figure 3.2: The upward arrows represent Affine Transformation inside BN layers, which maps normalized input x to output y ($y = AF(x)$). Conventional BatchNorm is trained simultaneously to align distribution between (a) Source GT (black, solid line) and source's prediction (blue, dashed line) under the guidance of the Cross-Entropy loss (proved at equation 3.9) given source domain input (blue, horizontal, dashed line), and to align the distribution between (b) Source's prediction and target's prediction (green, dashed line) guided by discriminative distribution (black, dotted line) given target domain input (green, horizontal, dashed line). (c) separate affine transformations are applied for source and target domain input. (d) eventually, the source's prediction and target's prediction will align with the GT's distribution in ideal condition when training goes on.

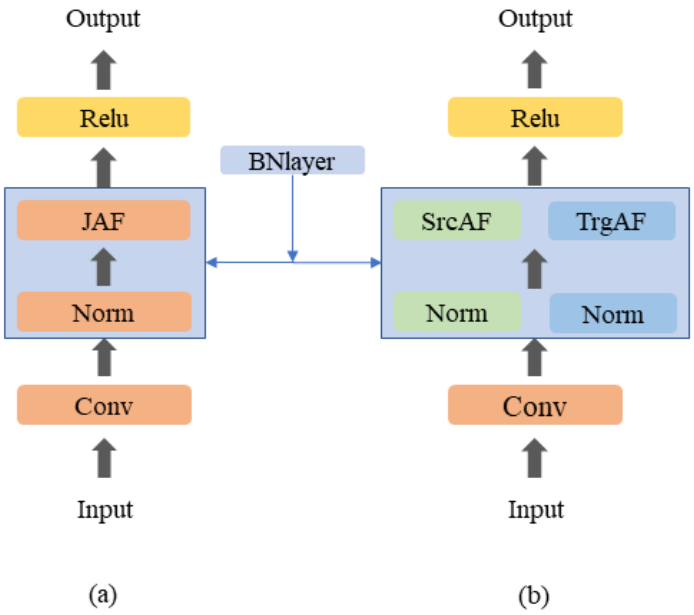


Figure 3.3: Architecture between the conventional BatchNorm and BatchNorm intergrating with SEAT.

Algorithm 3.2 Batch Normalization Intergrating with SEAT Over A Mini-Batch

Input: Values of source input x^s or target input x^t over a mini-batch: $\mathcal{B} = \{1...m\}$

Input: (γ_s, β_s) or (γ_t, β_t): A pair of learnable parameters for source input or target input respectively

Output: y_i^s or y_i^t

if input data belongs to source domain **then**

Calculate mini-batch mean: $u_{\mathcal{B}}^s \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i^s$

Calculate mini-batch variance: $(\sigma_{\mathcal{B}}^s)^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i^s - u_{\mathcal{B}}^s)^2$

Normalize: $\hat{x_i^s} \leftarrow \frac{x_i^s - u_{\mathcal{B}}^s}{\sqrt{(\sigma_{\mathcal{B}}^s)^2 + \epsilon}}$

Scale and shift: $y_i^s \leftarrow \gamma_s \hat{x_i^s} + \beta_s$

end if

if input data belongs to target domain **then**

Calculate mini-batch mean: $u_{\mathcal{B}}^t \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i^t$

Calculate mini-batch variance: $(\sigma_{\mathcal{B}}^t)^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i^t - u_{\mathcal{B}}^t)^2$

Normalize: $\hat{x_i^t} \leftarrow \frac{x_i^t - u_{\mathcal{B}}^t}{\sqrt{(\sigma_{\mathcal{B}}^t)^2 + \epsilon}}$

Scale and shift: $y_i^t \leftarrow \gamma_t \hat{x_i^t} + \beta_t$

end if

specifically, **G** is initialized and trained with equation 3.6 at stage 1 without any target domain's labels involved. After that, for each image in target domain train set, the trained **G** from stage 1 is used to predict semantic segmentation map $\phi \in \mathbb{R}^{\mathcal{N}\times\mathcal{H}\times\mathcal{W}}$. With ϕ, we could generate pseudo label $Y^t \in \mathbb{R}^{\mathcal{H}\times\mathcal{W}}$ by the following equation.

$$Y^t[i,j] = \begin{cases} argmax(\phi[:,i,j]),\ if\ \phi[\kappa,i,j] \geq \psi \\ 255 \end{cases} \tag{3.10}$$

Given pseudo labels Y^t, we could reinitialize and train **G** at stage 2 by equation 3.11.

$$\mathcal{L}_{ssn} = \beta\mathcal{L}_{adv} + \mathcal{L}_{seg} + \mathcal{L}_{st} \tag{3.11}$$
$$\mathcal{L}_{st} = -\frac{1}{\mathcal{M}}\sum_{i=0}^{\mathcal{M}} Y_i^t log(\alpha f_i^{lt} + (1-\alpha) f_i^{ht})$$

Chapter 4

Experiments and Results

In this chapter, we first show the datasets we use to evaluate our proposed method. After that, we present the overall results and the discussion of the ablation study.

4.1 Datasets

To verify the effectiveness of the proposed method, we conduct experiments on three commonly used datasets in this area, which are GTA5 [61], SYNTHIA [63] and Cityscapes [14]. Both of the GTA5 and SYNTHIA are source domain datasets with fully labelled GT data during training and the Cityscapes is the target domain dataset without any GT labels being used during training. The two adaptation scenarios are GTA5 $\rightarrow$ Cityscapes and SYNTHIA $\rightarrow$ Cityscapes. The Figure 4.1 shows the example of three datasets.

4.1.1 GTA5

GTA5 contains pixel-accurate semantic label maps for images collected from a famous computer game named GTA5. It contains 24,966 synthetic images with 33 classes of semantic annotations, with [1914, 1052] original image size. During training, 19 classes in common are used, and input images are resized into [1280, 720] and then randomly cropped

into [1024, 512] for data argumentation. The Table A.1 presents the label distribution of the dataset.

4.1.2 SYNTHIA-RAND-CITYSCAPES (SYNTHIA)

The SYNTHIA-RAND-CITYSCAPES is one of the subsets from the SYNTHIA dataset, which contains 9,400 images, with [1280, 760] original image size. During training, we resize the image into [1280, 720] and randomly crop the images into [1024, 512] for data argumentation. 16 standard classes are used to train for the SYNTHIA dataset, while 13 are used to evaluate for standard comparison. The Table A.2 presents the label distribution of the dataset.

4.1.3 Cityscapes

Cityscapes is a real-world dataset consisting of 30 classes collected from driving scenarios. Only 19 classes are used during evaluation for standard comparison, while 2,975 images are used to train, and 500 images are used for evaluation. In order to keep consistent with synthetic datasets, we resize the input image into [1024, 512] from [2048, 1024].

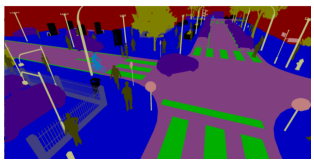
Name	GTA5 [61]	SYNTHIA [63]	Cityscapes [14]
Input			
Label			

Figure 4.1: Example of GTA5, SYNTHIA and Cityscapes datasets.

4.2 Training Details

A single RTX3090 is used to train our model in 150,000 iterations with a batch size of 1, which takes around 40 hours. For the segmentation network, the optimizer we use is Stochastic Gradient Descent (SGD) with an initial learning rate of 2.5e-4, weight decay of 5.0e-4 and momentum of 0.9. As for the discriminator, we use Adam with an initial learning rate of 1e-4 and the betas is set to (0.9, 0.99). The polynomial learning rate scheduler with a power of 0.9 is applied to all models to adjust the learning rate.

4.3 Evaluation Metric

IoU (Intersection over Union) is used to check to what degree the predicted output is similar to the GT. It is calculated by the area of overlap between the predicted output and the GT divided by the area of union between the predicted output and the GT (the Figure 4.2). For multi classes, the average of the IoU of each class is calculated (mIoU). In our experiments, we also calculate the mIoU of large-scale, medium-scale, small-scale objects according to the proportion of pixels occupied by each class in the test set (see more details at the A.3).

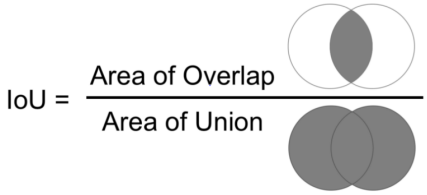


Figure 4.2: The Intersection over Union.

4.4 Overall Results

We present overall adaptation results at Table 4.1 and Table 4.2 comparing with other UDA methods on two adaptation scenarios and the experiments show that our proposed

methods is simpler but comparable with other UDA methods. For example, at Table 4.1, Intra [55] consists of two pairs of segmentation network and discriminator compared with only one pair used by ours. FDA [81] utilizes three segmentation networks (MBT) and entropy regularization compared with only one segmentation network, no entropy regularization used by ours. We also visualize the improvement of our method comparing with the network with the conventional BN at figure 4.3, according to the Table 4.1, we could see that our method outperforms others UDA methods on adapting the medium scale objects, while reaching comparable results on the large and small scale objects on GTA5 to Cityscapes scenario. The same situation also presents for the SYNTHIA to Cityscapes scenario.

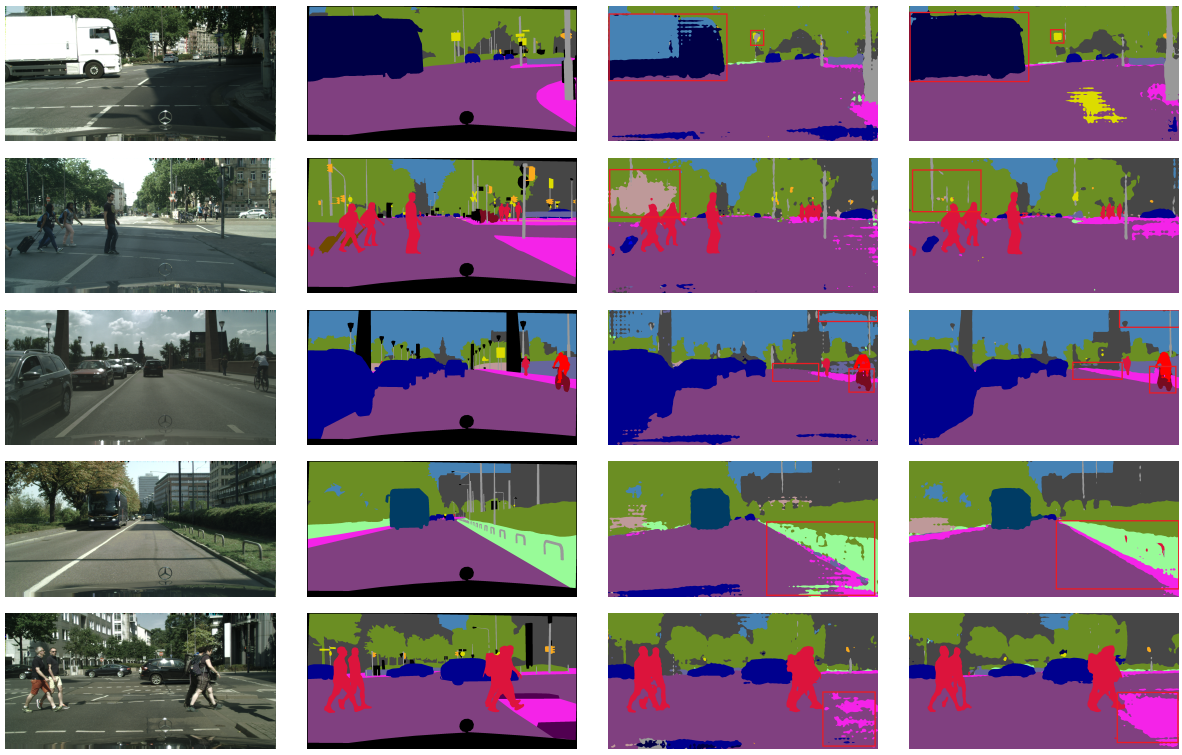


Figure 4.3: Visualization results on GTA5 to Cityscpaes. Black regions shown at the GT column are ignored during training.

GTA5→Cityscapes																						
Methods	*road*	*sidewalk*	*building*	*wall*	*fence*	*pole*	*light*	*sign*	*vegetation*	*terrain*	*sky*	*person*	*rider*	*car*	*truck*	*bus*	*train*	*motocycle*	*bicycle*	*mIoU*	*mIoU_l*	*mIoU_m*	*mIoU_s*
AdaStruct [71]	86.5	25.9	79.8	22.1	20.0	23.6	33.1	21.8	81.8	25.9	75.9	57.3	26.2	76.3	29.8	32.1	**7.2**	29.5	32.5	41.4	70.1	3.4	2.8
DCAN [79]	85.0	**30.8**	81.3	25.8	21.2	22.2	25.4	26.6	83.4	36.7	76.2	58.9	24.9	80.7	29.5	42.9	2.5	26.9	11.6	41.7	72.2	3.7	2.3
DLOW [22]	87.1	33.5	80.5	24.5	13.2	29.8	29.5	26.6	82.6	26.7	81.8	55.9	25.3	78.0	33.5	38.7	0.0	22.9	34.5	42.3	72.3	3.4	2.8
Cycada [26]	86.7	35.6	80.1	19.8	17.5	**38.0**	**39.9**	**41.5**	82.7	27.9	73.6	**64.9**	19.0	65.0	12.0	28.6	4.5	31.1	**42.0**	42.7	70.0	3.1	**3.5**
CLAN [49]	87.0	27.1	79.6	27.3	23.3	28.3	35.5	24.2	83.6	27.4	74.2	58.6	28.0	76.2	33.1	36.7	6.7	31.9	31.4	43.2	70.7	3.6	3.0
ABStruct [9]	**91.5**	**47.5**	82.5	31.3	25.6	33.0	33.7	25.8	82.7	28.8	82.7	62.4	**30.8**	85.2	27.7	34.5	6.4	25.2	24.4	45.4	**77.9**	3.7	2.9
AdvEnt [74]	89.4	33.1	81.0	26.6	**26.8**	27.2	33.5	24.7	83.9	36.7	78.8	58.7	30.5	84.8	38.5	44.5	1.7	31.6	32.4	45.5	74.4	3.9	3.0
Orthogonal [70]	89.4	30.7	82.1	23.0	22.0	29.2	37.6	31.7	83.9	37.9	78.3	60.7	27.4	84.6	37.6	44.7	7.3	26.0	38.9	45.9	74.1	3.9	3.2
Intra [55]	90.1	37.1	82.6	30.1	19.1	29.5	32.4	20.6	**85.7**	**40.5**	79.7	58.7	31.1	**86.3**	31.5	**48.3**	0.0	30.2	35.8	46.3	76.4	3.8	3.0
Ours	91.0	41.8	**85.0**	**36.4**	26.6	31.4	34.3	28.4	84.5	34.8	**84.7**	60.9	30.1	85.5	**43.7**	40.9	0.1	**32.0**	26.5	**47.3**	77.6	**4.1**	3.0

Table 4.1: Overall results with comparison to other unsupervised domain adaptation methods on GTA5 to Cityscapes dataset. 19 classes are used to train and evaluate. mIoU_l, mIoU_m, mIoU_s represent mIoU of the large, medium, small scale objects respectively.

SYNTHIA→Cityscapes																				
Methods	*road*	*sidewalk*	*building*	*wall*	*fence*	*pole*	*light*	*sign*	*vegetation*	*sky*	*person*	*rider*	*car*	*bus*	*motocycle*	*bicycle*	*mIoU*	*mIoU_l*	*mIoU_m*	*mIoU_s*
AdaStruct [71]	84.3	42.7	77.5	-	-	-	4.7	7.0	77.9	82.5	54.3	21.0	72.3	32.2	18.9	32.3	46.7	70.9	5.6	1.7
CLAN [49]	81.3	37.0	**80.1**	-	-	-	16.1	13.7	78.2	81.5	53.4	21.2	73.0	32.9	22.6	30.7	47.8	70.0	5.6	2.1
Advent [74]	85.6	42.2	79.7	-	-	-	5.4	8.1	80.4	84.1	**57.9**	23.8	73.3	36.4	14.2	33.0	48.0	72.2	**6.0**	1.7
Orthogonal [70]	88.3	42.2	79.1	-	-	-	**16.8**	**16.5**	80.0	**84.3**	56.2	15.0	**83.5**	27.2	6.3	30.7	48.2	74.6	5.6	1.7
ABStruct [9]	**91.7**	**53.5**	77.1	-	-	-	6.2	7.6	78.4	81.2	55.8	19.2	82.3	30.3	17.1	34.3	48.8	**76.6**	5.6	1.7
Intra [55]	84.3	37.7	79.5	-	-	-	9.2	8.4	80.0	84.1	57.2	23.0	78.0	38.1	20.3	**36.5**	48.9	71.9	**6.0**	1.9
Ours	81.8	39.1	79.7	-	-	-	10.9	15.8	**80.9**	82.5	54.5	**27.2**	72.8	**42.9**	**28.3**	32.1	**49.9**	70.9	**6.0**	**2.3**

Table 4.2: Overall results with comparison to other unsupervised domain adaptation methods on SYNTHIA to Cityscapes. 16 classes are used to train, and 13 classes are used to evaluate. mIoU_l, mIoU_m, mIoU_s represent mIoU of the large, medium, small scale objects respectively.

4.5 Ablation Study

4.5.1 Layer Switch

Target Norm and Affine Transformation are used only in normal situations as no source data are used during evaluation. However, we wonder how switching target Norm, Affine Transformation with source ones in certain layers affects the model's performance. Thus, we do the following experiments. First, we divide the ResNet-101 into 4 layers according to where downsampling happens. Second, we switch the target Norm and Affine Transformation to source one layer by layer and then evaluate the model performance. Due to limited computational resources, we only evaluate the switch starting from layer 4 to layer 1, where the experiment result is shown in figure 4.4. What is beyond our expectation is that switching some certain layers even improves the model's performance, while there is around 2 mIoU improvement at most on GTA5 to Cityscapes scenario. From the Figure 4.4, we could see that the mIoU increases when we do layer switch from layer4-4 or layer3-4. We consider that it is because layer 4 and layer 3 is the layer we select to adapt, which results in better adaptation quality than layer 1 and layer 2.

4.5.2 Effectiveness

In order to show the effectiveness and generally applicable of SEAT, we combine SEAT with other three adversarial learning based UDA methods, which are AdaStruct [71] with single level adaptation, Advent (adv model) [74] , Differential [76] respectively, and then evaluate on GTA5 to Cityscapes adaptation scenario. The results are shown at Table 4.3.

AdaStruct [71] with single level adaptation gains largest improvement, which has 0.6 mIoU increase after SEAT being added, and 1.9 mIoU increase if we also do layer switch. In contrast, Advent [74] obtains less improvement, increase from 43.8 to 44.0 mIoU. After the SEAT is added, the performance of Differential [76] decreases to 44.1 from 45.5 mIoU but gains 0.4 mIoU improvement if we switch the certain layer.

Also, we find that the network with SEAT converge faster than that of without SEAT.

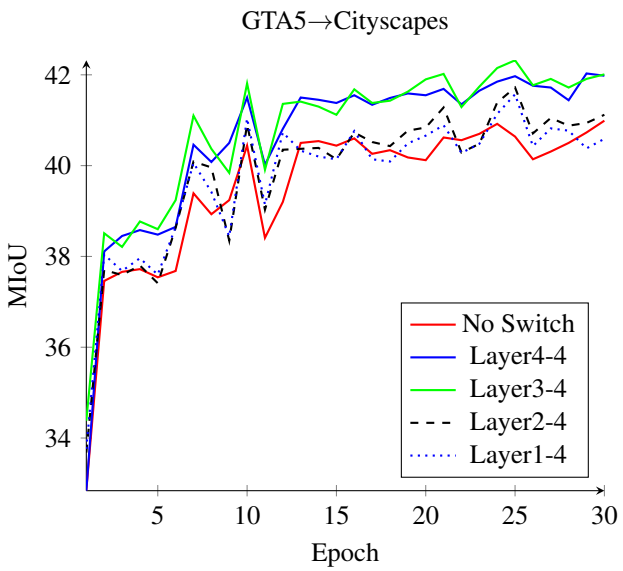


Figure 4.4: Evaluation curve of layer switch on GTA5 → Cityscapes. Layer(x)-(y) represent switching form layer(x) to layer(y). For example, Layer3-4 represent switching from layer3 to layer4.

The detail is shown at the Figure 4.5.

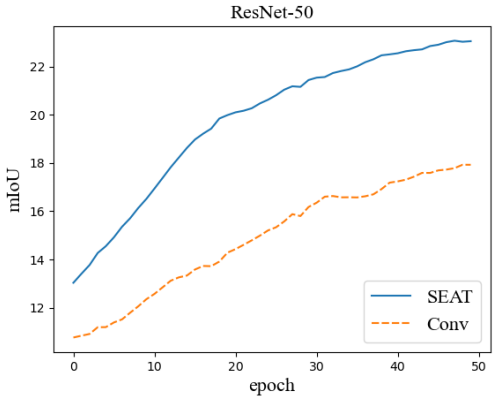
4.5.3 Models with SEAT

First of all, we choose ResNet-50 and ResNet-101, as the backbone, and then combine them with SEAT and multi-level adaptation. The experiments are done on two adaptation scenarios and the results are shown at the Table 4.4 and the Table 4.5. When using ResNet-50 as the backbone, the mIoU increases to 26.4 from 21.3 for GTA5 → Cityscapes scenario. As for SYNTHIA → Cityscapes, the model gains 10.2 mIoU improvement. When using ResNet-101 as the backbone, the mIoU increases to 42.9 from 40.4 for GTA5 → Cityscapes scenario. As for SYNTHIA → Cityscapes, the model gains 3.1 mIoU improvement.

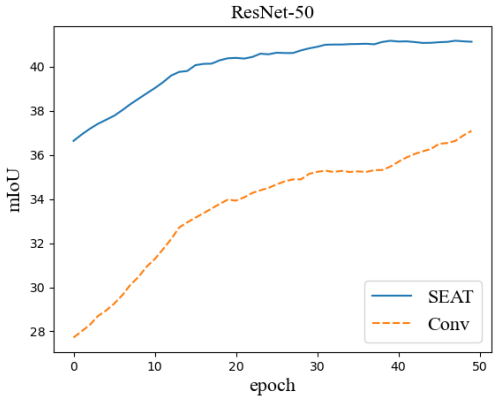
Table 4.6 and Table 4.7 show the contribution of each module to the overall model performance for two adaptation scenarios with ResNet-101 as backbone.
For GTA5 to Cityscapes scenario, we first follow the work from [71] with single level adaptation at the output space and achieve mIoU of 40.4, which is 1.0 lower than the

GTA5→Cityscapes			
Methods	mIoU	LS	Range
AdsStruct (single-level) [71]*	40.4	-	0
+SEAT	41.0	-	+0.6
+SEAT	**42.3**	✓	+1.9
Advent [74]	43.8	-	0
+SEAT	43.9	-	+0.1
+SEAT	**44.0**	✓	+0.2
Differential [76]+	45.5	-	0
+SEAT−	44.1	-	-1.4
+SEAT	**45.9**	✓	+0.4

Table 4.3: The performance of other adversarial based unsupervised domain adaptation methods when combined with the SEAT. LS stands for layer switch. *: reproduced result, 41.4 at the paper; +: reproduced result, 46.2 at the paper. −: we apply different random state during training, the result ranges from 44.1 to 45.8.



(a) ResNet-50 as backbone



(b) ResNet-101 as backbone

Figure 4.5: Evaluation results on GTA5 → Cityscapes scenario with two backbone. The blue line represent the network with SEAT, while the orange dash line represent the network without SEAT. Both ResNet-50 and ResNet-101 with SEAT converges faster than that of conventional version.

GTA5→Cityscapes					
Backbone	AA	SEAT	MUL	mIoU	Range
ResNet-50	✓			21.3	+0.0
	✓	✓		24.7	+3.4
	✓		✓	22.0	+0.7
	✓	✓	✓	26.4	+5.1
ResNet-101	✓			40.4	+0.0
	✓	✓		41.0	+0.6
	✓		✓	40.8	+0.4
	✓	✓	✓	42.9	+2.5

Table 4.4: Ablation study on modules' contributions on GTA5 → Cityscapes with different backbone. AA represents Adversarial Adaptation; SEAT represents separate affine transformation; MUL represents multi level adaptation;

SYNTHIA→Cityscapes					
Backbone	AA	SEAT	MUL	mIoU	Range
ResNet-50	✓			16.2	+0.0
	✓	✓		25.2	+9.0
	✓		✓	17.1	+0.9
	✓	✓	✓	26.4	+10.2
ResNet-101	✓			42.8	+0.0
	✓	✓		45.1	+2.3
	✓		✓	43.0	+0.2
	✓	✓	✓	45.9	+3.1

Table 4.5: Ablation study on modules' contributions on SYNTHIA → Cityscapes with different backbone. AA represents Adversarial Adaptation; SEAT represents separate affine transformation; MUL represents multi level adaptation;

paper shows. As mentioned at [76], transferred target-style source images are helpful to minimize the discrepancy of data distribution. Thus, we also adapt the transferred GTA5 images of [26] to improve the mIoU to 43.4, which is served as the baseline for our works. Then, the mIoU is improved to 45.5 by adding the proposed SEAT (layer switch 3-4). After that, our proposed multi-level adaptation is combined, with α being set to 0.05, and the mIoU achieves 46.4 (layer switch 3-4). Finally, mIoU is improved to 47.3 with self-training (layer switch 3-4).

As for SYNTHIA to Cityscapes scenario, SEAT is added. The mIoU is increased to 47.0 (no layer switch) compared with the mIoU of 44.6 from the baseline. After that, the mIoU is improved to 47.4 with our proposed multi-level adaptation ($\alpha = 0.15$, layer switch 4-4). Finally, the mIoU is improved to 48.9 when combined with self-training (layer switch from 4-4).

GTA5→Cityscapes						
AA	IT	SEAT	LS	MUL	ST	mIoU
✓						40.4
✓	✓					43.3
✓	✓	✓	✓			45.5
✓	✓	✓	✓	✓		46.4
✓	✓	✓	✓	✓	✓	47.3

Table 4.6: Ablation study on modules' contributions. AA represents Adversarial Adaptation; LS represents layer switch; IT stands for image style transferring; MUL represents multi level adaptation; ST represents self training.

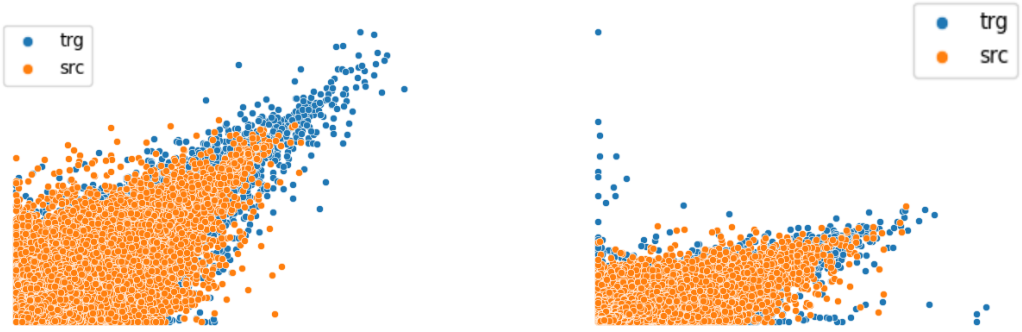
SYNTHIA→Cityscapes						
AA	IT	SEAT	LS	MUL	ST	mIoU
✓						42.8
✓	✓					44.6
✓	✓	✓	✓			47.0
✓	✓	✓	✓	✓		47.4
✓	✓	✓	✓	✓	✓	49.9

Table 4.7: Ablation study on modules' contributions. AA represents Adversarial Adaptation; LS represents layer switch; IT stands for image style transferring; MUL represents multi level adaptation; ST represents self training.

4.5.4 Parameter Analysis

In order to give a more straightforward proof that our proposed multi-level adaptation can improve the adaptation quality at the lower layers on two adaptation scenarios, we visualize the distribution of the output feature from the lower layer (see the Figure 4.6 and 4.7). The Figures show that the source domain features and target domain features are better aligned after the multi-level adaptation is used.

Also, to find out the best value of α, we keep all network settings the same except for the change in the value of α. The Table 4.8 and table 4.9 show the experiment results on two adaptation scenarios. We could see that our proposed multi-level adaptation ($\alpha \neq 0$) is beneficial to domain alignment compared with single-level adaptation ($\alpha = 0$) on GTA5 $\rightarrow$ Cityscapes scenario, while $\alpha = 0.05$ is the most suitable one among those. However, for the SYNTHIA $\rightarrow$ Cityscapes scenario, we need to choose a proper value for α. The model can gain improvement from the multi-level adaptation.



(a) w/o multi level adaptation (b) w multi level adaptaion

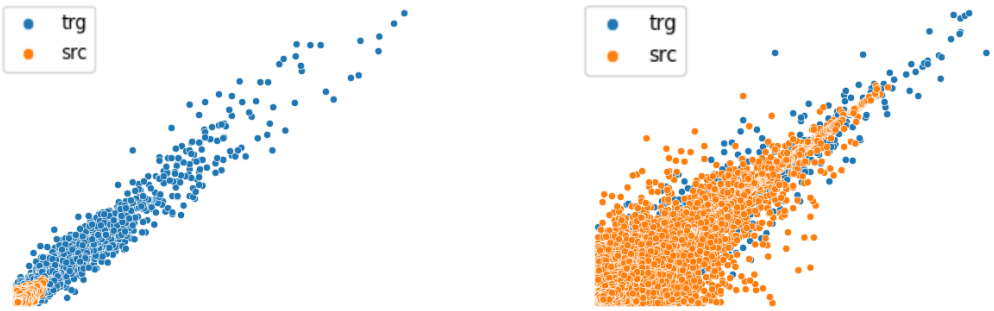
Figure 4.6: Lower layer's features visualization on GTA5 $\rightarrow$ Cityscapes scenario, where orange, blue cycle represent source, target features respectively.

4.5.5 Visualization of ASPP

As mentioned at 2.5.1, the ASPP consists of multiple paralleled atrous convolution layers with different atrous rates to capture the multi-scale context information. In our experi-

GTA5→Cityscapes						
α	0	0.01	0.05	0.1	0.2	0.4
mIoU	41.0	41.9	**42.9**	42.5	42.6	41.7
Range	0	+0.9	+1.9	+1.5	+1.6	+0.7

Table 4.8: α Analysis on GTA5 → Cityscapes (AA+SEAT+MUL).



(a) w/o multi level adaptation (b) w multi level adaptaion

Figure 4.7: Lower layer's features visualization on SYNTHIA → Cityscapes scenario, where orange, blue cycle represent source, target features respectively.

SYNTHIA→Cityscapes									
α	0	0.1	0.15	0.2	0.3	0.4	0.5	0.6	0.7
mIoU	45.1	45.7	45.7	45.9	45.8	46.4	45.9	45.3	45.6
Range	0	+0.6	+0.6	+0.8	+0.7	**+1.3**	+0.8	+0.2	+0.5

Table 4.9: α Analysis on SYNTHIA → Cityscapes (AA+SEAT+MUL), where α is the weight for controlling the contribution of lower-level features and higher-level features to the final output, which is used in the Equation 3.1.

ments, four paralleled atrous convolution layers with atrous rates of 6, 12, 18, 24 are used. In order to study to what extent the atrous rates affect features being captured by the atrous convolution layer, we visualize the mentioned four paralleled atrous convolution layers from the higher-level ASPP on GTA5 → Cityscapes scenario, where the input image is shown at the Figure 4.8 and the output from atrous convolution layers with atrous rates of 6, 12, 18, 24 are shown at the Figure 4.9, 4.10, 4.11, 4.12 respectively in heat map format (whiter area represents higher activation).

We could find that the atrous convolution layer with a smaller astrous rate is better at capturing the small scale objects while performing worse on large scale objects than that of a large astrous rate. For example, there are clear boundaries on car and sidewalk class at the output of the atrous convolution layer with atrous rate of 6. But, the boundaries become blur when the atrous rate becomes larger. On the contrary, the atrous convolution layer with atrous rate of 24 performs better on capturing the sky, road class than that of atrous rate of 6.

Figure 4.8: Input image, GTA5 → Cityscapes

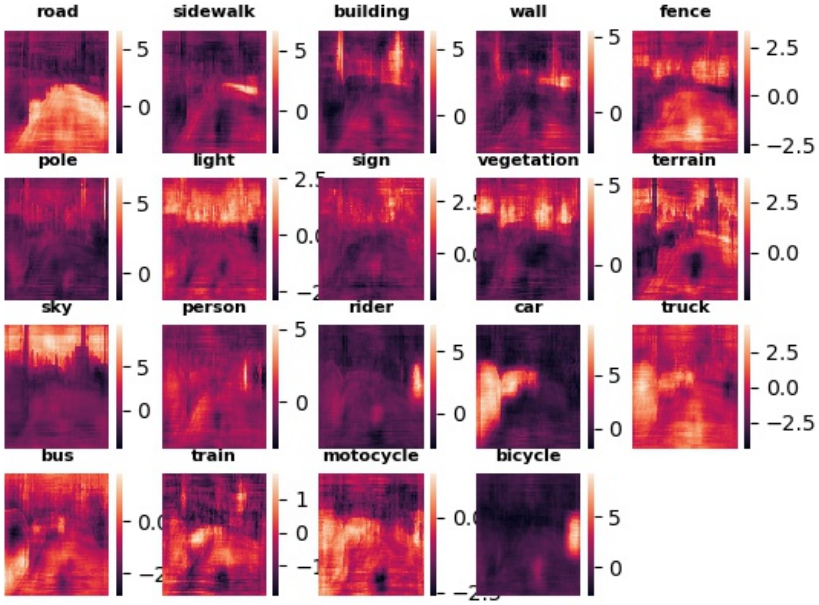


Figure 4.9: Astrous Rates=6, GTA5 → Cityscapes

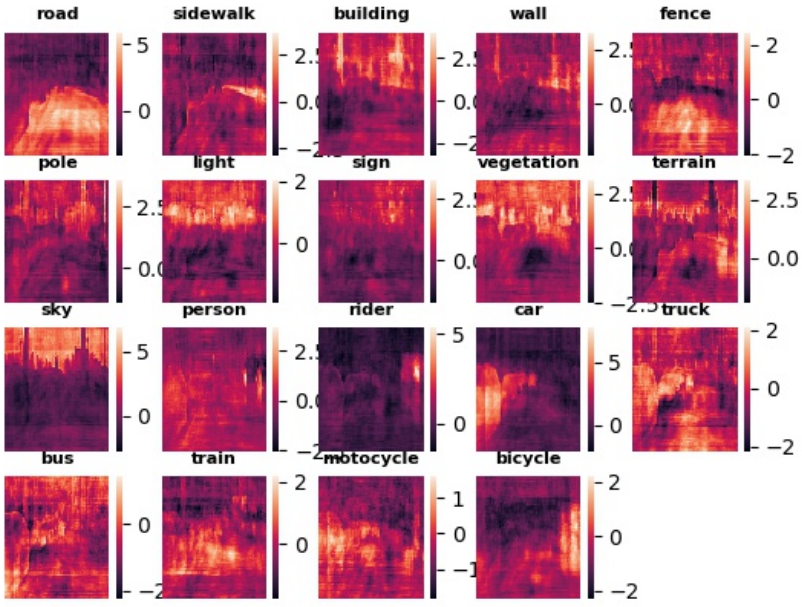


Figure 4.10: Astrous Rates=12, GTA5 → Cityscapes

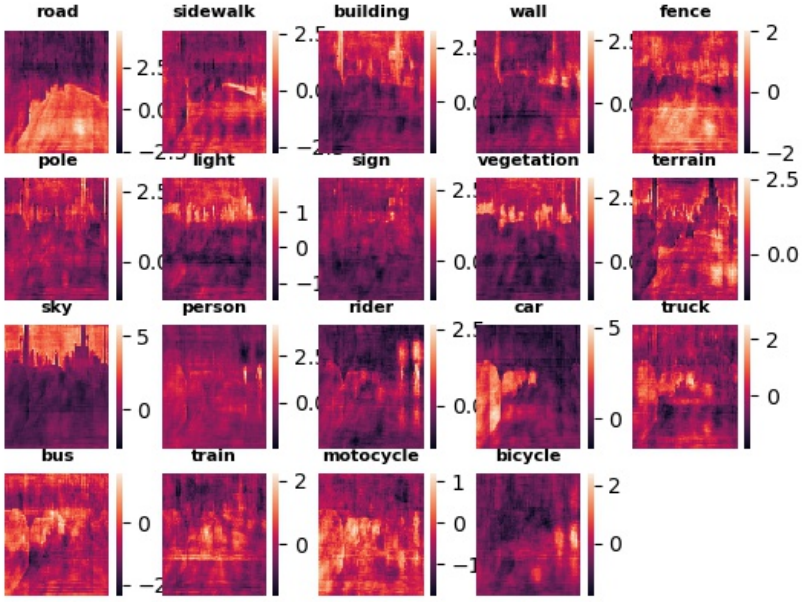


Figure 4.11: Astrous Rates=18, GTA5 → Cityscapes

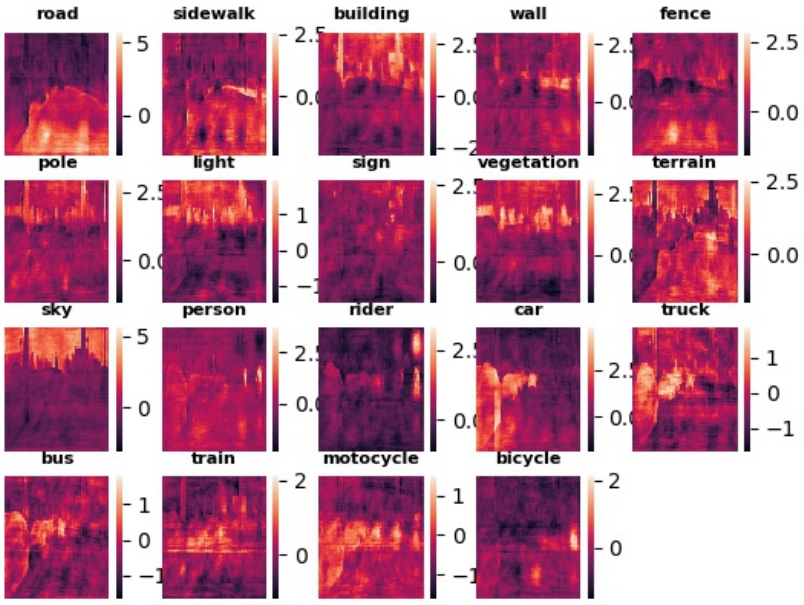


Figure 4.12: Astrous Rates=24, GTA5 → Cityscapes

Chapter 5

Conclusions and Future Work

This thesis set out to improve the performance of adversarial training based domain adaptation methods by solving the following two problems with SEAT and multi-level adaptation respectively. First, the conventional batch normalization layer confines the alignment of source distribution, target distribution and GT distribution. Second, poor adaptation in the lower-level features of the network. The study contributes to our understanding of the limitation of conventional batch normalization layer in domain adaptation task and how distribution being aligned during training, which may be of assistance to the design of normalization layer in this area.

The research has shown that both SEAT and multi-level adaptation are able to improve network's performance in different scales. The base model is able to gain 2.4 and 2.9 mIoU improvement respectively on two adaptation scenarios when it is combined with SEAT and multi-level adaptation. Also, a network with SEAT has a faster convergence speed than that of without SEAT. And SEAT is capable of boosting the performance of other domain adaptation methods by simply adding it durning training.

The major limitation of our study is the lack of experiments support on our assumption of the conflict of affine transformation in conventional batch normalization layer. The class imbalance may also affect the experiment results. As being shown at the appendices A.2, large scale objects dominate the two datasets (GTA5 and SYNTHIA). The phenomenon leads to model bias during training. In other words, the trained network performs better

at those dominated classes while worst at others.

Considerably more works will be done to prove the existence of the conflict on conventional batch normalization and creation of source domain dataset with equivalence class distribution. Finally, as there are many variants of batch normalization, like layer norm [3], group norm [78], etc. We believe the affine transformation in the variants can also be replaced with separate affine transformation if utilizing them in domain adaptation task.

References

[1] Grigory Antipov, Moez Baccouche, and Jean-Luc Dugelay. Face aging with conditional generative adversarial networks. In *2017 IEEE international conference on image processing (ICIP)*, pages 2089–2093. IEEE, 2017.

[2] André Araujo, Wade Norris, and Jack Sim. Computing receptive fields of convolutional neural networks. *Distill*, 4(11):e21, 2019.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[4] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[5] Shai Ben-David, John Blitzer, Koby Crammer, Fernando Pereira, et al. Analysis of representations for domain adaptation. *Advances in neural information processing systems*, 19:137, 2007.

[6] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[7] Xiao Bian, Ser Nam Lim, and Ning Zhou. Multiscale fully convolutional network with application to industrial inspection. In *2016 IEEE winter conference on applications of computer vision (WACV)*, pages 1–8. IEEE, 2016.

[8] Huang Bin, Chen Weihai, Wu Xingming, and Lin Chun-Liang. High-quality face image sr using conditional generative adversarial networks. *arXiv preprint arXiv:1707.00737*, 2017.

[9] Wei-Lun Chang, Hui-Po Wang, Wen-Hsiao Peng, and Wei-Chen Chiu. All about structure: Adapting structural information across domains for boosting semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1900–1909, 2019.

[10] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*, 2014.

[11] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[12] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. arxiv 2017. *arXiv preprint arXiv:1706.05587*, 2019.

[13] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 801–818, 2018.

[14] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

[15] Thomas M Cover. *Elements of information theory*. John Wiley & Sons, 1999.

[16] Ayushman Dash, John Cristian Borges Gamboa, Sheraz Ahmed, Marcus Liwicki, and Muhammad Zeshan Afzal. Tac-gan-text conditioned auxiliary classifier generative adversarial network. *arXiv preprint arXiv:1703.06412*, 2017.

[17] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE international conference on computer vision*, pages 2650–2658, 2015.

[18] Matheus Gadelha, Subhransu Maji, and Rui Wang. 3d shape induction from 2d views of multiple objects. In *2017 International Conference on 3D Vision (3DV)*, pages 402–411. IEEE, 2017.

[19] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The journal of machine learning research*, 17(1):2096–2030, 2016.

[20] Alberto Garcia-Garcia, Sergio Orts-Escolano, Sergiu Oprea, Victor Villena-Martinez, and Jose Garcia-Rodriguez. A review on deep learning techniques applied to semantic segmentation. *arXiv preprint arXiv:1704.06857*, 2017.

[21] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[22] Rui Gong, Wen Li, Yuhua Chen, and Luc Van Gool. Dlow: Domain flow for adaptation and generalization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2477–2486, 2019.

[23] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[24] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[26] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei Efros, and Trevor Darrell. Cycada: Cycle-consistent adversarial domain adaptation. In *International conference on machine learning*, pages 1989–1998. PMLR, 2018.

[27] Judy Hoffman, Dequan Wang, Fisher Yu, and Trevor Darrell. Fcns in the wild: Pixel-level adversarial and constraint-based adaptation. *arXiv preprint arXiv:1612.02649*, 2016.

[28] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[29] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[30] Yanghua Jin, Jiakai Zhang, Minjun Li, Yingtao Tian, Huachun Zhu, and Zhihao Fang. Towards the automatic anime characters creation with generative adversarial networks. *arXiv preprint arXiv:1708.05509*, 2017.

[31] Guoliang Kang, Yunchao Wei, Yi Yang, Yueting Zhuang, and Alexander G Hauptmann. Pixel-level cycle association: A new perspective for domain adaptive semantic segmentation. *arXiv preprint arXiv:2011.00147*, 2020.

[32] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[33] Alireza Kashanipour, Narges Shamshiri Milani, Amir Reza Kashanipour, and Hadi Haji Eghrary. Robust color classification using fuzzy rule-based particle swarm optimization. In *2008 Congress on Image and Signal Processing*, volume 2, pages 110–114. IEEE, 2008.

[34] Jack Kiefer and Jacob Wolfowitz. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, pages 462–466, 1952.

[35] Myeongjin Kim and Hyeran Byun. Learning texture invariant representation for domain adaptation of semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12975–12984, 2020.

[36] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[37] Teuvo Kohonen. The'neural'phonetic typewriter. *computer*, 21(3):11–22, 1988.

[38] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[39] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[40] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[41] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[42] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017.

[43] Claude Lemaréchal. Cauchy and the gradient method. *Doc Math Extra*, 251(254):10, 2012.

[44] Guangrui Li, Guoliang Kang, Wu Liu, Yunchao Wei, and Yi Yang. Content-consistent matching for domain adaptive semantic segmentation. In *European Conference on Computer Vision*, pages 440–456. Springer, 2020.

[45] Qing Lian, Fengmao Lv, Lixin Duan, and Boqing Gong. Constructing self-motivated pyramid curriculums for cross-domain semantic segmentation: A non-adversarial approach. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6758–6767, 2019.

[46] Weizhe Liu, David Ferstl, Samuel Schulter, Lukas Zebedin, Pascal Fua, and Christian Leistner. Domain adaptation for semantic segmentation via patch-wise contrastive learning. *arXiv preprint arXiv:2104.11056*, 2021.

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

[48] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael Jordan. Learning transferable features with deep adaptation networks. In *International conference on machine learning*, pages 97–105. PMLR, 2015.

[49] Yawei Luo, Liang Zheng, Tao Guan, Junqing Yu, and Yi Yang. Taking a closer look at domain shift: Category-level adversaries for semantics consistent domain adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2507–2516, 2019.

[50] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Citeseer, 2013.

[51] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[52] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

[53] Richard Nock and Frank Nielsen. Statistical region merging. *IEEE Transactions on pattern analysis and machine intelligence*, 26(11):1452–1458, 2004.

[54] Curtis G Northcutt, Anish Athalye, and Jonas Mueller. Pervasive label errors in test sets destabilize machine learning benchmarks. *arXiv preprint arXiv:2103.14749*, 2021.

[55] Fei Pan, Inkyu Shin, Francois Rameau, Seokju Lee, and In So Kweon. Unsupervised intra-domain adaptation for semantic segmentation through self-supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3764–3773, 2020.

[56] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2536–2544, 2016.

[57] Guim Perarnau, Joost Van De Weijer, Bogdan Raducanu, and Jose M Álvarez. Invertible conditional gans for image editing. *arXiv preprint arXiv:1611.06355*, 2016.

[58] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17, 1964.

[59] Aman Raj, Daniel Maturana, and Sebastian Scherer. Multi-scale convolutional architecture for semantic segmentation. *Robotics Institute, Carnegie Mellon University, Tech. Rep. CMU-RITR-15-21*, 2015.

[60] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. In *International Conference on Machine Learning*, pages 1060–1069. PMLR, 2016.

[61] Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In *European conference on computer vision*, pages 102–118. Springer, 2016.

[62] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[63] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio 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*, pages 3234–3243, 2016.

[64] Anirban Roy and Sinisa Todorovic. A multi-scale cnn for affordance segmentation in rgb images. In *European conference on computer vision*, pages 186–201. Springer, 2016.

[65] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *Proceedings of the 32nd international conference on neural information processing systems*, pages 2488–2498, 2018.

[66] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[67] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.

[68] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going

deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[69] Tijmen Tieleman, Geoffrey Hinton, et al. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[70] Marco Toldo, Umberto Michieli, and Pietro Zanuttigh. Unsupervised domain adaptation in semantic segmentation via orthogonal and clustered embeddings. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1358–1368, 2021.

[71] Yi-Hsuan Tsai, Wei-Chih Hung, Samuel Schulter, Kihyuk Sohn, Ming-Hsuan Yang, and Manmohan Chandraker. Learning to adapt structured output space for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7472–7481, 2018.

[72] Alan Turing. Intelligent machinery. 1948. *The Essential Turing*, page 395, 1969.

[73] Subeesh Vasu, Nimisha Thekke Madam, and AN Rajagopalan. Analyzing perception-distortion tradeoff using enhanced perceptual super-resolution network. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pages 0–0, 2018.

[74] Tuan-Hung Vu, Himalaya Jain, Maxime Bucher, Matthieu Cord, and Patrick Pérez. Advent: Adversarial entropy minimization for domain adaptation in semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2517–2526, 2019.

[75] Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, et al. Deep high-resolution representation learning for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 2020.

[76] Zhonghao Wang, Mo Yu, Yunchao Wei, Rogerio Feris, Jinjun Xiong, Wen-mei Hwu, Thomas S Huang, and Honghui Shi. Differential treatment for stuff and things: A

simple unsupervised domain adaptation method for semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12635–12644, 2020.

[77] Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T Freeman, and Joshua B Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 82–90, 2016.

[78] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[79] Zuxuan Wu, Xintong Han, Yen-Liang Lin, Mustafa Gokhan Uzunbas, Tom Goldstein, Ser Nam Lim, and Larry S Davis. Dcan: Dual channel-wise alignment networks for unsupervised scene adaptation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 518–534, 2018.

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

[81] Yanchao Yang and Stefano Soatto. Fda: Fourier domain adaptation for semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[82] Raymond A Yeh, Chen Chen, Teck Yian Lim, Alexander G Schwing, Mark Hasegawa-Johnson, and Minh N Do. Semantic image inpainting with deep generative models. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5485–5493, 2017.

[83] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 5907–5915, 2017.

[84] Yang Zhang, Philip David, Hassan Foroosh, and Boqing Gong. A curriculum domain adaptation approach to the semantic segmentation of urban scenes. *IEEE transactions on pattern analysis and machine intelligence*, 42(8):1823–1841, 2019.

[85] Yang Zhang, Philip David, and Boqing Gong. Curriculum domain adaptation for semantic segmentation of urban scenes. In *Proceedings of the IEEE international conference on computer vision*, pages 2020–2030, 2017.

[86] Sicheng Zhao, Xiangyu Yue, Shanghang Zhang, Bo Li, Han Zhao, Bichen Wu, Ravi Krishna, Joseph E Gonzalez, Alberto L Sangiovanni-Vincentelli, Sanjit A Seshia, et al. A review of single-source deep unsupervised visual domain adaptation. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[87] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

APPENDICES

Appendix A

A.1 Neural Networks

The Figure A.1 shows an example of an NN with three hidden layers. The NN consists of numerous neurons and connections, where each connection represents the output of one neuron, an input of another neuron as well. Also, a weight is assigned to each connection, representing how important one neuron is to another.

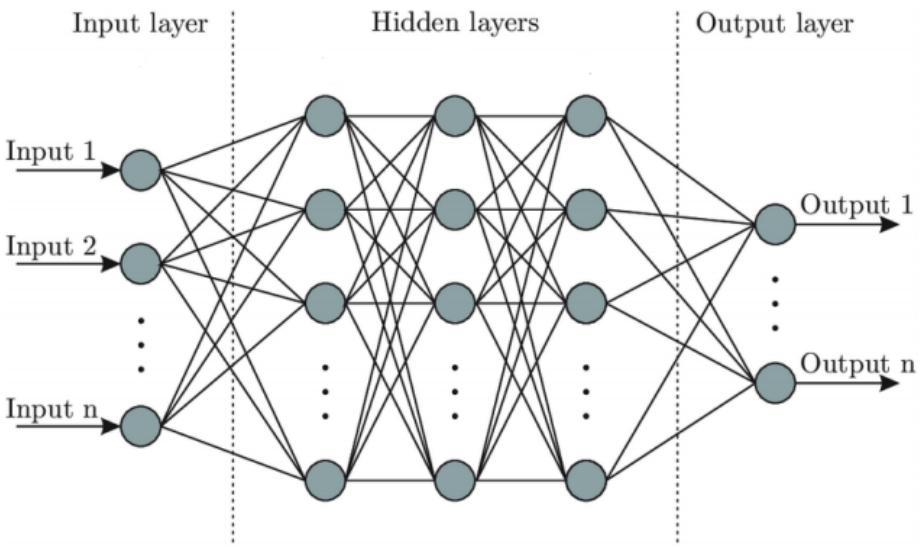


Figure A.1: An neural network with 3 hidden layers. Green cycles represent neurons; Black lines represent connection between each neuron.

A.1.1 Forward Propagation

The forward propagation refers to the calculation process of one NN starting from the input to the output, where the input provides the initial information, then propagates through each hidden layer and finally produces the output [23]. A simplified NN shown at figure A.2 is taken as an example to demonstrate the forward propagation, where the calculation process is shown below, where x, w, b, h, y represent input, weight, bias, hidden layer, output respectively.

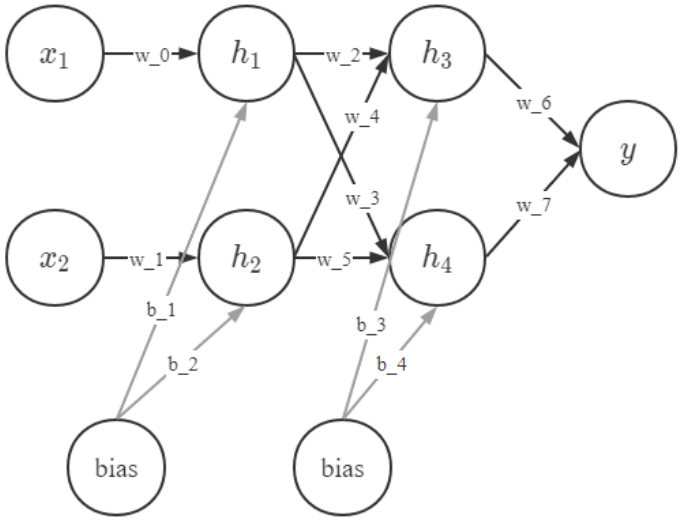


Figure A.2: A simplified neural network with two hidden layers.

$$h_1 = w_0 * x_1 + b_1 \tag{A.1}$$

$$h_2 = w_1 * x_2 + b_2 \tag{A.2}$$

$$h_3 = w_2 * h_1 + w_4 * h_2 + b_3 \tag{A.3}$$

$$h_4 = w_3 * h_1 + w_5 * h_2 + b_4 \tag{A.4}$$

$$y = w_6 * h_3 + w_7 * h_4 \tag{A.5}$$

$$= w_6 * (w_2 * h_1 + w_4 * h_2 + b_3) + w_7 * (w_3 * h_1 + w_5 * h_2 + b_4) \tag{A.6}$$

$$= w_6 * (w_2 * (w_0 * x_1 + b_1) + w_4 * (w_1 * x_2 + b_2) + b_3) \tag{A.7}$$

$$+ w_7 * (w_3 * (w_0 * x_1 + b_1) + w_5 * (w_1 * x_2 + b_2) + b_4)$$

A.1.2 Activation Function

As shown at the Equation A.7, the NN without activation functions is a linear system, which means that it is limited on learning complex mapping from data. Thus, non-linear activation functions, like Rectified linear Unit (ReLU) [52], Leaky Rectified linear Unit(Leaky ReLU) [50], are necessary to help the network learn complex data and representations. Normally, the activation function is add at the output of each neuron before being sent to the next neuron.

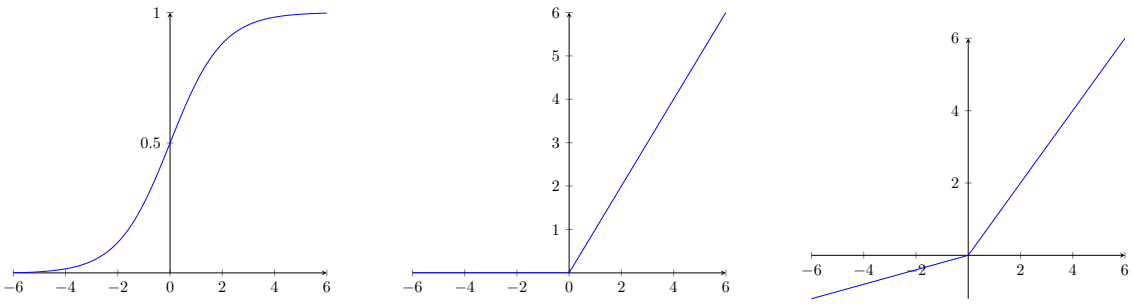
Also, the network does not know the bound of its output. It could be ranged from -inf to +inf. Therefore, some activation functions like sigmoid, softmax [23] are needed to scale the output into desired ranges.

The details for the mentioned activation functions are introduced below.

Sigmoid

The sigmoid function, ranges within (0, 1), which is commonly used in machine learning for producing the Bernoulli distribution, which is showed at the Figure A.3 (a).

$$f(x) = \frac{1}{1 + e^{-x}} \tag{A.8}$$



(a) sigmoid (b) ReLU (c) leakyReLU

Figure A.3: The sigmoid function

However, as shown in figure A.3 (a), the sigmoid function becomes very flat when the input tends to be very positive or very negative, which means that the function becomes

insensitive in the small changes of its input resulting in no gradient or minimal gradient during the backpropagation, so-called vanishing gradient problem.

Rectified Linear Unit (ReLU)

The NN and CNN make use of the ReLU function instead of the sigmoid function nowadays because of the vanishing gradient problem. The ReLU has output 0 if the input is negative, otherwise, the output is equal to the input. The Figure A.3 (b) shows how it looks like.

$$f(x) = max(0, x) \tag{A.9}$$

The ReLU function is non-linear, and it has the following advantages compared with the sigmoid function.

- Faster speed because 50% of hidden units are inactivated if the input is evenly distributed around zero.
- Better gradient propagation due to fewer gradient vanishing problems.
- More efficient on computation due to the only comparison, addition and multiplication operations.

However, the vanishing gradient problem still exists when the input lies below 0 for the ReLU function, and it is non-differentiable at Zero.

Leaky ReLU

One way to solve the gradient vanishing problem for the ReLU function is to use a small slope instead of deactivating all neurons when the input lies below 0, so called Leaky ReLU function. And it's shown at the figure A.3.

$$\phi(x) = \begin{cases} af(x) & x \leq 0 \\ x & x > 0 \end{cases} \tag{A.10}$$

Softmax

The softmax function is not only used to map the network's output lie in range [0, 1] but also map the total sum of the output along the channel to be 1. It's widely used at the final output for multi class logistic regression problems while the sigmoid function is for binary class logistic regression problems.

$$f(\mathbf{z})_i = \frac{e^{\mathbf{z}_i}}{\sum_{j=1}^{K} e^{\mathbf{z}_j}} \tag{A.11}$$

For i in 0,1,2...K, where i is the channel index.

A.1.3 Backward Propagation

During training, the forward propagation produces the output and the loss (cost). At the same time, the backward propagation allows the information from the loss to flow backward through the NN and calculates the gradient of each neuron [23]. After that, the learning algorithm such as stochastic gradient descent performs learning using the gradient, which is discussed at A.1.5. The Figure A.2 is used as an example NN to demonstrate the backward propagation. According to the chain rule of calculus, the gradient of w_6, w_2, and w_0 are shown below, where L denotes the loss.(activation functions are omitted for simplification).

$$\frac{\partial L}{\partial w_6} = \frac{\partial L}{\partial y} * \frac{\partial y}{\partial w_6} \tag{A.12}$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial y} * \frac{\partial y}{\partial h_3} * \frac{\partial h_3}{\partial w_2} \tag{A.13}$$

$$\frac{\partial L}{\partial w_0} = \frac{\partial L}{\partial y} * (\frac{\partial y}{\partial h_3} * \frac{\partial h_3}{\partial h_1} * \frac{\partial h_1}{\partial w_0} + \frac{\partial y}{\partial h_4} * \frac{\partial h_4}{\partial h_1} * \frac{\partial h_1}{\partial w_0}) \tag{A.14}$$

A.1.4 Gradient Descent

The gradient descent [43] is introduced to minimize the loss (cost) function, where the Figure A.4 gives an example of how a function $f(x) = \frac{1}{2}x^2$ reach its minimum by using the gradient descent algorithm. Identically, if we have the gradient of the w_i from the

backward propagation, where $i \in \mathbb{R}$, the following equation can be used to update the w_i.

$$w_i^{'} = w_i - \epsilon * \nabla_{w_i} f(w_i) \tag{A.15}$$

where ϵ denotes the learning rate.

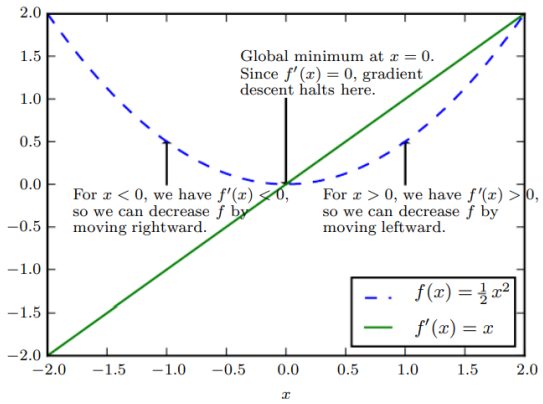


Figure A.4: An illustration of how the gradient descent algorithm find the global minimum by using the derivatives of a function, where the figure is captured from [23].

A.1.5 Loss Function and Learning Algorithm

The function we want to minimize or maximize is called the loss function or cost function [23]. Choosing the proper loss function for the NN and CNN can be seen as one of the important aspects for designing the network. Fortunately, there are many loss functions for us to select in different tasks. For examples, normally, for classification tasks, the cross-entropy loss (multi classes) and the binary cross-entropy loss (binary classes) [15] are commonly chosen. While for linear regression tasks, mean squared error is one of the popular loss functions.

After selecting the proper loss function, a good optimizer should also be selected to perform gradient descent as an appropriate optimizer can accelerate the convergence of the network and find the global minimum.

Stochastic Gradient Descent

SGD [34] is one of the most popular learning algorithm for DL. It performs gradient descent by randomly selecting m samples and calculating the average gradient. The algorithm 1.1 illustrates the process. The learning rate is a crucial parameter for the SGD. A too-small learning rate leads to slow convergence. On the contrary, a too-large learning rate could hinder it, as the loss function fluctuates around the minimal point.

Algorithm 1.1 Stochastic Gradient Descent

ϵ: Learning rate

θ: Initial parameters

X_m, Y_m: A mini-batch with m samples and its corresponds targets

X_n, Y_n: Training set with n samples and its corresponds targets

while criterion not met **do**

$X_m, Y_m \leftarrow X_n, Y_n$

$\hat{g} \leftarrow \nabla_\theta \frac{1}{m} \sum_{(X_m, Y_m)} L(f(x_m, \theta), y_m)$

$\theta \leftarrow \theta - \epsilon * \hat{g}$

end while

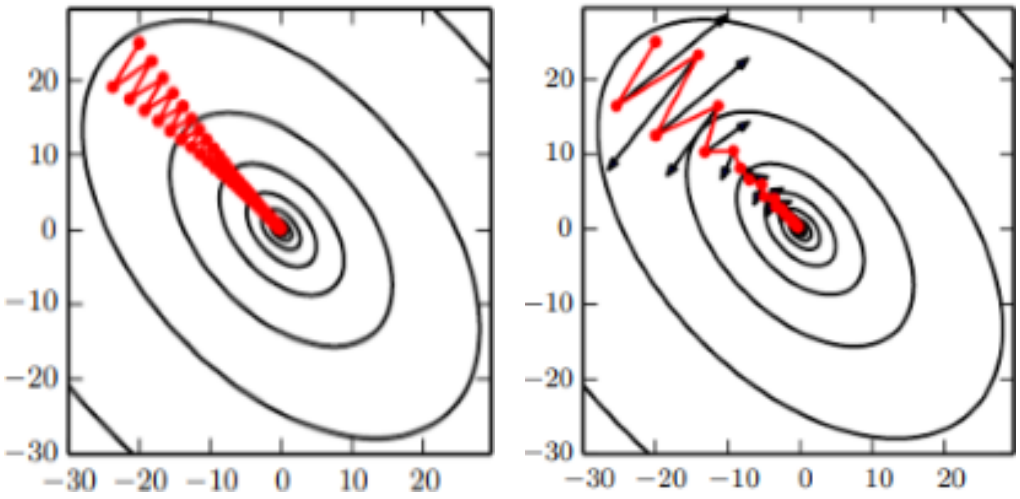
Adam

In order to faster convergence speed while using a small learning rate, momentum [58] based learning alorithms are proposed [36, 67, 69]. The momentum algorithm accumulates previous gradients with current one and continues to move in their direction.

$$v = \alpha v - \epsilon \nabla_\theta \frac{1}{m} \sum_{(X_m, Y_m)} L(f(x_m, \theta), y_m) \tag{A.16}$$

$$\theta = \theta + v \tag{A.17}$$

According to the Newton's law of motion, the equation A.16 can be seen as $v = v + at$, where a is the negative gradient, t is the unit time. As for the v in the equation A.17, it could be regarded as the momentum (mv) with $m = 1$.That's how the name "momentum"



(a) with out momentum (b) with momentum

Figure A.5: Illustration of gradient descent w/wo momentum. The red line indecates the path followed by gradient descent. The Figure is captured from [23].

comes from. The Figure A.5 (a) and (b) show the difference between the gradient descent with and without momentum respectively.

Setting the learning rate is difficult as it significantly impacts the model performance. Also, the loss function is susceptible to some directions in parameter space while insensitive to others [23]. The momentum algorithm can ease these problems, but the performance still needs improvement. Thus, the adaptive learning rate is introduced, using different learning rates for parameters and updating automatically throughout the learning.

Speaking of the momentum and adaptive learning rate based learning algorithms, Adam [36] is one of the most used ones by researches bacuase of its robustness on choosing hyperparameters. The Adam is presented in the algorithm 1.2.

A.2 Label Distribution of GTA5 and SYNTHIA

Algorithm 1.2 Adam

ϵ: Learning rate

θ: Initial parameters

δ: Small constant (default: 10^{-8})

ρ_1, ρ_2: A pair of decay rates in [0,1)

X_m, Y_m: A mini-batch with m samples and its corresponds targets

X_n, Y_n: Training set with n samples and its corresponds targets

Initialize time step t=0

Initialize 1st and 2nd moment varianles s=0, r=0

while criterion not met **do**

$X_m, Y_m \leftarrow X_n, Y_n$

$\hat{g} \leftarrow \nabla_\theta \frac{1}{m} \sum_{(X_m, Y_m)} L(f(x_m, \theta), y_m)$

$t \leftarrow t + 1$

$s \leftarrow \rho_1 s + (1 - \rho_1)\hat{g}$

$r \leftarrow \rho_2 r + (1 - \rho_2)\hat{g} \odot \hat{g}$

$\hat{s} \leftarrow \frac{s}{1-\rho_1^t}$

$\hat{r} \leftarrow \frac{r}{1-\rho_2^t}$

$\Delta\theta = -\epsilon \frac{\hat{s}}{\sqrt{\hat{r}}+\delta}$

$\theta \leftarrow \theta + \Delta\theta$

end while

Label Distribution of GTA5 (%)

road	sildewalk	building	wall	fence	pole	light	sign	vegetation	terrain
36.41	9.20	19.19	2.06	0.74	1.23	0.15	0.09	8.40	2.25
sky	person	rider	car	truck	bus	train	motocycle	bicycle	-
15.3	0.4	0.03	2.85	1.28	0.33	0.03	0.03	2.03	-

Table A.1: 2000 images are randomly sampled to represent the whole dataset.

Label Distribution of SYNTHIA (%)							
road	sildewalk	building	wall	fence	pole	light	sign
19.10	19.72	30.32	0.33	0.31	1.11	0.04	0.10
vegetation	sky	person	rider	car	bus	motocycle	bicycle
10.54	7.53	4.40	0.48	4.08	1.49	0.20	0.23

Table A.2: 2000 images are randomly sampled to represent the whole dataset.

A.3 Class Proportion

According to the Table A.3, we define large, medium, small scale objects by their average proportion. More specifically, large-scale, medium-scale, small-scale objects represent those objects with proportion of [5%, inf), [2%, 5%), [0, 2%) respectively.

Class Proportion (%)									
road	sildewalk	building	wall	fence	pole	light	sign	vegetation	terrain
38.2	6.0	22.8	2.73	3.5	1.6	1.0	1.3	17.8	3.0
sky	person	rider	car	truck	bus	train	motocycle	bicycle	-
4.2	2.7	1.2	7.3	3.8	3.8	3.5	1.7	2.0	-

Table A.3: The average class Proportion of the test set of the Cityscapes dataset.