Synthetic-to-Real Domain Adaptation for Semantic Segmentation of Street Scenes

Mikkel Kristensen, 201405741

Louis Marchant, 201505750

Master's Thesis, Computer Science

June 2020

Advisors: Henrik Pedersen, Ira Assent

AARHUS
UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

Abstract

Producing ground truth for supervised machine learning is often a laborious task. Synthetic data, where ground truth can be generated automatically, is becoming an increasingly popular approach to alleviating this problem. However, realistic data can be hard to fabricate, and models trained on one domain often perform poorly on another, different, domain. Domain adaptation techniques aim to reduce this drop in performance. In this thesis we investigate the topic of unsupervised domain adaptation for semantic segmentation, using the Playing for Data dataset as the source domain, and the Cityscapes dataset as the target domain. We categorise the adversarial approaches in this area into three levels. Feature-level approaches work by having a discriminator differentiate between feature maps extracted from the source and the target domain, with the purpose of utilising an adversarial loss to encourage the extraction of domain-invariant features, or the generation of target-like features. Output-level approaches instead have the discriminator operate on, and thus encourage the alignment of, the semantic segmentation outputs or a transformation thereof. Input-level approaches perform adaptation directly on the training images, producing a transformed dataset closer to that of the target domain. Furthermore, we investigate combinations of these different levels of approaches. Our findings indicate that, while combinations between these levels do not lead to an improvement in performance, adaptation can be successfully achieved at each of the three levels, with the input-level approach using the CycleGAN model achieving the best results.

Resumé

At producere *ground truth* til *supervised machine learning* er ofte en tidskrævende opgave. Syntetiske data, hvor *ground truth* automatisk kan genereres, bliver en stadig mere populær tilgang til at afhjælpe dette problem. Imidlertid kan realistiske data være svære at fremstille, og modeller, der er trænet på ét domæne, fungerer ofte dårligt på et andet, anderledes, domæne. *Domain adaptation* teknikker forsøger at reducere dette fald i ydeevne. I dette speciale undersøger vi emnet *unsupervised domain adaptation* til semantisk segmentering, ved hjælp af datasættet Playing for Data som *source*-domæne og Cityscapes-datasættet som *target*-domæne. Vi kategoriserer *adversarial* tilgange på dette område i tre niveauer. Tilgange på *feature*-niveau fungerer ved at have en diskriminator til at skelne mellem *feature maps*, der er udtrukket fra *source*-domænet og *target*-domænet, med det formål at bruge en *adversarial loss* til at tilskynde udtrækning af domæneinvariante *features*, eller generering af *target*-lignende *features*. Tilgange på outputniveau får i stedet diskriminatoren til at operere på, og dermed tilskynde ensartethed af, de semantiske segmenteringsoutput eller en transformering deraf. Tilgange på inputniveau udfører *domain adaptation* direkte på træningsbillederne og producerer et transformeret datasæt, der er tættere på *target*-domænet. Desuden undersøger vi kombinationer af disse forskellige niveauer af tilgange. Vores eksperimenter viser, at mens kombinationer af disse niveauer ikke fører til en forbedring af ydeevnen, kan *domain adaptation* opnås med succes på hvert af de tre niveauer, mens inputniveautilgangen, der bruger CycleGAN-modellen, opnår de bedste resultater.

Acknowledgments

We would like to thank our advisors Henrik Pedersen and Ira Assent for their invaluable guidance and feedback.

Mikkel Kristensen and Louis Marchant,
Aarhus, June 2020.

Contents

Chapter 1

Introduction

In supervised machine learning we use training data to train our model to, given an input, produce a certain desired output. To do this, the training data must include the correct answer for each input, often referred to as the ground truth. This ground truth is often laborious to produce, especially in the case of semantic segmentation, where a class is assigned to each pixel in an image. Synthetic data is becoming an increasingly popular approach for alleviating this problem, since the task of generating corresponding ground truth can be automated. However, synthetic data is not a perfect substitute for real data as, for instance, realistic imagery is hard to fabricate, and it is often the case that a model trained on one domain will perform poorly on another, different, domain. Domain adaptation is a set of techniques aimed at reducing this drop in performance. Inspired by the vision of autonomous driving, for which understanding city traffic scenes is essential, we explore approaches for domain adaptation for semantic segmentation of street scenes, using the synthetic Playing for Data dataset to train a model that performs well on the Cityscapes dataset of real street scene images.

We begin in Chapter 2 with a summary of concepts and technologies relevant to this thesis. In Chapter 3 we then present several works on unsupervised domain adaptation for semantic segmentation. Through our exploration of these related works we find that most approaches use a discriminator, which is given the job of differentiating between data or extracted information from the two different domains. Aligning these two domains can then be formulated as tricking this discriminator, and can be done at different levels of the training. In Chapter 4, we present the methods that we implement, including our own variations, which we categorise into three levels. We begin with feature-level approaches, which are more seminal in this area, and work on some feature space in the semantic segmentation network. We then move on to output-level approaches, which work on the output of the semantic segmentation network, or a transformation thereof. Then, we look at input-level approaches, which work on the images themselves to produce a new, adapted, dataset. Furthermore, we present implementations that combine approaches of different levels, to see whether or not this improves the adaptation. Finally, in Chapter 5 we present the results of our experiments with the aforementioned approaches, and reflect on these.

Chapter 2

Background

In this chapter we summarise the concepts, techniques and datasets the reader should be familiar with before proceeding with the remainder of this thesis.

2.1 Domain Adaptation

When training supervised models, it can often be the case that we have little or no labelled data in the domain that is our main focus, but have an abundance of labelled data in another similar domain, for which the same task can be learned. We thus want to transfer knowledge from the labelled domain, which we call the *source* domain, to the unlabelled domain that is our main focus, which we call the *target* domain. Assuming the domains are not identical, simply training the model on the labelled data in the source domain and using it to predict labels for the target domain, will inevitably result in worse performance than if we had labelled data for the target domain. Domain adaptation (DA) is a set of techniques that attempts to alleviate this issue. That is, we would like to have a model trained on the source domain be effective on the target domain. In this thesis we will specifically investigate unsupervised domain adaptation (UDA), which assumes no labelled data at all in the target domain. One well-established approach for unsupervised domain adaptation, that will be the main focus of this thesis, is to use principles from generative adversarial networks (GANs) (see Section 2.4). Much more detail will follow, but the general idea is to use a discriminator, which is given the job of differentiating between data or extracted information from the source and target domains. We can then, at various points in the training process, by training the network to trick the discriminator, decrease the distance between the domains and thus decrease the performance gap.

2.2 Semantic Segmentation

We will assume the problem of semantic segmentation is known to the reader, but in this section present a very short refresher and refer to a network architecture commonly used for this problem. Semantic segmentation is a classification task whereby the input is an image, and the output is a segmentation map, where

each pixel in the input image is associated with a class. An example of an image segmentation can be seen in Figure 2.1, where the left side represents an image from a street scene, and the right shows the corresponding ground-truth semantic segmentation, with each pixel being attributed a class. The colours are used to visually represent different classes (for example, blue for car and purple for road). Common networks used to perform semantic segmentation tasks include fully convolutional networks (FCNs).

An FCN, as the name suggests, consists solely of convolutional layers. As is the case with convolutional neural networks, earlier layers extract fine-grained details, and later layers extract more high-level, semantic information from the image. With this in mind, FCNs use skip layers, which combine the outputs from different stages in the convolutional pipeline, upscaling those of lower resolution, and summing these to produce a segmentation map with both fine details and semantic information. Typically, FCNs are used in the form of tested classification architectures (e.g. VGG [33] or GoogLeNet [34]) that has been fully convolutionalised. This also allows for fine-tuning a pre-trained model from classification to segmentation. We will be using VGG16-FCN8s [21], which is the VGG16 architecture fully convolutionalised with 2 skip connections. Details of the implementation of this can be found in Section 4.2.

2.2.1 Cityscapes dataset

Figure 2.1: A street scene image from the Cityscapes dataset, and its corresponding (strongly annotated) segmentation map.

The Cityscapes dataset [6] is a collection of 24998 images of street scenes taken from the roof of a car driven in various cities in Germany. Of these 24998 images, strongly annotated segmentation maps are provided for 5000 of them, and weakly annotated segmentation maps for the remaining 19998. There are 19 classes[1] such as road, building, person, and car. An example of an image and a strongly annotated segmentation map is shown in Figure 2.1. In this thesis, we will use Cityscapes as our *target* domain. As we wish to investigate *unsupervised* domain adaptation, which assumes no annotations from the target domain, we will disregard all Cityscapes annotations during training, and use them only for evaluation purposes.

[1] https://www.cityscapes-dataset.com/dataset-overview/

2.3 Use of Synthetic Data in Deep Learning

Synthetic data is best understood through the motivation for its existence. In supervised machine learning, the quality of a model is directly related to the quality and quantity of relevant labelled data. Procuring and labelling data, such as producing segmentation masks for images, is a laborious task, and thus large, high-quality datasets for many domains are hard to come by. Synthetic data is usually much easier to produce, and allows for a potentially infinite amount of data. In low-level computer vision (CV) problems, synthetic data has been used for quite a long time. For instance, in 1988, randomised geometric shapes were used for evaluating optic flow problems [20]. Modern approaches to low-level CV problems using synthetic data include, for example, Dosovitsky et al.'s Flownet [8], where they produce the *Flying Chairs* dataset, and showed that optical flow could be predicted using a CNN on this synthetic dataset with competitive results. In high-level CV problems 3D datasets have been created to train models to solve problems such as image segmentation and classification. An example of this is ShapeNet [2], a dataset of over 3 million CAD models of various objects. This has for example been used in PointNet [27], a system trained on ShapeNet as well as other 3D datasets for the purpose of classification and segmentation on point clouds. Aside from building datasets from scratch, work has been done in reusing assets from pre-existing sources, such as *Playing for Data* [29], a synthetic street image dataset with semantic segmentation maps, captured from the video game Grand Theft Auto V.

There is a large main downside to using synthetic data, however, which is also the focus of this thesis. It is often the case that synthetic data is not similar *enough* to the data it is attempting to mimic. This lack of similarity therefore manifests itself as an instance of the domain adaptation problem, and thus we can hopefully alleviate this problem using domain adaptation techniques.

2.3.1 Playing for Data dataset

The Playing for Data dataset [29], is a collection of 24966 images captured from the video game Grand Theft Auto V[2] with strongly annotated segmentation maps. The images are taken from mostly the same viewpoint as in the Cityscapes dataset, the roof of a car, and additionally use the same classes as in the Cityscapes dataset. Utilising the game engine, the average annotation time for images in this dataset was 7 seconds, which is 771 times faster than for the Cityscapes dataset, where each image took 1.5 hours on average to annotate. Since there are many strongly annotated images, significantly more than Cityscapes, and that many more could hypothetically be easily generated, this dataset and the Cityscapes dataset present a good use-case for synthetic-to-real domain adaptation. We will thus use the Playing for Data dataset as our *source* domain. An example of an image from the game, and a corresponding segmentation, can be seen in Figure 2.2.

[2] https://en.wikipedia.org/wiki/Grand_Theft_Auto_V

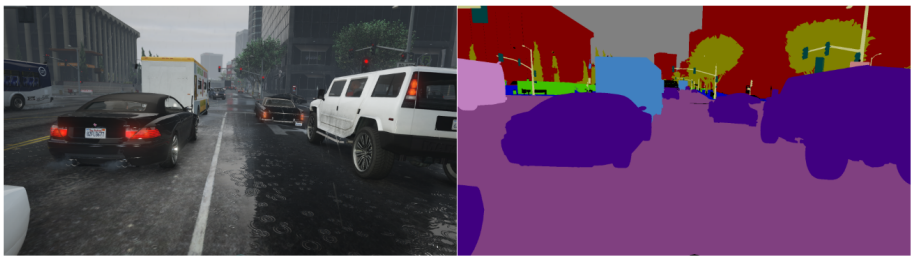


Figure 2.2: A street scene image from the Playing for Data dataset, and its corresponding segmentation map.

2.4 Generative Adversarial Networks

We will assume that the topic of generative models is known to the reader, but as a small summary, generative models attempt to generate new examples from a given distribution, rather than discriminate between instances of it. A popular approach to this is to use generative adversarial networks (GANs) [10]. A GAN consists of two networks, a discriminator and a generator, competing against each other in an adversarial fashion. The generator attempts to, typically taking a noise vector as input, produce new instances of a given dataset, and the discriminator attempts to distinguish between these generated instances and true instances from the dataset. The loss for the generator is defined based on how poorly it makes the discriminator believe the generated instances are real, and the discriminator's loss is defined as how poorly it is able to distinguish the real instances from the generated. These losses are optimised in a step-by-step fashion. Thus, at first, the generator will generate essentially random noise, which the discriminator will then learn to distinguish from real samples. This, in turn, will force the generator to change the distribution of its output to better match the true distribution of the training set, and so on. One hopes to eventually obtain a generator that is capable of producing convincing examples. Several extensions of GANs exist, and one that is especially relevant for domain adaptation is CycleGAN [44], which we will summarise in the next subsection.

2.4.1 CycleGAN

Several networks, such as autoencoders or pix2pix [17], exist to perform image-to-image translation using paired training data with input-output examples. CycleGAN [44], however, is a network that attempts to solve the problem of unpaired image-to-image translation. Unpaired image-to-image translation is the problem of, given two sets of unpaired image domains, taking an image from one domain, and translating it in such a way so as to make it look as though it was sampled from the other domain, but doing so without having any existing input-output examples of such a translation to learn from. Importantly, however, we wish to preserve the semantic information in the image. To illustrate this point with an example, we use Figure 2.3, where an image is translated from the domain *Zebra* to the domain *Horse*. We are not simply interested in being able to convert an image of zebras to any image containing horses. We want the

semantic information preserved, such as how many horses or zebras are in the image, where they are facing, etc. The result is an image that is recognisably the same on a high, semantic, level, but different on a lower, often textural, level.

Figure 2.3: An example of image to image translation, from one domain *Zebra* to another domain *Horse*. Image is from the original CycleGAN paper [44].

The CycleGAN architecture achieves this by utilising two GANs, each of which takes an image as input instead of a noise vector. The first GAN aims to map images from the first domain to the second domain, and the second GAN aims to do the opposite. Trained independently, these GANs might output images that are good examples of its target domain, but nothing would enforce the input and the output to be recognisably the same. To address this issue, a cycle consistency loss is added. This loss enforces that, if you translate the image to the other domain and back again, using the other GAN, you should get something that looks similar to the original image. The network is further detailed in Section 4.5.

2.5 A Synthetic-to-Real Domain Adaptation Example

While having simpler domains and a simpler task than ours, to provide some context, we mention an early application of unsupervised synthetic-to-real domain adaptation for visual recognition using deep learning. This example thus illustrates a successful combination of the techniques and concepts discussed in this chapter. The topic of interest was gaze estimation, which is the task of attempting to determine the direction an eye is looking based on an image of the eye. Shrivastava et al. [32], who were working on the gaze estimation problem, trained on synthetic images from the *UnityEyes* dataset [37], but prior to training, used a GAN they named SimGAN to refine these images based on the *MPIIgaze* dataset of real images of eyes [41]. The SimGAN architecture is based on traditional GANs, namely a discriminator for differentiating between synthetic and real images, and a generator for creating images to trick the discriminator. SimGAN however, instead of taking a vector of random noise as an input to the generator, uses the synthetic image as an input, and adds a loss to ensure that the refined image does not lose important annotated information, which in this case is gaze direction. The authors tried different approaches to achieving this loss, and their experiments used the L1 norm of the difference between the synthetic image before and after refinement. The network did not

utilise any of the labels from the real data, and instead only focused on refining in an unsupervised fashion. The ideas presented in SimGAN, especially that of balancing refinement with preserving annotated data, were extended by Sela et al. [31], who developed GazeGAN. They utilise CycleGAN to provide a method of refining the synthetic eye images. The cycle consistency loss from CycleGAN helps preserve the gaze direction, since the cycle consistency loss punishes the network for making macro-level changes in the data, such as the gaze direction, as opposed to micro-changes such as texture.

2.6 Finishing Up

Having given an introduction to the concepts and techniques being investigated in this thesis, we will in the next chapter look at related work done in the area of unsupervised domain adaptation for semantic segmentation of street scenes, to later use this as a foundation for investigation, experimentation, and combination of different techniques.

Chapter 3

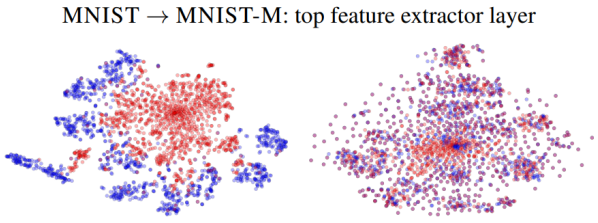
Related Work

In this chapter we present several works on unsupervised domain adaptation for semantic segmentation. Most of these deal with segmentation of street scenes, often including experiments on the GTA V → Cityscapes domains, the accuracy results of which we include in Section A.1. We also present some other works that exemplify main ideas in this area.

3.1 Adversarial Approaches

A common technique for domain adaptation is to encourage the extracted features to be domain-invariant, i.e. similar for the two domains, and then have the subsequent part of the network decode and upsample from these features. This therefore implicitly relies on an underlying assumption that source and target data share a prediction function in a domain-invariant feature space. One seminal approach presented by Ganin and Lempitsky [9] modifies a standard feed-forward network by introducing a domain classifier that takes as input the last layer of the feature extractor part of the feed-forward network. In addition to training the main feed-forward network on labelled source data to perform the necessary task, the domain classifier is trained to distinguish whether a feature map comes from a source or target image using a classification loss. Now, the idea is to negate this classification loss when the backpropagation reaches the feature extractor. Thus, the domain classifier is trained as normal, while the feature extractor learns to produce features that increase this loss, i.e., features that the discriminator has trouble classifying. Through this *gradient reversal*, domain-invariant features are achieved. See Figure 3.1, which shows a t-SNE [24] representation of activations of a feature extraction layer for the source and target domain. The source domain (blue) and target domain (red) are distributed a lot more similarly when adaptation is performed, thus making the features extracted more domain-invariant. Other works, which we will not explore further, produce domain-invariant features by minimising a distance metric between the domains called the Maximum Mean Discrepancy [22, 23]. The predominant way, however, to achieve the same dynamic is to, similar to a GAN, use an additional *adversarial* domain classification loss for the generator

with the labels flipped, i.e., it should trick the discriminator into classifying feature maps from the source domain as originating from the target domain, and vice versa.



(a) Non-adapted (b) Adapted

Figure 3.1: The effect of adaptation on the distribution of extracted features as presented by Ganin and Lempitsky [9], with *blue* being source domain examples and *red* being target domain examples.

The first to apply this form of *adversarial training* for unsupervised domain adaptation for semantic segmentation were Hoffman et al. in their paper *FCNs in the Wild* [14]. They use an FCN to perform the segmentation, and perform domain adaptation by introducing a combined loss function consisting of the standard segmentation loss, a *global domain alignment* loss, and a *category-specific adaptation* loss. The global domain alignment loss utilises adversarial training, but does so on each individual cell of the resulting feature map instead of the feature map as a whole. The category-specific adaptation draws inspiration from weak learning methods, where only image-level labels are provided. Since no target labels are available, these image-level labels are approximated using the model trained on the source domain with the global alignment loss. Additionally, for each image in the source domain with class c, the percentage of pixels attributed to c is computed. Over all images, a lower and upper 10% boundary, along with the average, is calculated for each class. Finally, for a given target image for which a class c is present, based on the approximated image-level labels, a loss is added constraining the number of pixels predicted for a given class to be within the expected range of the number of pixels for that class as indicated by the source data. This utilises the knowledge that, for example, roads typically have more pixels in a given street scene than bicycles.

Chen et al. [4] take heavy inspiration from FCNs in the Wild, utilising the same *global domain alignment*. Their main contribution is the addition of *class-wise domain alignment*. This works by first taking the feature map output of the global domain aligned network, and dividing it into a grid. For each cell in the grid, a soft label is generated depending on what percentage of the cell is occupied by each class, as determined by the cell's corresponding pixels in the original image. For source images this can be easily generated using the ground truth, but for target images the prediction from the global domain aligned model is used to generate soft pseudo labels. Then, a loss is generated by performing class-wise adversarial domain discrimination. Which is to say, with c classes,

c discriminators are trained, each with the task of determining whether a cell is from the source or target domain. The loss for each discriminator is then weighted by the probability of the cell belonging to the corresponding class. In addition to this, since their datasets include the same street scenes but at different periods of time, they use existing deep learning frameworks to mine static priors, which is to say, they can leverage the temporal data to see which pixels in a given image belong to static objects, such as buildings. These static priors are used to increase the probability of these specific pixels being assigned to a pseudo label from a static class.

Chen et al. [5] employ an approach that utilises the fact that they are specifically dealing with synthetic-to-real domain adaptation. They make use of an additional network, in the form of a frozen network pre-trained on real images, specifically ImageNet [7]. The main segmentation network is trained on the synthetic source images and labels. At the same time, real target images are passed through both networks, and the feature map output of the main semantic segmentation network is forced to be similar to the feature map output of the pre-trained network. This ensures that the main network does not overfit the synthetic style too much. Furthermore, they perform a domain alignment technique similar to the *global domain alignment* used in *FCNs in the Wild*, but use different discriminators for each region in the image, arguing that this eases the task, as the different regions tend to contain specific spatial information. For example, in an urban scene image, the objects in the central region are usually small, while objects in the outer region are relatively large.

Hong et al. [15] present an approach that, instead of having the main network produce domain-invariant features, uses a separate conditional GAN-inspired generator to transform features of synthetic images to real-image-like features. They thus relax the requirement of a common feature space. A diagram of their system can be seen in Figure 3.2. Specifically, the generator, taking both a noise vector and an early feature map as input, predicts the residual in the feature space between the source and target domain and adds this to the feature maps of

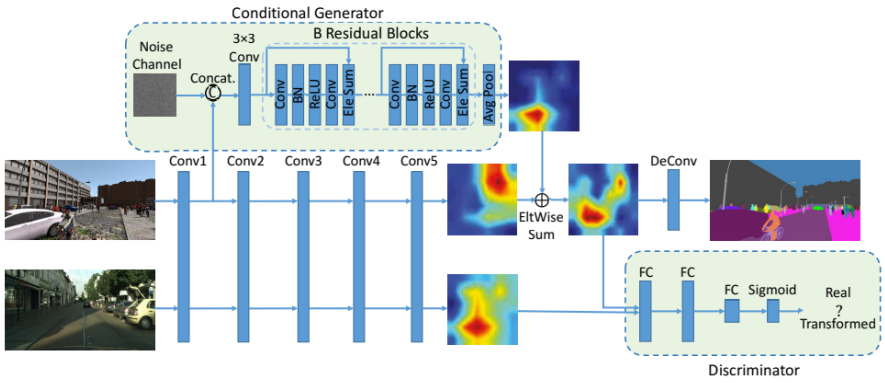


Figure 3.2: A diagram of the system presented by Hong et al. [15].

the source images before being passed to the pixel-wise classifier. As usual, the generator is trained adversarially to trick a discriminator which has been trained to distinguish between source and target feature maps. During test time, the target image is simply passed through the main segmentation network, whilst the generator and discriminator are unused.

3.1.1 Output-Focused Adversarial Approaches

Instead of performing the adversarial adaptation in the feature space, some approaches instead choose do so on the output of the semantic segmentation.

Tsai et al. [35] present an adversarial approach that is inspired by the observation that, whilst source and target domains can be very different in appearance, the segmentation maps share many similarities. An example of this can be seen in Figure 3.3, where it is clear that despite the actual images having a large gap in appearance, the segmentation maps are often still quite similar. Their solution is thus to use an adversarial approach on the output of the semantic segmentation network, using a domain discriminator tasked with differentiating source and target segmentation maps. The semantic segmentation network is then trained to, along with performing semantic segmentation, trick the discriminator into believing segmentation maps from target images originate from source images. However, they note that low-level details are not adapted well, as they are far from the output. To alleviate this issue, they generate a segmentation map from an earlier convolutional layer and perform the adversarial adaptation on this as well. Both layers have their own discriminator for this purpose.

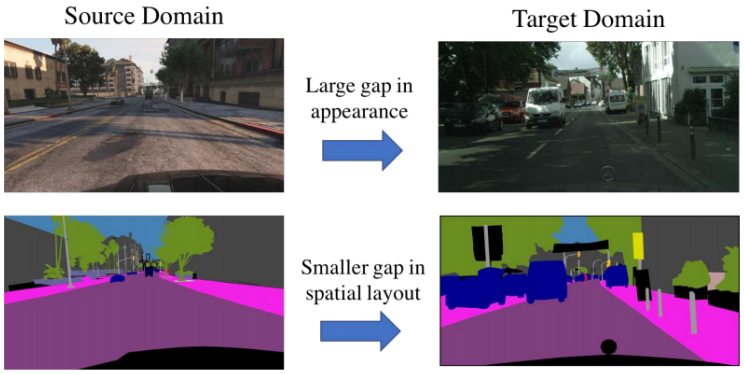


Figure 3.3: An illustration by Tsai et al. [35], illustrating how very different images in appearance can have many similarities in their segmentation map.

Vu et al. [36] have two main contributions. Their observation is that models trained on source data tend to produce over-confident, i.e. low entropy, predictions on source-like images, and under-confident, i.e. high entropy, predictions on target-like ones. An illustration of this can be seen in Figure 3.4. The first is an unsupervised entropy loss applied to the output of the segmentation network, which directly punishes uncertainty in class predictions. The second contribution is an adversarial entropy loss, which is an adversarial approach that uses a

domain discriminator, but in this case on the *weighted self-information space*, which is the space from which an entropy map can be obtained by pixel-wise aggregate. The logic here is that, since entropy is naturally very low on source outputs, an adversarial discriminator pushing source and target entropy maps to be similar will result in a reduction of entropy for target predictions, and thus improve the predictive power on target data. These two losses are used alongside the standard segmentation loss. Finally, in a fashion similar to FCNs in the Wild [14], they use a loss based on the distribution of class labels in the source data.

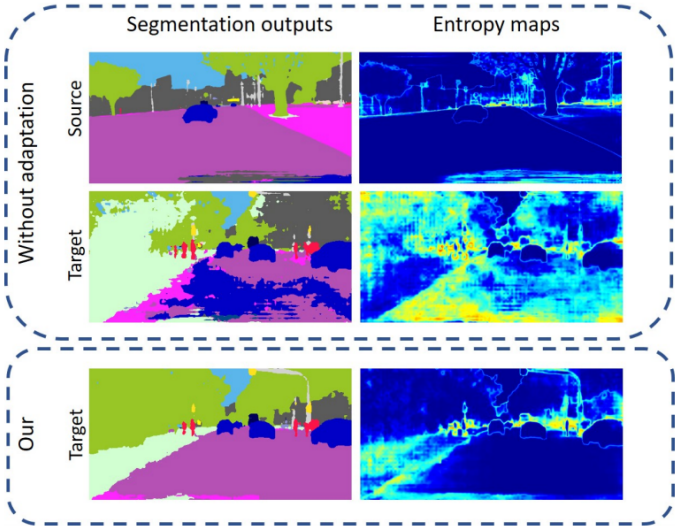


Figure 3.4: An example by Vu et al. [36] of how source and target images differ in entropy without adaptation, and how their entropy-based adaptation improves the segmentation.

3.1.2 Generative Adversarial Approaches

Whilst not a strict division, we have placed in this subsection approaches that generate or reconstruct images in the input space to assist in the domain adaptation task.

Hoffman et al. [13] present Cycle-Consistent Adversarial Domain Adaptation (CyCADA), adapting both pixel-level (see Figure 3.5) and feature-level representations. For pixel-level adaptation, they use CycleGAN, but present an additional loss to encourage semantic consistency before and after image translation. They do this by first training a task model on the source data and then encouraging an image to be classified in the same way after translation. Unfortunately, they do not apply this semantic loss in their experiments with semantic segmentation due to a lack of sufficient GPU memory. The feature-level adaptation done in CyCADA resembles the adversarial method for achieving domain-invariant features already discussed.

Figure 3.5: An example of GTA V (left) → Cityscapes (right) pixel-level adaptation using CycleGAN as presented in CyCADA [13].

Sankaranarayanan et al. [30] present another adversarial approach using a generator that recreates the original image from the feature space, and forces the segmentation network to learn features such that source features produce target-like images when passed to the reconstruction generator. Specifically, along with a base network for semantic segmentation, consisting of a feature extractor and a pixel-wise classifier, they have a generator and a discriminator. As seen in Figure 3.6, the generator G takes as input features from the feature extractor F and reconstructs the RGB image. The discriminator D, taking RGB images as input, is trained to classify between the four classes *real source*, *fake source*, *real target*, and *fake target*, and for training stabilisation purposes additionally performs an auxiliary semantic segmentation task for the source domain similar to the base network. The generator is trained to fool the discriminator into thinking *fake* images are *real*, along with a simpler pixel-wise reconstruction loss. The critical part then is that the feature extractor is trained to produce features that, on top of being good for semantic segmentation, trick the discriminator into thinking the generated *fake source* images are *real target* images, and that the

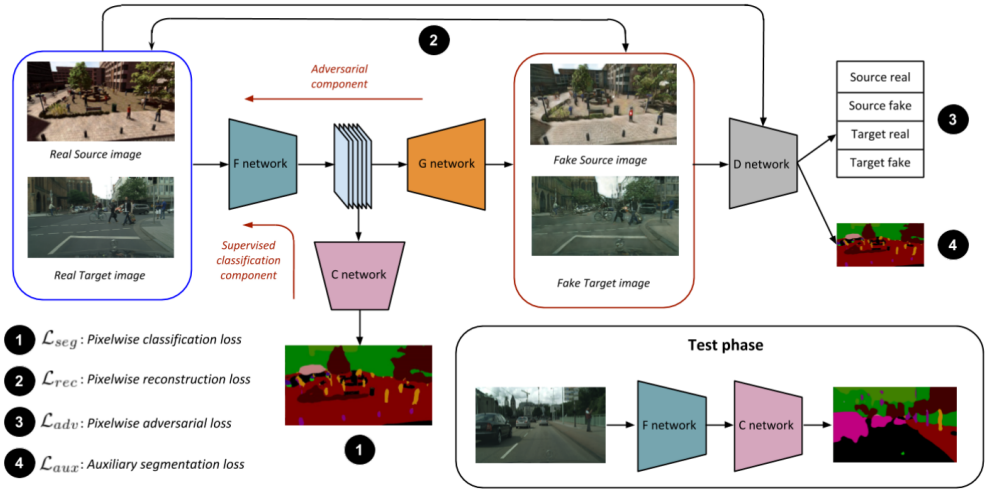


Figure 3.6: An illustration of the model described by Sankaranarayanan et al. [30].

generated *fake target* images are *real source* images. Thus, there is an adversarial game both between the generator and the discriminator, but also between the generator-discriminator pair and the feature extractor. In this way the source and target distributions are trained to be aligned in the feature space.

Murez et al. [26] combine several networks and losses to achieve domain-invariant features, arguing that making the source and target features indistinguishable as judged by a discriminator does not alone give a strong enough constraint for domain adaptation knowledge transfer, as there exist many mappings that could match the source and target distributions in the shared space. Following Figure 3.7 we will discuss these losses in order from top left to bottom right. Firstly, they have an encoder for each domain, mapping the images to a shared feature space, and a classifier that maps this feature space to an annotation. They then train the source encoder along with the classifier to perform the classification task, which would be pixel-wise classification in the case of semantic segmentation, using the source images and labels. Secondly, they have two decoders that should be able to reconstruct source and target images, respectively, from the shared feature space. Thirdly, like many of the other works, they have an adversarial loss to force the features originating from each domain to be indistinguishable by a discriminator trained to distinguish between source and target features. Fourthly, if the feature map originating from one domain is "reconstructed" to the other domain, the fake reconstruction should fool a discriminator trained to distinguish between real images from that domain and these fake reconstructions. Fifthly, to enforce that semantically similar images in both domains are projected into close vicinity of one another in the feature space, a cycle-consistency loss is added. This enforces that translating an image into the feature space, "reconstructing" it into the other domain, translating it back into the feature space, and finally reconstructing it back to its original domain, should be an identity mapping.

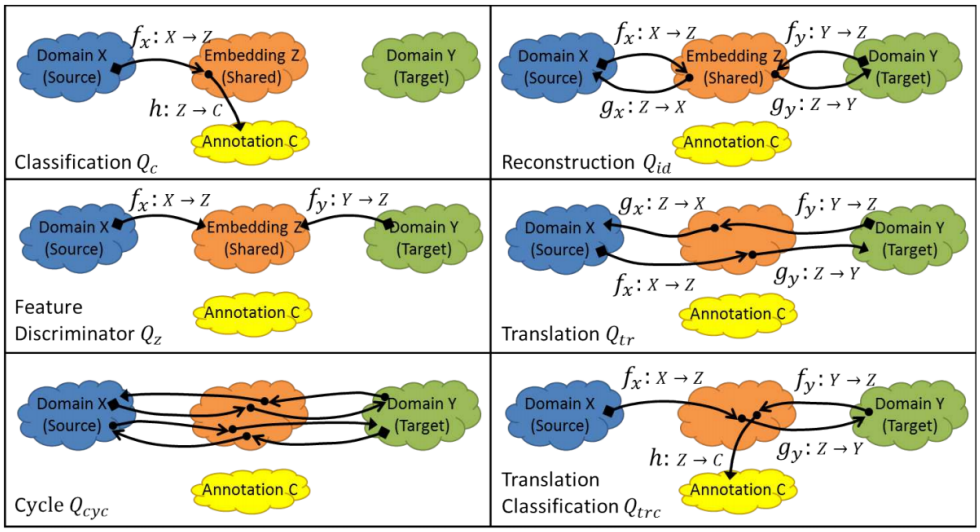


Figure 3.7: An illustration of the losses defined by Murez et al. [26].

Sixthly, to allow the target encoder to be trained with supervision on target-like images, a classification loss between source-to-target translations and the original source labels is added.

3.2 Non-Adversarial Approaches

In this section we introduce some works that do not make use of the predominant approach of using adversarial training.

Zhang et al. [42] present a curriculum-style domain adaptation approach to semantic segmentation, following an intuition that the assumption of source and target data sharing the same prediction function in a domain-invariant feature space may compromise the accuracy too much. Curriculum learning [1], meaning solving easy tasks first in order to infer some necessary properties about the target domain, is inspired by the way humans and animals learn better when examples are not randomly presented but organised in a meaningful order which gradually illustrates more concepts, and gradually more complex ones. The *easy* task in this case is predicting the label distribution for a given target image, with the argument that estimating this is less challenging than generating per-pixel predictions, despite both tasks being influenced by the domain mismatch. They predict the label distribution, using models trained on the source images, both globally, for the whole image, and locally, for each superpixel. They use an existing superpixel segmentation algorithm [19] to segment the image into these superpixels, i.e. groups of connected pixels with some degree of similarity. Using this estimated information about the label distribution in the target image, they regularise the predictions for the *hard* task, being semantic segmentation, such that they follow these estimated label distributions.

Zou et al. [45] utilises self-training for unsupervised domain adaptation for semantic segmentation. To put it briefly, they alternately generate a set of pseudo-labels corresponding to confident predictions in the target domain, and then fine-tune the network model based on these pseudo-labels together with the labelled source data, implicitly assuming that target samples with more confident predictions tend to have higher prediction accuracies. They take into account that some classes are easier to predict than others, and balance accordingly. They further utilise spatial prior knowledge, i.e., regularise the prediction based on per-class heat maps constructed from the source domain, thus utilising the knowledge that, for example, the sky is not likely to appear at the bottom of an image.

Chapter 4

Methods

Through our analysis of the related work, we note several things. Firstly, the most prevailing tactic in the area of unsupervised domain adaptation for semantic segmentation of street scenes is an adversarial approach using a discriminator. Secondly, where and how to perform this adversarial adaptation varies in the literature. To provide a structure, upon which we can base our decisions regarding which approaches to investigate and combine, we categorise the various approaches into the three different levels that the discriminator can operate on.

Firstly, the discriminator might differentiate between feature maps of the two domains, in which case we refer to it as a *feature-level* adversarial approach. Secondly, the discriminator might differentiate between outputs, or information extracted from the outputs, of the two domains, in which case we refer to it as an *output-level* adversarial approach. Finally, the discriminator might be used as part of a network designed to decrease the domain difference in the images before they are used for training the semantic segmentation network, in which case we refer to it as an *input-level* adversarial approach.

In this chapter we present our implementations of some of the key ideas in several works that perform adaptation in these three different ways. After an introduction to the data handling, and the semantic segmentation network we use, we begin by presenting our implementations of the feature-space adversarial adaptation approaches, as this is the approach used in the seminal papers in this area. We then move on to our presentation of the approaches that work on the output space, and then those that work on the input space. After introducing our implementations for all three approaches, we look at potential combinations across approaches. We look at whether combining feature, output, and input adversarial approaches leads to better results, or not.

Besides the CycleGAN network used in Section 4.5 and parts of the VGG16-FCN8s used as the base semantic segmentation network, all of the models, including data loading and augmentation, training, and evaluation, are implemented by ourselves using PyTorch 1.4.0[1]. For training we use a graphics card

[1] https://pytorch.org/

with 8 GB VRAM, which at the time of writing is high for a consumer graphics card. Given the large size of the models, the implementations and subsequent experiments are designed with this memory limit in mind. The number of experiments are also limited by the training time often being between 12 and 36 hours for feature and output-level adaptation, with input-level adaptation taking around 96 hours.

When attempting to implement some of the approaches, we found that important implementation details were sometimes missing, and sometimes even contradictory. Thus, as a result, we in some cases perform our own experimentation on top of the available information from the papers, and this is explained in detail in the coming sections. Before moving to the adaptation methods, we begin with relevant information about the datasets, and the base segmentation network.

In this chapter, we will use the short-hand notation shown in Table 4.1 for the various parameters of the layers in the networks.

`f`	Filters (no. of channels of output)
`k`	Kernel size
`s`	Stride
`p`	Padding
`op`	Output padding
`pr`	Probability (for dropout layers)
`out`	No. of features of output (for linear layers)

Table 4.1: Short-hand notation for the various parameters of the layers in the networks.

4.1 Datasets

This thesis focuses on the Cityscapes and Playing for Data datasets, introduced in Section 2.2.1 and Section 2.3.1 respectively, both consisting of images of street scenes. Cityscapes consists of images from German cities, while Playing for Data consists of screenshots from GTA V, specifically from a virtual city therein designed to be similar to Los Angeles, USA. We use the Playing for Data dataset as our source domain and the Cityscapes dataset as our target domain. Information about data augmentation, dataset partitioning, and more can be seen in Section 5.1.

4.2 Base Semantic Segmentation Network

For the base semantic segmentation network, we utilised the VGG16-FCN8s network, which is a fully convolutionalised version of the VGG16 network. Our motivation for using VGG16-FCN8s is that it is a seminal semantic segmentation network, and is commonly used in the literature in this area. Whilst there are more modern segmentation networks available that often give better results, this

is not too important for the investigations conducted in this thesis, as we are more interested in the difference of segmentation quality *between* domain adaptation techniques, as opposed to the absolute quality of the segmentation in general.

The VGG16-FCN8s network takes the VGG16 network and converts the last three fully connected layers into convolutional layers, resulting in the structure shown in Table 4.2. Take good note of the output names specified in this table, as we will use these to refer to the specified feature maps in the remainder of this chapter. A VGG16 network pre-trained on ImageNet [7] is used, so a bit of care has to be taken to correctly load the weights into the new convolutional layers.

Type	Parameters	Followed by	Output name
Conv2d	f=64, k=3, s=1, p=1	ReLU	
Conv2d	f=64, k=3, s=1, p=1	ReLU, MaxPool2d(k=2, s=2)	`pool1`
Conv2d	f=128, k=3, s=1, p=1	ReLU	
Conv2d	f=128, k=3, s=1, p=1	ReLU, MaxPool2d(k=2, s=2)	`pool2`
Conv2d	f=256, k=3, s=1, p=1	ReLU	
Conv2d	f=256, k=3, s=1, p=1	ReLU	
Conv2d	f=256, k=3, s=1, p=1	ReLU, MaxPool2d(k=2, s=2)	`pool3`
Conv2d	f=512, k=3, s=1, p=1	ReLU	
Conv2d	f=512, k=3, s=1, p=1	ReLU	
Conv2d	f=512, k=3, s=1, p=1	ReLU, MaxPool2d(k=2, s=2)	`pool4`
Conv2d	f=512, k=3, s=1, p=1	ReLU	
Conv2d	f=512, k=3, s=1, p=1	ReLU	
Conv2d	f=512, k=3, s=1, p=1	ReLU, MaxPool2d(k=2, s=2)	`pool5`
Conv2d	f=4096, k=7, s=1, p=0	ReLU, Dropout(pr=0.5)	
Conv2d	f=4096, k=1, s=1, p=0	ReLU, Dropout(pr=0.5)	`last_ft`
Conv2d	f=19, k=1, s=1, p=0		`score`

Table 4.2: VGG16 structure with fully connected layers replaced with convolutional layers.

As shown in Figure 4.1, the final segmentation map is predicted from three different stages of the network. By combining fine and coarse layers, both location and semantic information are incorporated into the final prediction. The upsampling is done as follows:

1. The `score` output is upsampled using a `ConvTranspose2d(f=19,k=4,s=2)` layer.

2. The `pool4` output is passed through a `Conv2d(f=19,k=1)` layer, cropped to match the size of the upsampled `score` output, and added to this. The result of this is then upsampled using another `ConvTranspose2d(f=19,k=4,s=2)` layer.

3. The `pool3` output is passed through a `Conv2d(f=19,k=1)` layer, cropped to match the size of the result of step 2, and added to this. The result of this is then upsampled using a `ConvTranspose2d(f=19,k=16,s=8)` layer and finally cropped to obtain the prediction.

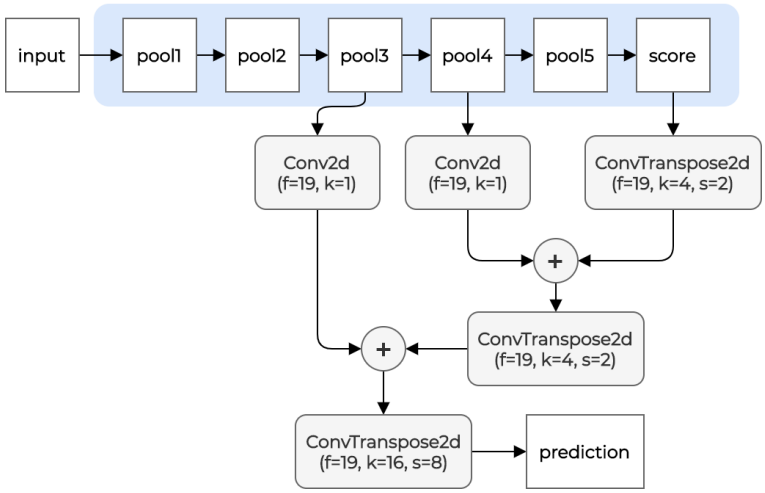


Figure 4.1: How different stages of the VGG16 network are combined to obtain a final prediction.

The network thus takes an $H \times W$ RGB image and produces an $H \times W \times 19$ (the number of classes) output, such that, after applying the softmax function on each pixel, we get a prediction of the probability of that pixel belonging to each class. To obtain a final segmentation map, the most probable class for each pixel is then chosen using the argmax function.

The network is trained to do semantic segmentation using the corresponding label and the cross-entropy loss. Specifically, for a pixel ij, we let $x_{ij} \in [0,1]^{19}$ be the softmaxed predictions of each class for that pixel, $y_{ij} \in \{1, \ldots, 19\}$ be the correct class for that pixel as given by the label, and n be the total number of pixels. The loss is then defined as follows:

$$\mathcal{L}_{sem} = \frac{1}{n} \sum_{i,j} -\log(x_{ij}[y_{ij}])$$

Minimizing this loss thus corresponds to predicting the correct class for each pixel as confidently as possible.

We note that the weights of the `ConvTranspose2d` layers are initialised such that they perform bilinear upsampling. The authors state that, in their experiments, these layers could be frozen for a slight speed-up without any significant difference in accuracy. To be on the safe side, we choose to keep these layers trainable.

The staged version of training the VGG16-FCN8s network includes the related networks VGG16-FCN32s and VGG16-FCN16s, which have zero and one skip layers respectively, as opposed to the two skip layers of VGG16-FCN8s. Firstly the VGG16-FCN32s is trained, is then upgraded to VGG16-FCN16s and trained, and then finally upgraded to VGG16-FCN8s and trained, dropping the learning rate by a factor of 100 in each stage, which the authors found necessary for continued improvements. However, the authors found that the VGG16-FCN8s

network could also be trained *all-at-once* with nearly equivalent results, as long as each skip layer is scaled by a fixed constant corresponding to the staged learning rate adjustments, with the purpose of producing a similar effect. Specifically the `pool3` output is multiplied by 0.0001 and the `pool4` output multiplied by 0.01.

During our preliminary experiments, however, we found that the network generated quite coarse predictions. Since this imprecision might signify that not enough location data is retrieved from the skip layers, we did tests with an adjustment where we would omit scaling the skip layers, despite training the network *all-at-once*. Since these preliminary experiments showed that this adjustment increased segmentation performance by a noticeable margin, we opted to keep this adjustment.

To establish a baseline result, where no domain adaptation techniques are used, we train the network on GTA V images and evaluate it on Cityscapes images. We also train an *oracle* model, which means a model that is allowed to see the ground-truth Cityscapes labels that we otherwise assume do not exist, to gain a reference point for the accuracy to strive for. The results for both the baseline and the oracle experiments can be seen in Section 5.2.

4.3 Feature-Level Adaptation Approaches

4.3.1 Feature Map Discriminator

The first domain adaptation method we experiment with is based on the *Global Domain Alignment* presented by Hoffman et al. [14]. This consists of adding a discriminator to the semantic segmentation network during training, taking as an input the `last_ft` feature map (see Table 4.2).

The argument for this is as follows: During a regular training of the semantic segmentation network, the network learns to extract features suitable for segmenting GTA V images, before decoding and upsampling these features into a semantic segmentation map. However, upon sending a Cityscapes image through the network, there is no guarantee in this case that the resulting features are suitable or interpretable by the part of the network doing the decoding and upsampling. If, however, it were possible to ensure that the features extracted by the encoder for GTA V and Cityscapes images were similar, the subsequent part of the network would now learn to decode and upsample from this shared domain-invariant feature space. This would therefore mean that the network would work effectively for both GTA V and Cityscapes images, despite only being trained on the former. We task the discriminator with differentiating features produced by Cityscapes and GTA V images. To then encourage a domain-invariant feature space, the semantic segmentation network is now required to extract features that fool the discriminator, alongside its regular semantic segmentation task. This is the core idea upon which most of the approaches experimented with in this thesis build upon.

Hoffman et al. [14] indicate that since recognition is desired at the pixel-level, applying the discriminator to the entire feature map may limit the alignment capability of the adversarial approach, and instead they apply the discriminator to each cell of the feature map individually. In this way, the discriminator is supplied with the same information that is used in the final prediction. In this case, using a linear classifier on each cell of the feature map output would be desirable. This can be effectively implemented using 1×1 convolutions and this is the approach we took. The structure can be seen in Table 4.3, and an image to help indicate which part of the VGG pipeline the discriminator is attached to can be seen in Figure 4.2. If $h \times w \times 4096$ is the size of the last feature map, the resulting output of the discriminator will be $h \times w \times 1$, where each cell in the grid is the confidence that the corresponding cell of the feature map is from a target image.

Type	Parameters	Followed by
Conv2d	f=1024, k=1, s=1	Dropout(pr=0.5), ReLU
Conv2d	f=512, k=1, s=1	Dropout(pr=0.5), ReLU
Conv2d	f=1, k=1, s=1	

Table 4.3: Discriminator used on the last feature map, tasked with differentiating between GTA V and Cityscapes features.

The model is trained end-to-end, with the forward pass generating, for a given source and target image, a source and target feature map for use in the discriminator, and a source semantic segmentation prediction for use in generating the regular task loss. When implementing the adversarial loss, a question arises, namely whether the network should encourage a shared feature space by making the discriminator believe source features are target features, target features are source features, or both. The predominant way in the literature appears to be to compute the adversarial loss by giving the discriminator target features and indicating that the correct answer was source, and this is thus how we implement it. For the discriminator and adversarial loss, binary cross-entropy loss is used.

Specifically, for the discriminator loss, if we let $D(x)_{ij} \in [0, 1]$ be the sigmoided element ij in the output of the discriminator D, and n be the number of elements in this output, we have:

$$\mathcal{L}_{disc} = \frac{1}{n} \sum_{i,j} -\log(1 - D(\text{feat}_{\text{source}})_{ij}) + \frac{1}{n} \sum_{i,j} -\log(D(\text{feat}_{\text{target}})_{ij})$$

The discriminator thus minimises this loss, which corresponds to predicting as low a score as possible for source features and as high a score as possible for target features. In an opposing fashion, the adversarial loss is then given by:

$$\mathcal{L}_{adv} = \frac{1}{n} \sum_{i,j} -\log(1 - D(\text{feat}_{\text{target}})_{ij})$$

Minimising this loss corresponds to fooling the discriminator into making it predict a low score for the target features, as if they were source features.

The semantic segmentation network then minimises:

$$\mathcal{L}_{sem} + \lambda_{adv} \cdot \mathcal{L}_{adv}$$

The λ_{adv} weight thus represents the trade-off between producing features that are good for semantic segmentation and features that fool the discriminator, and are therefore domain-agnostic. We experimented with different values of λ_{adv}, which can be seen in detail in Section 5.3.1.

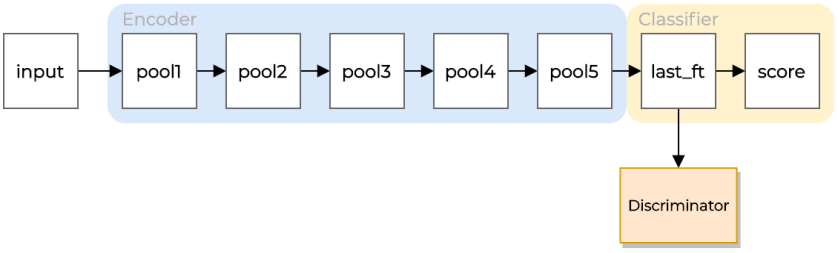


Figure 4.2: A diagram to indicate where the discriminator is placed.

4.3.1.1 A Larger Discriminator

Given the results obtained from implementing the above approach, we also implement a small adjustment. We add an extra layer to the discriminator and increase the number of filters, since we see that, for higher values of λ_{adv}, the discriminator loss quickly stagnates at $-\log(0.5)$, signifying uncertainty, and that perhaps a stronger discriminator could alleviate this issue. The larger discriminator can be seen in Table 4.4. The experiment results for this larger discriminator can be seen in Section 5.3.1.1.

Type	Parameters	Followed by
Conv2d	f=2048, k=1, s=1	Dropout(pr=0.5), ReLU
Conv2d	f=2048, k=1, s=1	Dropout(pr=0.5), ReLU
Conv2d	f=512, k=1, s=1	Dropout(pr=0.5), ReLU
Conv2d	f=1, k=1, s=1	

Table 4.4: Slightly larger discriminator.

4.3.1.2 Discriminator on pool5

Another slight change to the above described adversarial approach we present is to change the position of the discriminator, moving it to the `pool5` layer (see Table 4.2). The reason for this is twofold. Firstly, `pool5` is the feature map immediately prior to vectorisation in the original VGG16 network, and is thus the final feature map in the context in which the network is pre-trained. Secondly, another adaptation approach that we detail later in Section 4.3.2 places the discriminator here, and thus this result might function as a comparable baseline in the context of that approach. An illustration of this placement can be seen in Figure 4.3. Since the discriminator is now at a different place in the network,

the number of filters is adjusted slightly, with $f = 512$ and $f = 256$ in the first two layers (see Table 4.5). The results from this experiment can be seen in Section 5.3.1.2.

Type	Parameters	Followed by
Conv2d	f=512, k=1, s=1	Dropout(pr=0.5), ReLU
Conv2d	f=256, k=1, s=1	Dropout(pr=0.5), ReLU
Conv2d	f=1, k=1, s=1	

Table 4.5: Discriminator on `pool5`.

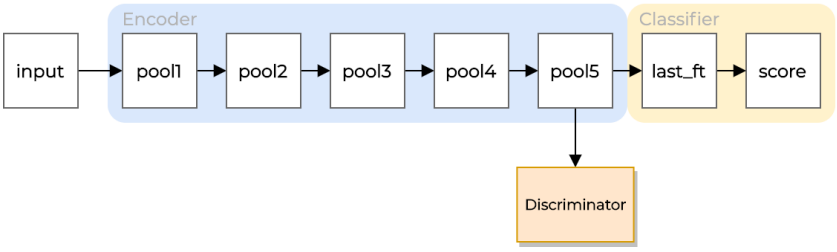


Figure 4.3: A diagram to indicate where the discriminator is placed for the implementation with a discriminator on `pool5`.

4.3.1.3 Three Discriminators

As seen previously in Figure 4.1, the `pool3`, `pool4` and `score` outputs are upscaled and summed to produce the semantic segmentation output, but only the feature map before `score` (`last_ft`) is passed to a discriminator. We suspect that better adaptation may be achieved by adding discriminators to all three places that directly contribute to the output. In this fashion, since different layers in the network extract different types of information from the image, we could potentially improve the adaptation by having specific discriminators that learn to differentiate source and target images at different granularity levels.

We thus add two more discriminators for earlier layers, one for `pool3` and one for `pool4`. These two discriminators have the same general structure as the discriminator on `last_ft` outlined in Table 4.3, but due to the smaller channel

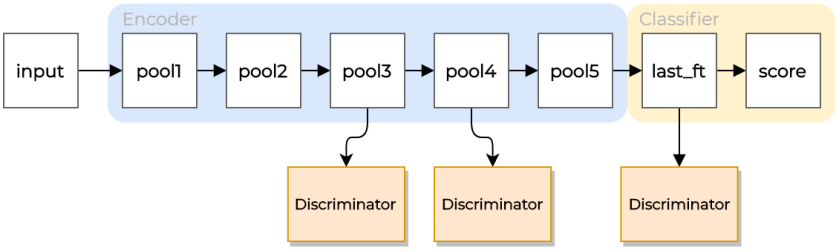


Figure 4.4: A diagram to indicate where the discriminators are placed for the implementation with three discriminators.

dimensions, they have an adjusted numbers of filters. The `pool4` discriminator has $f = 512$ followed by $f = 256$ in the first two convolutional layers, where the `pool3` discriminator has $f = 256$ followed by $f = 128$ in the first two convolutional layers. An image indicating the placement of the discriminators can be seen in Figure 4.4. The results from this experiment can be seen in Section 5.3.1.3.

4.3.2 Conditional Generator

The adversarial domain adaptation implementations outlined in Section 4.3.1 all rest on an assumption that a suitable feature space shared between the source and target domains exist. That is, the adversarial game between the encoder and the discriminator is used with the hope of emphasising the extraction of domain-invariant features to improve accuracy on the target data. Hong et al. [15] argue that this assumption comes with too big a cost in accuracy, and present a method with the aim of relaxing this assumption. They do this by introducing a conditional generator, whose purpose is to learn the residual between the feature maps from the two domains. While the target images pass through the encoder unaltered, for source images, the conditional generator takes as input the `pool1` features, and the generated output is added element-wise to the `pool5` features before being passed further to the classifier or discriminator. The purpose of this is to have the generator alter the source image features, so as to look like target image features. This means that the classifier is trained specifically on target-like features, and not just domain-invariant features. As usual, we have a discriminator differentiating between features of the two domains, and the generator learns to adapt the source image features such that the discriminator believes that they are target image features. We note that the generator is only used during training, and can be discarded thereafter.

For this approach, we will often refer to the layers in the VGG16-FCN8s network before the `pool5` output as the encoder, and the layers after as the classifier. The model is trained in a two-step fashion. The first step updates the classifier and the discriminator, and the second step updates the conditional generator and the encoder. Hong et al. state that, to ensure stability in the network, it helps to train the network using both adapted and non-adapted source images, and this is therefore also implemented. Our diagram of the network as presented in the paper can be seen in Figure 4.5.

Several variations on Hong et al.'s original network design are presented. This includes three main points of variation: the conditional generator, the discriminator, and variations regarding the training of the generator and the encoder. These variations are decided upon due to difficulties arising from inconsistencies in notation and a lack of implementation details for Hong et al.'s network. Each of the variations within these categories are combined with each variation within the other two categories, and for that reason we introduce all variations before discussing any experiments.

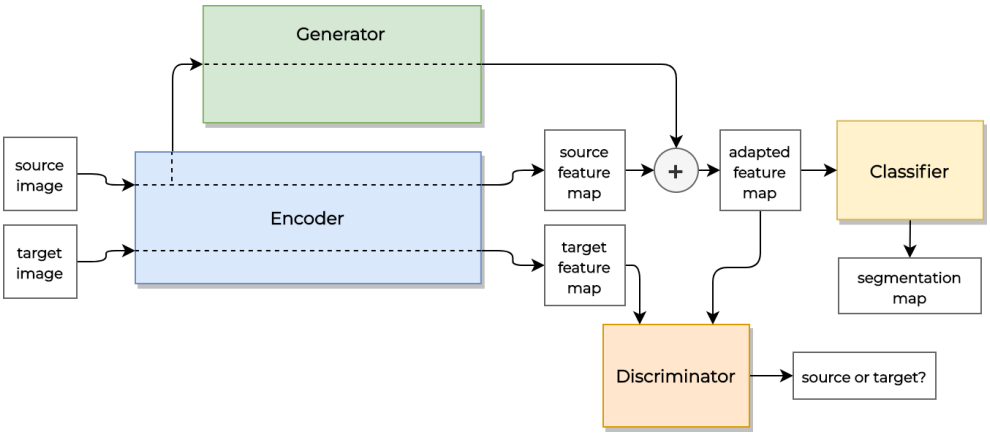


Figure 4.5: A diagram of the network using a conditional generator.

4.3.2.1 Generator as per Hong et al.

This first generator is an attempt at implementing the generator from Hong et al. as faithfully as possible. The input, taken from the encoder, is concatenated with a noise vector, and is then passed through a number of identical residual blocks, starting with a convolutional layer that alters the channel dimension after noise concatenation from 65 back to 64. For readers not familiar with the ResNet architecture [12], a residual block is a block of layers in which the input to the block is element-wise summed to the output, such that the layers of the block effectively predict the *residual*, i.e. the difference between the input and the final output of the block (see Figure 4.6). This helps deep networks learn faster by alleviating the vanishing gradients issue and by letting blocks easily learn the identity mapping if so desired.

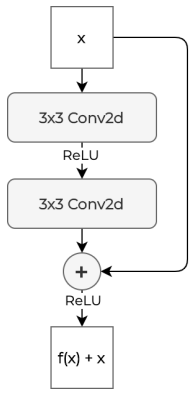


Figure 4.6: A sketch of a residual block.

Hong et al. use 16 residual blocks, as they find in their experiments that using more than 16 residual blocks gives only a minor improvement, and so when implementing the generator we also use 16 residual blocks. A detail missing in the paper is how the dimensions should change when being passed through the generator. Hong et al. state that the residual blocks are identical and all

have the same number of filters, 64, but if this is the case, then the generator's output cannot be element-wise summed with the `pool5` output from the encoder, which has 512 in the channel dimension. We solved this by making the final convolutional layer in the final block of the generator have 512 filters. In addition to this issue, the `pool5` feature map has a much smaller height and width due to the downsampling that occurs in the VGG16 encoder. While Hong et al. show in their diagram (which can be seen in Figure 3.2 in Chapter 3) that an average pooling layer is used at the end of the generator, this average pooling layer would have to have a kernel size and stride of 16 to make the dimensions fit. Despite the amount of information this would appear to be discarding, we implement it in this way to remain faithful to what is written in the paper. The full structure of the generator can be seen in Table 4.6.

Amount	Type	Parameters	Followed by
	Conv2d	f=64, k=3, s=1, p=1	
x15	Conv2d	f=64, k=3, s=1, p=1	BatchNorm, ReLU
	Conv2d	f=64, k=3, s=1, p=1	Skip connection
	Conv2d	f=64, k=3, s=1, p=1	BatchNorm, ReLU
	Conv2d	f=512, k=3, s=1, p=1	AvgPool2d(k=16,s=16)

Table 4.6: Our implementation of the generator provided by Hong et al. [15].

4.3.2.2 Generator with Downsampling

As mentioned above, the design of the generator presented by Hong et al. seemingly keeps the channel dimension, height, and width the same throughout the generator's layers. We have seen this design used before in generators where the output is the same dimension as the input, but this is not the case in this situation. Therefore, we believe it could be worth testing a generator design that is more directly inspired by the ResNet architecture, which progressively reduces the height and width by having a stride of 2 on convolutions in certain ResNet blocks, and also progressively increases the number of filters in these blocks. In

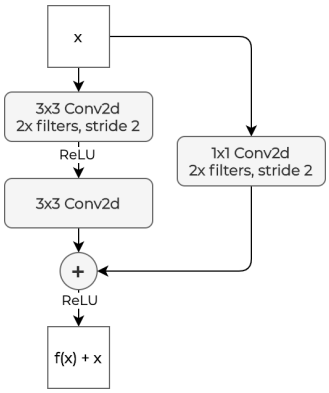


Figure 4.7: A sketch of a residual block with a 1×1 convolutional layer used in its skip connection.

this way, the output is gradually transformed into the desired shape. We still retain the noise vector, and the initial convolution that changes the channels from 65 to 64. For blocks with a stride of 2 and an increased number of filters, a 1×1 convolutional layer is used for the skip connection in order to match the dimensions (see Figure 4.7). The full network structure can be seen in Table 4.7.

Amount	Type	Parameters	Followed by
	Conv2d	f=64, k=3, s=1, p=1	
x4	Conv2d	f=64, k=3, s=1, p=1	BatchNorm, ReLU
	Conv2d	f=64, k=3, s=1, p=1	BatchNorm, Skip connection, ReLU
	Conv2d	f=128, k=3, s=2, p=1	BatchNorm, ReLU
	Conv2d	f=128, k=3, s=1, p=1	BatchNorm, Skip connection, ReLU
x3	Conv2d	f=128, k=3, s=1, p=1	BatchNorm, ReLU
	Conv2d	f=128, k=3, s=1, p=1	BatchNorm, Skip connection, ReLU
	Conv2d	f=256, k=3, s=2, p=1	BatchNorm, ReLU
	Conv2d	f=256, k=3, s=1, p=1	BatchNorm, Skip connection, ReLU
x3	Conv2d	f=256, k=3, s=1, p=1	BatchNorm, ReLU
	Conv2d	f=256, k=3, s=1, p=1	BatchNorm, Skip connection, ReLU
	Conv2d	f=512, k=3, s=2, p=1	BatchNorm, ReLU
	Conv2d	f=512, k=3, s=1, p=1	BatchNorm, Skip connection, ReLU
x3	Conv2d	f=512, k=3, s=1, p=1	BatchNorm, ReLU
	Conv2d	f=512, k=3, s=1, p=1	BatchNorm, Skip connection, ReLU
	AvgPool2d	k=2, s=2	

Table 4.7: Our generator implementation with downsampling.

4.3.2.3 Fully Connected Discriminator

The discriminator used in the adaptation presented by Hong et al. is a classifier consisting of three fully connected layers, with output sizes 4096, 1024, and 1 respectively. The input, which is the (possibly adapted) `pool5` output, is thus vectorised before being passed to this discriminator. When attempting to implement and use this discriminator, we unfortunately run out of VRAM, as this design results in a huge number of parameters. For us to be able to run this discriminator, it is therefore necessary to make some changes to either the discriminator itself, or the size of the input to the discriminator, which is dependent on the image crop size. We found that the crop size of the image would have to be reduced from 384, which we used in previous experiments, down to a very small crop size of around 64 to not hit the VRAM limit, so we choose to keep the crop size the same, and instead add an average pooling

Type	Parameters
AvgPool2d	k=2, s=2
Linear	out=1024
Linear	out=512
Linear	out=1

Table 4.8: An adjusted discriminator, based on the one presented by Hong et al. and adjusted due to VRAM limitations.

layer before the vectorisation to reduce the input size of the first layer of the discriminator. In addition to this, it is also necessary to reduce the output size of the layers. Instead of 4096 and 1024 for the first 2 layers, we use 1024 and 512. The full discriminator structure can be seen in Table 4.8.

4.3.2.4 Cell-Based Discriminator

Another discriminator we attempt in this adversarial approach is the same discriminator first mentioned in Section 4.3.1. We test the use of this discriminator for a couple of reasons. Firstly, since several changes are necessary to make the discriminator presented by Hong et al. work, we believe it would be a good idea to also test another discriminator. Secondly, since this discriminator works by discriminating each cell of the feature map separately, elaborated on in Section 4.3.1, it is possible that the benefits of this discriminator claimed by Hoffman et al. could be relevant in this adversarial approach as well. Since the discriminator in this adversarial approach is operating on the `pool5` output, the structure of this discriminator is the same as in Table 4.5.

4.3.2.5 Training Variations

As mentioned, Hong et al. present the training as being done in a two-step fashion. The first step is to update the classifier based on the semantic segmentation task loss, and to update the discriminator using the usual discriminator loss. The rest of the network remains fixed in this step. During the second step the classifier and discriminator are fixed, and the rest of the VGG16 network (the encoder) and the conditional generator are updated. What is unclear in the paper, due to inconsistencies in terminology, is which combinations of losses are used to update the encoder and the generator in this step. Due to the element-wise sum, the features passed to the discriminator are dependent on both the generator and the encoder. Because of this, both the conditional generator and the segmentation network could be chosen to be updated based on the semantic segmentation loss, as well as the adversarial loss. And thus comes the variations we experimented with. We test three plausible variations for parameter update in this step.

The first variation is where the semantic segmentation loss is used solely to update the encoder, and the adversarial loss is used solely to update the generator. We suspected that this way to train the network was the most likely to provide good results. We believe it makes most sense to have the generator only focusing on adapting the feature map to trick the discriminator, and the encoder only focusing on extracting features for segmentation. We call this training method *Disjoint Training*, and a diagram showing the path of the gradients used to update in this fashion can be seen in Figure 4.8. We use a red arrow to indicate the path of the gradients used to update the model with the adversarial loss, and a blue arrow to indicate the path of the gradients used to update the model with the semantic segmentation loss. The translucent parts of the arrows indicate that gradients are calculated for the purpose of backpropagation, but the weights for that part of the network are not updated in this step.

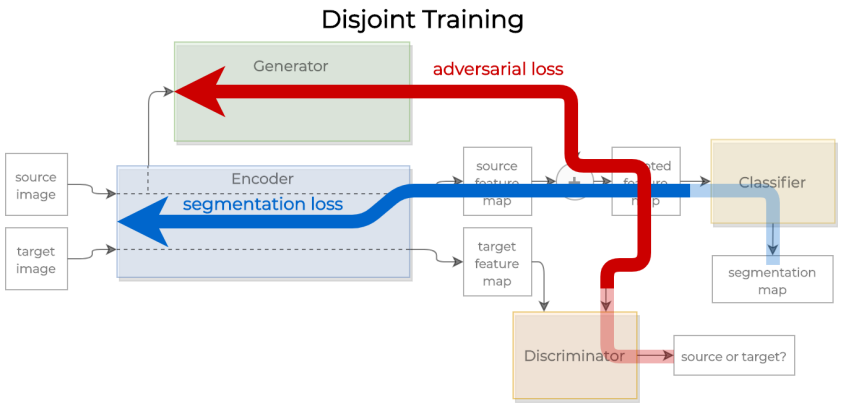


Figure 4.8: A diagram of the *Disjoint Training* training mode.

The second variation is where both the semantic segmentation loss and the adversarial loss are used to update the conditional generator, but the encoder is only updated using the semantic segmentation loss. This training mode still prevents the encoder having to balance two different objectives, and adding the semantic segmentation loss to the conditional generator could help stabilise the training process. We call this training method *Generator Learns Both*, and a diagram showing this training method can be seen in Figure 4.9.

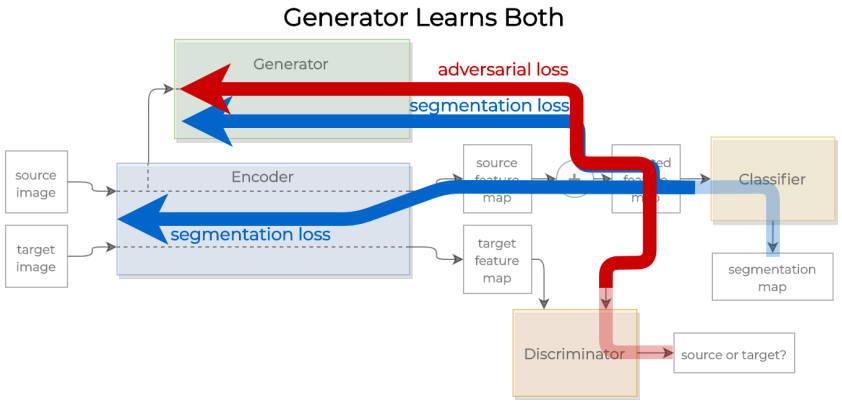


Figure 4.9: A diagram of the *Generator Learns Both* training mode.

The final variation is where both the conditional generator and the encoder are updated using both the task loss and the adversarial loss. While we suspect that this training mode would miss out on some of the beneficial effects of including the conditional generator, as the encoder is now back to finding a compromise between two tasks, it is a valid interpretation of the training process described by Hong et al., and we thus included it in our experiments. We call this training method *Both Learn Both*, and a diagram showing this training method can be seen in Figure 4.10.

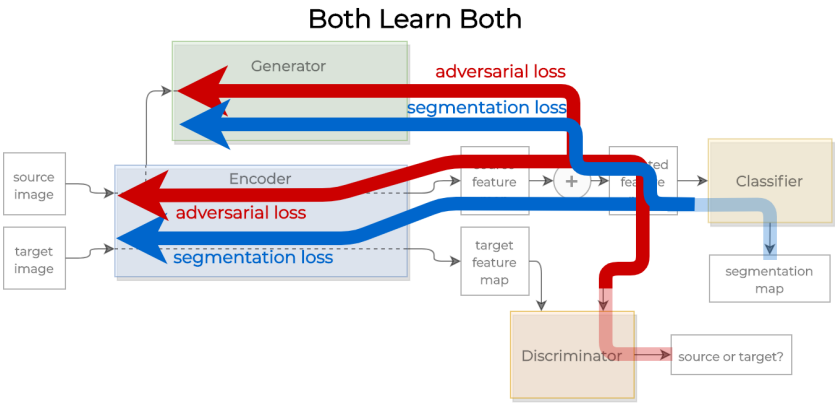


Figure 4.10: A diagram of the *Both Learn Both* training mode.

The results and details of the experiments with the all the above variations can be seen in Section 5.3.2.

4.3.2.6 Full Training

After experimenting with the above variations, we found that the most effective combination was to use the cell-based discriminator, the generator presented by Hong et al., and to use the training method we entitled *Disjoint Training*. This combination was then fully trained and the results from this experiment can be seen in Section 5.3.2.

4.4 Output-Level Adaptation Approaches

4.4.1 Direct Output Alignment

The first output-level adaptation approach we investigate is the direct output alignment approach presented by Tsai et al. [35]. As discussed in Chapter 3, Tsai et al. notice that, while source and target images can differ a lot, their ground truth segmentation maps can still have many similarities, as shown previously in Figure 3.3. They argue therefore that, regardless of whether an image is from the source domain, or the target domain, the generated segmentation maps should share strong similarities as well. The solution they therefore present is to add a discriminator on the output of the semantic segmentation network (prior to applying the argmax operation).

The model is trained end-to-end, with the forward pass generating a semantic segmentation prediction for both source and target images. As usual, the source semantic segmentation is used for generating the regular semantic segmentation task loss. For the adaptation, the semantic segmentation predictions are passed to the discriminator, which attempts to predict whether an output came from a source or target image. Additionally, an adversarial loss is calculated by showing the discriminator a target output, and indicating that the correct answer was source. The gradients from this loss are propagated backwards through

the entire semantic segmentation network. In contrast to the feature-level approaches operating on outputs of small height and width, it makes sense to use convolutional layers for the discriminator in the output-level approaches as it is operating on image-like outputs. The discriminator used by Tsai et al. consists of five convolutional layers, each followed by a leaky ReLU. This is also the discriminator used by Vu et al. in the approach discussed in the next subsection, and seems to originate from Radford et al. in their DCGAN network [28]. The full discriminator structure can be seen in Table 4.9. Additionally, a diagram showing the placement of the discriminator in our implementation of this approach can be seen in Figure 4.11.

Type	Parameters	Followed by
Conv2d	f=64, k=4, s=2, p=1	LeakyReLU(negative_slope=0.2)
Conv2d	f=128, k=4, s=2, p=1	LeakyReLU(negative_slope=0.2)
Conv2d	f=256, k=4, s=2, p=1	LeakyReLU(negative_slope=0.2)
Conv2d	f=512, k=4, s=2, p=1	LeakyReLU(negative_slope=0.2)
Conv2d	f=1, k=4, s=2, p=1	

Table 4.9: The discriminator used for the output-level approaches presented in this chapter.

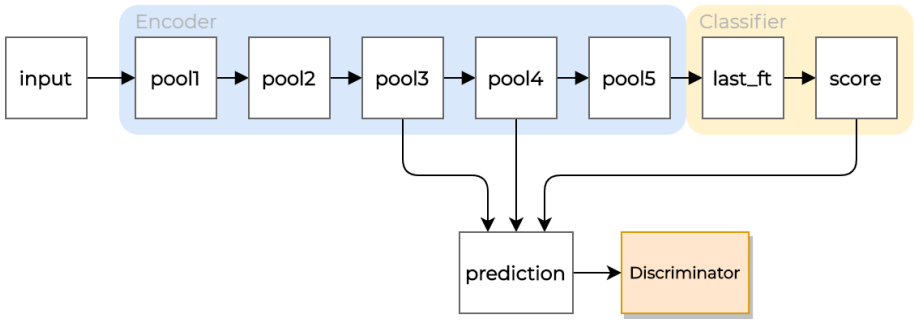


Figure 4.11: A diagram indicating the placement of the discriminator in the network by Tsai et al. [35].

We implemented the above network as described, and the results for the experiments can be seen in Section 5.4.1.

4.4.2 Output Entropy Alignment

The second output-level approach we discuss is the adversarial entropy alignment approach (AdvEnt) presented by Vu et al. [36]. As mentioned in Chapter 3, the authors observed that if a model is trained on source data, it will generate confident, i.e. low-entropy, predictions on source-like images, and uncertain, i.e. high-entropy, predictions on target-like images. An illustration of this was shown previously in Figure 3.4. One of the methods presented by Vu et al. to solve this is to add a domain discriminator on a modification of the output space, specifically on the *weighted self-information space*. This is the space from which

an entropy map, i.e. the confidence of the prediction for each pixel, can be obtained by pixel-wise aggregate. This weighted self-information map is easily obtained by replacing each probability p of a given pixel belonging to a given class with the value $-p \cdot \log_c(p)$, with $c = 19$ being the number of possible classes. The weighted self-information maps are passed to the discriminator, which has to determine whether the maps came from a source, or target, prediction. As usual, an adversarial loss is calculated by showing the discriminator a weighted self-information map from a target image, and indicating that it came from a source image. The gradients from this loss are propagated backwards through the entire semantic segmentation network. The idea is that by forcing the source and target images to have similar weighted self-information maps, the entropy of the target predictions will decrease, and semantic segmentation performance on the target data will therefore improve. The discriminator used is the same as for the implementation outlined in Section 4.4.1, and thus the full discriminator structure can be seen in Table 4.9. A diagram of the network showing the placement of the discriminator in this implementation can be seen in Figure 4.12. We implemented the above network as described, and the results for the experiments can be seen in Section 5.4.2.

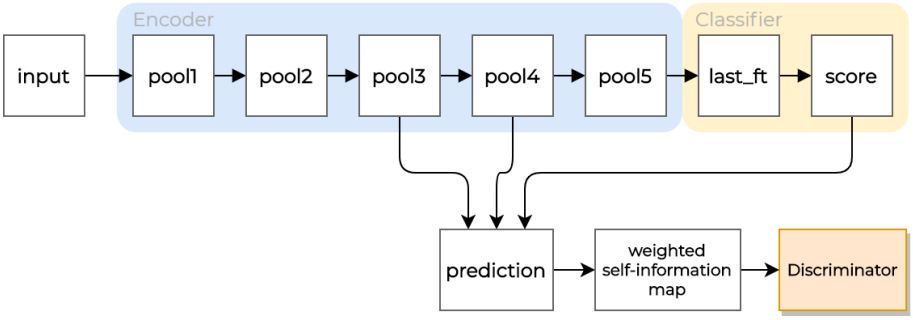


Figure 4.12: A diagram indicating the placement of the discriminator in the network by Vu et al. [36].

4.5 Input-Level Adaptation Approaches

4.5.1 CycleGAN

The input-level approaches looked at in this thesis differ from the feature-level and output-level approaches, as they function by altering the source dataset itself. We thus want a model that is able to translate source images into target-like ones, while keeping enough semantic information for the labels to still be valid. While the approach we use for this is presented by Hoffman et al. [13], it is essentially an unaltered use of the CycleGAN network as implemented by Zhu et al. [44], and we will thus discuss the implementation found in the first-party code for CycleGAN[2].

[2] https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix

The CycleGAN model, touched on in Section 2.4.1, learns to transform an image from one domain to another domain using an adversarial loss, retaining the semantic information by the addition of a cycle consistency loss. The cycle consistency loss expresses that, when transforming an image from one domain to another, it should be transformed back to the original image again by the opposite transformation.

Specifically, given two image domains, X and Y, CycleGAN is a network that consists of two mapping functions (generators), $G : X \mapsto Y$ and $F : Y \mapsto X$, as well as two discriminators D_X and D_Y, each tasked with determining whether an image in the style of their domain is real or generated from the other domain. Using D_Y, an adversarial loss is generated, encouraging G to translate images from X into images indistinguishable from images of domain Y, and vice versa for D_X and F. The previously discussed cycle consistency loss ensures that upon translating from one domain to the other, and then back again, we should arrive where we started. Which is to say $F(G(x)) \approx x$ and that $G(F(y)) \approx y$. In addition to the cycle consistency loss, and the regular adversarial and discriminator losses, there is another, regularising, loss, called the identity loss. This loss expresses that when an image from domain X is given to the generator F, that maps from Y to X, then the resulting image should be unaltered, as the image is already in domain X, and likewise for Y and G. Our diagram of the full network can be seen in Figure 4.13.

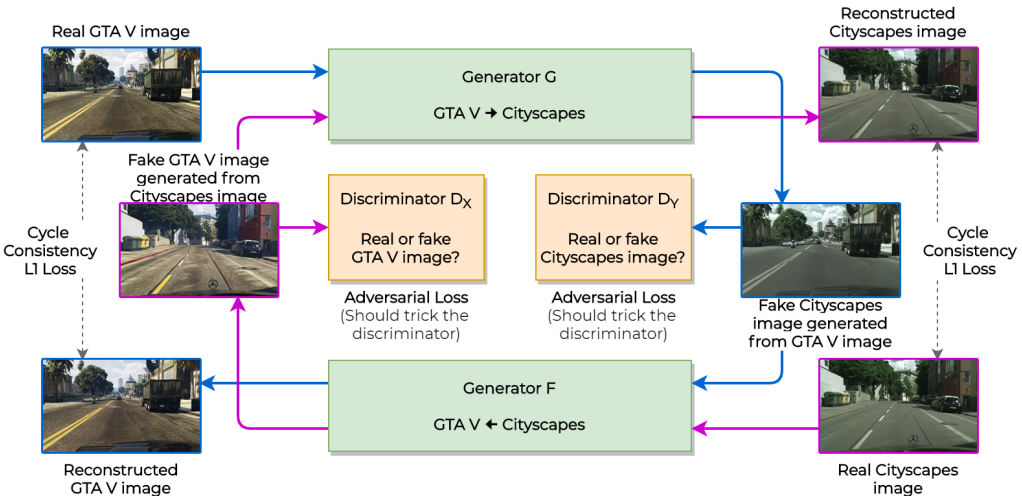


Figure 4.13: The CycleGAN network, with the identity loss omitted for clarity.

The two generators G and F have the same structure, and are implemented based on the ResNet architecture [12], using nine residual blocks. The generator network structure can be seen in Table 4.10. Interestingly, there is no ReLU activation at the end of the residual blocks, the lack of which supposedly leads to a small improvement[3].

[3] http://torch.ch/blog/2016/02/04/resnets.html

Amount	Type	Parameters	Followed by
	Conv2d	f=64, k=7, s=1, p=3	InstanceNorm2d, ReLU
	Conv2d	f=128, k=3, s=2, p=1	InstanceNorm2d, ReLU
	Conv2d	f=256, k=3, s=2, p=1	InstanceNorm2d, ReLU
9x	Conv2d	f=256, k=3, s=1, p=1	InstanceNorm2d, ReLU
	Conv2d	f=256, k=3, s=1, p=1	InstanceNorm2d, Skip conn.
	ConvTranspose2d	f=128, k=3, s=2, p=1, op=1	InstanceNorm2d, ReLU
	ConvTranspose2d	f=64, k=3, s=2, p=1, op=1	InstanceNorm2d, ReLU
	Conv2d	f=3, k=7, s=1, p=3	Tanh

Table 4.10: Generator architecture for both generators in CycleGAN [44].

Likewise, the two discriminators D_X and D_Y have the same structure. The discriminators are implemented as PatchGAN discriminators, as per Isola et al. [17]. This discriminator, instead of classifying the entire image, classifies several smaller patches in the image. In particular, a 70x70 PatchGAN discriminator is used, referring to the receptive field of each of its outputs. No sigmoid function is used after the last layer, as this implementation uses a mean squared error loss for the discriminator as proven effective by Mao et al. [25]. The structure can be seen in Table 4.11.

Type	Parameters	Followed by
Conv2d	f=64, k=4, s=2, p=1	LeakyReLU
Conv2d	f=128, k=4, s=2, p=1	InstanceNorm2d, LeakyReLU
Conv2d	f=256, k=4, s=2, p=1	InstanceNorm2d, LeakyReLU
Conv2d	f=512, k=4, s=1, p=1	InstanceNorm2d, LeakyReLU
Conv2d	f=1, k=4, s=1, p=1	

Table 4.11: Discriminator architecture for both discriminators in CycleGAN [44].

In order to concisely present the loss functions in detail, we slightly abuse matrix notation and let it be implicit that the operations are performed element-wise and that the final loss value is given by the mean of the resulting entries. We let 1_{M} and 0_{M} be, respectively, the matrix of ones and the matrix of zeroes with same shape as the output of the discriminator. For the two images X_{real} and Y_{real}, the loss for the discriminators D_Y and D_X is then given by using the mean squared error loss as follows:

$$\begin{aligned}\mathcal{L}_{disc} = &(D_Y(Y_{\text{real}}) - 1_{\mathrm{M}})^2 + (D_Y(G(X_{\text{real}})) - 0_{\mathrm{M}})^2 \\ &+ (D_X(X_{\text{real}}) - 1_{\mathrm{M}})^2 + (D_X(F(Y_{\text{real}})) - 0_{\mathrm{M}})^2\end{aligned}$$

Minimising the first column of terms here corresponds to predicting ones for real images, while minimising the second column of terms corresponds to predicting zeroes for generated images. For the generators $G : X \mapsto Y$ and $F : Y \mapsto X$, the adversarial loss is then, in opposing fashion, given by the mean squared error loss as follows:

$$\mathcal{L}_{adv} = (D_Y(G(X_{\text{real}})) - 1_{\mathrm{M}})^2 + (D_X(F(Y_{\text{real}})) - 1_{\mathrm{M}})^2$$

The generators minimise this loss by having the generated images fool the corresponding discriminator. Next, the cycle-consistency loss is given by the L1 loss as follows:

$$\mathcal{L}_{cc} = |F(G(X_{\text{real}})) - X_{\text{real}}| + |G(F(Y_{\text{real}})) - Y_{\text{real}}|$$

The generators minimise this loss by making sure that upon translating from one domain to the other, and then back again, the original image should be reconstructed. Finally, the identity loss is given by the L1 loss as follows:

$$\mathcal{L}_{idt} = |G(Y_{\text{real}}) - Y_{\text{real}}| + |F(X_{\text{real}}) - X_{\text{real}}|$$

The generators minimise this loss by making sure that when an image is mapped to its own domain, the resulting image should be unaltered. The final loss for the generators is then a combination of the above three losses:

$$\mathcal{L}_{adv} + \lambda_{cc} \cdot \mathcal{L}_{cc} + \lambda_{cc} \cdot \lambda_{idt} \cdot \mathcal{L}_{idt}$$

The weights λ_{cc} and λ_{idt} make it possible to adjust the trade-off between these different objectives, with the default values being $\lambda_{cc} = 10$ and $\lambda_{idt} = 0.5$.

Hoffman et al. thus perform input-level adaptation by training the CycleGAN model on the GTA V and Cityscapes datasets, and then using the model to generate a transformed GTA V dataset, that is now Cityscapes-like. They then train the semantic segmentation model on this transformed dataset. Examples from Hoffman et al. of GTA V images being Cityscapes-like after transformation can be seen previously in Figure 3.5.

In the same manner as Hoffman et al., we use the CycleGAN network unaltered for these experiments. All parameters for training the CycleGAN network are set to the default as provided by the authors of the CycleGAN network, apart from the image and the crop size, as well as the weight of the identity loss, which, following Hoffman et al., is increased from 0.5 to 1. After the CycleGAN network is fully trained on the GTA V and Cityscapes datasets, we use the generator that is trained to transform GTA V images into Cityscapes images to build a new, transformed, dataset. Then, we train the semantic segmentation network with no further domain adaptation techniques, but with this transformed GTA V dataset. Further details and results for this experiment can be seen in Section 5.5.1.

4.5.2 Class-Weighted Cycle Consistency Loss

In addition to the CycleGAN implementation outlined in the above subsection, we also perform another, adjusted, CycleGAN training based on the idea of weighing the cycle consistency loss based on the ground truth class labels. The logic for this follows from discussing which aspects of the transformed image, qualitatively speaking, are most revealing of the fact that a transformed GTA V image is not a true Cityscapes image. Roads, for example, are very convincingly transformed, while objects such as people and cars are often the revealing factor. In addition to this, we notice that the sky in particular is often transformed in

such a way as to cause visual artefacts, as can be seen in Figure 4.14, where visible artefacts are generated by the model in the sky of the image. Given these observations, we suspect that the GTA V ground truth labels could be used to alter the weight of the cycle consistency loss, punishing the network less or more harshly, depending on the class that the pixel corresponds to. So for example, we can have cycle consistency restriction be stricter for the sky, which is prone to visual artefacts, and less strict for cars which might benefit from a stronger style transfer.

Figure 4.14: A GTA V image (left) with artefacts being generated in the sky when being style transferred (right) to Cityscapes using CycleGAN.

We thus implement this class-weighted L1 loss and use it for the cycle consistency loss. Naturally, since we assume only ground truth labels for the GTA V images, it is only used for this domain. We increase the cycle consistency weight for the sky class to 3, and decrease the cycle consistency weight for object classes such as person, car, fence, etc. to 0.33. The weights for all of the classes can be seen in Table 5.13. The results for the experiments with the transformed dataset obtained using this loss can be seen in Section 5.5.2.

4.6 Combining Approaches

Having looked into feature-level, output-level, and input-level approaches separately, we now discuss our implementations of combinations between these three approaches. To do this, we combine the approaches that give the best results within each approach. For the feature-level approaches, we found that our implementation with three discriminators, described in Section 4.3.1.3, give the best results, which can be seen in Section 5.3.1.3. For the output-level approaches we found that both give similar results. The direct output alignment, described in Section 4.4.1, gives the better result by a very small margin, but, as discussed in Section 5.4.3, the output entropy alignment model, described in Section 4.4.2, gives similar results whilst also being less volatile to changes in the training, i.e. λ_{adv} values and number of epochs, which may be beneficial when combining with other approaches. For this reason, we choose to use the output entropy alignment approach in our combinations. Lastly, for the input-level we choose the regular CycleGAN approach, described in Section 4.5.1, since it gives the best results, which can be seen in Section 5.5.1.

To combine the input-level and feature-level approaches, we train the model with three discriminators using the images generated from the regular CycleGAN input-level adaptation approach. The results from this combination can be seen in Section 5.6.1.

To combine the feature-level and output-level approaches, we create a model with four discriminators, three of the discriminators being placed in the same position as in the model with three discriminators, and the final discriminator acting on the weighted self-information space of the output as in the output entropy alignment model. We use seperate λ_{adv} values for the three discriminators and the discriminator on the weighted self-information space. The values are chosen based on the results that were best for the corresponding experiments. The results from this combination can be seen in Section 5.6.2.

To combine the input-level and output-level approaches, we train the output entropy alignment model using the images generated from the regular CycleGAN input-level adaptation approach. The results from this combination can be seen in Section 5.6.3.

To combine all three approaches, we take the aforementioned feature-level and output-level combination model, and train this using the images generated from the regular CycleGAN input-level adaptation approach. The results can be seen in Section 5.6.4.

Chapter 5

Experiment Results

5.1 Experimentation Setup

5.1.1 Dataset Details

As previously mentioned, we are using the Playing for Data dataset, introduced in Section 2.3.1, as our source domain, and the Cityscapes dataset, introduced in Section 2.2.1, as our target domain.

The Playing for Data dataset consists of 24966 images, all with ground truth labels. We use all available images and labels as a training set, despite a partitioning into training, validation, and test sets being provided. The reasoning for this is that, since this is our source domain, we never evaluate the model on this dataset, and thus having as many images as possible for training is preferable.

The Cityscapes dataset consists of 24998 images. The dataset provides pixel-precise ground truth labels for 5000 of these images in a package entitled *trainvaltest*. The remaining 19998 images are provided with coarse, significantly less precise, labels in the *trainextra* package. We use all of these 19998 images in the *trainextra* package for the domain adaptation in our implementations, of course with the corresponding coarse labels discarded. For validation and testing, we use images from the *trainvaltest* package, since these images come with pixel-precise labels. However, Cityscapes does not provide access to any labels for their test set, instead providing an online service where you can submit predictions for the test set and receive a score. The use of this service, however, has restrictions on how often it can be used per institution. We deem these restrictions too restrictive for our experiments, and so we use the provided validation set of 500 images as our test set. To then construct our own validation set, we therefore take 494 images from the training subset of the *trainvaltest* package. Specifically, we use the images from Cologne, Hannover, and Tubingen.

5.1.2 Training Details

We note that the Cityscapes images are at resolution 2048×1024 whilst GTA V images are at resolution 1914×1052. However, to keep the dimensions reasonable

for input-level adaptation, we scale the width of the images to 1024, maintaining the original image aspect ratios. Due to limitations in VRAM for the models trained in this thesis, we take a random 384×384 crop of the image to pass into the model. This also means that downscaling the images has the added effect of making the crops more representative. To best utilise the pre-trained models provided by PyTorch we normalise the images with the same values[1]. For the R, G, and B channels respectively, the images are normalised with a mean of $0.485, 0.456$, and 0.406, and a standard deviation of $0.229, 0.224$, and 0.225. We also include a 50% horizontal flip chance when loading the images. The images are shuffled when loaded, and are loaded in varying batch sizes depending on the experiment. Our general principle, however, is to use as large a batch size as possible.

With regards to the model training, the often recommended Adam [18] optimisation algorithm is used for updating the parameters. Based on a preliminary hyperparameter search, a learning rate of 1×10^{-4} is used for the discriminators, and a learning rate of 5×10^{-6} is used for the semantic segmentation network. We especially find that, once the discriminator and the adversarial loss are added, fine-tuning the learning rate of the semantic segmentation network is paramount to good domain adaptation results. After setting the learning rate of the semantic segmentation network lower than the learning rate of the discriminator, we see a big increase in accuracy. The models are initialised using PyTorch's default initialisation setting, which for the layers used is the Kaiming [11] initialisation. We run the models for 7 epochs, which is roughly 175.000 iterations, and save the model after each epoch is completed, for evaluation.

Training details for the CycleGAN network used for input-level adaptation will be discussed separately in Section 5.5. We trained all of our models using an Nvidia RTX 2060 Super with 8 GB VRAM.

5.1.3 Evaluation Metric

The simplest approach to evaluation of a segmentation map would be the pixel accuracy metric, which simply counts the percentage of correctly predicted pixels. An issue with this approach, however, is that classes that have a proportionally large amount of pixels belonging to them, such as *road* or *sky*, would thus contribute very heavily to the accuracy measure, while classes such as *traffic light* or *person* would contribute little. Therefore, for semantic segmentation, the mean intersection-over-union metric (mIoU) is often used. The IoU for a class is calculated as follows.

$$IoU = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives} + \text{False Negatives}}$$

The mIoU, which we use for all of our evaluations, is then the mean of the IoUs for each class and thus each class contributes equally to the accuracy measure.

[1] https://discuss.pytorch.org/t/how-to-preprocess-input-for-pre-trained-networks/683/2

5.2 Oracle and Baseline Experiment Results

To have a model to serve as our *oracle*, the performance of which our remaining adaptation approaches should strive to imitate, we train the semantic segmentation network on all Cityscapes training images and labels in the *trainvaltest* package, minus those we used to construct our validation set described in Section 5.1, on which we evaluate it. Since the training set in the *trainvaltest* package only has 2481 images after taking out our own constructed validation set, 70 epochs are run to have a comparable amount of training iterations. We use a batch size of 4 for this training. The results from this training can be seen in Table 5.1.

Epoch						
10	20	30	40	50	60	70
54.5	58.8	60.2	60.4	62.6	62.5	**63.3**

Table 5.1: Experiment results, in mean IoU, for the semantic segmentation oracle model.

In addition to the oracle model, we train the semantic segmentation model detailed in Section 4.2 on the GTA V images, and evaluate them on the Cityscapes images. Which is to say, no domain adaptation techniques were used. We use a batch size of 4 for this training. The results from this experiment serves as a baseline for all adaptation models, and these results can be seen in Table 5.2.

Epoch						
1	2	3	4	5	6	7
29.4	28.9	28.5	31.8	29.4	**31.9**	29.5

Table 5.2: Experiment results, in mean IoU, for the semantic segmentation model, used as a baseline.

These results serve as a form of upper and lower bound for the accuracy range, within which we expect the accuracy of our models with domain adaptation techniques applied to lie.

5.2.1 Discussion

A noteworthy result regarding the baseline is the quality of the baseline when compared to seminal papers in this area. We achieve a baseline mIoU of 31.9, which is for example considerably higher than achieved by Hoffman et al. with the same semantic segmentation model, where they reported a baseline mIoU of 17.9 [13]. This is despite the fact that our oracle training attains roughly the same mIoU as Hoffman et al. (63.3 and 60.3 respectively). We are uncertain as to what causes this large gap in mIoU for the baseline. The preliminary experiments related to removing the scaling of the skip layers, discussed in Section 4.2, show that this change is not the cause, since only a slight mIoU improvement is gained when evaluating on Cityscapes as a result of this change. An additional point of difference is our decision to use the, now popular, Adam optimisation algorithm

[18], which was not available at the time Hoffman et al. presented their results, and was therefore not an optimiser they were likely to have used. We therefore trained a baseline with the stochastic gradient descent optimisation algorithm, but this gives only slightly worse results, and as such cannot be the sole cause. To be sure that the performance gain was not caused by our choice of validation set, we made use of the service provided by the Cityscapes team, described in Section 5.1, and a very similar resulting mIoU was achieved (31.6 instead of 31.9).

Since this difference in mIoU is an increase for our model, and since we have encountered at least one other paper in this area that gives similar baseline results with the same base network, Sankaranarayanan et al. [30] obtain 29.6, we do not feel it pertinent to do further testing to see whether the cause of this improvement can be determined. This increase in mIoU could however affect the way in which one might compare the results of our experiments with results from papers in this area. If a higher baseline accuracy makes it easier to obtain a high accuracy when applying domain adaptation techniques, a direct comparison between our mIoU results and, for example, Hoffman et al.'s mIoU results, might exaggerate the effectiveness of our domain adaptation. On the other hand, since one might expect that improving mIoU gets more and more difficult as the mIoU gets closer to that of the oracle, measuring the increase in mIoU compared to the respective baseline might, in turn, understate our results to some degree. As such, one should keep this in mind if comparing our results to those of other authors.

5.3 Feature-Level Experiment Results

5.3.1 Feature Map Discriminator

For the first feature-level experiment, with a discriminator on the `last_ft` output, we tested various values of the λ_{adv} weight on the adversarial loss. We ran the model with batch size 2 for 7 epochs for each value of λ_{adv}. The best mIoU across all experiments was 39.2, which was after 5 epochs with a λ_{adv} of 0.01. The results in their entirety can be seen in Table 5.3.

		λ_{adv}			
		0.1	**0.01**	**0.001**	**0.0001**
Epoch	1	35.8	34.9	34.7	29.9
	2	34.9	36.1	37.2	27.4
	3	35.8	37.7	37.2	32.5
	4	36.8	38.5	37.8	34.8
	5	**37.6**	**39.2**	36.8	34.3
	6	36.1	38.4	**39.1**	**36.7**
	7	36.8	38.3	37.5	35.8

Table 5.3: Experiment results, in mean IoU, for the feature-level approach with a discriminator on the `last_ft` output.

5.3.1.1 Larger Discriminator

For the feature-level experiment with a larger discriminator, we tested various values of the λ_{adv} weight on the adversarial loss. We ran the model with batch size 2 for 7 epochs for each value of λ_{adv}. The best mIoU across all experiments was 39.5, which was after 6 epochs with a λ_{adv} of 0.001. The results in their entirety can be seen in Table 5.4.

		λ_{adv}			
		0.1	**0.01**	**0.001**	**0.0001**
Epoch	1	32.1	35.8	29.2	32.6
	2	36.1	36.0	35.9	37.2
	3	36.0	37.7	35.9	38.7
	4	34.8	36.4	36.3	37.5
	5	35.8	37.3	37.5	37.2
	6	36.7	37.6	36.3	**39.5**
	7	**37.4**	**38.4**	**38.3**	38.9

Table 5.4: Experiment results, in mean IoU, for the feature-level approach with the larger discriminator.

5.3.1.2 Discriminator on pool5

For the feature-level experiment with a discriminator on `pool5`, we tested various values of the λ_{adv} weight on the adversarial loss. We ran the model with batch size 2 for 7 epochs for each value of λ_{adv}. The best mIoU across all experiments was 38.6, which was after 7 epochs with a λ_{adv} of 0.01. The results in their entirety can be seen in Table 5.5.

		λ_{adv}			
		0.1	**0.01**	**0.001**	**0.0001**
Epoch	1	34.6	34.1	30.1	25.3
	2	35.1	37.4	28.6	32.7
	3	34.2	36.5	34.1	29.6
	4	35.6	35.6	32.7	30.8
	5	**36.1**	37.7	31.9	30.7
	6	35.9	38.2	36.1	**33.5**
	7	36.0	**38.6**	**36.9**	30.5

Table 5.5: Experiment results, in mean IoU, for the feature-level approach with a discriminator on `pool5`.

5.3.1.3 Three Discriminators

For the feature-level experiment with three discriminators, we tested various values of the λ_{adv} weight on the adversarial loss. We ran the model with batch size 2 for 7 epochs for each value of λ_{adv}. The best mIoU across all experiments was 40, which was after 4 epochs with a λ_{adv} of 0.001. The results in their entirety can be seen in Table 5.6.

		λ_{adv}			
		0.1	**0.01**	**0.001**	**0.0001**
Epoch	1	31.5	35.9	33.6	33.6
	2	31.2	37.7	36.5	31.8
	3	32.1	37.7	37.5	33.2
	4	32.6	35.0	**40.0**	32.2
	5	32.9	**38.9**	39.0	**36.0**
	6	33.6	37.7	39.7	34.7
	7	**33.8**	38.0	38.4	33.0

Table 5.6: Experiment results, in mean IoU, for the feature-level approach with three discriminators.

5.3.2 Conditional Generator

For the feature-level experiment with a conditional generator, we first performed preliminary tests with the many combinations of potential network setups. There were four points of variation, being the discriminator used, the generator used, the training variation used, and the value of λ_{adv} used. Two discriminators were tested, the fully connected discriminator, and the cell-based discriminator. Two generators were tested, the one provided by Hong et al. [15], and the generator with downsampling. The three training methods, *Disjoint Training*, *Generator Learns Both*, and *Both Learn Both*, were tested. Finally, four values of λ_{adv} were tested. All of the resulting combinations, 48 in total, were trained for 1 epoch. Due to the size of the model, a batch size of 1 was used. Unlike the other experiments presented in this thesis, these preliminary combination tests were run with a learning rate of 1×10^{-4} for the semantic segmentation

		Disc. F		Disc. C	
		Gen. H	Gen. D	Gen. H	Gen. D
		Disjoint Training			
λ_{adv}	0.01	18.0	16.6	16.9	**21.8**
	0.1	17.9	**22.0**	15.7	16.3
	1	13.1	15.0	24.1	15.3
	10	**18.2**	14.7	**24.2**	18.2
		Generator Learns Both			
λ_{adv}	0.01	**14.4**	18.4	16.3	14.4
	0.1	14.1	**21.8**	13.9	16.3
	1	13.4	21.5	**21.2**	14.6
	10	14.3	19.0	17.2	**19.8**
		Both Learn Both			
λ_{adv}	0.01	1.2	16.0	**19.6**	20.5
	0.1	**2.0**	**18.9**	14.9	13.6
	1	**2.0**	18.5	13.6	21.5
	10	**2.0**	13.2	16.6	**20.7**

Table 5.7: Experiment results, in mean IoU, for the preliminary tests with the many combinations of network setups. Disc. F refers to the fully-connected discriminator. Disc. C refers to the cell-based discriminator. Gen. H refers to the generator as per Hong et al. Gen. D refers to the generator with downsampling.

network and conditional generator, since at the time these experiments were performed, we believed this to be the best learning rate, based on an earlier, admittedly less thorough, hyperparameter search. The best mIoU across all of these combinations was 24.2, which was using the cell-based discriminator, the generator as per Hong et al., the *Disjoint Training* variation, and a λ_{adv} of 10. The full results from this experiment can be seen in Table 5.7. After arriving at this combination, we ran this combination for a full 14 epochs. The reason for training for 14 epochs, compared to the 7 epochs used in other experiments, is that the training is done in a two-stage fashion, such that in the course of a full epoch, each part of the network only sees one half of the images each. Regarding the learning rate, for this full training, we used 5×10^{-6} as the learning rate for the semantic segmentation network and generator. The best mIoU was 33.5, which was arrived at after 4 epochs. The results for this training can be seen in Table 5.8.

Epoch						
2	4	6	8	10	12	14
27.5	**33.5**	29.2	28.3	28.5	29.9	26.4

Table 5.8: Experiment results, in mean IoU, for the feature-level approach with a conditional generator.

5.3.3 Discussion

5.3.3.1 Feature Map Discriminator and Variants

The experiments using a discriminator operating on feature maps show good results, with the best mIoUs for the different methods all lying in the range of 38.6 to 40.0 when evaluated on the validation set, which is slightly better than any other result we have seen for VGG16-based networks in this area. The fact that the different methods give mIoU results in this narrow range show that, at least to some extent, the size, placement and number of discriminators is not that important to achieving good results. Results also stay roughly the same across one, or sometimes two, orders of magnitude in regards to the value of λ_{adv}. As previously mentioned, we found the learning rate of the semantic segmentation network to be the most important thing to adjust, with a significant boost to the accuracy being gained when it is decreased to 5×10^{-6}. It is important to note that the baseline experiment without a discriminator is not nearly as sensitive to the learning rate of the semantic segmentation network, suggesting that the sensitivity to this learning rate comes from how it affects the adversarial game between the semantic segmentation network and the discriminator. Overall, the method with three discriminators shows the best results, but only by a small margin.

5.3.3.2 Conditional Generator

Whilst the method using the conditional generator sounds promising, our experiments, at the very least, show that it is difficult and time consuming to implement and train successfully. We were not able to improve significantly over the baseline using the conditional generator. Given more time, we would have liked to investigate this method further. An interesting thing to try would be to pre-train, and then freeze, the encoder, before attaching the conditional generator. This would simplify the adversarial game, and perhaps make it easier to investigate whether the network is being trained in a desirable way. Part of this investigation might be to then visualise the features to better see the behaviour of the adaptation. Another possible approach would be to change the structure of the generator to instead operate on the `last_ft` feature map, as has been done favourably in our previous feature-level approaches, to see if this would lead to any significant changes. There is also a fourth training variation that one could appropriately call *Encoder Learns Both*, where the conditional generator is only updated using the adversarial loss, but both the semantic segmentation loss and the adversarial loss are used to update the encoder. We did not experiment with this training variation as it is not a very probable interpretation of the original paper by Hong et al., and is a variation where the encoder once again has to generate features that result from a compromise between two tasks. On the other hand, if the conditional generator, given its ResNet architecture and single task, learns to adapt significantly faster than the semantic segmentation network, the semantic segmentation network would not have to perform as much adaptation as it would have to without the generator, so the approach could still be worth experimenting with. In conclusion, while we cannot rule out that using a conditional generator might produce good results, GANs are infamously hard to train, and that shows in this case as well. As such, we would not recommended this method as a starting point for unsupervised domain adaptation.

5.4 Output-Level Experiment Results

5.4.1 Direct Output Alignment

For the output-level experiment with a discriminator directly on the output, we tested various values of the λ_{adv} weight on the adversarial loss. We ran the model with batch size 2 for 7 epochs for each value of λ_{adv}. The best mIoU across all experiments was 37.7, which was after 5 epochs with a λ_{adv} of 0.01. The results in their entirety can be seen in Table 5.9.

5.4.2 Output Entropy Alignment

For the output-level experiment with a discriminator on the weighted self-information space, we tested various values of the λ_{adv} weight on the adversarial loss. We ran the model with batch size 2 for 7 epochs for each value of λ_{adv}. The best mIoU across all experiments was 37.6, which was after 5 epochs with a λ_{adv} of 0.01. The results in their entirety can be seen in Table 5.10.

		λ_{adv}			
		0.1	**0.01**	**0.001**	**0.0001**
Epoch	1	26.9	34.3	33.1	30.8
	2	29.8	36.1	35.3	34.5
	3	27.9	33.9	**37.5**	35.4
	4	34.1	36.4	35.0	**36.9**
	5	31.8	**37.7**	35.5	32.5
	6	**35.5**	35.9	34.5	34.7
	7	32.2	34.0	34.6	34.9

Table 5.9: Experiment results, in mean IoU, for the output-level approach with a discriminator directly on the output.

		λ_{adv}			
		0.1	**0.01**	**0.001**	**0.0001**
Epoch	1	**36.4**	33.9	34.9	27.8
	2	35.5	**37.6**	35.4	35.0
	3	35.2	36.5	35.6	34.5
	4	35.2	37.1	35.9	35.5
	5	35.2	36.8	36.8	34.6
	6	34.3	37.0	**37.3**	33.2
	7	34.1	37.2	37.1	**36.5**

Table 5.10: Experiment results, in mean IoU, for the output-level approach with a discriminator on the weighted self-information space.

5.4.3 Discussion

As could maybe be expected, since the two output-level approaches are very similar, the results are very similar as well, yielding a best mIoU of 37.7 and 37.6 respectively. These mIoU scores are slightly worse than what is achieved with the feature-level adaptation. Despite being worse, one could, however, argue that they are close enough to still support the findings, discussed in Section 5.3.3.1, that a relatively narrow range of mIoU scores are achieved despite changes in the position of, and in this case the type of, discriminator. While the approach with the discriminator directly on the output does, albeit slightly, achieve a better mIoU score, the approach with the discriminator on the weighted self-information space seems to be slightly more robust to changes to the value of λ_{adv} and the number of epochs. This is especially apparent for a λ_{adv} value of 0.1.

The output-level approach with a discriminator on the weighted self-information space is designed with the purpose of reducing the entropy of the predictions on target images. In Figure 5.1 we show the entropy of the predictions for an image for the baseline model, the feature-level approach with three discriminators, and both output-level approaches. We note that both output-level approaches give a large reduction in entropy over the baseline, more so than the feature-level approach, and thus it seems that explicitly reducing the entropy does not make a large difference. Furthermore, since our results indicate that the feature-level approach performs better than the output-level approaches, this suggests that less uncertainty will not always mean that a model performs better.

Figure 5.1: A Cityscapes image, as well as entropy maps for the indicated models, with black representing uncertainty.

5.5 Input-Level Experiment Results

The CycleGAN network is trained with a batch size of 1 using the Adam optimisation algorithm, with a learning rate of 2×10^{-4} and betas of 0.5 and 0.999, the settings provided by Zhu et al. [44]. For preprocessing, the width of the images are downscaled to 1024, maintaining the original image aspect ratios, and then randomly cropped to 360×360. We trained the network for 10 epochs, after which the learning rate was linearly decayed for an additional 10 epochs.

5.5.1 CycleGAN

For the input-level experiment with the CycleGAN translated images, the semantic segmentation network was trained in the same manner as the baseline experiment, but with the images generated from the CycleGAN model. The best mIoU was epoch 5 with 42.2. The results from all epochs can be seen in Table 5.11.

Epoch						
1	2	3	4	5	6	7
37.6	39.0	40.0	39.5	40.4	**42.2**	40.8

Table 5.11: Experiment results, in mean IoU, for the input-level approach using the CycleGAN model.

5.5.2 Class-Weighted Cycle Consistency Loss

For the input-level experiment with the translated images from the CycleGAN network trained with a class-weighted cycle consistency loss, the semantic segmentation network was trained in the same manner as the baseline experiment, but with these translated images. The best mIoU was epoch 5 with 41.5. The results from all epochs can be seen in Table 5.12.

Epoch						
1	2	3	4	5	6	7
36.5	39.2	39.6	41.0	**41.5**	40.5	41.2

Table 5.12: Experiment results, in mean IoU, for the input-level approach using the adjusted class-based cycle consistency CycleGAN model.

5.5.3 Discussion

As seen in Figure 5.2, the GTA V images have been adapted such that they have the style of a Cityscapes image. The images are a lot less saturated and the hue is shifted to more green colours, the road is a lot smoother with a white centre line instead of a yellow one, and, interestingly, in some instances hood ornaments have been generated. These images also highlight one of input-level adaptation's biggest strengths, namely human interpretability. With the adaptation done on the images themselves, it is possible to visually confirm that the adaptation produces reasonable results. Another advantage of the CycleGAN approach is that the images can be adapted once and then distributed as an improved dataset, usable in any network without any changes. Training on these images yields an mIoU score of 42.2, higher than that of feature-level and output-level adaptation. We reflect on these good results in our final discussion in Section 5.7.1.

Figure 5.2: An image depicting a GTA V image (left) before being translated (right) into the style of a Cityscapes image.

Regarding the class-weighted cycle consistency loss, our motivation was, as mentioned in Section 4.5.2, twofold. Our first aim was that by adding a class weight to the cycle consistency loss, we would be able to avoid artefacts from appearing in the sky, which may affect the models ability to classify the sky correctly. The model with class-weighted cycle consistency loss is, qualitatively speaking, fairly successful in visually reducing the artefacts in the sky of the image, as seen in Figure 5.3. However, this qualitative visual improvement interestingly does not mean that the model can better learn how to predict the pixels belonging to the sky class, as can be seen in Table 5.13. Our second aim was that by reducing the weight of the cycle consistency loss on various objects, that the CycleGAN model would be more aggressive in converting the style of certain classes from GTA V to Cityscapes, which would increase the domain adaptation performance without adding artefacts from the objects that are already being style-transferred well. We do not notice anything qual-

itatively different in the images by having a lower cycle consistency weight on these objects, and as can be seen in Table 5.13, it does not increase the mIoU on any classes significantly. A potential avenue of improvement could be to perform a hyperparameter search on the weights added to the per-class cycle consistency loss. However, since the CycleGAN model takes several days to train, this was not feasible for this thesis. In addition, considering the fact that the accuracy on the sky class is worse despite the sky being translated with fewer artefacts in the images, it is not obvious how one would guide such a search.

Figure 5.3: An image showing artefacts in the sky of a translated GTA V image (left) being reduced after using a class-weighted cycle-consistency loss (right).

	Road	Sidewalk	Building	Wall	Fence	Pole	Traffic Light	Traffic Sign	Vegetation	Terrain	Sky	Person	Rider	Car	Truck	Bus	Train	Motorcycle	Bicycle
Regular	88.5	46.0	80.6	26.1	29.6	27.4	28.6	22.1	82.7	48.3	77.9	58.3	20.1	82.2	19.8	5.0	12.2	29.6	17.2
Class-weighted	87.9	34.2	80.2	26.8	25.0	27.8	27.3	25.5	83.2	47.7	76.7	55.2	22.2	82.6	23.5	15.5	5.7	26.6	14.2
Gain	-0.6	-11.8	-0.4	0.7	-4.6	0.4	-1.3	3.4	0.5	-0.6	-1.2	-3.1	2.1	0.4	3.7	10.5	-6.5	-3.0	-3.0
Weight used	1.00	1.00	1.00	1.00	0.33	0.33	0.33	0.33	1.00	1.00	3.00	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33

Table 5.13: A comparison of class IoU scores from the two input-level models, with the IoU gain shown in the third row, and the per-class cycle consistency weights shown in the fourth row.

5.6 Combination Experiment Results

For the combination of approaches, our initial thoughts were to use the hyperparameters that gave the best results for each respective approach. However, we suspected that combining approaches could affect the optimal combination of hyperparameters. We especially suspected that using images from the input-level adaptation approach might affect the adversarial game of the other approaches, as one can reasonably assume that the features in that case would be closer to each other from the start, and thus make the job for the discriminator considerably harder. We therefore performed a three-dimensional random search for the three main hyperparameters in our model, being the learning rate for the semantic segmentation network, the discriminator learning rate, and the value

of λ_{adv}. However, a week-long hyperparameter search yielded no combination of hyperparameters that could outperform the original choice after three epochs.

5.6.1 Input-Level and Feature-Level Combined

For the experiment combining input-level and feature-level approaches, we combined the best approaches within the input-level and feature-level experiments. That being, training the feature-level approach with three discriminators on images generated from the regular CycleGAN network. We trained the network using the λ_{adv} value that gave the best results in the feature-level experiment with three discriminators, which was 0.001. We ran the model with batch size 2 for 7 epochs. The best mIoU was 41.6, which was after both 5 and 6 epochs. The results in their entirety can be seen in Table 5.14.

Epoch						
1	2	3	4	5	6	7
37.6	40.0	40.3	39.1	**41.6**	**41.6**	41.3

Table 5.14: Experiment results, in mean IoU, for the approach combining the chosen input-level and feature-level approaches.

5.6.2 Feature-Level and Output-Level Combined

For the experiment combining feature-level and output-level approaches, we combined the best approaches within the feature-level and output-level experiments. That being, we trained the network with four discriminators in total, three from the feature-level approach with three discriminators, and a fourth from the output-level approach on the weighted self-information space. We trained the network using two different λ_{adv} values. For the three discriminators on the feature space, we used the λ_{adv} value that gave the best results in the feature-level experiment with three discriminators, which was 0.001. For the discriminator on the weighted self-information space, we used the λ_{adv} value that gave the best results in the output-level experiment on the weighted self-information space, which was 0.01. We ran the model with batch size 2 for 7 epochs. The best mIoU was 40.1, which was after 6 epochs. The results in their entirety can be seen in Table 5.15.

Epoch						
1	2	3	4	5	6	7
36.3	37.5	39.2	39.2	39.6	**40.1**	39.8

Table 5.15: Experiment results, in mean IoU, for the approach combining the chosen feature-level and output-level approaches.

5.6.3 Input-Level and Output-Level Combined

For the experiment combining the input-level and output-level approaches, we combined the best approaches within the input-level and output-level experiments.

That being, training the output-level approach with a discriminator on the weighted self-information space on images generated from the regular CycleGAN network. We trained the network using the λ_{adv} value that gave the best results in the output-level experiment on the weighted self-information space, which was 0.01. We ran the model with batch size 2 for 7 epochs. The best mIoU was 40.9, which was after 5 epochs. The results in their entirety can be seen in Table 5.16.

Epoch						
1	2	3	4	5	6	7
35.8	39.3	40.6	40.4	**40.9**	39.6	40.7

Table 5.16: Experiment results, in mean IoU, for the approach combining the chosen input-level and output-level approaches.

5.6.4 All Three Levels Combined

For the experiment with all three levels combined, we took the best approaches within all levels of adaptation, being input, feature, and output. Thus our combined model was to train on images generated from the regular CycleGAN network, using a model with four discriminators, three from the feature-level approach with three discriminators, and a fourth from the output-level approach on the weighted self-information space. As in Section 5.6.2, we trained the network using the best λ_{adv} values obtained in the corresponding experiments. As such, the three discriminators from the feature-level experiment had a λ_{adv} value of 0.001, whilst the discriminator from the output-level experiment had a λ_{adv} value of 0.01. We ran the model with batch size 2 for 7 epochs. The best mIoU was 41.1, which was after 5 epochs. The results in their entirety can be seen in Table 5.17.

Epoch						
1	2	3	4	5	6	7
36.2	39.6	40.2	40.3	**41.1**	40.2	40.8

Table 5.17: Experiment results, in mean IoU, for the approach combining the chosen feature-level, input-level, and output-level approaches.

5.6.5 Discussion

Interestingly, our experiments show that, at least when using the VGG16-FCN8s network, combining different levels of adaptation does not increase the performance, and in fact gives slightly worse results than purely doing input-level adaptation. This suggests that the different levels of adaptation are, in the end, performing the adaptation in the same way, or are perhaps even working against each other to a slight degree. We return to this in our final discussion in Section 5.7.1.

5.7 Evaluation on Test Set

As mentioned in Section 5.1.1, we use the validation set provided by the Cityscapes team as our test set. We evaluated all of our final models on this test set. That being, the oracle, the baseline, each of the best models from feature-level, output-level, and input-level approaches, and finally the four combination models. For the evaluation on our test set, we also show the IoU scores per class, and the full table of results can be seen in Table 5.18.

	Road	Sidewalk	Building	Wall	Fence	Pole	Traffic Light	Traffic Sign	Vegetation	Terrain	Sky	Person	Rider	Car	Truck	Bus	Train	Motorcycle	Bicycle	mIoU
Baseline	51.8	20.1	62.5	10.3	22.2	22.4	22.5	13.9	80.4	14.3	53.5	51.5	16.0	52.7	15.8	9.7	2.9	13.6	8.0	28.6
Feature	87.4	35.0	77.6	25.8	21.0	26.8	24.1	11.8	80.9	29.9	64.9	53.5	18.0	81.1	22.4	13.9	**8.5**	**27.2**	12.8	38.8
Output	85.7	31.0	78.0	22.7	11.3	20.4	14.6	6.9	81.8	31.1	69.5	48.1	**22.3**	80.5	24.7	**31.0**	0.4	15.0	6.9	35.9
Input	87.8	42.5	**80.3**	**31.2**	**25.9**	**27.8**	27.3	19.7	81.7	34.0	**74.0**	**56.5**	17.8	81.5	16.0	19.7	2.1	26.2	**18.7**	**40.6**
Feat.+Out.	86.8	29.4	79.7	26.0	22.5	24.9	24.9	10.4	**82.5**	29.7	72.5	54.7	22.1	81.2	21.3	24.5	8.4	24.0	12.4	38.8
Feat.+In.	**88.9**	**43.6**	80.2	30.9	20.0	27.0	**29.2**	**20.0**	81.9	31.4	73.7	54.4	19.7	**82.4**	22.6	15.0	6.1	20.3	15.5	40.1
Out.+In.	88.3	41.0	79.9	27.7	22.8	27.1	26.8	11.7	79.6	29.5	70.7	54.8	15.5	**82.4**	**25.6**	24.5	0.4	23.4	6.6	38.9
All Comb.	88.5	41.2	79.9	30.1	22.7	26.2	27.3	9.8	81.4	**34.4**	71.2	55.3	20.5	82.0	25.4	29.8	0.1	24.0	6.2	39.8
Oracle	96.5	75.1	88.4	44.5	46.5	39.9	48.3	59.8	89.5	56.4	92.0	69.0	43.4	90.0	54.3	62.5	62.7	48.4	65.8	64.9

Table 5.18: Experiment results of the final models on the test set, including the IoU for each class, and the mean IoU. The best non-oracle value in each column is highlighted in bold.

Evaluating on our test set gives slightly lower mIoU scores for most of our models when compared to our validation set, but the approach with the best results is still the input-level approach, which achieves a score of 40.6 mIoU. We additionally use the Cityscapes service to evaluate the input-level approach on their official test set. This results in an official score of **43.6** mIoU[2], higher than the score on both our validation set and our test set. As is to be expected, since the input-level approach gives the best results in mean IoU, it is also the model with the best IoU for many of the classes.

To illustrate the improvement over the baseline achieved by the various adaptation methods presented in this thesis, we show in Figure 5.4 the semantic segmentation map predictions for two images from our test set. As can be seen, all models improve significantly over the baseline. The difference between the results of the models is somewhat difficult to see qualitatively, and reflects the fact that the models have similar performance for the main classes that often fill most of the image, such as road, sidewalk, person, and vegetation.

[2] https://www.cityscapes-dataset.com/anonymous-results/?id=52ccc3daf37af43f962217498d69ce047e725b3fe8639d52ce0590c5560e8f33

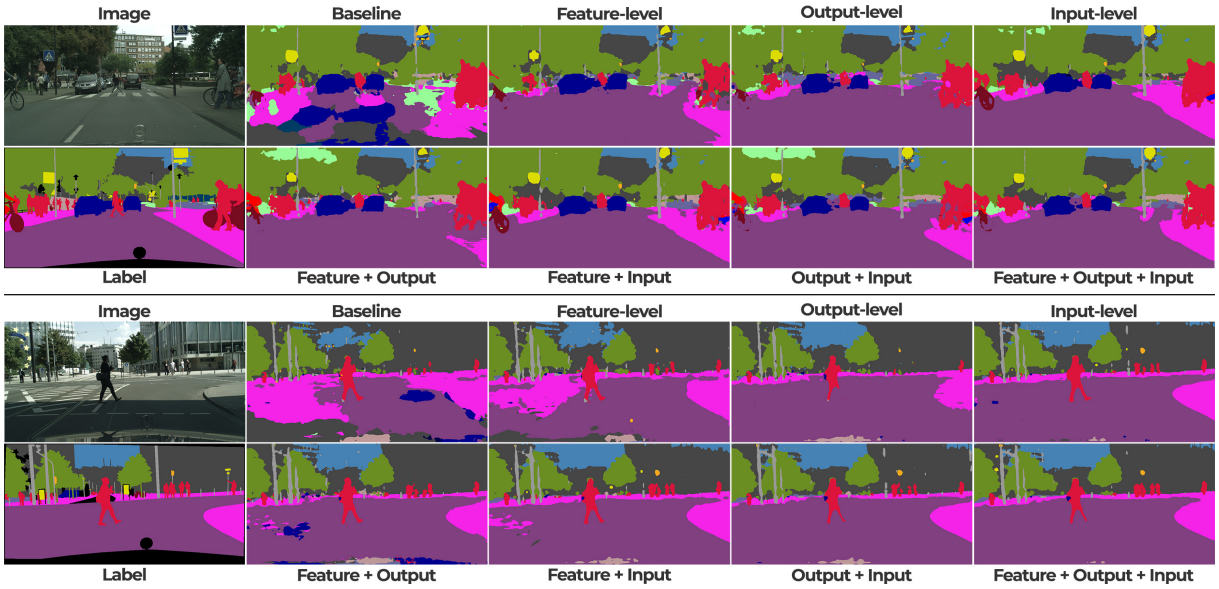


Figure 5.4: Semantic segmentation map predictions from various models.

5.7.1 Final Discussion

As we discussed in Section 5.6.5, the fact that combining models does not improve mIoU could be due to the fact that input-level, output-level, and feature-level approaches are all performing the adaptation in the same way. However, when looking at the IoU for each of the different classes in Table 5.18, there is something of note. For instance, the feature-level approach is the best model on the *train* class, attaining a score of 8.5, and the two other models that score similarly are combinations that include the feature-level approach. Another example is the *bus* class, where the output-level approach has a relatively high score of 31.0, and the other models that score similarly all include the output-level approach in their combination. This suggests, in fact, that to a certain degree, the various approaches do not perform the adaptation in the same way. The result still stands however, that the mean IoU is not increased by combining the approaches, and thus that the methods are not fully complementary, and even seem to work against each other somewhat. There is still a point to be made, however, that if in a certain situation, where one approach scores high on certain classes and low on others, and the opposite is true for another approach, one could perhaps beneficially combine these for more balanced accuracy scores, at the cost of a slightly lower mean IoU. There is no certainty, however, that combinations would behave like this, which can be seen with the *bicycle* class, where even though the well-performing input-level approach, with a score of 18.7, is combined with the poorly performing output-level approach, with a score of 6.9, the combination is in fact even worse, resulting in a score of 6.6.

Overall, our results show good improvement over the baseline model, but unsupervised domain adaptation using synthetic data is still, at least in this choice of

domains, a long way from fully closing the gap in performance to the oracle model. Our experiments show that combining effective feature-level, input-level, and output-level approaches does not give significant improvements in performance, with our best results coming from solely using the input-level approach using CycleGAN translated images.

Regarding these good results from the input-level approach, it is, in a way, both surprising, and unsurprising, that using CycleGAN translated images ends up being the best approach for this case of the unsupervised domain adaptation problem. On the one hand, it is surprising since performing the adaptation by making source images look like target images is a conceptually simple idea. On the other hand, this is not actually a simple thing to do, and CycleGAN is a fairly advanced model that was trained for several days to be able to perform this shift in the image-space. Performing the adaptation in the image-space also allows adaptation to be performed on the low-level features, such as textures, which may be important for the semantic segmentation task, since classification is done at the pixel level. Additionally, because the adaptation is done on the images themselves, the semantic segmentation network only needs to focus on the task of performing semantic segmentation, which may be beneficial. We note that the strong performance of the CycleGAN approach is not apparent in the literature as Hoffman et al. [13] only achieve a score of 34.8 when using the CycleGAN approach, which is worse than for other approaches presented in this area. By comparison, we achieved a score of 43.6 using the same base network. Thus, a main takeaway from our experiments is that performing input-level adaptation using CycleGAN serves as a very strong approach to the unsupervised domain adaptation problem for semantic segmentation, perhaps even stronger than the current literature would suggest.

Chapter 6

Conclusion

In this thesis we investigate the topic of unsupervised domain adaptation for semantic segmentation, and categorise the adversarial approaches in this area into three levels, depending on how a discriminator is used. To experiment with various approaches within these levels, we use the synthetic Playing for Data [29] dataset as our source domain, the Cityscapes [6] dataset as our target domain, and the VGG16-FCN8s [21] network as our base semantic segmentation network. To serve as reference points, we train and evaluate both a *baseline* model, trained without any domain adaptation techniques, and an *oracle* model, trained using the Cityscapes labels that we otherwise assume do not exist.

We first investigate the feature-level approaches, which work by having the discriminator differentiate between feature maps extracted from the source and the target domain, with the purpose of encouraging the semantic segmentation network to extract domain-invariant features through an adversarial loss. With domain-invariant features, the network would work equally well for both the source and target domain, despite only being trained on the former. We implement the feature-level adaptation approach using a feature map discriminator, and experiment with variations in the placement, size and number of discriminators. We find all variations improve significantly over the baseline model, with the best performing variation being one where we place a discriminator on each of the three feature maps that directly contribute to the final output. Another feature-level approach uses a separate conditional generator to adapt feature maps from the source domain, so as to look like feature maps from the target domain. In this way, the subsequent part of the network is trained specifically on target-like feature maps, instead of just domain-invariant ones. We experimented with this approach using various discriminators, generators, and training variations, but were unable to get significantly better results than the baseline model using this approach.

We next investigate output-level approaches, the first of which uses a domain discriminator directly on the output of the semantic segmentation network. The argument for this is that, while source and target images can differ a lot, their ground truth segmentation maps still have many similarities, and thus the gen-

erated segmentation maps should share strong similarities as well. The second approach instead uses a domain discriminator on the *weighted self-information space*, the space from which an entropy map can be obtained by pixel-wise aggregate, with the purpose of increasing the confidence of predictions on target-like images. We find that both approaches similarly improve significantly over the baseline model, but with slightly worse results than that of the successful feature-level approaches. By comparing entropy maps from several models, we find that explicitly aligning the entropy does not significantly reduce the entropy when compared to aligning the output directly, and that less uncertainty does not always mean that a model performs better.

We then investigate image-level approaches based on the CycleGAN [44] network, which takes images from one domain and translates it in such a way as to make it look as though it was sampled from another domain. In this way, we are able to translate the synthetic images from our source domain into the style of our target domain, producing a new, adapted, dataset that can be trained on directly. Additionally, we introduce a variation of CycleGAN that uses a class-weighted cycle-consistency loss, with the purpose of adjusting the strength of the style transfer for each class individually. Training on these CycleGAN translated images improves significantly over the baseline, more so than the feature-level and output-level approaches. The introduction of our class-weighted cycle-consistency loss, however, did not further improve the performance, despite leading to a qualitative visual improvement in the translated images.

Finally, we look into whether or not improvements can be attained through the combination of feature-level, output-level, and input-level approaches. We find that combining the various levels of approaches does not give better results, and in fact can produce slightly worse results. This could indicate that the approaches are, to a large degree, adapting in a similar way, and that for the degree to which they are adapting differently, they are working against each other. In conclusion, we find that the input-level approach using CycleGAN serves as the strongest approach to the unsupervised domain adaptation problem for semantic segmentation, and perhaps a stronger one than the current literature would suggest.

Bibliography

[1] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In Andrea Pohoreckyj Danyluk, Léon Bottou, and Michael L. Littman, editors, *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, volume 382 of *ACM International Conference Proceeding Series*, pages 41–48. ACM, 2009.

[2] Angel X. Chang, Thomas A. Funkhouser, Leonidas J. Guibas, Pat Hanrahan, Qi-Xing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository. *CoRR*, abs/1512.03012, 2015.

[3] 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 Trans. Pattern Anal. Mach. Intell.*, 40(4):834–848, 2018.

[4] Yi-Hsin Chen, Wei-Yu Chen, Yu-Ting Chen, Bo-Cheng Tsai, Yu-Chiang Frank Wang, and Min Sun. No more discrimination: Cross city adaptation of road scene segmenters. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2011–2020. IEEE Computer Society, 2017.

[5] Yuhua Chen, Wen Li, and Luc Van Gool. ROAD: reality oriented adaptation for semantic segmentation of urban scenes. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 7892–7901. IEEE Computer Society, 2018.

[6] 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 *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 3213–3223. IEEE Computer Society, 2016.

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 248–255. IEEE Computer Society, 2009.

[8] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Häusser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 2758–2766. IEEE Computer Society, 2015.

[9] Yaroslav Ganin and Victor S. Lempitsky. Unsupervised domain adaptation by backpropagation. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1180–1189. JMLR.org, 2015.

[10] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial networks. *CoRR*, abs/1406.2661, 2014.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1026–1034. IEEE Computer Society, 2015.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016.

[13] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei A. Efros, and Trevor Darrell. Cycada: Cycle-consistent adversarial domain adaptation. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1994–2003. PMLR, 2018.

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

[15] Weixiang Hong, Zhenzhen Wang, Ming Yang, and Junsong Yuan. Conditional generative adversarial network for structured domain adaptation. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 1335–1344. IEEE Computer Society, 2018.

[16] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 2261–2269. IEEE Computer Society, 2017.

[17] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 5967–5976. IEEE Computer Society, 2017.

[18] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[19] Zhengqin Li and Jiansheng Chen. Superpixel segmentation using linear spectral clustering. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 1356–1363. IEEE Computer Society, 2015.

[20] James J Little and Alessandro Verri. Analysis of differential and matching methods for optical flow. 1988.

[21] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 3431–3440. IEEE Computer Society, 2015.

[22] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael I. Jordan. Learning transferable features with deep adaptation networks. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 97–105. JMLR.org, 2015.

[23] Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I. Jordan. Unsupervised domain adaptation with residual transfer networks. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 136–144, 2016.

[24] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[25] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2813–2821. IEEE Computer Society, 2017.

[26] Zak Murez, Soheil Kolouri, David J. Kriegman, Ravi Ramamoorthi, and Kyungnam Kim. Image to image translation for domain adaptation. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 4500–4509. IEEE Computer Society, 2018.

[27] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 77–85. IEEE Computer Society, 2017.

[28] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[29] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II*, volume 9906 of *Lecture Notes in Computer Science*, pages 102–118. Springer, 2016.

[30] Swami Sankaranarayanan, Yogesh Balaji, Arpit Jain, Ser-Nam Lim, and Rama Chellappa. Learning from synthetic data: Addressing domain shift for semantic segmentation. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 3752–3761. IEEE Computer Society, 2018.

[31] Matan Sela, Pingmei Xu, Junfeng He, Vidhya Navalpakkam, and Dmitry Lagun. Gazegan - unpaired adversarial image generation for gaze estimation. *CoRR*, abs/1711.09767, 2017.

[32] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Joshua Susskind, Wenda Wang, and Russell Webb. Learning from simulated and unsupervised images through adversarial training. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 2242–2251. IEEE Computer Society, 2017.

[33] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[34] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 1–9. IEEE Computer Society, 2015.

[35] 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 *2018 IEEE Conference on Computer*

Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018, pages 7472–7481. IEEE Computer Society, 2018.

[36] Tuan-Hung Vu, Himalaya Jain, Maxime Bucher, Matthieu Cord, and Patrick Pérez. ADVENT: adversarial entropy minimization for domain adaptation in semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 2517–2526. Computer Vision Foundation / IEEE, 2019.

[37] Erroll Wood, Tadas Baltrusaitis, Louis-Philippe Morency, Peter Robinson, and Andreas Bulling. Learning an appearance-based gaze estimator from one million synthesised images. In Pernilla Qvarfordt and Dan Witzner Hansen, editors, *Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research & Applications, ETRA 2016, Charleston, SC, USA, March 14-17, 2016*, pages 131–138. ACM, 2016.

[38] Zifeng Wu, Chunhua Shen, and Anton van den Hengel. Wider or deeper: Revisiting the resnet model for visual recognition. *Pattern Recognit.*, 90:119–133, 2019.

[39] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[40] Fisher Yu, Vladlen Koltun, and Thomas A. Funkhouser. Dilated residual networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 636–644. IEEE Computer Society, 2017.

[41] Xucong Zhang, Yusuke Sugano, Mario Fritz, and Andreas Bulling. Mpiigaze: Real-world dataset and deep appearance-based gaze estimation. *CoRR*, abs/1711.09017, 2017.

[42] Yang Zhang, Philip David, and Boqing Gong. Curriculum domain adaptation for semantic segmentation of urban scenes. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2039–2049. IEEE Computer Society, 2017.

[43] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 6230–6239. IEEE Computer Society, 2017.

[44] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2242–2251. IEEE Computer Society, 2017.

[45] Yang Zou, Zhiding Yu, B. V. K. Vijaya Kumar, and Jinsong Wang. Unsupervised domain adaptation for semantic segmentation via class-balanced self-training. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part III*, volume 11207 of *Lecture Notes in Computer Science*, pages 297–313. Springer, 2018.

Appendix A

Appendix

A.1 Related Work Results

In Table A.1 we provide an overview of the results achieved by the papers discussed in Chapter 3. We show the results of the presented adaptation approach, measured in mean intersection over union (mIoU), which is described

Author	Base Network	mIoU	
Hoffman et al. [14] (FCNs in the Wild)	Front-end dilated VGG16 [39]	Baseline	21.1
		Adapted	27.1
Chen et al. [5]	DeepLab v2 VGG16 [3]	Baseline	21.9
		Adapted	35.9
	PSPNet ResNet-101 [43]	Baseline	27.8
		Adapted	39.4
Hong et al. [15]	VGG19-FCN8s [21]	Adapted	44.5
Tsai et al. [35]	DeepLab v2 VGG16 [3]	Adapted	35.0
	DeepLab v2 ResNet-101 [3]	Baseline	36.6
		Adapted	42.4
Vu et al. [36]	DeepLab v2 VGG16 [3]	Adapted	36.1
	DeepLab v2 ResNet-101 [3]	Adapted	45.5
Hoffman et al. [13] (CyCADA)	VGG16-FCN8s [21]	Baseline	17.9
		Adapted	35.4
	DRN-26 [40]	Baseline	21.7
		Adapted	39.5
Sankaranarayanan et al. [30]	VGG16-FCN8s [21]	Baseline	29.6
		Adapted	37.1
Murez et al. [26]	DRN-34 [40]	Adapted	31.8
	Dilated DenseNet [16]	Baseline	29.0
		Adapted	35.7
Zhang et al. [42]	VGG19-FCN8s [21]	Baseline	22.3
		Adapted	28.9
Zou et al. [45]	VGG16-FCN8s [21]	Baseline	24.3
		Adapted	36.1
	ResNet-38 [38]	Baseline	35.4
		Adapted	47.0

Table A.1: A table of results from the papers presented in Chapter 3.

in Section 5.1.3. Where available, we also present the baseline results, by which we mean that the model is trained solely on the Playing for Data images, without adaptation, and is then evaluated on the Cityscapes images. However, as seen in the table, many different semantic segmentation networks are used, and baseline results vary a lot, even between papers using the same base network. This makes a direct comparison between papers more difficult.