Learning to Adapt: Domain Adaptation with Cycle-Consistent Generative Adversarial Networks

School of Computer Science & Applied Mathematics
University of the Witwatersrand

Pierce William Burke
1172440

Supervised by Dr Richard Klein

August 23, 2023

A dissertation submitted to the Faculty of Science, University of the Witwatersrand, Johannesburg, in fulfilment of the degree of Master of Science in Computer Science.

Abstract

Domain adaptation is a critical part of modern-day machine learning as many practitioners do not have the means to collect and label all the data they require reliably. Instead, they often turn to large online datasets to meet their data needs. However, this can often lead to a mismatch between the online dataset and the data they will encounter in their own problem. This is known as domain shift and plagues many different avenues of machine learning. From differences in data sources, changes in the underlying processes generating the data, or new unseen environments the models have yet to encounter. All these issues can lead to performance degradation.

From the success in using Cycle-consistent Generative Adversarial Networks(CycleGAN) to learn unpaired image-to-image mappings, we propose a new method to help alleviate the issues caused by domain shifts in images. The proposed model incorporates an adversarial loss to encourage realistic-looking images in the target domain, a cycle-consistency loss to learn an unpaired image-to-image mapping, and a semantic loss from a task network to improve the generator's performance. The task network is concurrently trained with the generators on the generated images to improve downstream task performance on adapted images.

By utilizing the power of CycleGAN, we can learn to classify images in the target domain without any target domain labels. In this research, we show that our model is successful on various unsupervised domain adaptation (UDA) datasets and can alleviate domain shifts for different adaptation tasks, like classification or semantic segmentation. In our experiments on standard classification, we were able to bring the models performance to near oracle level accuracy on a variety of different classification datasets. The semantic segmentation experiments showed that our model could improve the performance on the target domain, but there is still room for further improvements. We also further analyze where our model performs well and where improvements can be made.

Declaration

I, Pierce William Burke, hereby declare the contents of this dissertation to be my own work. This report is submitted for the degree of Master of Science at the University of the Witwatersrand. This work has not been submitted to any other university, or for any other degree.

23 March 2023

Acknowledgements

I want to express my gratitude to my supervisor, Dr. Richard Klein, who guided me and offered support throughout the entire process without fail. Their feedback and expertise have been invaluable.

Next, I would like to thank the University of the Witwatersrand School of Computer Science for providing the means for such a research project to even begin. Especially for the constant access to the MS Cluster, most of the computations were performed using High-Performance Computing infrastructure provided by the Mathematical Sciences Support unit at the University of the Witwatersrand. Similarly, I would like to thank the Center for High Performance Computing for also providing access to their cluster, Lengau, which was also used to run experiments. Both of which were critical resources in making this project possible. Furthermore, I would like to thank the National Research Foundation for helping immensely with the funding they provided.

And last but not least, I would like to thank my parents, family, friends, and loved ones for their continuous support throughout this project. Even during the difficult times caused by the pandemic, their unwavering support helped shape this dissertation into what it is today.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The introduction of deep learning models has taken the data-driven world by storm, leading to a massive increase in the amount of usable data required. While the internet age has created no shortage of data, the amount of easily usable data is far less available. The main reason for this massive gap is that raw data is often unstructured and unlabeled, requiring extra work for labeling the data or more sophisticated models which can work with the unstructured and unlabeled data. Even with the apparent benefit of having large amounts of valuable data, the cost and effort associated with labeling so much data are infeasible in practice [Liang *et al.* 2022].

To address the high demands for usable labeled data for supervised learning, practitioners have led their attention to other methods for reducing the amount of labeling required to achieve their results. One such method involves using a similar or related prelabelled dataset to improve or augment their own. While this often shows promising results, it can lead to an issue where the slight differences between the training and test data cause performance degradation in the final model. This is known as domain shift. Even though we try to reduce any inherent bias in the dataset, it is impossible to eliminate all of it, leading to particular patterns in the training data biasing the model and causing poorer performance on the final testing data.

Domain shift is a common problem when applying machine learning to real-world applications. Many potential factors in the data collection process can cause discrepancies between the training and testing data. Some potential causes include data being captured differently, like using a phone camera instead of a DSLR camera. The data might come from different users, like how a spam filter trained on one user's emails would not necessarily work for a different user. Different times of year might have different distributions for the data, as traveling patterns of people during school holidays are very different from normal times of the year. Another example can be seen in how the COVID pandemic affected how the world behaved; most movements were restricted, people's shopping habits were altered, and many more aspects of human life changed. Many models using data from pre-covid times in one of the affected areas

would have had to be adjusted to account for the discrepancy caused by the global and lifestyle changes caused by the pandemic. Considering how many different facets of machine learning this problem affects, this problem warrants further exploration to minimize the problems caused by domain shift.

While any machine learning problems can be affected by domain shift, visual domains are especially susceptible as even small inconsequential changes in how a human perceives an image can significantly affect a deep learning model's performance. The extreme case of this can be seen when applying adversarial noise to an image to fool classifiers [Szegedy *et al.* 2013]. While these small perturbations were designed to fool the classifier, they show how even small changes unnoticeable to humans can have a large effect. Reducing this issue of domain shift in visual domains is the main aim of this research. In our experiments, we focus on extending the image-to-image translation networks proposed by Zhu *et al.* [2017] to bridge the gap between domains and improve the efficacy of the models. Their models were shown to be able to transfer stylistic differences in images between two domains with very little supervision required, allowing them to perform style transfer, object transfiguration and other image-to-image tasks. In this work, we will propose a solution to the domain shift problem which builds off of the work of Zhu *et al.* [2017] and Hoffman *et al.* [2018]. We evaluate the efficacy of the standard CycleGAN architecture when being applied to unsupervised visual domain adaptation and highlight the benefits and potential downfalls. We then provide our own proposed alterations to address these problems and evaluate how it performs on the downstream tasks for classification and semantic segmentation on a variety of datasets.

The rest of this dissertation is structured as follows; Chapter 2 outlines background information that will be required for the rest of the thesis, chapter 3 provides other related work that has been done in the field for visual domain adaptation specifically, chapter 4 summarises our research questions and the methodology used. Chapter 5 further explains CycleGANs. Chapter 6 presents our extension on the traditional CycleGAN to apply it to unsupervised domain adaptation. Chapter 7 presents the application of our technique to the classification of small digits using the MNIST, SVHN and USPS dataset, while chapter 8 presents the extension of this work to the more complex semantic segmentation tasks.

Chapter 2

Background

2.1 Introduction

This chapter gives an in-depth look at some of the ideas and concepts that are used in the rest of our work. First, we introduce deep learning and highlight some of the main techniques to be utilized. Next, section 2.3 covers domain adaptation and its different types. Section 2.4 addresses generative adversarial networks, the main component underlying our models, and the technique we use to solve the domain shift problem. Section 2.5 covers the different network architectures used.

2.2 Deep Learning

With the rapid developments in high-performance computing components, specifically the Graphics Processing Units, and the amount of readily available data to researchers, more and more focus is shifting towards using deep learning approaches to solve problems. The difference between deep learning models and traditional machine learning is that deep learning models are usually comprised of multiple layers that can learn representations directly from the data instead of relying on domain knowledge to design hand-crafted features directly. This is exceptionally beneficial in domains with complicated or very high dimensional state spaces, such as vision tasks or speech recognition [LeCun *et al.* 2015]. By learning the representations themselves from the data instead of inserting them in by humans, we eliminate human biases and encourage the models to learn state spaces that the models can use best. The new deep learning models which have emerged have greatly improved state of the art across many fields. The main component that has led to such advancement has been the Artificial Neural Network and the Deep Artificial Neural Network.

2.2.1 Neural Networks

Artificial Neural Networks (ANN) are mathematical models which act as universal function approximators [Hornik *et al.* 1989]. The design and structure of ANNs were inspired by the natural neural pathways found in living organisms' brains [Sharma *et al.* 2012]. The ANN comprises a collection of artificial neurons (nodes) and connections (weights). The first layer of nodes takes in the inputs of the problem weighted by some weights in a vector form. After each node calculates its corresponding response value, it is passed as the next layer's input. This process is repeated for each layer in the network propagating the signal forward until we reach the output layer. An example of this structure can be seen in figure 2.1. By chaining many of these layers together, artificial neural networks are capable of learning progressively more abstract and complicated features by constantly applying non-linear transformations between each layer. The idea of deep learning comes from using networks with many layers to learn better representations [Eldan and Shamir 2016].

To train these neural networks, we take the output response of the network and compare it to the expected response (supervised learning). By comparing these two results, we can calculate an error based on some loss metric and use this error to calculate the gradients for all the weights, which would best optimize the network for that response. Using the chain rule, the gradient is calculated and back-propagated through the network to update every neuron's weights allowing the network to learn by updating the weights to reduce the overall error at the output. The assumption is that the training dataset is representative of the final task we are trying to solve and these learned representations/features adequately describe the input data when the model is deployed to unseen data.

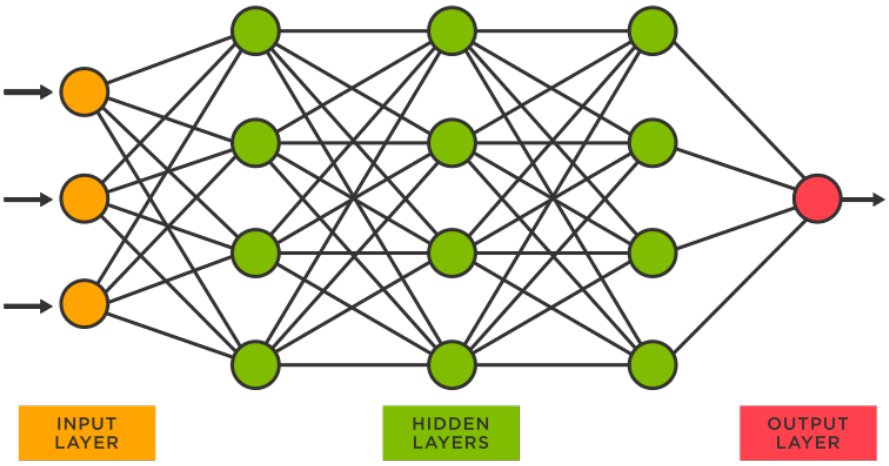


Figure 2.1: Example of an artificial neural network architecture diagram.

2.2.2 Convolutional Neural Networks

Convolutional layers for neural networks were first proposed by LeCun *et al.* [1998] as a way to reduce the number of parameters, encourage translation invariance and reduce overfitting. They were later popularised when the model AlexNet proposed by Krizhevsky *et al.* [2017] dominated the 2012 ImageNet Large Scale Visual Recognition Competition [Russakovsky *et al.* 2015]. This represented a pivotal moment in deep learning when CNN's first outperformed the available techniques at the time. The convolutional layers replace the typical fully connected layer with convolutional kernels.

The convolutions in a CNN calculate a feature map by sliding the kernels across the data and applying the convolution operation as seen in figure 2.2. The weights of these convolutions are calculated using the standard back-propagation method. However, the number of parameters is drastically reduced as the same convolutions are used across the entire image. Another benefit of sharing the same weights across the entire image is that it allows the network to learn generalized features that apply to the entire image. It does not matter where in the image an object appears as the same kernels are used everywhere; this translation equivariance is what makes them so appealing [Li *et al.* 2021]. By chaining many of these layers together, we can learn richer and more robust features from the images while also reducing the number of parameters needed overall. Fully Convolutional Neural Networks [Long *et al.* 2015a] are when the entire network only consists of these convolutional layers, even the final prediction layer. These networks also have the extra benefit of being able to take in different input sizes.

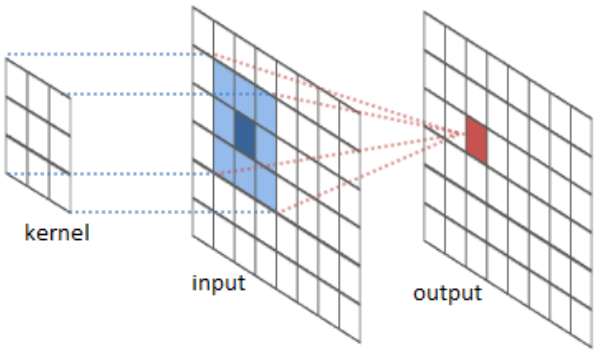


Figure 2.2: Example of a convolution kernel applied to an input image.

2.3 Domain Adaptation

A significant problem with relying on learning all the information from the data is that the data might be biased in specific ways [Wang and Deng 2018]. These biases might

manifest in poor performance when using the same model to classify different variations of the same data. These variations can occur when the generation process which produced the data has changed in some way, such as visual data captured in different lighting conditions or satellite image data taken from different locations worldwide [Wang and Deng 2018]. Another potential cause for the variation in the data is when the data is non-stationary, causing changes in the data with time. One example of non-stationary data is how the yearly seasons affect the world between summer and winter. While humans can usually quickly adapt to the domain shift caused by the variations in the data, most machine learning models will suffer some form of performance drop due to these slight changes. This has led to an interest in developing domain adaptation techniques that are either invariant to such domain shifts or methods capable of transforming the data so that this shift is no longer a problem. This has significant importance for the entire field of machine learning as it allows for more robust methods which are capable of generalizing to new and unseen environments better.

2.3.1 Types of domain adaptation

The general definition of domain adaptation can be given by the following. If we consider a domain $\mathcal{D}$ as being a feature space $\mathcal{X}$ and a marginal probability distribution $P(X)$, where $X = \{x_1, ...x_n\} \in \mathcal{X}$. Then $\mathcal{D} = \{\mathcal{X}, P(X)\}$. In the standard machine learning formulation, our goal is to learn a conditional probability distribution $P(Y|X)$. However, in the context of domain adaption, we assume that we have two separate but related domains, the source domain $\mathcal{D}_s = \{\mathcal{X}_s, P(X_s)\}$ and the target domain $\mathcal{D}_t = \{\mathcal{X}_t, P(X_t)\}$. With the assumption that the two domains' marginal probability distributions may differ so that $P(X_s) \neq P(X_t)$. Using the information provided by the source domain $\mathcal{D}_s$, we want to learn the conditional probability distribution for the target domain $P(Y_t|X_t)$.

The field of domain adaptation has many different sub-problems and types that we can explore. While there are many potential ways to classify the different types of domain adaptation, we will follow the hierarchy as given by Wang and Deng [2018]. The first differentiation in types of domain adaptation problems comes from the level of supervision. The three major levels of supervision are supervised, semi-supervised and unsupervised.

Supervised Domain Adaptation

In the supervised domain adaptation case, the target domain has a small amount of labeled training data, which is insufficient to train a model fully. Therefore we have to use the labeled source domain data to help in training. This is a popular use case when a large labeled dataset like ImageNet [Deng *et al.* 2009] is used in conjunction with a much smaller dataset.

Semi-Supervised Domain Adaptation

The semi-supervised domain adaptation problem is similar to the supervised one because it has some labeled data, but only a small fraction of the total amount of data is labeled. The goal of the semi-supervised approach is to leverage the small amount of labeled data to improve the information that can be garnered from the unlabeled data. This allows the practitioner to learn and exploit some structural information about the data to try and maximize results. An example of this is when someone has collected a dataset on hand-written digits, but only has the resources to label 10% of the images. Instead, they can utilize a larger labeled dataset like MNIST to help in the training process, while leveraging the few labeled examples to better utilize the unlabeled examples.

Unsupervised Domain Adaptation

The last domain adaptation problem type is the unsupervised problem set. In this problem, we have no labels in our target domain, and all the label data must be learned from the source domain. The primary way of overcoming this hurdle is by learning some shared representation between the two domains such that the labels can be correlated across this shared representation space. One such unsupervised domain adaptation problem we look at in this study is the mapping between the MNIST and USPS datasets. Both are black-and-white handwritten datasets with similar characteristics. In the unsupervised case, we use the labels from the MNIST dataset to train a model that can classify the USPS images without any labels. The unsupervised domain adaption case is the area of domain adaptation we focus on in this work.

2.4 Generative Adversarial Networks

While the rapid growth of machine learning has helped solve many big data problems, the goal of image synthesis is still an extremely challenging task. This is partly because many models work on a discriminative basis that excels in classifying data, but not necessarily generating the data [Wu *et al.* 2017]. Goodfellow *et al.* [2014] introduced a novel idea that could learn to generate unseen data with minimal supervision. This breakthrough was called a Generative Adversarial Network (GAN).

This method introduces a way of learning the loss function through adversarial learning. It uses a second neural network called a discriminator to act as the adversary, while the generator aims to produce realistic data to deceive the discriminator. The generator's goal is to learn a mapping from some vector space, usually random noise, to an image that looks like it belongs to the domain we are trying to generate. The discriminator, on the other hand, has to learn to classify the difference between the fake

images generated by the generator and the real images we are trying to recreate. This locks the two networks in a constant adversarial game where the generator tries to fool the discriminator, and the discriminator tries to classify the images from the generator correctly. A significant benefit to GANs is that they are inherently unsupervised, as only the data from the domain we are trying to reproduce is required.

Formulation

The original GAN formulation is derived from the cross entropy loss between the real and generated domains,

$$\min_G \max_D L(G, D, x, z) = \mathbb{E}_{x \sim p_d}[log(D(x))] + \mathbb{E}_{z \sim p_g}[log(1 - D(G(z)))] \tag{2.1}$$

where:

- L is the GAN loss function,
- x is the real data,
- z is the input noise,
- $D(x)$ is the output from the discriminator on real data,
- $G(z)$ is the image generated from noise,
- $D(G(z))$ is the output of the discriminator on the generated/fake data,
- p_d is the data distribution,
- p_g is the generator's input distribution.

From the above, we can see that the two networks fall into a min-max game where the discriminator is trying to maximize the likelihood of correctly classifying an image as real or fake, and the generator is trying to minimize it. This leads to these two networks playing a zero-sum game where the optimal solution is found when the Nash equilibrium [Osborne and others 2004] is reached between these two networks. The original paper showed that minimizing the GAN loss function with an optimal discriminator is equivalent to minimizing the Jensen–Shannon divergence between the two data distributions.

Issues with GAN

While GANs were an interesting new technique that showed promising results, they were not without their faults. GANs became notorious for being difficult to train as they often suffered from issues such as mode collapse, non-convergence, vanishing gradients, sensitivity to hyper-parameter choices, and instability. Many of these issues stem from the fact that while there are theoretical guarantees for convergence, these often rely on assumptions that cannot be met in the real world and end up failing when empirically analyzed [Goodfellow *et al.* 2020].

One of the problems with modeling highly complex data domains such as sets of images is that they are often multi-modal. This can lead to issues where the generator model only captures specific modes in the distribution while leaving out some other ones. In the standard GAN architecture, the loss function cannot correct this issue because it only penalizes the generator for using non-realistic images and not for modeling the entire data space. This problem is called mode collapse [Saxena and Cao 2021]. A more extreme case can occur when the generator learns to map all inputs to a very limited set of outputs. If these outputs are enough to fool the discriminator, the entire system can fall into a degenerate solution where nothing of use is learned [Goodfellow 2016]

While the technique of using two separate networks to train in an adversarial way is what bought about this innovation in the topic of data generation, it also leads to some of the instability issues that arise from training GANs. Some of these can be attributed to the dynamics of having two competing loss functions acting against each other, leading to potential issues with the GAN's training. The other issue is if the discriminator gets sufficiently good at classifying all the images generated by the generator with high accuracy, the gradients passed on to the generator diminish and get close to 0 [Saxena and Cao 2021]. This leads to the generator being unable to learn anything and the training cycle effectively halts. This can frequently happen as learning to classify an image as real or fake is easier than generating an entire image from noise.

2.4.1 GAN loss variants

Since the introduction of the GAN paper by Goodfellow *et al.* [2014] in 2014, extensive research has been conducted to address the issues we outlined above. This section covers some of the different loss functions introduced to improve the GAN paradigm and alleviate the accompanying issues.

Modified min-max loss

In the original paper on GANs [Goodfellow *et al.* 2014], a simple modification to the loss function was suggested as a potential way to reduce the issue of the vanishing gradient behavior encountered with the conventional min-max loss function. By changing the second term from $\mathbb{E}_{z\sim p_g}[log(1 - D(G(z)))]$ to $-\mathbb{E}_{z\sim p_g}[log(D(G(z)))]$ the generator receives strong gradients. This change was called the modified min-max loss or the non-saturating loss.

Least Squares GAN

The standard implementation of a GAN usually puts a sigmoid function as the final activation for the discriminator. Mao *et al.* [2017] propose replacing the traditional cross-entropy loss function with a least squares loss function. The new proposed formulation becomes:

$$\begin{aligned} \min_D V_{LSGAN}(D) &= \frac{1}{2}\,\mathbb{E}_{x\sim p_d}[(D(x) - b)^2] + \frac{1}{2}\,\mathbb{E}_{z\sim p_z}[(D(G(z)) - a)^2] \\ \min_G V_{LSGAN}(G) &= \frac{1}{2}\,\mathbb{E}_{z\sim p_z}[(D(G(z)) - c)^2] \end{aligned} \tag{2.2}$$

where:

- $V_{LSGAN}(D)$ is the Discriminator's loss function,
- $V_{LSGAN}(G)$ is the Generator's loss function,
- x is the real data,
- z is the input noise,
- $D(x)$ is the output from the discriminator,
- $G(z)$ is the image generated from noise,
- p_d is the data distribution,
- p_g is the generator's input distribution.
- a is the encoding for the fake data,
- b is the encoding for the real data,
- c is the value that G wants D to predict for fake data.

Formulating the problem in this way has been shown [Mao *et al.* 2017] to be equivalent to minimizing the Pearson $\mathcal{X}^2$ divergence between $p_d + p_g$ and $2p_g$, if correct values for a, b, and c are chosen. One such set of values for a, b, and c that fulfills these constraints is when $c = b = 1$ and $a = 0$. Which gives the following objective functions:

$$\begin{aligned} \min_D V_{LSGAN}(D) &= \frac{1}{2}\,\mathbb{E}_{x\sim p_d}[(D(x)-1)^2] + \frac{1}{2}\,\mathbb{E}_{z\sim p_z}[(D(G(z)))^2] \\ \min_G V_{LSGAN}(G) &= \frac{1}{2}\,\mathbb{E}_{z\sim p_z}[(D(G(z))-1)^2] \end{aligned} \tag{2.3}$$

It should be noted that there are other variations for the parameters a, b, and c, but the original authors found similar performance.

2.4.2 Deep Convolutional GAN

After the rapid interest that grew from the original GAN paper by Goodfellow *et al.* [2014], much work was put into finding different architectures that could improve the original formulation even more. One such idea was to adopt ideas from Convolutional Neural Networks that have seen success in other computer vision-driven tasks. Most of these attempts at using CNNs as a generator architecture had yet to be successful as they often failed to scale up. Radford *et al.* [2015] propose a new architectural design with guidelines that allow a fully convolutional neural network to act as the generator. These guidelines can be listed as follows:

- Do not use any pooling layers; instead, use strided convolutions and fractional-strided convolutions.
- Use batch normalization in both the generator and discriminator.
- Do not use a fully connected layer for deep networks.
- Use ReLU for all layers in the generator except for the final layer, which should be a Tanh activation.
- Use LeakyReLU for all layers in the discriminator.

By adopting these five guidelines, they found they could use a CNN-based generator on various datasets like the Large-scale Scene Understanding (LSUN) [Yu *et al.* 2015], ImageNet-1k [Deng *et al.* 2009], and a Faces dataset they created.

2.4.3 Conditional GAN

A downfall to the typical generation progress of standard GAN architectures is the lack of control over the type of image being generated. An example of this is in the MNIST dataset [LeCun *et al.* 1998], there are ten different classes, one for each digit between zero and nine. A GAN trained in the standard way to generate images from this domain would arbitrarily generate images of each class with no possible way to choose the output type. Mirza and Osindero [2014] proposed conditioning the two networks during training to add control to which outputs should be expected from the generator. The proposed technique achieves conditioning by incorporating the label y as part of the inputs for both the generator and the discriminator. The generator takes this additional input by combining the input noise vector z with the encoded label y. This process can be seen in figure 2.3. The discriminator then uses the generated image from the generator and the input label to make its final decision. This allows the discriminator to reject any samples produced by the generator that does not match the label provided, forcing the generator to learn specific classes of the data. By adding the label information, we convert the problem from unsupervised to supervised. However, with this trade-off, the generator can generate specific classes from the data. This objective function for the GAN becomes:

$$\min_G \max_D L(G, D, x, y, z) = \mathbb{E}_{x \sim p_d}[log(D(x|y))] + \mathbb{E}_{z \sim p_g}[log(1 - D(G(z|y)))] \tag{2.4}$$

where:

- L is the GAN loss function,
- x is the real data,
- y is the label for the given data,
- z is the input noise,
- $D(x|y)$ is the output from the discriminator given some label,
- $G(z|y)$ is the image generated from noise given some label,
- $D(G(z|y))$ is the output of the discriminator on the generated/fake data given some label,
- p_d is the data distribution,
- p_g is the model distribution.

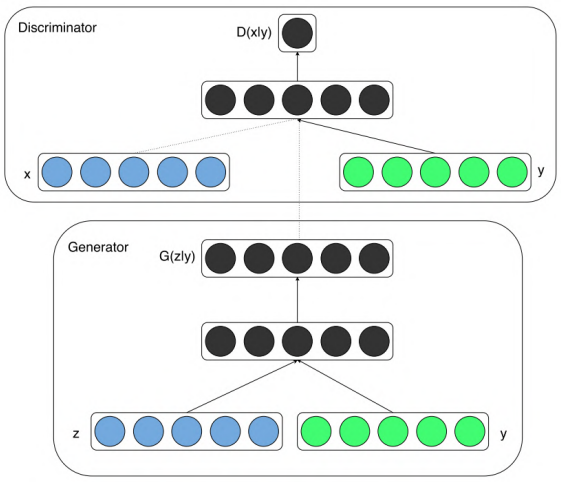


Figure 2.3: Example of a Conditional GAN architecture. The blue vectors represent the input and the green vectors are the labels.

2.4.4 Image-to-Image Translation

Image-to-Image translation is a sub-problem in computer vision that aims to take some source image in a specific domain and transform it into another image from a target domain. The goal of this is to be able to keep some core characteristics or information from the source image but still transform other parts of it to match the target domain. A typical example is style transfer, where we want to take some image, usually a real-world image, and transform it to match some artistic style from a specific genre or artist [Gatys *et al.* 2015].

The applications of this are more comprehensive than just generating art. They can also be used for applications like creating depth maps from RGB images [Isola *et al.* 2017], image super-resolution [Saharia *et al.* 2022], image colorization [Isola *et al.* 2017], semantic segmentation [Chen *et al.* 2017b], domain adaptation [Liu and Tuzel 2016], and many more. While the standard GAN architecture is not suited precisely Image-to-Image translation, some similarities can be made. For example, in the GAN formulation, we take some noise vector z from some random distribution that represents our source latent space and transforms it into a target image by applying $G(z)$. This is the typical image generation process that the standard GAN goes through. However, suppose we replace the random noise distribution where z gets drawn from by a distribution represented by an image from our source domain. In that case, we could generate images that appear from the target distribution while encoding some properties from the source distribution.

Pix2Pix

A significant issue with most standard image-to-image translation methods is that they need specifically crafted loss functions, architectures, or methods to perform the transformation, and it usually only works for some specific problem. An example case is the Neural Style Transfer algorithm [Simonyan and Zisserman 2014], which involves utilizing outputs from different layers from a deep convolutional network, like VGG-16. These outputs have been found to correspond to what humans consider to be "style" and "content". Allowing us to combine the style of one image with the content of another. This entire algorithm had to be finely tuned to figure out which layers in the network capture style and which networks capture content. While it is exceptional at producing style-transferred images, it does not work well for other image-to-image problems.

To combat this lack of generalization, Isola *et al.* [2017] proposed a method Pix2Pix, that uses Conditional GANs to perform image-to-image translation. The primary benefit of using this method over the older methods is that its loss function and architecture are not directly linked to the problem but are generalizable. The first major component of this approach is that it generates images from other images instead of using a noise vector. To do this, they needed to constrain the generator in some way to enforce that it maintained the relevant structural information and features from the source image while still translating the image to the target domain. This is where the conditioning of the GAN is useful. By feeding the discriminator with the expected target image, the generator can be forced to keep the important information we want from the source domain. This method is highly effective, as it generates high-quality results across a wide range of datasets, with figure 2.4 illustrating a few of its use cases. This demonstrates its versatility compared to earlier methods, which often required significant manual adjustments or specialized architectures and loss functions to achieve similar results.

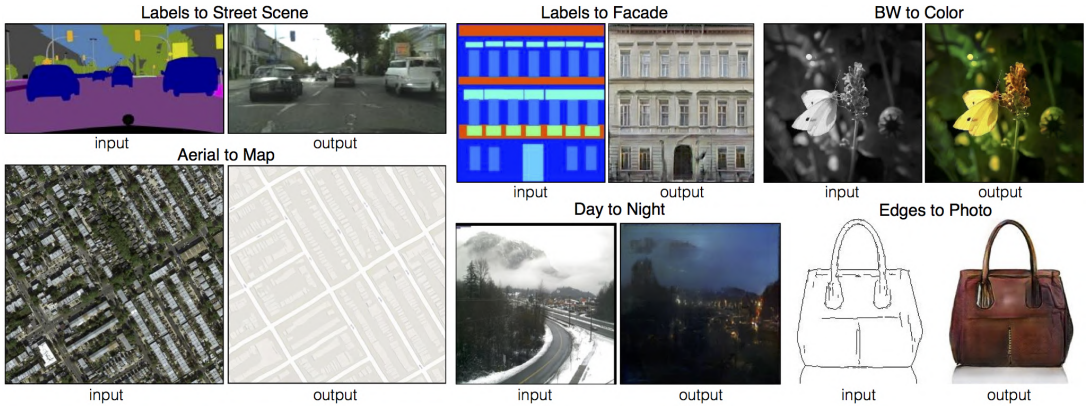


Figure 2.4: Qualitative results taken from the pix2pix paper

Cycle Consistent Generative Adversarial Network

The Pix2Pix model was a significant advancement in the field of versatile image-to-image translation, but it was not without its problems. While effective for specific tasks such as semantic segmentation or photo colorization where there is a direct correspondence between the image and label, this method may not be practical or feasible for other types of tasks or domains. This is because it requires the availability of both the image and the corresponding label in the dataset, which may be challenging to acquire for other types of tasks or domains. Such an example of this would be the style transfer problem, some paintings have been created from actual objects or physical places, but it is infeasible to get a real-world photo-realistic version of every painting an artist has created.

In an attempt to rectify this problem, two independent groups Zhu *et al.* [2017] and Yi *et al.* [2017] proposed a solution inspired by the concept of the cycle consistency constraint used in language translation [He *et al.* 2016a]. This constraint enforces that if a sentence or word is translated from language X to language Y, translating it back to language X from language Y should give the same starting point. In the context of images, we can use a GAN to map an image from domain X to domain Y and then have a similar but opposite GAN trained to map an image from domain Y to domain X. Using the original image and the image that has gone through both GANs we can calculate the difference between the two to obtain a reconstruction loss. This creates a reason for the models to maintain the contextual information needed to recreate the original.

To achieve cycle consistency in the model, Zhu *et al.* [2017] add an additional loss term to the typical GAN loss function. The extra term is given by:

$$L_{cyc}(G, F, x, y) = \mathbb{E}_{x\sim X}\left[\, \|F(G(x)) - x\|_1 \,\right] + \mathbb{E}_{y\sim Y}\left[\, \|G(F(y)) - y\|_1 \,\right] \tag{2.5}$$

where:

- x is a sample from domain X,
- y is a sample from domain Y,
- $\mathbb{E}_{x\sim X}$ is expected value across all samples from domain X,
- $\mathbb{E}_{y\sim Y}$ is expected value across all samples from domain Y,
- $\|\cdot\|_1$ is the L1 norm of a vector,

By incorporating this term the two GANs are encouraged to maintain semantic information about the images so they can be reconstructed in both domains to maintain cycle consistency. We further expand on the CycleGAN's in chapter 5 for a more in-depth explanation and analysis.

2.5 Model Architectures

In this section, we cover some of the different network architectures that are relevant to this work. These architectures are used for classifiers, discriminators or generators in the chapters that follow.

2.5.1 LeNet

LeNet was one of the first examples of a successful convolutional neural network. In the paper LeCun *et al.* [1998], they showed that by reducing the number of free parameters with convolutional layers, they could achieve much better generalization and outperform all other methods on the digit classification task. The architecture initially proposed for LeNet-5 was:

- Layer 1 consists of six convolutional kernels with a size of 5×5 and a sigmoid activation.
- Layer 2 consists of an average pooling layer with a kernel size of 2×2.
- Layer 3 consists of a convolutional layer with 16 5×5 kernels and a sigmoid activation.
- Layer 4 consists of an average pooling layer with a kernel size of 2×2
- Layer 5 Is a fully connected layer with 120 neurons with sigmoid activation.
- Layer 6 Is the second fully connected layer with 84 neurons with sigmoid activation.
- Layer 7, the final output layer, is a fully connected layer with ten fully connected neurons (number of classes) and softmax activation.

While this network architecture is very simplistic compared to many modern-day CNN architectures, it can still be seen as the starting point for a lot of today's success in the field of vision. We will use a variation of the LeNet5 model proposed by Hoffman *et al.* [2018] as our baseline classifier for the small digits experiments.

2.5.2 DTN Classifier

We denote the classifier used by Hoffman *et al.* [2018] as their task network for their SVHN→MNIST experiments as the DTN Classifier. The DTN Classifier has more parameters than the LeNet variation described above and utilizes more modern-day layers like dropout and batch normalization layers. The architecture is as follows:

- Layer 1 consists of a convolutional layer with 64 5×5 kernels. The output is then fed into a batch norm layer, dropout, and a Relu activation function.
- Layer 2 consists of a convolutional layer with 128 5×5 kernels. The output is then fed into a batch norm layer, dropout, and a Relu activation function.
- Layer 3 consists of a convolutional layer with 256 5×5 kernels. The output is then fed into a batch norm layer, dropout, and a Relu activation function.
- Layer 3 is then connected to a fully connected layer with 512 neurons. These are then fed into a batch norm layer and a Relu activation.
- The last layer is a fully connected layer with ten neurons for each output class.

The DTN Classifier outlined above has more parameters in the convolutional and fully connected layers. In conjunction with the added batch normalization and dropout, layers allow for higher potential capacity than the LeNet variation. The DTNClassifier is another classifier we will use in our small digits experiments, allowing us to test our model on a variety of task networks.

2.5.3 SE Classifier

The last classifier we will use for our classification models will be a classifier introduced in a paper called Self Ensemble Visual Domain Adaptation by French *et al.* [2017]. This classifier was used for their MNIST⇔USPS experiments. For brevity, we will call it the **S**elf **E**nsemble classifier or SE Classifier for short. The network consists of the following:

- Layer 1 is a convolutional layer with an output of 32 feature maps and a 5×5 kernel. The output is fed into a batch normalization layer, a 2×2 max pool layer, and a Relu activation.
- Layer 2 consists of another convolutional layer with an output of 64 feature maps with a 3×3 kernel. The output is also fed into the batch norm, max pool, and Relu activation.
- Layer 3 is the final convolutional layer with an output feature map of 64 channels with a 3×3 kernel. It is followed by a batch norm, max pool, dropout, and Relu activation.
- Layer 4 is the first of the fully connected layers, with the outputs from the convolutional layer being fed into a linear layer with 256 nodes. The linear layer also has dropout and a Relu activation.
- layer 5 is the final output layer. This is a fully connected layer with the number of outputs equal to the number of classes. A sigmoid function follows this if it is needed.

2.5.4 VGG FCN

While the fully convolutional neural networks had seen much success in traditional image classification problems, they had not yet been expanded onto dense prediction tasks such as semantic segmentation. That was until Long *et al.* [2015a] proposed taking the previously powerful FCN networks and replacing their final classification layers with upsampling layers to give them the ability to perform semantic segmentation.

By using these networks, which have been pre-trained on massive datasets, they can utilize the early feature layers as a good starting point for the problem they are trying to solve. Firstly the fully connected layers often found at the end of standard CNNs can be thought of as a convolutional layer where the kernel covers the entire input region. Next, they add layers to the end, which are used for upsampling the dense feature maps to the output size required. There are a variety of upsampling techniques that can be used. However, the ones used in the original were bilinear upsample filters, which can also have their weights learned during the backpropagation process. A final improvement they make to the network is to allow skip connections from the earlier downsampling layers to the later upsampling layers to allow for more information and gradients to propagate through the network.

This methodology of using convolutional layers through the entire network allows for the network to maintain a small number of free parameters and allows for it to work for any arbitrary size input. This technique can also be trained end to end in a supervised manner, as every network section is fully differentiable. In our experiments we use an altered version of the VGG network to make it an FCN. We adapted the model from Hoffman *et al.* [2016] to use as the dense classifier in our semantic segmentation experiments, but it was originally proposed in Yu and Koltun [2015].

2.5.5 U-Net

The U-Net is a deep learning architecture based on the fully convolutional network usually used for image segmentation problems. It was first proposed by Ronneberger *et al.* [2015] as a deep learning architecture to perform supervised biomedical image segmentation with a better sample efficiency than other traditional methods.

The network architecture is developed in a way that has a contracting section of the network that downsamples the original image to its lower-dimensional feature map with the standard convolutional layers. This is followed by the next section of the network consisting of upsampling layers, which use fractionally strided convolutional layers instead of the unpooling layers that some other architectures use. This upsampling section of the network is often seen as symmetrical to the downsampling section, which gives the entire model a U-like shape. Another critical aspect of the U-Net architecture is the use of skip connections between the downsampling layers and upsampling

layers. This allows for the convolutional layers in the upsampling layers to use both the low-level information from the deep feature layers and the higher-level contextual information from the shallower levels while also allowing for deeper networks without the worry of vanishing gradients. Below in figure 2.5, we can see the U-Net architecture and how it got its iconic name from its shape. The U-Net will make up our second task-network to test our models performance on domain adaptation for semantic segmentation.

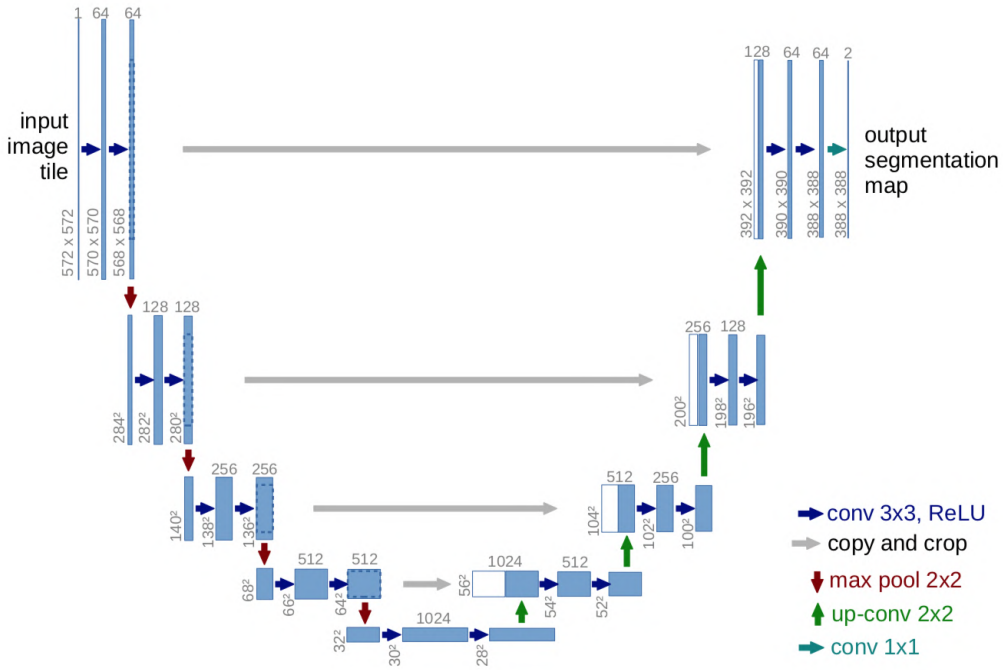


Figure 2.5: U-Net architecture

2.5.6 Residual Neural Network

A significant problem observed with traditional deep neural networks is that while the performance tends to increase as more layers get added, it reaches a point where the model starts to degrade, with issues like non-convergence or decreased performance. One potential reason for this behavior is that as our networks get deeper, the gradients that pass through the network start to shrink to zero as the reapplications of the chain rule can lead to increasingly smaller values [Bengio *et al.* 1994]. This is known as the vanishing gradient problem.

He *et al.* [2016b] propose adding skip connections between blocks in the network to model a residual mapping instead of the underlying mapping. More formally, let a network $\mathcal{F}$ learn the representation $\mathcal{F}(x) = \mathcal{H}(x)$, where x is the input to the layer, $\mathcal{F}(x)$ is the learned representation and $\mathcal{H}(x)$ is the underlying representation. We instead introduce the relation of $\mathcal{F}(x) = \mathcal{H}(x) - x$, which is our function approximator $\mathcal{F}(x)$

learning the residual instead. It is hypothesized in the original paper that the final learned approximator should be asymptotically similar, as $\mathcal{F}(x)$ can learn to offset the difference from the additional term.

By taking the formulation of $\mathcal{F}(x) = \mathcal{H}(x) - x$ the original function becomes $\mathcal{F}(x) + x = \mathcal{H}(x)$. This is done by combing the input of the layer with the output of the layer to create the effect of a mini-skip connection, as seen in figure 2.6. This creates a simple way of implementing the residual function that does not introduce any extra complexity, computation time, or extra weights to learn. In the original paper, they could train networks with over 100 layers successfully and even explore a network with 1000 layers. While the actual shape and size of a ResNet architecture can vary greatly, they showed the success of training a 152-layer network on ImageNet, which was the deepest network at the time, while managing to beat all other networks' performance. In our experiments, we use a residual neural network with 6 residual blocks as the backbone for both generators in the CycleGAN.

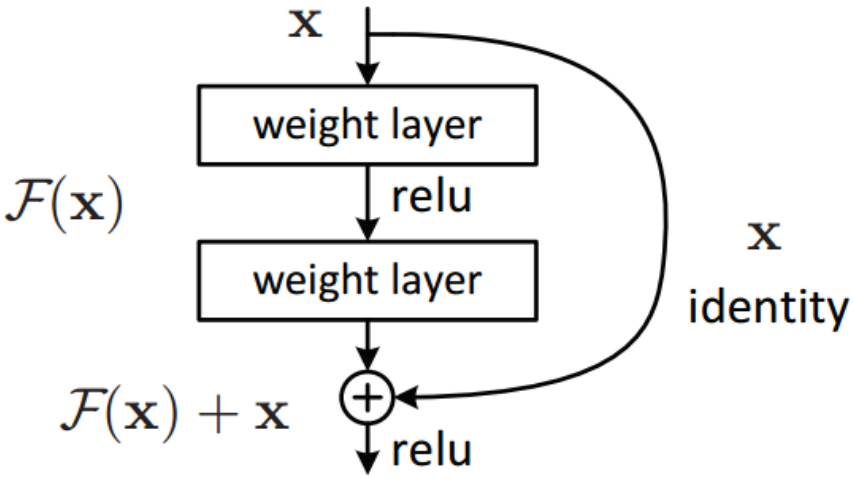


Figure 2.6: Example of a residual block.

2.5.7 DeepLab

The DeepLab architectures are a class of fully convolutional networks designed for semantic segmentation. There are three primary variants of DeepLab, with each subsequent one building off the previous generation of DeepLab. DeepLabv1 [Chen *et al.* 2014] was the first of the DeepLab series to be released, and its novelty that allowed for it to get good results was the use of the atrous convolutions instead of pooling layers in the downsampling section of the network. These atrous convolutions can maintain a sizeable receptive field on the original input without reducing feature resolution. The second major part of the DeepLabv1 model is the use of a fully connected conditional random field to try and incorporate some of the spatial and contextual information based on the surrounding area of each pixel. The DeepLabv2 [Chen *et al.* 2017a] ver-

sion was the second major DeepLab model released; it introduced the use of atrous spatial pyramid pooling (ASPP). The idea behind using ASPP was to improve the network's performance at classifying objects that appear at different resolutions. Finally, the most recent version of DeepLab that has been released is DeepLabv3 [Chen *et al.* 2017b]. DeepLabv3 augments the ASPP module of the network with image-level features to give better global contextual information. They start using depth-wise separable convolutions to improve the computational costs and reduce the number of parameters. Deeplabv3 will make up the final model we will use in our semantic segmentation downstream task.

2.5.8 PatchGAN

PatchGAN was a discriminator architecture introduced in Isola *et al.* [2017] and also utilized in Zhu *et al.* [2017]. The PatchGAN discriminator is a fully convolutional neural network with a receptive field of 70×70 patches. The motivation to create a discriminator architecture of this kind is that when using standard neural networks, they will often learn global structural information in their discrimination process. However, for image-to-image translation, we want to more closely model a style/texture loss more with our discriminator. In this way, the PatchGAN discriminator treats these 70×70 patches across the input image as their independent sections, which can help the attention of the discriminator focus only on local structure and style changes. This process can be seen as equivalent to the image being a Markov random field where all pixels of more than a patch-width away are treated as independent of each other. The PatchGAN is the discriminator we use in all our experiments across all datasets and domains. We use the same model as outlined by Isola *et al.* [2017] and Zhu *et al.* [2017].

2.5.9 Summary

In this chapter, we covered the major ideas and concepts that will be used throughout this dissertation. We covered core aspects of deep learning which included models like artificial neural networks and convolutional neural networks which can learn to solve tasks in a supervised manner. We further discussed how this reliance on learning from data has created issues like domain shift when the training and testing data distributions don't match. This led to the field of domain adaptation which attempts to reduce the impact of domain shift. Finally, we explored the ideas of generative adversarial networks which could learn how to generate data in an unsupervised manner.

Chapter 3

Related Work

This chapter focuses on some of the other methods that have been proposed in the field of domain adaptation. It consists of two sections which focus on shallow domain adaptation and deep domain adaptation methods respectively.

3.1 Shallow Domain Adaptation

This section consists of works done using shallow domain adaptation methods. As defined in Csurka [2017], shallow domain adaptation methods are used when we are working on already extracted features or when we are using methods that have a form of importance sampling on the samples used in training. The ideas discovered in many shallow domain adaptation methods were often precursors that were extended in the deep domain adaptation methods.

One of the earliest methods researchers used to help domain shift was to apply instance re-weighting methods to the problem. Instance re-weighting works by assigning larger weights to certain samples during training. Zadrozny [2004] introduce a separate domain classifier to estimate the likelihood of a sample belonging to either the source or target dataset and using this estimated likelihood as the weighting factor. Sugiyama *et al.* [2007] introduce a novel algorithm based on the Kullback-Leibler divergence called the Kullback-Leibler Importance Estimation Procedure (KLIEP) that estimates the importance of each sample.

Another approach that became very popular was to minimize some metrics between the distributions in some feature space. Huang *et al.* [2006] discusses a method that reduces the means between the features in a Reproducing Hilbert Kernel space (RHKS) called Maximum Mean Discrepancy (MMD). In the same vein, Pan *et al.* [2010] learn a new feature space by using the MMD to learn the optimal transfer components, which

map the datasets into a feature space that minimizes the distribution distances. Sun *et al.* [2016] introduces CORrelation ALignment (CORAL), which aligns the second-order statistics of the datasets by calculating a transformation that whitens and recolors the source data to be more similar to the target distribution.

3.2 Deep Domain Adaptation

Following the success of deep learning in other fields of machine learning, it was only natural to apply these methods for domain adaptation as well. Shortly after the successful usage of CNN models for classification, it was shown that using the feature extraction layers from a deep CNN trained on a massive dataset like ImageNet [Deng *et al.* 2009] could already outperform the previous shallow domain adaptation methods [Donahue *et al.* 2014]. This showed the potential of methods to come, as this increase in performance was achieved with no extra domain adaptation methods applied.

Donahue *et al.* [2014] use the activation features from a CNN trained on a supervised object recognition task to learn a deeper representation that is less prone to biases for a specific dataset. This feature set can be fed directly into a classifier or used as the feature representation for other DA methods. However, the authors note that the datasets they used to train the feature extractor were similar to the target datasets. The performance drops quite significantly when there is a significant discrepancy between domains, like if the target domain is paintings or sketches.

Another method used to learn the feature extractor layers was to train autoencoders and use their early layers to extract features. In a paper by Ghifary *et al.* [2015], they use a multi-task autoencoder trained to reconstruct the image into multiple domains. This is then used as a robust way of extracting features that share a space as all relevant domains. This was similar to the work proposed by Vincent *et al.* [2008] that used denoising auto-encoders to learn a shared feature space. The auto-encoders are trained to denoise a corrupted input image from both the source and target domains, and the encoding layers are then used for the classifier. In a much more recent paper by Wang *et al.* [2021], they use a reconstruction-based method that uses an encoder-decoder network to learn a feature space. This feature space is encouraged to use a Gaussian prior with the use of a KL-divergence penalty. The source and target Gaussian latent spaces are then aligned using a novel regularizer based on the L1-loss. Hoyer *et al.* [2022] utilizes a multi-resolution method that combines both a low-resolution view of the entire image and high-resolution cropped patches. This allows the model to capture global semantic information and detailed local information. This incorporates the benefits of high-resolution image training while using fewer GPU resources.

While many papers tried to improve the feature-extracting layers, some authors used the later layers in the network to perform the adaptation. Long *et al.* [2015b] employs a discrepancy-based method that embeds the later task-specific layers in a network into

a Reproducing kernel Hilbert space(RHKS) and attempts to align the means of each domain embedding. Their theory was that the early feature extraction layers were usually generalized. However, the later layers become more specialized, which causes them to become more susceptible to the performance drop from the domain shift. However, in Aljundi and Tuytelaars [2016], it is suggested that even the early layers can be prone to domain shift. Instead, they recalibrate the early feature extraction layers by analyzing the difference between the activation outputs between the two domains and aligning them to be more similar. A method by Tzeng *et al.* [2014] introduces a novel adaptation layer and domain confusion loss that learns a more domain-invariant feature space. Inspired by the original CORAL paper [Sun *et al.* 2016], they introduce a new loss function that minimizes the different domain correlations. This is done by trying to align the second-order statistics of the feature activation layers in the networks which encourages the network to better align the domains. In French *et al.* [2017], a modified version of the self-ensemble mean teacher model [Tarvainen and Valpola 2017] is used. In this modified version, instead of using labelled and unlabelled samples from the dataset, they use the source domain's labelled samples and the target domain's unlabelled samples.

With the massive success of Generative Adversarial networks introduced in Goodfellow *et al.* [2014], many researchers have been using adversarial learning for more than image generation. One such paper uses a discriminative adversarial approach which utilizes a gradient reversal layer and a domain classifier to learn a feature space that promotes the source and target features to be similar [Ganin *et al.* 2016]. This was followed by Tzeng *et al.* [2017] which uses an adversarial domain classifier to fine-tune a feature extractor to create a shared latent space where the extracted features are indistinguishable. Saito *et al.* [2017] describe a novel technique that utilizes dropout and a critic to fine-tune a feature extractor not to generate features close to the classifier's decision boundaries and thus create a more discriminative feature space. Following the idea of achieving generalizability by staying away from the decision boundary, Saito *et al.* [2018] propose using adversarial training with task-specific classifiers to learn a feature generator that tries to learn features that are not too close to the decision boundaries.

While the above approaches use adversarial learning directly to learn discriminative feature spaces, using standard generative adversarial networks to learn mappings between the domain image spaces has also been explored. In Liu and Tuzel [2016], they employ two GANs for each domain and use a weight-sharing strategy to better learn a joint distribution of the different domains. This method has shown uses in multiple applications, including unsupervised domain adaptation. In Taigman *et al.* [2016], they attempt to learn a mapping between the two domains with a GAN by ensuring that another network, which accepts inputs from both domains, outputs the same result from before and after the mapping. Another generative approach was introduced in Bousmalis *et al.* [2017], which used a standard GAN approach to generate and change the training images to appear like they have been sampled from the target domain to lower the domain shift gap. Following the theme of making the source images appear as they have been sampled from the target dataset, Hoffman *et al.* [2018] utilize the unsuper-

vised image-to-image translation abilities of CycleGAN [Zhu *et al.* 2017] to learn the mappings between source and target domain. They also utilize a task network, similarly to Taigman *et al.* [2016], during training to ensure that semantic meaning is kept during the translation process. Once they have learned a mapping, a modified version of ADDA [Tzeng *et al.* 2017] is performed on the mapped source images. Ghifary *et al.* [2016] uses the idea of giving the classification network, which is trained on the labelled source images, an auxiliary task of reconstructing the target images. By forcing the same feature extraction layers to do both tasks, the network should learn a feature space representative of both domains.

3.2.1 Summary

In this chapter, we have discussed some of the shallow and deep domain adaptation methods that are currently being used in the field and how their underlying mechanisms alleviate the issues caused by domain shifts. In the next chapter, we will explore our deep domain adaptation method using CycleGANs and the associated research questions.

Chapter 4

Research Questions and Methods

In Chapter 2 and Chapter 3 we covered the background work that will be required and methods that other people have put forward in the field of Unsupervised Domain Adaptation. In this chapter we discuss more definitively what our goals are and the approaches we will use to achieve them.

4.1 Aims

The main goal of this research is to evaluate the efficacy of CycleGANs in the application of unsupervised homogeneous visual domain adaptation. Furthermore, introduce the source labels in the training process to improve image generation for the pixel-level adaptation between the source and target domains, where the target domain labels are unavailable. The proposed method should work under various architectures and be robust to as many different domain shifts as possible. Lastly, the final model should minimize the information lost about the source domain during training and should be able to classify both the source and target domains adequately.

4.2 Research Hypothesis

By using CycleGAN to produce images that look like the target domain, we should bridge the gap caused by domain shift and reduce the performance lost. The semantic loss provided by the task network should guide the generators into creating more realistic results and improve the images for the downstream tasks. By training the task network on the images generated by the CycleGAN, the task network will get better accustomed to the target domain and provide better information to the generators. Because the method works on pixel-level adaptation, it should be robust to most domains

that share a reasonably close visual resemblance and work with various architectures.

4.3 Research Questions

- Can CycleGANs be used to reduce the adverse effects of domain shift for classification when we do not have any labels in the target domain?
- Can we incorporate the source labels in the CycleGAN process to better guide or improve the image transformation for better results?
- Can this method be used to bridge the gap between synthetic and real domain sets to improve performance when using synthetic data?
- Can this work be extended onto more difficult problem sets like semantic segmentation?

4.4 Methodology

Now that we have highlighted the exact research questions we aim to answer, we present the methodology we will use to answer these questions. This section is split into the datasets section and the experiments section. In the datasets section, we take a deeper look into all the datasets considered in this work, how we used them and why we chose these datasets in particular. Then finally, we discuss the experimental setup we used to answer the questions listed above.

4.4.1 Model

In this subsection, we give a brief outline of how we alter the standard CycleGAN model to adapt it to the problem of unsupervised domain adaptation. The first major alteration we adopt is by utilizing a task network trained on the source domain to classify the images before and after they have been mapped between the domains. This allows the CycleGAN networks to receive more gradient information directly from the task network as well as encourages the network to maintain semantic meaning between domains. We further train the task network on the generated images from the CycleGAN to improve its performance in classifying the target domain images, which in turn should improve the gradient information provided to the CycleGAN. We further elaborate on this in Chapters 5 and 6.

4.4.2 Datasets

In this subsection, we review all the different datasets used in our research and further discuss some of the attributes associated with each of them.

MNIST

The Modified National Institute of Standards and Technology dataset (MNIST) [LeCun *et al.* 1998] is a large handwritten digit dataset used in character recognition and classification. The dataset consists of a collection of digits ranging between 0 and 9, which were hand drawn by employees from the United States Census Bureau and a group of high school students. This dataset has become one of the most widely used datasets in machine learning, often being the starting point of projects to be extended onto more difficult domains and datasets. The images themselves are black and white and are of size 28×28. The total size of the dataset is 60000 training examples and 10000 testing examples. Figure 4.1 illustrates an example of each digit ranging from 0-9.

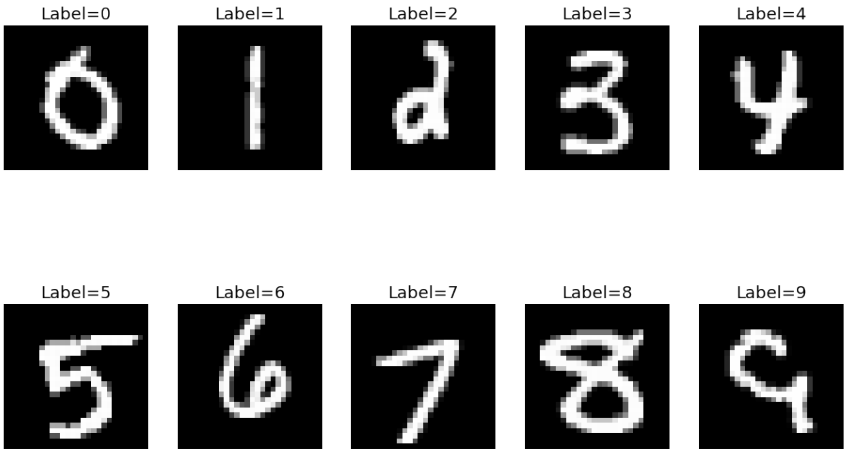


Figure 4.1: Example of each digit from the MNIST dataset.

USPS

The USPS dataset [Hull 1994] is a character dataset collected from scanned envelopes from the U.S. Postal Service. Similar to the MNIST dataset mentioned above, it contains a collection of black and white 16×16 handwritten digits with numbers ranging from $0 - 9$, as can be seen in figure 4.2. This dataset is often used in conjunction with the MNIST dataset to test domain adaptation methods. This is because they share many similarities making knowledge easily transferable, but the slight domain shift is

sufficient to degrade the performance in a meaningful way. We used a training set of 7438 and a test set of 1860 samples in our experiments.

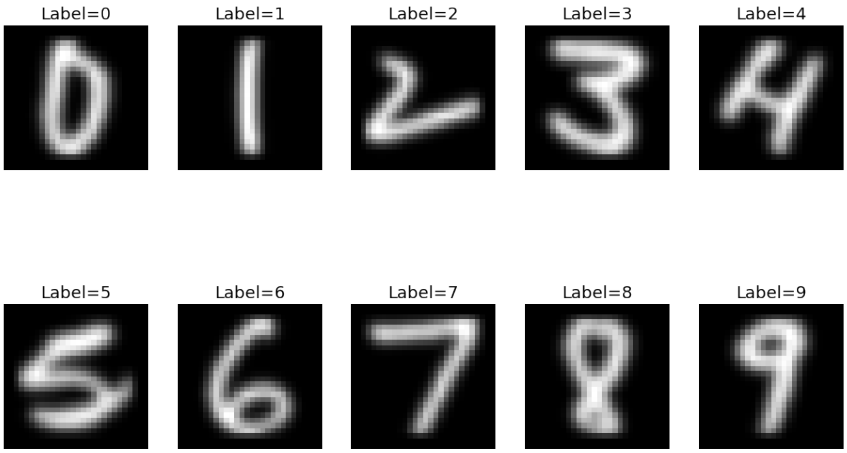


Figure 4.2: Example of each digit from the USPS dataset.

SVHN

The Street View House Numbers (SVHN) dataset [Netzer *et al.* 2011] is a real-world digits dataset that consists of Street house numbers taken from Google street view images. These images are then cropped around the main digit and resized to 32×32. While each number only has a label between $0 - 9$, other distracting digits are kept in the images, which can lead to a much more difficult job for classifiers. This can be seen in figure 4.3 where a few images contain multiple digits. This extra difficulty, paired with the fact that they are taken from a real-world setting and are now colour images, creates a significant domain shift between the other small digit datasets like MNIST and USPS. The dataset consists of 73257 training images and 26032 testing images. There is also an extra training set consisting of 531131 less difficult examples, which can be used for extra training material.

Below in figure 4.4, we can see that that the SVHN dataset is inherently imbalanced. With a very disproportionate amount of samples belonging to label one class. To help this problem we subsample the training set for SVHN to create a uniform class distribution. For the majority of this research any mention of the SVHN dataset will refer to this balanced variation.

Figure 4.3: Example of each digit from the SVHN dataset.

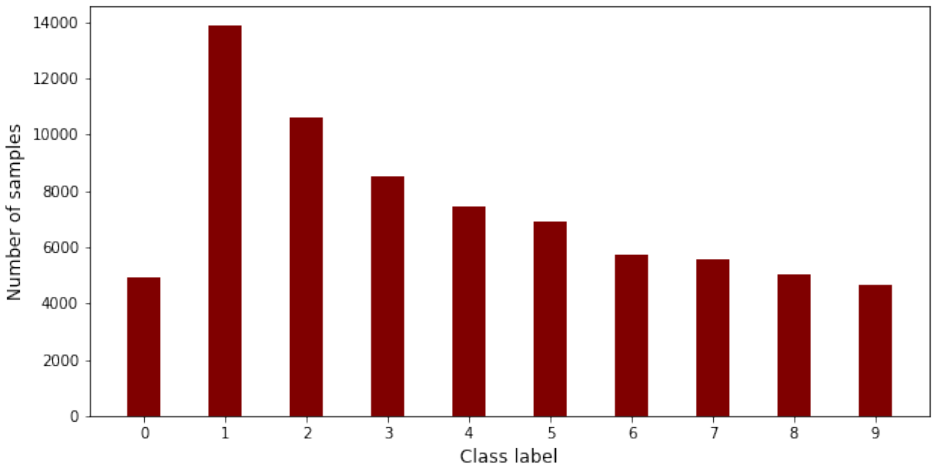


Figure 4.4: Class count for SVHN.

Cityscapes

The Cityscapes dataset [Cordts *et al.* 2016] is a large-scale dataset whose focus is on urban understanding. These city scene images are annotated with labels for commonly occurring objects in typical city settings, including cars, buildings, people, signposts, and more. The dataset consists of multiple types of labels that could be used for various problems. These include pixel annotations for semantic segmentation, panoptic semantic labeling, bounding boxes, and instance-level segmentation for people and cars.

The image scenes were captured from 50 different European cities, with the large

majority of them coming from Germany. All the images were captured during the day in good quality weather conditions, with no heavy storms, snow, or other weather conditions that could obstruct visibility. While there are no adverse conditions, the dataset still contains a high variability in setting as the images are captured at different times of the year, cities, times of day, and in different weather conditions. For the pixel annotations for semantic segmentation, the dataset consists of 5000 high-quality annotations and an additional 20000 images, which only contain coarse annotations. Of the 5000 high-quality annotated images, 2975 are in the training set, 500 are in the validation set, and 1525 are in the test set. Below in figure 4.5, we can see an example of a high-quality annotated image from the dataset.

In table 4.1 we show the list of all classes, with each class belonging to a group:

Table 4.1: Table of all types of classes contained in the Cityscapes dataset including the group they belong to.

Group	Classes
Flat	road, sidewalk, parking, rail track
Human	person, rider
Vehicle	car, truck, bus, on rails, motorcycle, bicycle, caravan, trailer
Construction	building, wall, fence, guard rail, bridge, tunnel
Object	pole, pole group, traffic sign, traffic light
Nature	vegetation, terrain
Sky	sky
Void	ground, dynamic, static

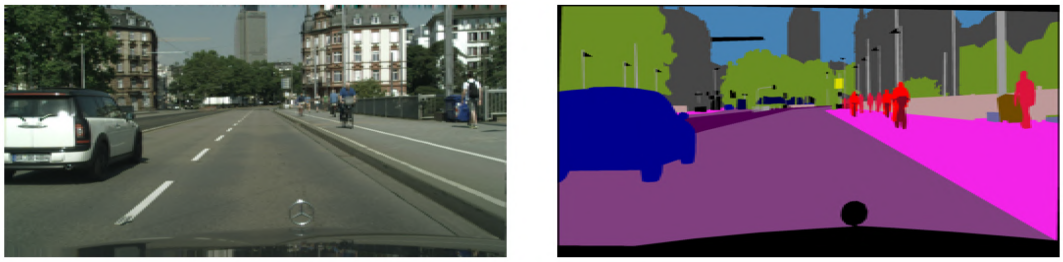


Figure 4.5: Example image and corresponding segmentation map from the Cityscapes dataset.

GTA5

The Grand Theft Auto 5 (GTA5) dataset [Richter *et al.* 2016a] is a massive synthetically generated dataset used to mimic outdoor scenes, similar to Cityscapes. The dataset is designed to be close to the other datasets used in urban scene understanding like CamVid, KITTI, and Cityscapes. The GTA5 dataset consists of 19 classes which are a

subset of the Cityscapes classes system, making them compatible with domain adaptation experimentation.

All images and labels were generated using the Grand Theft Auto 5 game engine, creating an opportunity to generate a large number of labeled images while also having control over factors such as classes, lighting, and weather conditions. In total, the dataset contains 24966 pixel-level annotations, with each frame having a resolution of 1914x1052. The classes in the GTA5 dataset are road, building, sky, sidewalk, vegetation, car, terrain, wall, truck, pole, fence, bus, person, traffic light, traffic sign, train, motorcycle, rider, and bicycle. An example of an image and segmentation map found in the GTA5 dataset can be seen in figure 4.6.

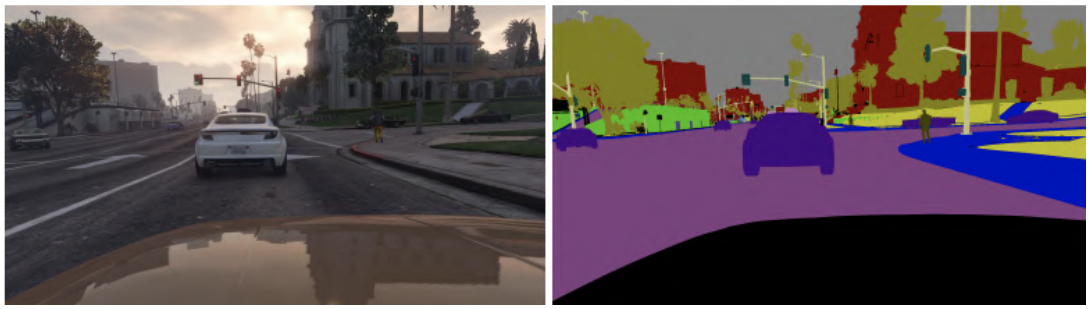


Figure 4.6: Example image and corresponding segmentation map from the GTA5 dataset.

4.4.3 Experiments

This section presents our experimental design. This covers an overview of the different experimental setups for the small digits experiments and the semantic segmentation. Each experiment's exact setup is covered in its respective chapters.

While there are many different avenues in the domain adaptation space, we only focus on the homogeneous unsupervised domain adaptation case. For a problem to be considered homogeneous, both the source and target domains should share the same feature and label space, but the difference is in how the data is represented [Wang and Deng 2018]. Furthermore, because it is an unsupervised domain adaptation problem, we enforce the assumption that there are no target domain labels to train on. In our experiments, we only use the target labels in the testing phase and as an oracle baseline to benchmark our model against to consider the case where we have a full target dataset. For brevity we will use the notation of Source→Target to represent which dataset is the source and target domains respectively. Below in table 4.2 we list all the experimental architecture combinations as well as the table which contains their respective results.

Table 4.2: Architecture setup for each experiment.

Experiment	Classifier/Task Network	Generator	Discriminator	Results Table
MNIST $\Leftrightarrow$ USPS	LeNet	ResNet	PatchGAN	Table 7.1
MNIST $\Leftrightarrow$ USPS	DTNClassifier	ResNet	PatchGAN	Table 7.2
MNIST $\Leftrightarrow$ USPS	SEClassifier	ResNet	PatchGAN	Table 7.3
SVHN $\Leftrightarrow$ MNIST	DTNClassifier	ResNet	PatchGAN	Table 7.4
GTA5 $\rightarrow$ Cityscapes	VGG	ResNet	PatchGAN	Table 8.1
GTA5 $\rightarrow$ Cityscapes	UNet	ResNet	PatchGAN	Table 8.1
GTA5 $\rightarrow$ Cityscapes	DeeplabV3	ResNet	PatchGAN	Table 8.1

Small Digits Experiments

We first test the method's effectiveness for the small-digit experiments with a baseline set of architectures and hyperparameters. The task network for these experiments will be a variation of the LeNet model as described in Hoffman *et al.* [2018]. For the rest of the dissertation, all mentions of the LeNet model refer to this variation and not the one proposed in LeCun *et al.* [1998]. The CycleGAN architecture uses the same models described in Zhu *et al.* [2017]. This includes a ResNet generator with six residual blocks and a PatchGAN discriminator. This experimental setup will be used for testing four different experiments: MNIST→USPS, USPS→MNIST, SVHN→MNIST, and MNIST→SVHN.

We then analyze how the performance changes as we increase the model capacity of the task network to account for differences in how well each model naturally deals with the domain shift. This also accounts for the fact that specific classification tasks are more difficult and require more powerful network architectures to achieve relevant results.

We then test the robustness and generalizability of our method by showing how well it performs under different hyperparameter selections. Furthermore, we test how much each constituent part contributes to the performance by an ablation study with the removal of certain aspects of the model. This set of experiments provides an understanding of how well the proposed method works and what the limitations are.

Semantic Segmentation

To show that our method works on more than just domain adaptation in standard classification, we will extend the method to the dense prediction problem domain. We look into bridging the GTA5→Cityscapes gap for training an urban scenery semantic segmentation model. This specific domain adaptation challenge is especially beneficial as it addresses the problem of being able to use synthetically generated data (GTA5) to

train a model to use in the real world (Cityscapes). In a similar fashion to the small digits experiments, we start by testing the performance of the task-net with a variation of the VGG model as outlined in Hoffman *et al.* [2018] and use a CycleGAN with a ResNet generator with nine residual blocks and the same PatchGAN discriminator. We then check how well this method works for other models by testing it on a U-Net [Ronneberger *et al.* 2015] architecture and a DeepLabv3 architecture [Chen *et al.* 2017b].

4.4.4 Summary

This chapter was used to highlight our research hypothesis, research questions and the methodology we will use to verify our hypothesis. In our methodology, we discussed the model architectures, datasets and how we plan to use them as a downstream task to evaluate our model's ability to perform domain adaptation. In the next chapter, we go over the core aspects of the CycleGAN and experiment how well it can perform visual domain adaptation.

Chapter 5

Cycle Consistency Generative Adversarial Network

In this section, we look deeper at one of the backbone algorithms of our methodology, the Cycle-Consistent Generative Adversarial Network or CycleGAN. As discussed in Section 2, CycleGAN was developed to perform image-to-image translation without requiring paired examples during training. This makes CycleGAN a prime candidate for unsupervised domain adaptation as we can learn a mapping between source and target domain with little supervision.

5.1 Cycle-consistency Loss

Here we dive deeper into the theoretical component of the cycle-consistency loss used in CycleGAN. We define a set of generators G and F that takes in an image as an input and outputs another image. Along with these generators, we also have a pair of discriminators D_x and D_y, which take in an image and predict if they are real or fake. The D_x discriminator is used to predict if an image is a real image in the domain X, and conversely, D_y is used to predict if an image is real in the Y domain. In the standard GAN formulation, each generator would be fed some input (either random noise or an image) and generate the output image. This output image is then given to the discriminator to determine if it is real or fake. However, when we feed an input image as the conditioning vector for the generator. There is no guarantee that the output image must maintain the semantic information contained in the input image, as the generator's only goal is to fool the discriminator into thinking the generated images are real. We introduce the cycle-consistency loss propsoed by Zhu *et al.* [2017] to encourage the model to keep the semantic information. We train G to map an image x from domain X to appear like it was sampled from Y by training D_y to discriminate between real images sampled from domain Y and fake images generated by G. We

similarly train F to map an image y from Y to domain X. Training these separate parts uses the traditional GAN paradigm. To help keep semantic meaning in the images, we use the fact that $F(G(x)) \approx x$ and $G(F(y)) \approx y$ Zhu *et al.* [2017]. This encourages the networks to create a bijective mapping between the domains and not lose any crucial semantic information needed in reconstructing the images. So in the case of the summer to winter example, if x is an example of an image taken during summer time. Then $G(x)$ would be the scene mapped in the winter theme. If we pass $G(x)$ into F, we can enforce the network to output an image similar to the original x we started with, as this should be the summer scene we would expect.

So if we consider the first GAN G's loss function, which maps an image from domain X to domain Y, we have:

$$L_G(G, D_y, y, x) = \mathbb{E}_{y \sim Y}[log(D_y(y))] + \mathbb{E}_{x \sim X}[log(1 - D_y(G(x)))] \tag{5.1}$$

The second GAN F's loss function, which maps images from domain Y to domain X, can similarly be given by:

$$L_F(F, D_x, x, y) = \mathbb{E}_{x \sim X}[log(D_x(x))] + \mathbb{E}_{y \sim Y}[log(1 - D_x(G(y)))] \tag{5.2}$$

Now we can represent the new cycle consistency loss Zhu *et al.* [2017] with:

$$L_{cyc}(G, F, x, y) = \mathbb{E}_{x \sim X}\left[\, \|F(G(x)) - x\|_1 \,\right] + \mathbb{E}_{y \sim Y}\left[\, \|G(F(y)) - y\|_1 \,\right] \tag{5.3}$$

To get our final objective function for the entire CycleGAN Zhu *et al.* [2017], we combine the three separate loss functions and obtain the following:

$$L_{full}(G, F, D_x, D_y) = L_G(G, D_x) + L_F(F, D_y) + \lambda L_{cyc}(G, F) \tag{5.4}$$

where:

- x is a sample from domain X,
- y is a sample from domain Y,
- $\mathbb{E}_{x \sim X}$ is expected value across all samples from domain X,
- $\mathbb{E}_{y \sim Y}$ is expected value across all samples from domain Y,
- $\|\cdot\|_1$ is the L1 norm of a vector,
- λ is a weighting factor used to control how much the cycle consistency loss should impact the final loss.

By including the reconstruction loss in both directions, we can enforce that the cycle consistency is preserved in both directions and that $F(G(x)) \approx x \iff G(F(y)) \approx y$. In figure 5.1, a visual representation of the CycleGAN pipeline is illustrated. The second column shows how the cycle-consistency constraint enforces the distance between the original image x and the final image $G(F(x))$ to remain as small as possible, helping maintain the bijective mapping.

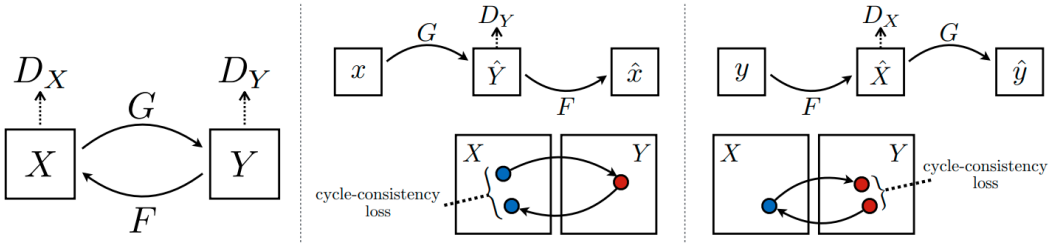


Figure 5.1: CycleGAN's bijective mapping as given by [Zhu *et al.* 2017]

5.2 Image-to-Image translation

As mentioned before, the main appeal of using the CycleGAN over other methods is that it can change the appearance of images to suit other domains with very little supervision. Other methods often require the same image in both domains, called image pairs, during training to learn meaningful mappings between the domains. However, the only supervision that CycleGAN needs is the label for which domain each image belongs, which we often know in any case. CycleGAN has been demonstrated to have a wide range of applicability in the field of image-to-image translation.

5.2.1 Style Transfer

In Figure 5.3, CycleGAN was used to learn a mapping between images painted by the famous painter Claude Monet to everyday photographs. Not only is it possible to recreate the actual scenes that Monet would have seen with photo realism, it also lets us turn modern-day scenes that Monet could never have dreamed of into paintings of his style. This is known as style transfer, and CycleGAN was able to transform these images into many different styles, as seen in Figure 5.2.

Figure 5.2: Mapping between different art styles as shown in [Zhu *et al.* 2017]

In figure 5.3 we can see an example of how Zhu *et al.* [2017] used CycleGAN to map from Monet paintings to photographs. In this case, we could treat the Monet pictures as domain X and the photographs as domain Y. The model then learns to map the Monet picture x to a fake photograph of $G(x)$, and the inverse case of mapping a photograph y to a fake Monet picture $F(y)$.

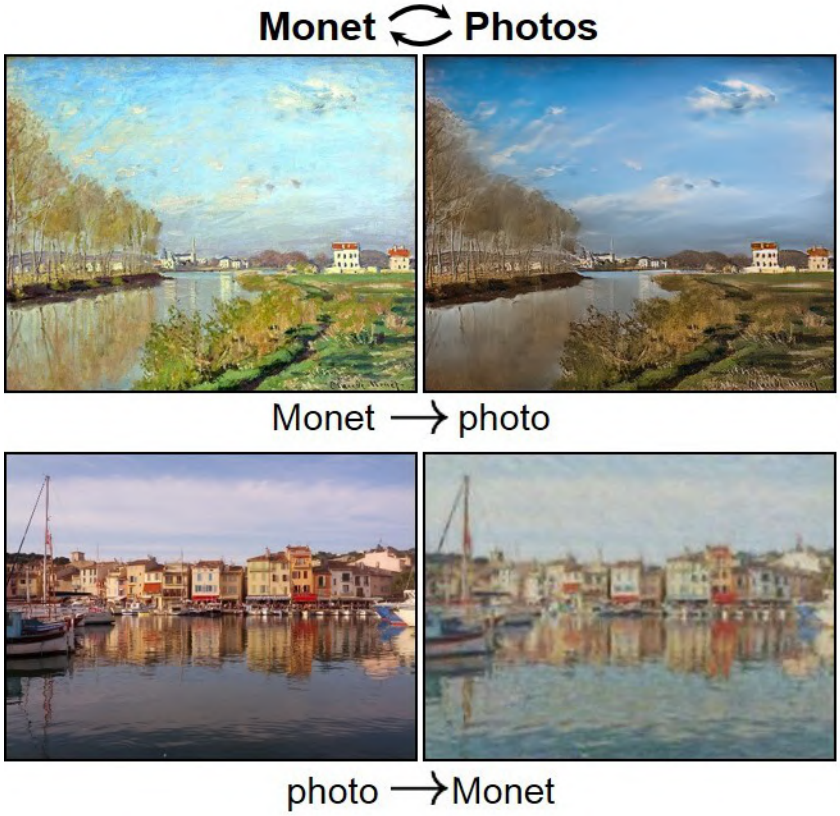


Figure 5.3: Mapping between photographs and Monet paintings using CycleGAN [Zhu *et al.* 2017]

5.2.2 Seasonal Change

In figure 5.4, CycleGAN was used to change the appearance of the warm summers at Yosemite park into the snowy winters. The summer images can be treated as the domain X and the winter images as the domain Y. This allows us to train a model which can map real summer images X to fake winter images $G(X)$. In these examples, we can see how it learns to transfer critical contextual information associated with each of the seasons, like the abundance of snow and ice. This makes it possible to make winter wonderland scenes from areas that do not usually experience such weather.

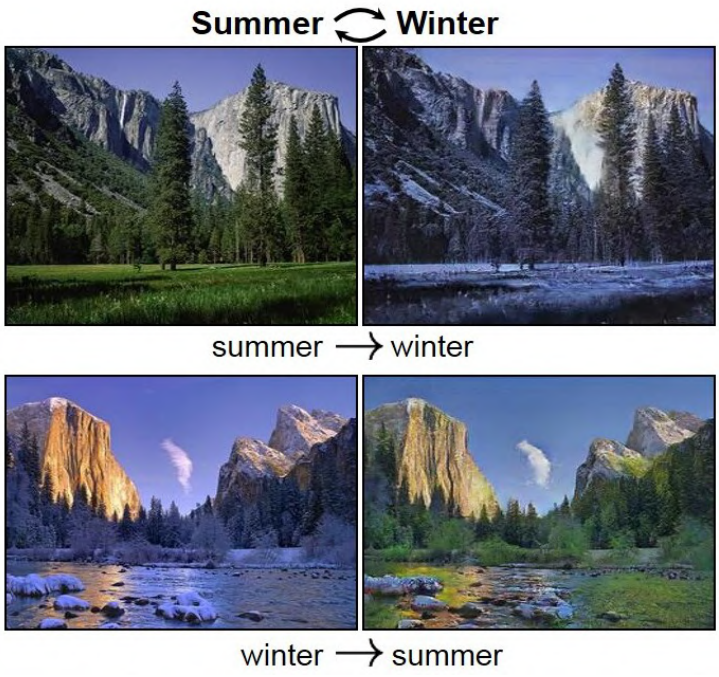
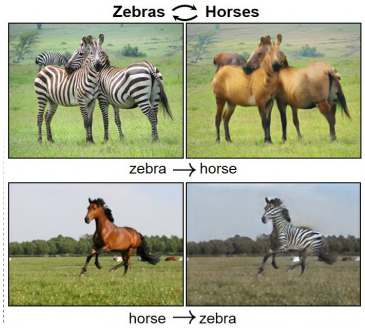


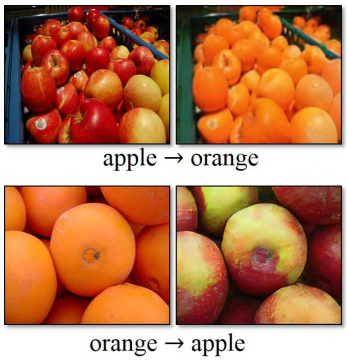
Figure 5.4: CycleGAN changing between seasons [Zhu *et al.* 2017].

5.2.3 Object Transfiguration

The next application we will look at will be object transfiguration. This involves altering the appearance of one object into another while keeping some of the original characteristics. In figure 5.5a, we can see how they learned to convert images of zebras into horses and horses back into zebras. This shows how CycleGAN can make more significant changes to the images than just lighting or color shifts. However, CycleGAN requires the two images to be in somewhat relatable domains to make a viable mapping possible.



(a) Transforming zebras into horses.



(b) Transforming apples into oranges.

Figure 5.5: Examples of CycleGAN performing object transfiguration taken from Zhu *et al.* [2017].

5.3 Domain adaptation

From the examples listed above, we can see that CycleGAN holds a great deal of potential for learning mappings between domains that will reduce the performance gap caused by the domain shift without the need for much supervision. To explore this idea further, we test CycleGAN on a basic domain adaptation problem, MNIST $\Leftrightarrow$ USPS. While these two datasets share many similarities, the difference in how the images were captured causes appearance differences that lead to performance degradation when using the same classifier on both. Below in table 5.1, we test to see if we get any performance improvement by training the classifier on the images altered by CycleGAN.

In the following examples, we test out the standard CycleGAN architecture on a sample unsupervised domain adaptation problem. The source domain with labels will be denoted as X and the target domain without labels will be denoted as Y. The CycleGAN model is trained to generate fake $G(X)$ images to appear as similar to Y as possible. This is done with the cycle-consistency term we outlined above. Once the CycleGAN has finished training we use the mapped $G(X)$ images with their labels to train a classifier. In these experiments, we use the same architectures as outlined in Zhu *et al.* [2017] for the generators and discriminators and the LeNet model [Hoffman *et al.* 2018] as the classifier.

Table 5.1: CycleGAN Domain Adaptation MNIST$\Leftrightarrow$USPS Training and Testing data Accuracy (%)

Name	Training Source	Training Target	Testing Source	Testing Target
MNIST$\rightarrow$USPS	92.22 ± 1.79	$\mathbf{94.09 \pm 0.79}$	92.48 ± 1.56	$\mathbf{94.45 \pm 0.93}$
USPS$\rightarrow$MNIST	91.37 ± 2.25	$\mathbf{75.01 \pm 7.56}$	91.01 ± 2.511	$\mathbf{75.96 \pm 7.44}$
Train on MNIST	99.11 ± 0.06	$\mathbf{86.68 \pm 1.04}$	98.64 ± 0.12	$\mathbf{86 \pm 1.03}$
Train on USPS	96.96 ± 0.21	$\mathbf{64.66 \pm 2.77}$	96.42 ± 0.18	$\mathbf{66.22 \pm 2.64}$

As we can see in figure 5.6 and table 5.1, by first transforming the images to look more like the target domain and then training the classifier, we can gain a substantial increase in performance, with MNIST$\rightarrow$USPS improving the testing accuracy from 86% to 94.45% which brings it just a few percent below the model that was trained directly on the target data set. In the reverse direction of USPS$\rightarrow$MNIST, we see an improvement of 66.22% to 75.96%, while being a substantial improvement over performing no adaptation, still falls short of training on the target sets accuracy.

A major issue that can occur, which is not always apparent by just looking at the testing accuracies, is the problem of label flipping. Label flipping occurs when the source image's and target image's semantic meaning do not align after the CycleGAN maps it between domains. An example of this can be that a "1" in the MNIST domain may be generated to appear like a "2" in the USPS domain and then maps it back to a "1" in the MNIST domain in the reverse direction. Label flipping occurs when the network learns to map between semantically different images in each domain, but the

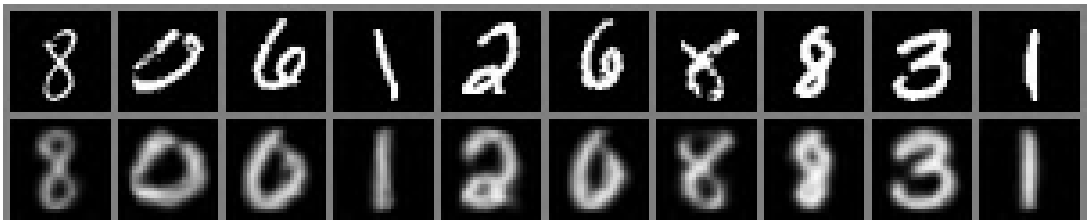


Figure 5.6: Example outputs from MNIST to USPS mapping. The top row are the MNIST images and the bottom row are the corresponding generated USPS images

reconstruction will still be similar to the original image. This happens because the cycle-consistency loss does nothing to ensure that the intermediate domain maintains the same semantic meaning as the starting domain, and the GAN loss is only concerned with whether the image looks realistic. In figure 5.7, we can see a few examples of label flipping occurring in the MNIST⇔USPS problem.

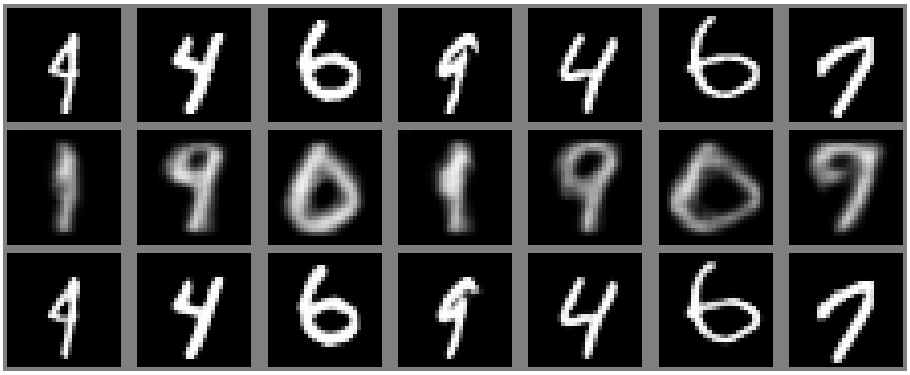


Figure 5.7: Examples of label flipping. Row 1 is the original image, row 2 is the mapped image and row 3 is the reconstructed image.

In the MNIST⇔USPS experiments, the domains are aligned enough such that the CycleGAN can learn well enough without any further guidance. However, in the case of the more significant domain shift of SVHN→MNIST, the network struggles to learn anything that is semantically meaningful and instead degenerates into mode collapse, where the final classifier becomes useless, with a testing accuracy of 48.89% as seen in table 5.2. An example this can be seen in figure 5.8, where the network has learned a degenerate solution from mapping the source domain to the target domain. However, it has learned to reconstruct the source image almost identically to avoid penalization from the cycle-consistency loss.

5.3.1 Summary

In this chapter, we highlighted some of the key aspects of the CycleGAN model and showed some of its potential use cases. We further explored some preliminary experiments for using the CycleGAN model as a method for performing unsupervised domain

Figure 5.8: Example of mode collapse from the SVHN→MNIST experiment. Row 1 is the original image, row 2 is the mapped image and row 3 is the reconstructed image.

Table 5.2: CycleGAN Domain Adaptation SVHN→MNIST Training and Testing data Accuracy (%)

Name	Training Source	Training Target	Testing Source	Testing Target
SVHN→MNIST	35.29 ± 3.88	$\mathbf{47.68 \pm 6.06}$	28.78 ± 4.13	$\mathbf{48.89 \pm 6.28}$
Trained on SVHN	96.98 ± 0.07	$\mathbf{65.67 \pm 1.96}$	91.72 ± 0.12	$\mathbf{67.41 \pm 2.36}$
Trained on MNIST	99.99 ± 0.003	$\mathbf{19.23 \pm 1.52}$	99.28 ± 0.02	$\mathbf{26.23 \pm 2.39}$

adaptation for images. In our experiments, we found that the CycleGAN can be used to improve the performance of a classifier model trained only using the source domain labels by first mapping the source images to look more similar to the target domain images. However, issues such as label flipping and mode collapse occurred, especially on the harder domain adaptation tasks. We hypothesize that the issues above can be addressed by introducing further supervision into the training process. By introducing a task network (classification/semantic segmentation), we can enforce semantic meaning through additions to the loss function. This could be easily done by introducing labels from the target domain, but in Chapter 6, we further improve on this idea by using only the labels from the source domain.

Chapter 6

Improving learning with task network

In the previous chapter, we found that training with Cycle Consistency alone does not fully solve the problem. While the model can learn a good mapping between the different domains to make realistic-looking images, there is nothing in the training procedure that explicitly enforces that the semantic meaning is the same before and after the mapping. This leads to the case of potential label flipping and mode collapse. This chapter considers the addition of a task network to maintain the semantic meaning of images on either side of the domain mapping process.

6.1 Task Network

A task network is an auxiliary network in the model that performs the same action as the downstream task; this would be classification in our case. The goal is to use the outputs of the task network to improve the model's overall performance in some way. Below we will look at how Hoffman *et al.* [2018] utilize their task network, and from there, we cover the extensions implemented in this work.

6.1.1 Cycada Task Network

The problem of label flipping was also observed by Hoffman *et al.* [2018], where they were also using CycleGANs as part of their UDA pipeline. To tackle this problem, they introduce a task network, which is essentially a noisy classifier that is pretrained on the source domain before the GAN training to guide the training procedure. Since there is no access to the target labels in the UDA problem, the classifier was only trained on the source domain and was used to classify the before and after images. A task loss was then introduced, which calculated a loss based on the difference between the before

and after image labels and penalized a difference in classification. While this did add additional guidance and was shown to help with label flipping, the task network was only trained using the source domain, making it a potentially weak classifier for the target domain.

6.1.2 Our Task Network

Following up on the work proposed by Hoffman *et al.* [2018], we also utilized a task network in guiding our CycleGAN's training. However, instead of just pre-training the task network on the source domain and using it throughout training, we continuously train the network during the GAN training procedure on both the source images and the fake target images generated from the source images. This allows for the classifier to both improve as the GANs improve and allow for better guidance through the training procedure.

With this technique, we aim to alleviate the issues that surround using CycleGANs for domain adaptation, like label flipping and mode collapse, while also adding more information for the generators to learn from in the form of the task network. While the generator's performance improves, the fake-generated images should also improve, thus reducing the domain shift between the generated images and target images, which should improve the task network's performance on both domains.

6.1.3 Task Network loss function

Following on from the objective function given in (5.4), we shall now extend it to include the semantic loss from the task network to guide the GAN training and the self-supervised loss to improve the task network as the GAN trains.

Let $P(f, x)$ be the set of class probabilities outputted by the network f for the input image x. For the task network, we use Cross-Entropy loss which, for a typical classification problem, can be given by:

$$L_T(f, D, L) = -\mathbb{E}_{(x,y)\sim(D,L)}\big[\sum_{k=1}^{K} \mathbb{1}_{k=y} \cdot log(\sigma(f(x)))\big] \tag{6.1}$$

where:

- f is some network that is performing the classification,
- D is the set of all inputs,
- L is the set of all labels for D,

- $\mathbb{E}_{(x,y)\sim(D,L)}$ is the expected value over all input/label pairs from D and L,
- x is a single sample from set D,
- y is the corresponding label for x sampled from L,
- K is the number of classes for the classification problem,
- $\mathbb{1}_{k=y}$ is the indicator function that is 1 when $k = y$ and 0 otherwise,
- σ is the softmax function,
- $f(x)$ is the outputs of the network f given an input x.

6.1.4 Semantic loss

The first loss the task network provides us is the semantic loss, which is used to help stabilize the CycleGAN training to prevent mode collapse and label flipping. This loss utilizes the outputs of the task network to ensure that the semantic meaning of the images is similar before and after they have been transformed into different domains. In Hoffman *et al.* [2018], they propose using a "noisy" classifier which is pre-trained on the source network, and the network is encouraged to keep the same labels between the different domains.

This loss can be given as:

$$L_{sem_0}(f_t, G, F, X, Y) = L_T(f_t, G(X), P(f_t, X)) + L_T(f_t, F(Y), P(f_t, Y)) \tag{6.2}$$

where:

- f_t is the task network.
- G and F are the generators used to map from domain $X \rightarrow Y$ and $Y \rightarrow X$ respectively.
- X is the source domain and Y is the target domain.
- $P(f_t, X)$ is the set of class probabilities outputted by the network f_t for domain X.
- $P(f_t, Y)$ is the set of class probabilities outputted by the network f_t for domain Y.

While the loss (6.2) helps alleviate the issues of label flipping, it is still only comparing the outputs of the task network and not the actual label. Because our method has the task network learning alongside the GAN to classify both the source and target domain, after sufficient training, we switch the semantic loss in (6.2) to use the source

domain labels for the generated image as we believe at this point the generator should be sufficiently good at generating realistic target images with the same semantic meaning and we eliminate the possibility of the task network learning bad classifications from using its outputs as a label.

The new loss function becomes:

$$L_{sem_1}(f_t, G, F, X, Y, L) = L_T(f_t, G(X), L) + L_T(f_t, F(Y), P(f_t, Y)) \tag{6.3}$$

where:

- f_t is the task network.
- G and F are the generators used to map from domain $X \rightarrow Y$ and $Y \rightarrow X$ respectively.
- X is the source domain and Y is the target domain.
- L is the set of labels for the source domain.

By changing the semantic loss to not only include the outputs of the task network but also the labels in the source domain. This alteration is intended to change the task network from a "noisy" classifier that slightly guides the training to a continuously improving self-supervised classifier that utilizes the generated images to better improve the training process. We change from L_{sem_0} to L_{sem_1} at 50 epochs, which we found experimentally to yield good results.

6.1.5 Task-network Loss

The next major addition to the training regime is the continual training of the task network during the GAN training. The task network will have two main training phases: the source-only phase and the self-supervised phase.

The source-only phase trains the network with the unaltered source images and their labels. This usually consists of a pre-training phase to train the classifier to a sufficiently accurate level to give meaningful results to the generators through the semantic loss and then a continual learning phase where it learns alongside the GAN training phase.

After a set amount of epochs, the model will start training the task network in a self-supervised manner. This is done by using the GAN-generated images as target images to expose the generators to a more similar target "style". Then as the GANs improve the quality of the generated target images, the task network has better quality data to train on. In turn, this should give the generators better information to learn.

The combined task loss can be given by:

$$L_{task}(f_t, G, X, L) = L_T(f_t, X, L) + \alpha L_T(f_t, G(X), L) \tag{6.4}$$

While in the source-only phase, the weighting parameter α is set to 0, once the self-supervised phase begins, we set $\alpha > 0$ where α becomes a weighting factor between how important the new target images are compared to the source images. In the self-supervised loss phase, we maintain the original $L_T(f_t, X, L)$ term to maintain optimal performance as a semantic labeler for the generators, as the network needs to be able to classify both the source and target images accurately. This has the added benefit of the network maintaining its performance in the original domain instead of trading source domain accuracy for target domain accuracy.

6.1.6 Total Loss

By adding the Cycle-loss from (5.4), the semantic loss from (6.2) and (6.3) and the task loss from (6.4) we get our final loss function which can be given by:

$$\begin{aligned} L_{full}(G, F, D_x, D_y, f_t, X, Y, L) = L_G(G, D_x) + L_F(F, D_y) + \lambda L_{cyc}(G, F) + \\ L_{sem}(f_t, G, F, X, Y, L) + L_{task}(f_t, G, X, L) \end{aligned} \tag{6.5}$$

Where:

- L_G and L_F are the losses for Gan's G and F respectively,
- L_{cyc} is the Cycle-Consistency loss from (5.4),
- L_{sem} is the semantic loss from (6.3),
- L_{task} is the task networks loss from (6.4).

Equation (6.5) constitutes the complete loss for our model. However, we can break the training up into three separate phases. During phase 1, we pretrain the classifier with the loss function outlined in (6.4) where $\alpha = 0$. This pre-training step ensures the classifier has a reasonable baseline accuracy to give meaningful gradients during the rest of training. Phase 2 is the training of the CycleGAN with the full loss outlined in (6.5) with $\alpha = 0$ in the L_{task} term, and phase 3 constitutes the full CycleGAN loss where we set $\alpha > 0$ to initiate the self-supervised training of the task network. With this loss, we can learn a network that can perform well on both the supervised source domain and the unsupervised target domain, even without the target labels present. The next chapter demonstrates how the methods outlined here can be used in practice on the Small Digits domain.

6.1.7 Summary

This chapter has laid down the theoretical basis of how we will modify the traditional CycleGAN loss to be more adept at performing unsupervised visual domain adaptation. The main aspects are to add a task network that can help guide the GAN components during training, and in addition to continuously train the task network with the latest outputs from the GANs. This aims to give the task network samples with high likeliness to the target domain. In the next chapter, we will use the formulated model above and explore how well it performs on the Small Digits Experiments.

Chapter 7

Small Digits Experiments

We used the methods proposed in Chapter 6 to answer the questions outlined in Chapter 4. The first set of datasets used to test the practicality of our proposed model will be the small digit datasets. These consist of the hand-written character dataset MNIST, the hand-written character dataset USPS and the natural world images SVHN dataset.

7.1 MNIST $\Leftrightarrow$ USPS

Each of these datasets fundamentally represents the same thing. However, the generation and capturing processes for each of them are different. The baseline domain shift experiment we will look at uses MNIST as the source domain and USPS as the target domain. While these datasets are extremely similar in character orientation, color, and generating process, slight differences have occurred due to the nature of how the data is captured. Due to the USPS images initially having a resolution of 16×16, when they get up-sampled, they have a slightly blurrier appearance than the MNIST images. Another significant difference between these datasets is the number of training samples, with MNIST having 60000 samples and USPS having only 7438 training images.

7.1.1 Preliminary Experiments

For our first experiments, we used the same methodology and architectures outlined in section 4.4. The USPS images were up-sampled to be 28×28, as outlined in Liu and Tuzel [2016], to match the resolution of the MNIST images. For each training epoch of the CycleGAN, we took a random subsample of the MNIST dataset to match the number of samples in the USPS dataset. For the task network, we used the variation of LeNet used in Hoffman *et al.* [2018]; for the generators, we used a Resnet which has

six residual blocks. Furthermore, we used the PatchGAN architecture described by Zhu *et al.* [2017] for the discriminator. The task network was pretrained for 50 epochs on the source domain data (phase 1), the CycleGAN section is trained for 30 epochs with $\alpha = 0$ (phase 2), and then CycleGAN is then trained for a further 70 epochs with $\alpha = 1$ (phase 3).

These slight differences between the datasets give us a good starting place to test our method. The tasks are similar enough to share important characteristics but different enough that training a naive classifier on only the source domain would cause performance degradation issues. For these reasons, we used this as our baseline test for our new method.

MNIST $\rightarrow$ USPS

The first experiment treated MNIST as the source domain with labels and USPS as the target domain without labels. Due to the large difference in the number of samples in the training set, we subsample a new subset of the MNIST dataset during each epoch, such that the two datasets are the same size, when phase 2 and phase 3 training takes place. The entire dataset gets used in phase 1 for pretraining the classifier.

USPS $\rightarrow$ MNIST

The second experiment involved the reverse of the one above. While the base CycleGAN is completely uni-directional, making it irrelevant to which domain is set as domain A, the alterations we added to the model's loss can create stark differences in which direction the model goes. The first significant difference arises from the fact that we only have labels for the source domain, which can mean specific domain adaptation problems change depending on which domain is the source domain. In this case, the massive size imbalance between the two datasets can cause issues as we have a lot more unlabelled data than we have labeled data, which is contrary to the reverse problem mentioned above.

Results

In table 7.1, we present the results from the experiments we listed above. The numbers highlighted in bold represent the target domain. The first observation we can make is that the introduction of the semantic loss and the self-supervised learning from the task network has improved the performance of both target domains for MNIST$\rightarrow$USPS and the USPS$\rightarrow$MNIST cases.

For the case of MNIST→USPS, the performance increases to 95.41%; despite this being a large increase, it is only around a $\sim 1\%$ increase over the traditional CycleGAN as the cycle-consistency loss alone manages to get the classifier close to target trained accuracy. However, this is still a net increase of $\sim 10\%$ compared to performing no adaptation at all, which has a testing accuracy of $\sim 86\%$. The additional benefit of the semantic loss from the classifier is that the issue of label flipping has been alleviated, and the variance between experiment runs has been lowered from the classifier becoming more consistent. Figure 7.1 shows some qualitative results after performing the mapping. In the first row we can see the original MNIST images X, in the second row we can see the generated USPS images which we can denote as $G(X)$, and finally, the last row is the reconstructed images MNIST images $F(G(X))$. Another observation that should be noted from the experiments is how the task network maintains its ability to classify the source domain while learning how to classify the target domain. This allows the original classifier to perform its original goal while learning to classify a new domain, essentially making it a cross-domain classifier.

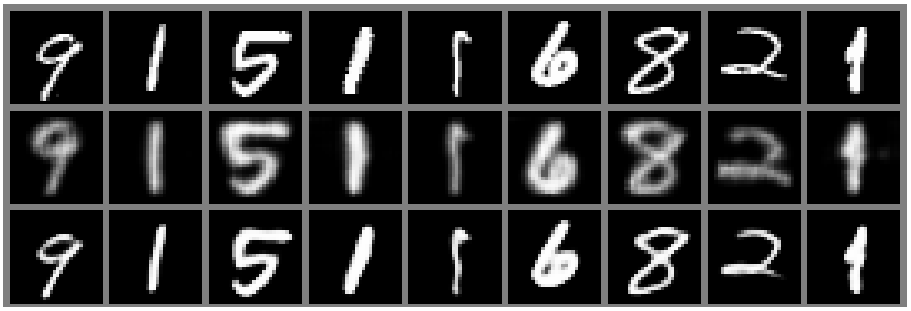


Figure 7.1: Example image output for our model on MNIST→USPS. Row 1 is the original image from MNIST, row 2 is the fake USPS image generated from the GAN and the final row is the reconstructed MNIST image.

The next result we will look at is the case of USPS→MNIST unsupervised domain adaptation. From table 7.1, it can be noted that the classifier trained on MNIST data performed better on USPS data than the classifier trained on USPS classifying MNIST data. A few potential causes of this can be that the MNIST dataset has a larger training set allowing the model to learn a more generalized classifier. The other potential case is that the features learned while training on MNIST provide a better-generalized representation for USPS than in the reverse case. The gap created by this discrepancy, regardless of its cause, leads to a more significant domain shift than in the case of USPS→MNIST.

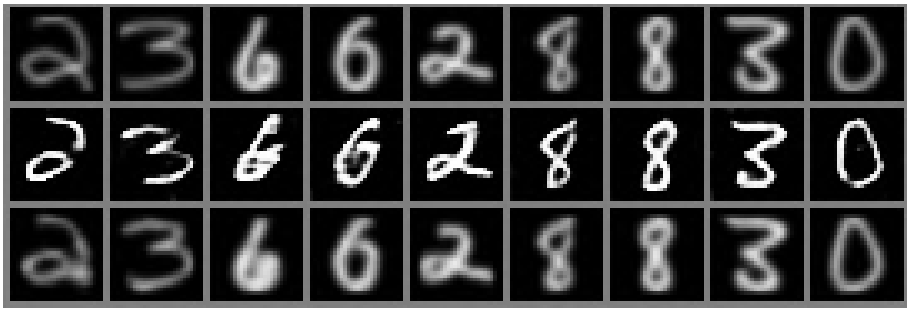


Figure 7.2: Example image output for our model on USPS→MNIST. Row 1 is the original image from USPS, row 2 is the fake MNIST image generated from the GAN and the final row is the reconstructed USPS image.

From table 5.1, it can be noted that the cycle-consistency loss significantly improved the testing accuracy on the MNIST dataset from 66.22% to 75.96%, which represents a notable enhancement. However, a significant discrepancy remains compared to models trained on target data. This is where our proposed model has shown to yield the most significant performance increase as it takes the 75.96% and improves it further to 92.02%. This brings it within just a few percent of a model trained on the MNIST dataset. In figure 7.2 we have some results for the USPS→MNIST, as we can see the model learns to take the sightlier blurrier USPS images and create clearer detail to match the style of the MNIST images.

Table 7.1: SSGAN Domain Adaptation MNIST⇔USPS Training and Testing data Accuracy ($\%$).

Name	Training Source	Training Target	Testing Source	Testing Target
MNIST→USPS	99.08 ± 0.07	$\mathbf{94.89 \pm 0.49}$	98.54 ± 0.13	$\mathbf{95.41 \pm 0.47}$
USPS→MNIST	98.76 ± 0.03	$\mathbf{90.83 \pm 1.21}$	97.78 ± 0.16	$\mathbf{92.02 \pm 1.18}$
Trained on MNIST	99.11 ± 0.06	$\mathbf{86.68 \pm 1.04}$	98.64 ± 1.18	$\mathbf{86 \pm 1.03}$
Trained on USPS	96.96 ± 0.21	$\mathbf{64.66 \pm 2.77}$	96.42 ± 0.18	$\mathbf{66.22 \pm 2.64}$

7.1.2 Task-network Architectures

The above results show remarkable improvement over the non-adapted models, with the post-adaptation models nearly reaching the same accuracy as models trained directly on the target labels. However, these results were obtained with a simple classifier model for the task network. The next set of experiments involves various classifiers with higher capacity for the task network. This helps illustrate whether our method can scale to not just multiple problems but multiple architectures as well. Secondly, it helps show that our model will not only improve the performance of low-performing models but can also add benefit to more powerful models.

The first model we look at is a variation of the DTNClassifier as used in Hoffman *et al.* [2018]. The DTNClassifier has more capacity than the version of LeNet we used earlier. It contains more convolutional layers and filters, a larger fully connected layer with both dropout and batch normalization layers. The first thing to note from table 7.2 is how the models trained without adaptation already perform better on the target domains from the increased capacity of the networks.

A DTNClassifier trained on the MNIST training set now achieves a testing accuracy of 90.39% when tested directly on USPS, compared to the 86% achieved by the LeNet variant. The MNIST→USPS adaptation model achieves a target performance of 97.04%, which is an increase of $\sim 7\%$ over training on MNIST data alone and is only $\sim 1\%$ below the performance of a DTNClassifier trained on the target labels directly. It outperforms the LeNet model trained on the target set. Similarly to the results in table 7.1, the USPS→MNIST shows a more significant net improvement than the MNIST→USPS experiment as the classifier trained on the USPS data alone seems to generalize poorly with a target MNIST accuracy of 74.2%. Our model can improve the MNIST target accuracy by $\sim 23\%$, bringing it up to 95.92%. This brings it closer to the USPS-trained DTNClassifier and outperforms the LeNet model trained on USPS.

Table 7.2: SSGAN Domain Adaptation MNIST⇔USPS Training and Testing data Accuracy (%) with the DTNClassifier.

Name	Training Source	Training Target	Testing Source	Testing Target
MNIST → USPS	99.98 ± 0.003	$\mathbf{97.15 \pm 0.02}$	99.26 ± 0.07	$\mathbf{97.043 \pm 0.14}$
USPS → MNIST	99.96 ± 0.001	$\mathbf{94.12 \pm 0.57}$	98.64 ± 0.11	$\mathbf{95.5 \pm 0.55}$
Trained on MNIST	99.99 ± 0.003	$\mathbf{91.03 \pm 0.49}$	99.28 ± 0.02	$\mathbf{90.39 \pm 0.68}$
Trained on USPS	99.94 ± 0.008	$\mathbf{72.53 \pm 0.65}$	98.78 ± 0.08	$\mathbf{74.2 \pm 0.29}$

The second additional model we will look at is the classification network used in French *et al.* [2017] which we will denote as SE classifier. The SE classifier is larger than the LeNet classifier but smaller than the DTN classifier, letting us test an intermediate-level classifier. In table 7.3, we can see how the results follow the same trend as the previous two experiments. The MNIST→USPS experiment increases the performance of the non-adapted classifier from 88.66% to 96.95%, and for the USPS→MNIST we get an increase from 72.03% to 95.92%. Overall it is clear that the model increases results across the board regardless of the baseline architecture used for the task network.

Table 7.3: SSGAN Domain Adaptation MNIST⇔USPS Training and Testing data Accuracy (%) with the SE Classifier.

Name	Training Source	Training Target	Testing Source	Testing Target
MNIST → USPS	99.36 ± 0.05	$\mathbf{96.83 \pm 0.17}$	99.17 ± 0.14	$\mathbf{96.95 \pm 0.22}$
USPS → MNIST	99.91 ± 0.32	$\mathbf{95 \pm 0.25}$	98.3 ± 0.3	$\mathbf{95.92 \pm 0.21}$
Trained on MNIST	99.45 ± 0.02	$\mathbf{89.02 \pm 4.22}$	99.25 ± 0.02	$\mathbf{88.66 \pm 4.95}$
Trained on USPS	98.74 ± 0.11	$\mathbf{69.89 \pm 6.05}$	99.13 ± 0.32	$\mathbf{72.03 \pm 5.71}$

7.2 SVHN ⇔ MNIST

The next problem set we will use to showcase our model will be adapting the real-world image set of SVHN to the handwritten dataset of MNIST. Unlike the previous problem of MNIST ⇔ USPS, the SVHN⇔MNIST problem represents a much more significant domain shift. The SVHN digits, are all color images captured from the real world with different lighting, angles, and visual occlusions involved. The MNIST⇔USPS, on the other hand, had extremely clean images for their digits, with the only real hindrance being the handwriting of the people involved in the image capture. The next issue is because the SVHN images were captured from real-world images where the numbers are more than a single digit long, each image was cropped to be 32×32 centered around the number of interest, which often includes partial parts of other digits or background clutter. This makes the domain shift between the SVHN dataset and the MNIST dataset quite substantial.

Due to the extra difficulty associated with this task, we use the DTN classifier as our baseline task network. For all our experiments, we use the balanced SVHN dataset as outlined in section 4.4, but we will call it the SVHN dataset for brevity.

SVHN → MNIST

The first experiment we will be testing is the SVHN→MNIST case where the SVHN dataset is the source domain, and MNIST is the unlabelled target domain. Following the same suite as the MNIST⇔USPS experiments, the MNIST dataset will be subsampled at each epoch to match the number of samples between the SVHN and MNIST datasets. This experiment will outline how the model performs when crossing an extremely large domain shift gap between the source and target.

MNIST → SVHN

Following the experiments outlined above, we will see how the model performs in the reverse case of MNIST→SVHN. The difference for this example is that the unlabelled target domain is no longer the easier classification task. This could potentially cause issues as the GANs rely on the classifications from the task network to guide them through training, and if they underperform at the beginning of training, it can lead to performance issues. Another major issue with this direction is that it is easier to convert a multi-colored image to a black-and-white image than to go from a black-and-white image to a multi-colored one. When we go from SVHN→MNIST, the GANs need to primarily focus on making a black-and-white digit from a colored image. However, in the reverse direction, they need to figure out realistic backgrounds and colors that would improve the task network's performance on the target domain, creating a much

more complex training dynamic to learn any meaningful information that improves the classification.

Results

This section analyzes the results presented in table 7.4 and see where our model succeeds and fails. The table highlights the results of using the DTN Classifier as the task network and training the CycleGAN for 100 epochs. The target domains are both highlighted in bold. As mentioned above, all the places we reference SVHN refer to the balanced SVHN dataset.

The first adaptation direction we discuss is from SVHN→MNIST. In this case, the source domain with the labels generalizes better to the target domain without the labels. This can be seen from table 7.4 where the classifier trained on SVHN achieves an accuracy of $\sim 67\%$ without any adaptation whereas a classifier trained on MNIST achieves a much lower accuracy of $\sim 26\%$ on the SVHN dataset. After adaptation with our model, we are able to improve the target accuracy on the MNIST dataset from the aforementioned 67.41% to 85.47% which amounts to $\sim 20\%$ increase. While the average accuracy has increased similarly to the MNIST⇔USPS experiments, it is worth noting that the variance between the target accuracies is much higher for the SVHN→MNIST results, as can be seen by the standard deviations given in the table.

Figure 7.3: Example image output for our model on SVHN→MNIST. Row 1 is the original image from SVHN, row 2 is the fake MNIST image generated from the GAN and the final row is the reconstructed SVHN image.

From figure 7.3, we can see that the model does have the potential to perform adaptation between large domain shifts. In the above case, we can see how it adapts from SVHN→MNIST without any issues. However, the reverse case is not true, as we can see in table 7.4. In the case where we have the MNIST labels and not the SVHN labels, the model yields no significant performance increase, with both models achieving an accuracy of $\sim 26\%$ on the SVHN testing dataset. This has been the first failure case of our model so far.

Figure 7.4: Example image output for our model on MNIST→SVHN. The first row is the original image from MNIST, the second row is the fake SVHN image generated from the GAN and the final row is the reconstructed MNIST image.

We believe the issue comes from the fact that the task network classifier is unable to use the features learned from MNIST to give any useful information about the SVHN images. This is also supported by the fact that the classifier had a $\sim 19\%$ training accuracy on the target domain before being adapted. This leads us to believe that instead of guiding the GANs during training, it is instead giving it arbitrary information that yields no benefit during training. Another possible cause is that the mapping from MNIST→SVHN is too complex a task for the CycleGAN to learn as there are potentially infinite possible ways to color and fill the blank areas for the images, this is further supported in figure 7.4 where the generated SVHN images have the correct digit, but share very similar style and colouring for the foreground and background. However, in the reverse case, it is possible as the network needs to only map the background to black and the digit to white.

Table 7.4: SSGAN Domain Adaptation SVHN⇔MNIST Training and Testing data Accuracy ($\%$).

Name	Training Source	Training Target	Testing Source	Testing Target
SVHN → MNIST	92.59 ± 0.85	$\mathbf{84.14 \pm 8.01}$	88.79 ± 0.58	$\mathbf{85.47 \pm 7.58}$
MNIST → SVHN	99.97 ± 0.02	$\mathbf{23.19 \pm 2.68}$	99.18 ± 0.06	$\mathbf{26.58 \pm 4.06}$
Trained on SVHN	96.98 ± 0.07	$\mathbf{65.67 \pm 1.96}$	91.72 ± 0.12	$\mathbf{67.41 \pm 2.36}$
Trained on MNIST	99.99 ± 0.003	$\mathbf{19.23 \pm 1.52}$	99.28 ± 0.02	$\mathbf{26.23 \pm 2.39}$

7.3 Training time Variations

The time spent training a model can often significantly affect its performance. In some cases, models trained for more extended periods can outperform models trained for lower periods. In other cases overtraining the model can lead to issues like overfitting or the model reaching peak performance, and little to no benefit is gained from further

training. Often practitioners will implement methods like early stopping to stop training when some metric, like the difference in loss between epochs, is sufficiently small. However, it does not work well for models like GANs as the loss function is not always indicative of how the models are doing, and it can often spike or oscillate even when the model is performing as expected.

For the above reasons, our following experiments will compare how the model performs with different training lengths. We run all the experiments with the default hyperparameters with four different amounts of epochs for the CycleGAN training section: 50, 100, 200, and 500 epochs. We split this subsection into two parts. The first part contains the experiments on the three domains MNIST→USPS, USPS→MNIST, and SVHN→MNIST with their original hyperparameters and setups with the only change being the number of epochs for the CycleGAN section. For completeness, we also include tests on how well our model performs in the best of situations and will contain the MNIST→USPS and USPS→MNIST experiments with the DTN classifier as more powerful task networks can significantly influence the results. To further explore this, we will test our model on a variety of training times to see how the performance changes over time.

7.3.1 Variable epochs

The first set of experiments will involve using the original setups as used in table 7.1 and table 7.4 but with variable length epochs for the CycleGAN section of training. Initially, we trained each model for 100 epochs; we now include the experimental results for 50, 200, and 500 epochs in table 7.5. The numbers in bold represent the baseline number of epochs from which the other experiments were conducted.

MNIST→USPS

The first unsupervised domain adaptation problem test is the MNIST→USPS. In table 7.5, the first two rows correspond to the model accuracy on the test set for MNIST and the test set for USPS, respectively. From the first row, it is apparent that the model reaches its max capacity for performance on the source domain in a short number of epochs, as the difference between 50 and 500 epochs is marginal. This is most likely because the task network is trained on the source domains with labels to reach its asymptotic performance early in training. The second row shows how the performance consistently increases as we train for more epochs. However, we believe the model is near its max performance already at 50 epochs, as the increase is quite gradual compared to the increase in training time. The 50 epochs training is only $\sim 1.2\%$ less performance than the 500 epoch training while requiring 10x the amount of training.

USPS→MNIST

After MNIST→USPS, we consider the USPS→MNIST direction. Historically our other experiments have shown that this direction seems to have a larger domain shift than its reverse counterpart. This time it seems to be no different as the difference between 50 epochs and 100 epochs is quite substantial. Training the model for an extra 50 more epochs increases the accuracy on the test set by $\sim 5\%$; training it further gives more diminishing returns but is still substantial enough to make it arguably worth it. The difference between 100 epochs and 500 epochs is another extra $\sim 4\%$ accuracy. This leads us to believe that spending longer training epochs on this experiment is warranted as the model does not reach its asymptotic performance early on in training like the MNIST→USPS.

SVHN→MNIST

Our final experiment will be on the largest domain shifts, SVHN→MNIST. From the first row of the last two rows in table 7.5, we can see that the source accuracy is steady regardless of the training time. The target results, however, vary significantly between the 50 epochs and 100 epochs mark, with an increase of $\sim 8\%$. After the initial jump in performance from 50 to 100 epochs, we get a minor increase of $\sim 2\%$ from each increment of training time. This alludes to a similar result from the USPS→MNIST experiment that most of the performance can be achieved with just $\sim$100 epochs, but further training can be warranted to achieve maximum performance. A key caveat to the training regime is that different datasets have a different amount of training steps per epoch, as it is based on the size of the dataset. In the case of the MNIST⇔USPS experiments, the amount of training steps is small compared to the SVHN→MNIST experiments because the USPS dataset is only a fraction of the size of the other two. Therefore while the SVHN→MNIST experiment appears to require the same amount of epochs as the above, the actual length of training is far greater.

Table 7.5: SSGan results over different epochs. Bold indicates the baseline experiments.

Name	50 Epochs	100 Epochs	200 Epochs	500 Epochs
M→U LeNet MNIST	98.54 ± 0.05	$\mathbf{98.54 \pm 0.13}$	98.63 ± 0.07	98.70 ± 0.13
M→U LeNet USPS	95 ± 0.73	$\mathbf{95.41 \pm 0.47}$	95.63 ± 0.4	96.24 ± 0.34
U→M LeNet USPS	96.74 ± 0.54	$\mathbf{97.78 \pm 0.16}$	98.06 ± 0.2	98.41 ± 0.17
U→M LeNet MNIST	87.41 ± 1.14	$\mathbf{92.02 \pm 1.18}$	93.32 ± 0.34	94.95 ± 0.12
S→M DTN SVHN	89.67 ± 0.46	$\mathbf{88.79 \pm 0.58}$	89.71 ± 0.5	89.83 ± 0.69
S→M DTN MNIST	77.27 ± 6.69	$\mathbf{85.47 \pm 7.58}$	87.07 ± 7.47	89.78 ± 7.83

7.3.2 Variable Epochs with DTN Classifier

The second set of experiments include a similar setup as above, except we now use the more powerful DTN classifier for the MNIST⇔USPS experiments and vary the training time. This will give insight into how well extending training time works for high-performing and low-performing models, as well as showcase the potential of our model.

MNIST→USPS DTN Classifier

Once again, we can see from the first row in table 7.6 that the model's performance on the source accuracy does not improve by any significant margin as we increase the epochs. The target accuracy follows the same trend as in table 7.5 with even the improvements between each epoch increment being similar. However, because the DTN Classifier performs better on the target domain, we have achieved our all-time high so far on the MNIST→USPS experiment with a target accuracy of 98.03%.

USPS→MNIST DTN Classifier

Finally, we look at the USPS→MNIST experiment set with the DTN Classifier. Like the previous experiment, the DTN classifier offers a massive improvement in performance across the board. At the lowest epoch length, we get an improvement of 6% over the LeNet model. A slight difference between these results compared to the LeNet model is that each increase in epochs yields minimal improvement in accuracy after the first epoch increment. When we increase the model's training time from 50 to 100 epochs, we get a 2% improvement, but each interval afterward only increases the performance by $< 1\%$. We believe this means that by 100 epochs, the model has reached close to its max accuracy for these hyper-parameters, model, and data. Like in the previous case, the DTN classifier with 500 epochs achieves our highest performance on the USPS→MNIST domain with an accuracy of 96.77%.

Table 7.6: SSGan results over different epochs with DTN Classifier.

Name	50 Epochs	100 Epochs	200 Epochs	500 Epochs
M→U DTN Source	99.25 ± 0.02	$\mathbf{99.26 \pm 0.37}$	99.32 ± 0.12	99.40 ± 0.03
M→U DTN Target	96.56 ± 0.37	$\mathbf{97.04 \pm 0.14}$	97.38 ± 0.03	98.03 ± 0.14
U→M DTN Source	98.67 ± 0.16	$\mathbf{98.64 \pm 0.11}$	98.62 ± 0.08	98.80 ± 0.12
U→M DTN Target	93.34 ± 0.51	$\mathbf{95.5 \pm 0.55}$	96.05 ± 0.32	96.77 ± 0.09

7.3.3 Conclusion

Based on these experiments, we conclude that a training length of around 100 epochs for the small digit datasets is sufficient for achieving reasonable accuracy. For the MNIST→USPS dataset, one can argue that 50 epochs may also be sufficient. However, if computation resources are available, it may be beneficial to continue increasing the number of epochs, as performance can continue to improve. However, the training time may vary depending on the specific dataset and model architecture being used, as each problem is unique, and some models may reach optimal performance faster than others.

7.4 Ablation Study

The results thus far have shown how our model performs under different domains, architectures, and training regimes. However, for our next investigation, we conduct an ablation study on our model to assess the individual contributions of each component to the overall performance. This ablation study shows the importance of the different losses we introduce, give credence as to why we included each of them, and hopefully add further insight into how all these components interact.

The main sections we consider for our ablation study are the cycle-consistency loss, the task-networks semantic loss, the self-supervised loss, and the standard GAN loss. For these experiments, we trained our model on three of the unsupervised domain adaptation problems we have looked at so far: MNIST→USPS, USPS→MNIST, and SVHN→MNIST. We exclude the MNIST→SVHN experiment as our model has shown to give the minimal benefit that we do not believe there to be any insight to be gained from performing an ablation study on it. We use the same hyperparameters as in our previous experiments with the MNIST⇔USPS using the LeNet task-network variant and the SVHN→MNIST using the DTN Classifier network.

7.4.1 Results

Below we show the results of performing our ablation study on the different losses using tables and diagrams. Each table shows the performance of the model on the source (MNIST) and target (USPS) domains after each of the constituent losses has been removed. For example row 1 in figure 7.5a are the results of our model when we remove the cycle-consistency loss. Table 7.7 shows the results for the MNIST→USPS ablation study, table 7.8 shows for USPS→MNIST, and table 7.9 is for SVHN→MNIST. Further discussion on each of the individual results can be found below.

Table 7.7: Ablation Study: MNIST→USPS Testing data Accuracy (%)

Name	Source	Target	Source Delta	Target Delta
Cycle-Consistent Loss	98.56 ± 0.03	$\mathbf{94.62 \pm 0.43}$	$+0.02$	$\mathbf{-0.79}$
Semantic Loss	98.4 ± 0.21	$\mathbf{93.03 \pm 0.5}$	-0.14	$\mathbf{-2.38}$
GAN Loss	98.59 ± 0.07	$\mathbf{89.14 \pm 1.48}$	$+0.05$	$\mathbf{-6.27}$
Self-supervision Loss	98.71 ± 0.02	$\mathbf{94.62 \pm 0.97}$	$+0.17$	$\mathbf{-0.79}$
Cycle+Semantic Loss	98.54 ± 0.02	$\mathbf{84.84 \pm 2.81}$	$+0.00$	$\mathbf{-10.57}$
Baseline	98.54 ± 0.13	$\mathbf{95.41 \pm 0.47}$	—	—

Table 7.8: Ablation Study: USPS→MNIST Testing data Accuracy (%)

Name	Source	Target	Source Delta	Target Delta
Cycle-Consistent Loss	97.53 ± 0.14	$\mathbf{91.1 \pm 0.54}$	-0.25	$\mathbf{-0.92}$
Semantic Loss	97.58 ± 0.23	$\mathbf{92.75 \pm 0.86}$	-0.2	$\mathbf{+0.73}$
GAN Loss	97.67 ± 0.24	$\mathbf{70.52 \pm 1.2}$	-0.11	$\mathbf{-21.5}$
Self-supervision Loss	97.42 ± 0.14	$\mathbf{82.89 \pm 1.46}$	-0.36	$\mathbf{-9.13}$
Cycle+Semantic Loss	95.63 ± 0.24	$\mathbf{39.35 \pm 3.3}$	-2.15	$\mathbf{-52.67}$
Baseline	97.78 ± 0.16	$\mathbf{92.02 \pm 1.18}$	—	—

Table 7.9: Ablation Study: SVHN→MNIST Testing data Accuracy (%)

Name	Source	Target	Source Delta	Target Delta
Cycle-Consistent Loss	88.22 ± 1.42	$\mathbf{82.49 \pm 1.33}$	-0.57	$\mathbf{-2.98}$
Semantic Loss	70.64 ± 5.08	$\mathbf{63.57 \pm 2.39}$	-18.15	$\mathbf{-21.9}$
GAN Loss	91.74 ± 0.45	$\mathbf{69.2 \pm 3.83}$	$+2.95$	$\mathbf{-16.27}$
Self-supervision Loss	90.68 ± 0.04	$\mathbf{73.56 \pm 3.01}$	$+1.89$	$\mathbf{-11.91}$
Cycle+Semantic Loss	50.79 ± 15.54	$\mathbf{32.39 \pm 6.28}$	-38.00	$\mathbf{-53.08}$
Baseline	88.79 ± 0.58	$\mathbf{85.47 \pm 7.58}$	—	—

7.4.2 Cycle-Consistency Loss

The first loss we remove from training is the Cycle-consistency loss we formulated in equation (5.3). This loss's original function was to constrain the network to keep enough information about the original input to create a bijective mapping when transferring between domains. This bijective property allowed the GANs to learn image-to-image mapping in a completely unsupervised way.

In our experiments, we can note that in table 7.7 and table 7.8 that losing the cycle-consistency constraint seems to have a marginal decrease of $\sim 1\%$ performance, making it appear like it is not very valued. Even in table 7.9, the cycle-consistency loss has the least net performance loss when removed. However, it is still a substantial loss

to the final result showing it adds value in the more complex domains. The reason for this apparent lack of importance will be discussed later.

7.4.3 Semantic Loss

The following experiment we conduct is to remove the semantic loss. We originally formulated this loss to help guide the network for the intermediate domains, as the cycle consistency only deals with the beginning and final images, which has made a complete cycle. This led to potential issues of mode collapse and label flipping.

Table 7.7 appears to show a similar result as the cycle-consistency loss with the removal of the semantic loss only attributing to a reduction of about 2%. Contrasting to previous results, though, the performance slightly increased in table 7.8. However, this is within the experiment's margin of error and could easily be explained by the variance between tests. However, in table 7.9, we can see that the performance drastically reduces when the semantic loss is removed, with a total performance drop of $\sim 22\%$ and the issues of mode collapse and label flipping started to reoccur. We believe this to be caused by the extra difficulty and noise associated with the SVHN→MNIST problem set, which makes it more likely for the network to have mode collapse or label flipping. However, in the previous two experiments with MNIST⇔USPS, the domain shift is small enough that the extra semantic loss is not always needed.

7.4.4 GAN loss

The GAN loss in the context of this ablation study is the gradients provided by the discriminators to the generators to help guide it on whether it is creating realistic fake images. The discriminators were left untrained for these experiments, and we completely removed their loss to the generators.

In table 7.7, the total net performance drop does not seem that substantial at only $\sim 6\%$ from the baseline. However, considering how high the performance was before adaptation, this is essentially the same performance as no adaptation. The SVHN→MNIST shows expected results of a substantial drop in accuracy of $\sim 16\%$, bringing it almost on par with a no adaptation classifier. The more surprising result shown in table 7.8 is that for the USPS→MNIST case, removing the GAN loss drops the performance even below, performing no adaptation at all. The collection of these three experiments shows how vital the discriminator's loss is in guiding the model to generate realistic images.

7.4.5 Self Supervision Loss

By introducing the task network, which was only trained on the source dataset, we introduce a potential bias in which the networks will favor target images more closely related to the source domain. This can lead to issues where our model does not learn an accurate mapping to the actual target domain, which can be generalized.

To solve this potential issue, we introduce the self-supervision loss in 6.4, which starts training the task-network with fake generated target images to better accustom it in dealing with the target domain images and to provide better guidance to the generators. In this experiment, the self-supervision loss is removed to help highlight the impact that it provides in the training of the network.

The first set of results we will discuss are from table 7.7, from here we can see that the self-supervision loss seems to only account for a minor increase in performance between the baseline and the no self-supervision loss. This could be because the domain shift is small enough that the rest of the model can get high accuracy without needing self-supervision. The USPS→MNIST in table 7.8 shows a much more substantial difference in performance. The no self-supervision model only achieves an accuracy of $\sim 82.89\%$ on the target domain, whereas the full model achieves $\sim 92.02\%$. This difference of about $\sim 10\%$ accounts for a large portion of the gained accuracy and indicates how much the self-supervision benefits training. The last experiment of SVHN→MNIST shows a similar performance change as seen in table 7.9. The lack of self-supervision in the task-networks training accounts for a decrease of $\sim 12\%$ for the SVHN→MNIST experiment.

From the points above, we conclude that the self-supervision loss plays a vital role in the successful training of our model. Its benefit also seems to increase as the domain shift increases, creating a vital part of the training regime in all but the smallest cases of domain shift.

7.4.6 Cycle-consistency + Semantic Loss

The experiments in section 7.4.2 appear to show the cycle-consistency loss having minimal effect on the result. We believe this is caused by the semantic loss acting as a pseudo-constraint on maintaining the bijective mapping, as the task network penalizes the network for creating images with different semantic meanings. This is especially prevalent in the case of MNIST⇔USPS, as the domain shift is slight enough such that the task network is sufficiently accurate enough to guide the generators during early training.

To further investigate whether this is true, we tried an ablation study where we removed both the semantic loss and the cycle-consistency loss. If these losses truly were

independent, we would not expect much of a performance drop from removing both individually, as they did not appear to contribute much. However, as we can see from table 7.7, 7.8 and 7.9, all three of their performances dropped below no adaptation classifiers. With the USPS→MNIST dropping $\sim 53\%$ below the baseline method and $\sim 27\%$ below performing no adaptation at all. In figure 7.5, mode collapse afflicts all three problem sets, with the generators generating unusable results.

(a) MNIST→USPS (b) USPS→MNIST (c) SVHN→MNIST

Figure 7.5: Generators results when the cycle consistency and semantic loss are removed.

This further shows how important it is to have at least one of these losses to preserve the mapping between domains. While there is an argument to make for removing one of the losses, we believe they both provide sufficient benefit to the result. Especially when the problem domain is a much harder domain adaptation jump like in SVHN→MNIST.

7.5 Initial task-network Impact

A major concern brought to light in the experiments from table 7.4 was how poorly the model performed on the MNIST→SVHN compared to the SVHN→MNIST. One of our speculated reasons for this difference in performance was how well the pre-adaptation classifier worked on the target domain, allowing better-performing models to provide better guidance during training. In this section, we further explore this by analyzing the pre-adaptation models compared to the post-adaptation models.

As mentioned before, the standard deviations for the SVHN→MNIST were much higher than the MNIST⇔USPS counterpart. While the average accuracy was high, this increased variance between runs led to some experiments underperforming, with our lowest individual experiment result being 73.67% and others overperforming, with our highest being 94.95%. This extreme difference between the potential highs and lows for any given run makes understanding what impacts the final results even more crucial.

The first set of data we explore is the initial SVHN→MNIST experiments. As mentioned above, these experiments exhibited the highest levels of variance between re-

sults. Considering how it has one of the largest domain shifts, it makes it a good case study on studying the impact of the initial task network on the final result. The four main contributing factors we will be taking into account are the performance metrics of the task network before adaptation, and we will compare them to the final testing accuracy on the target dataset.

In figure 7.6, we can see by inspection that the data tends to be spread out for the most part but, in some cases, does exhibit signs of trends. Included in the figures, we added the lines of best fit to show the general trend of the data; however, this does not necessarily imply any correlation. The two strongest correlations appear to be the initial target training accuracy and the initial target test accuracy, with their distributions looking almost identical. While this is expected as the pre-adapted classifier has trained on none of the training or test data for the target domain, it is still a good sanity check that it is generalizing relatively well between the two datasets.

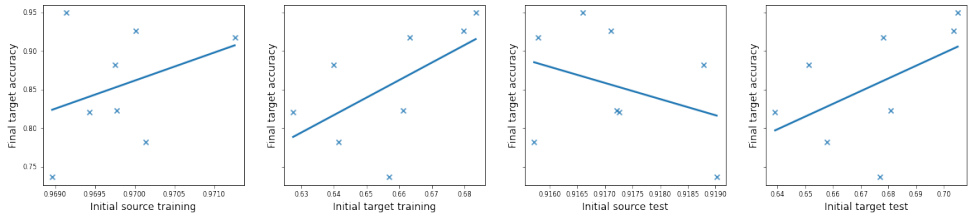


Figure 7.6: A scatter plot for each different initial accuracy metric on the x-axis and the final target test accuracy on the y-axis. Initial source training and initial target training are the pre-adaptation task-networks performance on the source and target domains training sets, respectively. Initial source test and Initial target test are the pre-adaptation network's performance on the source and target domains test sets respectively, ie the models testing accuracy on both the source and target domains before the model has performed any adaptation.

To further explore if there is any correlation or dependence between the variables, we will use the Pearson correlation coefficient to verify if our interpretations from above are corroborated. In figure 7.7, we plot the Person correlation coefficients for each metric in a bar chart. As we can see, the training and test accuracies on the training and testing sets have the highest correlation for how well the model will perform after adaptation. However, with the highest being an r value of 0.58, the best we can say is that it is medium to weakly correlated. Conversely, the source domain accuracies have approximately the same magnitude Pearson coefficient, but one is positive, and one is negative.

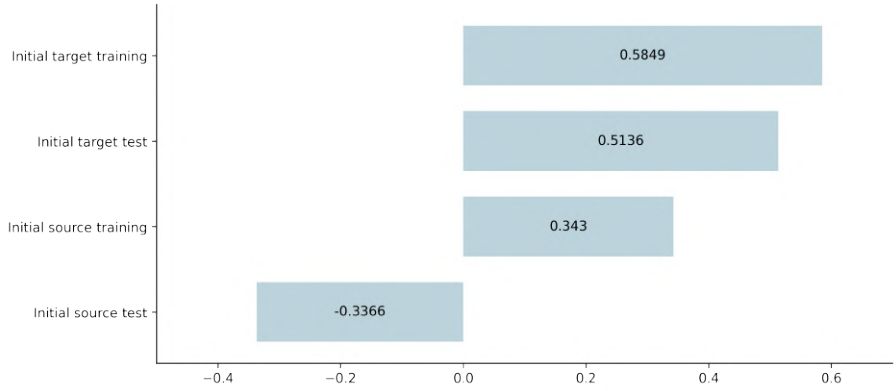


Figure 7.7: Pearson's correlation coefficient for each of the task-networks pre-adaptation performance metrics.

While the correlation values discussed above suggest that there is at least some correlation between the task-networks pre-adaptation performance on the target set and the final accuracy, the results are inconclusive alone as the correlation is not that strong and the sample size of our experiments for this setup is relatively small leading to potential outliers skewing the results. To further investigate, we include the same correlation analysis for the variable epoch experiments to see if they exhibit similar behavior.

The first performance metric we will discuss is the training set for the SVHN (source) dataset. From figure 7.8, we can see that all the runs seem to show very little correlation, with the highest having an r value of 0.343. This leads us to believe that in the case of SVHN→MNIST, the model's initial performance on the SVHN training set is not impactful on the final performance. A potential reason for this is that in all our experiments, the task networks performed extremely consistently on the source datasets, with most having a variance of less than 1.

We discuss the following two metrics together as they exhibit extremely similar trends. The Pearson coefficients for the MNIST (target) training and testing sets have almost identical distribution shapes across the variable epoch runs. Both these different performance metrics tend to show an extremely low correlation when the model has only trained for 50 epochs. But it steadily increases as we increase the training time, with the 500 epoch experiment showcasing a correlation coefficient of 0.82 and 0.86 for the MNIST training and testing set, respectively. Both these values suggest an extremely high positive correlation between the initial target performance metrics and the final model's performance.

The final metric is how well the task network performs on the SVHN test set before adaptation training has begun. From here, the results are more chaotic than the others, with the 50 epoch run suggesting a high positive correlation but the 100 epoch run suggesting a weakly negative correlation. The 200 epoch run and 500 epoch run suggest a weakly to medium positive correlation, but nothing can conclusively lead us

to believe they are related.

To summarise the discussion above, while we cannot make any claims about the causal relationship between these values from the statistics alone, we believe that the task-networks performance before it begins the cycle-consistency training section can impact the final model's results. This is especially important as the model is trained for longer as it approaches its asymptotic accuracy; the initial performance of the task network seems to have a more significant effect.

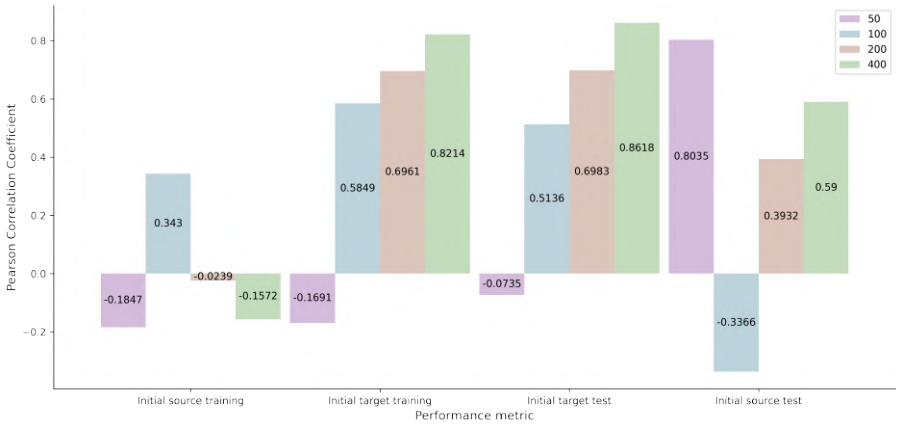


Figure 7.8: Pearson's correlation coefficient across all SVHN→MNIST variable epoch runs.

7.6 Discussion on Results

Following the results of our experiments above, we now conclude the chapter with an overall discussion about the results and how they relate to the research questions we originally stated. We will also compare how our proposed model performs to others from literature.

The first observation from the above experiments is that the model was successful on most domains it was tested on, with the only exception being the MNIST→SVHN. We attributed this failure because of two potential causes. The first is the MNIST-trained classifier's inability to generalize well enough to the SVHN domain, making it incapable of providing meaningful information to the generators during the early phases of training. This result is further backed by the results in section 7.5, which seem to show a correlation between the performance of the pre-adapted task network and the final network's performance. In addition, we speculate that the increased difficulty of going from MNIST→SVHN makes it much harder to learn than the reverse direction.

On the other hand, the other domains all significantly improved the classifiers' performance on the target domains inputs, with the MNIST⇔USPS experiments bringing

the final classifiers' accuracy close to the performance of a model trained on the target dataset. While the SVHN→MNIST experiment still had a bit to go before reaching oracle-level accuracy, its increase in performance over the non-adapted method was still substantial. A major aspect observed about the SVHN→MNIST experiments was the very high variance between results, which appears to be correlated to how well the initial classifier performed. This leads us to believe that finding a reliable method of improving the classifier pre-adaptation even marginally can lead to even higher results than the ones we achieved. We consider this a promising avenue for future research to get the best from our method.

In table 7.10, we compare our method's performance to other models for the same domains. The rows labeled source are the baseline models trained on the source domain without any adaptation. At the top, we include the results given by Hoffman *et al.* [2018], but for completeness, we also include the models we used as our baselines further down. At the bottom, the rows labeled target is the oracle classifiers trained on the target dataset. We consider these to be the general target for what would be close to a perfect classifier.

From table 7.10, we can see how our model achieves competitive performance compared to various methods. With our MNIST→USPS model with the DTN classifier achieving close to French *et al.* [2017] and Wang *et al.* [2021]. Even the LeNet model achieves high-level performance with only being beaten out by the same models mentioned above. The USPS→MNIST model with the DTN classifier achieves comparable results to Hoffman *et al.* [2018] and Wang *et al.* [2021] with only a marginal increase in performance. Finally, our model's performance on the SVHN→MNIST dataset achieves similar accuracies as Hoffman *et al.* [2018] with a slight decrease in performance.

In conclusion, our model has performed exceptionally well on the small digits datasets on various domain shifts. This is a good indication of how well our method can perform on domain adaptation problems with a classification downstream task. Another key result that does not get highlighted in the results above is how our model maintains accuracy on the source domain and learning the target domain; while this may not always be a needed aspect, in certain real-world use cases, it will be a significant feature to have.

Table 7.10: Comparison of results.

Name	MNIST→USPS	USPS→MNIST	SVHN→MNIST
Source [Hoffman *et al.* 2018]	82.2 ± 0.8	69.6 ± 3.8	67.1 ± 0.6
DANN [Ganin *et al.* 2016]	-	-	73.6
DTN [Taigman *et al.* 2016]	-	-	84.4
CoGAN [Liu and Tuzel 2016]	91.2	89.1	-
ADDA [Tzeng *et al.* 2017]	89.4 ± 0.2	90.1 ± 0.8	76.0 ± 1.8
ADR [Saito *et al.* 2017]	93.2 ± 2.5	93.1 ± 1.3	95.0 ± 1.9
Self-Ensemble [French *et al.* 2017]	98.23 ± 0.13	99.54 ± 0.04	99.26 ± 0.05
Cycada+pixel [Hoffman *et al.* 2018]	95.6 ± 0.2	96.4 ± 0.1	70.3 ± 0.2
Cycada+feat [Hoffman *et al.* 2018]	95.6 ± 0.2	96.5 ± 0.1	90.4 ± 0.4
DFA [Wang *et al.* 2021]	98.6 ± 0.1	96.6 ± 0.2	98.9 ± 0.2
Source Lenet (Ours)	86 ± 1.03	66.22 ± 2.64	-
Source DTN (Ours)	90.39 ± 0.68	74.2 ± 0.29	67.41 ± 2.36
SSGAN LeNet	96.24 ± 0.34	94.95 ± 0.12	-
SSGAN DTN	98.03 ± 0.14	96.77 ± 0.09	89.78 ± 7.83
Target LeNet	96.42 ± 0.18	98.64 ± 1.18	-
Target DTN	98.79 ± 0.08	99.28 ± 0.02	99.28 ± 0.02

Chapter 8

Semantic Segmentation Domain Adaptation

Following the success of the proposed model on the standard classification task, we will now explore using it for more difficult tasks. The next task we will test on will be the dense prediction task of semantic segmentation. In this problem, the classifier needs to assign a label to each individual pixel instead of one label for the entire image.

Dense prediction problems have a wide variety of applications and is especially useful in problems that involve scene understanding like robotics or self driving cars. One such popular problem set for this is the urban scene semantic segmentation dataset Cityscapes [Cordts *et al.* 2016]. The purpose of this dataset is to help train models in dense prediction tasks like semantic segmentation and instance segmentation in common city scenes as seen from a car. While this dataset is one of the best of its kind, it still has a lot of potential problems.

The first issue is how difficult it is to get labels for dense prediction tasks like semantic segmentation. Instead of having a labeler look at an image and giving a single label, they have to go through the entire image and label every pixel. This becomes exceptionally difficult if high precision accuracy is required, as often multiple labelers will be asked to label the same images with a majority score being used to decide the final outcome. The next issue is ensuring the data collected is representative of all the potential data that could be seen during the application of the models. The Cityscapes dataset for example, was captured in European cities which could lead to issues when the model is tried on other cities around the world. Further complexity is added with external factors like weather conditions or rarely occurring classes, it can be difficult to ensure that dataset properly encapsulates all the relevant information.

A common attempt at solving this problem is to use synthetically generated data. This gives the user the benefit of generating as much labeled data as needed without the cost associated with real-world datasets. Generating data synthetically also gives the

user control over how the data is generated, such as choosing specific classes, weather conditions or time of day. However, the major restricting factor in using synthetically generated data is most of the time the models trained on the synthetic data do not generalize well to the intended domain making it essentially useless. This performance degradation is caused by the domain shift between the generated data and the real world data. Making this a perfect problem to test or model on.

8.1 GTA5→Cityscapes

The GTA5 dataset is a synthetically generated dataset created to mimic the Cityscapes dataset. We will be using the GTA5 dataset and Cityscapes dataset as the benchmark for testing if our model performs well on the Synthetic→Real domain adaptation problem and the semantic segmentation problem set.

8.1.1 Experimental Design

The training regime will follow mostly the same as used for the small digit experiments with alterations to the networks architectures used. The generators have been increased in size as advised in the original paper [Zhu *et al.* 2017]. The task network in this case will be a semantic segmentation model, making it also an image-to-image network. To evaluate how well our model performs on a variety of different task network architectures. These include the VGG network as used in Hoffman *et al.* [2018], the UNet architecture [Ronneberger *et al.* 2015], and the DeepLabv3 architecture with a ResNet-101 backbone [Chen *et al.* 2017b].

The VGG network and UNet were pretrained on the GTA5 dataset for 50 epochs, while the DeepLabv3 model was pre-trained on the GTA5 dataset in two phases. The first phase trained the network for 30 epochs on the GTA5 dataset and the second phase freezes the batch normalization layers and trains it for another 30 epochs as recommended by the original paper [Chen *et al.* 2017b]. The CycleGAN section was then trained for 100 epochs in the same manner as used in chapter 6. The semantic loss is calculated in a similar way equation (6.3), except each pixel is assigned a label instead a single label for the entire image.

8.1.2 Results

Table 8.1 shows the results of our experiments on the GTA5→Cityscapes experiments. The first three rows represent the models that are only trained on the source domain, in this case, the GTA5 dataset. Each row represents a different task-network architecture

which is stipulated in the second column. Rows 4,5 and 6 represent three other unsupervised domain adaptation models used for GTA5→Cityscapes, with HRDA [Hoyer *et al.* 2022] being the current state of the art. The following three rows represent our model's performance on the Cityscapes dataset while only using the labeled images from GTA5 and unlabeled images from Cityscapes, again there is one row for each task network architecture. The bottom 3 rows are the performance of the models trained with labeled Cityscapes images.

From table 8.1, we can see that we can gain a substantial increase in performance when using a VGG backbone for our task network. The mean intersection over union (mIoU) score increased from 17.9 to 34.7 compared to the no adaptation classifier, and the pixel accuracy also increased from 54% to 80.2%. On the other hand, the UNet and DeepLab architectures have less of an improvement. While there is still an increase over the non-adapted method, it is only a 5% increase in the mIoU.

The difference in performance increases between the different models leads us to believe that further research and insight could be done in our model's semantic segmentation use case. We also believe that further research can be done in incorporating the information the networks learn during the model's training process through methods like weight sharing and shared feature extractors to learn shared representations of the data better and further bridge the gap between domains.

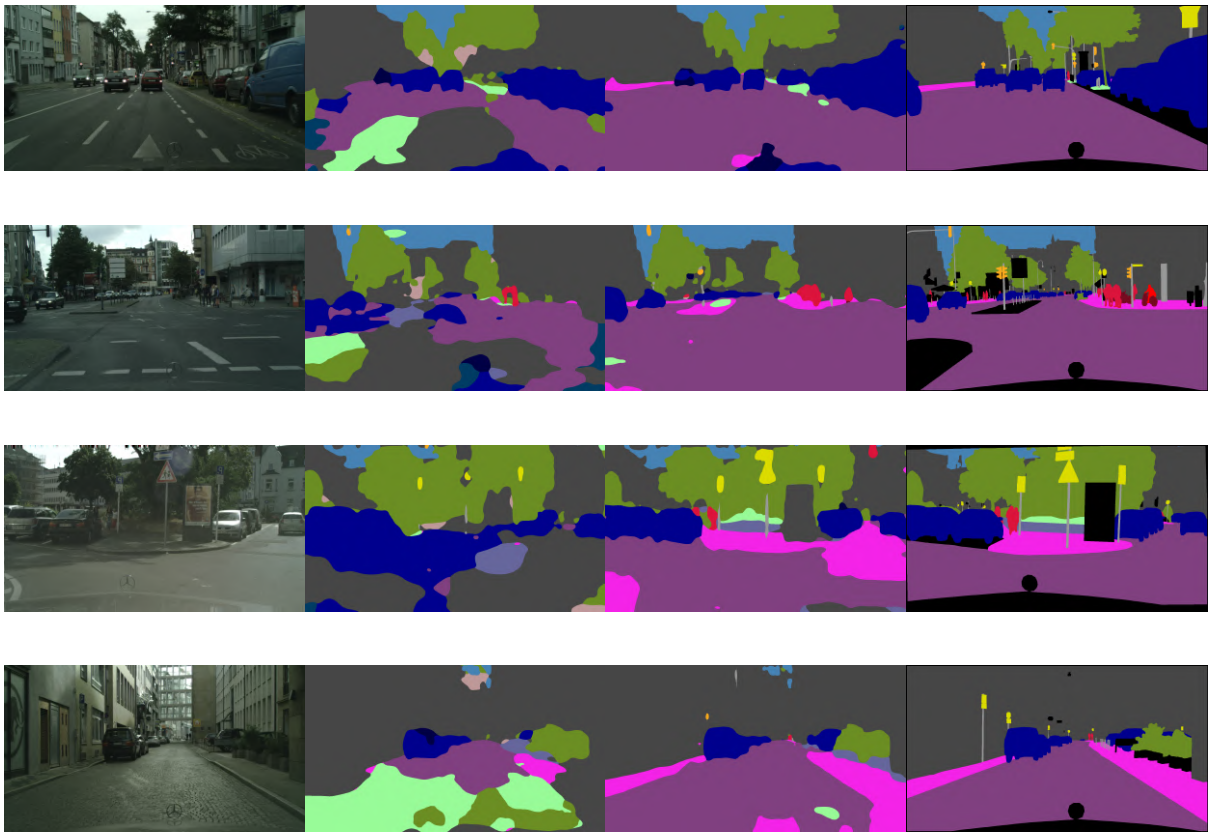
Some visual examples of how the model performs can be seen in figure 8.1. From the images, we can see how the model that was only trained on the GTA5 dataset struggles with classifying many of the pixels correctly, with most of the image being chaotic and noisy. Our model that can more accurately predict the classes for many pixels, and the class boundaries in the image become more defined.

	architecture	road	sidewalk	building	wall	fence	pole	traffic light	traffic sign	vegetation	terrain	sky	person	rider	car	truck	bus	train	motorbike	bicycle	mIoU	Pixel acc
Source [Hoffman *et al.* 2018]	VGG	26.0	14.9	65.1	5.5	12.9	8.9	6.0	2.5	70.0	2.9	47.0	24.5	0.0	40.0	12.1	1.5	0.0	0.0	0.0	17.9	54.0
Source	Unet	4.5	3.9	48.8	2.2	0	10.1	0	0	55.3	1.5	48.1	1.4	0	13.9	0.9	0.02	0	0	0	10.1	34
Source	DL101	55.1	21.7	52.9	15.0	18.3	18.6	34.2	16.7	81.8	23.0	72.7	56.2	7.2	73.5	27.2	28.8	0.1	17.7	5.1	32.9	71.6
FCNs in the wild [Hoffman *et al.* 2016]	VGG	70.4	32.4	62.1	14.9	5.4	10.9	14.2	2.7	79.2	21.3	64.6	44.1	4.2	70.4	8.0	7.3	0.0	3.5	0.0	27.1	—
Cycada [Hoffman *et al.* 2018]	VGG	85.2	37.2	76.5	21.8	15.0	23.8	22.9	21.5	80.5	31.3	60.7	50.5	9.0	76.9	17.1	28.2	4.5	9.8	0.0	35.4	83.6
HRDA [Hoyer *et al.* 2022]	DL101	96.4	74.4	91	61.6	51.5	57.1	63.9	69.3	91.3	48.4	94.2	79	52.9	93.9	84.1	85.7	75.9	63.9	67.5	73.8	—
Ours (Cityscapes)	VGG	74.7	28.4	73	18.8	13.1	10.6	23.3	16.6	82.9	27.3	70.6	51.2	17.9	80	19	17.9	1.3	21.9	9.8	34.7	80.2
Ours (Cityscapes)	Unet	17.2	11.1	45.0	1.4	0.4	18.5	0	0	68.3	4.0	65.3	19.4	0	34.9	1.0	0.02	0	0	0	15.1	48.6
Ours (Cityscapes)	DL101	55.5	20.7	69.2	16.3	20.8	25.3	37.7	34.1	83.9	29.0	80.7	58.1	13.1	69.8	18.3	15.4	5.0	24.9	20.7	36.8	73.4
Target [Hoffman *et al.* 2018]	VGG	96.4	74.5	87.1	35.3	37.8	36.4	46.9	60.1	89.0	54.3	89.8	65.6	35.9	89.4	38.6	64.1	38.6	40.5	65.1	60.3	93.1
Target	Unet	88.6	44.1	70.1	0	0	25.6	0	0.5	83.2	36.5	73.7	32.6	0	62.7	0	0	0	0	0	27.2	84.1
Target	DL101	97.5	80.1	89.7	47.6	52.2	46.6	51	63.1	90.1	58.9	91.4	72.3	49.7	92.4	71.5	76.6	61.7	56.3	68.3	69.3	94.5

Table 8.1: IoU scores for GTA5→Cityscapes experiments. The results for source and target were taken from Hoffman *et al.* [2018] as they aligned very closely with our own experiments. The only deviation was our target results were $\sim 2\%$ lower on average.

In figure 8.2, we show the qualitative results of the image-to-image process. The images shown are the generator's output given an image from the GTA5 dataset (source domain). The first significant difference between the two adaptation phases is how drastically the lighting changes. The generated Cityscapes images are much darker and have less vibrance than the GTA5 counterparts. This suggests that the images in the Cityscapes datasets are darker on average. This could be caused by the types of cameras

used, lighting conditions, time of day, or because the GTA5 game is generally brighter to make gameplay more enjoyable. Another noticeable effect is how the generators tend to remove a lot of reflections and granular details from the roads and buildings. Besides the perceptual differences we can see as humans, the network would have also altered very subtle changes that would significantly impact the classifier.



Original Image | Source Only Model | Our Model | Ground Truth

Figure 8.1: Semantic segmentation results from the GTA5→Cityscapes experiment.

GTA5 image GTA5→Cityscapes

Figure 8.2: Qualitative results from the GTA5→Cityscapes experiment.

Chapter 9

Conclusion

In this research, we proposed a new method for alleviating the performance degradation caused by domain shift in an unsupervised manner. Domain shift occurs when there is a discrepancy between the training data and the new data the model receives. In the case of visual domain shift, this discrepancy can be caused by many different reasons, from other recording equipment being used, different generating processes for the data, new locations, lighting, weather conditions, and many more factors.

We learn mappings between the source and target domains by utilizing the ability of CycleGAN [Zhu *et al.* 2017] to perform unpaired image-to-image translation. We extended the CycleGAN architecture by utilizing a semantic loss provided by a task network with the same downstream task as the final classifier. The task network provides early guidance in the generating process. We concurrently trained the task network with the CycleGAN with generated images to continuously improve its performance on the target domain, which would provide better guidance for the generators.

The model was experimentally shown to be successful in most of its use cases, with positive results coming from various domains, datasets, architectures, and hyperparameters. The method was also found to work for classification, semantic segmentation and synthetic to real adaptation. The only failure case was the MNIST→SVHN domain adaptation problem. We concluded that the main contributing factor for this failure case is probably due to the task network's initial performance being low and the generators' inability to perform the adaptation in a robust and generalizable way. We further found evidence that the model's final performance on the target domain was correlated to the task network's initial performance, leading us to believe finding a reliable way to improve the initial task network, even marginally, would be incredibly beneficial and a promising avenue for future research.

References

[Aljundi and Tuytelaars 2016] Rahaf Aljundi and Tinne Tuytelaars. Lightweight unsupervised domain adaptation by convolutional filter reconstruction. In *European Conference on Computer Vision*, pages 508–515. Springer, 2016.

[Arjovsky *et al.* 2017] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

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

[Bousmalis *et al.* 2017] Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan, and Dilip Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3722–3731, 2017.

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

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

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

[citation needed 2022] citation needed. Still needs citation. In *citation needed*, page citation needed, 2022.

[Cordts *et al.* 2016] 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 *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[Csurka 2017] Gabriela Csurka. Domain adaptation for visual applications: A comprehensive survey. *arXiv preprint arXiv:1702.05374*, 2017.

[Deng *et al.* 2009] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[Donahue *et al.* 2014] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pages 647–655. PMLR, 2014.

[Eldan and Shamir 2016] Ronen Eldan and Ohad Shamir. The power of depth for feed-forward neural networks. In *Conference on learning theory*, pages 907–940. PMLR, 2016.

[French *et al.* 2017] Geoffrey French, Michal Mackiewicz, and Mark Fisher. Self-ensembling for visual domain adaptation. *arXiv preprint arXiv:1706.05208*, 2017.

[Ganin *et al.* 2016] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17(1):2096–2030, 2016.

[Gatys *et al.* 2015] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[Gatys *et al.* 2016] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423, 2016.

[Ghifary *et al.* 2015] Muhammad Ghifary, W Bastiaan Kleijn, Mengjie Zhang, and David Balduzzi. Domain generalization for object recognition with multi-task autoencoders. In *Proceedings of the IEEE international conference on computer vision*, pages 2551–2559, 2015.

[Ghifary *et al.* 2016] Muhammad Ghifary, W Bastiaan Kleijn, Mengjie Zhang, David Balduzzi, and Wen Li. Deep reconstruction-classification networks for unsupervised domain adaptation. In *European conference on computer vision*, pages 597–613. Springer, 2016.

[Goodfellow *et al.* 2014] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[Goodfellow *et al.* 2020] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[Goodfellow 2016] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

[Gulrajani *et al.* 2017] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*, 2017.

[He *et al.* 2016a] Di He, Yingce Xia, Tao Qin, Liwei Wang, Nenghai Yu, Tie-Yan Liu, and Wei-Ying Ma. Dual learning for machine translation. *Advances in neural information processing systems*, 29, 2016.

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

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

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

[Hornik *et al.* 1989] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[Hoyer *et al.* 2022] Lukas Hoyer, Dengxin Dai, and Luc Van Gool. Hrda: Context-aware high-resolution domain-adaptive semantic segmentation. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXX*, pages 372–391. Springer, 2022.

[Huang *et al.* 2006] Jiayuan Huang, Arthur Gretton, Karsten Borgwardt, Bernhard Schölkopf, and Alex Smola. Correcting sample selection bias by unlabeled data. *Advances in neural information processing systems*, 19, 2006.

[Huang *et al.* 2018] Sheng-Wei Huang, Che-Tsung Lin, Shu-Ping Chen, Yen-Yi Wu, Po-Hao Hsu, and Shang-Hong Lai. Auggan: Cross domain adaptation with gan-based data augmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 718–731, 2018.

[Hull 1994] Jonathan J. Hull. A database for handwritten text recognition research. *IEEE Transactions on pattern analysis and machine intelligence*, 16(5):550–554, 1994.

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

[Johnson *et al.* 2016] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer, 2016.

[Krizhevsky *et al.* 2017] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

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

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

[Li *et al.* 2021] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE transactions on neural networks and learning systems*, 2021.

[Liang *et al.* 2022] Xuefeng Liang, Xingyu Liu, and Longshan Yao. Review–a survey of learning from noisy labels. *ECS Sensors Plus*, 1(2):021401, 2022.

[Liu and Tuzel 2016] Ming-Yu Liu and Oncel Tuzel. Coupled generative adversarial networks. *Advances in neural information processing systems*, 29, 2016.

[Liu *et al.* 2017] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. Unsupervised image-to-image translation networks. In *Advances in neural information processing systems*, pages 700–708, 2017.

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

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

[Long *et al.* 2018] Mingsheng Long, Zhangjie Cao, Jianmin Wang, and Michael I Jordan. Conditional adversarial domain adaptation. *Advances in neural information processing systems*, 31, 2018.

[Mao *et al.* 2017] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2794–2802, 2017.

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

[Netzer *et al.* 2011] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[Osborne and others 2004] Martin J Osborne et al. *An introduction to game theory*, volume 3. Oxford university press New York, 2004.

[Pan *et al.* 2010] Sinno Jialin Pan, Ivor W Tsang, James T Kwok, and Qiang Yang. Domain adaptation via transfer component analysis. *IEEE transactions on neural networks*, 22(2):199–210, 2010.

[Radford *et al.* 2015] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[Richter *et al.* 2016a] Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II 14*, pages 102–118. Springer, 2016.

[Richter *et al.* 2016b] 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, *European Conference on Computer Vision (ECCV)*, volume 9906 of *LNCS*, pages 102–118. Springer International Publishing, 2016.

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

[Russakovsky *et al.* 2015] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[Russo *et al.* 2018] Paolo Russo, Fabio M Carlucci, Tatiana Tommasi, and Barbara Caputo. From source to target and back: symmetric bi-directional adaptive gan. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8099–8108, 2018.

[Saharia *et al.* 2022] Chitwan Saharia, Jonathan Ho, William Chan, Tim Salimans, David J Fleet, and Mohammad Norouzi. Image super-resolution via iterative refinement. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[Saito *et al.* 2017] Kuniaki Saito, Yoshitaka Ushiku, Tatsuya Harada, and Kate Saenko. Adversarial dropout regularization. *arXiv preprint arXiv:1711.01575*, 2017.

[Saito *et al.* 2018] Kuniaki Saito, Kohei Watanabe, Yoshitaka Ushiku, and Tatsuya Harada. Maximum classifier discrepancy for unsupervised domain adaptation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3723–3732, 2018.

[Saxena and Cao 2021] Divya Saxena and Jiannong Cao. Generative adversarial networks (gans) challenges, solutions, and future directions. *ACM Computing Surveys (CSUR)*, 54(3):1–42, 2021.

[Sharma *et al.* 2012] Vidushi Sharma, Sachin Rai, and Anurag Dev. A comprehensive study of artificial neural networks. *International Journal of Advanced research in computer science and software engineering*, 2(10), 2012.

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

[Stutz *et al.* 2019] David Stutz, Matthias Hein, and Bernt Schiele. Disentangling adversarial robustness and generalization. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[Sugiyama *et al.* 2007] Masashi Sugiyama, Shinichi Nakajima, Hisashi Kashima, Paul Buenau, and Motoaki Kawanabe. Direct importance estimation with model selection and its application to covariate shift adaptation. *Advances in neural information processing systems*, 20, 2007.

[Sun and Saenko 2016] Baochen Sun and Kate Saenko. Deep coral: Correlation alignment for deep domain adaptation. In *European conference on computer vision*, pages 443–450. Springer, 2016.

[Sun *et al.* 2016] Baochen Sun, Jiashi Feng, and Kate Saenko. Return of frustratingly easy domain adaptation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[Szegedy *et al.* 2013] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[Taigman *et al.* 2016] Yaniv Taigman, Adam Polyak, and Lior Wolf. Unsupervised cross-domain image generation. *arXiv preprint arXiv:1611.02200*, 2016.

[Tarvainen and Valpola 2017] Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. *Advances in neural information processing systems*, 30, 2017.

[Tzeng *et al.* 2014] Eric Tzeng, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell. Deep domain confusion: Maximizing for domain invariance. *arXiv preprint arXiv:1412.3474*, 2014.

[Tzeng *et al.* 2015] Eric Tzeng, Judy Hoffman, Trevor Darrell, and Kate Saenko. Simultaneous deep transfer across domains and tasks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4068–4076, 2015.

[Tzeng *et al.* 2017] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7167–7176, 2017.

[Vincent *et al.* 2008] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.

[Wang and Deng 2018] Mei Wang and Weihong Deng. Deep visual domain adaptation: A survey. *Neurocomputing*, 312:135–153, 2018.

[Wang *et al.* 2021] Jing Wang, Jiahong Chen, Jianzhe Lin, Leonid Sigal, and Clarence W de Silva. Discriminative feature alignment: Improving transferability of unsupervised domain adaptation by gaussian-guided latent alignment. *Pattern Recognition*, 116:107943, 2021.

[Wu *et al.* 2017] Xian Wu, Kun Xu, and Peter Hall. A survey of image synthesis and editing with generative adversarial networks. *Tsinghua Science and Technology*, 22(6):660–674, 2017.

[Yan *et al.* 2017] Hongliang Yan, Yukang Ding, Peihua Li, Qilong Wang, Yong Xu, and Wangmeng Zuo. Mind the class weight bias: Weighted maximum mean discrepancy for unsupervised domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2272–2281, 2017.

[Yi *et al.* 2017] Zili Yi, Hao Zhang, Ping Tan, and Minglun Gong. Dualgan: Unsupervised dual learning for image-to-image translation. In *Proceedings of the IEEE international conference on computer vision*, pages 2849–2857, 2017.

[Yu and Koltun 2015] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[Yu *et al.* 2015] Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

[Zadrozny 2004] Bianca Zadrozny. Learning and evaluating classifiers under sample selection bias. In *Proceedings of the twenty-first international conference on Machine learning*, page 114, 2004.

[Zhang 2000] Guoqiang Peter Zhang. Neural networks for classification: a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 30(4):451–462, 2000.

[Zhao *et al.* 2019] Shanshan Zhao, Huan Fu, Mingming Gong, and Dacheng Tao. Geometry-aware symmetric domain adaptation for monocular depth estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9788–9798, 2019.

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