FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Combining simulated and real images in deep learning

Pedro Xavier Tavares Monteiro Correia de Pinho

Mestrado Integrado em Engenharia Informática e Computação

Supervisors: Luís Filipe Teixeira & Isabel Rio-Torto

July 24, 2021

Combining simulated and real images in deep learning

Pedro Xavier Tavares Monteiro Correia de Pinho

Mestrado Integrado em Engenharia Informática e Computação

July 24, 2021

Abstract

To train a deep learning (DL) model, considerable amounts of data are required to generalize to unseen cases successfully. Furthermore, such data is often manually labeled, making its annotation process costly and time-consuming. We propose the use of simulated data, obtained from simulators, as a way to surpass the increasing need for annotated data.

Although the use of simulated environments represents an unlimited and cost-effective supply of automatically annotated data, we are still referring to synthetic information. As such, it differs in representation and distribution comparatively to real-world data. The field which addresses the problem of merging the useful features from each of these domains is called domain adaptation (DA), a branch of transfer learning (TL). In this field, several advances have been made, spanning across divergence-based, adversarial-based and reconstruction-based methods. In this work, we combine multiple techniques across all DA branches. Being the simplest form of TL, fine-tuning provides a straightforward way to better generalize to the target domain. Employing a method called domain mapping and using pseudo-labels to obtain new data sources allows us to decrease the distance to the real domain. Domain mapping makes use of Generative Adversarial Networks (GANs) to convert images from one domain to another preserving the original images' information. While pseudo-labels takes advantage of an already trained model to further increase its robustness. With previous approaches, training data was being sourced from already existent datasets, and the usage of simulators as a means to obtain new observations was an alternative not fully explored. We survey possible DA techniques and apply them to this context of obtaining simulated data with the purpose of training DL models.

Stemming from a previous project, aimed to automate quality control at the end of a vehicle's production line, a proof-of-concept was developed. Previously, a DL model that identified vehicle parts was trained using only data obtained through a simulator. By obtaining real counterparts of simulated images through domain mapping models, we can maintain original annotations and gain access to an additional source of labeled data. Two domain mapping models were considered for this step: CycleGAN and CUT. We progressively fine-tuned the vehicle parts detector model with the newly attained converted samples and with pseudo-labels from real images. Furthermore, we introduced a global NMS post-processing step to more effectively make use of our model's predictions. Compared to the baseline model, trained only on simulated images, this process yielded us a 14.7% and a 28.6% example-based F1-score improvement when fine-tuning with CycleGAN and CUT converted images, respectively. Further fine-tuning with real images' pseudo-labels provided an additional 4% example-based F1-score improvement. These results demonstrate the potential of using simulated data alongside domain mapping approaches in real-world applications where data is scarce. Therefore, we believe the use of simulated data can be expanded to new areas where, until now, the usage of DL was not feasible due to the constraints imposed by data collection.

Keywords: transfer learning, domain adaptation, deep learning, neural networks, simulated data

Resumo

De modo a treinar modelos de *deep learning* (DL), são necessárias enormes quantidades de dados de modo a generalizar para novos casos. Além disso, esses dados são normalmente anotados manualmente, tornando este processo demorado e dispendioso. Propomos a utilização de dados provenientes de simuladores de modo a ultrapassar a crescente necessidade de dados anotados.

Embora a utilização de simuladores signifique um fornecimento ilimitado e rentável de dados automaticamente anotados, continuamos a lidar com informação sintética. Como tal, difere em representação e distribuição face a dados reais. A área que aborda o problema de junção de informação útil proveniente de dois domínios denomina-se *domain adaptation* (DA), um ramo de *transfer learning* (TL). Neste campo, foram feitos vários avanços, desde métodos divergentes, adversariais e baseados em reconstrução. Neste trabalho, combinamos múltiplas técnicas de todos os ramos de DA. Sendo a forma mais simples de TL, *fine-tuning* proporciona uma forma simples de generalizar para o domínio alvo. A utilização de mapeamento de domínios e a utilização de *pseudo-labels* para obter novas fontes de dados permite-nos diminuir a distância ao domínio real. Mapeamento de domínios usa redes neuronais adversariais para converter imagens de um domínio para outro, preservando informação das imagens originais. Enquanto *pseudo-labels* tiram partido de um modelo já treinado para o tornar ainda mais robusto. Abordagens anteriores, usam dados de treino já existentes, e a utilização de simuladores como meio de obter novas observações era uma alternativa ainda não totalmente explorada. Levantamos possíveis técnicas de DA e aplicamo-las a este contexto de obtenção de dados com o objectivo de treinar modelos DL.

Com base num projecto anterior, que automatizava o controlo de qualidade no final da linha de produção de um veículo, foi desenvolvida uma prova de conceito. Anteriormente, um modelo detetor que identificava peças de veículos foi treinado utilizando apenas dados simulados. Através de mapeamento de domínio conseguimos ter acesso a uma fonte adicional de dados, ao converter images simuladas e obter versões reais das mesmas. Dois modelos foram considerados para esta etapa: CycleGAN e CUT. Progressivamente fizemos *fine-tune* do modelo detector com as amostras convertidas e com *pseudo-labels* de imagens reais. Além disso, introduzimos um pós-processamento de *NMS* para utilizar mais eficazmente as previsões obtidas. Em comparação com o modelo treinado apenas em imagens simuladas, este processo rendeu-nos uma melhoria de 14,7% e 28,6% de F1-score ao utilizar images convertidas com CycleGAN e CUT, respectivamente. Subsequentemente, dando *fine-tune* com *pseudo-labels* de imagens reais obtemos uma melhoria adicional de 4% de F1-score. Estes resultados demonstram o potencial da utilização de dados simulados juntamente com abordagens de DA em aplicações do mundo real onde dados são escassos. Acreditamos que isto pode ser alargado a novas áreas onde, até agora, a utilização de DL não era viável devido às limitações impostas pela recolha de dados.

Palavras-chave: transferência de conhecimento, adaptação de domínios, aprendizagem profunda, redes neuronais, dados simulados

Agradecimentos

Gostaria de agredecer aos meus orientadores Professor Luís Teixeira, pela orientação dada ao longo de todo este projeto, e estudante de doutoramento Isabel Rio-Torto, pela constante simpatia e disponibilidade em ajudar a resolver vários desafios que surgiram ao longo deste período.

Aos meus amigos e por todas as sessões de desanuviamento que fizemos ao final de quase todos os dias para tirar a cabeça do projeto. Pela paciência e divertimento que tivemos em ouvir-nos uns aos outros a falar dos seus problemas da tese.

Aos meus Pais por sempre me lembrarem da importância dos estudos e estarem sempre presentes com muita paciência nestes 5 anos de universidade. Por fim, ao meu irmão por me fazer companhia divertida nestes últimos semestres passados.

Xavi

"Keep learning.
Enjoy thinking."

InspiroBot[1] - I am an artificial intelligence dedicated to generating unlimited amounts of unique inspirational quotes for endless enrichment of pointless human existence..

[1] https://inspirobot.me/

Contents

1 Introduction **1**
- 1.1 Context and Motivation . . . 1
- 1.2 Problem Description and Objectives . . . 2
- 1.3 Document Structure . . . 3

2 Deep Learning: Fundamentals **5**
- 2.1 Introduction . . . 5
- 2.2 History . . . 6
- 2.3 Core Concepts . . . 7
  - 2.3.1 Neurons and Layers . . . 8
  - 2.3.2 Loss Function . . . 8
  - 2.3.3 Gradient and Backpropagation . . . 9
  - 2.3.4 Activation Function . . . 10
  - 2.3.5 Regularization . . . 11
- 2.4 Convolutional Neural Networks . . . 12
  - 2.4.1 Overview . . . 12
  - 2.4.2 Evolution . . . 14
  - 2.4.3 Object Detection . . . 15
- 2.5 Variational Autoencoders . . . 17
- 2.6 Generative Adversarial Networks . . . 19
- 2.7 Contrastive Learning . . . 23

3 Domain Adaptation: Literature Review **27**
- 3.1 Introduction . . . 27
- 3.2 Notation . . . 28
- 3.3 Taxonomy . . . 29
- 3.4 Divergence-based Methods . . . 30
  - 3.4.1 Divergence Measures . . . 32
- 3.5 Adversarial-based Methods . . . 34
  - 3.5.1 Generative Models . . . 35
  - 3.5.2 Non-generative Models . . . 37
- 3.6 Reconstruction-based Methods . . . 39
  - 3.6.1 Encoder-Decoder Reconstruction . . . 39
  - 3.6.2 Adversarial Reconstruction . . . 41
- 3.7 Domain Randomization . . . 45
- 3.8 Applications and Datasets . . . 46
- 3.9 Summary and Discussion . . . 48

4 Methods **51**
4.1 Previous work: Automatic quality inspection . . . 51
4.1.1 Datasets . . . 52
4.1.2 Object-detection stage . . . 53
4.1.3 CBIR stage . . . 53
4.2 Method overview . . . 54
4.2.1 Goal . . . 54
4.2.2 Solution . . . 55
4.2.3 Challenges . . . 57
4.2.4 System Evaluation . . . 57
4.3 Domain Translation . . . 58
4.3.1 Datasets . . . 58
4.3.2 Differentiable Augmentation . . . 59
4.3.3 CycleGAN . . . 61
4.3.4 CUT . . . 64
4.4 Detector Fine-tuning . . . 64

5 Results and Discussion **67**
5.1 Domain Translation . . . 67
5.1.1 CycleGAN . . . 67
5.1.2 CUT . . . 73
5.2 Detector Fine-tuning . . . 76
5.2.1 Fine-tuning on converted images . . . 76
5.2.2 Global NMS . . . 78
5.2.3 Fine-tuning on real pseudo-labels . . . 80
5.3 Summary . . . 82

6 Conclusion **85**
6.1 Overview . . . 85
6.2 Future Work . . . 87

A Extra experiment plots **89**
A.1 CycleGAN's FID scores . . . 90
A.2 Domain translation comparisons . . . 92
A.3 Detectron2's fine-tuning processes . . . 95
A.3.1 Detectron2's fine-tuning on converted images . . . 95
A.3.2 Detectron2's fine-tuning on real images' pseudo-labels . . . 96
A.4 Detection module real images assessment . . . 96
A.4.1 Fine-tuned Detectron2 using CUT and pseudo-labels . . . 96

References **99**

List of Figures

1.1 DL and traditional ML performance regarding data quantity . . . 2

2.1 Style transfer . . . 6
2.2 AlexNet initial convolutional filters . . . 7
2.3 Neocognitron and AlexNet architectures . . . 8
2.4 Unit and Multi-layer perceptron . . . 9
2.5 Computational graph example . . . 10
2.6 Activation functions . . . 11
2.7 Normalization layers . . . 12
2.8 CNN architecture . . . 13
2.9 Residual block . . . 15
2.10 VAE architecture . . . 18
2.11 GAN diagram . . . 20
2.12 GAN latent-code arithmetic . . . 21
2.13 Contrastive learning in the ML spectrum . . . 23

3.1 Domain-adaptation scenarios . . . 29
3.2 Domain-adaptation taxonomy . . . 30
3.3 CCD metric depiction . . . 34
3.4 Unified adversarial framework . . . 35
3.5 DANN model architecture . . . 38
3.6 DSN model architecture . . . 40
3.7 Cycle-consistency loss . . . 42
3.8 CUT's distribution matching phenomenon . . . 44
3.9 MUNIT reconstruction losses . . . 45
3.10 Dataset image translation . . . 47

4.1 Automatic quality inspection system diagrams . . . 52
4.2 Simulated and real datasets samples . . . 53
4.3 Progressive fine-tune approach . . . 56
4.4 Real dataset crop creation . . . 59
4.5 Overview of DiffAugment . . . 60
4.6 Diagram of the modified CycleGAN with semantic consistency . . . 62

5.1 CycleGAN FID scores for both translation directions . . . 68
5.2 CycleGAN cycle-consistency validation losses for both domains . . . 68
5.3 CycleGAN discriminator validation losses for both domains . . . 69
5.4 FID scores obtained on CycleGAN with semantic consistency . . . 70
5.5 CycleGAN semantic consistency losses on simulated to real translation . . . 71

5.6 Translation comparison between CycleGAN variations 71
5.7 Real to simulated counterparts image examples using CycleGAN 72
5.8 CUT discriminator and generator training losses 73
5.9 CycleGAN vs CUT FID scores with different PatchNCE weights 74
5.10 Comparison between CycleGAN and CUT translations 75
5.11 The effect of using global NMS on detectron2's predictions 79

A.1 FID for all CycleGAN's experiments during simulated to real translation 90
A.2 FID for all CycleGAN's experiments during real to simulated translation 90
A.3 Domain translation comparisons (part 1) 92
A.4 Domain translation comparisons (part 2) 93
A.5 Domain translation comparisons (part 3) 94
A.6 Detectron2's loss during fine-tuning on CycleGAN converted images 95
A.7 Detectron2's AP during fine-tuning on CycleGAN converted images 95
A.8 Detectron2's loss during fine-tuning on CUT converted images 95
A.9 Detectron2's AP during fine-tuning on CUT converted images 96
A.10 Detectron2's loss during fine-tuning on real images' pseudo-labels 96

List of Tables

3.1 Transfer Learning settings 28
3.2 Discussed methods according to [89] framework 35
3.3 Summary of the discussed DA methods 49

4.1 CBIR results on simulated data: object crops from the ground-truth vs output of the detection module [68] 54
4.2 Results on real data, reported as a multi-class multi-label classification problem [68] 55
4.3 Detection and instance segmentation results on simulated data [68] 55
4.4 Number of images for the different splits of both simulated and real datasets . . . 59
4.5 Detectron2's backbone output layers' details 63

5.1 Results on real data for the detection module when converting the input real image into simulated counterparts 72
5.2 Detectron2 preliminary validation results on converted images 76
5.3 Fine-tuned Detectron2 preliminary validation results on simulated and converted images 77
5.4 Results on real data for the fine-tuned Detectron2 with converted images 77
5.5 Results on real data for the detection module using the fine-tuned Detectron2 on converted images 78
5.6 Results on real data for the fine-tuned Detectron2 when applying global NMS . . 79
5.7 Results on real data for the detection module using the new fine-tuned Detectron2 when applying global NMS 80
5.8 Preliminary validation results on simulated and converted images for the fine-tuned Detectron2 using pseudo-labels from real images 81
5.9 Results on real data for the fine-tuned Detectron2 with real images' pseudo-labels 81
5.10 Results on real data for the detection module using the fine-tuned Detectron2 on the converted images and generated pseudo-labels 82
5.11 Results on real data for the detection module using the fine-tuned Detectron2 on the converted images and generated pseudo-labels when applying the global NMS 82

A.1 Results on real data for the fine-tuned Detectron2 with real images' pseudo-labels 96
A.2 Results on real data for the detection module using the fine-tuned Detectron2 on the converted images and generated pseudo-labels 97
A.3 Results on real data for the detection module using the fine-tuned Detectron2 on the converted images and generated pseudo-labels when applying the global NMS 97

Abbreviations

AdaIN	Adaptive Instance Normalization
AdaBN	Adaptive Batch-normalization
ADDA	Adversarial discriminative domain adaptation
AE	Autoencoder
AI	Artificial Intelligence
ANN	Artificial Neural Network
AP	Average Precision
BN	Batch-normalization
CAN	Contrastive Adaptation Network
CBIR	Content-based image retrieval
CCD	Contrastive Domain Discrepancy
CORAL	Correlation Alignment
CGAN	Conditional Generative Adversarial Network
CNN	Convolutional Neural Network
CoGAN	Coupled Generative Adversarial Network
CyCADA	Cycle-consistent adversarial domain adaptation
CUT	Contrastive Unpaired Translation
CV	Computer Vision
DA	Domain Adaptation
DAE	Denoising Autoencoder
DANN	Domain-adversarial neural networks
DCGAN	Deep Convolutional Generative Adversarial Network
DCRN	Deep Reconstruction-Classification Network
DiffAugment	Differentiable Augmentation
DL	Deep Learning
DNN	Deep Neural Network
DSN	Domain Separation Networks
EM	Earth Mover
FID	Fréchet Inception Distance
FPN	Feature Pyramid Network
GAN	Generative Adversarial Network
GD	Gradient Descent
GPU	Graphical Processing Unit
GRL	Gradient Reversal Layer
I2I	Image-to-image
IoU	Intersection over Union
IS	Inception Score
JS	Jensen-Shannon

KL	Kullback–Leibler
mAP	Mean Average Precision
ML	Machine Learning
MLP	Multi-layer Perceptron
MMD	Maximum Mean Discrepancy
MNIST	Modified National Institute of Standards and Technology
MUNIT	Multimodal Unsupervised Image-to-Image Translation
NLP	Natural Language Processing
NMS	Non-maximum suppression
NT-Xent	Normalized temperature-scaled cross entropy
RCAN	Randomized to-Canonical Adaptation Network
ReLU	Rectified Linear Unit
RKHS	Reproduced Kernel Hilbert Space
RPN	Region Proposal Network
RoI	Region of Interest
SKU	Stock-keeping Unit
SVHN	Street View House Numbers
SYNTHIA	SYNTHetic collection of Imagery and Annotations
UNIT	Unsupervised Image-to-Image Translation Network
USPS	U.S. Postal Service
VAE	Variational Autoencoder
Tanh	Hyperbolic Tangent
TL	Transfer Learning
WGAN	Wasserstein Generative Adversarial Network
YOLO	You Only Look Once

Chapter 1

Introduction

1.1 Context and Motivation

Traditional Machine Learning (ML) models are not as effective at solving broader and more complex problems, such as image segmentation or object detection, as other approaches. In recent years Deep Learning (DL), a sub-field of ML, has emerged, using Artificial Neural Networks (ANNs) with a high layer depth to better approach these challenges. DL models have become ubiquitous for their considerable performance improvements compared to other methods [2]. Their ability to tackle a wide variety of tasks has been realized, with significant advances in multiple research areas, especially in the field of Computer Vision (CV) where Convolutional Neural Networks (CNNs) are a staple to image understanding.

One of the main particularities that make DL methods so distinguishable from others is the need for enormous amounts of data to successfully build a well-performing model. Traditional ML models seem to hit a performance plateau when a certain dataset size is reached, while having better results when dealing with smaller sized datasets. However, this performance threshold is higher for DL models and much harder to reach [2]. In fact, the more data DL models are given, the better their performance improvement, as shown in figure 1.1.

Therefore, to fully make use of DL models' superior data understanding capabilities, large amounts of data are required. Unfortunately, gathering and labelling the necessary data is costly and extremely labour-intensive since the annotations must be done manually by humans. Furthermore, in some scenarios, it can be nearly impossible to annotate the data. For example, considering the eye gaze estimation problem [80], where given an image of a person, we want to determine the direction that person is looking at. In this situation, collecting the eye direction information for each individual image is remarkably difficult. Another possible area where data collection is cumbersome is healthcare [7]. When evaluating images produced by medical equipment, for example, magnetic resonance imaging scanners, a person is required and must be subjected to the medical process and the subsequent evaluation, by a medical expert, of the produced image. This not only imposes equipment constraints but also takes a considerable amount of time for the data to be manually annotated.

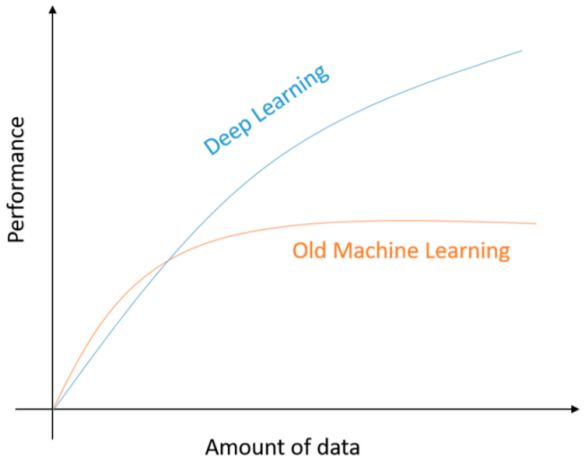


Figure 1.1: Comparative performance of DL and traditional ML models with respect to the amount of data. Extracted from [2]

To surpass the difficulties associated with data collection, synthetic images produced by simulators can be a cost-effective and reliable alternative [80, 87, 5, 6, 36]. By making use of simulators, we would be able to automatically generate annotated data, on-demand, bypassing all the inherent obstacles when trying to gather real-world information. Going back to the previously given example of eye gaze estimation, measuring the eye direction in a simulated environment, with a simulated model of a person's head, is substantially easier than trying to make such measurement in real life.

By utilizing simulated and real images jointly, we can give DL models the necessary amount of data to achieve their maximum potential. Nonetheless, image representation and distribution are distinct since we are consolidating information from different sources. When trying to merge information from two different domains, with the same final task, we are within the field of DA, a branch of TL [94, 97].

1.2 Problem Description and Objectives

This work aims to address the domain shift problem that emerges when jointly utilizing synthetic and real images to train a DL model. Domain shift occurs when there are discrepancies, whether in feature space or in images' distribution between the conflicting domains, thus requiring additional steps to diminish the domain differences. This is discussed in more detail in section 3.3. We intend to survey possible domain adaptation techniques and apply them to this context.

The objectives and possible advantages we expect to achieve are the following:

- **Boost performance of DL models:** leveraging different, but with useful information, data distributions to broaden our training set and effectively augment models' learning abilities;

- **Develop a more scalable method:** obtaining data is no longer a project's liability;
- **Reduce the effort of collecting labeled data:** simulated data is already annotated;
- **Apply DL to new areas:** time and budget constraints are mitigated since there is no need to manually gather annotated data.

Stemming from a previous project, aimed to automate quality control at the end of a vehicle's production line, a proof-of-concept will be developed. Previously, a DL model that identified vehicle parts was trained using only data obtained through a simulator. By making use of DA techniques to combine simulated and real images, a new model will be trained to be applied to the real-world more effectively. The model's performance, using both types of data, will be compared to its performance when using exclusively one of the two types.

1.3 Document Structure

The remainder of this document is structured into 4 chapters as follows:

- **Deep Learning: Fundamentals** (chapter 2) explains the core concepts of DL models, a brief history of their evolution and some prominent model frameworks that are broadly used to perform DA.
- **Domain Adaptation: Literature Review** (chapter 3) introduces DA as a field of TL, an overall taxonomy of DA scenarios and addresses the different approaches made and their differences.
- **Methods** (chapter 4) describes the previous work [68] on which this thesis's proof-of-concept is based on, providing an overview of the proposed improvements and associated challenges. Furthermore, the executed experiments are detailed.
- **Results and Discussion** (chapter 5) analyzes the obtained experiments' results.
- **Conclusions and future work** (chapter 6) reflects upon and concludes this work with possible future paths this project can take.

Chapter 2

Deep Learning: Fundamentals

This chapter provides an insight into the fundamental blocks present in any DL model. First, the distinction between traditional ML and DL is addressed and a brief history of how ANNs came to existence is shown. Following that, we focus on the core concepts that make up the foundation of DL, which enable us to train these massive high layer count ANNs. Afterwards, CNNs are introduced as the prevalent models used for image understanding alongside their evolution over the years. Finally, Variational Autoencoders (VAEs) and GANs are taken into consideration since they are widely used in the subject of DA.

2.1 Introduction

DL is a subset of ML. DL models are distinguishable from the traditional ML models, like Random Forests or Support Vector Machines, mostly for their feature learning capabilities, allowing them to achieve an overwhelming performance gap when dealing with complex problems. Feature learning can be thought of as feature engineering integrated into the model itself.

DL is based on ANNs with multiple and distinct building blocks, developed over the years, to better grasp the input data it is given. It is referred to as "deep" learning because these models are built with a high layer count and have been getting increasingly bigger with the expectation of enhancing their performance even further.

One other discernible characteristic of these models is the computational power necessary to train them. As DL models get bigger in layer count, so do the learnable parameters that make up those layers. Even though layer and parameter count are not strictly correlated, especially since correct usage of model architectures and specific building blocks can help diminish parameter count without sacrificing accuracy, this is what tends to occur. However, over the years, hardware constraints have become less of an issue. By taking advantage of Graphical Processing Units (GPUs) we are able to train these models even faster, allowing us to experiment with ever-increasingly deeper networks.

Figure 2.1: Style transfer of Van Gogh's Starry Night into a given image. Extracted from [16].

One of the main areas affected by the appearance of DL is Computer Vision (CV), to the point where tasks like image classification, semantic segmentation and object detection can be solved with considerable less effort and more reliably than with former standard methods. In fact, DL has become the new standard when dealing with any sort of image understanding task. One possible DL application is image style transfer, where the main goal is to produce a new image by merging one image's content with the style of another. This particular task is depicted in figure 2.1.

2.2 History

In 1943, neurophysiologist Warren McCulloch and mathematician Walter Pitts released their paper "A logical calculus of the ideas immanent in nervous activity" [51], in which they presented a simplified model of how biological neurons might work together in animal brains to perform complex computations [18].

Frank Rosenblatt, in 1958, introduced the Perceptron [71] as one of the first algorithms that could learn from data. It was implemented as a piece of hardware and learned to recognise letters of the alphabet. Today, it can be seen as a Linear Classifier or as a single layer of an ANN.

Another contributing piece of research was Hubel and Wiesel's study of how cats perceive images [32]. They inserted an electrode into the cat's visual cortex, an area in the back of the brain, and recorded neuron activity. The cat was shown multiple images, and by analyzing its neurons' responses, they noticed that neurons were reacting to oriented edges, while the images were being drawn on the projector. They devised a hierarchical representation of the visual system, composed of different cell types, ranging from simple to complex cells, responding to different visual stimuli. Later on, in 1981, the authors were awarded the Nobel Prize for their research regarding information processing in the visual system.

Even though Hubel and Wiesel's research was not strictly connected to ANNs or DL, they made a fascinating and curious discovery. It is curious since today's ANNs, with their multiple layers, seem to show signs of learning in a similar fashion. When dealing with image understanding, the first layers of a CNN usually capture some information regarding edges present in the image. In figure 2.2 we can see some visual representations of the first layers of a CNN.

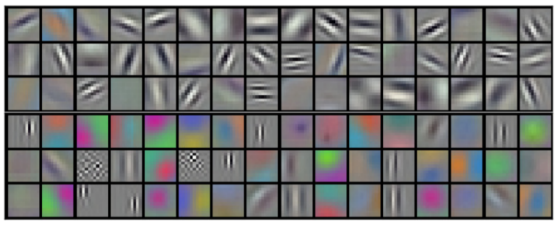


Figure 2.2: First layer convolution filters of an AlexNet [42]. These filters learned to react to oriented edges and opposing colors.

Coming back to computer science territory, in 1969, Marvin Minsky and Seymor Papert, with their book called "Perceptrons" [53], demonstrated that Perceptrons are incapable of solving trivial problems. They were incapable of replicating the XOR function using a single-layered Perceptron. This book became very famous, and this limitation caused much disillusionment in the field of ML. However, one fact that was overlooked was that the book also mentioned a Multi-Layered Perceptron (MLP), which could represent different kinds of functions, including the XOR function. Sometimes ANNs are referred to as MLPs in homage to these earlier models.

In 1980, Fukushima created the Neocognitron [12], and managed to programmatically manifest Hubel and Wiesel's ideas, which enabled the model to compute the behaviour of simple and complex cells. With recent terminology, we could see these operations as convolution and pooling operations, respectively. Despite having the right concepts, which would manifest and become the core blocks of today's DL models, there was no practical training algorithm to effectively train the model. The resemblance between the Neocognitron's and AlexNet's architectures is depicted in figure 2.3.

This changed in 1985, with the revolutionary work [72] of Rumelhart, Hinton and Williams. The backpropagation algorithm for computing gradients in ANNs was introduced, allowing MLPs to be trained successfully. This is the core algorithm used to train any DL model currently.

There were some periods of time when not many breakthroughs were made in the field of artificial intelligence (AI). This was caused either due to newly discovered limitations or false promises of future advances. These periods are referred to as "AI Winters", where progress stagnated, and funding for new projects was extremely low. Despite such setbacks, researchers kept going, and their efforts manifested as DL is, nowadays, a well established and active field of research.

2.3 Core Concepts

As previously mentioned, a current ANN is comparable to a MLP, being composed of multiple layers of Perceptrons stacked on top of each other. The simplest ANN comprises three layers, an input layer, a hidden layer and an output layer. The number of hidden layers and their respective size is arbitrary. An ANN is referred to as a Deep Neural Network (DNN) if it has multiple hidden

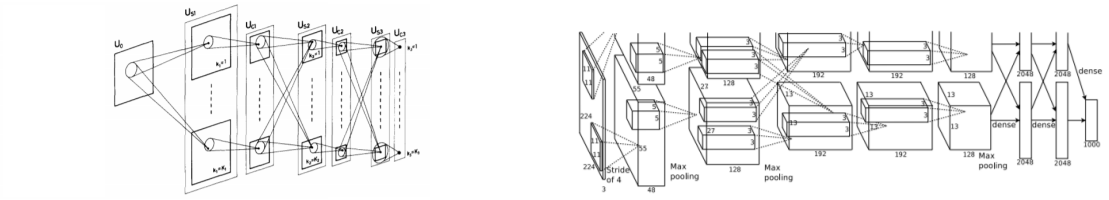


Figure 2.3: On the left we can see Fukushima's model arquitecture [12]. On the right shows the more recent AlexNet CNN arquitecture [42].

layers. In the remaining part of this section some core concepts are explained, from the definition of neurons and layers to the backpropagation algorithm used to train these networks.

2.3.1 Neurons and Layers

Each layer is composed of multiple neurons[1]. Each unit has a set of learnable weights and a bias term. With these parameters it calculates a weighted sum over its inputs and applies a non-linear function to the final output. For a fully connected ANN, each unit of a layer is connected to every unit of the following layer. With the repetition of these seemingly simple lower-level computations, the artificial neural network learns, at each layer, multiple feature representations. These are combined at the final layer to create a prediction. This can be seen at figure 2.4.

2.3.2 Loss Function

The loss function indicates a network's performance while training, and more importantly, how to improve such a network by changing the learnable parameters in a way that results in a lower loss value. Depending on the task, some loss functions are better suited than others, focusing on different aspects we want to manifest in our network. Broadly speaking, a loss function takes as input the real output of a model and the desired output the model should aim for. By comparing the two values, we learn how to adjust the network to produce better results. For example, a common loss used for regression problems is the least squared error loss, while in the context of DL, for classification tasks, the cross-entropy loss is commonly used, and it can be defined as follows:

$$H(y,\hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log \hat{y}_i) \tag{2.1}$$

where N is the number of instances in y and $\hat{y}$, y_i contains the target class labels for instance i and $\hat{y}_i$ are the classifier's predictions for instance i.

[1]The term "neuron" is often discouraged since our understanding of biological neurons has changed since the term was first introduced. The term "unit" seems to be more scientifically adequate.

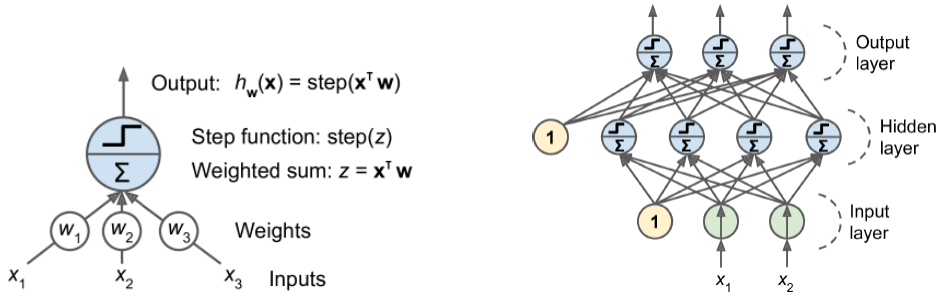


Figure 2.4: On the left we can see a single unit. On the right shows a simple MLP with two inputs, one hidden layer of four units and three output units, along with a bias term. Extracted from [18].

2.3.3 Gradient and Backpropagation

To produce its results, a DNN takes its input and propagates them throughout the following layers, applying numerous transformations until the final layer is reached, giving the desired output. The loss is then calculated, using a specific loss function. This is a normal forward pass that is repeated multiple times during the training process. With the information given by the loss function, several adjustments are made to the network's parameters to reduce such loss.

To explain how to train DNNs, we must first introduce the concept of Gradient Descent (GD). A loss function can be seen as a high-dimensional landscape, and our goal is to find the global minimum of that landscape. Theoretically, the gradient of the loss function with respect to the networks' parameters tells us how to alter these values in a way that increases the loss function's value. Intuitively, the negative of that gradient, tells us how to decrease it. So, by calculating the inverse of the loss's gradient we are descending that high-dimensional landscape, and hopefully moving towards the direction of that global minimum[2]. This process is called Gradient Descent.

The backpropagation algorithm [72] is a programmatical way of computing the gradient. A computational graph, depicted in figure 2.5, is created during the forward pass of the network, composed of all the executed operations. Then, by making use of the chain rule (see equation 2.2), we traverse the created graph in reverse order and calculate the local gradient at each node. The name backpropagation comes from this notion of reversely traversing the computational graph. When all the gradients are computed for all parameters, a so-called gradient step (see equation 2.3) is made, updating the network's parameters. This process is repeated several times over the course of training.

The chain rule and gradient step operations can be seen in equations 2.2 and 2.3, respectively, with α as the network's learning rate (how far to descent in the landscape), k has the current training iteration and $\frac{\partial L}{\partial w}$ as the gradient of the loss with respect to the network's weights w.

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} \tag{2.2}$$

[2]There is no guarantee that the global minimum is reached since the loss functions used by DNNs are not convex.

$$w_k = w_{k-1} - \alpha \frac{\partial L}{\partial w} \tag{2.3}$$

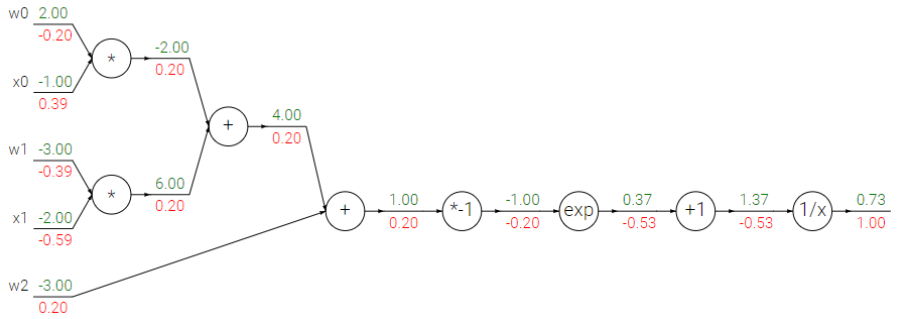


Figure 2.5: A simple computational graph, with forward values in green and backwards gradients in red. Image retrieved from [92].

2.3.4 Activation Function

The essence of an ANN is the presence of non-linearities. These allow the network to manipulate the input feature space, in a non-linear way to more effectively distinguish between classes, for example in a standard classification task. Programmatically, the operations from layer to layer, composed of multiple unit calculations, are matrix multiplications. Stacked matrix multiplications can be represented, and have the same effect, as a single matrix multiplication. If such non-linear transformations were not in place, a normal ANN would have the same generalization power as a Linear Classifier.

An activation function is what applies these non-linear operations on the weighted sum that the unit produces. There is a wide variety of activation functions, each one having its advantages and disadvantages. In the early years of DL, the Sigmoid and Hyperbolic Tangent (Tanh) functions were commonly used in-between hidden layers of networks. These are still found in current DNNs, albeit less frequently due to causing vanishing gradients when the values are farther away from zero. This happens since both of them are saturating functions. Intuitively speaking, this means that they squeeze its inputs to some compact interval, for example, the Sigmoid function has an output range of $[0, 1]$.

Nowadays, the Rectified Linear Unit (ReLU) is the clear choice since it mitigates some drawbacks of the previous functions, being more computationally efficient and having constant gradients. ReLU allows the network to rapidly converge with no noticeable performance loss. It is not perfect though, and other variants, like Leaky ReLU, allow a small, positive gradient when the input is a negative value, while the normal ReLU does not. These functions can be seen in figure 2.6.

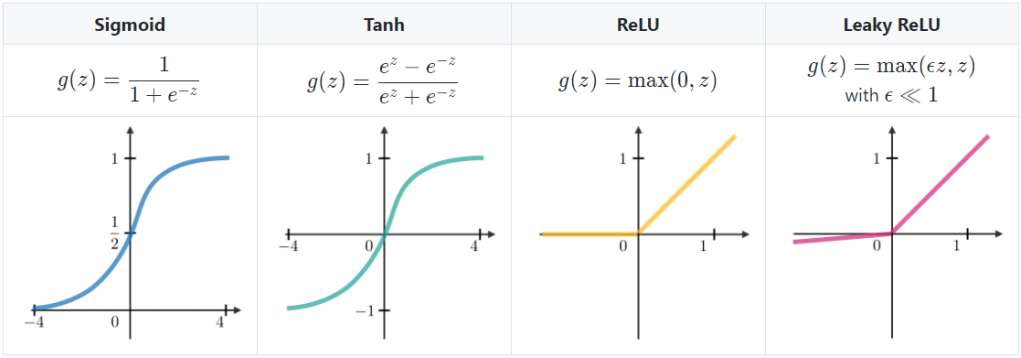


Figure 2.6: Popular activation functions used in DL. Image retrieved from [3].

Another important non-linearity is the Softmax function. This is commonly used in the output layer of a network and is essential in classification tasks. It converts the class scores into a normalized probability distribution, allowing us to choose the most probable class.

2.3.5 Regularization

As the name implies, regularization is implemented to regularise the network's training process. As the number of layers inside DNNs grows larger, we expect to see better results since more non-linear operations lead to a feature-space where classes are more distinguishable. The problem that emerges is one that every ML model faces when it starts to memorize the data instead of learning from it, and that recurring problem is overfitting. The main goal of regularization is to make the training of large networks possible without memorizing the training data.

One way this can be implemented is by changing the loss function. Specifically, L1 and L2 norms are the most common types of regularization used. These penalties constrain the size of the network's weights, preventing them from being too high so that a single unit can significantly affect the output. Instead, these norms encourage the distribution of weights throughout the units, meaning that the output will be a unanimous decision between them. The use of these norms helps reduce the network's bias.

We can regularize the network by implementing the regularization logic into the model's architecture itself. This is done through a batch normalization layer, first introduced in 2015 by Sergey Ioffe and Christian Szegedy [34]. Briefly, this layer normalizes each batch of data by subtracting the mean and dividing by the variance of the batch. This eliminates the need for the following layer to readjust to a new distribution in each iteration. Later versions of layer normalization appeared, such as instance normalization, layer normalization and group normalization. These can be seen in figure 2.7.

Another way to regularize the network is to randomly remove units from the networks' layers during each iteration of the training process. This method is called dropout, and has a similar effect to L1 and L2 norms. It prevents the network from relying on important units for its decision process by literally removing them.

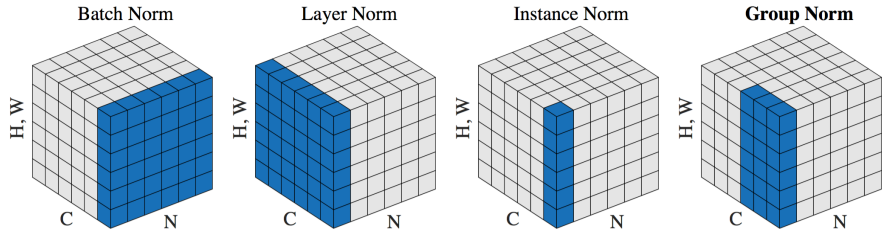


Figure 2.7: Different kinds of layer normalization, applied to distinct sections of data. Image retrieved from [1].

2.4 Convolutional Neural Networks

2.4.1 Overview

Up to this point, we have discussed regular feed-forward, fully connected neural networks. However, despite their usefulness, they do not maintain images' structure since they have a one-dimensional vector as input. For the task of image understanding, it would be helpful to operate on images without destroying spatial information.

In 1998, LeCun et al. [46] presented LeNet, named after one of the authors, Yann LeCun. This was the paper that established the supremacy of CNNs over all other methods. LeNet is a CNN, trained using the backpropagation algorithm, that was able to recognize handwritten digits. It was so effective at this task, that is was commercially deployed to process handwriting in checks.

LeNet has the fundamental blocks that make up today's CNNs: Convolutional and Pooling Layers. These will be discussed in the following sections. The first layers of a typical CNN are a combination of convolution and pooling operations, acting as a feature extraction section. The last layers generally are fully connected and can be seen as the classification section of the model. A standard overall architecture of a CNN can be seen in figure 2.8.

2.4.1.1 Convolution Layer

Normal fully connected layers have a matrix of learnable parameters, composed of a set of weights and one bias term per unit. A convolutional layer's parameters consist of a set of learnable filters, also called kernels. The depth hyperparameter controls the number of filters to be used. These filters are, more often than not, square and are spatially small compared to the input image. Each filter extends through the full depth of the input image. During the forward pass, we slide (more precisely, convolve) each filter across the input image's width and height and compute dot products between the entries of the filter and the input at any position. As we slide the filter we will produce a feature map, which can also be referred to as an activation map, that gives the responses of that filter at every spatial position[3]. As there are multiple filters, we will produce a feature map for each filter. These stacked feature maps will form the input for the next layer. The size of the next

[3]Adapted from https://cs231n.github.io/convolutional-networks/

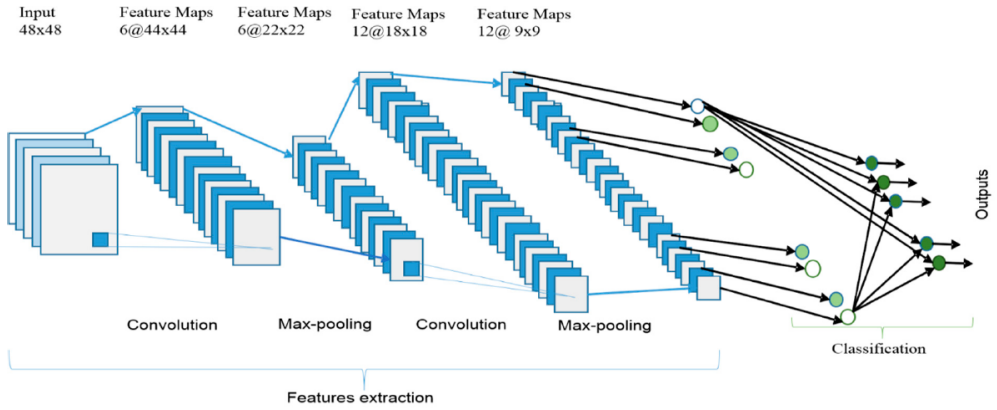


Figure 2.8: Standard CNN architecture with convolution, max-pooling and fully connected layers. Extracted from [2].

layer N depending on the filter size F and the input size W, using padding P and stride S, can be formulated as the following:

$$N = \frac{W - F + 2P}{S} + 1 \tag{2.4}$$

Each value in the produced feature maps was computed as an operation between a filter and a spatial section of the input. This spatial section is called the receptive field and evidently has the same size as the used filter. As we progress through the CNN's layers, the receptive field increases in size with every convolution operation. At the deeper layers, a single value may contain information regarding the entire input image. With this understanding, we can deduce that generic features are being extracted at shallow layers. In contrast, domain specific features, contemplating a broader section of the input image, can be obtained at deeper sections of the network.

Besides filter size and depth, there are other hyperparameters associated with these layers – stride and padding influence how the convolution operation is executed and affect the feature map's resulting size. Stride value indicates the distance between consecutive receptive fields, with a higher stride producing smaller feature maps. Padding indicates the border size around the input. Zero-padding is very commonly used, filling the border around the input with zeros.

2.4.1.2 Pooling Layer

It is common to periodically insert a pooling or sub-sampling layer in-between successive convolutional layers in a CNN architecture. It is used to reduce the input maps' spatial size, resulting in fewer parameters and computation, thus controlling overfitting. It does not have any learnable parameters, and it will not alter the input's number of features maps. For example, if there are N input maps, there will be exactly N output maps [2]. Hence, it only has size and stride as hyperparameters.

The max-pooling layer, used in the LeNet architecture, operates independently on every depth slice of the input and resizes it spatially, using the max operation. Instead of the max function, other operations can be used, for example, average-pooling outputs an average over each receptive field.

Sometimes, pooling layers can even serve as a substitute to some of the last fully connected layers of a network. GoogLeNet [84], referred in the next section 2.4.2, uses global average pooling to achieve just this.

2.4.2 Evolution

Multiple CNN architectures have risen over the years. One major incentive was the ImageNet [73] challenge. ImageNet is a very popular classification dataset, and from 2010 to 2017 an annual competition was in place for several visual recognition tasks using this dataset. This challenge gave rise to novel DL architecture blocks which are still present in most recent DNNs.

In 2012, AlexNet [42] won the ImageNet challenge, marking an important milestone in CV by turning CNNs into a mainstream topic. The massive decrease in error rate, from 25.8% of the previous winner to AlexNet's 16.4%, showed how powerful CNNs could be when dealing with image understanding tasks. It was similar in architecture to LeNet [46], just with more layers and filters. The authors used dropout and data augmentation to improve the models' performance and trained the model for six days on two GPUs. The next year's winner was ZFNet [105], with an 11.7% error rate. This model was a more refined and deeper AlexNet without many breakthroughs.

The year 2014 brought two significant architectures, the VGGNet [81], and GoogLeNet [84] architectures, with the latter winning the ImageNet challenge that year. VGGNet, named after the Visual Geometry Group where it was designed, moved to a more uniform architecture, having some design rules one could follow to scale the network without having to set each layer experimentally. This model had around 140 millions parameters, compared to AlexNet's 61 million, and required an incredible amount of computational power to train. Therefore, to converge the network, the authors trained a shallower VGGNet, with fewer layers, and then inserted newer layers into the already trained ones, resuming training afterwards.

The other prominent architecture, GoogLeNet [84], named in homage to LeNet, focused on efficiency, reducing the number of learnable parameters to just 4 million. This network's notable contributions were introducing a stem network, novel inception modules, auxiliary classifiers, and the removal of most of the last fully connected layers. The stem network used in the very first convolutional layers drastically downsamples the input's spatial resolution to reduce the amount of computation needed for the following layers. The network was composed of consecutive inception modules, which allowed parallel computation and were composed of layers with different kernel sizes. Another difference was the usage of global average pooling, to collapse spatial information, and further reduce parameter count. A single fully connected layer produced the final scores. Lastly, auxiliary classifiers were implemented at several intermediate points to propagate gradients throughout the network more effectively. The need for these classifiers was due to batch

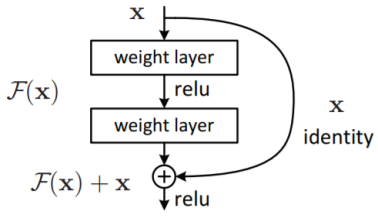


Figure 2.9: Residual block from ResNet [26].

normalization [34] not yet having been proposed, and such techniques were needed to converge deeper networks.

Another major design change was introduced by ResNet [26], in 2015, with residual blocks, shown in figure 2.9. These blocks have skip connections that improve gradient flow and allow network blocks to learn the identity matrix more easily, enabling deeper networks to emulate shallower networks. ResNet had a considerable increase in layers, jumping 152 layers from GoogLeNet's 22 layers. It also made use of an initial stem network, global average pooling and took much advantage of batch normalization [34], already proposed at that time.

These were the most notable architectures that originated from the ImageNet challenge. However, novel models are still being proposed with interesting ideas, one of such is the EfficientNet [86]. The authors argue that a CNN can be scaled in 3 ways: depth (number of layers), width (number of feature maps) or resolution (input resolution), and rethink how to effectively scale CNNs. [86] shows that scaling modes are connected and that if one is increased, another should be increased as well to produce better results. To account for this, the authors propose a novel compound scaling method, that balances these scaling modes, by scaling them according to a constant ratio. A new family of networks using this method is created, called the EfficientNets, that outperform state-of-the-art CNNs with substantially fewer parameters and computation.

2.4.3 Object Detection

Since this work's proof-of-concept ultimate task is to detect vehicle manufacturing parts, this section will briefly explain the task of object detection in images, one of the many applications of DL models.

Object detection is a multi-task problem where the goal is not only to label each object individually but also to ascertain their position within the image by means of a bounding box. Each object's bounding box varies in position, width, and height depending on the detected object. Since a single image can contain several objects to detect, this task requires more data to train effectively, more extended training periods, and the input images cannot be as substantially downsampled as in classification tasks, since this risks failing to detect small objects within the image.

Object detectors can be separated into two classes: two-stage and single-stage. Two-stage detectors first predict regions of interest where objects might be located and then perform their respective classification. In contrast, single-stage detectors try to address these two problems

simultaneously. The latter form predictions faster than the former and can process more images per second, albeit at the cost of accuracy. To explain these two approaches, two models are used as references, Faster R-CNN [66] and You Only Look One 3rd iteration (YOLOv3) [65].

Faster R-CNN is a two-stage detector, and its name derives from its region-based approach. This model had two previous iterations: R-CNN [21] and Fast R-CNN [20]. R-CNN was one of the first detector models based on DL. It calculates regions of interest (RoIs), using a heuristic method called Selective Search [91], where objects are likely to be found. Afterward, each region is reshaped and passed through a CNN that predicts a classification label and a bounding box transformation over the region proposal given as input. Making the output transformation depend on the original region size makes it so that re-scaling the image for the classification section does not affect the final output.

Passing each region proposal through a CNN is computationally inefficient. Fast R-CNN addresses this issue by first passing the input image through a single backbone network that serves as a feature extractor. The same Selective Search algorithm was applied to the original images producing multiple RoIs. These RoIs were projected onto the feature maps, obtained by the backbone network, forming their respective region features. This is possible since the feature maps can be traced back to their respective receptive fields in the original image. The calculated region features are in turn passed through a lightweight CNN predicting classification labels and bounding box transformations.

Both R-CNN and Fast R-CNN make use of the same RoI algorithm, which constitutes most of the model's processing time. Faster R-CNN drops this algorithm and instead incorporates the region proposal step into the network itself by means of a region proposal network (RPN). This network places different anchor boxes of varying sizes in the feature maps obtained by the backbone network and classifies them as containing or not an object. For the positive cases, the model performs as previous iterations. A lightweight CNN predicts a label and a bounding box transformation on each region proposal. With this approach, the network has a total of four losses: two for the RPN (classification loss for each region plus a regression loss for each region transformation) and two for the final lightweight CNN (object classification loss plus a regression loss for each bounding box transformation).

YOLO is a single-stage detector and implements the concept of making predictions on a grid. This means that the input image is divided into equally sized cells, each one responsible for detecting objects present in them. An object is considered to belong to a certain cell if its bounding box center is within that cell. For each cell, different sized anchor boxes are placed with several parameters being predicted for each one:

- The likelihood of that bounding box containing an object
- 4 bounding box parameters: top left corner position (x, y), width and height
- For each class, the likelihood that the object belongs to that class

Ultimately, this network will provide many outputs for each anchor box in each cell. Assume the existence of C class labels, an input image is divided into an N by N cell grid, with each cell containing B anchor boxes and each anchor box characterized by the parameters described above. So, we can expect the number of output values to be:

$$N * N * B * (5 + C) \tag{2.5}$$

Since many outputs are produced for every cell, most do not contain any objects or correspond to the same object instance. To eliminate these duplicate predictions, non-maximum suppression (NMS) is used. This process eliminates similar bounding boxes that belong to the same class, leaving the ones which have a higher confidence in their prediction. Bounding boxes are deemed similar if their intersection over union (IoU) is above a certain threshold. This metric is explained in the next section 2.4.3.1. NMS is also used for the RPN inside Faster R-CNN.

Another DL task that can be calculated in conjunction with object detection is instance segmentation. Standard semantic segmentation divides an entire image in sections, predicting for each pixel a particular label. In contrast, instance segmentation consists of segmenting each object instance present within an image, with unique labels for each distinct instance. One example of this is Mask R-CNN [25], which adds a mask prediction branch to Faster R-CNN. This added branch performs standard semantic segmentation for each detected object, effectively performing instance segmentation for the entire image.

2.4.3.1 Object Detection Evaluation Metrics

To assess the accuracy of bounding box predictions, IoU is used. It ranges from $[0,1]$ and is calculated as the ratio between the area of overlap and the area of union of the ground-truth and the predicted bounding boxes. This metric cannot be used for training since it is non-differentiable, it is only used for evaluation. IoU can be a somewhat strict metric since values near 1 are hard to obtain. As a reference, a score of 0.5 is considered decent, 0.7 is very good and 0.9 is near perfect.

A metric that depicts a model's overall performance is the Mean Average Precision (mAP) score. It uses another metric called Average Precision (AP) which is calculated as the area under the precision-recall curve. AP is averaged over all classes and across multiple IoU thresholds to obtain a mAP score. AP and mAP nomenclatures are often interchanged in the literature and different datasets have different IoU thresholds to calculate these metrics.

2.5 Variational Autoencoders

Before we introduce Variational Autoencoders (VAEs) [41], we must first mention normal Autoencoders (AEs). These are applied in unsupervised settings, requiring no labels from the input data to operate. We can think of them as high-level model architectures since any network can be used for the different components that compose them.

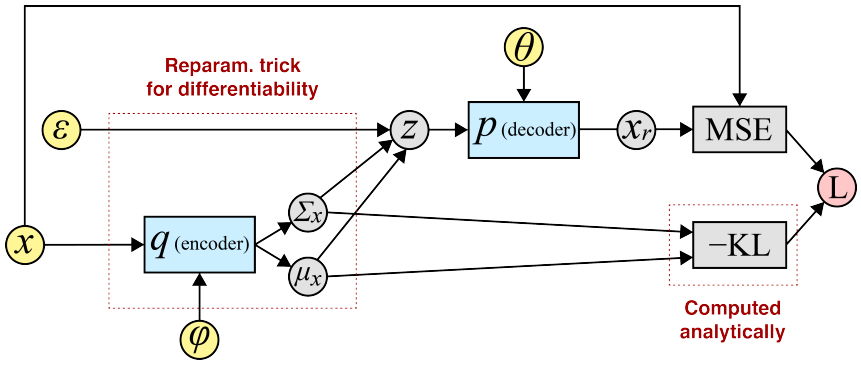


Figure 2.10: VAE architecture with the input or root nodes in yellow. Image retrieved from [24].

AEs are a type of architecture with the goal of compressing image features into a lower-dimension latent-space representation. They are composed of two sections. First, the encoder section compresses the input image producing the latent-space code, commonly denominated z. Afterwards, a decoder part tries to reconstruct the original image from the produced code.

AEs can be used for dimensionality reduction. After training, the decoder section is discarded, and the encoder serves as a way to extract high-level features from the data. This section can be later complemented, for example, with another network serving as a classifier. Although being an interesting concept, AEs are not commonly used in real-worlds applications since, in practice, they do not perform better than their alternatives.

VAEs [41] were first introduced in 2013 by Diederik P. Kingma and Max Welling. These are different from normal AEs as they are probabilistic, even after training. Most importantly, contrary to the specific goal of finding a latent feature representation only to then reconstruct the original image, VAEs are generative models. This means novel images can be created, which are non-existent in the training set but rather sampled from it. A visual representation of a VAE can be seen in figure 2.10.

In VAEs, the encoder, receiving images as input, instead of producing a latent-space code, will produce a probability distribution over the latent-space, represented as a vector of means and a vector of standard deviations. Comparatively, the decoder will receive concrete latent-code representations, sampled from the probability distribution created by the encoder, and try to reconstruct the input image. When the training is complete, we can feed a random z vector to the decoder network, and it will generate novel images.

In VAEs, a problem arises during training, since sampling is a random operation, we cannot backpropagate through it. In the original paper, the authors implemented what they call the reparameterization trick [41] to overcome this setback.

As for that actual loss function used to train these models, the first part is a reconstruction loss, for example mean squared error loss that compares the input image with the final reconstructed one. The second part says that the encoder network should produce compact latent-code representations by roughly following a unit Gaussian distribution. This is achieved by forcing the

latent-space distribution to roughly follow a target distribution, and expressed mathematically as the Kullback–Leibler (KL) divergence between the target distribution (normally a unit Gaussian distribution) and the latent-space distribution. The complete loss function, extracted from [2], with $p(z)$ as the chosen target distribution for z, the encoder as $q_\theta(z|x)$, decoder as $p_\phi(x|z)$, being θ and ϕ as their respective parameters, is as follows:

$$L(\theta,\phi) = -\mathbb{E}_{z\sim q_\theta(z|x)}[log p_\phi(x|z)] + D_{KL}(q_\theta(z|x)\,|\,p(z)) \tag{2.6}$$

In a way, the two terms are fighting against each other. The KL divergence causes the latent distribution to be as simple as possible, by forcing it to be close to the simple prior target distribution. Simultaneously, the reconstruction term encourages the latent-code to contain sufficient information to reconstruct the input data.

Another interesting notion is that of denoising autoencoders (DAEs) [93]. The concept behind these models is that high-level features, which the encoder extracts into a lower dimension latent space z, should be robust enough to a certain level of input distortion. To encourage this, the AE receives a noisy version of the input images and its reconstruction is compared to the original input images. This training process instigates useful features extraction. Furthermore, DAEs can be composed of multiple hidden layers, iteratively reducing their size until the desired latent representation is reached. These are called stacked denoising autoencoders. This naming scheme applies to AEs as well, although most AEs have multiple hidden layers, often omitting this nomenclature.

2.6 Generative Adversarial Networks

Generative Adversarial Networks (GANs) [23] were introduced by Goodfellow et al. in 2014, as a new generative model framework. These types of models have become so popular that they are ubiquitous for generating realistic images and have shown enormous success in different areas, including DA.

The GAN framework is an unsupervised deep learning approach where two neural networks, a generator, G and a discriminator, D, compete against each other in a zero-sum non-cooperative game [2]. While receiving random noise as input, the generator generates fake images that then serve as input to the discriminator. The discriminator takes a batch of real and fake images and tries to classify them accordingly. The discriminator's goal is to correctly evaluate the images' origin, while the generator tries to fool the discriminator into believing its images are real. One important aspect of GANs is that both networks are jointly trained by backpropagation, this means that the generator makes a gradient step using gradients originating from the discriminator network. This method can be seen in figure 2.11.

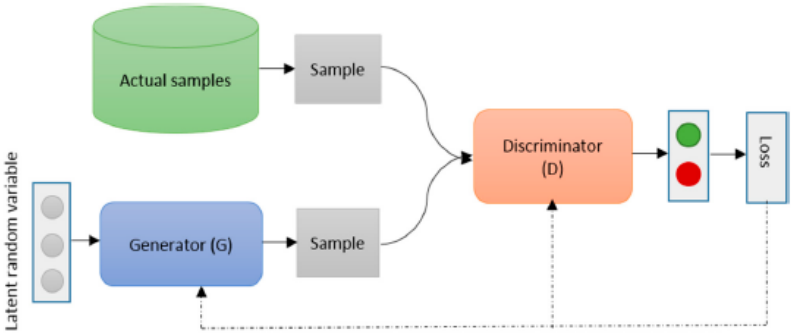


Figure 2.11: Conceptual diagram of a GAN. Extracted from [2].

The base GAN framework is quite intuitive to understand, and we can easily see how this competition among networks can provide us with the results we expect. Although producing impressive results, training these networks in practice is a challenging task. Since GANs are fundamentally trained in a minimax game, they converge when the discriminator and the generator reach a Nash equilibrium [57]. This means that, for each player, even if the opponent would change his next action, a player would not have anything to gain by changing theirs. This is the optimal point for the following minimax equation, obtained from the original paper [23]:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim P_{data}(x)}[log(D(x))] + \mathbb{E}_{z\sim P_z(z)}[log(1-D(G(z)))] \tag{2.7}$$

To reach Nash equilibrium, one player cannot be substantially better at his job than the other. Rather both players must evolve together, in such a way that they learn from each other, and one does not overcome the other. In the original paper [23], at the early stages of training, the discriminator would easily deduce which images were real and which were fake since the generator would basically produce random noise. The effect of the discriminator being so good is that it would not provide enough information for the generator to make progress and it would fail due to vanishing gradients. In the original implementation [23], an alternative cost function is proposed to counteract this issue. Instead of training the generator to minimize $log(1-D(G(z)))$ we can train it to maximize $log(G(z))$. This objective provides much better gradients in early stages of training [2]. One other approach that tries to prevent vanishing gradients when the discriminator is optimal is the Wasserstein GAN (WGAN) [4], which introduces a novel loss function for the discriminator.

In the original GANs work [23] it was shown that if we decompose equation 2.7, we could conclude that GANs are trying to minimize the Jensen-Shannon (JS) divergence. In WGANs [4], the discriminator is called a critic, since it does not try to classify images as being real or fake, but instead tries to maximize the difference between real and fake outputs, which are not bound to $[0,1]$. This proved to be equivalent as minimizing the Wasserstein distance between the image distributions. This distance can also be referred as Earth Mover's (EM) distance, because

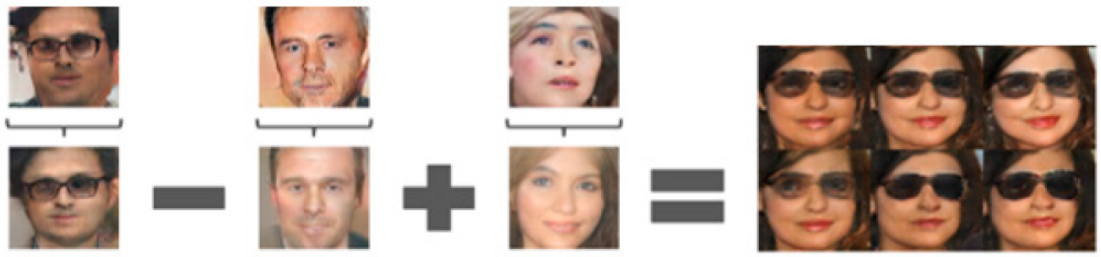


Figure 2.12: Example of latent-code arithmetic using GANs. Extracted from [63].

both distributions can be informally interpreted as being piles of dirt, with the minimum cost of transforming one pile of dirt into the other being the distance between distributions. This cost is the amount of dirt to be moved times the distance it was moved. This metric proves to be more useful than the JS divergence since it provides smoother measures between distributions. This is important when training models with gradient descent, since it offers more stable gradient values even when the critic network is well trained [96].

One other issue that may occur is the infamous mode collapse problem, when the generator's outputs gradually become less diverse, producing images from a single mode, for example containing the same color or texture themes or variations of the same subject [22]. In practice, this means that different input z values correspond to the same or very similar images. In a scenario where the generator produces only convincing images of a single mode, the discriminator will perform worse, being unable to distinguish real from fake images. The discriminator's gradients will encourage the generator to produce even more images of the same mode; after all, the generator's main goal is to confuse the discriminator, regardless of what images it produces. Progressively producing more images of a specific mode, it can no longer generate images of other modes. The discriminator will eventually catch up, and at that point, when it can recognise real from fake images, the generator's only choice is to start producing realistic images of another mode. This will continue over the course of training, and the generator will cycle through modes, never actually becoming able to produce a distribution containing all of the modes in the training set [22]. Some techniques that try to prevent this problem are experience replay and Unrolled GANs [52].

Experience replay tries to prevent mode collapse by feeding to the discriminator past images, produced by past versions of the generator, along with current fake and real images. This reduces the chances that the discriminator will overfit the latest generator's outputs [18]. Unrolled GANs [52] are another approach which consists of unrolling multiple discriminator's gradient steps while the generator model stays the same. Afterwards, the generator will backpropagate through all these steps, learning from future versions of the discriminator. This increases stability and diminishes mode collapse occurrence since the generator cannot over-optimise for a single discriminator.

Being generative models, the multiple losses used during training may not represent the final images' quality. To measures a GAN's performances the Inception Score (IS) [75] was widely

used as it is based on the KL-divergence, with a higher IS meaning better image quality. One major setback of the IS is that it is not sensible to model collapse. Another later introduced metric, the Fréchet Inception Distance (FID) [27] is now the most used metric. FID compares the generated and real image distributions computing the divergence between them, with a lower FID score implying closer distributions. The FID is sensible to model collapse, meaning that if the generator only outputs the same image the FID score will be high, meaning less output quality. Both these metrics give their names to the Inception network [84] which is used to calculate, in different ways, both metrics.

Despite its performance not being easily measured and being susceptible to all the aforementioned problems, a GAN generator can produce outstanding results if successfully trained. An interesting discovery is that one can intuitively manipulate the several latent-codes given to the generator, by applying simple arithmetic operations, and such operations would manifest visually in the produced images. This can be seen in figure 2.12, where latent-codes producing images of men wearing glasses, minus images of man without glasses, plus images of women without glasses, are actually equivalent to images of women with glasses. This indicates that the generator is, in fact, understanding some underlying structure of these images, and not just formulating random images that look somewhat realistic.

Some models that made GANs what they are today are Conditional GANs (CGANs) [54] and Deep Convolutional GANs (DCGANs) [63]. CGANs extend upon normal GANs for being able to choose which class to generate an image from. This is done by feeding an extra one-hot input vector, unique for each class, to both the discriminator and the generator during training. Afterwards, we give the generator the corresponding class vector, and an image from that class is obtained. DCGANs were influential since they were the first to produce reasonable results while only using convolution layers for their networks.

Some examples of successful GANs include StyleGAN [38], CycleGAN [107] and Pix2Pix [35]. StyleGAN is probably unrivalled when it comes to realistic image generation. It uses Adaptive Instance Normalization (AdaIN) [30] and adds noise multiple times after each convolution layer before applying non-linearities, throughout the generator model. It also separated coarse styles from fine styles, enabling mixing these two styles to produce new images with related characteristics. CycleGAN introduced a new cycle-consistency loss allowing GANs to operate without requiring paired images for image-to-image translations tasks. This model is particularly known for the specific example of converting horses into zebras, and vice-versa [79]. Pix2Pix is also an image-to-image translation network. It uses paired training images and improved network architecture to produce better results. In particular it uses a U-Net [69] architecture for the generator and a novel PatchGAN [47] architecture for the discriminator. Instead of classifying the entire image as being real or fake, PatchGAN classifies patches of an image and averages out the results to produce the final output of the discriminator. This can be seen a texture or style loss.

2.7 Contrastive Learning

The goal of contrastive learning is to learn an embedding space where similarly related instances are grouped together and dissimilar instances are distant from each other. This approach can be applied to various scenarios, although it is mainly utilized in self-supervised tasks.

Self-supervision aims to learn valuable representations from unlabelled data through a pretext task. This means that labels are automatically generated, and a task is created to guide the learning process. One example in the CV field is the task of image colorization, which aims to color grayscale images. For this goal, image pairs can be created by applying a grayscale augmentation to the colored dataset images, effectively labeling the dataset and allowing it to be used in a supervised manner. It is relevant to note that self-supervised and unsupervised scenarios are related but not exactly identical. Both scenarios take advantage of unlabelled data but unsupervised techniques tend to focus more on clustering, whereas self-supervision targets representational learning with the ultimate goal of solving supervised tasks.

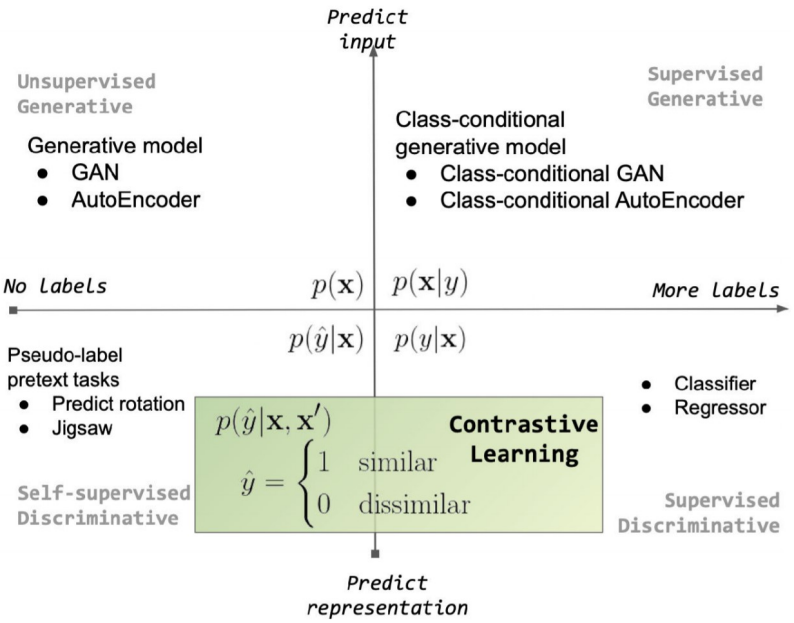


Figure 2.13: Illustration of how contrastive methods are located in the spectrum of generative-discriminative and supervised-unsupervised scenarios. Extracted from [45].

Contrastive learning is not a novel concept; however, it has recently become famous for achieving performance results in line with supervised alternatives without any labeled data. One example is SimCLR [8] which introduces a contrastive self-supervised framework that can be applied to most architectures without the need for custom modifications. There are multiple contrastive losses, but the idea on which they are based remains the same: they all intend to distinguish between positive and negative pairs of samples, hence the name, contrastive learning. In practice, a sample from the dataset is used as a query to which we aim to correlate with other positive samples

but find discrepancies when comparing with negative samples. In SimCLR, positive samples are generated from the query by applying data augmentations that do not significantly alter the sample content. At the same time, the remaining images in the training batch are used as negative samples since they are expected not to be closely related to the query image.

The main contrastive loss function used in SimCLR, named normalized temperature-scaled cross-entropy loss (NT-Xent), is as follows:

$$\mathscr{L}_{NT-Xent}(q, v^+, v^-) = -\log\left[\frac{\exp(q \cdot v^+/\tau)}{\sum_{v \in \{v^+, v^-\}} \exp(q \cdot v/\tau)}\right] \tag{2.8}$$

with q as the query embedding, v^+ and v^- as the positive and negative embeddings respectively, $(\cdot)$ as the dot product between vectors, and τ as an adjustable temperature parameter. As can be seen, NT-Xent resembles the commonly used cross-entropy loss, and by comparing the two, we can easily understand the contrastive loss's purpose. The main difference is that instead of probability values, the dot product between embeddings is used to measure the distance between embedding vectors. The dot product is preferred because when working in a high dimensional space, euclidean distances, such as L1 or L2, do not adequately depict embeddings' relative position. Since the embeddings in SimCLR are L2 normalized, the dot product between vectors effectively measures the angle between them, making its value fit within the range $[-1, 1]$. Therefore, the dot product between high resemblance embeddings will be closer to 1, while completely unrelated samples will tend towards -1.

The authors of SimCLR mention three major findings in their experiments:

- The data augmentations used to create the positive samples are crucial to generate useful embeddings. They found most success using both random cropping and random color distortion on the query sample, since cropping forces the model to have both local and global understandings of the input, and color distortion does not allow the model to "cheat" by simply matching similarly colored crops.

- Applying the contrastive loss directly on the learned representations provided suboptimal results. By passing the embeddings through a 2-layered MLP, denominated as projection head, and applying the loss at the end helped the projection head's previous layer to retain more useful information.

- More training epochs, bigger networks and increased batch sizes during the training process yields better representations, as more negative embeddings are available to differentiate from the query. This is true for most supervised methods, but these changes produce more noticeable improvements on SimCLR.

Furthermore, SimCLR's NT-Xent loss is not the only contrastive loss available. Other variations like InfoNCE [60] do not employ the dot product between vectors as an embedding similarity

function but instead make use of a simple linear transformation on the embeddings. Another example is the Triplet loss [76], which takes into account the relative distance between positive and negative pairs from the query:

$$\mathscr{L}_{triplet}(q, v^+, v^-) = max(0, D(q, v^-)^2 - D(q, v^+)^2 + m) \tag{2.9}$$

with $D(v_1, v_2)$ as a function that measures the distance between embeddings v_1 and v_2 and m as a margin parameter, which represents the desired minimum gap between positive and negative pairs.

To conclude, contrastive learning has led to significant improvements to self-supervised learning, making DL use in real-world applications, where labeled data is scarce, more feasible. It is a broad field with the potential to create powerful data embedding models without requiring labels.

Chapter 3

Domain Adaptation: Literature Review

In this section, DA will be explained as a sub-field of TL. It starts with a brief introduction of TL to contextualize the problem of DA, followed by an overview of what types of DA can be categorized. Afterward, the focus will be shifted to the different one-step homogeneous DA methods found in the reviewed literature. Domain Randomization (DR) is addressed, followed by some applications and popular datasets for DA. The chapter is concluded with a summary of the discovered methods.

3.1 Introduction

As already mentioned, DA derives from the field of TL. TL's main goal is to re-utilise knowledge across different domains and tasks. In TL problems we can expect a source domain, $\mathfrak{D}_s$, in which we want to perform a certain task, $\mathfrak{T}_s$, and a target domain, $\mathfrak{D}_t$, in which we perform another task, $\mathfrak{T}_t$. In most cases, the source domain is substantially larger than the target domain.

Both [61] and [97] define TL as a method to improve a model's predictive performance on a certain task in the target domain while making use of the knowledge obtained from performing another task in the source domain, where the domains and/or the tasks differ. From this definition distinct scenarios arise in which TL occurs, as can be seen on table 3.1. More complex scenarios can involve the transfer of knowledge between more than 2 domains. This is later briefly mentioned when the distinction between One-step and Multi-step DA is addressed.

As seen in table 3.1, a distribution mismatch between the source and target domain is assumed in the area of DA, whereas the goal of achieving the same task is fixed. The level of relatedness between the source and target domains usually determines how successful the adaptation will be [29].

Both [61] and [94] try to define TL and DA as well as the distinct sub-areas they can be divided into. However, their definitions of how DA fits into the TL landscape are not quite aligned. In [61], while DA is described as a Transductive TL approach, where we try to combine different, but related, source and target domains to achieve one unique task, it is also said that there are no target

Table 3.1: Transfer Learning settings

Learning Settings		$\mathfrak{D}_{source} = \mathfrak{D}_{target}$	$\mathfrak{T}_{source} = \mathfrak{T}_{target}$
Traditional Machine Learning		✓	✓
Transfer Learning	*Inductive Transfer Learning* |	✓	✗
	Unsupervised Transfer Learning	✗	✗
	Transductive Transfer Learning **(DA)**	✗	✓

domain labels available. This conflicts with [94] since, in this work, DA is divided into supervised, semi-supervised and unsupervised DA, which inherently says that, in fact, the absence of target labeled data is not always true. Besides their discrepancies, they tend to agree more than disagree, and this contradiction seems to be more of a semantic concern. DA's main focus is that the target domain data is insufficient for the task at hand, either due to the target dataset being too small or due to labels being scarce or non-existent.

One question worth mentioning, albeit not the focus of this work, is determining if a source domain can be adapted to a target domain. Apparently, this issue of defining similar tasks does not have a simple theoretical answer. Nonetheless, simple reasoning is often sufficient to determine task relatedness.

3.2 Notation

Here the notation used in the reviewed surveys [61, 94, 85] and in the following sections is clarified:

- $\mathscr{X}$ represents the input feature space
- $\mathscr{Y}$ represents the output feature space, i.e. the set of all dataset labels
- $P(X)$ is a marginal probability distribution of a particular learning sample X, where $X = \{x_1, ..., x_n\} \in \mathscr{X}$
- $f(.)$ is an objective predictive function used to predict an instance's, x, corresponding label, $f(x)$. From a probabilistic perspective, $f(.)$ can be viewed as a conditional probability distribution $P(Y|X)$. This function is not specified but it can be learned from labeled training data $\{x_i, y_i\}$, where $x_i \in X$ and $y_i \in Y$
- A domain $\mathscr{D}$ consists of a feature space $\mathscr{X}$ and a marginal probability distribution $P(X)$, i.e. $\mathscr{D} = \{\mathscr{X}, P(X)\}$
- Given a specific domain $\mathscr{D}$, a task $\mathscr{T}$ consists of a feature space $\mathscr{Y}$ and an objective predictive function $f(.)$, i.e. $\mathscr{T} = \{\mathscr{Y}, f(.)\}$ or even $\mathscr{T} = \{\mathscr{Y}, P(X|Y)\}$
- From the definitions above, $\mathscr{D}_s = \{\mathscr{X}_s, P(X_s)\}$ and $\mathscr{T}_s = \{\mathscr{Y}_s, P(X_s|Y_s)\}$ represent the source domain and task, respectively. The same logic applies to the target domain $\mathscr{D}_t = \{\mathscr{X}_t, P(X_t)\}$ and corresponding task $\mathscr{T}_t = \{\mathscr{Y}_t, P(X_t|Y_t)\}$

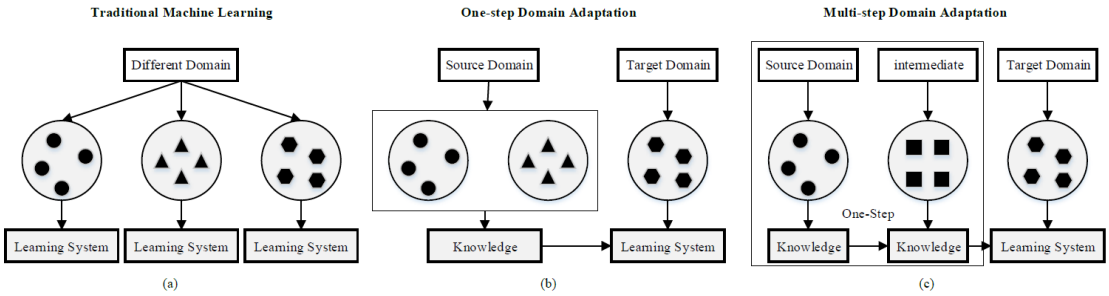


Figure 3.1: Different learning processes between (a) traditional machine learning, (b) one-step domain adaptation and (c) multi-step domain adaptation. Extracted from [94].

3.3 Taxonomy

As TL can be categorized into different areas, so can DA be present in different scenarios. Firstly, the distinction between deep and shallow DA must be addressed. Pan et al. [85] states that deep DA can be broadly referred to as a task where f_T is represented by a DNN. Besides mentioning this notion, Wang et al. [94] also provide a more narrow definition. Deep DA is based on DL architectures designed for DA and can be directly affected via backpropagation. To explain, this means that the DA process is embedded into the neural network's feature learning layers, storing a semantically meaningful and domain invariant representation of features. In contrast, shallow or traditional DA is performed by methods that do not take advantage of DNNs. Despite not being the focus of this work, shallow DA can be briefly categorized as being:

- **Instanced-based:** the discrepancy between domains is reduced by re-weighting the source samples, which are closer to the target samples, and then train a classifier on these re-weighted samples.
- **Feature-based:** the different domain distributions are matched on a learned shared feature-space.

Moving the focus onto deep DA, in some scenarios combining useful representations between two domains may prove to be extremely difficult due to the disparity between them. Such a problem can be solved by applying DA multiple times. If one intermediate domain can be found, that acts as a connection, sharing aspects of both source and target domains, one can perform two DA steps, from source to intermediate, followed by intermediate to target domain. This is called multi-step or transitive DA, as opposed to normal one-step DA. The differences between these learning paradigms are shown in figure 3.1, first depicted in [61] and further complemented in [94].

As already mentioned, DA represents a scenario where we want to perform a unique task, i.e. $\mathscr{T}_s = \mathscr{T}_t$, by joining knowledge from different domains, i.e. $\mathscr{D}_s \neq \mathscr{D}_t$. Therefore, by definition, domain divergence can be caused by either feature space difference ($\mathscr{X}_s \neq \mathscr{X}_t$) or by distribution

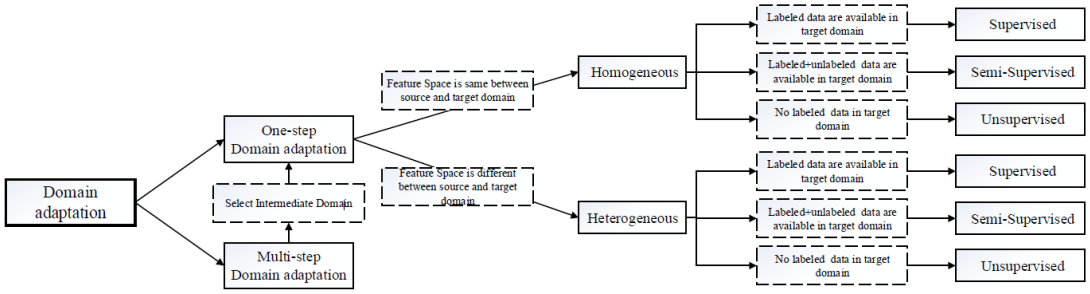


Figure 3.2: An overview of different settings of domain adaptation. Extracted from [94].

shift ($P(X_s) \neq P(X_t)$). Pan et al. [61] made a very intuitive analogy to distinguish these causes. If we consider a document classification task, feature space difference occurs when the two domains, in this case sets of documents, are written in different languages. At the same time, distribution shift can be analogous to topic differences between source and target documents.

Wang et al. [94] divided DA into homogeneous and heterogeneous DA, depending on which divergence exists between domains:

- **Homogeneous DA:** feature spaces are identical ($\mathcal{X}_s = \mathcal{X}_t$) with the same dimension ($d_s = d_t$), but the source and target domains have dissimilar data distributions ($P(X_s) \neq P(X_t)$).
- **Heterogeneous DA:** features spaces are nonequivalent between domains ($\mathcal{X}_s \neq \mathcal{X}_t$) with generally different dimensions as well ($d_s \neq d_t$). Since data features are different it makes sense to assume some sort of distribution shift ($P(X_s) \neq P(X_t)$) as well.

Among all these categories, as in traditional ML tasks, DA can also be divided according to label prominence:

- **Supervised DA:** both domains are labeled, but there is a shortage of samples in the target domain, being insufficient to perform the target task provided only with the target dataset.
- **Semi-supervised DA:** source domain is labeled and only a few target samples have annotations.
- **Unsupervised DA:** source domain is labeled and there are no target annotations to be used.

In figure 3.2 we can see an overall taxonomy of DA possibilities. The following sections focus on **one-step DA techniques**. These can be broken down into **Divergence/Discrepancy-based**, **Adversarial-based** or **Reconstruction-based** methods.

3.4 Divergence-based Methods

Divergence-based methods, can also be called discrepancy-based methods, take advantage of previously trained networks on the source domain and fine-tune them to the target domain. In this

section, the major techniques used for fine-tuning are enumerated, with one of them being highlighted, the statistical criterion. Intending to find a domain-invariant representation of features between the two domains, this criterion enforces a divergence measure during the fine-tuning process. Several criteria measure the difference between probability distributions, which are discussed later in section 3.4.1.

The process of fine-tuning a network consists of taking already trained networks' layers and either freezing them or continue the training process on a different training dataset. By freezing layers, it is meant that their parameters are static and are not altered by gradient descent. Normally, the first layers of a network are frozen since, as mentioned in section 2.4.1.1, these layers retain non-specific dataset features, containing useful information that can be re-utilized on a new dataset. This process is not uncommon, and using parameters from prominent network architectures like VGGNet [81] or ResNet [26], trained on a broad dataset with a wide variety of classes like ImageNet [73], is often applied when dealing with CV tasks [104].

If such an invariant representation is found, a single classifier can be applied and perform effectively on both datasets. This assumes that the domains are somehow related and that such representation exists and can be found. However, extracting useful features from both datasets is as important as trying to merge them. In theory, fine-tuning appears to be a simple process, but in practice, some difficulties must be overcome for a network to retrieve relevant features properly.

Yosinski et al. [104] studied feature transferability between networks and concluded that initializing with already trained layers can substantially improve performance. Nevertheless, the authors also observed two main issues: fragile co-adaptation and specialization of layers to source domain features. Fragile co-adaptation happens when units from two consecutive layers heavily rely on each other for feature extraction in such a way that is not re-learnable by the following layers. This is problematic if one unintentionally decides to freeze co-adapted layers. Layer specialization manifests in the last layers of a network, where features are domain-dependent. This is the reason why the first layers are re-purposed while the last ones are often discarded. The choice to either freeze or to fine-tune even the first layers depends on dataset size and similarity between domains [100]. When achieving a matching feature representation for DA, these last layers are the ones where the divergence criterion is applied due to their discrepancy.

There are four prominent criteria for fine-tuning a DNN:

- **Class Criterion:** can also be referred to as label criterion, takes advantages of the labeled samples in the source domain, and applies retrieved knowledge to the target domain. This is the most basic method of fine-tuning a network. Techniques like soft-labels [88] and pseudo-labels [101] are examples of this approach. In [88] the authors applied a cross-entropy soft-label loss, where a sample does not have a single label but rather a probability distribution over all the labels. These probabilities contain important information that would normally be lost, describing the similarities and differences between classes. After training on source data, soft-labels are formed by averaging the samples' class scores. When fine-tuning with target data, the model enforces the output target class scores to be similar to the source class scores. One disadvantage of this technique is that some target samples must

already be annotated to apply this soft-label loss. Another approach [101] that does not require target labels is using pseudo-labels, which are predicted target labels obtained from a network trained on source data. Model training proceeds as usual, with an unlabeled loss using pseudo-labels weighted into the final normal labeled loss.

- **Statistic Criterion:** this criterion tries to measure how different two probability distributions are. If the network's loss is based on these domain discrepancy measures, one can force the source and target distributions to align through backpropagation. These metrics are discussed in section 3.4.1.
- **Architecture Criterion:** as the name implies, this criterion refers to altering networks' layer structure to close the gap between domains. Some examples of this are adaptive batch-normalization (AdaBN) [48] and domain-guided dropout [99]. The idea behind AdaBN is straightforward and easy to implement. To explain, standard batch-normalization (BN) [34] makes each layer receive data from an identical distribution. When fine-tuning, target samples do not benefit from BN layers due to domain differences. In contrast, AdaBN recalculates these layer statistics when training with target data in an attempt to align both distributions. Xiao et al. [99] tackles the issue of some units capturing features of only a single domain or reacting to non-useful information presented in the input image by introducing domain-guided dropout. This mechanism measures the impact of every single particular unit on the network's performance. Evidently, calculating this for all units is unfeasible. Hence, the authors apply a technique to approximate a unit's impact on the final score. Based on this assessment, units with adverse impact are deactivated when a specific domain batch is used for training. This proved to be more effective than normal dropout, most notably on small datasets.
- **Geometric Criterion:** this criterion tries to find a geometric connection between both domains. Chopra et al. [10] try to create new interpolated domains between the source and target datasets. This approach combines features from these intermediate samples in an attempt to bridge the domain gap. This is a less prominent criterion.

Some of these criteria require labeled data on the target domain and are not applicable in unsupervised DA scenarios. In contrast, adversarial and reconstruction-based methods assume the absence of labeled data in the target domain, a more realistic and generally more applicable scenario.

3.4.1 Divergence Measures

As a way to align distributions, some metrics for measuring similarities/differences between probability distributions can be implemented as loss functions, as a way to encourage the network itself to retain domain-agnostic features. In this section, some of those metrics are discussed. Besides being used for DA, these metrics are also used for evaluating differences between generated images or assess the discrepancy between features captured by different layers of a network [90].

3.4.1.1 Maximum Mean Discrepancy

Maximum Mean Discrepancy (MMD) tells us if two distributions are equal by comparing their means after mapping them to a reproduced kernel Hilbert Space (RKHS). Tzent et al. [90] used this metric in a single hidden layer, and empirically approximated this distance as follows:

$$MMD^2(X_s, X_t) = \left\| \frac{1}{|X_s|} \sum_{x_s \in X_s} \phi(x_s) - \frac{1}{|X_t|} \sum_{x_t \in X_t} \phi(x_t) \right\|^2 \tag{3.1}$$

where ϕ is a function that maps its inputs to a RKHS, X_s and X_t represent source and target batches of data, respectively.

In [90] the authors modeled two networks, one for labeled images that calculates a classification loss at the end and another for unlabeled images used to calculate a domain loss. These networks have shared weights, which means that the layers have the same parameters. Hence, updating parameters by backpropagating on one network will also affect the other. The domain loss is calculated in a single layer using MMD over batches of sampled source and target data, as specified in equation 3.1. The questions of which layer one should calculate this loss in and the dimensions of such layer are also contemplated. This is done empirically by finding in which layer the MMD distance is smaller. Afterward, a grid search is made to decide the layer's dimensions. These are considered as descent steps to achieve domain confusion. Since the network trains with these two losses simultaneously, it learns a feature representation that is not only domain invariant but also useful for classification tasks, allowing the classifier to be used in both domains. One downside of this is that if the task should change, the network must be re-trained to optimize the classification section.

Bousmalis et al. [6] applied a reconstructive approach (section 3.6.1) to the DA problem. In this work, the authors experimented with two similarity losses, MMD and an adversarial based loss, concluding that the MMD did not perform as well as the adversarial loss.

3.4.1.2 Correlation Alignment

Correlation Alignment (CORAL) [82] is similar to MMD and minimizes domain shift by aligning the second-order statistics (covariances) of both dataset distributions, and requiring no target labels. Sun et al. [83] applied this metric to DNNs (deep CORAL). This work shares a similar architecture and thought process to [90], but instead of an MMD based loss, the CORAL metric was used. The CORAL loss was defined as follows:

$$L_{CORAL} = \frac{1}{4d^2} \|C_s - C_t\|_F^2 \tag{3.2}$$

where $\|.\|_F^2$ represents the squared Frobenius norm (matrix euclidean norm), C_s and C_t are the source and target feature covariance matrices, respectively.

3.4.1.3 Contrastive Domain Discrepancy

Contrastive Domain Discrepancy (CCD) [37], being a more recent metric, distinguishes itself from MMD and CORAL because it also takes into consideration sample labels. It tries to minimize the domain gap while retaining features that allow class discrimination. This is depicted in figure 3.3.

In their work [37] the authors propose a Contrastive Adaptation Network (CAN) that utilizes both a normal cross-entropy loss for the classification task and a CCD domain divergence loss. However, for CCD to take into account sample labels, there must be target labels. To accommodate this, a clustering algorithm, more specifically spherical K-means, is applied to the target domain, with corresponding labels attached to samples. The outliers of the clustering process that do not confidently associate with a class are not considered when calculating the CCD loss.

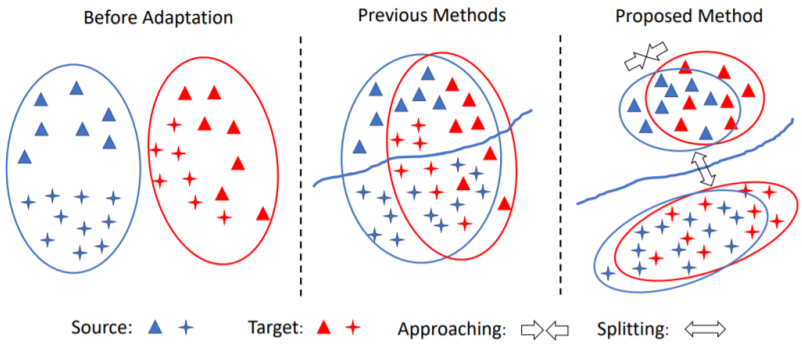


Figure 3.3: Comparison of previous discrepancy metrics and what CCD aims to achieve. Compared to previous methods that only try to approximate the distributions without considering their inner classes, CCD, besides minimizing divergence inside each class it also maximizes discrepancy among distinct classes. Extracted from [37].

3.5 Adversarial-based Methods

In this section, adversarial methods, which focus on unsupervised settings of DA, are introduced. In contrast to divergence methods that try to find a domain invariant feature representation through fine-tuning with different criteria as their loss functions, adversarial methods implement an adversarial loss, which does not directly measure differences between distributions, but rather encourages the domains to be indistinguishable. Contrary to other losses like MMD, CORAL, or CCD, an adversarial loss is not a predefined function. The adversarial loss calculation does not measure domain invariance in any way. It can be seen as an intuitive concept for which the network must learn to optimize. This optimization will grant us domain confusion in some form. This notion comes from GANs [23], explained in section 2.6, where the generator network and the discriminator are jointly trained with distinct goals. In fact, some models use the GAN framework to achieve a form of DA, while other models prefer to take advantage of an adversarial loss without generating novel images. In the following sections, these adversarial DA sub-groups are explained: generative models and non-generative models.

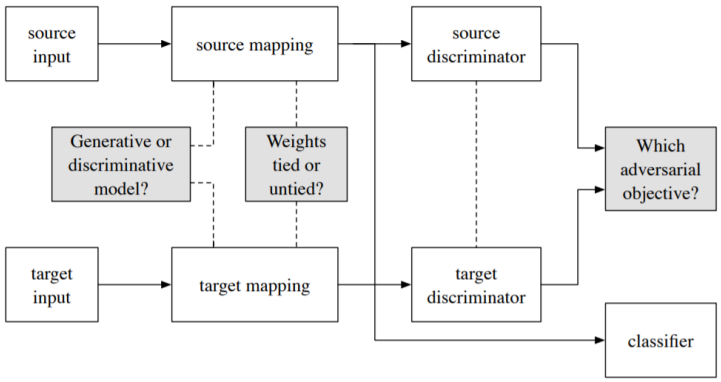


Figure 3.4: Unified adversarial framework. Extracted from [89].

Tzeng et al. [89] proposed a unified framework for adversarial methods, which can be seen in figure 3.4. Table 3.2 shows the discussed literature methods in later sections as instances of this framework.

Table 3.2: Discussed methods according to [89] framework.

Method	Base model	Weight sharing	Adversarial loss
Domain confusion [88]	discriminative	shared	domain confusion
CoGAN [50]	generative	semi-shared	GAN (feature loss)
SimGAN [80]	generative	unshared	GAN (pixel loss)
PixelDA [5]	generative	unshared	GAN (pixel + semantic loss)
DANN [14, 15]	discriminative	shared	minimax
ADDA [89]	discriminative	unshared	GAN

3.5.1 Generative Models

Generative models aim to map one domain to another using the GAN [23] framework instead of searching for a middle-ground feature space. As their name implies, a generator network is in place, which tries to convert labeled images from the source domain to the target domain while maintaining their distinctive features. Therefore, in some way, we are enriching the target dataset with labeled samples. Afterward, a target domain classifier is trained with the newly converted labeled data. An adversarial loss can focus on different aspects, or the model implements extra losses which complement the adversarial one for better results. In this section three losses are mentioned, a feature loss [50], a pixel-level loss [80, 5] and a semantic loss [5].

3.5.1.1 Feature Loss

Coupled GANs (CoGANs) [50] makes use of two GANs with shared weights. Both generators, G_1 and G_2, receive a random input vector z and attempt to create images from both domains, $\hat{x}_1 = G_1(z)$ and $\hat{x}_2 = G_2(z)$. The discriminators, D_1 and D_2, receive images drawn from the training

set and images drawn from the respective generative networks and attempt to correctly classify images according to their origin, $D_1(G_1(z))$ and $D_2(G_2(z))$.

Both generators share weights in their first layers, while both discriminators share weights in their last layers. These layers, as previously mentioned in the former section 2.4.1.1, are responsible for capturing high-level semantics in images. In classification tasks, standard DNNs encode information to provide an output score; however, since the generators decode the input z, it would make sense that their first layers, rather than the last, would decode some high-level information. Weight sharing is done to find high-level feature information common to both domains. Therefore, it is said that CoGANs incorporate an adversarial feature loss [97]. CoGANs training is the same as regular GANs, backpropagating alternating between discriminator and generator gradients.

The authors applied CoGANs to unsupervised DA settings by adding a softmax classification layer to both discriminators. After training, optimizing jointly for the classification task in one domain, and for the CoGAN main objective of domain confusion using images from both domains, two classifiers were produced, one for each corresponding domain.

CoGAN brings a particular advantage over previous methods since there are two generators, from a single z input, a pair of images from corresponding domains can be generated. Nevertheless, this approach can have difficulties converging, not only because of GANs' sensibility to hyperparameters but also when both domains are not closely related [89]. Furthermore, the classification portion is trained jointly with CoGAN, requiring a new model if the task changes with the domains remaining the same.

3.5.1.2 Pixel-level Loss

SimGAN [80] and PixelDA [5] are models that work at the image's pixel level in some way. SimGAN and PixelDA both work on DA methods for synthetic and real images. Improving upon the original GAN framework, these methods convert synthetic images into realistic counterparts. By doing so, the classifier portion is disentangled from the DA task and does not require to be jointly trained with the rest of the network.

SimGAN presents a simulated+unsupervised learning method, considering the GAN generator as an image refiner for synthetic images. This refiner network receives as input simulated images instead of random vector inputs. SimGAN's refiner is only comprised of convolution layers with zero stride and no pooling operations. This means that it does not encode nor decode any image features. Instead, it modifies the original image at a pixel-level with the sole intent of producing a more realistic one. A pixel-level regularization term complements this, implemented using the L1 distance between the synthetic and refined images.

Another modification on the discriminator's loss was made. Rather than comparing as a whole refined and real images, it classifies corresponding patches of the images as real or fake, with the final output being the sum of multiple cross-entropy losses. During the training process, experience replay was used, maintaining a history of past images and sampling past and current refined images to the discriminator. This, combined with the local-patch loss, reduces the chances of mode collapse occurring and improves image clarity, generating fewer artifacts in the refined

images. Seo et al. [78] applied SimGAN to the problem of detecting and classifying aircraft in satellite images by refining aircraft images overlayed on some background image, producing more realistic samples.

PixelDA's approach is identical to SimGAN but with an important difference. It maintains the GAN's original random input z, concatenating it together with the synthetic images before feeding them to the generator. This noise allows for an infinite generation of novel target images. The generator can also be seen as an image refiner, adjusting individual pixels of the source image. A pixel-level loss is also in place. This time it makes use of the simulator's z-buffer, which provides depth information, to create a mask between background and foreground, penalizing inconsistent foreground differences. The discriminator remains the same as the original GAN.

3.5.1.3 Semantic-level Loss

PixelDA's authors noticed that providing the final classifier with only refined images would sometimes cause shifts in class assignments. This means that the model would distinguish classes successfully but assigned the wrong class label. This was corrected by feeding source images along with refined ones, forcing the refined label to be equal to the corresponding synthetic image label. This can be seen as a semantic consistency loss for the task at hand. Also, PixelDA obtained better results than CoGAN in its tests.

3.5.2 Non-generative Models

Non-generative methods do not attempt domain mapping. Thus, no novel images are generated. These methods share the same goal as divergence methods, that is to find a domain-invariant feature representation, this time in an adversarial manner.

As previously mentioned, Tzeng et al. [89] introduced an adversarial framework based on previous works (fig. 3.4). Having evaluated previous methods, the authors created a new model as a new instance of the framework, called Adversarial discriminative domain adaptation (ADDA). This approach, despite applying a GAN-based loss, does not generate novel images. Its training comprises two steps. First, an encoder and normal classifier are trained on the source data, where labeled data is available. Subsequently, this source encoder, in conjunction with a target encoder, are trained in an adversarial manner, where a discriminator will attempt to assign domain labels to each one's latent representation to encourage domain confusion. During this step, the source encoder weights are fixed. If we compare this adversarial adaptation step to GANs, the GAN's generator is analogous to the target encoder, trying to extract useful features from the target domain while making them indistinguishable from the features outputted by the source encoder. At testing time, the target encoder and the source classifier are used to classify target data.

Surprisingly, there are no loss functions nor regularization measures besides the adversarial loss. During the adaptation phase, the target encoder might produce outputs that blend in with the source outputs without preserving any characteristics from the target domain. After all, the classifier had already been trained on source data, and it is not used in this step. To ensure the

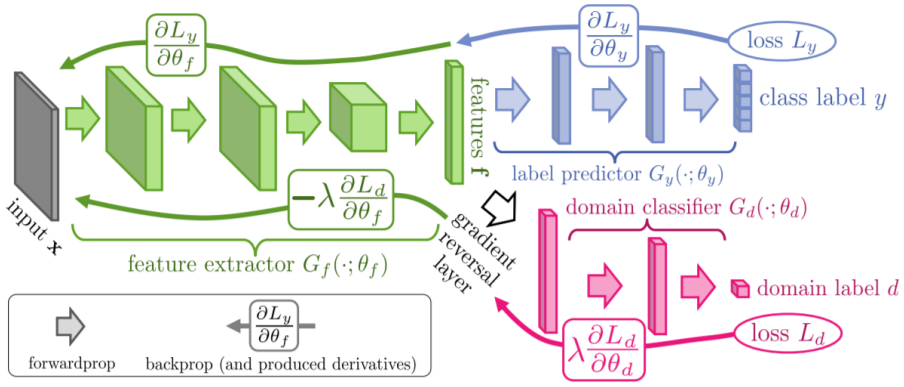


Figure 3.5: DANN model architecture, including a feature extractor, label predictor and a domain classifier connected to the feature extractor by a gradient reversal layer. Extracted from [15].

target encoder would produce useful features, the authors used the source encoder as a base for the target encoder and fine-tuned it during training.

Another approach that does not focus on image generation are Domain-adversarial neural networks (DANN) [14, 15]. DANN's model components are similar to ADDA in some ways. They both have feature extractors (denominated as source and target encoders in ADDA), a normal class classifier, and a domain classifier (in ADDA this is the discriminator network). While in ADDA these are treated as separate networks, DANN considers all these components as part of a single DNN and introduces a novel Gradient Reversal Layer (GRL) to simulate a GAN-like loss. This is depicted in figure 3.5.

DANN minimizes the final task's label prediction loss only for source samples but promotes domain confusion for all samples, instead of only for the target encoder compared to ADDA. ADDA could be interpreted as converting target data to be as close to source data encodings as possible since the source encoder is not fine-tuned during adversarial training. DANN, on the other hand, minimizes the domain gap jointly for source and target samples.

The domain classifier loss is not adversarial. It just tries to classify outputs of the feature extractor as belonging to the source or target domains. Hence, this loss produces gradients that encourage the feature extractor portion of the network to increase the domain gap. To achieve domain invariance, a GRL connects the feature extractor to the domain classifier and only affects the gradients propagating from the domain classifier to the feature extraction layers. This layer has no effect during the forward pass, behaving as the identity function (equation 3.3). During the backward pass, the GRL multiplies the gradients by a non-learnable negative scalar (equation 3.4), thus performing gradient ascent on the domain loss and promoting domain alignment through the backpropagation process itself. The GRL, with λ as an arbitrary scalar and $\mathbf{I}$ as the identity matrix, can be defined as such:

$$GRL_\lambda(x) = x \tag{3.3}$$

$$\frac{\partial GRL_\lambda}{\partial x} = -\lambda \mathbf{I} \tag{3.4}$$

The authors conclude that the GRL can easily be implemented into any DNN architecture trained with backpropagation, proving to be a flexible and scalable option that could be applied to tasks other than DA [14, 15].

3.6 Reconstruction-based Methods

Up to this point, divergence and adversarial methods tried reducing the domain gap by utilizing divergence criteria or adversarial losses that in some way measure this gap and encourage the model to lessen the domain differences. Reconstruction methods try to recreate the original images/features from the intermediate representations found between source and target domains. Hence these models implement some form of reconstruction loss. This loss encourages the intermediate representation to preserve sufficient information from the original image so that recreating it is possible. We can separate these methods, depending on whether the networks' base structure is an AE, a VAE (section 2.5) or a GAN (section 2.6), into encoder-decoder methods or adversarial methods, respectively.

3.6.1 Encoder-Decoder Reconstruction

Deep Reconstruction-Classification Network (DCRN) [19] is a simple implementation of this method. It makes use of a normal AE to encode and decode the input target images. For image understanding tasks, since AEs do not have any fully connected layers, these were added when a loss was to be calculated. A cross-entropy classification loss is calculated from the encoder's outputs, whereas a mean squared error loss between the input image and the decoded image is added as a reconstruction loss. Source and target data are trained jointly, with only source labeled data being used for the classification loss and only target data for the reconstruction loss. Notice that DCRN does not apply any domain divergence loss. Training both domains jointly can be seen as a form of fine-tuning, which in conjunction with the reconstruction loss, was enough for the authors to believe domain invariance was being achieved. Dropout in the fully connected layers, data augmentation, and the denoising process from DAEs [93] acted as regularizers to prevent overfitting and improved model generalization.

Ghifary et al. [6] created Domain Separation Networks (DSN), encoding distinct aspects from each domain and also imposing a similarity loss between the shared encodings of different domains. In DSN, the notion of a private domain subspace is introduced. Each domain has shared

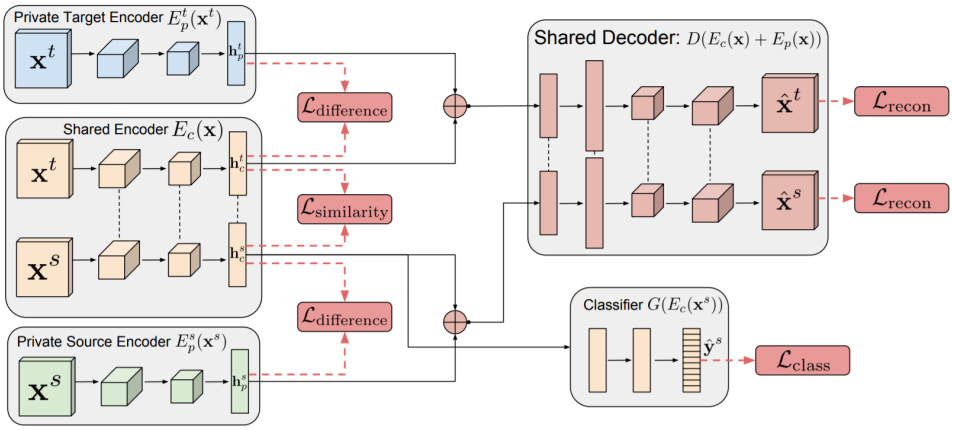


Figure 3.6: DSN model architecture and used loss functions. Extracted from [6].

useful information, assuming they are related in some manner. However, there is also undesired information that causes the domain gap to exist. This is encoded into a domain-specific private space, containing noisy features inherent to each domain but not shared by both. Since the shared features are encoded separately from the private ones, they are expected to be more meaningful to achieve our final goal.

They used various losses to ensure domain invariance and promote suitable shared encodings that, coupled with their private counterparts, can reconstruct the input image. This process is finalized with the addition of a classification portion that receives the shared representations and tries to classify them accordingly. Specifically, four distinct loss functions were implemented:

- **Shared encodings similarity loss:** both MMD and the GRL from DANN [14, 15] were tested. The use of the GRL obtained better results in their experiments.
- **Shared - Private encodings difference loss:** promoted orthogonality between encodings and was implemented as the squared Frobenius norm of the matrix multiplication between the shared and private encodings of each domain.
- **Reconstruction loss:** as in [19], a mean squared loss was used.
- **Classification loss:** depends on the task. For the authors' task, a cross-entropy loss was used.

The authors did not specify if the classification network was trained jointly with the encoders and decoders. One could think this would be possible. This disentanglement would prove helpful if the ultimate task changed. Nevertheless, if this is not expected, perhaps training it jointly with the remaining portions of the network produces more task-specific encodings, which could improve a specific task's performance. An overview of the DSN encoding spaces and corresponding losses can be seen in figure 3.6.

A more recent approach is Sim2Real [64]. In this work, a segmentation and depth task were the ultimate goal. For this, besides the usual encoders and decoders, which had semi-shared weights for source and target domains, two additional decoders were added. These auxiliary decoders, which were trained after the other portions of the network, reconstructed segmentation and depth maps from the shared latent space representations. Another important addition was the application of a cross-domain cycle-consistency loss, whose notion first appeared in other adversarial reconstruction methods [107, 40]. An input from a specific domain is encoded into the shared latent space and then decoded into the other domain. Subsequently, this process is inverted, with the now converted image being encoded to the latent space and then decoded back to its original domain. At this point, the output image cycled through domains and is compared to the original image. This cycle-consistency loss will prove to be a critical component for the adversarial reconstruction methods discussed in the next section.

3.6.2 Adversarial Reconstruction

These methods are based on GANs [23] and reconstruct images by applying a cycle-consistency loss. Some methods apply this domain mapping operation directly, while other methods apply a strategy similar to Encoder-Decoder Reconstruction (section 3.6.1) and reconstruct the images in an adversarial manner after encoding them into a shared latent space. The latter stem from VAE-GANs [43] where the VAE decoder and GAN generator are the same network and reconstruct samples from the encoder's latent space while the discriminator judges their realism. Hence, this section is divided into methods that perform direct domain mapping and methods that learn domain-invariant features before reconstructing them to another domain.

3.6.2.1 Domain Mapping

These methods comprise two generators, one that converts images to the target domain and another one that converts them to the source domain. A cycle-consistency loss, usually the L1 or L2 distance, between the original and the reconverted images is applied to force feature preservation during the cross-domain conversion. This is depicted in figure 3.7. Any model that performs this transfer process from one domain to another can be used for DA since it does not require paired images from both domains. A classifier trained on source labeled data, for example, images from a simulator, can be used to classify converted images from the target domain.

Some models were concurrently released that used this cycle-consistency loss, albeit with some small variations regarding their specific loss functions and mechanisms to help the convergence of the applied GANs. These works were CycleGAN [107], DiscoGAN [40] and DualGAN [103], all sharing the same core concepts. Nevertheless, they have some distinct implementation details, besides network layers constitution. For example, DualGAN chooses to implement a critic instead of a discriminator, effectively using the Wasserstein loss [4]. Both CycleGAN and DualGAN use the L1 distance as reconstruction loss, while DiscoGAN uses the L2 distance. In fact, Larsen et al. [43] suggest using the L1 norm instead of the L2 since the latter tends to produce

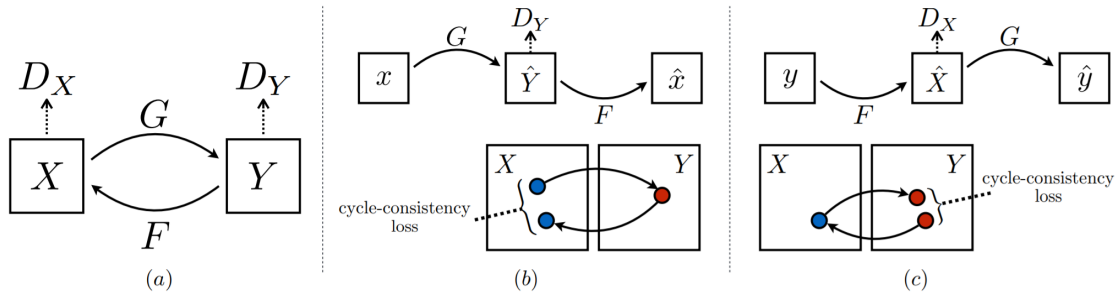


Figure 3.7: (a) Method overview with two mapping functions and corresponding adversarial discriminators (b) forward cycle-consistency loss (c) backward cycle-consistency loss. Extracted from [107].

more blurry images. In addition, CycleGAN has a hyperparameter that determines how influential the cycle-consistency loss is in the final result. DualGAN takes this even further, allowing to tweak this hyperparameter for each domain cycle. CycleGAN adopted a PatchGAN architecture [47, 35, 80] for the discriminator and experience replay, with a buffer of 50 past images. The common problem of label permutation still occurred in some scenarios. Perhaps the inclusion of a semantic consistency loss, like PixelDA [5] implemented would originate better consistency regarding this subject.

The inclusion of an identity loss in CycleGAN proved helpful, especially in the task of generating photos from paintings. With regular cycle-consistency, the model had too much freedom to manipulate the images. Normally, this would not pose an issue, but if a shortcut is possible the network will most likely take it. This would cause tint shift to occur between images. To correct this, an identity loss maintains color consistency between the original and converted images. This loss was applied to the generator. When an image from the target domain was given as input, then the output is expected to be the same image, encouraging the generator to reproduce the identity function.

Cycle-consistent adversarial domain adaptation (CyCADA) [28] tries to combine all losses from previous attempts to create a robust model. This approach was applied to the task of converting from synthetic to real images. CyCADA comprises four losses: a pixel [80, 5], feature [50], semantic [5] and cycle-consistency losses [107, 40, 103]. The authors concluded that the pixel-level loss, although producing more visually interpretable results and being particularly effective for semantic segmentation tasks, was not adequate for large discrepancies between domains. Applying only the feature-level loss achieves better results than only using the pixel-level loss. These losses coupled together outperform the use of only one of them. The Semantic consistency loss is necessary to maintain correct label prediction, as previously mentioned [5, 107]. In combination with the DA process, a task classifier is jointly trained with source labeled data.

None of the aforementioned methods can be applied to multi-step DA without training more than one GAN model for each domain transfer. Choi et al. [9] addresses this problem and promotes

StarGAN. Taking inspiration from CGANs [54], StarGAN devises multi-domain image-to-image translation while comprising a single generator and discriminator pair. The generator network, in addition to source images, also receives a vector with label information in their corresponding domain. Since domains can be inconsistent when it comes to label definition, a mask vector is also added to make StarGAN focus solely on the known label of a particular dataset. The used losses are also standard compared to previous works: adversarial loss between generator and discriminator, domain classification loss for both generator and discriminator, and an L1 distance for the reconstruction loss. Needless to say, labeled data from all domains is required to train StarGAN. Therefore, this method can only be applied to supervised DA, which is not the general case. However, it is an interesting approach, producing outstanding results and more efficiently than previous methods.

3.6.2.2 Contrastive Domain Mapping

More recently, the authors of CycleGAN [107] proposed a novel approach to unsupervised domain translation using contrastive learning called Contrastive Unpaired Translation (CUT) [62]. CUT is more lightweight and faster than CycleGAN while obtaining better results. This improvement in efficiency comes from solely converting from the source to the target domain and not vice-versa. These improvements allow us to experiment with higher resolution images since hardware constraints are alleviated. In addition, CUT does not use any pre-defined cycle-consistency or semantic losses; it instead uses a contrastive loss based on InfoNCE [60] as a way to maintain structural information between the original and converted images.

CUT uses the same generator and discriminator architectures as CycleGAN, except that it consists only of a single pair for a one-way domain translation. The encoder portion of the generator produces the image embeddings used in the contrastive loss. Contrary to other contrastive learning approaches like SimCLR, CUT does not compare embeddings from different images. It instead divides the input image into patches, encouraging the original and converted patch embeddings in the same location to correlate positively while forcing all the remaining image patches to be located further away in the embedding space. The authors tested sampling negative embeddings from the remaining dataset images, but this approach yielded worse results. Therefore, CUT's implementation of the InfoNCE loss was named PatchNCE loss. CUT also passes the patch embeddings through a 2-layered MLP before computing the contrastive loss, similarly to SimCLR's projection head, which was found to produce more valuable representations.

Furthermore, the identity loss used in CycleGAN is kept in CUT as the authors argue that it regularizes training and prevents model collapse. In CycleGAN, this is achieved using the L1 loss, while in CUT, the PatchNCE loss is used instead. CUT's complete loss function is composed of the GAN loss (equation 2.7) in conjunction with the PatchNCE loss (equation 2.8), as follows:

$$\mathscr{L}_{CUT}(G,D,H,X,Y) = \mathscr{L}_{GAN}(G,D,X,Y) + \lambda_X \mathscr{L}_{PatchNCE}(G,H,X) + \lambda_Y \mathscr{L}_{PatchNCE}(G,H,Y) \quad (3.5)$$

with G as the generator, D as the discriminator, H as the 2-layered MLP, λ_X and λ_Y as weight parameters for each contrastive loss, and X and Y as the source and target domains, respectively.

An interesting side-effect of CUT's method is also documented by the authors. In CUT's horse to zebra translation [62] the authors observed that the zebra training set contained more close-up viewpoints of zebras compared to the horse set. This created a training set imbalance as the percentage of pixels belonging to zebras outweighed the percentage of pixels belonging to horses. Due to this bias towards more zebra pixels, CUT would increase horses' body size to match the zebras' image distribution more adequately. This effect can be seen in figure 3.8.

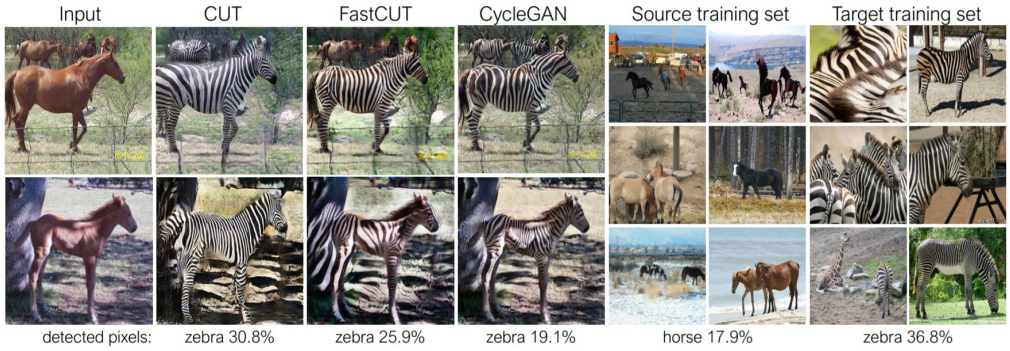


Figure 3.8: CUT's distribution matching phenomenon. We can see that percentage of pixels belonging to the horse/zebra bodies are unbalanced, with zebras usually appearing larger than horses. Due to this, CUT standard approach enlarges the horses' bodies to better fit the zebra's distribution. Comparisons between the authors' other methods' outputs, FastCUT [62] and CycleGAN [107], are also made. Extracted from [62].

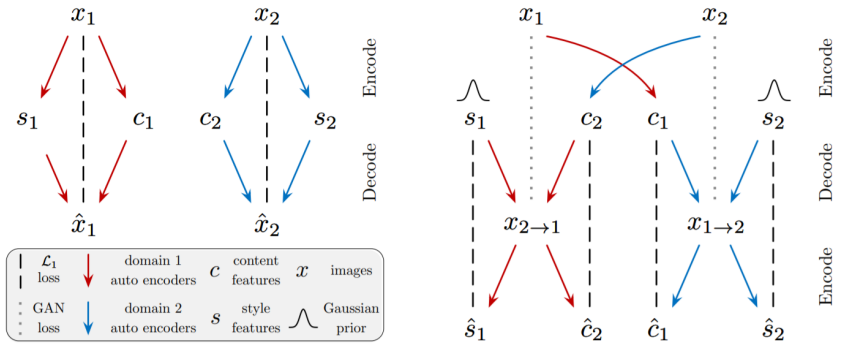
3.6.2.3 Domain Invariant Feature Learning

Murez et al. [55] proposed an image-to-image (I2I) adaptation framework that learns a domain-invariant latent space, from which image reconstruction targeting a specific domain is performed. I2I comprises a total of six losses: classification loss, feature discriminator loss, reconstruction loss, translation loss, cycle loss, and a translation classification loss.

For small domain gaps, feature discrimination in the latent space, reconstruction and translation classification outperform other loss combinations. For larger gaps, the junction of all losses provides better results. I2I outperforms previous adversarial approaches [50, 5, 19, 14, 15] which solely perform domain mapping or invariant feature learning.

Unsupervised Image-to-Image Translation Network (UNIT) [49] and its successor Multimodal Unsupervised Image-to-Image Translation (MUNIT) [31] also try to find an invariant latent space before converting images. UNIT uses dual VAE-GANs [43] with shared weights to achieve this goal. MUNIT is focused on style transfer and separates content and style spaces. This is comparable to DSN [6] which applied the same concept without resorting to GANs. The main contribution MUNIT adds is that it also encourages the reconstruction of the content and style codes in addition

to reconstructing the original image. This can be seen in figure 3.9. Leveraging this separation of shared content and distinct styles, MUNIT manages to outperform UNIT by a significant margin.



(a) Within-domain reconstruction (b) Cross-domain translation

Figure 3.9: Overview of MUNIT reconstruction losses. Extracted from [31].

3.7 Domain Randomization

DR is an important notion, especially when extracting training samples from simulators. These allow us to obtain labeled data and easily change the simulated environments to allow for DR. The main goal of DR is to provide a model with such a wide variety of source data that it generalizes enough to apply its knowledge to target data effectively. By formulating multiple variations of the source data, expanding its image distribution, it is expected that these discrepancies encompass the target domain. Target data would just be another variant that the model was trained to deal with. DR is an unsupervised paradigm and can be achieved with even fewer target data than DA.

Tremblay et al. [87] applied DR to the task of object detection by generating synthetic data to train a DL model. Objects of interest, in this case cars, were randomly placed inside virtual environments alongside unimportant objects, denominated as *flying distractors*. Random textures were applied to all objects and environmental variables like camera position, viewpoint, lights, and background image were altered for each sample. This method proved to obtain better results than using just real data, and subsequent fine-tuning increased the performance gap even more. This is an appealing technique that increases performance by producing more diverse synthetic samples without the need to modify an *off-the-shelf* DL model.

In their work [87], the authors also argued that a pre-trained network on ImageNet [73] reached much higher performance scores, particularly when working with small-sized datasets. A DNN initialized with random weights would drop from 8 to 21 percentage points in AP@0.50 IoU compared to fine-tuning an already trained DNN in their object detection experiments.

Sim-to-Real via Sim-to-Sim [36] introduces a Randomized to-Canonical Adaptation Network (RCAN) and applies DA to improve the performance of a grasping reinforcement learning agent.

Normally, these agents are trained using only synthetic data. RCAN uses generative DA, but instead, it inverts the domain mapping process, converting target data to source data using Pix2Pix [35], a pair-wise image translation network briefly mentioned in section 2.6. By applying randomized textures over synthetic images and converting them to their normal variants, denominated canonical images, the model also learns to transform real images to canonical images. In this manner, the reinforcement learning model, already trained on simulated environments, does not suffer from domain shift from receiving real instead of simulated images. This can also be seen as a form of DR since domain-randomized images are converted to canonical non-randomized versions. Despite assuming the DR process expands to real images [95], this is a promising approach since the model itself does not require any modifications, and the focus is directed to providing the model with consistent observations similar to what was encountered in training.

3.8 Applications and Datasets

Most DA applications are in the field of CV with tasks such as image classification, object detection, image segmentation, and image-to-image translation. As an application example, we can assume a hypothetical problem in the context of autonomous vehicles. The car's sensors and cameras gather data to segment the received images so that the vehicle is aware of its surroundings. During this model's training, a specific dataset of Porto was used. If the self-driving car were to be driven in Lisbon, it would probably perform relatively well. However, let us assume that the model is tested in England instead. The environment is completely distinct; the system would probably drive on the wrong side of the road, not being able to accommodate its surroundings. Obviously, there was a domain shift regarding the input data, whilst the main task of segmentation and driving remained the same. This is what DA aims to improve, by combining different but related sources of data, in this case from multiple cities, we can obtain higher accuracy on the target task.

Nevertheless, areas outside CV can also make use of TL. For example, natural language processing (NLP), a field vastly affected by DL techniques, can also have data shortages, which can be mitigated by TL. Yang et al. [102] tackles the problem of question answering, i.e., correctly answer given questions regarding a specific paragraph. Needless to say that this task requires question-answer pairs for the training process. The authors use adversarial methods to generate new training question-answer pairs and form a unified representation between the model-generated and the human-generated question-answer pairs.

As previously mentioned in section 3.4, the most basic method of DA, fine-tuning, is often used, especially with the ImageNet dataset [73]. As images can vary immensely in their composition and different datasets share relevant information, TL is an essential process to tackle vast and complex scenarios. Additionally, datasets that serve as standard benchmarks for TL or DA problems have more abundance in the CV field. Some commonly used datasets are Modified National Institute of Standards and Technology (MNIST) [46], MNIST-M [15], U.S. Postal Service (USPS) [44] and Street View House Numbers (SVHN) [58], with the various pair permutations of source and target domains between them. These are used for digit classification tasks with images

varying in size, color channels, and environment. One can see how all the aforementioned datasets are related, with a small, albeit existent, domain gap.

More complex and broader sets of data must be used to more effectively determine how one method addresses a certain DA task. The Office dataset [74] is also frequently used to measure model effectiveness on larger domain gaps. This particular dataset is tailored for DA, being used for image classification tasks. It has 31 object classes in three distinct domains: images taken from the Amazon website, real images taken from a DSLR camera, and images taken from a webcam with lower quality and resolution.

Some methods aim to perform image segmentation or object recognition tasks after the DA process, which KITTI [17], SYNTHetic collection of Imagery and Annotations (SYNTHIA) [70], and Cityscapes [11] datasets are widely used for. The KITTI dataset is composed of real-world video sequences of traffic environments. It can be used for video and image understanding tasks, like multi-object tracking and object detection. The SYNTHIA and Cityscapes datasets provide semantic segmentation masks of urban scenes. To create SYNTHIA, the Unity engine[1] was used to render a virtual city and label, with pixel-level accuracy, the extracted frames. Moreover, using a simulated world allows for the same scenes to be rendered in different conditions, such as distinct seasons, lighting, weather, and from multiple viewpoints. In contrast, Cityscapes comprises real-world video captured from 50 different cities, with some samples varying in annotation quality. These coarse segmentation masks are useful for methods that aim to tackle semi-labeled data.

Furthermore, Virtual KITTI [13] is a synthetic variant of the KITTI dataset. It also takes advantage of the Unity engine and a novel real-to-virtual cloning method to simulate and automatically annotate video segments, using the original KITTI samples as a point of origin. The GTA5 dataset [67] is also composed of synthetic images obtained from the Grand Theft Auto V video game. This is an efficient way of reusing rendered environments to obtain semantic label maps automatically. This is done by intercepting data passed to the graphics hardware, parsing it, and applying custom shaders to annotate large portions of the rendered images. Some examples of domain mapping using these datasets can be seen in figure 3.10.

(a) GTA5 (b) GTA5 → Cityscapes (c) CityScapes (d) CityScapes → GTA5

Figure 3.10: Examples of image translation between the GTA5 and Cityscapes datasets using the CyCADA method [28]. (a) Original GTA5 dataset images; (b) Corresponding conversion from GTA5 to Cityscapes; (c) Original Cityscapes dataset images; (d) Corresponding conversion from Cityscapes to GTA5.

[1] https://unity3d.com/

As can be seen, several datasets have already demonstrated the usefulness of simulations for labeled data generation and augmentation. Ultimately, any dataset pair, whether real or synthetic, deemed to have related knowledge concerning some particular task, can be used for DA.

3.9 Summary and Discussion

In this section, multiple approaches to DA were enumerated and explained. A summary from the reviewed papers can be seen in table 3.3. These approaches try to find domain-invariant features shared by both domains and perform the final task based on these features. This can be done directly by fine-tuning existing networks or by domain mapping across datasets, consisting of converting one image from one domain to another while maintaining its characteristics and annotations. One other approach that can take full advantage of simulators is DR, by broadening one domain through multiple variations on original images. This approach expects that these abnormalities will encompass the other domain. One model trained on the skewed images from one domain is expected to generalize to the other.

There are three main approaches to achieving domain-invariant models: divergence measures, adversarial methods, and reconstruction or cycle-consistency losses. The criteria on which divergence methods are based on are adopted from traditional ML methods, and by themselves, do not achieve performance levels as the other alternatives. However, they are modular enough to be used in other networks to improve the DA process further. Adversarial approaches do not directly measure the domain gap. Instead, intuitive adversarial losses that lead to domain-invariance are applied. Reconstruction methods force the image features to be preserved by restoring the original images during training. This is accomplished through a cycle-consistency loss. This loss can be applied to the same domain by reconstructing an image from a latent space or between different domains by converting an image to the other domain and back again.

When addressing adversarial methods, other losses were mentioned: feature, pixel-level, and semantic-level losses. These losses might be necessary depending on how large the domain-shift is. In [28], the authors concluded that pixel-level losses were sufficient for smaller domain gaps since they produce realistic images. However, this can be limiting when the domains vary significantly. The existence of a semantic or feature loss greatly increases the chances of success for more considerable domain differences. These losses help to maintain important image characteristics in accordance with their annotations.

Adversarial or reconstruction methods, mostly based on GANs, offer the most general and reliable applications. Domain mapping allows separating the DA problem from the final task. This means that if the task should change, it is only required to train a model on our new task, and the DA process needs not be repeated. Also, unsupervised DA is a much more prevalent scenario, which these methods aim to address. DR is a mentioned alternative, and since GANs demand high quantities of training data [97], the usage of DR to generate more samples would allow us to perform DA using adversarial methods with less apprehension.

Table 3.3: Summary of the discussed DA methods (domain-invariant feature learning [DI], domain mapping [DM], normalization [N], ensemble [En]). Adapted and extended upon [97].

Name	Method	Loss functions					Adversarial loss		Generator	Shared weights
		Distance	Diff.	Cycle	Sem.	Task	Feature	Pixel		
Tzent et al. [88]	DI					✓	✓			✓
DDC [90]	DI	MMD				✓				✓
Deep CORAL [83]	DI	CORAL				✓				✓
CAN [37]	DI	CCD				✓				not BN layers
AdaBN [48]	N									not BN layers
Domain guided dropout [99]	En					✓				✓
CoGAN [50]	DM					✓	✓		✓	semi-shared
SimGAN [80, 78]	DM							✓	✓	
PixelDA [5]	DM				✓	✓		✓	✓	
DANN [15, 14]	DI					✓	✓			✓
ADDA [89]	DI					✓	✓			
Sim2Real [64]	DI,DM	MMD		✓						semi-shared
DSN [6]	DI	(tried MMD)	✓	✓		✓	✓			semi-shared
DRCN [19]	DI			✓		✓				✓
UNIT [49]	DI,DM			✓					✓	semi-shared
MUNIT [31]	DI,DM			✓	✓			✓	✓	
I2I Adapt [56]	DI,DM			✓	✓	✓	✓	✓	✓	semi-shared
DiscoGAN [40]	DM			✓				✓	✓	
DualGAN [103]	DM			✓				✓	✓	
CycleGAN [107]	DM			✓				✓	✓	
CUT [62]	DM				✓				✓	
StarGAN [9]	DM			✓				✓	✓	✓
CyCADA [28]	DI,DM			✓	✓	✓	✓	✓	✓	

Chapter 4

Methods

This chapter defines this work's proof-of-concept and provides an overview of the datasets and automatic quality inspection system [68], on which this work extends upon. This is followed by analyzing the distinct approaches to our DA problem and the subsequently chosen solution. In the later sections all the accomplished experiments made during the course of this work are described along with the thought process behind them. During this work's experiments, we focus on a DA technique called domain translation or domain conversion, or even domain mapping. All these notions are interchangeable, referring to the same process of obtaining real counterparts of simulated images or vice-versa. All the experiments were run on an RTX 2080 Ti graphics card.

4.1 Previous work: Automatic quality inspection

To better understand the usefulness of using simulated and real images in the training process of a DL model, this thesis aims to improve upon previous work done in [68]. The authors propose a system with the final goal of automating quality control at the end of a vehicle's production line. Quality control is normally a manual process where factory workers resort to paper conformity lists, which provide a written description of the parts that compose the respective vehicle that is used to verify each product's final state. The system showed in figure 4.1a is composed of multiple modules. We aim to improve the detection module, shown in figure 4.1b, responsible for analyzing vehicle images and outputting a list of parts that compose the corresponding vehicle. This module is divided into two stages: an object detection and segmentation stage followed by a content-based image retrieval (CBIR) stage, explained in detail in sections 4.1.2 and 4.1.3, respectively.

Real labeled data was not available for the implementation of the inspection pipeline. Therefore, a simulated dataset was created and used in its stead. Despite being trained with only simulated images, the system obtained promising results when applied to real-world data.

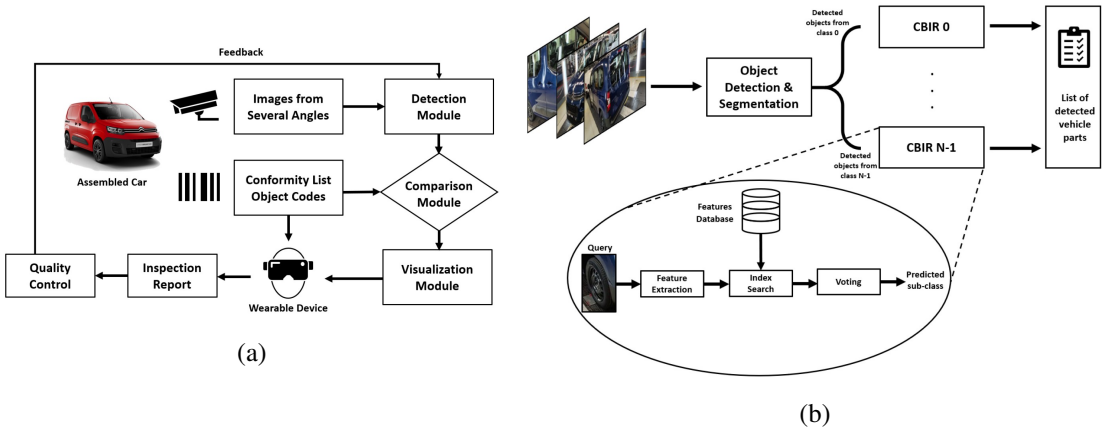


Figure 4.1: (a) Diagram of the complete support tool proposed in [68]; (b) Diagram of the detection module of the automatic quality inspection system, composed of an object detection and segmentation sub-module and several CBIR sub-modules. Both images were extracted from [68].

4.1.1 Datasets

At our disposal there are two datasets: one composed of fully annotated simulated images and one consisting of weakly annotated real vehicle images. The latter was solely used for evaluating the system's real-world performance in [68], taking no part in the training process of the detection module.

The simulated dataset was obtained from a 3D vehicle generator [68] developed using the Unity engine[1] and is fully labeled with instance and segmentation annotations. In total, 340 vehicle configurations were gathered by looping through all possible combinations of brands, models, versions, colors, and customer personalization options. For the dataset creation, 100 synthetic images were collected for training and 20 for validation from each vehicle configuration, resulting in 34000 training images and 6800 validation images, with 363719 and 72430 objects, respectively. The collected images are 512 by 512 pixels RGB frames having multiple zoom and rotation angles applied to them by continuously rotating the car during the simulation. In our experiments, we use the same dataset, with the same instance-level annotations, but with two different backgrounds, the Unity Engine default background and a factory environment background. Some samples of the datasets can be seen in figure 4.2.

The real dataset is considerably smaller than the synthetic one, with a total of 100 images. It is composed of 10 vehicle configurations, each one with 10 real images. Furthermore, the real images have image-level annotations, meaning that we have access to a list of the vehicle parts present in each image but not their bounding-boxes. Therefore, to assess the system's performance on real-world data, we consider a multi-label multi-class classification problem.

[1] https://unity3d.com/

(a) (b) (c)

Figure 4.2: (a) Real dataset samples (weakly annotated with image-level labels); (b) & (c) Simulated dataset samples with factory and unity backgrounds respectively (obtained from simulator with instance-level labels).

4.1.2 Object-detection stage

This stage performs a macro detection of vehicle parts in a given input image. By macro detection we mean that only the location and part codes of each instance are inferred, leaving the task of determining the model, brand, and material codes to the CBIR stage.

In practice, Detectron2 [98] was used to accomplish this stage. Using pre-trained weights of a ResNet-101 Feature Pyramid Network (FPN) backbone trained on the COCO dataset, a Detectron2 model was fine-tuned on the simulated datasets, with better results obtained using the factory background.

4.1.3 CBIR stage

Vehicle configurations are not permanent and can suffer modifications by removing or adding new stock-keeping units (SKUs) of the manufacturing parts. To address this issue, multiple CBIR modules, one for each macro class detected by the Detectron2 model, are in place to facilitate the modification of SKUs during the manufacturing process. The macro-categories of each part detected by the Detectron2 model are expected not to undergo continuous revisions; hence, if the system requires an update, one only needs to update the CBIR modules with the examples of the new parts.

In practice, a search index is built using embeddings obtained from image crops of the simulated vehicles parts. Hence, for this stage, bounding box annotations for each car part are required so that we can extract the correct embeddings. One index is in place for each macro class, containing the feature embeddings of the specific part codes of said macro class. A ResNeXt-50 32x4d pre-trained on the Imagenet dataset is used as a feature extractor to form the embedding dataset. When receiving the macro codes from Detectron2, each detected instance is directed to the CBIR module of the corresponding class. According to its bounding box, a cropped image is produced, and an embedding vector is obtained from the feature extractor network. Afterward, the embedding's nearest neighbor is voted using the L2 Euclidean distance, obtaining the final code.

4.2 Method overview

4.2.1 Goal

As annotations for real-world vehicles are not at our disposal we approached this scenario as a DA problem, where simulated and real images must be combined to retrieve useful information from both data sources and increase the detection system's overall performance. It proved to be a great opportunity since simulated and real datasets had already been collected, and the generation simulator that produced the former provided us with easy dataset manipulation.

In the following section we consider all stages that compose the detection system to determine which one we should improve, to better close the gap between simulated and real domains.

4.2.1.1 Detection system's bottleneck

When analyzing the previous work's findings [68] we reach the conclusion that the major bottleneck in the detection module is the object detection and segmentation stage, performed by Detectron2. When evaluating the CBIR stage independently from the detection system, as we can see in table 4.1 extracted from the previous work [68], the authors obtain very good results. This shows Detectron2 is at fault for the system's relatively poor performance on real data (table 4.2) compared to simulated data.

Table 4.1: CBIR results on simulated data: object crops from the ground-truth vs output of the detection module. Retrieved from [68].

Task	Metrics	Default Unity Background	Factory Background
Ground-truth + CBIR	Accuracy	0.990	0.989
	Precision	0.981	0.979
	Recall	0.947	0.940
	F1-score	0.961	0.954
	MCC	0.990	0.989
	Cohen's Kappa	0.990	0.989
Prediction + CBIR	Accuracy	0.964	0.975
	Precision	0.951	0.960
	Recall	0.913	0.92
	F1-score	0.927	0.935
	MCC	0.963	0.975
	Cohen's Kappa	0.963	0.975

Table 4.2: Results on real data, reported as a multi-class multi-label classification problem. Note that for the Hamming loss, lower is better. Retrieved from [68].

Task	Metrics	Default Unity Background	Factory Background
Example-based	Accuracy	0.247	**0.280**
	Precision	0.296	**0.386**
	Recall	**0.594**	0.517
	F1-score	0.395	**0.442**
	Hamming Loss	0.478	**0.343**
Label-based	Accuracy	0.522	**0.657**
	Precision	0.256 / 0.299	**0.292 / 0.390**
	Recall	**0.516 / 0.599**	0.438 / 0.521
	F1-score	0.304 / 0.399	**0.307 / 0.446**

Furthermore, without annotations, there are no real data performance results for the intermediary Detectron2 stage, where macro categories are detected. Only validation metrics for the simulated domain were calculated, as shown in table 4.3. We address this by not only calculating the system's global performance, but also computing label and example-based metrics for Detectron2's performance on real data, to further assess our approach's effectiveness. This means we have both performance results for the intermediary task, of detecting macro categories, and for the final task of obtaining the complete parts' codes. The used metrics are explained in more detail in section 4.2.4.

Table 4.3: Detection and instance segmentation results on simulated data. The vehicle parts' reference codes corresponding to the best and worst classes are shown in-between parenthesis. Retrieved from [68].

	Metrics	Default Unity Background	Factory Background
Detection	mAP	86.36	86.76
	mAP@50	98.16	98.49
	best class AP	98.11 (S.SpSm)	97.34 (S.SpSm)
	worst class AP	61.42 (F.LiCv)	54.18 (F.LiCv)
Segmentation	mAP	70.08	71.54
	mAP@50	87.11	87.84
	best class AP	97.57 (B.Wn)	96.94 (B.SLiC)
	worst class AP	0 (SBL.DrCv)	0 (SBL.DrCv)

4.2.2 Solution

As seen in the reviewed literature, there are several ways to approach the domain gap issue. Methods to deal with the discrepancies between simulated and real images and combine both sources of data may yield better results with less effort. This specific scenario can be considered a one-step homogeneous DA process. The two domains are expected to have the same feature spaces, and the conversion from the simulated to the real-word domain can be made in a single transition step.

As an overview of the DA landscape, three approaches stand out: divergence-based methods (see section 3.4), adversarial-based methods (see section 3.5), and reconstruction-based methods (see section 3.6). To segregate the DA task from the object detection and segmentation tasks, we

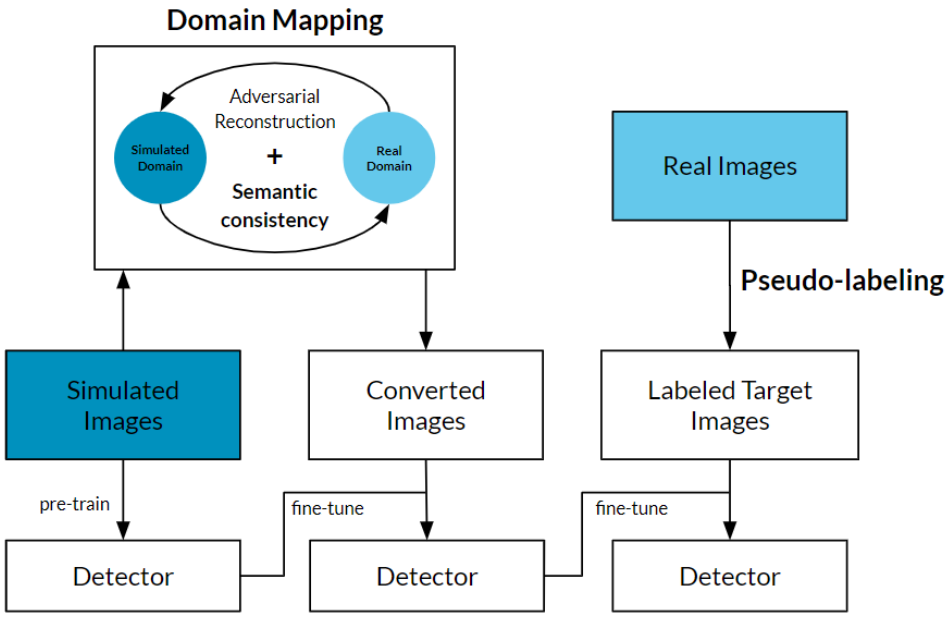


Figure 4.3: The adopted approach of progressively fine-tuning a detector model. A domain mapping model is in place to convert simulated images into their real counterparts. Adapted from [33].

can discard most of the divergence-based methods since these achieve domain-invariance through a set of criteria that must be trained jointly with the final task. In addition, these methods are more generally applied to standard classification tasks, whereas we want to combine DA and the tasks of object detection and segmentation.

Inoue et al. [33] applied DA to the task of weakly-supervised object detection. The authors used CycleGAN [107] and pseudo-labeling to progressively fine-tune an already existent object detector model. By taking advantage of the CycleGAN, domain-translated images from the source to the target domain are produced, which serve as additional data for the fine-tuning process. This step helps to increase our available labeled data, and the converted images can be reused and applied to other tasks, as we are essentially building a new converted dataset with the same features as the original one, albeit with a different style. Additionally, pseudo-labeling is used to obtain weak labels for the target samples.

Pseudo-labeling allows the model to bridge the gap between simulated and real domains even further. The process involves generating predictions for real images using the fine-tuned Detectron2 and consider them as weak annotations. By training the same Detectron2 on these real images with pseudo-generated annotations we create a more robust model as clusters for the real distribution are further established.

This method [33] applies to our specific vehicle parts detection problem since the real-world samples, representing the target domain, do not contain bounding box annotations, only the vehicle parts present in the image. In summary, this approach, shown in figure 4.3, fine-tunes the model

three times, first with the labeled source data, then with the source samples converted to the target domain, and lastly with the pseudo-labeled target samples.

We applied this method to our DA problem and two domain conversion methods were experimented with: CycleGAN [107] and CUT [62].

4.2.3 Challenges

The domain mapping approaches are GAN-based models. GANs are notorious for their hyperparameter sensitivity, mode collapse issues, and high chances of failing to converge compared to other models. The models' authors already perform most of this tuning process during their research; however, some additional changes were required to produce better results in our experiments (sections 4.3.3 and 4.3.4).

Another setback encountered was the degradation of features due to poorly executed image translations from simulated to real domains. This was mitigated by applying a semantic consistency loss, detailed in section 4.3.3.2, to the CycleGAN, and by increasing CUT's contrastive loss's weight, encouraging feature preservation among the converted images.

4.2.4 System Evaluation

As previously mentioned, the available real images are weakly annotated with only a list of vehicle parts, without bounding boxes, present in each sample. Since a single vehicle is composed of multiple parts, we consider this a multi-label multi-class classification problem. As such, we can evaluate this problem label-wise and sample-wise.

Label-wise metrics are computed for all labels across all samples. The obtained results represent how performant the system is for each specific part's label. Moreover, we can calculate label-wise metrics in different manners:

- **Macro**: macro values are obtained by calculating metrics for each label and finding their unweighted mean.
- **Micro**: micro values are obtained by computing the desired metrics on the global predictions' results. This makes it the most rigid metric of all three and was the one for which we guided ourselves for most of our experiments.
- **Weighted**: weighted values are macro values with label unbalance taken into account.

Sample-wise metrics provide us with metrics for each evaluated sample, with all sample values then averaged to obtain the final sample-wise metric.

In our scenario, the accuracy metric does not correctly measure our model's performance since it takes into account true negative predictions. As a vehicle incorporates only a few of all the parts, true negatives can have a strong influence on the model's accuracy. This implies that, compared to the baseline, a model may have a lower F1-score with higher accuracy, meaning that among the different model predictions, it managed to increase the true negatives at the cost of some true

positives. We do not want to reward these occurrences; hence, we focus on the F1-score since it makes use of the precision and recall metrics.

For the DA experiments, regarding the domain translation methods, as they are GAN based we will be analyzing specific CycleGAN and CUT losses to assess both models' stability, as well as using the ubiquitous FID which compares the generated and real image distributions, to obtain a measurement of the converted images' quality and their resemblance to the target domain. The FID score, on all experiments, was obtained by adapting Seitzer's implementation [77]. Furthermore, the FID scores are computed using the available training sets since a minimum quantity of 2048 samples is required to calculate such score, which the validation sets cannot provide.

Lastly, Detectron2's performance will be measured individually as a single stage in the process, as well as its effects in the entire module. To fully evaluate this approach's effectiveness, the modified detection module's performance will be compared to its performance when using exclusively simulated images.

4.3 Domain Translation

This section explains in detail the domain translation process. Firstly, the datasets used in all experiments with both CycleGAN and CUT are introduced. Then, we detail all translation experiments conducted with the primary goal of finding the model that provides the most realistic results while preserving the original annotations.

4.3.1 Datasets

For the task of domain translation, both CycleGAN and CUT require source and target datasets. Since CycleGAN depends on a cycle-consistency loss, domain mapping is made in both directions. However, our goal is to obtain more realistic translations of the simulated images; therefore, we will focus on the simulated to real domain conversion.

For the simulated domain, we created two datasets. We used the same 3D vehicle generator used for the original inspection system [68], and maintained the same 512 by 512 pixels RGB format. These datasets are composed of the same vehicles with identical annotations, with the only difference being the background image used. One dataset tries to replicate a factory scenario while the other retains Unity's default background. Modifying the simulator to create an additional vehicle mask for each image allows us to customize the background to our liking without corrupting any annotations. This enables us to more precisely determine the images' background influence on the conversion process. The simulated training and validation sets are composed of 3184 and 762 images, respectively. This decrease in dataset size was to resemble the size of the corresponding real dataset. Due to data limitations we could not gather sufficient real vehicles' frames to reach the simulated datasets' original size. Hence, to balance both domains and prevent the domain mapping models to skew towards a particular dataset we compromised on the simulated dataset size.

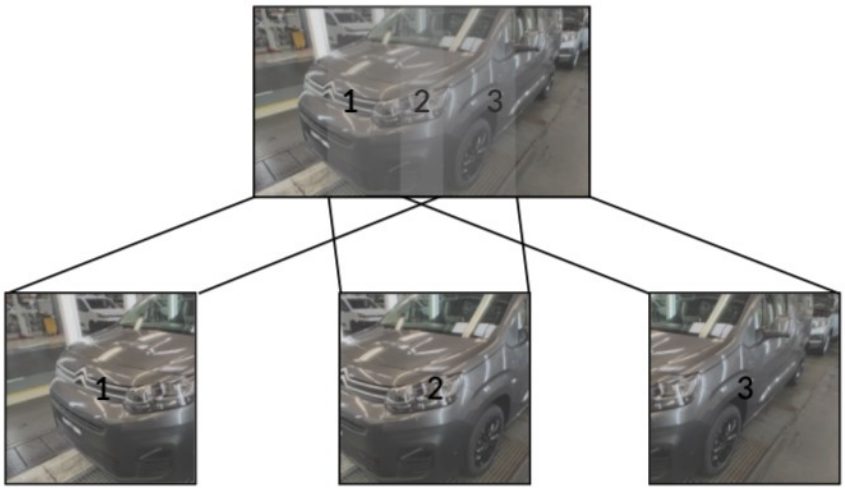


Figure 4.4: Process of cropping real frames to produce dataset images.

The real dataset was assembled by extracting frames from available vehicle videos in a factory setting. Subsequently, all frames were divided into crops, providing 3 final images for each extracted frame (figure 4.4). This is needed because during the training process, all images from both domains must be in the same shape and size. By default, this is 256 by 256 pixels on CycleGAN and CUT. Had we used the original 1280 by 720 pixels real frames, we would have to either produce random crops or alter the images' aspect ratio. Neither of these options is ideal since random cropping could omit too much vehicle information, and altering the aspect ratio would distort the real images, making them difficult to relate to the non-distorted simulated images. Nevertheless, we did not intend to limit the amount of variance in the data, and so, for the training set, we divided an image into 3 overlapping square crops with a side length equal to the image's height. In this manner, the obtained crops cover the entire original image and can be resized to the same size as the simulated images. For the validation set, a single square crop was randomly sampled from each frame. Ultimately, 6855 real training and 762 real validation 512 by 512 pixels images were collected.

Table 4.4: Number of images for the different splits of both simulated and real datasets.

Set	**Simulated (Factory & Unity)**	**Real**
Training	3184 images	6855 images
Validation	762 images	762 images

4.3.2 Differentiable Augmentation

Differentiable Augmentation (DiffAugment) [106] is a method that lessens the need for large datasets to train a GAN model successfully. The authors demonstrate their technique applied to StyleGAN2 [39]. DiffAugment can prevent mode collapse, when using limited amounts of

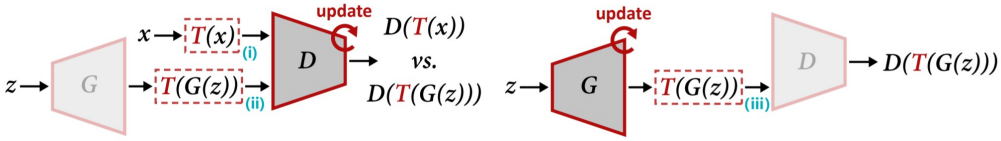


Figure 4.5: Overview of DiffAugment for updating D (left) and G (right). DiffAugment applies the augmentation T to the discriminator's input data distributions and allows its gradients to back-propagate to the generator network since they are differentiable.

data [106], and achieve a near-identical FID score with only 20% of training data compared to the standard StyleGAN2 with 100% training data.

The main issue one has to deal with when tackling GAN training with limited data is discriminator overfitting. The absence of diversity in the training set allows the discriminator to guide its predictions by memorizing the given images instead of extracting valuable features covering the real data distribution. Being overfitted to the training data, the discriminator performs exceptionally well when asked to distinguish between real and fake images. This, in turn, results in near zero-valued gradients flowing to the generator network halting its learning process and ultimately ending the zero-sum non-cooperative game, collapsing the GAN.

To prevent the discriminator from memorizing training samples, diversifying the data without requiring more images is necessary. The data augmentation typically used in standard classification cannot be directly applied to GANs since this worsens the model's performance rather than improving it [106]. Applying augmentation only to the real images given to the discriminator would defeat the GAN's purpose since the discriminator would not learn the real data distribution, but instead an augmented one. This would cause the generator to mimic data augmentations in an attempt to replicate the distribution the discriminator learned.

Another alternative would be to apply augmentations to both real and generated images provided to the discriminator. However, this would not work as intended either since the generator and discriminator would model 2 different data distributions. The discriminator would learn from augmented data, and the generator would try to create realistic images with meaningless gradients provided by a discriminator that never sees the real distribution.

DiffAugment applies differentiable image augmentations to both real and generator-made images propagating gradients through those augmentations for both generator and discriminator parameters update cycles, as shown in figure 4.5. In this manner, the discriminator training is improved while the generator does not suffer from the different augmented distribution since the received gradients also flow through the applied augmentations. Among the experimented transformations, color, translation, and cutout augmentations are the most notable, providing more favorable results. The authors denominated the removal of a random patch of the image as cutout.

DiffAugment is a simple method that can be applied to almost every GAN model since, at its core, it is a data-augmentation method for GANs. Hence, we decided to apply this technique to

CycleGAN and CUT to observe its effects on domain translation. For all experiments where DiffAugment was used, all recommended augmentations (color, translation and cutout) were applied.

4.3.3 CycleGAN

We adopted an iterative approach to decide the best combination of possible modifications. Thus, we determined which dataset background provided the best results, followed by assessing DiffAugment's influence in the training process. Finally, we applied a novel semantic consistency loss by using the previous Detectron2 model.

Besides the change explained in section 4.3.3.1, all CycleGAN experiments were run for 100 epochs and with an identity loss weight of 1.0 instead of the default 0.5. This change prevents unnecessary changes in the cars' color. The remaining hyperparameters were unaltered according to the original paper [107]. All datasets were resized to 256 by 256 pixels, the default input size of CycleGAN.

4.3.3.1 Checkerboard artifacts

This was a simple model modification which was documented in the CycleGAN author's GitHub repository and also by Odena et al. [59].

CycleGAN adopts a ResNet-like generator architecture. The generator encodes the given images by diminishing kernel sizes and increasing its channels, followed by the opposite operation to obtain the final 256 by 256 3-channeled (RGB) image. The decoding portion is composed of deconvolution operations which were found to produce checkerboard artifacts in the final images [59]. This happens when the kernel size is not divisible by its stride, causing different inputs to affect the same output and create unevenness in the hidden representations. Despite being composed of multiple stacked deconvolutional layers, the network may struggle to learn how to deal with these occurrences and continue to produce artifacts in the final images. For this reason, the deconvolutional layers were replaced by an upsample layer using bi-linear interpolation followed by a standard convolutional layer. This discourages high-frequency artifacts because it produces more evenly weighted hidden representations.

In practice, all generator deconvolutional layers with kernel size 3x3, stride 2, padding 1, and output padding 1 were replaced by a bi-linear upsample layer that increases the input's size by a factor of 2, followed by a convolutional layer with kernel size 3x3, stride 1 and padding 1. The padding is added by a reflection layer that reflects the input's boundary as padding. Both these implementations have the same input and output sizes, and permuting them does not require further changes to other generator sections.

4.3.3.2 Semantic consistency

The main challenge of domain translation is to retain the original images' features in the converted ones. CycleGAN is known for preserving structural information during the domain translation, mostly adapting textures and colors in the resulting images [107, 33]. However, in some cases,

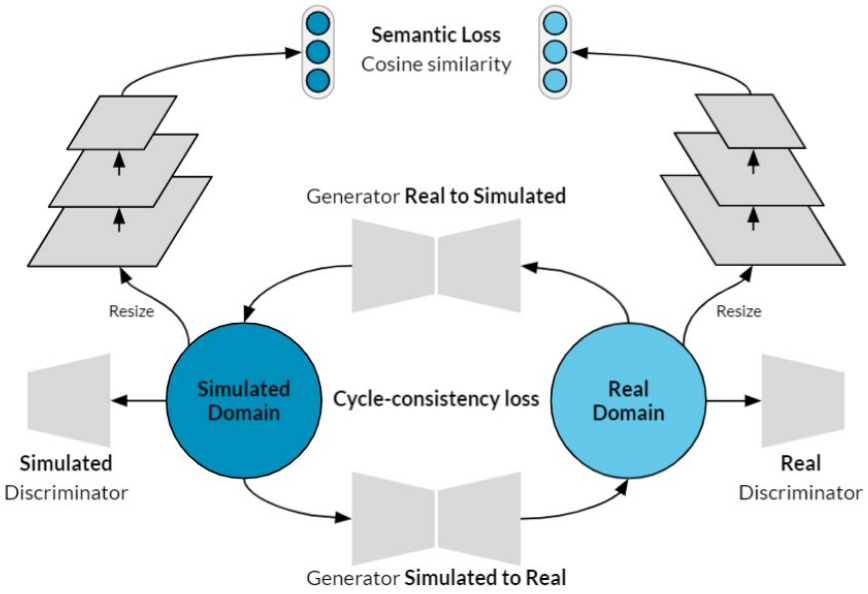


Figure 4.6: Diagram of the modified CycleGAN with semantic consistency. In addition to the standard CycleGAN's generators and discriminators, we introduce a semantic consistency loss using Detectron2's FPN backbone. The original and converted images are both transformed to the expected input format for the backbone, with two embeddings being obtained, one from the original image and one from the converted one. The semantic consistency is employed by the cosine similarity loss between both embeddings.

the model would revamp most of the simulated image, which would lead to severe information loss. To accommodate for these cases and to further improve other instances, a semantic loss was proposed. Its objective is to guide the generator training into preserving the embeddings of the original image during conversion.

In practice, this was achieved by re-utilizing the pre-trained Detectron2's backbone on simulated images. The generator's inputs were passed through the pre-trained backbone, obtaining the input embeddings. Afterward, the corresponding converted images' embeddings were obtained in the same fashion, and we computed the cosine similarity between both embeddings (figure 4.6). The pre-trained backbone was expecting 512 by 512 pixels sized inputs, while CycleGAN worked with 256 by 256 pixels images. Therefore, all generator's input and outputs were properly converted to the expected backbone format during the training process. The backbone's parameters were not optimized during CycleGAN's training; it simply served to regularize the generator's outputs.

Detectron2's backbone is a FPN, meaning it has several output layers L from which RoIs are extracted. We experimented with which layers to use to apply the semantic consistency loss: only on the last layer or on all layers. More precisely, the backbone has 5 output layers (p2, p3, p4, p5 and p6). From p2 to p6 the embeddings are iteratively encoded, diminishing in size as shown in table 4.5, meaning layer p2 holds more detail information while layer p6 has a higher-level image

Table 4.5: Detectron2's backbone output layers' details.

Layer	Output size	Output channels	Flattened output size
p2	128x128	256	4,194,304
p3	64x64	256	1,048,576
p4	32x32	256	262,144
p5	16x16	256	65,536
p6	8x8	256	16,384

representation.

Formally, the new semantic consistency loss for a single input image x being converted by the generator G is as follows:

$$\ell_{sem}(G,B,l,x) = \frac{B_l(t(x)) \cdot B_l(t(G(x)))}{\|B_l(t(x))\|_2 \times \|B_l(t(G(x)))\|_2} \tag{4.1}$$

where t represents the transformations needed to obtain the expected input format for the backbone B, responsible for embedding the input at a specific layer l. All embeddings generated by Detectron2 are flattened before applying the semantic consistency loss.

The semantic loss was applied to both domains expecting helpful embeddings to be generated in both directions:

$$\mathscr{L}_{sem}(G_1,G_2,B) = \mathbb{E}_{x\sim P_{data}(x)} \sum_{l\in L} \ell_{sem}(G_1,B,l,x) + \mathbb{E}_{y\sim P_{data}(y)} \sum_{l\in L} \ell_{sem}(G_2,B,l,y) \tag{4.2}$$

where G_1 and G_2 represent the CycleGAN's generators, one for each domain conversion, x being the simulated domain samples and y the real domain samples. This is based on the promising results obtained by Detectron2 when applied to real data, despite being trained with only simulated images [68]. The new global loss can be formulated as such (all our experiments had $\lambda_{sem} = 1.0$, the same weight as the cycle-consistency and identity losses):

$$\mathscr{L}_{CycleGAN-Semantic}(G_1,G_2,D_X,D_Y,B) = \mathscr{L}_{CycleGAN}(G_1,G_2,D_X,D_Y) \tag{4.3}$$

$$+\lambda_{sem}\mathscr{L}_{sem}(G_1,G_2,B) \tag{4.4}$$

4.3.3.3 Real to simulated detection

We also experiment with converting the real images to their simulated counterparts using CycleGAN and evaluating them on the original Detectron2 trained on only simulated data. This is a small experiment which can only be done with CycleGAN since it can translate images in both directions, contrarily to CUT. Ultimately, this serves to analyze CycleGAN's conversion method in the opposite direction from the real to the simulated domain.

4.3.4 CUT

CUT's novel attempt at unpaired domain translation stated that it obtained better results than CycleGAN while being more memory and time-efficient. Furthermore, it achieved this by implementing a form of contrastive loss between image patches. To experiment with other domain translation methods we applied CUT to our simulated-to-real conversion scenario with the previously mentioned DiffAugment's improvements to it.

Besides the weight of the PatchNCE loss, all CUT's default settings were left unchanged according to its original paper [62]. We used the dataset with the factory background for all CUT runs, first with 256 by 256 pixels resized images and increasing to 512 by 512 pixels for the final experiments. This was possible due to CUT's lower hardware requirements since it does not depend on the opposite domain translation to apply a cycle-consistency loss as CycleGAN does.

Furthermore, for the higher resolution runs, we increased the weight of the PatchNCE loss to obtain more conservative translations. Two weights were tested, $\lambda_{PatchNCE} = 2.0$ and $\lambda_{PatchNCE} = 5.0$. A weight of 5.0 could prove to be too restrictive; hence, a lower weight value of 2.0 (but higher than the default 1.0) was also considered.

4.4 Detector Fine-tuning

To choose which conversion method to use to fine-tune our Detectron2 we decided to convert a subset of the simulated dataset and determine the original Detectron2's performance on it. This subset was composed of 1134 images and had the same class distribution as the complete simulated dataset.

Since Detectron2 was previously trained only on simulated data, it is expected not to generalize well to translated images. Nevertheless, considering that the translation models managed to preserve the original annotations, we argue that the model should be able to extract useful information to some extent and produce a few valid predictions. The translation models that would provide us with the best validation accuracy would be chosen to perform fine-tuning with.

Furthermore, in the following detailed experiments, we introduce a novel post-processing step which we denominated as global NMS. This prunes Detectron2's overlapping bounding boxes, which have an IoU value above 50%, independent of their classes. As such, we also evaluate and compare the system's performance with and without applying global NMS (section 5.2.2).

We first fine-tune the Detectron2 on the converted images for 10 epochs with the same hyperparameters as in Detectron2's initial training on the simulated domain. The epoch with the lowest validation loss was used to verify its performance on real data. The initial fine-tuning results are discussed in section 5.2.1.

For the second and final Detectron2 fine-tuning process using pseudo-labels we generated predictions for real images using the previously fine-tuned Detectron2, and considered them as weak annotations. We fine-tuned for 10 epochs and re-generated pseudo-labels at the beginning of each epoch using the previous epoch's model. For the pseudo-label generation, we also applied global

NMS to Detectron2's predictions, meaning that we only consider as pseudo-labels the predictions that prevail after applying the global NMS. This will discourage the appearance of multiple bounding boxes in the same location.

Furthermore, we also explored a different learning rate of 0.005 for this last Detectron2 fine-tuning, half of the initial learning rate which was used to train the Detectron2 on simulated images. The final fine-tuning results are approached in section 5.2.3.

Chapter 5

Results and Discussion

This chapter is dedicated to the experiments' results and subsequent discussion. The conversions from simulated to real images of the selected two domain mapping methods are compared, along with their corresponding effects on the fine-tuning process of the Detectron2 model. The domain translation models' stability is determined by their training losses, whereas their conversion quality is determined by the corresponding FID scores. The performance of the multi-label multi-class classification problem is measured primarily by the F1-score. As explained in section 4.2.4 this is the most accurate metric for our vehicle parts detection task. In the presented results tables, for our label-based metrics, one can observe the macro value followed by the micro value of each metric.

5.1 Domain Translation

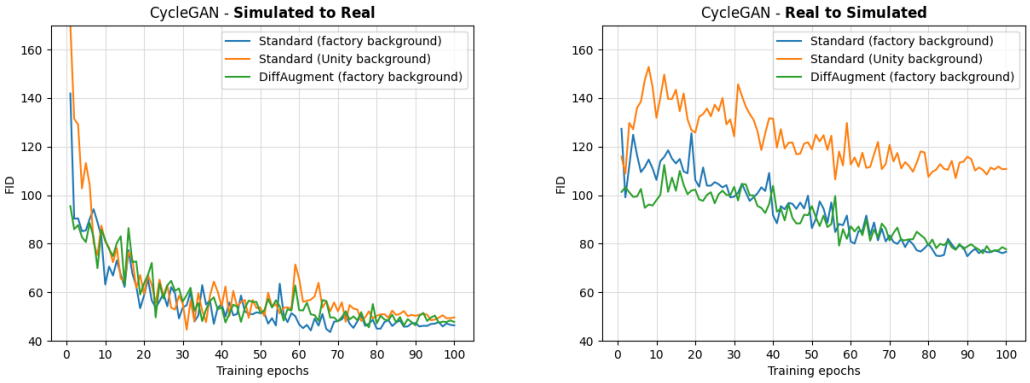
In this section we discuss both domain translation methods experimented with, along with comparisons between each one's conversion ability.

5.1.1 CycleGAN

5.1.1.1 Factory vs Unity background

Both standard CycleGANs with different datasets reached stability in converting from the simulated to real domain. As we can see in figure 5.1 both GAN's FID scores converge and become stable during the last training epochs, with the factory background dataset obtaining a slightly lower value. However, from the real to the simulated domain, Unity's default background got a considerably worse FID score. The generator struggled to completely overhaul the real images' environment to replicate Unity's background. This does not seem to occur with the factory background dataset, probably due to just having to make slight modifications to the factory environment to produce a simulated version of it.

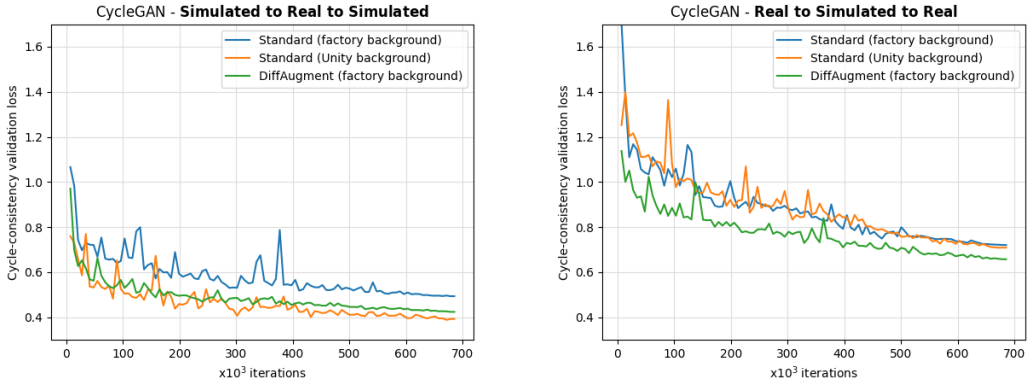
However, the cycle-consistency loss for the simulated domain, i.e. converting from simulated to real and back to the simulated domain, was lower using Unity's background (figure 5.2a). Meanwhile, the cycle-consistency loss for the real domain, i.e., converting from real to simulated



(a) (b)

Figure 5.1: (a) FID score on training set during simulated to real translation; (b) FID score on training set during real to simulated translation.

and back to the real domain, behaved equally in both datasets (figure 5.2b). The Unity dataset's lower cycle-consistency loss did not manifest in the obtained FID scores, demonstrating that measuring a GAN's performance based on its network losses may not be enough to evaluate its actual behaviour.



(a) (b)

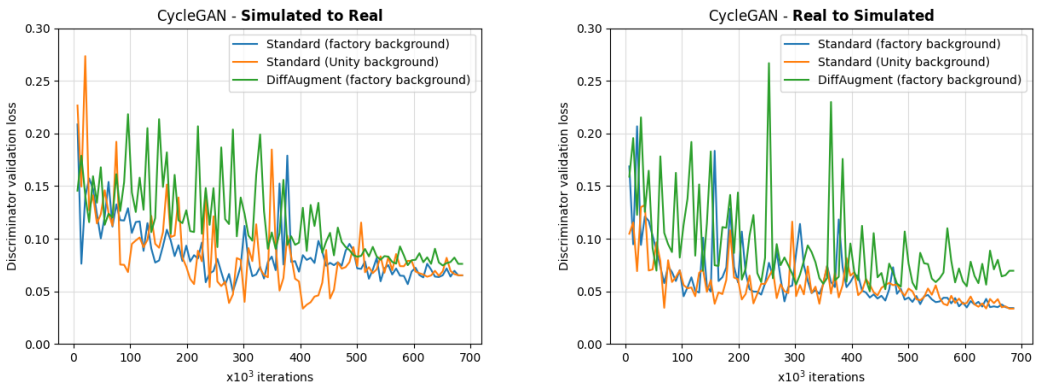
Figure 5.2: (a) Simulated domain cycle-consistency validation loss; (b) Real domain cycle-consistency validation loss.

The FID score is a more adequate measurement of the CycleGAN's performance over its cycle-consistency loss since it is computed using the final converted images. In contrast, CycleGAN's losses do not necessarily imply better translation results. Furthermore, CycleGAN's training process is mainly guided by its cycle-consistency loss. Therefore, if one direction of the translation process is not performing up to par, it might affect the entire GAN's training. Based on the obtained FID scores, we opted to discard the dataset with Unity's background. Despite not varying substantially in the conversion from the simulated to the real domain, the contrary translation was considerably worse.

5.1.1.2 DiffAugment

With Unity's background dataset discarded, we moved to incorporate DiffAugment into CycleGAN. DiffAugment did not drastically affect CycleGAN's performance as the authors demonstrated with StyleGAN2 [106]. In fact, there was almost no difference between the FID scores obtained with and without the differential augmentations (figure 5.1). However, it did make an impact on the discriminators, generators, and respective cycle-consistency losses. Now, with augmented images, the discriminators' task of distinguishing between real and fake instances was more difficult to solve, as can be seen by their losses (figure 5.3). This, in theory, should provide the generators with more helpful gradients, which would help stabilize training and promote better quality images. Even though the FID score does not corroborate better image quality, cycle-consistency losses were lower in both translation directions (figure 5.2) compared to the standard CycleGAN trained with the factory background dataset.

Ultimately, even though DiffAugment did not provide the same boost in performance as it did for StyleGAN2, it regularized training and lowered GAN's losses without increasing FID scores. Hence, we decided to maintain it for future experiments. In chapter 5.2 we can see that the Detectron2 model, trained on simulated data, managed to detect more vehicle parts on the images generated by the CycleGAN with the DiffAugment method than the images generated by the standard CycleGAN. This indicates that DiffAugment does, in fact, aid in providing translations that better preserve the images' annotations.



(a) (b)

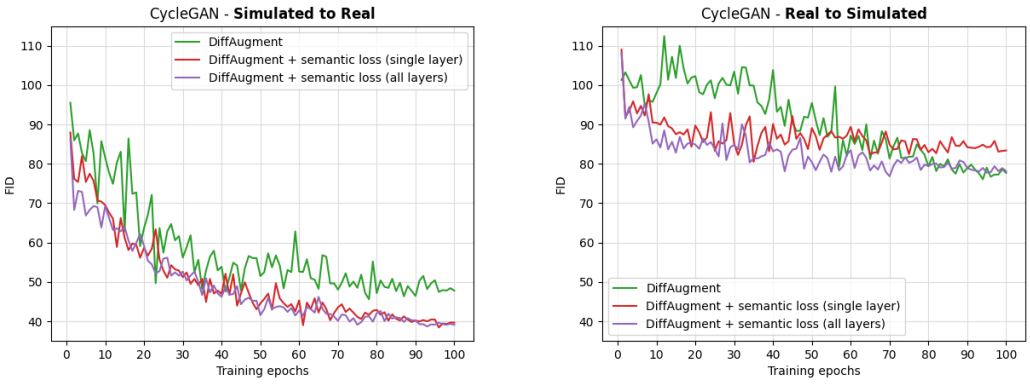
Figure 5.3: (a) Simulated to real domain discriminator validation loss the discriminator receives the simulated images converted into real variants as having fake labels and real images as having real labels; (b) Real to simulated domain discriminator validation loss the discriminator receives the real images converted into simulated variants as having fake labels and simulated images as having real labels.

5.1.1.3 Semantic consistency

Applying the proposed semantic loss (section 4.3.3.2) to CycleGAN yields better results, further improving the FID score for the simulated to real conversion, as shown in figure 5.4a, with no-

ticeable image improvements. One failure case was with black vehicles where the model would produce a side view of a vehicle's door. The simulated images' black vehicle's chassis was converted into a glass-like surface, while the white floor would be the door where the corresponding glass window rested. The semantic loss lessened this effect drastically and protected the vehicle's composition. This can be observed in more detail in figure 5.6.

When analyzing in more detail the semantic loss values, we can see that they start at their lowest value and steadily rise throughout the training process (figure 5.5). This makes sense since the generator does not yet apply significant modifications to the input image in the early training stages, making the corresponding embeddings not that distinct. As the training progresses, the semantic loss in all layers reacts to the more significant adjustments made by the generator, reaching its highest value at the end of training. At first glance, one could think of this as unusual behavior since model losses are expected to decrease during training. However, the backbone's parameters are not being optimized during training, meaning that the generator must produce the exact original image for the semantic loss to be zero. Since the other losses (GAN losses and cycle-consistency loss) guide the training process by introducing changes to obtain an image that more adequately belongs to the target distribution, it becomes apparent why the semantic loss would increase during training.

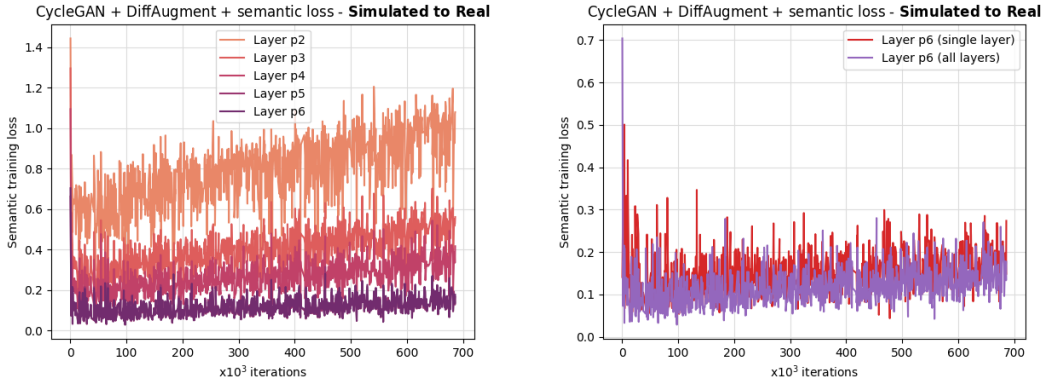


(a) (b)

Figure 5.4: The effect of the semantic consistency losses on FID scores; (a) FID score on training set during simulated to real translation; (b) FID score on training set during real to simulated translation.

When using all the backbone layers to apply the semantic loss, we can observe that the upper FPN layers, with larger sizes, produce a higher semantic loss compared to the lower layers. This is due to the fact that the overall image features to not fluctuate much during conversion. For example, layer p6 obtains the lowest loss of all layers since it encodes global image information. In contrast, layer p2 yields the highest loss due to image details being changed the most during translation (figure 5.5a). Moreover, when layer p6 is backed up by additional layers to enforce the semantic loss, its individual loss is lower. This can be explained by the fact that the incoming

features have already been influenced by the previous backbone layers whose embeddings are also being incorporated into the final semantic loss (figure 5.5b).



(a) (b)

Figure 5.5: Semantic consistency losses, during simulated to real translation; (a) Semantic consistency on all layers with each layer's individual loss; (b) Comparison between the same p6 layer when applying the semantic loss on all layers and when only applying it on the last (p6) layer.

Overall, this loss behaved exceptionally well for its purpose, which was to regularize training and not to fully guide it. If the semantic loss directed the entire CycleGAN's training, it would have taken priority over the other losses, with the final images not suffering the expected modifications as they would try to maintain their embedding consistency above all else. By not being the main focus, the new loss can assist the cycle-consistency and GAN losses, adjusting certain discrepancies in the translations.

Figure 5.6: Translation comparison given the same input on the 100th (final) epoch of each CycleGAN variation. We can see that the semantic loss prevents hallucinations and helps to preserve the vehicle's format. Furthermore, the use of DiffAugment, compared to the standard CycleGAN, slightly improves vehicle clarity despite not being near as noticeable as the semantic loss outputs.

5.1.1.4 Real to simulated detection

In addition to fine-tuning, we also experimented with converting the real images to simulated counterparts using the CycleGAN (figure 5.7) and evaluating them on the original Detectron2 trained on only simulated data. However, this did not reveal to be fruitful since the conversion from real to simulated did not maintain most of the original features, as shown in table 5.1. This is evidenced by the higher FID score obtained for the real to simulated domain conversion during CycleGAN experiments (figure 5.1), indicating a lower quality conversion compared to the simulated to real translation.

Furthermore, in table 5.1 the label-based accuracy when using simulated counterparts of real images is higher than the baseline accuracy, seemingly exhibiting better performance despite all the other metrics proving otherwise. This is evidence that the accuracy metric does not truthfully reflect the system's performance on real data.

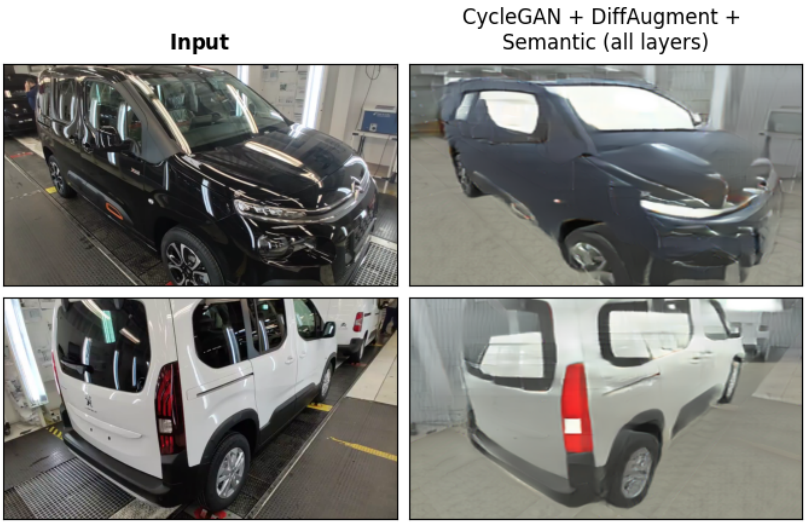


Figure 5.7: Real to simulated counterparts image examples on the 100th (final) epoch of CycleGAN + DiffAugment + Semantic consistency on all layers.

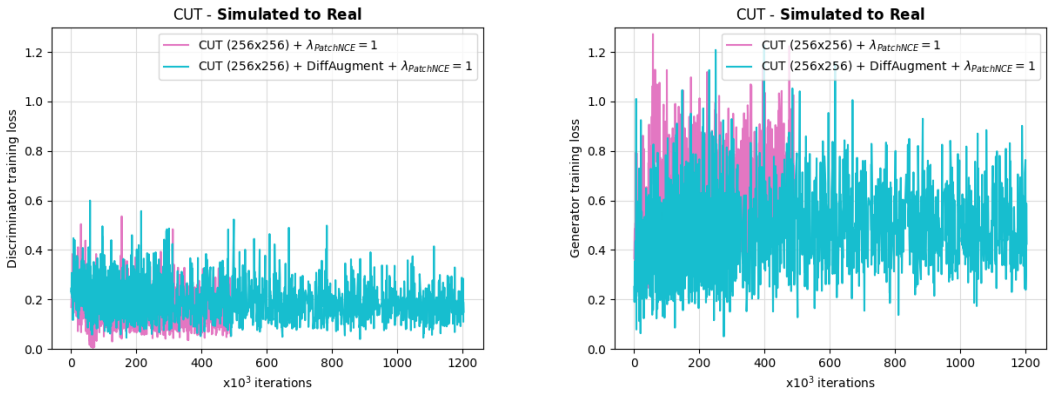
Table 5.1: Results on real data for the detection module when converting the input real image into simulated counterparts.

	Metrics	No fine-tuning (baseline) [68]	CycleGAN (Real to Simulated)
Example based	**Accuracy**	**0.280**	0.142
	Precision	**0.386**	0.284
	Recall	**0.517**	0.205
	F1-score	**0.442**	0.238
	Hamming loss	**0.343**	0.309
Label based	**Accuracy**	0.657	**0.691**
	Precision	**0.292 / 0.390**	0.163 / 0.357
	Recall	**0.438 / 0.521**	0.162 / 0.207
	F1-score	**0.307 / 0.446**	0.134 / 0.262
	MCC	**0.134 / 0.210**	0.026 / 0.089
	Jaccard index	**0.221 / 0.287**	0.097 / 0.151

5.1.2 CUT

As mentioned in section 4.3.4, CUT's lower computational complexity enabled us to experiment with higher resolution images. The initial CUT assessments used 256 by 256 pixels images and allowed to identify some issues occurring during domain translation. For the later experiments we used the full image resolution of 512 by 512 pixels with the appropriate modifications to account for the problems discovered during the first runs.

Overall, when inspecting discriminator and generator losses (figure 5.8) all runs behave as expected during training and reached stability as their FID scores plateaued (figure 5.9). However, not all obtained translations were successful (figure 5.10). Despite not theoretically collapsing, when using the default weight of 1.0 for the contrastive PatchNCE loss, we witness a side-effect of CUT's approach mentioned in detail in the next section 5.1.2.1.



(a) (b)

Figure 5.8: CUT discriminator and generator training losses; (a) CUT discriminators' training loss; (b) CUT generators' training loss. These models were stopped earlier, without completing 200 epochs, due to their severe hallucination cases.

5.1.2.1 Distribution matching

In section 3.6.2.2 CUT's method [62] is explained, along with its distribution matching side-effect of enlarging horses' bodies to better replicate zebras' body structure. In fact, distribution matching may not always be an undesirable side-effect. It helps create images that more adequately belong to the target domain, allowing for more impactful changes in the source images. However, if a more conservative approach is desired, it can introduce artifacts and produce changes far too dramatic to the input image during translation. In our first experiments we can witness this effect in action.

Our real dataset comprises more close-up images of vehicles, while the simulated one allows for more camera viewpoints to be rendered, exposing more of the environment. Due to this, CUT's lower resolution experiments with a PatchNCE loss weight of 1.0 (CUT's default), would increase vehicles' height or introduce nonexistent windows with reflections to more appropriately replicate the real images' scenario (figure 5.10 (c, d, e) - 3rd and 4th columns). In some cases, it produced

entirely novel images without any resemblance to the original ones (figure 5.10 (a,b) - 3rd column). Introducing DiffAugment helped to restore some of the original images' structure, resolving some of these issues, yet, some hallucinations remained (figure 5.10 (a,b) - 4th column). These hallucinations ought to be removed, as we cannot, in any way, alter the information contained in the source images during the translation process since it would defeat the goal of preserving the original images' annotations. These failure cases are addressed in the higher resolution experiments, explained in the following section 5.1.2.2.

5.1.2.2 PatchNCE's weight

For this section we discuss the influence of changing PatchNCE's weight loss for CUT, applied to our scenario of simulated to real conversion. As mentioned in section 4.3.4, we tested two different weights values of $\lambda_{PatchNCE} = 2.0$ and $\lambda_{PatchNCE} = 5.0$.

When comparing the FID scores of CycleGAN's best run (DiffAugment + semantic consistency on all backbone layers) and the new CUT models, we would not expect much visual improvement (figure 5.9). However, as can be seen in side-by-side comparisons (figure 5.10), this is not the case. CUT maintains vehicles' structures while also preserving their original colors, something that CycleGAN had trouble with in all its variants. This comes at the cost of a more conservative, but arguably preferable, translation.

Moreover, at first sight, the distinction between the 2 experimental PatchNCE weights is not immediately perceptible in some cases. But when inspecting more examples (figure 5.10 (a,b,f,g) - 5th and 6th columns), we can observe that the lower weight loss struggles in some instances ending up with similar hallucinations as previously seen in some CycleGAN runs (figure 5.10 (a,b) - 5th column).

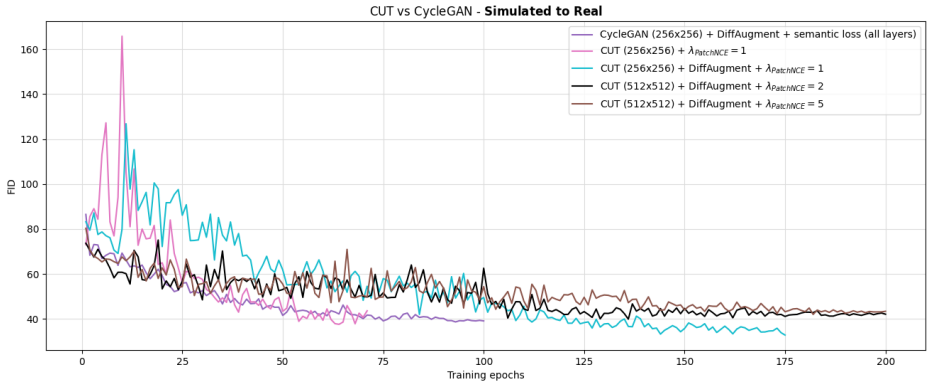


Figure 5.9: FID scores for the CycleGAN and CUT models with different PatchNCE loss weights during training. CycleGAN completed 100 epochs while CUT using 512 by 512 pixels images completed 200 epochs. CUT models using 256 by 256 pixels images were stopped earlier, without completing 200 epochs, due to their severe hallucination cases.

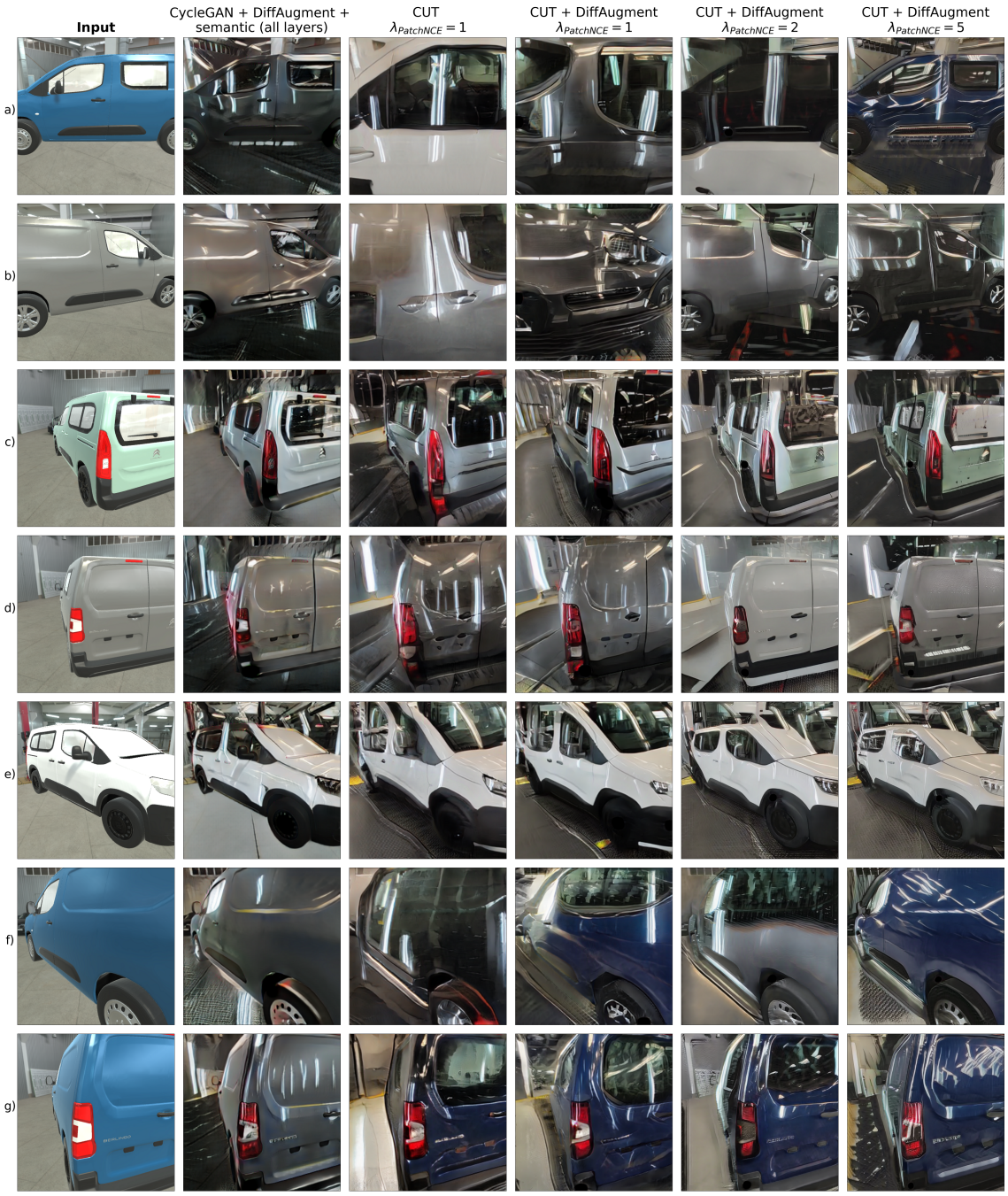


Figure 5.10: Comparison of translation quality given the same input on the last epoch of each model (CUT models with a PatchNCE weight of 2 and 5 were trained using 512 by 512 pixels images, all the other runs used 256 by 256 pixels resized images); **(a)** we can see a failure case of the default CUT model applied to our scenario, even with an increased contrastive loss ($\lambda_{PatchNCE} = 2$) it does not perform up to par with CycleGAN; **(b)** the use of DiffAugment fixes a failure case compared to the default CUT, proving that it can play an important role in training; **(c, d, e)** some cases of distribution matching where the generator stretches the vehicle chasis increasing its height to better match the real samples; **(f, g)** the regularizing effect of the higher contrastive loss weight ($\lambda_{PatchNCE} = 5$) can be seen as it does not introduce windows which are not present in the input image. CUT also properly preserves the input images' color at the cost of a more conservative translation.

5.2 Detector Fine-tuning

With all the previous conversion methods at our disposal, we have to choose which to use to fine-tune our Detectron2 model properly. We chose to select a variant from each domain translation model, i.e. one CycleGAN variant and one CUT variant, so as to ascertain how the two models affect the detection system's performance. For this purpose, as explained in section 4.4, we applied the original methods' Detectron2, trained on only simulated data, to a subset of converted images using both methods with the intent of identifying which experiment better preserved the original images' annotations. To perform this comparison, in addition to the last epoch, the epoch with the lowest FID score, as well as the epoch with the best compromise between FID score and semantic loss for each run were also considered.

From table 5.2 we can see that the model obtained from the epoch with best FID score on both default experiments (factory vs Unity background) retains more image information during conversion, therefore yielding better mAP than using the model from the last training epoch. Furthermore, the usage of DiffAugment also increases Detectron2's performance, proving that it helps to regularize the conversion process.

Table 5.2: Detectron2 preliminary validation results on converted images for each method's last and best FID epoch. Standard CUT models were left out due to their severe hallucination failure cases. "Sem" is an abbreviation for semantic loss.

Method	**Run**	**Detection**			**Segmentation**		
		mAP	**mAP@50**	**mAP@75**	**mAP**	**mAP@50**	**mAP@75**
No conversion	factory background	87.052	98.598	96.643	71.707	87.490	79.098
	Unity background	86.529	98.132	95.408	70.586	87.268	75.839
CycleGAN	Default - factory background (last epoch)	4.192	5.925	4.817	2.918	5.126	2.922
	Default - factory background (best FID)	4.861	7.004	5.543	3.571	5.659	3.661
	Default - Unity background (last epoch)	4.338	6.491	4.689	2.840	4.805	2.877
	Default - Unity background (best FID)	4.518	6.609	5.070	2.984	5.017	3.074
	DiffAugment (last epoch)	6.437	9.453	7.170	4.385	7.231	4.457
	DiffAugment (best FID)	9.172	12.887	10.565	6.747	10.839	7.012
	DiffAugment + Sem last layer (best FID)	36.965	51.395	42.844	21.967	38.940	21.760
	DiffAugment + Sem last layer (best FID + Sem loss)	39.753	54.697	45.813	24.358	42.933	23.816
	DiffAugment + Sem all layers (last epoch)	43.850	58.142	51.454	28.285	46.597	29.242
	DiffAugment + Sem all layers (best FID)	46.569	61.039	54.944	30.493	49.086	32.832
	DiffAugment + Sem all layers (best FID + Sem loss)	**48.748**	**63.433**	**57.412**	**32.964**	**52.885**	**34.961**
CUT	DiffAugmet + $\lambda_{PatchNCE} = 2$ (last epoch)	13.931	20.686	**15.917**	8.464	14.702	8.495
	DiffAugmet + $\lambda_{PatchNCE} = 5$ (last epoch)	**14.339**	**21.083**	15.798	**8.752**	**15.387**	**8.700**

Looking at the validation results (table 5.2), we see that CycleGAN with DiffAugment and semantic loss on all layers and CUT with the highest PatchNCE loss weight manage to obtain the best scores from each method. With these models, we converted the entire simulated dataset used for the initial training of Detectron2 and followed up with the fine-tuning and subsequent pseudo-label generation.

5.2.1 Fine-tuning on converted images

Following the experiment specifications detailed in section 4.4 we can observe the performance of the epoch with lowest validation loss in table 5.3 where we can see that Detectron2's fine-tuning, using converted images from both CycleGAN and CUT, was successful. Both fine-tuning

processes obtain good results on the corresponding converted datasets but also do not substantially reduce their ability to assess simulated images. For more specific model behaviour during training please refer to the appendix A.3.1.

Table 5.3: Fine-tuned Detectron2 preliminary validation results on simulated and converted images. We can see that the fine-tuned models successfully generalize to both simulated and real domains.

Method	**Dataset**	**Detection**			**Segmentation**		
		mAP	**mAP@50**	**mAP@75**	**mAP**	**mAP@50**	**mAP@75**
No fine-tuning	Simulated	87.052	98.598	96.643	71.707	87.490	79.098
	CycleGAN Converted	48.748	63.433	57.412	32.964	52.885	34.961
	CUT Converted	14.339	21.083	15.798	8.752	15.387	8.700
Fine-tuned with CycleGAN converted images	Simulated	82.921	98.261	94.146	62.698	84.236	69.084
	CycleGAN Converted	84.063	97.990	93.332	66.271	85.631	73.397
Fine-tuned with CUT converted images	Simulated	82.802	97.877	94.312	62.186	82.860	69.125
	CUT Converted	82.878	97.473	93.433	63.128	85.176	70.060

The detection module is composed of the macro detection of parts, executed by Detectron2, followed by the CBIR stage, in which a search for the complete parts' code is done. To assess the performance on real data, we measured solely Detectron2's performance on the macro class it predicts (table 5.4) in addition to the performance of the complete module (table 5.5).

Table 5.4: Results on real data, reported as a multi-class multi-label classification problem, for the fine-tuned Detectron2. Both CycleGAN and CUT refer to the previously mentioned models chosen to perform fine-tuning with, i.e. CycleGAN + DiffAugment + Sem all layers and CUT + DiffAugment + $\lambda_{PatchNCE} = 5$. For our label-based results, one can observe the macro value followed by the micro value for each metric.

	Metrics	**No fine-tuning (baseline) [68]**	**CycleGAN**	**CUT**
Example based	**Accuracy**	0.442	0.440	**0.459**
	Precision	**0.596**	0.517	0.535
	Recall	0.654	0.750	**0.766**
	F1-score	0.624	0.612	**0.630**
	Hamming loss	**0.352**	0.412	0.388
Label based	**Accuracy**	**0.648**	0.588	0.612
	Precision	0.481 / 0.582	0.492 / 0.516	**0.490 / 0.535**
	Recall	0.579 / 0.657	0.658 / 0.750	**0.667 / 0.769**
	F1-score	0.479 / 0.617	0.495 / 0.611	**0.512 / 0.631**
	MCC	**0.164 / 0.296**	0.096 / 0.220	0.126 / 0.267
	Jaccard index	0.372 / 0.447	0.377 / 0.440	**0.406 / 0.461**

Table 5.5: Results on real data, reported as a multi-class multi-label classification problem, for the detection module using the new fine-tuned Detectron2. Both CycleGAN and CUT refer to the previously mentioned models chosen to perform fine-tuning with. For our label-based results, one can observe the macro value followed by the micro value for each metric. In parenthesis are the relative percentage differences concerning the baseline.

	Metrics	No fine-tuning (baseline) [68]	CycleGAN	CycleGAN + Converted CBIR	CUT	CUT + Converted CBIR
Example based	**Accuracy**	0.280	0.294	0.280	0.309	0.298
	Precision	0.386	0.357 **(-7.5%)**	0.345 **(-10.6%)**	0.370 **(-4.1%)**	0.356 **(-7.8%)**
	Recall	0.517	0.621 **(+20.1%)**	0.587 **(+13.5%)**	0.645 **(+24.8%)**	0.632 **(+22.2%)**
	F1-score	0.442	0.453 **(+2.5%)**	0.434 **(-1.8%)**	0.470 **(+6.3%)**	0.456 **(+3.2%)**
	Hamming loss	0.343	0.396	0.405	0.382	0.394
Label based	**Accuracy**	0.657	0.604	0.595	0.618	0.606
	Precision	0.292 / 0.390	0.302 / 0.357 **(+3.4%/-8.5%)**	0.293 / 0.345 **(+0.3%/-11.5%)**	0.330 / 0.373 **(+13%/-4.4%)**	0.338 / 0.361 **(+15.8%/-7.4%)**
	Recall	0.438 / 0.521	0.513 / 0.622 **(+17.1%/+19.4%)**	0.497 / 0.590 **(+13.5%/+13.2%)**	0.534/ 0.650 **(+21.9%/+24.8%)**	0.545 / 0.636 **(+24.4%/+22.1%)**
	F1-score	0.307 / 0.446	0.323 / 0.454 **(+5.2%/+1.8%)**	0.317 / 0.435 **(+3.3%/-2.5%)**	0.356 / 0.474 **(+16%/+6.3%)**	0.357 / 0.461 **(+16.3%/+3.4%)**
	MCC	0.134 / 0.210	0.113 / 0.194	0.105 / 0.166	0.140 / 0.227	0.150 / 0.205
	Jaccard index	0.221 / 0.287	0.228 / 0.293	0.221 / 0.278	0.263 / 0.311	0.261 / 0.299

As previously explained in section 4.2.4 we adopted the F1-score as our main metric to gauge the system's overall performance, since we believe it provides a more accurate representation of its behaviour in real scenarios. With this in mind, both tables (table 5.4 and table 5.5) show that both methods increase Detectron2's performance on real images, with example-based metrics obtaining a 2.5% F1-score improvement using CycleGAN converted images and 6.3% using CUT's conversion method compared to the baseline [68]. This is an interesting result since the original Detectron2, trained on simulated data, obtained better validation results with CycleGAN (table 5.2). However, this signifies that CUT's converted images are from a distribution closer to the real domain that Detectron2 cannot fully assess. This in turn, provides a better fine-tuning process for Detectron2 since it brings its image features closer to the real domain.

Improving Detectron2's model yields better results when using the original CBIR, composed of embeddings of simulated images. However, creating a new CBIR with the corresponding translation method, meaning that the CBIR embeddings are generated from crops of converted images, does not improve the overall module's performance. Perhaps the use of a ResNeXt-50 32x4d pre-trained on Imagenet is no longer a viable option if we wish to incorporate slightly different embeddings.

5.2.2 Global NMS

Among the Detectron2's predictions multiple object instances overlapped to a much higher degree than expected. This should not happen since it introduces false positives into our predictions and hurts the performance of the overall module. The cause of the problem is that Detectron2 applies intra-class NMS, eliminating overlapping bounding boxes only if they belong to the same class. To tackle this issue, we applied a post-processing step in which global NMS with an IoU threshold of 50% is enforced (figure 5.11). This prunes overlapping bounding boxes independently of their

classes. With the addition of this step, we managed to reduce false positives and make use of our predictions more efficiently, which ultimately leads to better performance in the real domain. The effect this new post-processing step has on Detectron2's performance can be seen in table 5.6 and in the complete detection module in table 5.7.

Figure 5.11: The effect of using global NMS on detectron2's predictions: standard predictions on the left and applying global NMS on the right. In this example we prune 10 overlapping bounding boxes, with only the most confident predictions remaining.

Table 5.6: Results on real data for the fine-tuned Detectron2 when applying the global NMS post-processing step.

		Applying global NMS		
	Metrics	**No fine-tuning (baseline) [68]**	**CycleGAN**	**CUT**
Example based	**Accuracy**	0.357	0.435	**0.491**
	Precision	**0.647**	0.600	0.637
	Recall	0.446	0.617	**0.684**
	F1-score	0.528	0.609	**0.660**
	Hamming loss	0.344	0.348	**0.308**
Label based	**Accuracy**	0.656	0.652	**0.692**
	Precision	0.462 / 0.647	0.491 / 0.594	**0.541 / 0.632**
	Recall	0.419 / 0.449	0.550 / 0.616	**0.603 / 0.685**
	F1-score	0.394 / 0.530	0.455 / 0.605	**0.525 / 0.658**
	MCC	0.198 / 0.284	0.166 / 0.294	**0.247 / 0.380**
	Jaccard index	0.305 / 0.361	0.350 / 0.433	**0.429 / 0.490**

Table 5.7: Results on real data for the detection module using the new fine-tuned Detectron2 when applying the global NMS post-processing step. In parenthesis are the relative percentage differences in relation to the baseline.

		Applying global NMS				
	Metrics	**No fine-tuning (baseline) [68]**	**CycleGAN**	**CycleGAN + Converted CBIR**	**CUT**	**CUT + Converted CBIR**
Example based	**Accuracy**	0.244	0.289	0.287	0.337	0.320
	Precision	0.426	0.403 **(-5.4%)**	0.411 **(-3.5%)**	0.440 **(+3.3%)**	0.422 **(-0.9%)**
	Recall	0.356	0.497 **(+39.6%)**	0.472 **(+32.6%)**	0.577 **(+62.1%)**	0.550 **(+54.5%)**
	F1-score	0.388	0.445 **(+14.7%)**	0.440 **(+13.4%)**	0.499 **(+28.6%)**	0.478 **(+23.2%)**
	Hamming loss	0.291	0.329	0.326	0.302	0.316
Label based	**Accuracy**	0.709	0.671	0.674	0.698	0.684
	Precision	0.295 / 0.438	0.284 / 0.401 **(-3.7%/-8.4%)**	0.291 / 0.402 **(-1.4%/-8.2%)**	0.366 / 0.445 **(+24.1%/+1.6%)**	0.356 / 0.426 **(+20.7%/-2.7%)**
	Recall	0.322 / 0.359	0.428 / 0.498 **(+32.9%/+38.7%)**	0.413 / 0.475 **(+28.3%/+32.3%)**	0.467 / 0.581 **(+45.0%/+61.8%)**	0.468 / 0.553 **(+45.3%/+54.0%)**
	F1-score	0.264 / 0.395	0.293 / 0.444 **(+11%/+12.4%)**	0.294 / 0.436 **(+11.4%/+10.4%)**	0.362 / 0.504 **(+37.1%/+27.6%)**	0.354 / 0.481 **(+34.1%/+21.8%)**
	MCC	0.154 / 0.207	0.140 / 0.217	0.150 / 0.210	0.201 / 0.297	0.192 / 0.264
	Jaccard index	0.199 / 0.246	0.213 / 0.286	0.214 / 0.278	0.277 / 0.337	0.267 / 0.317

The use of global NMS decreases the baseline's overall performance and increases all the other fine-tuned variants. With this supplementary step, example-based metrics achieve a F1-score increase of 14.7% when fine-tuning with CycleGAN converted images and a 28.6% gain when using CUT compared to the new global NMS baseline.

Since global NMS removes the overlapping bounding boxes with the least probability associated with them, it could mean that the baseline was benefiting from other less confident auxiliary predictions. One could argue that applying this post-processing step demonstrates the complete module's performance more accurately since only the predictions which Detectron2 finds more probable are used for evaluation, not tainting the subsequently collected metrics with other less meaningful predictions. With this in mind, the collected performance numbers further highlight the improvement offered by fine-tuning as the gap between the baseline and the fine-tuned Detectron2 widens compared to not using global NMS.

5.2.3 Fine-tuning on real pseudo-labels

As fine-tuning with CUT obtained better results, we proceeded with the pseudo-label generation using the corresponding fine-tuned Detectron2. With the fine-tuning process explained in section 4.4 we generated pseudo-labels from Detectron2's predictions and proceeded with the second fine-tuning process. Furthermore, we also discovered that lowering the learning rate from 0.01 to 0.005 provided us with better results. For more specific model behaviour during training please refer to the appendix A.3.2.

Table 5.8 shows the performance of the Detectron2 fine-tuned with real images' pseudo-labels on the preliminary simulated and converted datasets. The fine-tuning process decreases the Detectron2's ability to correctly assess both these domains. One could argue that this is due to the model bridging the domain gap and reaching closer to the real domain, as these reduced performance results do not translate to poor system's performance on real data (table 5.10 and table 5.11).

Table 5.8: Preliminary validation results on simulated and converted images for the fine-tuned Detectron2 using pseudo-labels from real images. We can see that fine-tuning with pseudo-labels diminishes the performance on the simulated and converted domains as the model shifts towards learning the real domain.

Method	**Dataset**	**Detection**			**Segmentation**		
		mAP	**mAP@50**	**mAP@75**	**mAP**	**mAP@50**	**mAP@75**
No fine-tuning	Simulated	87.052	98.598	96.643	71.707	87.490	79.098
	CUT Converted	14.339	21.083	15.798	8.752	15.387	8.700
Fine-tuned with CUT converted images	Simulated	82.802	97.877	94.312	62.186	82.860	69.125
	CUT Converted	82.878	97.473	93.433	63.128	85.176	70.060
Fine-tuned with CUT converted images + pseudo-labels	Simulated	63.667	85.333	77.612	42.953	65.409	47.778
	CUT Converted	55.865	78.414	66.681	35.161	58.227	37.280

In table 5.9 we see the results for the Detectron2 stage where macro categories are detected, comparing both the fine-tuned Detectron2 with only CUT converted images and with CUT followed by pseudo-labels. As shown, the use of pseudo-labels benefits Detectron2 and applying global NMS manages to further improve this stage's ability to analyze real data. These results are consistent with the previous fine-tuning stage using only translated images, as both fine-tuning processes take advantage of the new post-processing step.

Table 5.9: Results on real data for the fine-tuned Detectron2 with real images' pseudo-labels. Previous results for the fine-tuned Detectron2 with only CUT converted images are also present for comparison.

				Applying global NMS	
	Metrics	**CUT**	**CUT + pseudo-labels**	**CUT**	**CUT + pseudo-labels**
Example based	**Accuracy**	0.456	0.501	0.491	**0.525**
	Precision	0.535	0.523	**0.637**	0.561
	Recall	0.766	**0.921**	0.684	0.853
	F1-score	0.630	0.667	0.660	**0.677**
	Hamming loss	0.388	0.402	**0.308**	0.354
Label based	**Accuracy**	0.612	0.598	**0.692**	0.646
	Precision	0.490 / 0.535	0.490 / 0.520	**0.541 / 0.632**	0.525 / 0.559
	Recall	0.667 / 0.769	**0.801 / 0.921**	0.603 / 0.685	0.737 / 0.822
	F1-score	0.512 / 0.631	0.566 / 0.664	0.525 / 0.658	**0.559 / 0.675**
	MCC	0.126 / 0.267	0.119 / 0.320	**0.247 / 0.380**	0.195 / 0.356
	Jaccard index	0.406 / 0.461	0.458 / 0.498	0.429 / 0.490	**0.453 / 0.510**

In table 5.10 we can see that fine-tuning with pseudo-labels from real images further improves the generalization abilities of the Detectron2 for the real domain, albeit not as considerably as fine-tuning with translated images. When not applying global NMS, fine-tuning with the generated pseudo-labels increases the overall system's performance with the example-based F1-score increasing 4% compared to only fine-tuning with CUT's converted samples.

However, when employing global NMS (table 5.11) the positive effect is not immediately noticeable since some metrics improve but the F1-score slightly decreases (-0.4%) when comparing to not using pseudo-labels for fine-tuning. Despite this, we manage to have a 111% improvement for the example-based recall and a 28.1% for the example-based F1-score over the baseline method, meaning that the system is retrieving more relevant predictions when given real data. For

the system's performance results using the converted CBIR with CUT created images please refer to the appendix A.4.1.

Table 5.10: Results on real data for the detection module using the fine-tuned Detectron2 on the converted images and generated pseudo-labels. We can also see the relative percentage differences concerning the baseline and in regard to the previous fine-tuned Detectron2 with only CUT converted images (without pseudo-labels).

	Metrics	No fine-tuning (baseline) [68]	CUT		
			w/ PL	% WRT baseline	% WRT w/o PL
Example based	**Accuracy**	0.280	0.324	**+15.7%**	**+4.9%**
	Precision	0.386	0.348	**-9.8%**	**-5.9%**
	Recall	0.517	0.824	**+59.4%**	**+27.8%**
	F1-score	0.442	0.489	**+10.6%**	**+4.0%**
	Hamming loss	0.343	0.456	**-33.0%**	**-19.4%**
Label based	**Accuracy**	0.657	0.544	**-17.2%**	**-12.0%**
	Precision	0.292 / 0.390	0.303 / 0.348	**+3.8%/-10.8%**	**-8.2%/-6.7%**
	Recall	0.438 / 0.521	0.659 / 0.825	**+50.5%/+58.3%**	**+23.4%/+27.0%**
	F1-score	0.307 / 0.446	0.382 / 0.489	**+24.4%/+9.6%**	**+7.3%/+3.2%**
	MCC	0.134 / 0.210	0.105 / 0.244	**-21.6%/+16.2%**	**-25.0%/+7.5%**
	Jaccard index	0.221 / 0.287	0.282 / 0.324	**+27.6%/+12.9%**	**+7.2%/+4.2%**

Table 5.11: Results on real data for the detection module using the fine-tuned Detectron2 on the converted images and generated pseudo-labels when applying the global NMS post-processing step. We can also see the relative percentage differences concerning the baseline and in regard to the previous fine-tuned Detectron2 with only CUT converted images (without pseudo-labels).

			Applying global NMS		
	Metrics	No fine-tuning (baseline) [68]	CUT		
			w/ PL	% WRT baseline	% WRT w/o PL
Example based	**Accuracy**	0.244	0.331	**+35.7%**	**-1.78%**
	Precision	0.426	0.371	**-12.9%**	**-15.7%**
	Recall	0.356	0.751	**+111%**	**+30.2%**
	F1-score	0.388	0.497	**+28.1%**	**-0.4%**
	Hamming loss	0.291	0.402	**-38.1%**	**-33.1%**
Label based	**Accuracy**	0.709	0.598	**-15.7%**	**-14.3%**
	Precision	0.295 / 0.438	0.333 / 0.371	**+12.9%/-15.3%**	**-9.0%/-16.6%**
	Recall	0.322 / 0.359	0.601 / 0.751	**+86.6%/+109.2%**	**+28.7%/+29.3%**
	F1-score	0.264 / 0.395	0.380 / 0.497	**+43.9%/+25.8%**	**+5.0%/-1.4%**
	MCC	0.154 / 0.207	0.157 / 0.260	**+1.9%/+25.6%**	**-21.9%/-12.5%**
	Jaccard index	0.199 / 0.246	0.285 / 0.331	**+43.2%/+34.6%**	**+2.9%/-1.8%**

5.3 Summary

As the absence of fully annotated real-world data prevents us from further improving the detection system's ability to generalize to real samples with standard methods, we adopted the DA techniques of domain translation and pseudo-labeling. Domain translation involves converting simulated images into real counterparts while preserving their original annotations. To perform this task, two methods were tested: CycleGAN [107] and CUT [62]. Using the converted images to fine-tune our object detection and segmentation model (Detectron2), both translation methods

managed to improve the overall system's performance on real data, with CUT yielding better results.

Furthermore, a post-processing step was added to more efficiently make use of Detectron2's predictions. Denominated global NMS, this step prunes overlapping bounding boxes independently of their class labels, and provides more realistic performance metrics of the detection system. By diminishing false positives, global NMS further highlights our approach's improvement over the baseline. More specifically, when fine-tuning with CycleGAN's converted images the example-based F1-score obtained a 2.5% improvement without global NMS, and a 14.7% improvement when applying global NMS, compared to the baseline results [68]. When using CUT's converted images, we get closer to the real domain as the example-based F1-score obtained a 6.3% improvement without global NMS, and a 28.6% improvement when applying global NMS.

By generating pseudo-labels for real samples using the newly fine-tuned Detectron2 we gain an additional source of information. Applying fine-tuning with pseudo-labels on top of the fine-tuned Detectron2 with CUT's converted images manages to further generalize the system for real cases, yielding an additional example-based F1-score 4% improvement compared to only fine-tuning with CUT, when not applying global NMS.

Chapter 6

Conclusion

For this chapter we conclude our project's development with an overview of the entire system's pipeline and the improvements made to said system, with possible future modifications to be made.

6.1 Overview

Deep learning (DL) represents an active field of research, with many methods and techniques developed to overcome problems that traditional machine learning models could not solve effectively. Despite DL's progress, applying research ideas to some real scenarios still proves to be a challenge. The absence of the necessary annotated data along with budget and time constraints inhibit the possible applications of supervised methods to real-world problems. One way to tackle this data shortage is by using a related source of information, such as simulators, to automatically produce annotated samples. By using simulated labeled samples in conjunction with unlabeled real data, the aforementioned difficulties might be overcome to allow the use of DL in new unexplored areas.

As discrepancies arise when merging different data sources, additional procedures must be in place to accommodate them. These discrepancies form what is called a domain gap. Domain adaptation is a field of transfer learning that tackles the issue of using different data distributions for the same final task. This is achieved by either finding a domain invariant representation of the shared features present in each dataset or by performing domain mapping. The latter converts an image from one domain to the other while trying to maintain its original features.

Therefore, this thesis extends upon the system developed in a previous work [68], by combining simulated and real sources of data to improve the overall system performance on real images. With the final goal of verifying a vehicle parts' configuration at the end of a production line, the previously proposed system was composed of an object detection and segmentation stage, combined with a content-based image retrieval (CBIR) stage to allow for modular future parts' updates. The former stage, when given a vehicle's image, detects and classifies its parts' macro-categories which, in turn, would be directed to one of the composing CBIR modules, one for each macro-category, where a search for each specifics part's code is made. The CBIR stage was in place to

allow future updates to the system. Macro-categories are not expected to change, but should it be necessary to add novel parts' codes, one must only update the corresponding CBIR module.

The first stage of detecting and segmenting vehicle parts was performed by Detectron2, which, by analyzing previous work's findings, we concluded that it was the system's main bottleneck for good generalization to the real domain (section 4.2.1.1). Hence, our approach focuses on improving Detectron2's ability to detect and classify parts' macro-categories for real samples using DA techniques. Moreover, simulated and real datasets had already been collected, and the simulator which produced the former provided us with easy dataset manipulation.

We progressively fine-tuned the object detection and segmentation model to increase the system's performance in real cases. First, with the repurposed simulated dataset converted into its real counterpart, and afterward, with pseudo-labels obtained from the weakly annotated real samples. We experimented with two GAN-based translation methods: CycleGAN [107] and CUT [62]. CycleGAN is based on a cycle-consistency loss which encourages the preservation of features when an image from the source domain is converted to the target domain and back to its original domain. This method inherently requires conversion in both directions to apply the cycle-consistency loss. Therefore, we are able to convert from the source to the target domain and vice-versa. In contrast, CUT only needs to compute a single-sided domain translation since it does not implement any concrete cycle-consistency loss. Instead, it uses a contrastive loss to ensure feature preservation during conversion. This provides a more lightweight and efficient method than CycleGAN and allows for higher resolution image experiments.

We proposed adding a semantic loss to CycleGAN with the intent of regularizing the training process and encouraging feature preservation during domain conversion (section 4.3.3.2). Furthermore, a simple method that improves GAN's data efficiency, denominated DiffAugment [106], was applied to both methods and its effects were analyzed. DiffAugment provides differentiable data augmentations for GANs and lessens the need for a large dataset by preventing discriminator overfitting. Differentiable image augmentations allow the flow of gradients back to the generator from the data-augmented discriminator's outputs without the side-effects of other implementations (section 4.3.2). When DiffAugment was applied to our domain translation methods, it regularized training and improved translation consistency.

It was challenging to determine our approach's performance on a real scenario since fully annotated real images were unavailable. Addressed as a multi-class multi-label problem (section 4.2.4), **we can conclude that both conversion methods improve the module's ability to analyze real images**, with CUT providing better results compared to CycleGAN when using a higher contrastive loss weight to produce more conservative translations. Moreover, both conversion methods benefit from DiffAugment as it helps create more reliable conversions, fixing some failure cases (figure 5.10). The use of a semantic loss on CycleGAN is crucial (section 5.1.1.3), not to fully guide the training process but to aid the other existing losses into achieving better outcomes. As for CUT's implementation, some hyperparameter tuning was required to prevent distribution matching from happening and distorting the simulated images' annotations in the process (section 5.1.2).

In addition to the improvements mentioned above, we added a global NMS post-processing step (section 5.2.2) to the obtained predictions as these often contained multiple distinct macro-categories in the exact location. This step prunes less probable overlapping bounding boxes, providing us with metrics that more accurately portray our approach's effectiveness. **This further widens the gap between the baseline**, using only simulated images, and our fine-tuned models taking advantage of multiple data sources.

Despite a failed attempt at using CycleGAN to convert real images into simulated equivalents and using previous work's Detectron2 to analyze them, which ultimately diminished the performance of the system (section 5.1.1.4), the primary fine-tuning process yielded good results. With the newly fine-tuned Detectron2 on **CycleGAN's converted images** from simulated to real domain, the **example-based F1-score obtained a 2.5% improvement without global NMS, and a 14.7% improvement when applying global NMS**, compared to the baseline results [68]. When using **CUT's converted images**, we get closer to the real domain as the **example-based F1-score obtained a 6.3% improvement without global NMS, and a 28.6% improvement when applying global NMS** (table 5.5 and table 5.7).

We further performed fine-tuning using pseudo-labels (section 5.2.3). Pseudo-labeling is the process of fabricating weak annotations for real samples by applying the fine-tuned Detectron2 on real images and considering its predictions as pseudo-labels. By training the same Detectron2 on these real images with pseudo-generated annotations, we create a more robust model as clusters for the real distribution are further established. This process further improves the system's results as the gap between simulated and real domains diminishes. **Applying fine-tuning with pseudo-labels** on top of the fine-tuned Detectron2 with CUT's converted images **yields an additional example-based F1-score 4% improvement** compared to only fine-tuning with CUT, without applying global NMS (table 5.10).

In conclusion, this approach's success in improving the previous work's system [68], without the need for fully annotated real images in any step of the pipeline, demonstrates that DA techniques are exceptionally useful at improving previous applications and extending the use of DL to novel areas where data requirements are a nuisance.

6.2 Future Work

Simulated data has many applications since we can replicate an endless amount of annotated samples for any field we desire. While research methods do not always consider real-world applications, DA is a remarkably important field as it takes information from multiple sources, allowing the transference of useful features from a seemingly unimportant simulated domain to our real-world problems.

Nevertheless, some possible improvements can be further explored for our application scenario, including:

- Analyzing other domain translation methods.

- Improving the CBIR stage since it also plays an important role in obtaining the final parts' codes.
- Annotate a few real samples with bounding boxes to measure the system's performance in real-world cases more accurately.
- Further fine-tune Detectron2 with more generated pseudo-labels.

Appendix A

Extra experiment plots

A.1 CycleGAN's FID scores

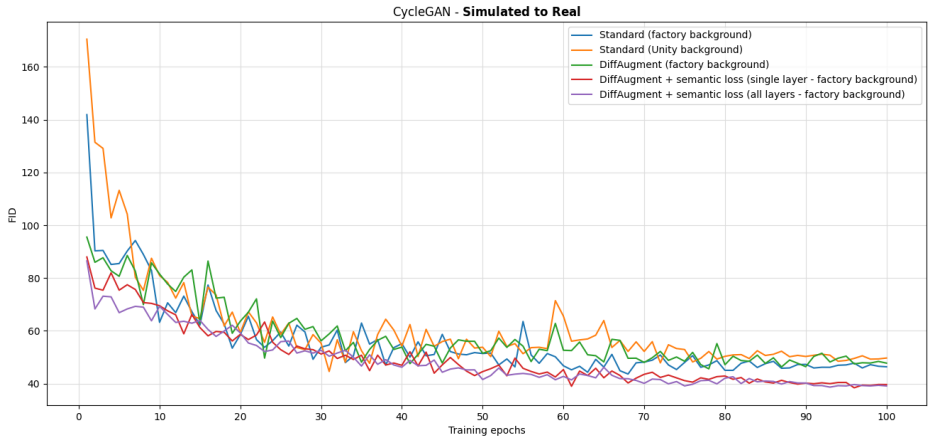


Figure A.1: FID for all CycleGAN's experiments during simulated to real translation.

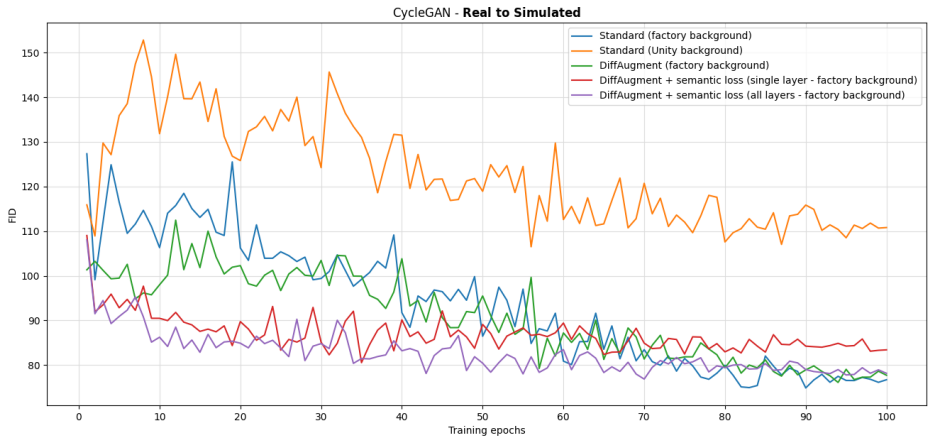


Figure A.2: FID for all CycleGAN's experiments during real to simulated translation.

A.2 Domain translation comparisons

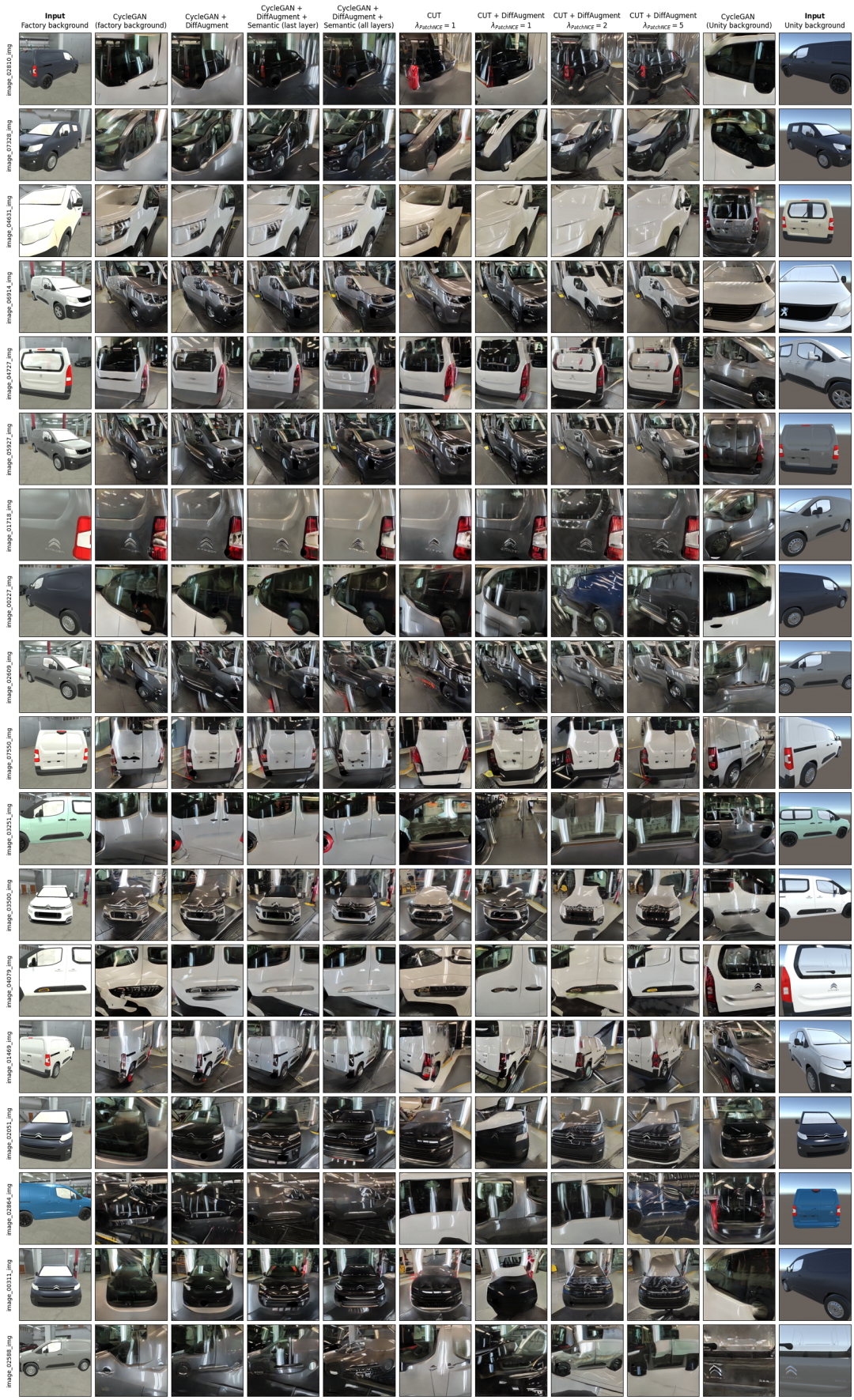


Figure A.3: Domain translation comparisons (part 1)

Figure A.4: Domain translation comparisons (part 2)

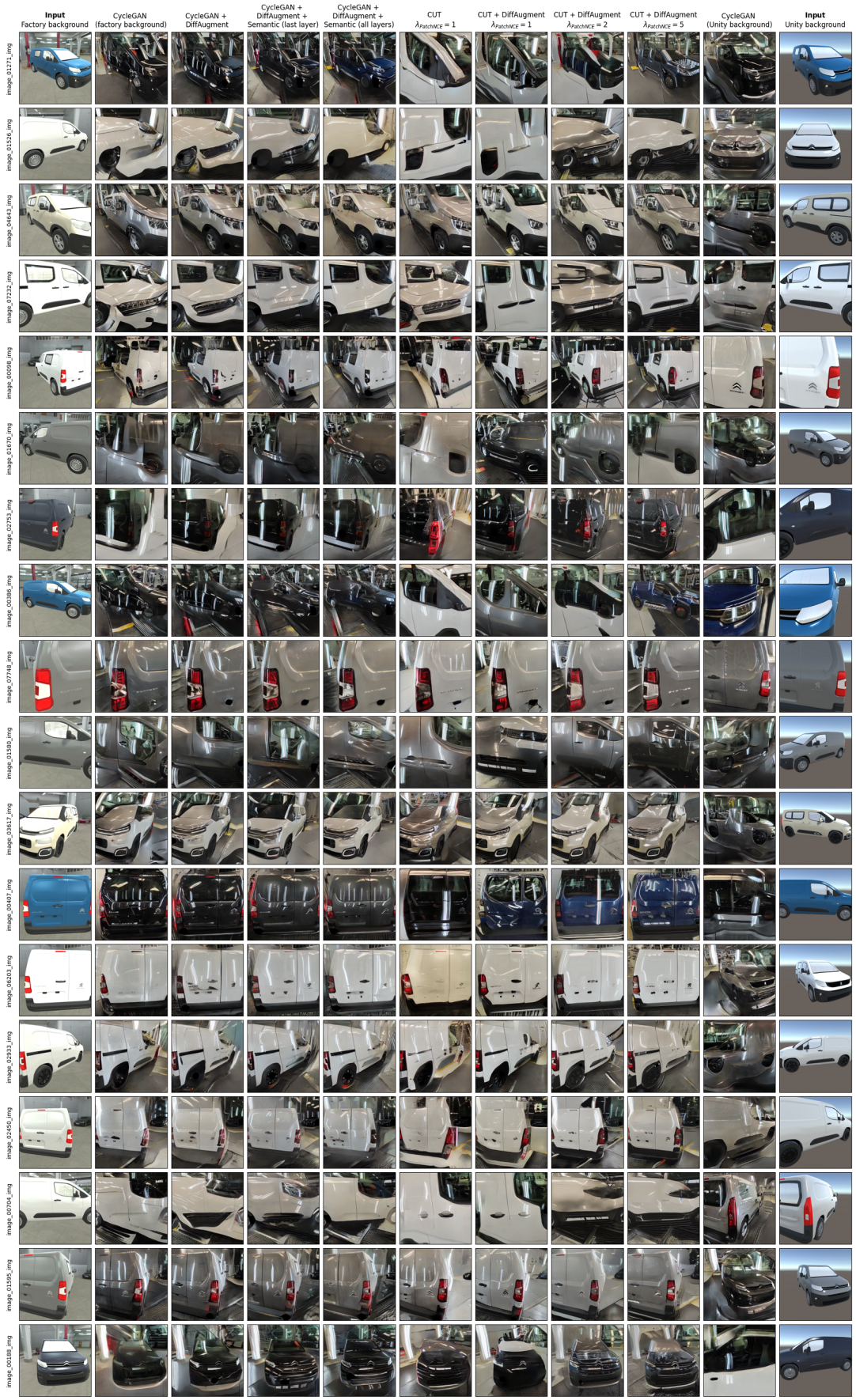


Figure A.5: Domain translation comparisons (part 3)

A.3 Detectron2's fine-tuning processes

A.3.1 Detectron2's fine-tuning on converted images

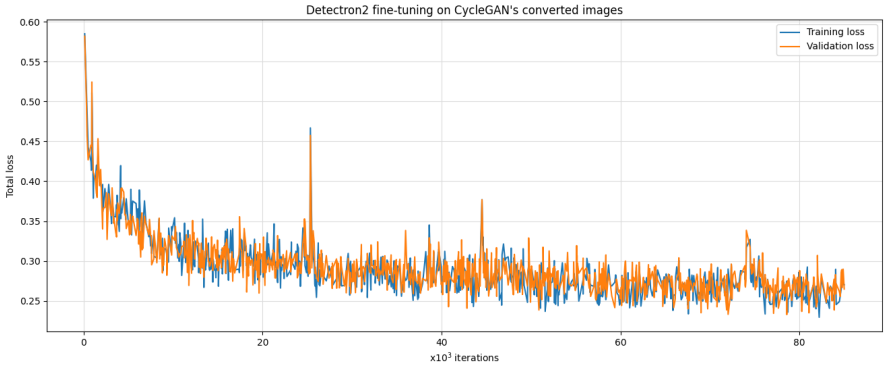
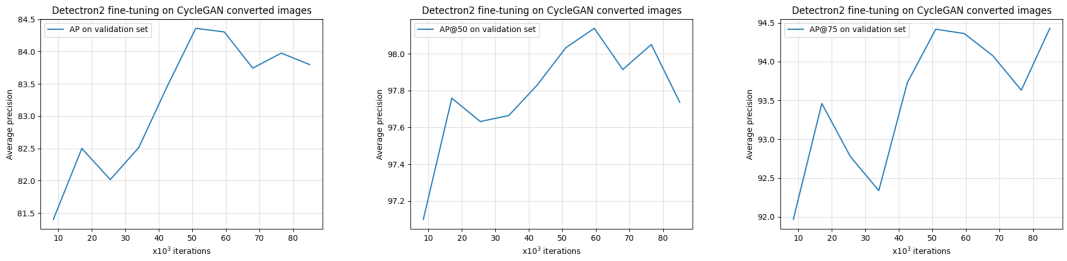


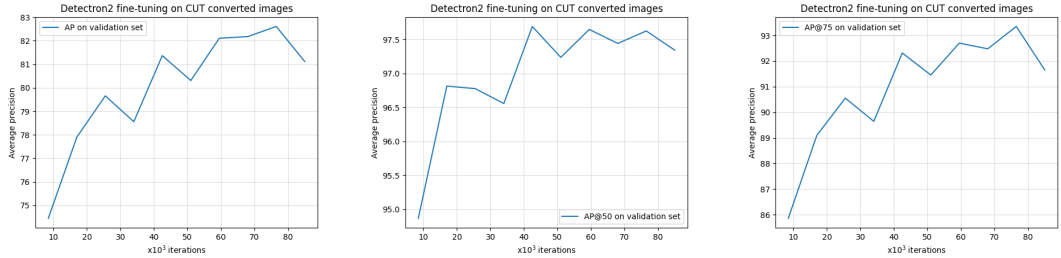
Figure A.6: Detectron2's loss during fine-tuning on CycleGAN converted images.



(a) (b) (c)

Figure A.7: Detectron2's AP during fine-tuning on CycleGAN converted images.

Figure A.8: Detectron2's loss during fine-tuning on CUT converted images.



(a) (b) (c)

Figure A.9: Detectron2's AP during fine-tuning on CUT converted images.

A.3.2 Detectron2's fine-tuning on real images' pseudo-labels

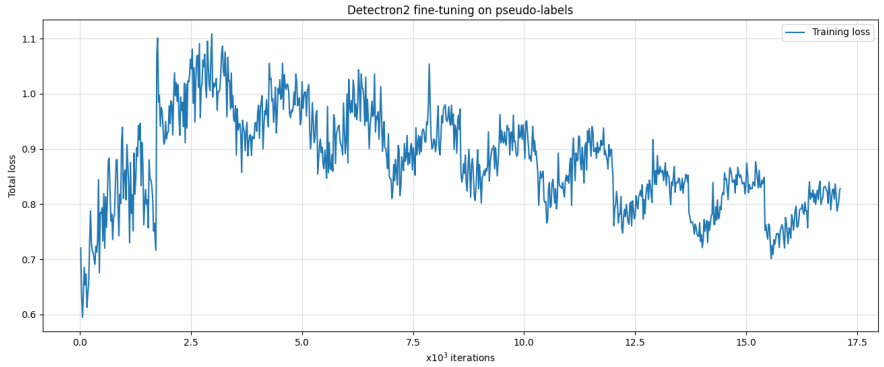


Figure A.10: Detectron2's loss during fine-tuning on real images' pseudo-labels.

A.4 Detection module real images assessment

A.4.1 Fine-tuned Detectron2 using CUT and pseudo-labels

Table A.1: Results on real data for the fine-tuned Detectron2 with real images' pseudo-labels.

				Applying global NMS	
	Metrics	**No fine-tuning (baseline) [68]**	**CUT + pseudo-labels**	**No fine-tuning (baseline) [68]**	**CUT + pseudo-labels**
Example based	**Accuracy**	0.442	0.501	0.357	0.525
	Precision	0.596	0.523	0.525	0.561
	Recall	0.654	**0.921**	0.446	0.853
	F1-score	0.624	0.667	0.528	**0.677**
	Hamming loss	0.352	**0.402**	0.344	0.354
Label based	**Accuracy**	0.648	0.598	**0.656**	0.646
	Precision	0.481 / 0.582	0.490 / 0.520	**0.462 / 0.647**	0.525 / 0.559
	Recall	0.579 / 0.657	**0.801 / 0.921**	0.419 / 0.449	0.737 / 0.822
	F1-score	0.479 / 0.617	0.566 / 0.664	0.394 / 0.530	**0.559 / 0.675**
	MCC	0.164 / 0.296	0.119 / 0.320	0.198 / 0.284	**0.195 / 0.356**
	Jaccard index	0.372 / 0.447	0.458 / 0.498	0.305 / 0.361	**0.453 / 0.510**

Table A.2: Results on real data for the detection module using the fine-tuned Detectron2 on the converted images and generated pseudo-labels. In parenthesis are the relative percentage differences concerning the baseline.

	Metrics	No fine-tuning (baseline) [68]	CUT	CUT + Converted CBIR
Example based	**Accuracy**	0.280	0.324	0.312
	Precision	0.386	0.348 **(-9.8%)**	0.336 **(-3.4%)**
	Recall	0.517	0.824 **(+59.4%)**	0.810 **(+56.7%)**
	F1-score	0.442	0.489 **(+10.6%)**	0.475 **(+7.5%)**
	Hamming loss	0.343	0.456	0.474
Label based	**Accuracy**	0.657	0.544	0.526
	Precision	0.292 / 0.390	0.303 / 0.348 **(+3.8%/-10.8%)**	0.286 / 0.336 **(-2.1%/-13.8%)**
	Recall	0.438 / 0.521	0.659 / 0.825 **(+50.5%/+58.3%)**	0.638 / 0.811 **(+45.7%/+55.7%)**
	F1-score	0.307 / 0.446	0.382 / 0.489 **(+24.4%/+9.6%)**	0.364 / 0.475 **(+18.6%/+6.5%)**
	MCC	0.134 / 0.210	0.105 / 0.244	0.068 / 0.215
	Jaccard index	0.221 / 0.287	0.282 / 0.324	0.266 / 0.312

Table A.3: Results on real data for the detection module using the fine-tuned Detectron2 on the converted images and generated pseudo-labels when applying the global NMS post-processing step. In parenthesis are the relative percentage differences concerning the baseline.

		Applying global NMS		
	Metrics	**No fine-tuning (baseline) [68]**	**CUT**	**CUT + Converted CBIR**
Example based	**Accuracy**	0.244	0.331	0.326
	Precision	0.426	0.371 **(-12.9%)**	0.366 **(-14.1%)**
	Recall	0.356	0.751 **(+111%)**	0.743 **(+108.7%)**
	F1-score	0.388	0.497 **(+28.1%)**	0.490 **(+26.3%)**
	Hamming loss	0.291	0.402	0.409
Label based	**Accuracy**	0.709	0.598	0.591
	Precision	0.295 / 0.438	0.333 / 0.371 **(+12.9%/-15.3%)**	0.307 / 0.366 **(+4.1%/-16.4%)**
	Recall	0.322 / 0.359	0.601 / 0.751 **(+86.6%/+109.2%)**	0.582 / 0.742 **(+80.7%/+106.7%)**
	F1-score	0.264 / 0.395	0.380 / 0.497 **(+43.9%/+25.8%)**	0.363 / 0.490 **(+37.5%/+24.1%)**
	MCC	0.154 / 0.207	0.157 / 0.260	0.128 / 0.247
	Jaccard index	0.199 / 0.246	0.285 / 0.331	0.271 / 0.325

References

[1] Vardan Agarwal. Different types of normalization in tensorflow. https://towardsdatascience.com/different-types-of-normalization-in-tensorflow-dac60396efb0, 2020. [Accessed: 03-01-2021].

[2] Md. Zahangir Alom, Tarek Taha, Chris Yakopcic, Stefan Westberg, Paheding Sidike, Mst Nasrin, Mahmudul Hasan, Brian Essen, Abdul Awwal, and Vijayan Asari. A state-of-the-art survey on deep learning theory and architectures. *Electronics*, 8:292, 03 2019.

[3] Afshine Amidi and Shervine Amidi. Cs 229 - machine learning - deep learning cheatsheet. https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-deep-learning. [Accessed: 04-01-2021].

[4] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[5] 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.

[6] Konstantinos Bousmalis, George Trigeorgis, Nathan Silberman, Dilip Krishnan, and Dumitru Erhan. Domain separation networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 343–351, 2016.

[7] Agisilaos Chartsias, Thomas Joyce, Rohan Dharmakumar, and Sotirios A Tsaftaris. Adversarial image synthesis for unpaired multi-modal cardiac data. In *International workshop on simulation and synthesis in medical imaging*, pages 3–13. Springer, 2017.

[8] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[9] Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8789–8797, 2018.

[10] Sumit Chopra, Suhrid Balakrishnan, and Raghuraman Gopalan. Dlid: Deep learning for domain adaptation by interpolating between domains. In *ICML workshop on challenges in representation learning*, volume 2. Citeseer, 2013.

[11] 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. 06 2016.

[12] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[13] Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig. Virtual worlds as proxy for multi-object tracking analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4340–4349, 2016.

[14] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In *International conference on machine learning*, pages 1180–1189. PMLR, 2015.

[15] 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.

[16] LA Gatys, AS Ecker, and M Bethge. A neural algorithm of artistic style. In *16th Annual Meeting of the Vision Sciences Society (VSS 2016)*, page 326. Scholar One, Inc., 2016.

[17] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[18] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.

[19] 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.

[20] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[21] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

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

[23] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27:2672–2680, 2014.

[24] Gregory Gundersen. The reparameterization trick. http://gregorygundersen.com/blog/2018/04/29/reparameterization/, 2018. [Accessed: 03-01-2021].

[25] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017.

[26] 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.

[27] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *NIPS*, 2017.

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

[29] Branislav Holländer. Deep domain adaptation in computer vision. https://towardsdatascience.com/deep-domain-adaptation-in-computer-vision-8da398d3167f, 2019. [Accessed: 03-01-2021].

[30] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1501–1510, 2017.

[31] Xun Huang, Ming-Yu Liu, Serge Belongie, and Jan Kautz. Multimodal unsupervised image-to-image translation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 172–189, 2018.

[32] David H Hubel and Torsten N Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574, 1959.

[33] Naoto Inoue, Ryosuke Furuta, Toshihiko Yamasaki, and Kiyoharu Aizawa. Cross-domain weakly-supervised object detection through progressive domain adaptation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5001–5009, 2018.

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

[35] 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.

[36] Stephen James, Paul Wohlhart, Mrinal Kalakrishnan, Dmitry Kalashnikov, Alex Irpan, Julian Ibarz, Sergey Levine, Raia Hadsell, and Konstantinos Bousmalis. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12627–12637, 2019.

[37] Guoliang Kang, Lu Jiang, Yi Yang, and Alexander G Hauptmann. Contrastive adaptation network for unsupervised domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4893–4902, 2019.

[38] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019.

[39] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119, 2020.

[40] Taeksoo Kim, Moonsu Cha, Hyunsoo Kim, Jung Kwon Lee, and Jiwon Kim. Learning to discover cross-domain relations with generative adversarial networks. In *International Conference on Machine Learning*, pages 1857–1865. PMLR, 2017.

[41] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

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

[43] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. In *International conference on machine learning*, pages 1558–1566. PMLR, 2016.

[44] Yann Le Cun, Ofer Matan, Bernhard Boser, John S Denker, Don Henderson, Richard E Howard, Wayne Hubbard, LD Jacket, and Henry S Baird. Handwritten zip code recognition with multilayer networks. In *[1990] Proceedings. 10th International Conference on Pattern Recognition*, volume 2, pages 35–40. IEEE, 1990.

[45] Phuc H Le-Khac, Graham Healy, and Alan F Smeaton. Contrastive representation learning: A framework and review. *IEEE Access*, 2020.

[46] 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.

[47] Chuan Li and Michael Wand. Precomputed real-time texture synthesis with markovian generative adversarial networks. In *European conference on computer vision*, pages 702–716. Springer, 2016.

[48] Yanghao Li, Naiyan Wang, Jianping Shi, Jiaying Liu, and Xiaodi Hou. Revisiting batch normalization for practical domain adaptation. *Pattern Recognition*, 80, 03 2016.

[49] 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.

[50] Ming-Yu Liu and Oncel Tuzel. Coupled generative adversarial networks. In *Advances in neural information processing systems*, pages 469–477, 2016.

[51] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[52] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. *arXiv preprint arXiv:1611.02163*, 2016.

[53] Marvin Minsky and Seymour A Papert. *Perceptrons: An introduction to computational geometry*. MIT press, 2017.

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

[55] Zak Murez, Soheil Kolouri, David Kriegman, Ravi Ramamoorthi, and Kyungnam Kim. Image to image translation for domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4500–4509, 2018.

[56] Zak Murez, Soheil Kolouri, David Kriegman, Ravi Ramamoorthi, and Kyungnam Kim. Image to image translation for domain adaptation. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4500–4509. IEEE, 2018.

[57] John Nash. Non-cooperative games. *Annals of mathematics*, pages 286–295, 1951.

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

[59] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016.

[60] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[61] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[62] Taesung Park, Alexei A Efros, Richard Zhang, and Jun-Yan Zhu. Contrastive learning for unpaired image-to-image translation. In *European Conference on Computer Vision*, pages 319–345. Springer, 2020.

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

[64] Nithin Raghavan, Punarjay Chakravarty, and Shubham Shrivastava. Sim2real for self-supervised monocular depth and segmentation. *arXiv preprint arXiv:2012.00238*, 2020.

[65] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.

[66] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39, 06 2015.

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

[68] Isabel Rio-Torto, Ana Teresa Campaniço, António Pereira, Luís F Teixeira, and Vítor Filipe. Automatic quality inspection in the automotive industry: a hierarchical approach using simulated data. In *2021 IEEE 8th International Conference on Industrial Engineering and Applications (ICIEA)*, pages 342–347. IEEE, 2021.

[69] 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.

[70] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3234–3243, 2016.

[71] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[72] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[73] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[74] Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting visual category models to new domains. In *European conference on computer vision*, pages 213–226. Springer, 2010.

[75] Tim Salimans, Ian J Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *NIPS*, 2016.

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

[77] Maximilian Seitzer. pytorch-fid: FID Score for PyTorch. https://github.com/mseitzer/pytorch-fid, August 2020. Version 0.1.1.

[78] Junghoon Seo, Seunghyun Jeon, and Taegyun Jeon. Domain adaptive generation of aircraft on satellite imagery via simulated and unsupervised learning. *arXiv e-prints*, pages arXiv–1806, 2018.

[79] Connor Shorten. Must-read papers on gans. https://towardsdatascience.com/must-read-papers-on-gans-b665bbae3317, 2019. [Accessed: 03-01-2021].

[80] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Joshua Susskind, Wenda Wang, and Russell Webb. Learning from simulated and unsupervised images through adversarial training. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2107–2116, 2017.

[81] K Simonyan and A Zisserman. Very deep convolutional networks for large-scale image recognition. 2015.

[82] 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.

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

[84] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[85] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In *International conference on artificial neural networks*, pages 270–279. Springer, 2018.

[86] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[87] Jonathan Tremblay, Aayush Prakash, David Acuna, Mark Brophy, Varun Jampani, Cem Anil, Thang To, Eric Cameracci, Shaad Boochoon, and Stan Birchfield. Training deep networks with synthetic data: Bridging the reality gap by domain randomization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 969–977, 2018.

[88] 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.

[89] 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.

[90] Eric Tzeng, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell. Deep domain confusion: Maximizing for domain invariance. 12 2014.

[91] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[92] Stanford University. Cs231n convolutional neural networks for visual recognition. https://cs231n.github.io/optimization-2/. [Accessed: 03-01-2021].

[93] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol, and Léon Bottou. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12), 2010.

[94] Mei Wang and Weihong Deng. Deep visual domain adaptation: A survey. *Neurocomputing*, 2018.

[95] Lilian Weng. Domain randomization for sim2real transfer. *lilianweng.github.io/lil-log*, 2019. [Accessed: 23-01-2021].

[96] Lilian Weng. From gan to wgan, 2019.

[97] Garrett Wilson and Diane J Cook. A survey of unsupervised deep domain adaptation. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 11(5):1–46, 2020.

[98] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019.

[99] Tong Xiao, Hongsheng Li, Wanli Ouyang, and Xiaogang Wang. Learning deep feature representations with domain guided dropout for person re-identification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1249–1258, 2016.

[100] Wen Xu, Jing He, and Yanfeng Shu. Transfer learning and deep domain adaptation. In *Advances in Deep Learning*. IntechOpen, 2020.

[101] 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.

[102] Zhilin Yang, Junjie Hu, Ruslan Salakhutdinov, and William Cohen. Semi-supervised qa with generative domain-adaptive nets. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1040–1050, 2017.

[103] 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.

[104] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[105] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[106] Shengyu Zhao, Zhijian Liu, Ji Lin, Jun-Yan Zhu, and Song Han. Differentiable augmentation for data-efficient gan training. *Advances in Neural Information Processing Systems*, 33, 2020.

[107] 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.