A Sim-to-Real Deformation Classification Pipeline using Data Augmentation and Domain Adaptation

by

Joel Sol
B.A.Sc., University of British Columbia-Okanagan, 2022

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF APPLIED SCIENCE

in the Department of Electrical and Computer Engineering

A Sim-to-Real Deformation Classification Pipeline using Data Augmentation and Domain Adaptation

by

Joel Sol
B.A.Sc., University of British Columbia-Okanagan, 2022

Supervisory Committee

Dr. Homayoun Najjaran, Supervisor

(Department of Mechanical Engineering)

Dr. Stephen W. Neville, Supervisory Committee Member

(Department of Electrical and Computer Engineering)

ABSTRACT

Geometrical quality assurance is critical for improving manufacturing time and cost. This is more inhibiting when human operators' visual or haptic assessment is necessary. Modern machine learning (ML) methods can solve this problem but require large datasets with diverse deformations. However, preparing those deformations using physical objects can be difficult and costly. This thesis uses Blender, an open-source simulation tool, to imitate object deformities and automate the preparation of synthetic datasets. The utility of these datasets is improved using two methods; data augmentation such as background randomization and domain adaptation networks. The background randomization approach provides a way to generalize the image distribution to various environments, whereas the domain-adapted approach provides a better-targeted distribution. This thesis showcases that synthetic data created in Blender can be effective for training deformation classification networks. The discrepancies between real and simulated environments can be mitigated to create models for sim-to-real deformation detection.

Table of Contents

List of Tables

List of Figures

Abbreviations

- **API**: Application Programming Interface
- **BG-20k**: Background 20k dataset
- **CAD**: Computer Aided Design
- **CADT**: Content-Adaptive Domain Transfer
- **CNN**: Convolutional Neural Network
- **DCGAN**: Deep Convolutional Generative Adversarial Network
- **DANN**: Domain-Adversarial Neural Network
- **DRANet**: Disentangling Representation Adaptation Network
- **GAN**: Generative Adversarial Network
- **HDRI**: High Dynamic Range Image
- **IDE**: Integrated Development Environment
- **ML**: Machine Learning
- **MNIST**: Modified National Institute of Standards and Technology database
- **NLP**: Natural Language Processing
- **PCA**: Principle Component Analysis
- **RGB**: Red-Green-Blue
- **RGB-D**: Red-Green-Blue-Depth
- **SGD**: Stochastic Gradient Descent

- **SWD**: Sliced Wasserstein Discrepancy
- **UDA**: Unsupervised Domain Adaptation
- **WGAN-GP**: Wasserstein Generative Adversarial Network - Gradient Penalty

ACKNOWLEDGEMENTS

I would like to thank all the wonderful smart people I've worked with in the ACIS lab in particular Matthew Tucsok, Sara Hatami, Jamil Fayyad, Amir Soufi and Shadi Alijani. Their contribution of encouragement and colloration throughout my Masters has elevated my understanding and abilities.

I'd also like to thank my supervisor Dr. Homayoun Najjaran for encouraging me to do my Masters as well as the wonderful opportunity to join his lab.

Lastly I must also thank my parents and grandma for their help and support throughout my life.

DEDICATION

I dedicate this achievement to my family for their support, encouragement and guidance throughout my life.

Chapter 1

Introduction

1.1 Motivation

The ever-growing complexity of machine learning models has created a need for vast quantities of high-quality data for training and evaluation. Acquiring, annotating, and managing real-world data is often prohibitively expensive and time-consuming.

Synthetic data generation emerges as a promising solution. This technique involves creating artificial data that closely mimics the characteristics of real-world data. By leveraging algorithms and statistical models, synthetic data generation can produce large, diverse datasets tailored to specific needs. In certain scenarios, synthetic data can even replace real-world data altogether, significantly reducing the time and resources required for model development.

However, directly using synthetic data for training models often leads to performance limitations. Synthetic data, while carefully crafted, may not fully capture the intricate complexities and subtle nuances that exist in the real world. This can lead to models that perform well on synthetic data but struggle to generalize effectively to real-world situations.

Bridging this gap between synthetic and real-world data is crucial to unlocking the full potential of synthetic data generation. This thesis contributes to this effort by exploring the application of domain generalization and adaptation techniques. These techniques aim to train models that can learn from synthetic data while simultaneously being able to adapt and perform well on unseen real-world data, even when the real-world data exhibits variations or deviations not explicitly captured in the synthetic data.

This thesis delves into the development of a comprehensive data generation pipeline specifically designed for classifying deformations in real-world objects. This pipeline utilizes Blender, a powerful 3D creation suite, to generate synthetic datasets containing diverse variations of object deformations. By leveraging machine learning techniques within the pipeline, the goal is to achieve accurate classification of deformations in real-world scenarios. This has the potential to revolutionize the efficiency and accuracy of visual geometrical quality assurance processes in manufacturing environments.

By addressing the limitations of synthetic data and applying it to a practical industrial application, this thesis aims to make a significant contribution to synthetic data utilization and creation, machine learning and industrial quality control through automated defect detection.

1.2 Contributions

In this thesis, a pipeline for synthetic image dataset generation and sim-to-real deformation detection of objects is developed. The proposed pipeline, illustrated in Figure 1.1, leverages a simulation environment and deep learning for deformation detection. A pipeline was implemented for synthetic data generation. Following this, there is a choice based on whether there is real-world data. The data augmentation approach is used when there is no real-world data and the domain adaptive method is used when there is real-world data. Both methods improve classification accuracy and usefulness of the synthetic data. A full implementation of the sim-to-real deformation detection pipeline was created and analyzed using Coca-Cola pop cans.

Using this pipeline, key contributions are made in the following:

- **Synthetic deformation dataset generation using Blender**
  A software program was written to use features and tools in Blender to generate synthetic data utilizing the BlendTorch [11] application programming interface (API). Key components include:
  - Procedural deformation from created shape keys
  - Procedural lighting setup using High Dynamic Range Image (HDRI) shaders
  - Stochastic camera positioning
  - Image rendering with different backgrounds

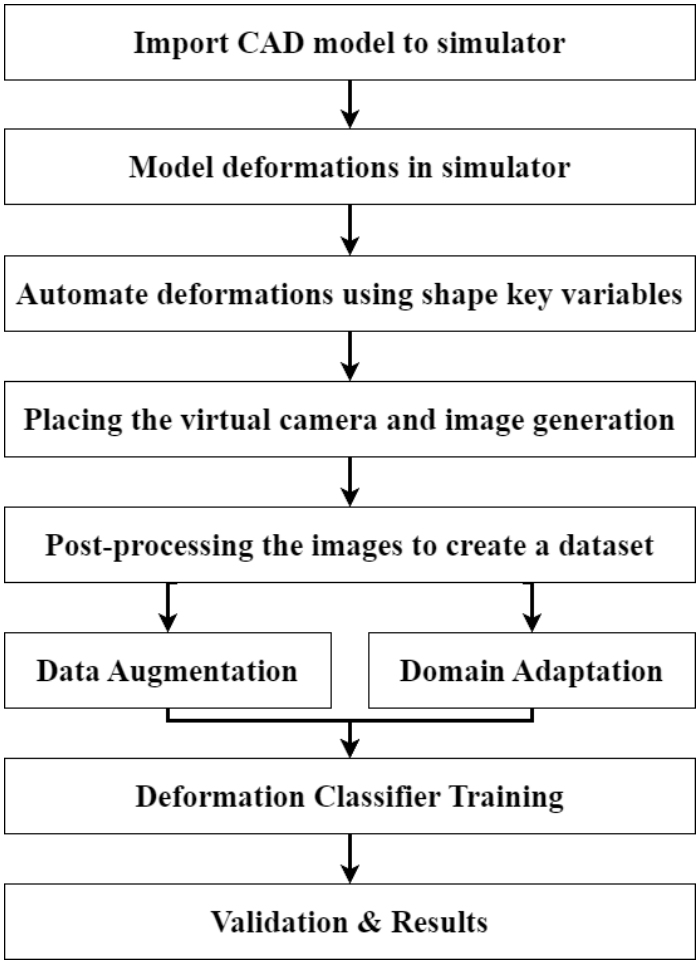


Figure 1.1: Proposed simulation-based deformation inspection pipeline.

- **Implementation of deformation detection network and analysis of data augmentation on sim-to-real deformation detection**
  A convolutional neural network (CNN) classifier is implemented to identify when an object is deformed. The effect of noise injection and randomized backgrounds on performance is analyzed.

- **Implementations and analysis of generative adversarial networks (GANs) on sim-to-real deformation detection**
  Several generative adversarial networks (GANs) were implemented and evaluated for generative performance in the task of converting synthetic data to mimic real-world data better. This includes detailed experimentation and analysis with a GAN, cycleGAN [77] and disentangling respresentation adaptation network (DRANet) [36].

1.3 Thesis Outline

Chapter 2 is a comprehensive literature review and provides a background on deformation detection in manufacturing, deformation detection methods, synthetic data generation, data augmentation, domain adaptation and generative adversarial networks (GANs). This chapter provides important background information relevant to key ideas present in each chapter.

Chapter 3 describes a new method for synthetic deformed object data generation using Blender. This chapter includes the simulation environment, creating deformations, shading and lighting, rendering, post-processing and showcases the generated datasets.

Chapter 4 uses the datasets generated in chapter 3 to train and generalize neural networks to classify objects as deformed or non-deformed. A real-world analysis is done to measure the "sim-to-real" gap in how well the model performs on real-world data.

Chapter 5 presents and analyzes three generative adversarial networks for domain adaptation. Images are generated by these networks and the performance of the domain adaptive method is compared with the generalization approach from chapter 4.

The impact and results from this method are discussed further.

Chapter 6 concludes the thesis and suggests future directions for this research.

Chapter 2

Literature Review

2.1 Geometric Quality Assurance Methods

The process of geometrical quality assurance based on coordinate measurement is an essential aspect of ensuring the accuracy and consistency of manufactured products in assembly lines. By utilizing this method, manufacturers can reduce production time and costs, while also increasing the overall quality of their precision products [48]. The accuracy of geometrical quality inspection in industrial settings heavily relies on the visual or haptic assessment of human operators, which can be time-consuming and prone to errors [14]. This can lead to significant delays and increased costs in the manufacturing process, making it a critical issue that needs to be addressed.

Heuristic methods, such as conventional photogrammetry [5], have been popular in industrial inspection, but they often lack generalizability and are limited by the complexity of the inspection task. As a result, there is a growing interest in the development of more advanced and automated inspection methods, such as computer vision and machine learning, that can provide more accurate and efficient solutions to quality assurance in industrial settings. Modern quality assurance methods powered by machine learning can solve this problem [21] but require large datasets with diverse deformations for training [63]. Generating a large and diverse dataset of deformations in physical objects can be a difficult and time-consuming task. Moreover, the physical limitations and constraints in object deformations might not cover all the possible scenarios, making it difficult to achieve a comprehensive dataset [7].

Therefore, alternative approaches, such as simulations, can provide a more efficient and versatile way of generating such datasets for machine learning-based quality assur-

ance methods [26]. While simulation is a viable option for creating training datasets, there have been efforts to explore data augmentation [65] or generation methods like generative adversarial networks (GANs) [55]. However, these methods are not as interpretable or controllable by expert knowledge, and may not provide the same level of meaningful diversity as simulations. Independent research has been conducted in this area, specifically aimed at automating the simulation of deformations [67]. As a result, this technology is now available for data synthesis applications. This advancement has enhanced the efficiency and accuracy of data synthesis, enabling researchers to generate realistic simulations of deformations more quickly and reliably.

While simulated input can be useful in training machine learning models, there is a risk that the model's performance will be degraded when transferred to a physical environment due to discrepancies with the real world. Therefore, it is important to consider the limitations of simulated input. One approach to improving the robustness of machine learning models in quality control applications is to add noise to images [74]. This can help the model to better generalize and perform well on unseen data in the real world. Adding noise augments the dataset and helps prevent networks from overfitting. However, it is important to carefully select the type and amount of noise to add, as too much noise can spoil the model's performance. Too much noise may obscure the data needed for the network to learn.

Geometrical assessment of manufacturing products has taken advantage of a wide range of measurement techniques to accurately identify and classify deformities in objects. Researchers have explored different methods for shape measurement, including the use of laser optics [51, 71]. However, this technique can be expensive and requires specialized equipment, making it less accessible for some applications. An alternative approach that has gained popularity in recent years is the use of RGB-D images for deformity detection and classification. These images capture both the colour and depth information of objects, allowing for a more comprehensive analysis of their geometries. By leveraging machine learning algorithms and computer vision techniques, researchers have been able to identify different types of deformations in objects, such as rigid, elastic, plastic, etc., from images [30].

On one hand, the advantage of using RGB-D images over laser optics is the ease and accessibility of data collection which allows for faster and more cost-effective data acquisition, making it a viable option for many industrial and manufacturing settings. On the other hand, using point-cloud laser optic measurement of objects in combination with image inputs has shown great accuracy suitable for real-time de-

formation detection in an industrial setting [64]. Another approach in this regard is using conventional heuristic photogrammetry techniques to obtain 2D readings from 3D computer-aided design (CAD) models and compare the outcome with real 2D images taken from the object [5]. However, the availability of an easier implementation able to address a wider range of applications is important. Therefore, exploiting raw CAD files and RGB images would be the priority over using other techniques discussed.

Machine learning manufacturing applications using image analysis have been a research topic for many years in the manufacturing and production industries [61]. For instance, in a recent paper, a computer vision module was designed to substitute human judgment in wear analysis as a part of manufacturing assessments [70]. Additionally, supervised learning although the predominant machine learning framework, is not the only one explored. A semi-supervised learning pipeline suggested by [39], uses auto-encoders to reduce the dependence of regular supervised-learning-based methods on huge labelled datasets. Although this improvement will facilitate the inclusion of unlabeled data in many cases, the total available data, whether labelled or not, might not always be enough. This necessitates using simulation-based data generation pipelines. The competence of such an approach has been verified in different problems of this field [10], and this thesis proposes a pipeline for another application whose efficiency is validated in a simulated environment.

Another factor to consider when working with machine vision is the camera position and calibration. To ensure faster and more accurate evaluations in the real world, it is imperative to reduce the sensitivity of machine vision to camera calibration [3].

2.2 Deformation Detection Methods

The detection of deformations has been a critical area of research within the field of computer vision, essential for applications ranging from structural health monitoring [13] to manufacturing and production quality control [54]. Over time, deformation detection approaches have significantly developed to improve the quality and capability of the proposed methods, leading to enhanced precision, reliability, and speed in identifying structural deformities. Existing approaches can be divided into two main streams: Classical approaches, characterized by their reliance on traditional image processing techniques and heuristic algorithms, and Deep Learning-based approaches. Classical approaches often involve edge detection, feature extraction, and template

matching, leveraging geometric and photometric invariants to identify deformations. Erkan et al. propose the integration of gravimetric and magnetic fields through deformation analysis for near-surface detection problems [15]. In [42], Mares et al. use colour encoding to measure transient 3D deformation. Similarly, in [57], bending deformation is detected through changes in colour ratios, observed by compact colour sensors. While these approaches are foundational and important to addressing the deformation detection problem, they often come with certain drawbacks such as their dependency on manual feature selection and engineering, and their sensitivity to noise.

Deep learning-based approaches, on the other hand, leverage the power of neural networks to learn and extract features relevant to deformation automatically. Zhao et al., present an approach for the deformation prediction method using a deep learning model during the numerical control machining process [73]. Tabernik et al. introduce a segmentation-based methodology for automating the detection of surface anomalies, including deformations [62]. Their strategy leveraged deep learning for segmentation and a decision network, enabling precise identification and analysis of irregularities on surfaces. One such deep learning approach is convolutional neural networks (CNNs).

CNNs are one of the most significant networks in the deep learning field [38]. CNNs are widely useful for a variety of vision-based tasks such as classification, object detection and segmentation. While CNNs are a mature technology, they are still widely useful and can be used to classify and detect deformation in images for industrial quality assurance.

As mentioned previously, classification through edge detection and feature matching methods have to be performed and created by humans manually. CNNs perform this automatically and learn features directly from raw image data. The internal architecture of CNNs is inspired by visual perception [29]. Artificial neurons in CNNs respond to kernels which can respond to various features. Activation functions only allow signals to fire if they exceed a certain threshold. This process is similar to how neural electrical signals function as well. CNN architectures use these convolutional kernel layers to extract features, pooling layers that reduce data size, and activation layers that introduce non-linearity. By stacking these layers, CNNs learn increasingly complex features, leading to accurate classification.

CNNs offer several advantages for identifying deformations in images. They eliminate the need for hand-crafted features from heuristic methods, saving time and effort. Additionally, they can learn complex features that classical methods might

miss. Furthermore, their convolutional layers preserve spatial information, making them robust to minor variations in object position or rotation. CNNs trained on large datasets often generalize well to unseen images.

These strengths make CNNs ideal for deformation detection in industrial quality control. By training a CNN on images with and without deformations, the model can learn to identify variations indicative of defects.

2.3 Synthetic Data

Synthetic data has emerged as a useful tool for machine learning and offers significant benefits, making it appealing for a wide variety of applications [41]. This can include vision, voice, natural language processing (NLP), healthcare and more. Key features that make synthetic data valuable can include privacy-preserving properties in the medical field [50], reducing the expense from not needing to label collected data and relative ease of creation compared to non-synthetic datasets. In scenarios where real-world data is scarce or limited, synthetic data generation techniques can be employed to artificially expand datasets. This enriched data pool allows for more robust training of ML models, mitigating issues like overfitting that can plague models trained on limited data.

There are a variety of ways to generate synthetic data. These can include stochastic processes [22], simulations (eg. game and graphics engines) [17], and deep learning methods [17].

One challenge with using synthetic data is assessing its quality. Metrics for assessing this can be split into three categories: fidelity, diversity and generalization [1]. Fidelity can be described as the quality of the samples, which can be done either computationally such as "Fréchet Inception Distance" [28] or by human evaluation. Often it is not enough to simply have just high-quality synthetic samples, there must also be diversity [17]. The data generated should show variety and cover the range of possible scenarios and edge cases present in the real world. Lastly, an important quality is generalization. It is essential that the synthetic data are not copies of the original data [17]. Samples with high fidelity and diversity do not guarantee synthetically generated samples are not copies of the original data [17]. Our generated data should not be replicas and be samples that support model training such that it can be effectively applied to new unseen real data.

2.4 Image Data Augmentation

Data augmentation is a valuable technique for overcoming limitations in real-world data acquisition. This approach involves artificially manipulating existing data to create a broader and more diverse training set [58]. This is done assuming that more data can be extracted from the original dataset through augmentation.

Geometric transformations are one type of data augmentation. Examples of this include scaling, rotating, cropping, flipping and translating images. It is important when considering some of these augmentation operations to think of their effects on data. For example, in the MNIST handwritten digits dataset, this could transform an image of a 6 into a 9. Noise injection is another technique that is useful for data augmentation. Moreno-Barea et al. [44] have shown adding noise to images can help CNNs learn more robust features.

Photometric transformations are another class of data augmentation. Modifying pixel intensities through brightness, contrast, and colour jittering helps the model cope with lighting changes and material variations that might occur in real-world settings. Kernel filters can be used to sharpen and blur images. Blurring images can lead to greater motion blur tolerance in trained networks. Sharpening images can result in the network understanding more details in the image. Random erasure is another method that selects a rectangular region within an image, fills it with random values, and improves image recognition issues with occlusion [75].

Data augmentation can also be extended to manipulate the background of the image. This is particularly useful in industrial settings where objects might be inspected against different backgrounds (e.g., conveyor belts, workstations). This technique was used by Patrizi et al. [47] for augmenting a dataset for automated sorting of littered waste. This was one of the methods used to augment datasets in this thesis. This method can allow models to learn to identify deformations irrespective of the surrounding environment.

The main benefit of data augmentation is that models are exposed to a wider range of realistic variations. This makes them less prone to overfitting and perform more accurately on unseen data. However, it's crucial to strike a balance. Over-augmentation can create unrealistic distortions or create inflated datasets which result in poor network performance.

2.5 Domain Adaptation

Unsupervised Domain Adaptation (UDA) aims to leverage labelled data in one or more source domains to perform well on the unlabeled target domain, despite differences in the data distribution between these domains [6]. This learning paradigm is crucial for various computer vision tasks where acquiring labelled data is limited or highly costly. UDA can be classified into discrepancy-based [9, 32, 69, 16] reconstruction-based [4], and adversarial-based approaches [52, 40, 68, 8]. Among these, adversarial methods have gained prominence due to their effectiveness in minimizing the domain discrepancy by framing the adaptation as a two-player game, where one network aims to distinguish between the source and target domains, while another tries to make the domains indistinguishable [18, 66]. This technique has been further refined by recent advancements such as the gradient-aligned domain adversarial network [52], which introduces novel mechanisms for enhancing the alignment of domain features, and the multi-level joint distribution alignment-based domain adaptation [40], demonstrating the potential of adversarial methods in bridging domain gaps more effectively.

The utilization of GANs in adversarial domain adaptation showcases significant advancements in domain transfer tasks. GANs, through their generator and discriminator networks, have been used for domain adaptation to either generate synthetic data that is indistinguishable from the target domain or learn a transformation from the source to the target domain. Notable implementations include the Domain-Adversarial Neural Network (DANN) [18], which introduces a gradient reversal layer to fool the domain classifier, effectively making the feature extractor domain-invariant, and CycleGAN [76], which has been used for image-to-image translation tasks, demonstrating the capability to adapt domains without paired examples. In recent studies, Yao et al. introduced a Shared Wasserstein adversarial learning approach [72], while Ge and Sadhu [20] utilized physics-informed learning and self-attention mechanisms to enhance GAN performance. Further enriching the domain, Wei et al. [68] proposed a self-training method via weight transmission between generators, and Cai et al. [8] explored the potential of symmetric consistency with cross-domain mixup. These contributions collectively illustrate the methodological advancements GANs bring to UDA, driving the field towards more effective adaptation techniques.

2.6 Generative Adversarial Networks

GANs are a unique two-network architecture that provides a way to learn deep representations without extensively annotated training data [12]. Originally proposed in 2014 by Goodfellow et al. [23], two networks are pitted against each other where one called the generator acts as an artist creating new images. The other network, the discriminator, acts as a critic analyzing real and generated images trying to spot the fake. Through this competition with each other, both networks improve. The generator learns to create increasingly real data and the discriminator becomes highly adept at spotting fakes.

Adversarially training these networks involves finding parameters for a discriminator to maximize its classification accuracy, while for the generator finding parameters to maximize discriminator confusion. These networks are trained separately where the parameters of one network are fixed while the other is updated. Goodfellow et al. [23] show there is a unique optimal discriminator for every fixed generator able to identify the generated data. An optimal generator has the discriminator predicting 0.5 for all samples indicating maximum confusion. In an ideal scenario, each network is trained until optimum parameters are reached. However, in practice, each network is trained for a small number of iterations with the generator updating simultaneous to the discriminator.

Training GANs is unstable and difficult which can be due to a variety of reasons. There can be difficulty in getting the generator and discriminator to converge [49], the generator can collapse providing similar samples for a given input [56] or the discriminator loss can quickly converge to zero [2]. These shortcomings can be addressed in several ways. One method is through deep convolutional generative adversarial network (DCGAN) architectures which came as a result of exploring CNN architectures used in computer vision [49]. Batch normalization [31] in both networks is also recommended for stability in deeper models. The leaky ReLU activation function in the discriminator has also been shown to give superior performance and training stability over other ReLU functions [49]. Other methods can include changing the architectures of the generator and discriminator, changing hyperparameters such as batch size, learning rate etc. and how many training iterations each network receives separately.

Despite these training drawbacks, GANs are widely useful in several tasks. For example, GANs have shown excellent performance in classification and regression

tasks [59], image synthesis [53], image-to-image translation [76], and super-resolution [34].

In the context of this thesis, generative adversarial networks provide an excellent method for understanding the distribution of real-world data. This understanding can adapt synthetic samples to mimic real-world data, effectively crossing the sim-to-real gap and increasing the accuracy of deformation detection. GANs can be specifically trained to adapt images relevant to the industrial setting.

2.7 Conclusion

The field of deformation detection has seen significant advancements in industrial quality assurance. However, approaches that rely on real-world data collection for training can be a time-consuming and expensive bottleneck.

While data augmentation and domain adaptation using GANs offer alternative solutions for leveraging existing data, they can lack the control and interpretability that simulations provide. The proposed pipeline provides a unique solution to this problem and addresses this gap by incorporating these established techniques within a simulation-based data generation pipeline.

This framework leverages the strengths of both worlds. Simulations are utilized to create a diverse and controlled dataset of deformations, overcoming the limitations of real-world data collection. This data is then augmented using established techniques to further enhance its generalizability. Alternatively, domain adaptation methods using GANs can be employed to ensure the model performs well on real-world data, despite potential differences between the simulated and physical environments. Finally, a machine learning model is trained on this enriched dataset to identify deformations in RGB images.

By drawing from these established fields, simulation-based data generation with existing data manipulation techniques, combined with domain adaptation offers a unique and potentially more efficient solution for deformation detection in industrial settings. This approach aims to improve the accuracy and efficiency of the process while maintaining control and interpretability.

Chapter 3

Data Generation and Dataset Preparation

This chapter introduces the data generation environment for creating synthetic data for deformation detection. Within this section, the Blender simulation environment, procedural deformation and post-processing techniques are discussed. The generated datasets are also shown. The data generated and discussed in this chapter are later used in Chapters 4 and 5.

3.1 Blender Environment

Blender is an open-source computer graphics, rendering, and simulation environment with a variety of tools that aid in the generation of synthetic visual data. Blender accepts most file types associated with computer-aided design (CAD) and a Python interface for automation of the visual dataset creation process. Blendtorch is a Python framework for integrating Blender with Pytorch for deep learning applications [11]. Blendtorch was used to integrate the Blender Python scripting environment into an integrated development environment (IDE) with the rest of the project so rendered images could be used to automate dataset generation. The Python interface allowed for the automation of creating datasets by automating camera placement, object tracking and procedural deformation of the CAD object. An initial proof of concept was done using the built-in Blender monkey head, Suzanne, to model deformation and dataset creation as well as basic classification. The object was selected as it has complex features. Later, a Coca-Cola pop can was used for the full sim-to-real

pipeline. The object was selected as it is easy to acquire a sizeable physical dataset for and easy to deform in simulation and real life to create the deformed class. The metallic nature of the object also presents an interesting classification challenge. For each object, the Blender environment was used to generate synthetic images of the deformed and non-deformed objects.

3.2 Creating Deformed Objects

Blender has a wide variety of tools available for creating deformations. The shape key in Blender can be assigned to a complex deformation and allows for interpolation between that deformed state and the original by setting a value between 0 and 1. This tool is especially helpful for creating new and different deformations as complex deformations can be mixed using the shape keys to create new and different ones. Specific deformation tools that were found to be most useful for creating these datasets were the displace and lattice deform modifiers to create specific deformations based on human expert information.

3.2.1 Blender Monkey Head "Suzanne" Deformation

Deformation of the Suzanne object was done by creating a mesh deform modifier for the object. To do so, another simpler lattice object was created outside of the Suzanne object. This lattice was then linked to the Suzanne object. By dragging the faces and vertices of the polygonal object, the Suzanne object would be pulled or pushed in the same manner. These deformations were set to shape keys in Blender which can be used to modify the extent of the deformation. 12 shape keys were arbitrarily chosen as the number of deformations created to provide some variety to the deformations. The extent of the deformation created by the shape keys was also arbitrarily chosen to be a uniform random value from 0.3 to 1. Shape key values below this threshold were more difficult to see and identify visually.

3.2.2 Pop Can Deformation

The deformation of the pop can object was done by combining a lattice deformation and a displace modifier. A simple cube lattice was created around the can to control major deformations coined as *crushes*, *pinches*, *folds*, *twists* and *crunches*. When the

lattice is deformed it controls the resulting deformation in the pop can. 12 lattice deform shape keys were created of the various macro deformation types. Movement and rotation deformations were applied to both the seal and tab of the pop cans and mapped to shape keys and make the cans appear open. In all samples, the tab and seal of the pop cans are set to a random state between fully open and fully closed. The tab is always set to a state more closed than the seal to prevent impossible geometries and part clipping.

Smaller deformations on the surface of the can were created by using a displace modifier. Vertex groups were used to limit these deformations to the side of the can leaving the top and bottom unaffected. The displace modifier slightly inflates the object and applies a texture to deform the surface. For the pop can object, the most realistic texture was a hard *Stucci* texture scaled to look similar to the sharp edges of a crinkled pop can. 3 different hard *Stucci* textures were created and applied to the sides of the pop can to deform the surface. Each *Stucci* surface texture was assigned a corresponding shape key. By combining the lattice deformation and displacement deformations, high-quality procedural deformations can be created. In the deformed class, 12 lattice deformations and 3 displacement deformations were combined. The displacement deformations were each assigned a randomized weight and combined in a way to ensure there was a procedural deformation on the surface of the object. For the lattice deformations, 3 or more types (*crush*, *pinch*, etc.) were randomly selected and combined to produce a wide variety of deformations to the same pop can and better match the deformations seen in the physical dataset. Each type was assigned a randomized weight to differ in the extent each lattice deformation type makes on the object. For each lattice deformation type, various shape keys were mixed and selected. The result is a realistic, procedural deformation.

3.3 Shading and Lighting

In Blender, the shading editor plays a crucial role in defining the visual appearance of objects. It functions like a digital artist's palette, offering various tools and options to control the materials, textures, and lighting applied to 3D models.

One key aspect of the shading editor is defining the material properties. It allows users to specify whether an object is metallic, glassy, or a combination of different materials. This significantly impacts how light interacts with the object, influencing its overall look and feel. Additionally, the editor provides access to a vast library of

textures, which can be used to add realistic details like wood grain, fabric patterns, or even complex procedural textures like clouds or fire.

Furthermore, the shading editor offers powerful tools for manipulating light and shadows. Users can create various light sources, such as point lights, spotlights, and area lights, to illuminate the scene. Additionally, tools like the environment texture slot allow users to incorporate a High Dynamic Range Image (HDRI) as the environment background, realistically simulating real-world lighting conditions.

The Suzanne object material properties were set to be metallic and feature a surface similar to brushed aluminum. Visual methods of deformation detection often struggle with picking up features of metallic and shiny objects which is why the object was shaded in this manner. Many processes in factories have components that have similar surface qualities, so the object was textured similarly to reflect that. The lighting environment utilized hand-placed point light sources.

The pop can had a more complex shader environment than the Suzanne object. A UV map was created to unwrap the vertices of the object mesh onto a 2D plane. From there a Coca-Cola can label was placed on the side of the can. For the label area, a shiny, slightly reflective shader was created to mimic the look of a real-world pop can. The top and bottom of the can were given a metallic shader mimicking aluminum material properties. A dynamic procedural lighting environment was created for this object using the world shading module. An EXR file was randomly selected and plugged into the HDRI lighting module. This was then given a random rotation to create realistic and different light and shadows on the can object. The HDRI lighting module also creates a background for the object when used. This was not desirable for the dataset so a background color was mixed with the HDRI lighting. This allowed the lighting to affect the object but only for camera rays hitting the object and a solid colour background for the object.

3.4 Rendering Images

Rendering was done by creating new cameras set up in polar coordinates around the object. The cameras were set to be aimed at the object's center of mass using built-in tools in Python. For each procedurally deformed object, 4 cameras were set up in a uniform random polar position according to certain parameters. The parameters for the camera positions for the Suzanne object are shown in Table 3.1 and the parameters for the pop can are in Table 3.2. In the tables, the camera position parameters are

given in polar coordinates θ, ϕ, r with the origin as the object's center of mass.

Table 3.1: Suzanne camera position parameters in Blender

	Camera			
	No. 1	No. 2	No.3	No.4
θ	$[20° - 70°]$	$[110° - 160°]$	$[200° - 250°]$	$[290° - 340°]$
ϕ	$[60° - 100°]$ for all cameras			
r	$[6m - 12m]$ for all cameras			

Table 3.2: Pop Can Camera position parameters in Blender

	Camera			
	No. 1	No. 2	No.3	No.4
θ	$[20°, 70°]$	$[110°, 160°]$	$[200°, 250°]$	$[290°, 340°]$
ϕ	$[50°, 70°]$ for all cameras			
r	$[0.3m - 0.45m]$ for all cameras			

The images were rendered using the offscreen-renderer available in BlendTorch. Depth images are possible in this pipeline but require the use of the Blender compositor and a different renderer available in BlendTorch. The Suzanne images were rendered as 1920x1080 RGB and the pop can images were rendered as 512x512 RGB. Both image datasets were later modified.

3.5 Post-Processing

The post-processing steps are essential for ensuring the datasets look correct and are useful for future steps. The Suzanne dataset and pop can datasets had different post-processing steps needed. However, one step needed for both datasets is gamma correction. The rendered images for both objects are created in the linear colour space and require pixel-wise gamma correction for the images to look correct. The formula used for pixel-wise gamma correction is shown in equation 3.1. A gamma value γ of 1/2.2 was used to correct the colour values. I_{in} represents each pixel in the input image and I_{out} is each pixel in the output image.

$$I_{\text{out}} = \left(\frac{I_{\text{in}}}{255.0}\right)^{\gamma} \tag{3.1}$$

3.5.1 Blender Monkey Head "Suzanne" Image Post-Processing

The Suzanne images were post-processed to reduce the image sizes to be more suitable for use with a neural network. The 1920x1080 RGB images were rescaled to 480x270 grayscale. This data transformation reduces the size of the data making it more suitable for machine learning reducing memory usage and improving training time. The dataset was duplicated with another copy having Gaussian noise added as described in equation 3.2. Gaussian noise can increase diversity in training data artificially and therefore model stability. The noise follows a normal distribution with a mean of 0 ($\mu = 0$) and a standard deviation of 0.1 ($\sigma = 0.1$).

$$\varphi(z) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(z-\mu)^2/(2\sigma^2)} \tag{3.2}$$

The final step in the Suzanne dataset post-processing was to combine the 4 camera views of each created object into a single dataset sample.

3.5.2 Coca-Cola Pop Can Image Post-Processing

The pop can dataset was generated with both a black background and a green background. The green background version was given additional post-processing to give the images backgrounds from the BG-20k dataset [37]. The dataset features 20,000 high-resolution background images excluding salient objects that are made for high-quality synthetic images. The green background images were rendered with an RGB value of (0, 255, 0) for BG-20k images to be green-screened. For the green screen post-processing to happen, a mask must be generated over the images. Using OpenCV, a mask was created capturing pixels with the color value of the greenscreen. Once the mask was created, morphological operations were applied to improve the quality. Opening and closing morphological operations were performed to ensure there were no holes in the generated mask where the new background would replace the object or green specs of the background remained. Examples of these operations are shown in Figures 3.1a and 3.1b. Lastly, a small erosion kernel was applied to ensure no green pixels remained on the border of the object, which would affect network training. An example of this operation is shown in Figure 3.1c. After these operations, the mask is

complete and a randomly selected image from the BG-20k dataset replaces the background. The BG-20k dataset was chosen for the backgrounds due to it being created for image matting tasks where foreground features are recognized from a background image. This is ideal for the use case of creating a generalized background with the important pop can features in the foreground.

(a) Opening Morphological Operation

(b) Closing Morphological Operation

(c) Erosion Morphological Operation

Figure 3.1: Examples of morphological transformations from OpenCV used to create green-screen mask [46]

3.6 Resulting Datasets

In this research human evaluation is sufficient as the full sim-to-real deformation pipeline is a proof of concept but for full commercial implementation, the pipeline would benefit from computational fidelity metrics.

In the case of this thesis, this means our synthetic data from Blender should have enough variety in deformations to mimic the real world.

4 datasets were created using the steps detailed in the previous sections. In

addition to these 4 synthetic datasets, 1 real-world physical dataset was created. Each of the resulting datasets is discussed below.

- **Suzanne Monkey Head:** 12000 images of the Suzanne monkey head object were created, 6000 of which were of the non-deformed object and 6000 contained a deformation. A sample of the non-deformed object is shown in Figure 3.2a and a deformed sample in Figure 3.2b. In the deformed figure, the right eyebrow was pulled further out from the object. Due to the post-processing step that combines each of the 4 images into a single sample, this dataset contains 3000 samples. This dataset will be used to provide baseline results for a deformation detection network on synthetic data.

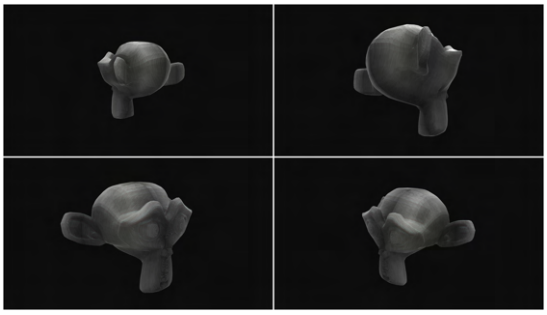
- **Suzanne Monkey Head with Gaussian Noise:** This dataset consists of the same images as the Suzanne dataset with Gaussian noise applied. Examples of the non-deformed object are shown in Figure 3.2c and Figure 3.2d. Due to the post-processing step that combines each of the 4 images into a single sample, this dataset also contains 3000 samples. This dataset will be used to analyze the performance of a deformation detection network in the presence of noise.

- **Coca-Cola Pop Can - Black Background:** 6000 images of a pop can were rendered for this dataset consisting of 3000 deformed pop can images and 3000 non-deformed images. Images for this dataset were rendered with a solid black background around the object. Samples from this dataset are shown in Figure 3.3b for the non-deformed object and Figure 3.3a for the deformed object. This dataset was created to provide a baseline for sim-to-real deformation detection.

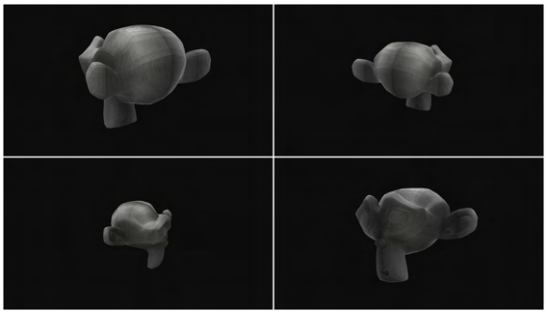
- **Coca-Cola Pop Can - BG-20k:** 6000 images of a pop can were rendered for this dataset consisting of 3000 deformed pop can images and 3000 non-deformed images. Images for this dataset feature backgrounds from the BG-20k dataset. Samples from this dataset are shown in Figure 3.3d for the non-deformed object and 3.3c for the deformed object. This dataset was created to provide a data augmentation approach for deformation detection.

- **Pop Can - Physical Dataset:** A physical dataset was created as well to provide a real-world object to test our synthetic data against and measure the sim-to-real performance gap. Like the synthetic dataset, 4 camera views of each deformed object were used. 40 deformed cans were used to create the 160

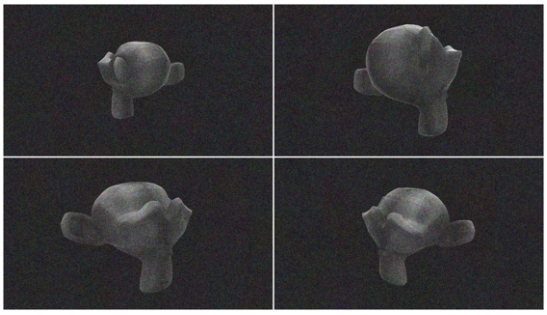
images making up the deformed class. 3 cans were used for the non-deformed class with 5 images from each camera view quadrant. These 60 images are used for the non-deformed class for the physical dataset. The images were taken with a smartphone camera in a 1:1 aspect ratio with positions and angles similar to those set up in the synthetic dataset. Examples of the real-world data are shown in Figure 3.3e and Figure 3.3f.



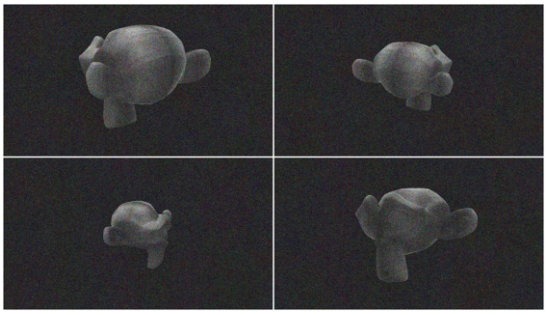
(a) Non-deformed object images



(b) Deformed object images

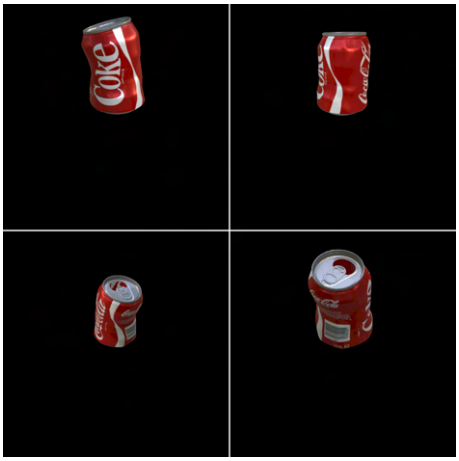


(c) Non-deformed object images with Gaussian noise

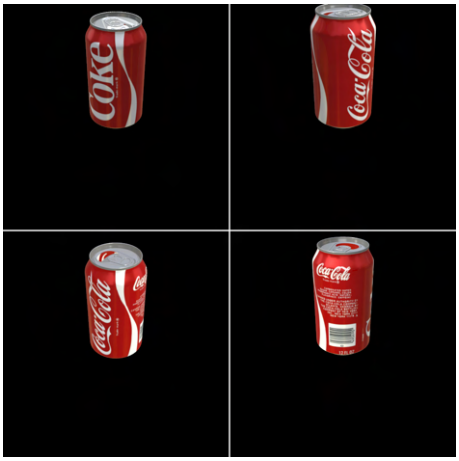


(d) Deformed object images with Gaussian noise

Figure 3.2: Suzanne object images in the Blender environment



(a) Synthetic Deformed Can Black Background



(b) Synthetic Non-deformed Can Black Background

(c) Synthetic Deformed Can BG-20k Background

(d) Synthetic Non-Deformed Can BG-20k Background

(e) Real Deformed Can

(f) Real Non-Deformed Can

Figure 3.3: Synthetic and Real Can Dataset Examples

Chapter 4

Data Augmentation

This chapter explores the potential of synthetic data augmentation for deformation detection in object condition assessment. The chapter is divided into two main sections. The first section details the creation and evaluation of two neural networks for classifying deformed Suzanne objects from synthetic images. The purpose of doing this is to ensure that synthetic deformation in Blender can be detected by a neural network before moving to a full sim-to-real experiment. The second section explores the challenges of sim-to-real transfer in the soda can dataset and uses background generalization as a method to improve real-world applicability.

4.1 Synthetic Validation

This section explores the development and evaluation of two neural networks designed for deformation detection of the Suzanne object. The primary objective was to determine which network architecture achieved superior performance and robustness in identifying deformations. The section delves into the design choices, training procedures, and validation results of both networks, ultimately highlighting the advantages of the residual block network in this application.

4.1.1 Deformation Detection Networks

Two simple neural networks were created to see if deformations could be detected in the generated synthetic images. Each of the networks used 4 images of a single object sample as input, and output a binary classification if the object was deformed or not.

One of the networks was based on residual blocks with the architecture shown in Figures 4.1 and 4.2. The residual block network starts with 4 different streams, one for each 480×270 gray-scale picture. Each stream begins with a 2D convolution with 128 features, a 3×3 kernel size, and a stride of 3. This layer is used to find features in the image as well as reduce parameter size using the stride value of 3. Three residual blocks follow the convolution layer to find and detect features and recognize any deformations in the object in the image. Each residual block contains a convolutional layer with batch normalization and ReLU activation followed by another convolution. This output then is added to the original input. Each residual block is padded to remain the same size as its initial input. Another convolution layer is then used to reduce the channel size to 2 as well as a 3×3 max-pooling to condense the total size needed for the linear layers. All four of the separate image streams are then flattened and concatenated together. The first linear layer reduces the size from 6360 to 128 and the final linear layer reduces 128 to 2 for binary classification. The activation function used for the output was sigmoid as it has been shown to be the optimal activation function in feed-forward binary classification networks [45].

The second network as shown in Figure 4.3 was created to have a similar architecture and number of convolution layers as the residual block network. The network features the same 4 image streams and the same first convolution layer. Instead of residual blocks, 6 convolutional layers using batch normalization and ReLU activation were used. These layers maintained a feature size of 128 and used a 3×3 kernel and a stride of 1. The same image size was maintained by padding the image. The remaining layers are the same as the residual block network with the same convolution, max-pooling, and linear layers. The final activation function, sigmoid, was used as well.

4.1.2 Network Training

Both networks were trained similarly. The Adam optimizer was used and the given loss function for the networks was binary cross entropy. A learning rate of 0.001 was used and a scheduler was used to decay the learning rate across the 15 training epochs. Different milestones for the learning rate decay were used for each network. The residual block network decay milestones were epochs 3, 7, and 11. The convolution network decay milestones were epochs 4, 7, and 9. These milestones were selected based on several trial runs by visualizing when the validation loss was no longer

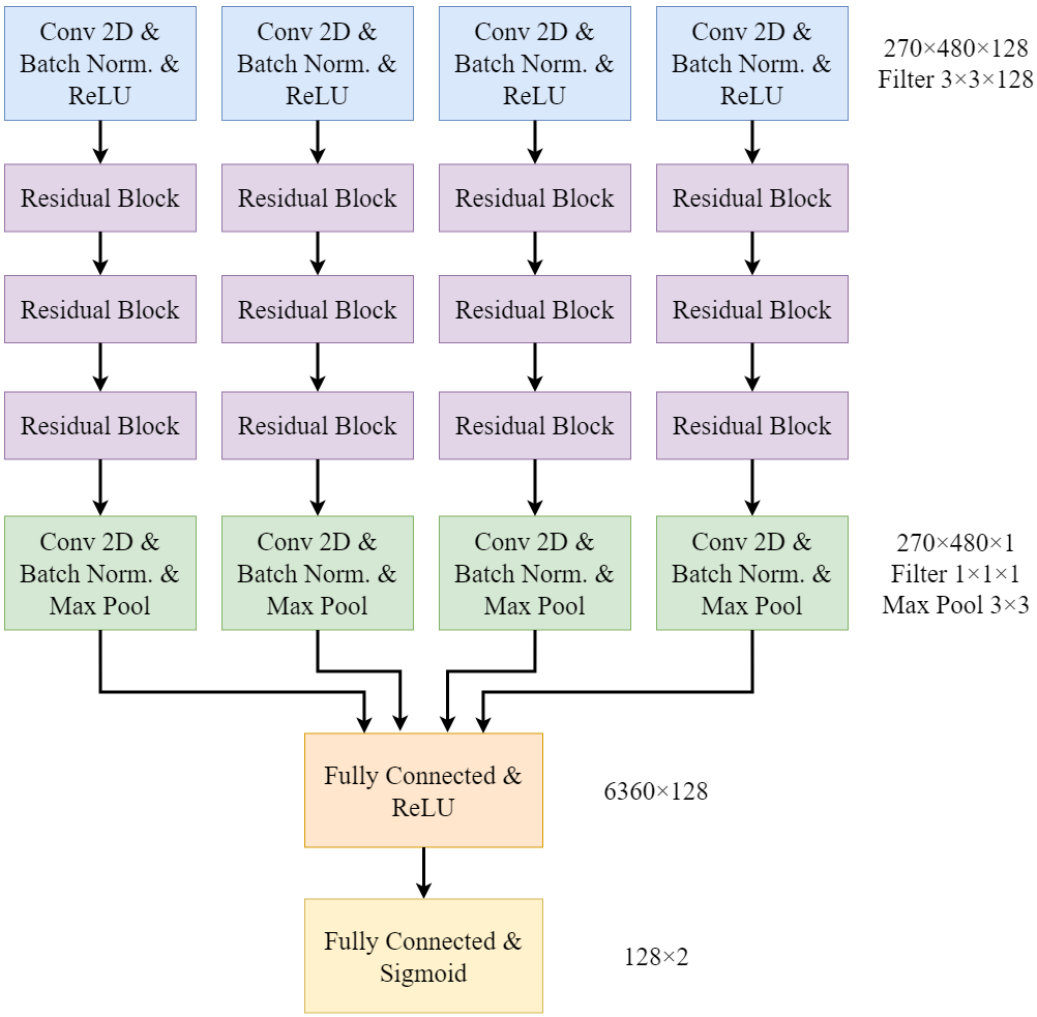


Figure 4.1: Architecture of the residual block network.

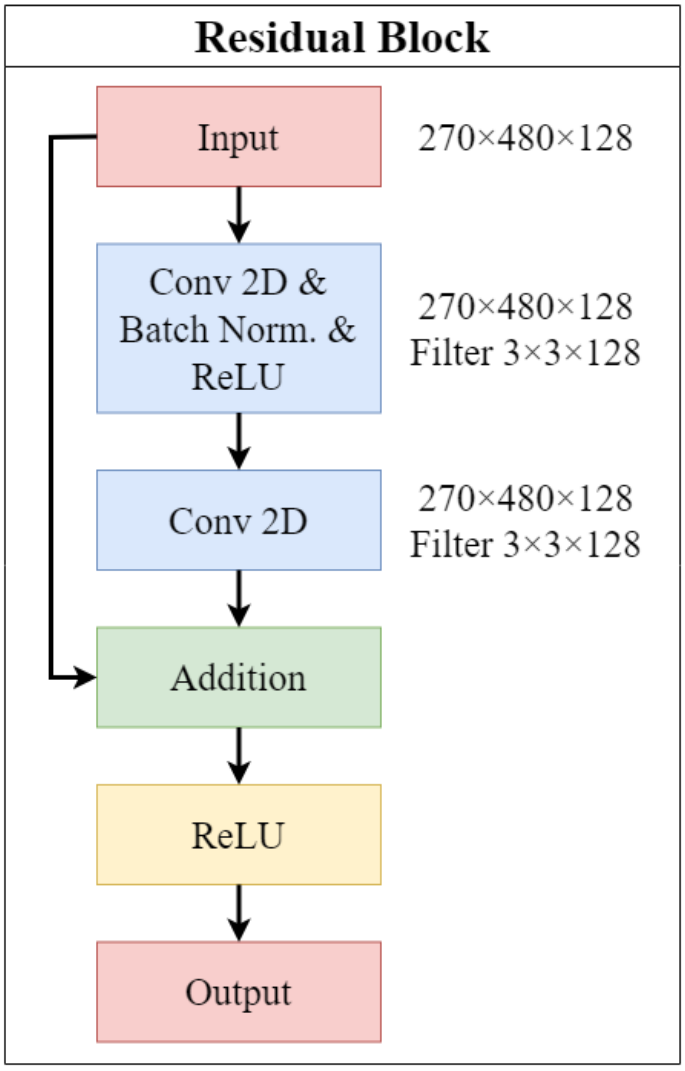


Figure 4.2: Internal workflow of a residual block.

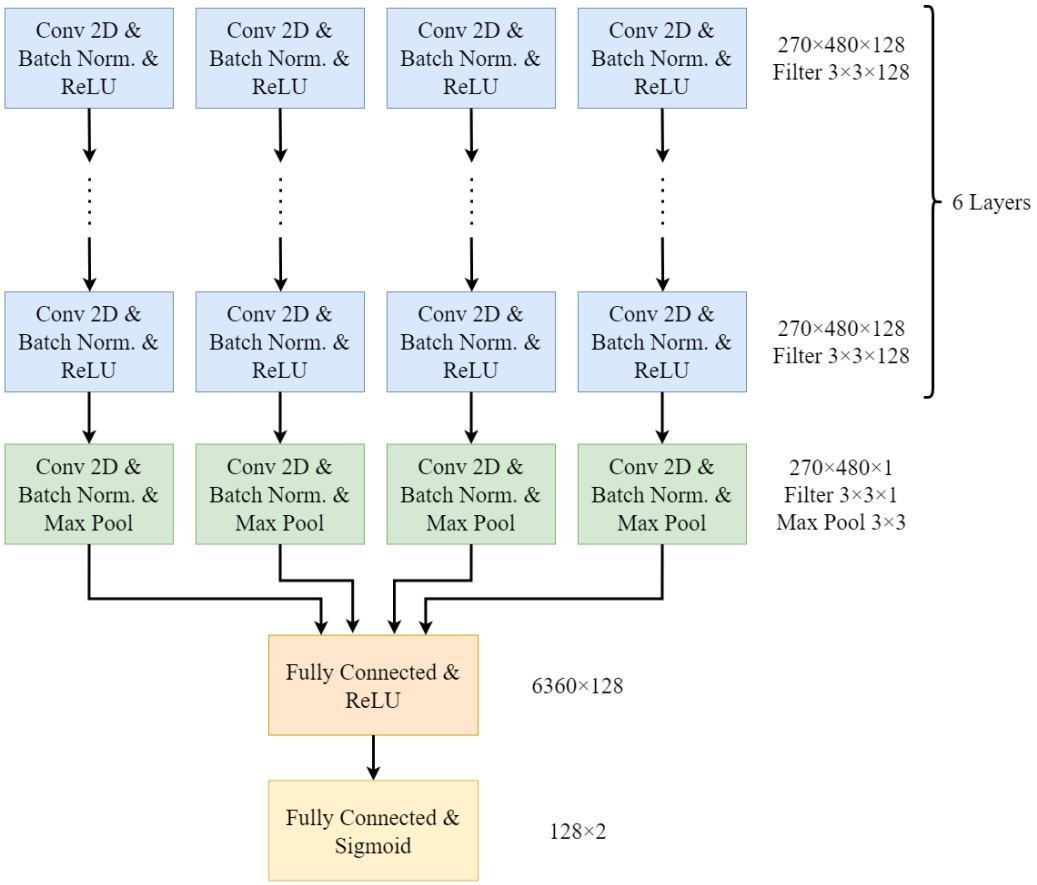


Figure 4.3: Architecture of the convolutional neural network.

decaying. When this occurred instead of early stopping, the learning rate decayed. During training the residual block network seemed easier to train and less impacted by hyperparameters such as learning rate, decay milestones and batch size. The networks were able to learn effectively using these specified parameters and the results of this training are discussed in section 4.1.3.

4.1.3 Validation Results

Overall, the residual block network demonstrated superior performance compared to the convolutional neural network (CNN) based on accuracy, a common metric used to evaluate classification models. The results are visualized in Figure 4.4. The residual network achieved validation accuracies of 90.9% and 85.6% for the normal and Gaussian noise datasets, respectively. In comparison, the CNN's accuracies were slightly lower at 90.2% and 85.8% for the normal and Gaussian noise datasets, respectively.

To delve deeper into the performance of the residual block network, the confusion matrices were analyzed, which are illustrated in Figure 4.5. Confusion matrices provide a detailed view of how the model classified the data points. In this case, the rows represent the predicted labels (deformed or non-deformed), and the columns represent the actual labels. Ideally, the values should be high along the diagonal, indicating accurate classifications.

Figure 4.5a shows the confusion matrix for the clean dataset. Figure 4.5b presents the confusion matrix for the noisy dataset, which is slightly less accurate compared to the clean dataset. This suggests that the presence of noise introduced some challenges for the model, particularly in correctly classifying non-deformed objects.

While accuracy is a valuable metric, it is essential to consider other performance measures to gain a more comprehensive understanding of the model's effectiveness. Therefore, the F1-score, recall, and precision for the residual network on both datasets were calculated as shown in Table 4.1.

Recall measures the proportion of actual positive cases (deformed objects) correctly identified by the model. The model achieved recall values of 0.8816 and 0.9248 for the clean and noisy datasets, respectively. This suggests that the model was effective in identifying most of the deformed objects, even in the presence of noise.

Precision, on the other hand, reflects the proportion of the model's positive predictions that were truly deformed. The model exhibited precision values of 0.9348 and 0.8164 for the clean and noisy datasets, respectively. This indicates that the

majority of the objects classified as deformed by the model were indeed deformed, especially in the clean dataset.

F1-score is a harmonic mean of precision and recall, providing a balance between the two metrics. The model achieved F1-scores of 0.9074 and 0.8672 for the clean and noisy datasets, respectively, indicating precision and recall are balanced.

The validation results demonstrate that the residual block network achieved promising performance in classifying deformed Suzanne objects from synthetic images. The network exhibited high accuracy recall and precision values with a balanced F1-score even in the presence of noise. These findings showcased that the images produced using the data generation pipeline are sufficient to create a deformation classifier and can proceed with a full sim-to-real experiment as outlined in section 4.2.

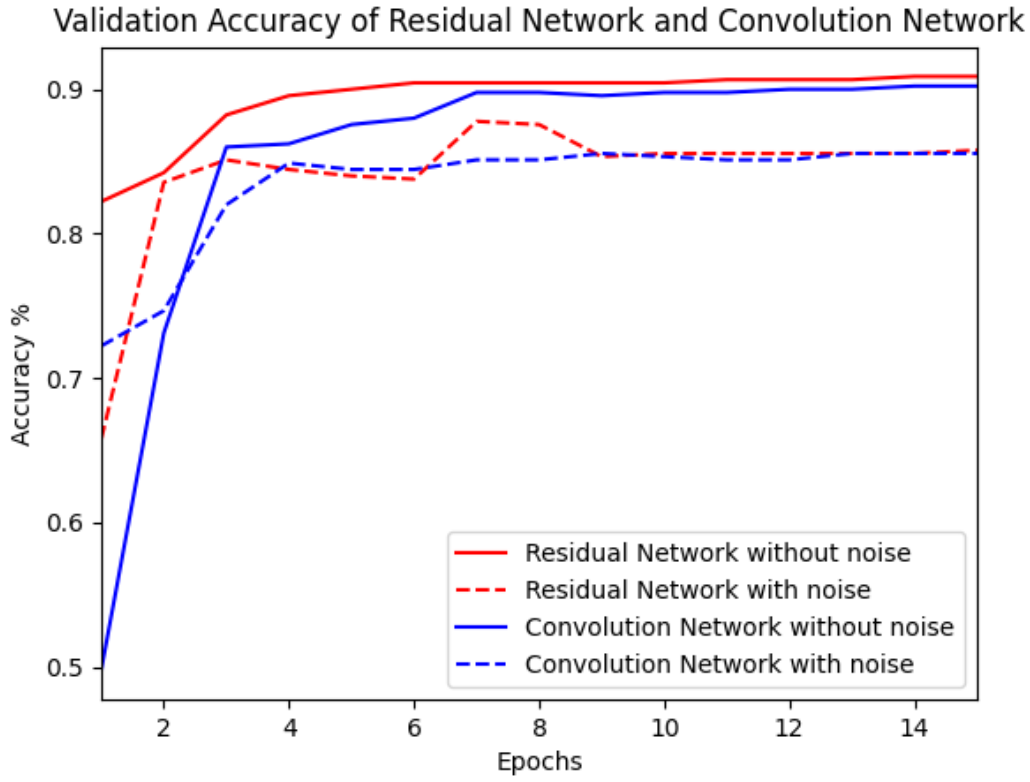
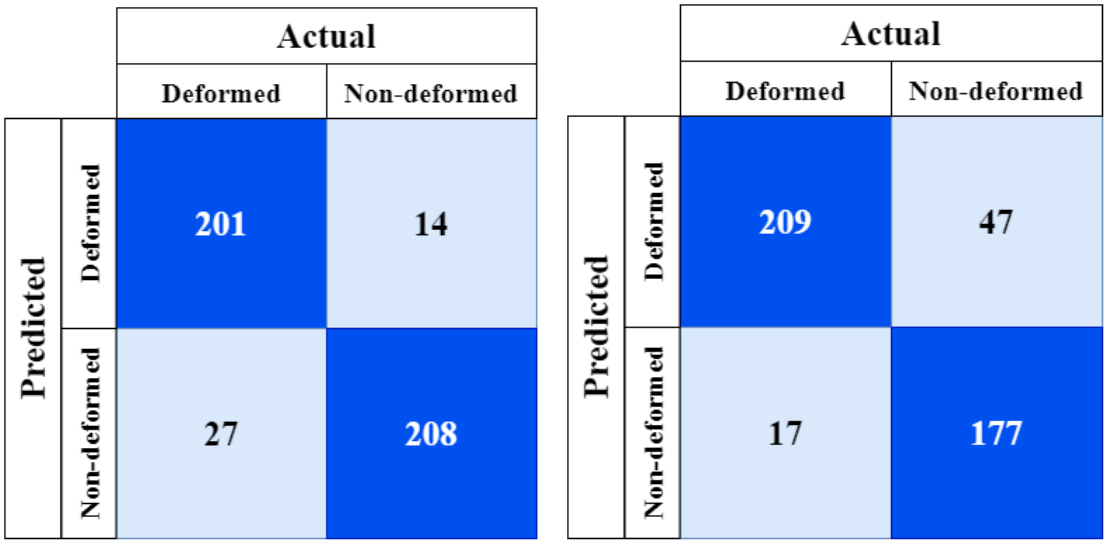


Figure 4.4: Validation accuracy of residual block network and convolutional neural network for both datasets



(a) Clean dataset (b) Noisy dataset

Figure 4.5: Confusion matrices for the residual block model

Table 4.1: Performance measure of the residual network

Metrics	Training dataset	
	Clean	Noisy
Accuracy	0.9089	0.8578
F_1	0.9074	0.8672
Recall	0.8816	0.9248
Precision	0.9349	0.8164

4.2 Sim-to-Real Deformation Detection

This section investigates the challenges and potential solutions for overcoming the sim-to-real gap in deformation detection using the soda can datasets generated in chapter 3. It explores the use of a modified VGG-16 network trained on the synthetic pop can datasets and evaluated on a real-world scenario. The section delves into the network architecture, training process, and the impact of background generalization on performance. The findings highlight the limitations of a simple black background and the benefits of employing diverse backgrounds like the BG-20k dataset to improve generalizability and reduce the sim-to-real gap.

4.2.1 Sim-to-Real Network and Training

The network used for crossing the sim-to-real was a modified version of pre-trained VGG-16. The VGG-16 model architecture [60], is widely useful in various computer vision tasks such as image classification, object detection, and image segmentation. The simplicity of the architecture makes it easily adaptable and effective, making it a good network to use as a starting benchmark. The architecture of the original VGG-16 network is shown in Figure 4.6. The original VGG-16 model consists of 13 convolutional layers and 3 fully connected layers. The convolutional layers use 3x3 filters with a stride of 1 and padding to maintain the spatial dimensions of the input. Max pooling layers use a 2x2 filter with a stride of 2 and are used after every 2-3 convolution layers. This reduces the spatial dimensions by half.

The VGG-16 network used for this task has been adapted. The input image sizes (512x512x3) are different than the sizes used by the original VGG-16 (224x224x3). The network has the same convolution and max pooling layers, but the fully connected layers have been changed. After the convolution layers, an adaptive average pooling layer is used to fix the output size. This is then followed by 3 fully connected layers with ReLU activation and a dropout layer. The 3 fully connected layers go to a one-hot output with a sigmoid activation function. The sigmoid activation function has been shown to be the optimal activation function in feed-forward binary classification networks [45].

While training, all convolution feature layers except the final convolutional layer were frozen in order to preserve learned features from its previous training on ImageNet-1k. This pre-training allows the model to learn general features of images, which can be transferred to the task of classifying the state of cans. The network was trained

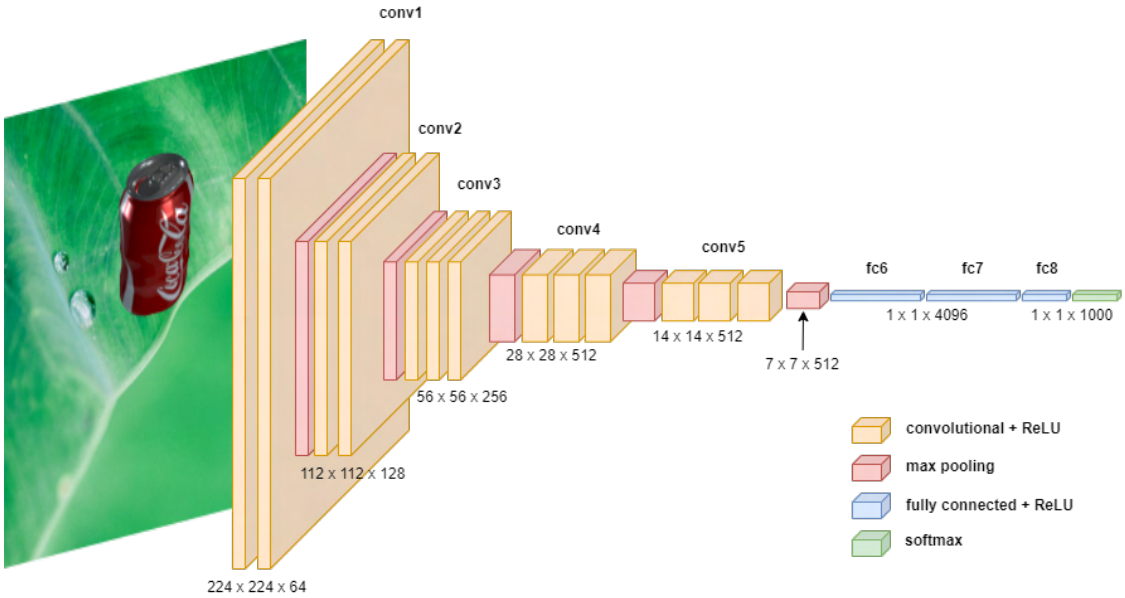


Figure 4.6: Original VGG-16 architecture for ImageNet-1k classification

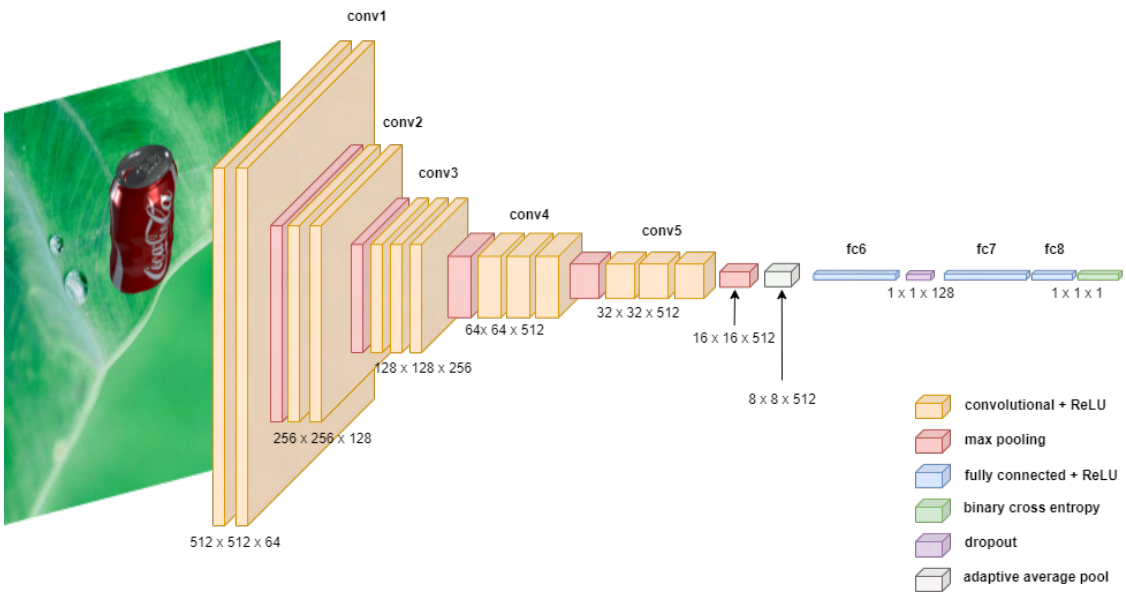


Figure 4.7: Adapted VGG-16 architecture for deformation classification

using the Adam optimizer with a learning rate of 0.001 and a batch size of 32, for 15 epochs.

For validating the synthetic datasets, the synthetic data was split into training and testing groups with 85% used to train and 15% used for testing. For the sim-to-real experiment, the entire synthetic dataset was used to train and the entire physical dataset was used the test.

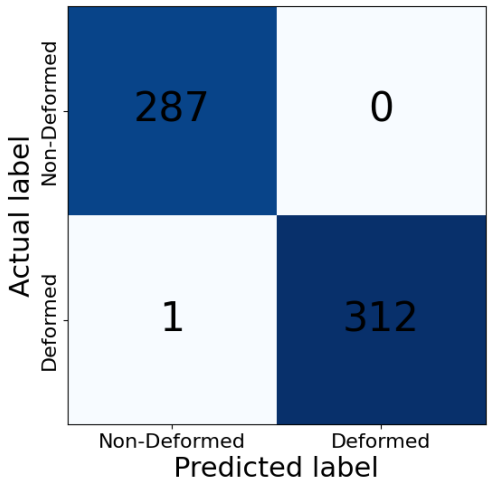
4.2.2 Sim-to-Real Results and Discussion

Table 4.2 summarizes the best performance measurements obtained training VGG-16 on the synthetic and real-world datasets. The confusion matrices shown in Figure 4.8 provide a breakdown of the results for the deformed and non-deformed classes. The network achieved very high accuracy in both background settings evaluating only synthetic data as shown in Figures 4.8a and 4.8c. However, performance dropped significantly when evaluated on the real-world dataset shown in Figures 4.8b and 4.8d. This is expected due to the inherent differences between the controlled synthetic environment and the complexity of the real world, including variations in deformations seen, camera angles, camera intrinsics and lighting conditions.

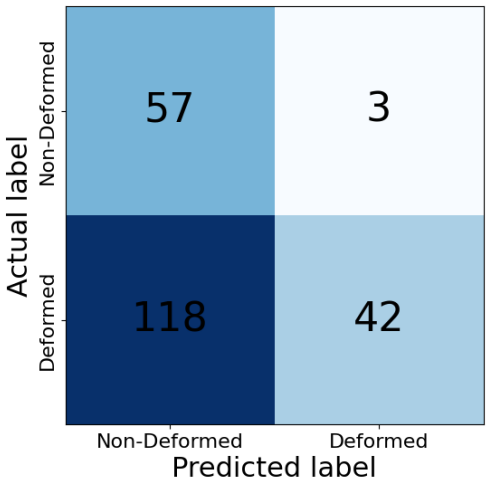
Introducing background variations using the BG-20k dataset significantly improved the network's ability to generalize to real-world data compared to the black background, as shown in both the confusion matrices, Figures 4.8b and 4.8d as well as Table 4.2. The principal component analysis (PCA) visualization in Figure 4.9 supports this observation. The physical dataset distribution is captured within the BG-20k distribution, whereas the tightly clustered black background has no overlap with the physical dataset distribution. This indicates the BG-20k backgrounds provide a more diverse and generalizable representation of the real world and the network trained from it has more real-world utility.

Table 4.2: Performance measure of VGG-16

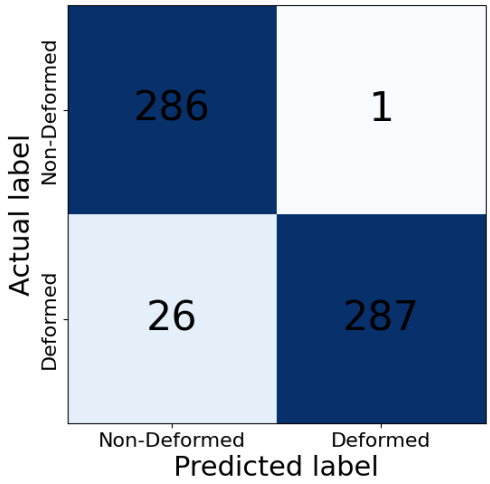
Metrics	Synthetic dataset		Sim-to-real	
	Black	BG-20k	Black	BG-20k
Accuracy	0.998	0.955	0.450	0.755
F_1	0.998	0.955	0.485	0.550
Recall	1.0	0.997	0.950	0.550
Precision	0.997	0.917	0.326	0.550



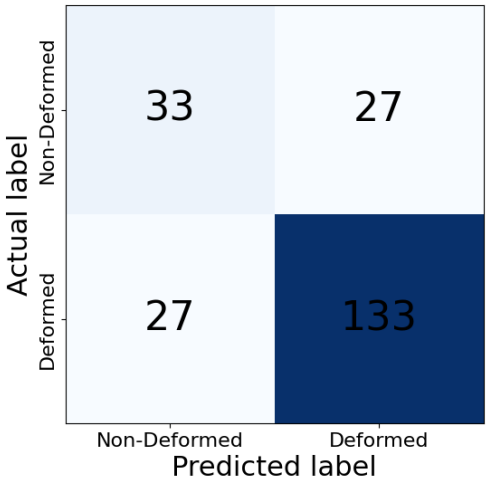
(a) Black background - synthetic dataset



(b) Black background -sim-to-real



(c) BG-20k background - synthetic dataset



(d) BG-20k background - sim-to-real

Figure 4.8: Confusion matrices for the network performance

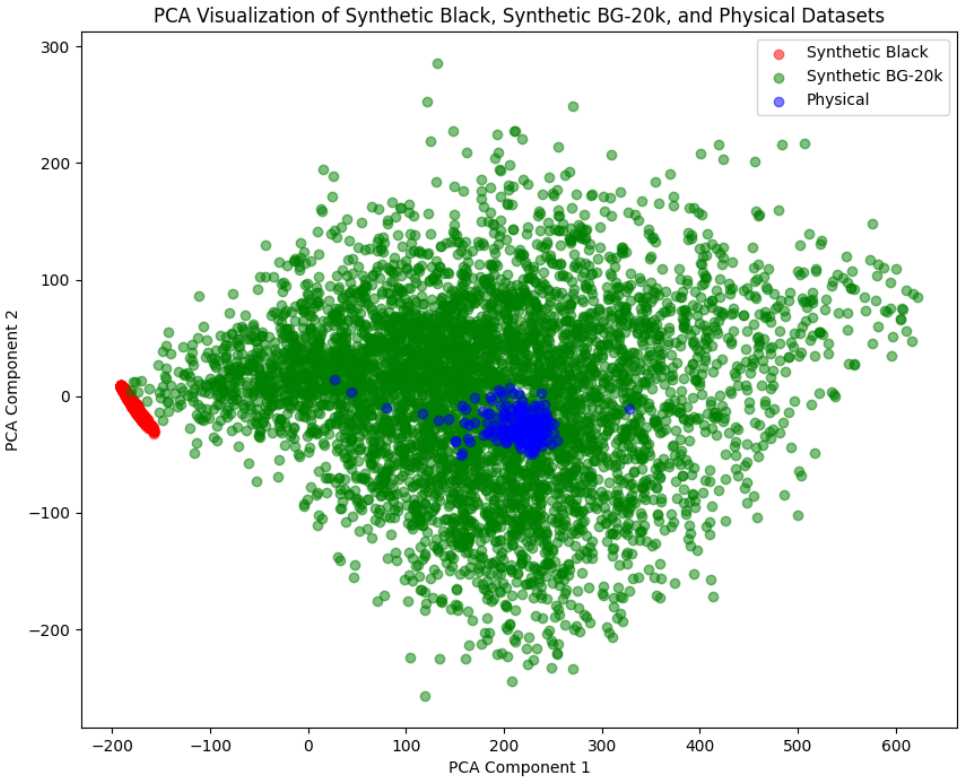


Figure 4.9: PCA visualization of all datasets

These results demonstrate the potential of using synthetic data with background variations for training deep learning models for deformation detection. This approach offers several advantages, particularly in scenarios where traditional methods are limited. Additionally, this method has no reliance on real-world data for training the network which can significantly reduce the time and resources needed to create large extensive training datasets, making it a more cost-effective solution.

Future improvements in simulation quality, such as incorporating more accurate shape keys or depth information, are expected to further enhance the generalizability of this method to real-world scenarios. This could allow for the detection of even finer deformations and a broader range of object types.

Despite the inherent stochasticity of real-world camera variations and the challenges posed by complex metallic surfaces, the background variation approach improves sim-to-real deformation detection.

Chapter 5

Domain Generative Adaptive

In the preceding chapter, a data augmentation approach to crossing the sim-to-real boundary was used. The networks were trained using only synthetic data and evaluated on the real data; the goal being to train a network solely on synthetic data and having acceptable real-world performance.

This chapter introduces a domain generative adaptation approach. A challenge inherent in adapting a neural network from one dataset to another is that when training, the network learns based on the distribution of the dataset. When this trained network is then tested on a different dataset, there is a drop in accuracy. For example, the network trained in the preceding chapter was trained on the synthetic dataset and had a drop in real-world performance. The goal is to design a network to handle the distribution shift from the synthetic Blender environment to the physical real world using pop cans as an example. In this chapter, the training scenario has changed, where we are assuming limited real-world training data is available. The goal is then to utilize the small real-world dataset to adapt the created synthetic dataset. Generative Adversarial Networks (GANs) are uniquely well suited to this task and several different architectures are explored in this chapter.

5.1 Implementation of a Generative Adversarial Network

GANs provide a way to learn deep representations without extensively annotated training data [12]. This is very useful for a variety of tasks including image synthesis, style transfer, image superresolution, and classification. The goal of a generative

model is to study a collection of training examples and learn the probability distribution that generated them [24]. In this case, a GAN is being used to take the synthetic pop can image and convert it into the style and domain of the real-world pop can dataset. The result of this process is expected to generate new data from the synthetic pop can dataset that more closely aligns with the real-world dataset. In this example, the datasets have been rescaled to be smaller with an image size of 256x256x3.

5.1.1 Architecture

A GAN operates through a competition between two neural networks, a generator G and discriminator D. The generator is given an image from the synthetic can dataset and is tasked with creating a new image mimicking the real-world dataset. The discriminator is given images from the real-world dataset and images from the generator and must decide if they are real or generated. The architecture of a GAN is shown in Figure 5.1.

The architecture of the generator used has 3 parts, an encoder, residual blocks, and a decoder. The encoder section consists of expanding convolutional layers increasing the channels from 3 to 64, to 128, and finally 256. A kernel size of 4 with a stride of 2 and a padding of 1 was used. The residual block portion consisted of 9 residual block layers with 256 channels. The decoder section is the exact inverse of the encoder architecture with decreasing channel sizes from 256 to 128, to 64 and finally 3. The final output from the generator is an image.

The discriminator consists of 6 convolutional layers, a linear layer then a sigmoid activated, 1 hot output. The convolutional layers first increase in channel size from 3 to 64, to 128, to 256, to 512 then finally decrease to 16. The single linear layer goes from 3136 to just 1. A sigmoid activation function is applied to the output to generate a probability the input image is generated or not.

Finally the classifier from Section 4.2.1 was used to classify the images.

5.1.2 Loss Functions

The overall objective of a GAN is captured in the minimax loss function shown as $V(D, G)$. This function aims to find an equilibrium between the generator and the discriminator. The generator continuously improves its ability to generate realistic data, while the discriminator becomes better at distinguishing between data created

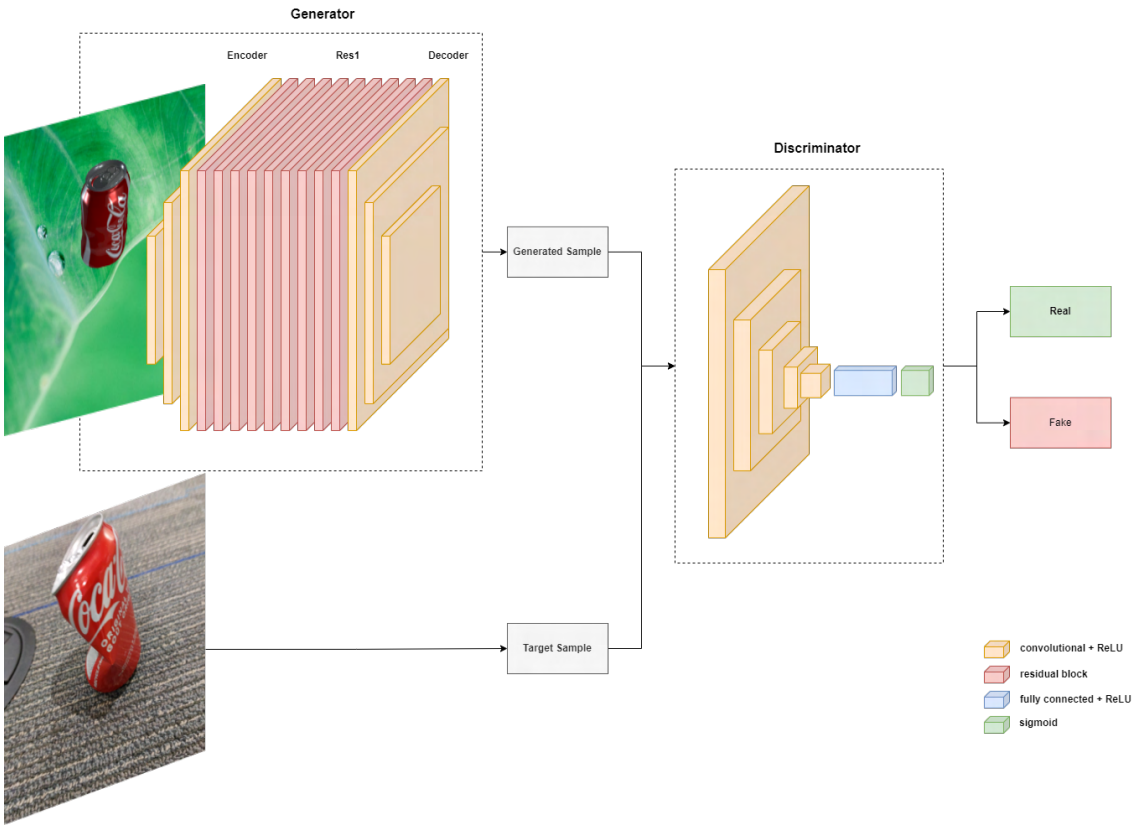


Figure 5.1: GAN architecture

by the generator and real data. The $\mathcal{L}_{GAN}$ term measures how well the discriminator performs. It encourages the discriminator to assign high probabilities to real images and low probabilities to fake images created by the generator. An identity loss function $\mathcal{L}_{Id}$ was added as well to encourage the GAN network not to change the image too drastically and the structure of the can not be changed too much. The $\mathcal{L}_{Id}$ loss is a pixel-wise L1 loss between the original image $x(p)$ and the generated image $y(p)$. In addition to these losses, a task loss $\mathcal{L}_{Task}$ was added. This was used to encourage the generator to make the data easier to classify and the data it generates belongs to the correct class.

- **Full Objective**:

$$\min_{G} \max_{D} V(D,G) = \mathcal{L}_{GAN} + \mathcal{L}_{Id} + \mathcal{L}_{Task} \tag{5.1}$$

- **Adversarial Loss**:

$$\mathcal{L}_{GAN} = \mathbb{E}_{\mathbf{x}\sim p_{data(\mathbf{x})}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{y}\sim p_{y(\mathbf{y})}}[\log(1 - D(G(\mathbf{y})))] \tag{5.2}$$

- **Identity Loss**:

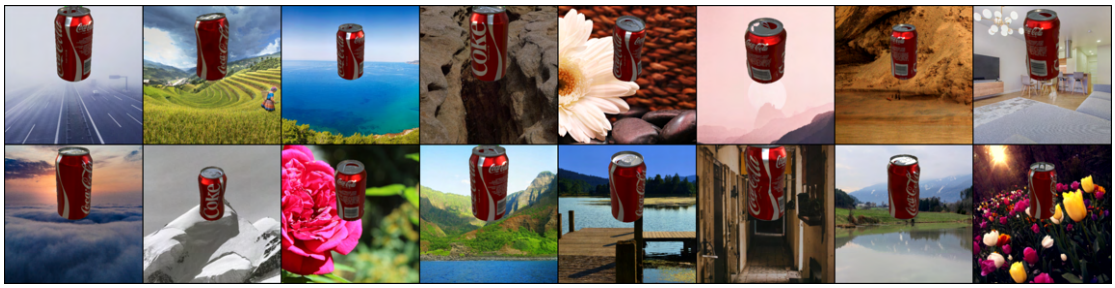
$$\mathcal{L}_{Id} = \frac{1}{N}\sum |x(p) - y(p)| \tag{5.3}$$

5.1.3 Training and Results

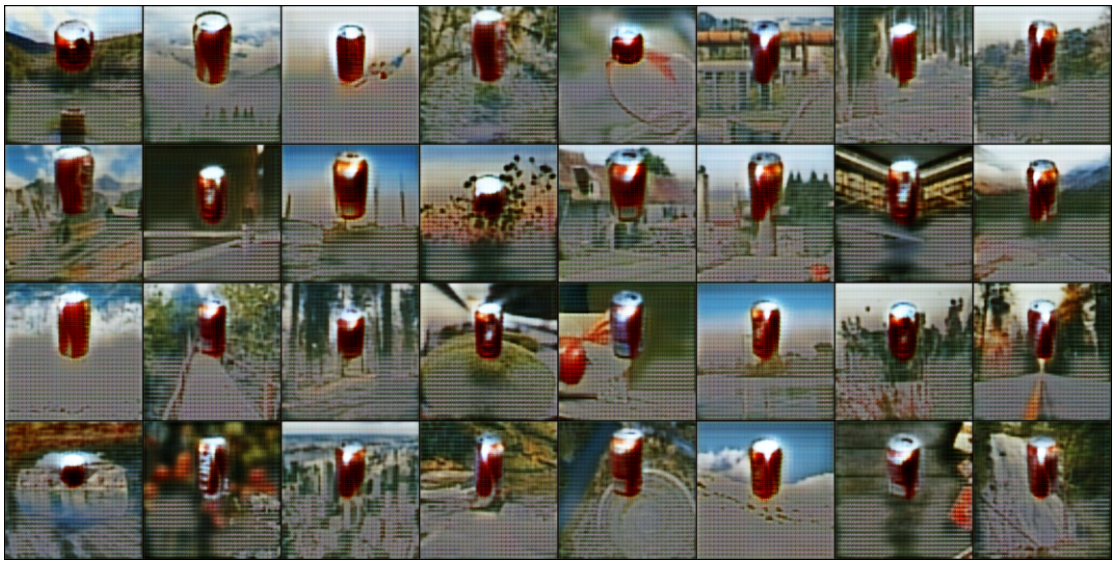
The learning rate for the generator and discriminator was 0.0002 and the learning rate for the classifier was 0.005. The network was trained for 200 steps before the generated images were manually inspected. In each step, the network takes an input batch and the parameters are updated. Throughout the training processes the generator and discriminator losses appeared balanced and the networks were training well. The pop can dataset with the BG-20k background was used during training as the source dataset and the real-world pop can dataset as the target dataset.

The results are shown in Figure 5.2. The input images are shown in Figure 5.2a and the images converted by the generator are shown in Figure 5.2b. The image quality is not acceptable and showed that the architecture shown in this section is not suitable for the task and further investigation was needed. The content remained the same but the background and style failed to change much. There is a variety in the images output which does signal that the generator was learning but perhaps

the discriminator was overpowered by the generator. Further results for classification accuracy were not investigated in favour of another GAN model architecture.



(a) GAN Input Images



(b) Generated GAN Images

Figure 5.2: Input images and generated images from the GAN

5.2 Implementation of CycleGAN

CycleGAN takes a different approach to domain adaptation compared to a traditional GAN. It utilizes two generator networks and two discriminator networks. The first generator G focuses on translating an image from the source domain X to the target domain Y. In this case, it's converting from the synthetic domain to the real-world domain. The second generator F performs the opposite translation and tries to convert from the target domain Y to the source domain X. The two discriminators are used similarly to the GAN discussed in the previous section but each discriminator is

used for a specific domain. The discriminator performs the same job of distinguishing between existing datasets and data created by the generators. Part of what makes CycleGAN useful for this task is that it is an unpaired image-to-image translation network [77]. This allows for the original content of an image to remain the same, while the style, texture and background can be adapted. The black background and real-world datasets were used as the source and target datasets respectively.

5.2.1 Architecture

The architecture used for CycleGAN is shown in Figure 5.3. CycleGAN functions with two generators in competition with their respective generators. The generators are tasked with translating an image into another respective domain while ensuring enough data from the original image is present for the other generator to reconstruct the image. The generators and discriminators used in the implementation of CycleGAN had the same feature layers used in the GAN architecture in section 5.1.1.

The architecture of the generator used has 3 parts, an encoder, residual blocks, and a decoder. The encoder section consists of expanding convolutional layers increasing the channels from 3 to 64, to 128, and finally 256. A kernel size of 4 with a stride of 2 and a padding of 1 was used. The residual block portion consisted of 9 residual block layers with 256 channels. The decoder section is the exact inverse of the encoder architecture with decreasing channel sizes from 256 to 128, to 64 and finally 3. The final output from the generator is an image.

The discriminator consists of 6 convolutional layers, a linear layer then a sigmoid activated, 1 hot output. The convolutional layers first increase in channel size from 3 to 64, to 128, to 256, to 512 then finally decrease to 16. The single linear layer goes from 3136 to just 1. A sigmoid activation function is applied to the output to generate a probability the input image is generated or not.

5.2.2 Loss Functions

- **Adversarial Loss**: Adversarial loss is applied to both mapping functions. The equation below is used for the mapping function $G : X \rightarrow Y$ and the discriminator D_Y. The generator G tries to produce images $G(x)$ that are similar to distribution Y while the discriminator D_Y tries to differentiate between generated images $G(x)$ and real samples Y. The generator is trying to minimize this

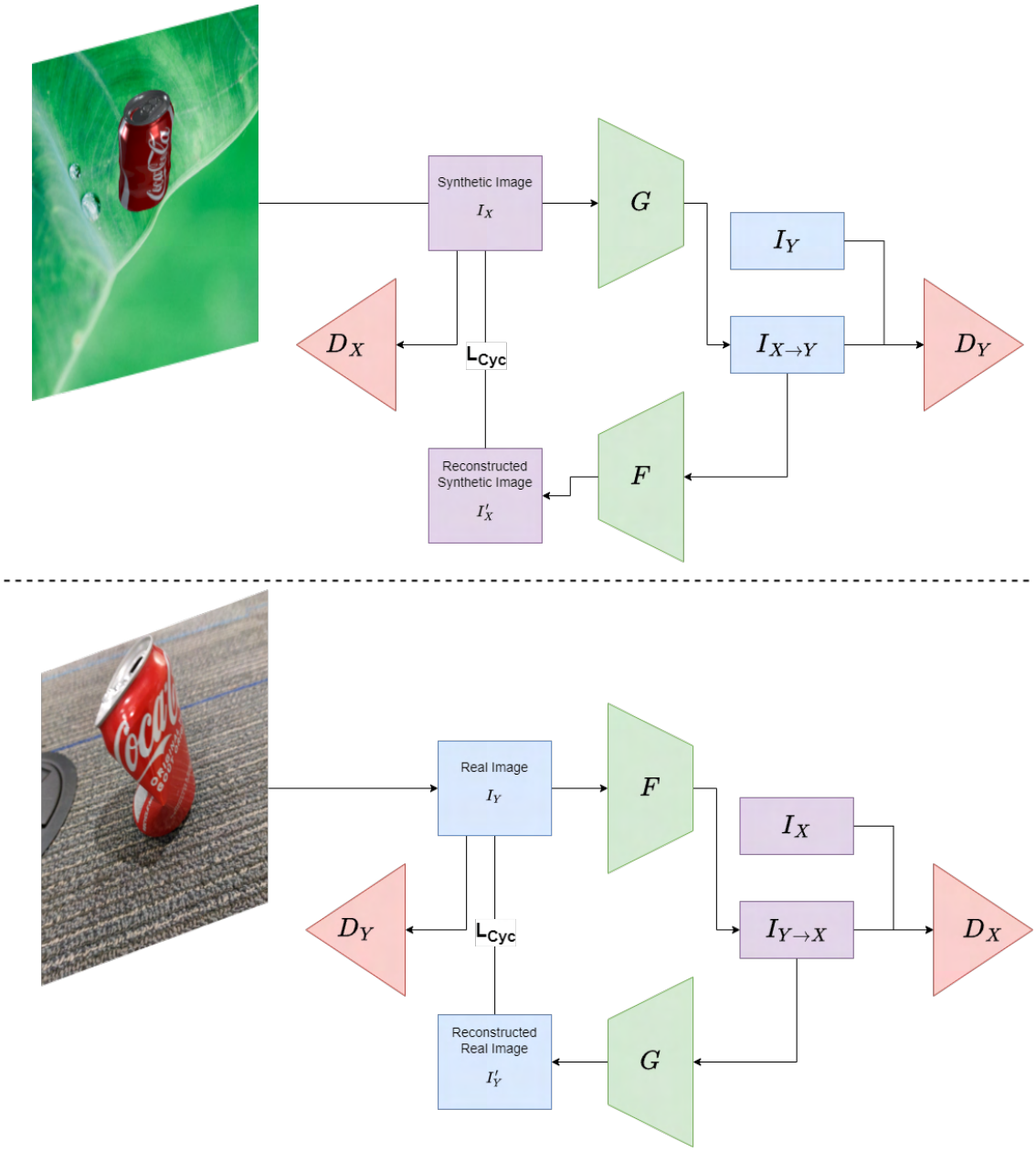


Figure 5.3: CycleGAN architecture

function, while the discriminator is trying to maximize the function.

$$\mathcal{L}_{GAN}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{data}(y)}[\log D_Y(y)] + \mathbb{E}_{(x) \sim p_{data}(x)}[\log(1 - D_Y(G(x)))] \tag{5.4}$$

- **Cycle Loss**: Adversarial loss can in theory produce results that are identical to the target distribution. Cycle loss encourages the generators to be able to change the translated image back to its original state. This is important for making the generated image unique from the target images while still falling into the distribution.

$$\mathcal{L}_{cyc}(G, F) = \mathbb{E}_{x \sim p_{data}(x)}[||F(G(x)) - x||_1] + \mathbb{E}_{y \sim p_{data}(y)}[||G(F(y)) - y||_1] \tag{5.5}$$

- **Full Objective**: The full objective of the loss function is shown below. The adversarial loss of both generators and discriminators are used in addition to the cycle loss. The term λ is used to balance the importance of the cycle loss. Typical values for λ are around 10 which was also used for this experiment, making the cycle loss have a lot of impact on the total loss.

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{GAN}(G, D_Y, X, Y) + \mathcal{L}_{GAN}(F, D_X, Y, X) + \lambda \mathcal{L}_{cyc}(G, F) \tag{5.6}$$

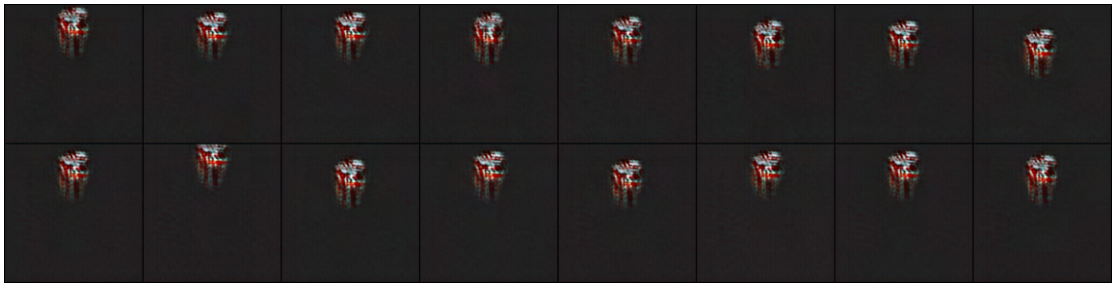
5.2.3 Training and Results

The model was trained for 200 steps with a learning rate of 0.001 and a batch size of 32 using the Adam optimizer. In each step, the network takes an input batch and the parameters are updated. To reduce oscillation spectral normalization was used in the discriminator. For each training step, the generator was trained for 2 cycles and the discriminator was trained once.

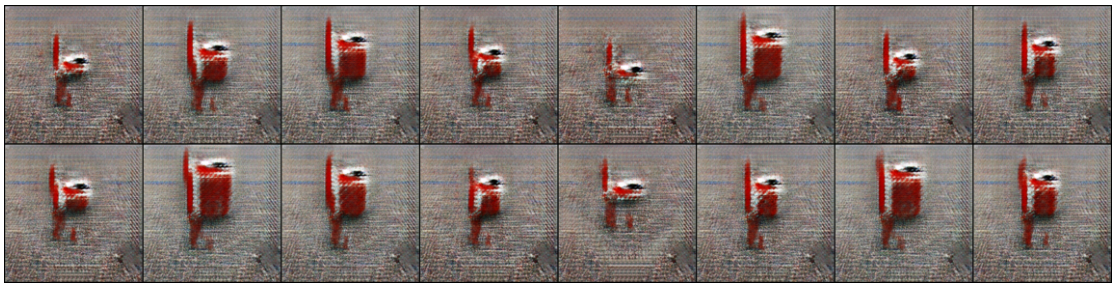
The results from the training using the black background dataset are shown in Figure 5.4 and showcase the adaptation from the synthetic domain to the real-world domain. The conversion shown in Figure 5.4c has poor results. The background seems to have been converted quite well but there is a significant issue with how the can is converted in the image. The converted image heavily distorts it and has limited variation. The reconstructed image shown in Figure 5.4b shows poor reconstruction as well. The results from CycleGAN proved unsatisfactory for the image translation

between the synthetic and real-world datasets and classification was not performed.

(a) Black background CycleGAN input



(b) Black background CycleGAN reconstruction



(c) Black background CycleGAN converted

Figure 5.4: CycleGAN image analysis

Additionally, results using the BG-20k random background dataset were collected which is shown in Figure 5.5. The results using this dataset resemble that of the black background. However, the reconstruction results in Figure 5.5b are improved. The converted results shown in Figure 5.5c are still unsatisfactory and training a deformation classifier was not performed.

(a) Black background CycleGAN input

(b) Black background CycleGAN reconstruction

(c) Black background CycleGAN converted

Figure 5.5: CycleGAN image analysis

5.3 Implementation of Disentangling Representation and Adaptation Network

This section focuses on the implementation of a disentangling representation and adaptation network (DRANet). DRANet is a network architecture that can disentangle the content and style from a latent space and transfer the visual attributes for unsupervised cross-domain adaptation. The architecture, loss functions, attempted improvements and results are discussed in this section to provide a comprehensive analysis of using this network as a component in the sim-to-real pipeline.

5.3.1 Architecture

The architecture of DRANet is shown in Figure 5.6. DRANet uses an encoder E, a feature separator S, generator G, two discriminators for source and target D_X, D_Y, a perceptual network P and a classifier C.

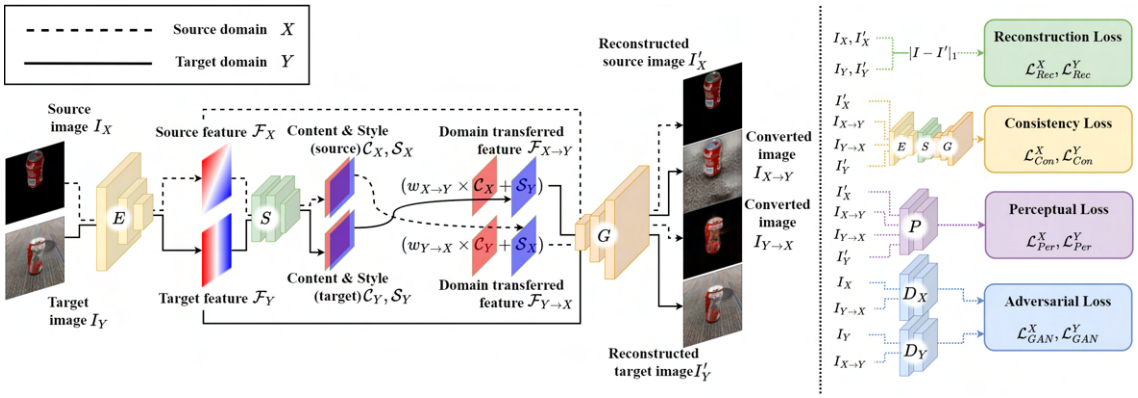


Figure 5.6: DRANet architecture

The workflow through the network is to take an image from both domains I_X, I_Y, and use an encoder to create a feature space that contains the essential information from the images. The feature space then passes through the separator where the content and style are extracted and the styles are swapped. The generator then takes the content and style from the separator and decodes it to form an image. The discriminator analyzes the output of the converted images from the generator $I_{X\to Y}$, $I_{Y\to X}$ and compares them to the source images I_X, I_Y. There is a discriminator for each domain. The synthetic discriminator looks at the images converted from the physical domain and original synthetic images. The physical discriminator looks at

the images converted from the synthetic domain and the original synthetic images. These networks work against the generator to encourage more convincing generated images.

The encoder E consists of a convolutional layer increasing the channels to 32, followed by 3 residual block layers. Finally, a last convolutional layer is used and expands the channels from 32 to 64. The task of the encoder is to take an input image and capture its features.

The separator S consists of three convolutional layers with 64 channels. The purpose of the separator is to split the features from the encoder into content and style feature layers.

The generator G consists of a transpose convolutional layer from 64 to 32, 3 residual block layers and a last transpose convolution layer from 32 to 3. A tanh activation function is used on the output. The purpose of the generator is to take content and style feature layers and generate an image.

All layers in the encoder, separator and generator use ReLU as the activation function and spectral normalization [43].

The discriminator takes in images ($I_X, I_{Y\to X}$ for D_X, $I_Y, IX \to Y$ for D_Y) and analyzes if the given image is generated or real. Initially the discriminators D_X, D_Y were standard discriminators with five convolution layers with channels from 3 to 16, to 32, to 64, to 128, to 256 and a single linear layer to a singular output. In each layer, spectral normalization was used and ReLU was used as the activation function. However, for the final results, this discriminator was changed to a patchGAN discriminator. The architecture for the patchGAN discriminator was convolution layers from 3 to 64, to 128, to 256, to 512 and finally down to 1 channel. The architecture uses LeakyReLU as an activation function with a negative slope of 0.2. The advantage of using a patchGAN discriminator was to provide better feedback to the generator on which sections of the generated images best mimic target samples as opposed to binary output.

The perceptual network P is a pre-trained version of VGG-19 on ImageNet1k. The role of the perceptual network is to analyze the images for content and style and help train the separator network.

The perceptual network inputs the reconstructed I'_X, I'_Y and converted images $I_{X\to Y}$, $I_{Y\to X}$ and creates feature maps from several middle layers. The network finds perceptual features to create constraints on the content and style of the images to train the separator network in an unsupervised manner.

Finally, the classifier is the same VGG-16 pre-trained network used previously, that analyzes the images and tries to classify if the image contains a deformed or non-deformed pop can.

5.3.2 Loss Functions

- **Reconstruction loss**: A reconstruction loss was used to minimize the difference between the input image I and the output image from the generator I'. The image I' is created by taking the input image I and passing it through the encoder E which then skips the separator and is passed directly to the generator G. Finally, the L1 loss is taken between I and I'.

$$\mathcal{L}^{d}_{Rec} = \mathcal{L}_1(I_d, I'_d), \text{ where } I'_d = G(E(I_d)) \tag{5.7}$$

- **Adversarial loss**: Two discriminators $D_{d\in\{X,Y\}}$ were used for evaluating the adversarial loss for the source and target domains. The images used for this loss are the input images I and the image converted by passing through the encoder E, separator S and generator G, to form images $I_{X\to Y}$ and $I_{Y\to X}$. For the transfer of both domains, the adversarial loss is applied and the domain adaptation of $X \to Y$ is shown below.

$$\mathcal{L}^{Y}_{GAN} = \mathbb{E}_{y\sim p_{data(Y)}}[\log D_Y(y)] + \mathbb{E}_{(x,y)\sim p_{data(X,Y)}}[\log(1 - D_Y(I_{X\to Y}(x,y)] \tag{5.8}$$

- **Consistency loss**: Consistency loss is added to encourage the network to keep the content and style information, scene structure and appearance consistent in the representation space from the separator S. The consistency loss is calculated by passing an images I_X, I_Y through the encoder E and separator S to get content $\mathcal{C}_X$, $\mathcal{C}_Y$ and style $\mathcal{S}_X$, $\mathcal{S}_Y$. The images in then fully converted to the other domain to form images $I_{X\to Y}$ and $I_{Y\to X}$. $I_{X\to Y}$ and $I_{Y\to X}$ are then passed through the same encoder and separator to get the converted contents $\mathcal{C}_{X\to Y}$, $\mathcal{C}_{Y\to X}$ as well as styles $\mathcal{S}_{Y\to X}$, $\mathcal{S}_{X\to Y}$. The L1 loss is taken between both contents $\mathcal{C}_X$ and $\mathcal{S}_{X\to Y}$ and styles $\mathcal{S}_X$ and $\mathcal{S}_{Y\to X}$ for domain X. The L1 loss of the other contents $\mathcal{C}_Y$ and $\mathcal{C}_{Y\to X}$ as well as $\mathcal{S}_Y$ and $\mathcal{S}_{X\to Y}$ are used for domain Y.

$$\mathcal{L}^{X}_{Con} = \mathcal{L}_1(\mathcal{C}_X, \mathcal{C}_{X\to Y}) + \mathcal{L}_1(\mathcal{S}_X, \mathcal{S}_{Y\to X}), \quad \mathcal{L}^{Y}_{Con} = \mathcal{L}_1(\mathcal{C}_Y, \mathcal{C}_{Y\to X}) + \mathcal{L}_1(\mathcal{S}_Y, \mathcal{S}_{X\to Y}) \tag{5.9}$$

- **Perceptual loss**: The perceptual loss[33] consist of content and style losses. The purpose of the style loss is to teach the separator network how to disentangle the content and style from the encoder in an unsupervised manner. This can defined by the following function where $\mathcal{L}^X_{Content}$, $\mathcal{L}^Y_{Content}$ are the content losses and $\mathcal{L}^X_{Style}$, $\mathcal{L}^Y_{Style}$ are the style losses.

$$\mathcal{L}^X_{Per} = \mathcal{L}^X_{Content} + \lambda \mathcal{L}^X_{Style}, \quad \mathcal{L}^Y_{Per} = \mathcal{L}^Y_{Content} + \lambda \mathcal{L}^Y_{Style} \tag{5.10}$$

The perceptual network is a pre-trained version of VGG-19. of which, L_C and L_S are various feature layers. The image I_X and the converted image $I_{X \to Y}$ are input into the perceptual network and the output consists of various feature layers L_C and L_S. The output of these feature layers is then used for further calculation of the content and style. The function $\mathcal{G}$ forms the Gram matrix of the feature layers and is used to calculate the style. The content and style use a weight parameter λ to balance the losses. This was the style loss used for generating the pop can dataset.

$$\mathcal{L}^Y_{Content} = \sum_{l \in L_C} ||P_l(I_X) - P_l(I_{X \to Y})||_2^2 \,, \quad \mathcal{L}^Y_{Style} = \sum_{l \in L_S} ||\mathcal{G}(P_l(I_Y)) - \mathcal{G}(P_l(I_{X \to Y}))||_F^2 \tag{5.11}$$

Additional experimentation in style loss was performed by replacing the Gram matrix with Sliced Wasserstein Discrepancy (SWD). SWD is proposed to have numerous advantages over Gram loss as outlined in [27] for style transfer and texture synthesis. An implementation as outlined by [35] has been used for calculating the SWD. In the following equation, the gram matrix was replaced with the SWD.

$$\mathcal{L}^Y_{Style} = \sum_{l \in L_S} ||\mathcal{SWD}(P_l(I_Y)) - \mathcal{SWD}(P_l(I_{X \to Y}))||_F^2 \tag{5.12}$$

Additionally taking the SWD of the Gram matrix was investigated as shown in the formula below.

$$\mathcal{L}^Y_{Style} = \sum_{l \in L_S} ||\mathcal{SWD}(\mathcal{G}(P_l(I_Y))) - \mathcal{SWD}(\mathcal{G}(P_l(I_{X \to Y})))||_F^2 \tag{5.13}$$

- **Full Objective**: The encoder, separator and generator are trained to minimize a loss function while a discriminator tries to maximize it. The domain d is either the source or target domain X or Y.

$$\min_{E,S,G}\left(\sum_{d\in\{X,Y\}}\max_{Dd}\mathcal{L}^d\right) \tag{5.14}$$

  $\mathcal{L}^d$ is the combination of the previous losses discussed previous; $\mathcal{L}^d_{Rec}$ is the reconstruction loss, $\mathcal{L}^d_{GAN}$ is the adversarial loss, $\mathcal{L}^d_{Con}$ is the consistency loss, and $\mathcal{L}^d_{Per}$ is the perceptual loss. The alpha terms α are used to balance the total loss function.

$$\mathcal{L}^d = \alpha_1\mathcal{L}^d_{Rec} + \alpha_2\mathcal{L}^d_{GAN} + \alpha_3\mathcal{L}^d_{Con} + \alpha_4\mathcal{L}^d_{Per} \tag{5.15}$$

5.3.3 Wasserstein GAN with Gradient Penalty

DRANet-WGP builds upon the foundation of DRANet by employing a Wasserstein GAN with Gradient Penalty (WGAN-GP) architecture. This shift from the original deep convolutional GAN (DCGAN) structure was intended to have several advantages, particularly in training stability. DCGANs rely on weight clipping to enforce a specific property called Lipschitz continuity. Lipschitz continuity ensures the discriminator's input leads to proportional changes in its output. This would lead to similar images being classified with similar scores in the critic. However, using weight clipping in the loss used by DCGANs can introduce an uneven loss landscape, underutilized network capacity and gradient explosion or vanishing. WGAN-GP addresses these problems by implementing a gradient penalty. This penalty term ensures the critic adheres to the Lipschitz constraint inherently, without the need for explicit clipping. Enforcing the Lipschitz constraint in WGAN-GP essentially limits how drastically the critic's output can change in response to changes in its input. This constraint ensures the critic's behavior is well-behaved and facilitates a smoother training process. Additionally, the critic provides a more useful output to the generator, leading to improved generated images. The downside to WGAN-GP is that while it has improved training stability, they generally require more training iterations to converge in comparison to DCGANs. While WGAN-GP might require more training time, the enhanced stability and theoretical advantages showed promise of being a superior method specifically in bridging the sim-to-real gap.

The equation characterizing the WGAN-GP loss[25] is shown below.

$$L = \underbrace{\mathop{\mathbb{E}}_{\tilde{\mathbf{x}}\sim\mathbb{P}_g}[D(\tilde{\mathbf{x}})] - \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathbb{P}_r}[D(\mathbf{x})]}_{\text{WGAN critic loss}} + \underbrace{\lambda \mathop{\mathbb{E}}_{\hat{\mathbf{x}}\sim\mathbb{P}_{\hat{\mathbf{x}}}}[(||\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})||_2 - 1)^2]}_{\text{Gradient Penalty}} \tag{5.16}$$

$\mathbb{P}_{\hat{\mathbf{x}}}$ is defined as a uniform sampling between pairs of points in the data distribution $\mathbb{P}_r$ and generator distribution $\mathbb{P}_g$. The value used for λ was 10, to try to encourage smoother learning from the discriminator. The discriminator architecture had to be changed to remove normalization from the generator and discriminator. The normalization layers hinder the enforcement of the Lipschitz constraint which in turn allows the gradient penalty to function more effectively.

Upon implementation of this concept, the generator struggled to compete with the discriminator. Despite changes in various hyper-parameters, training the generator more often than the discriminator and expanding network layers, stability in training was unable to be achieved. The DCGAN approach for DRANet has much improved stability and faster convergence which is important as the training time for this network is long. Spectral normalization seems to have a more positive effect on stability than could be achieved by using the WGAN-GP architecture.

5.3.4 Style Loss Results

Figure 5.7 showcases how the different style losses affect a generated image. For all images, the networks were trained for 6000 steps then the images were saved for analysis. Analysis for the style loss was done using the MNIST and MNIST-M datasets. These datasets were chosen due to the relatively small image size which allows for faster computation for fine-tuning for an optimal style transfer that is difficult to do with the much larger pop can datasets. Computation time using MNIST and MNIST-M datasets took about an hour to reach training step 6000 whereas using the pop can datasets computation time is about 24 hours to reach training step 600. All computation was performed using an AMD 3600 CPU with an Nvidia RTX 3060 GPU. Figure 5.7a shows the images input into DRANet to be generated for domain transfer. Figure 5.7b showcases Gram loss and is the original style loss used in the original DRANet [36]. The results from using Gram loss are excellent and provide a baseline for comparing the other style losses. Figure 5.7c showcases the SWD style loss from the same feature layers from the perceptual network VGG-19. The results from

using this style loss have better and clearer content results than the Gram style loss. The purpose of using SWD loss is to enforce the generated image to exhibit similar stylistic features as the target image. However, upon inspection of the content, using the SWD loss causes some of the content features, the numbers in the MNIST dataset, to not be transferred. This behaviour makes this style loss unsuitable. Figure 5.7d showcases the effects of using the SWD loss of the Gram matrix. This style loss is intended to capture more soft features. This loss has the same higher content clarity as the SWD method previously but without the content transfer issues. This loss is intended to strike a balance between Gram and SWD style loss. This loss struggles to transfer the styles in these images and has noticeably worse results than using just Gram style loss. Overall the style loss that led to the best image quality was Gram and was used for the domain adaptation of the pop can datasets.

5.3.5 Content-Adaptive Domain Transfer

Style transfer can struggle with complex scenes and features. This is particularly evident when the images have different scene structures and object compositions [36]. This is particularly evident in trying to adapt the synthetic scene to the real-world scene. The effect from style transfer struggling is shown by *ghosting* artifacts between datasets this is shown in 5.8a. Content-Adaptive Domain Transfer (CADT) aims to solve this by searching the target features for content components most similar to the source features and then reflecting style information from more suitable target features. This is achieved by using a content similarity matrix $\mathcal{H}_{row}$.

$$\mathcal{H}_{row} = \sigma_{row}(\mathcal{C}_X \cdot \mathcal{C}_Y^\top) = \begin{bmatrix} \mathcal{C}_{11} & \cdots & \mathcal{C}_{1b} \\ \vdots & \ddots & \vdots \\ \mathcal{C}_{b1} & \cdots & \mathcal{C}_{bb} \end{bmatrix}, \quad \mathcal{C}_X, \mathcal{C}_Y \in \mathbb{R}^{B \times N} \tag{5.17}$$

σ_{row} is a softmax in the row dimension, the size of content factor $\mathcal{C}_X$ is defined by batch size B and feature dimension N. This matrix is used to create the content-adaptive style feature described in the following equation:

$$\hat{\mathcal{S}_Y} = \mathcal{H}_{row}\mathcal{S}_Y, \quad \text{where} \quad \mathcal{S}_Y \in \mathbb{R}^{B \times N} \tag{5.18}$$

Content-adaptive style transfer can be applied in the opposite direction by using the column direction and the following equation:

(a) Input images from MNIST and MNIST-M

(b) Converted images using gram style loss $\mathcal{G}(I_d)$

(c) Converted images using SWD style loss $\mathcal{SWD}(I_d)$

(d) Converted images using gram and SWD style loss $\mathcal{SWD}(\mathcal{G}(I_d))$

Figure 5.7: Analysis of style loss functions

$$\mathcal{H}_{col} = (\sigma_{col}(\mathcal{C}_X \cdot \mathcal{C}_Y^\top))^\top \tag{5.19}$$

The results using CADT stylistically are much better, with *ghosting* artifacts removed. A comparison of results generated with and without CADT is shown in Figure 5.8. The images generated with CADT show much better style transfer and a better understanding of the content of the image as well. Images generated with CADT are shown in Figure 5.8b.

5.3.6 Training and Results

In the original implementation of DRANet, the classifier is trained at the same time as the GAN network. In this implementation, these networks were trained separately. The GAN network was trained first and all the images were generated and saved. The classifier was then trained on the saved converted images.

DRANet was initially trained for 600 steps on both the black background and BG-20k background datasets without CADT using a regular discriminator. This was done to evaluate which dataset produced better-quality images. The resulting converted images can be found in Figure 5.9. Overall, DRANet generates better images using the dataset with the black background. The black background image has captured most of the content and has fewer *ghosting* artifacts. The BG-20k conversion has issues with ghost images from the BG-20k dataset in the background as well as the can from the real-world dataset being mixed in. The images converted from the black background dataset show much better style and domain transfer.

This result can be explained by referencing the principal component analysis (PCA) visualization from Figure 4.9 in chapter 4 comparing the datasets. The black background has a closer distribution within its dataset compared to BG-20k. This makes the processes of understanding the content and style much simpler for DRANet leading to improved generated image quality. The much wider distribution that the BG-20k pop can dataset is more difficult for the network to understand for domain adaptation.

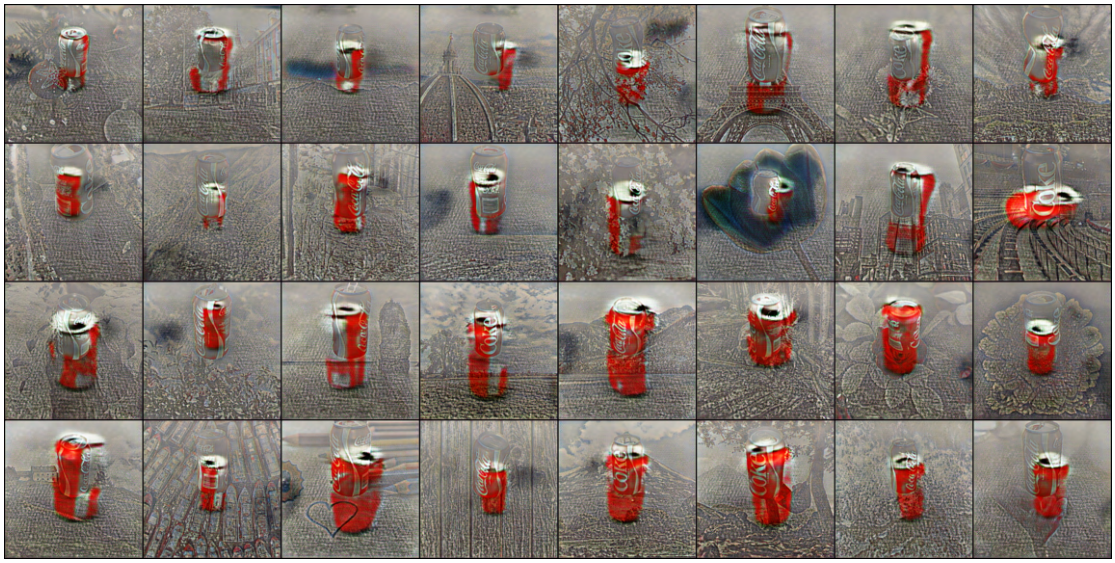
For the final iteration of training, DRANet was trained with CADT and a PatchGAN discriminator. It was trained for 2000 steps using the Adam optimizer with a learning rate of 0.001, a decay rate of 0.95 every 600 steps and a batch size of 2. In each step, the network takes an input batch and the parameters are updated. The results of the training are shown in Figure 5.10. Using the input images from

(a) Converted synthetic images without CADT

(b) Converted synthetic images with CADT

Figure 5.8: Impact of content-adaptive style transfer between synthetic and real-world datasets

(a) Converted images using pop can black dataset



(b) Converted images using pop can BG-20k dataset

Figure 5.9: DRANet converted images from black and BG-20k backgrounds from training step 600

the black background and real-world datasets in Figures 5.10a and 5.10b DRANet was effectively able to discover the content and styles of the images. The converted images in Figures 5.10e and 5.10f have excellent results. While the background of the synthetic cans isn't very high quality, the content transferred is excellent. The background is the synthetic converted images is a decent attempt at transferring the surface the cans are placed on from the real-world dataset. The black background from the synthetic dataset is applied to the background of the real-world dataset well and shows excellent style transfer. Overall the converted image quality from DRANet is excellent and shows a great understanding of the content and style of the images. The reconstructed images in Figures 5.10c and 5.10d are very similar to the original which shows that DRANet is performing well.

Only the converted images from synthetic to real are used for creating the generated dataset and further classification. However, in different applications or circumstances, the real-world images converted to synthetic could be used as well.

The PCA visualizations shown in Figure 5.11 show that DRANet was effective at capturing the real-world dataset and was successful in its domain transfer task. Figure 5.11a shows the PCA visualization of the domain transfer from the synthetic black background images into the physical real-world dataset. The generated images capture the physical distribution well and validate DRANet's effectiveness in domain transfer.

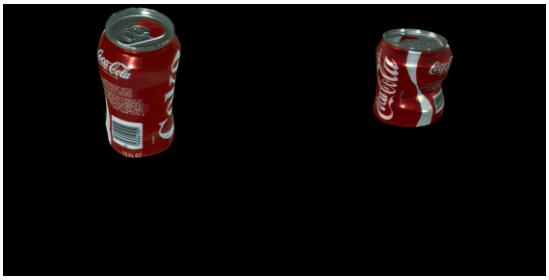
Figure 5.11b compares the generated distribution to the previously discussed dataset that uses backgrounds from BG-20k. In this visualization, the generated dataset very closely matches the real-world dataset and does a better job of capturing the real-world dataset.

The VGG-16 classifier was trained on the generated images for 15 epochs using the stochastic gradient descent (SGD) optimizer. The learning rate was set to 0.005 and a batch size of 32 was used. Table 5.1 summarizes the performance metrics for the VGG-16 classifier on the synthetic and real-world datasets and provides a comparison to the best results from chapter 4. The confusion matrices shown in Figure 5.12 provide a breakdown of the results for the deformed and non-deformed classes. The network achieved excellent metrics evaluating only synthetic data as shown in Figure 5.12a. However, the performance still dropped when the network was evaluated on real-world data as shown in Figure 5.12b. This is still expected due to the different camera angles and deformations present in the images.

The classification measures in Table 5.1 are from the best results obtained during

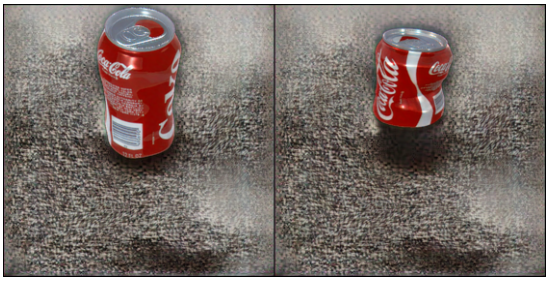
(a) Input images from black background dataset

(b) Input images from real-world dataset



(c) Reconstructed images from black background dataset

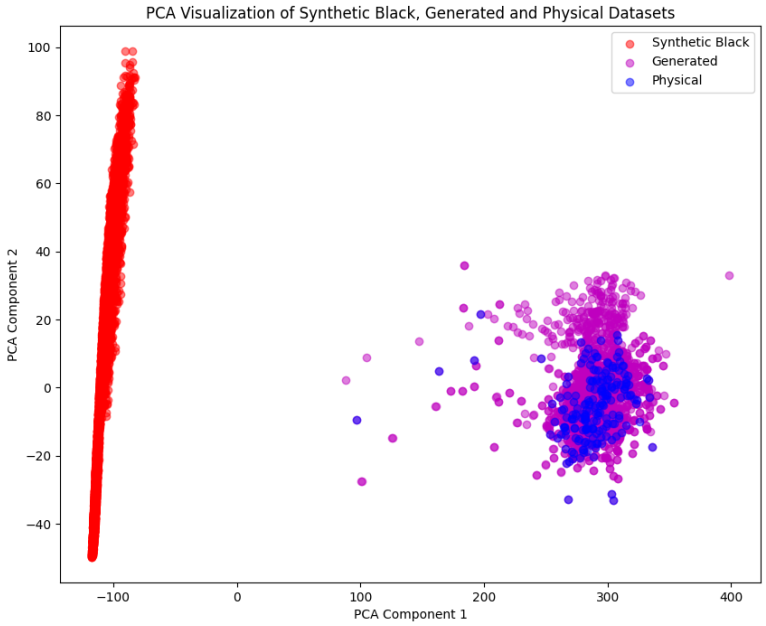
(d) Reconstructed images from real-world dataset



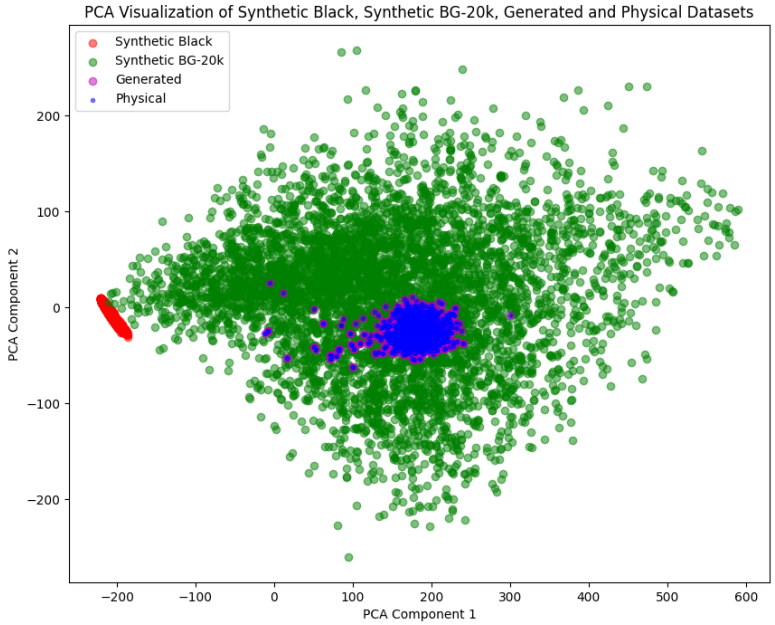
(e) Converted images from black background dataset

(f) Converted images from real-world dataset

Figure 5.10: Domain adaptation results from DRANet



(a) PCA visualization of generated data



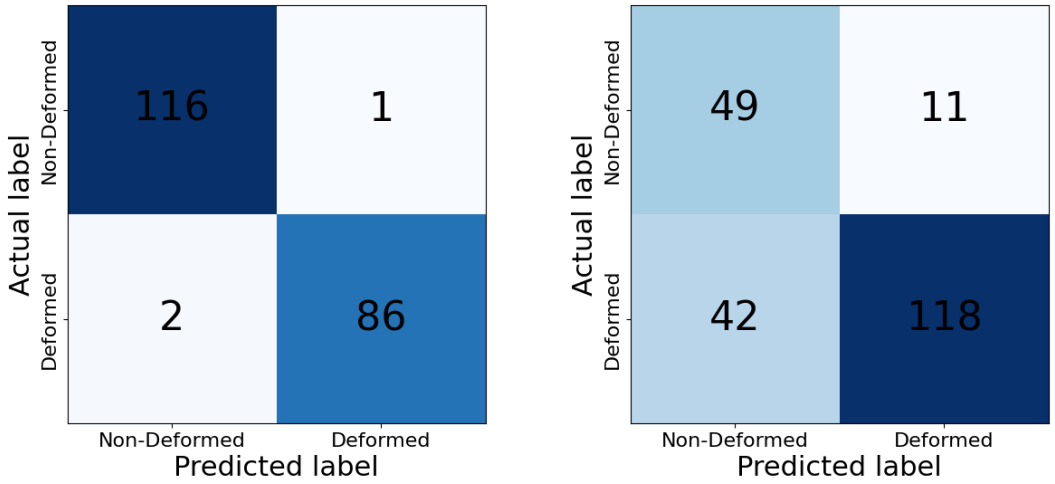
(b) PCA visualization with all datasets

Figure 5.11: DRANet generated images PCA visualizations

training. Similar values were obtained on different training runs. The results show a slight improvement in accuracy, a slight decrease in precision, and improvements to F1-score and recall in comparison to the best classification from chapter 4. The results from DRANet provide a smaller sim-to-real gap and improved performance to previously discussed methods. The use of DRANet was able to provide a distribution more similar to the real world which resulted in better performance. DRANet is shown to have the best performance of all methods discussed in this thesis.

Table 5.1: DRANet Comparative Performance to Best Data Augmentation

Metrics	Synthetic dataset		Sim-to-real	
	Generated	BG-20k	Generated	BG-20k
Accuracy	0.985	0.955	0.759	0.755
F_1	0.987	0.955	0.649	0.550
Recall	0.991	0.997	0.817	0.550
Precision	0.983	0.917	0.538	0.550



(a) Generated dataset - synthetic dataset (b) Generated dataset - sim-to-real

Figure 5.12: Confusion matrices for comparing DRANet performance

5.4 Discussion

Comparing the generated images from a GAN (section 5.1), CycleGAN (section 5.2) and DRANet (section 5.3) shows that the best performance from DRANet. The other

methods struggle with understanding how to adapt the synthetic source images into the target real-world images. The GAN network was able to produce images that still contained the pop can, the image quality was quite blurry and the background was a poor attempt at creating the target background that contained elements from the source background. The CycleGAN network produced images that copied the style of the background quite well but had a very negative effect on the content of the pop can. The cans in the images produced very loosely resemble their original look and the reconstruction doesn't match the original image. DRANet produced images where the content is clear and matches the content of the source images. The backgrounds in the generated images are a decent attempt at recreating the target background but are worse than the background produced by CycleGAN. Additional efforts to improve the background style transfer would improve the images generated by DRANet. A possible avenue for improving this would be using SWD for the style loss and adding additional parameters to ensure the content isn't affected.

The classification results using the images generated from DRANet are much improved in comparison to the results from the generalization methods. Generalization methods can be good for a variety of domains however, there is a cost for that generalization in terms of F1 score and other metrics. The results from all methods are shown in Table 5.2. The generated images from DRANet provide the best metrics in comparison to all methods discussed in this thesis. The results are particularly impressive when considering the sim-to-real results of its source dataset with the black background with much-improved accuracy, F1-score and precision.

Table 5.2: Classification Measure of all Methods

Metrics	Synthetic dataset			Sim-to-real		
	Generated	Black	BG-20k	Generated	Black	BG-20k
Accuracy	0.985	**0.998**	0.955	**0.759**	0.450	0.755
F_1	0.987	**0.998**	0.955	**0.649**	0.485	0.550
Recall	0.991	**1.0**	0.997	0.817	**0.950**	0.550
Precision	0.983	**0.997**	0.917	0.538	0.326	**0.550**

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The primary contribution of this thesis has been designing, implementing and validating a pipeline for creating and using synthetic data for sim-to-real deformation detection. Chapter 2 introduced the background and information relevant to this thesis including deformation detection in manufacturing, deformation detection methods, synthetic data generation, data augmentation and domain adaptation. These topics cover the core concepts behind the research achieved in this thesis.

Chapter 3 focused on the generation of synthetic data in Blender using BlendTorch [11]. With the use of expert knowledge, realistic macro deformations were created using a lattice deform modifier, and surface deformations were created using a hard *Stucci* displacement modifier. Using BlendTorch, an interpolation between these shape keys can be controlled to create high-quality procedural deformations. Additionally, shading tools were used to create realistic surface material properties and procedural lighting, improving realism. Post-processing such as noise injection and setting backgrounds were used to finalize the produced synthetic datasets.

Chapter 4 explored and evaluated the using the synthetic using traditional machine learning (ML) techniques. Using the Suzanne dataset and a basic convolutional neural network (CNN) classifier, a network was able to be trained and have high classification accuracy able to withstand noise injection. A full sim-to-real analysis was completed using both the solid black and BG-20k [37] background soda can datasets.

Chapter 5 implemented and trialled three generative adversarial networks (GANs) for domain adaptation. A traditional GAN, CycleGAN and disentangling representa-

tion and adaptation networks (DRANet) were implemented and the generated images were evaluated. DRANet showed better performance in unpaired image translation for converting the synthetic data for sim-to-real transfer in comparison to the other networks. The generated dataset from DRANet was classified and evaluated. An analysis comparing the results to those in chapter 4 shows improved results and is effective in its goal of crossing the sim-to-real gap.

The results from this research show the proposed pipeline is effective for using synthetic data to classify real-world deformations. The implications of this system can be to improve quality assurance processes in industrial and manufacturing settings, especially in circumstances where dataset collection is expensive, difficult to produce, or costly.

6.2 Future Work

The results discussed in this thesis show clear effectiveness in solving sim-to-real deformation detection, there are a few avenues for improvement:

- **Improved Blender Deformation Simulations**
  Improvements to the environment for closing the sim-to-real gap could be made. In this paper, only 12 different deformations were used. Additional and higher-quality deformations would improve classification accuracy and improve each network's understanding of the classification object. Additionally, the environment could be changed to allow more changes to the camera positioning. Improving the dataset by allowing the object to not always be in the center of the image could help improve how the robustness of the resulting network. Additional lighting environments and more accurate material shaders could be used to improve the data generation quality. If finer detail analysis is required, depth sensing using a normal map in Blender and an RGB-Depth camera for the real-world dataset could be used to improve small deformation accuracy.

- **Explore Alternative Networks for Deformation Classification**
  The network used in both the augmentation approach in chapter 4 and the domain adaptation approach in chapter 5 used a modified pre-trained version of VGG-16 [60]. While this network was effective in classifying the deformations, other networks and architectures may have improved performance and led to better classification accuracy.

- **Improve Style Transfer in DRANet**
  The generated dataset results from DRANet showcased impressive performance, however, style transfer provides the avenue for the most improvement. Analysis of Sliced Wasserstein Discrepancy (SWD) [35] for style loss showed superior results in style transfer compared to the Gram loss [19] used. However, SWD caused issues that affected the content of the images generated. Additional experimentation and improvement to the content-adaptive domain transfer (CADT) function to accommodate SWD style loss would significantly improve performance and image generation quality. Additional hyperparameter tuning and fine-tuning layers in the encoder, separator, generator and discriminator could improve generated image quality.

Bibliography

[1] Ahmed Alaa, Boris Van Breugel, Evgeny S. Saveliev, and Mihaela van der Schaar. How faithful is your synthetic data? Sample-level metrics for evaluating and auditing generative models. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 290–306. PMLR, 17–23 Jul 2022.

[2] Martín Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. *ArXiv*, abs/1701.04862, 2017.

[3] Luigi Barazzetti, Luigi Mussio, Fabio Remondino, and Marco Scaioni. Targetless camera calibration. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 38:335–342, 2012.

[4] Finn Behrendt, Debayan Bhattacharya, Robin Mieling, Lennart Maack, Julia Krüger, Roland Opfer, and Alexander Schlaefer. Guided reconstruction with conditioned diffusion models for unsupervised anomaly detection in brain mris. *arXiv preprint arXiv:2312.04215*, 2023.

[5] Hamdi Ben Abdallah, Igor Jovančević, Jean-José Orteu, and Ludovic Brèthes. Automatic inspection of aeronautical mechanical assemblies by matching the 3d cad model and real 2d images. *Journal of Imaging*, 5(10):81, 2019.

[6] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine learning*, 79:151–175, 2010.

[7] Aleksei Boikov, Vladimir Payor, Roman Savelev, and Alexandr Kolesnikov. Synthetic data generation for steel defect detection and classification using deep learning. *Symmetry*, 13(7):1176, 2021.

[8] Zhuotong Cai, Jingmin Xin, Siyuan Dong, John A Onofrey, Nanning Zheng, and James S Duncan. Symmetric consistency with cross-domain mixup for cross-modality cardiac segmentation. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1536–1540. IEEE, 2024.

[9] Zhiming Cheng, Shuai Wang, Defu Yang, Jie Qi, Mang Xiao, and Chenggang Yan. Deep joint semantic adaptation network for multi-source unsupervised domain adaptation. *Pattern Recognition*, page 110409, 2024.

[10] Josef Scharinger Christoph Heindl, Sebastian Zambal. Learning to predict robot keypoints using artificially generated images. In *24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2019.

[11] Sebastian Zambal Christoph Heindl, Lukas Brunner and Josef Scharinger. Blendtorch: A real-time, adaptive domain randomization library. In *1st Workshop on Industrial Machine Learning at International Conference on Pattern Recognition (ICPR2020)*, 2020.

[12] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath. Generative adversarial networks: An overview. *CoRR*, abs/1710.07035, 2017.

[13] Chuan-Zhi Dong and F Necati Catbas. A review of computer vision–based structural health monitoring at local and global levels. *Structural Health Monitoring*, 20(2):692–743, 2021.

[14] Gui-Jiang Duan and Xin Yan. A real-time quality control system based on manufacturing process data. *IEEE Access*, 8:208506–208517, 2020.

[15] Kamil Erkan, Christopher Jekeli, and CK Shum. Fusion of gravity gradient and magnetic field data for discrimination of anomalies using deformation analysis. *Geophysics*, 77(3):F13–F20, 2012.

[16] Yijia Feng, Chenhui Ye, Ruoyi Li, Heng Pan, and Dani Korpi. Dda-net: A discrepancy-based domain adaptation network for csi feedback transferability. In *ICC 2023-IEEE International Conference on Communications*, pages 4157–4162. IEEE, 2023.

[17] Alvaro Figueira and Bruno Vaz. Survey on synthetic data generation, evaluation methods and gans. *Mathematics*, 10(15), 2022.

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

[19] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423, 2016.

[20] Liangfu Ge and Ayan Sadhu. Domain adaptation for structural health monitoring via physics-informed and self-attention-enhanced generative adversarial learning. *Mechanical Systems and Signal Processing*, 211:111236, 2024.

[21] Radu Godina and João CO Matias. Quality control in the context of industry 4.0. In *Industrial Engineering and Operations Management II: XXIV IJCIEOM, Lisbon, Portugal, July 18–20 24*, pages 177–187. Springer, 2019.

[22] Grigoriy Gogoshin, Sergio Branciamore, and Andrei S Rodin. Synthetic data generation with probabilistic bayesian networks. *Mathematical biosciences and engineering: MBE*, 18(6):8603, 2021.

[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, 2014.

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

[25] Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville. Improved training of wasserstein gans. *CoRR*, abs/1704.00028, 2017.

[26] Pierre Gutierrez, Maria Luschkova, Antoine Cordier, Mustafa Shukor, Mona Schappert, and Tim Dahmen. Synthetic training data generation for deep learning based quality inspection. In *Fifteenth International Conference on Quality Control by Artificial Vision*, volume 11794, pages 9–16. SPIE, 2021.

[27] Eric Heitz, Kenneth Vanhoey, Thomas Chambon, and Laurent Belcour. Pitfalls of the gram loss for neural texture synthesis in light of deep feature histograms. *CoRR*, abs/2006.07229, 2020.

[28] 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 I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[29] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106–154, 1962.

[30] Fei Hui, Pierre Payeur, and Ana-Maria Cretu. Visual tracking of deformation and classification of non-rigid objects with robot hand probing. *Robotics*, 6(1), 2017.

[31] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[32] Gao-Feng Jiang, Su-Mei Wang, Yi-Qing Ni, and Wen-Qiang Liu. Unsupervised discrepancy-based domain adaptation network to detect rail joint condition. *IEEE Transactions on Instrumentation and Measurement*, 2023.

[33] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. *CoRR*, abs/1603.08155, 2016.

[34] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 105–114, 2017.

[35] Chen-Yu Lee, Tanmay Batra, Mohammad Haris Baig, and Daniel Ulbricht. Sliced wasserstein discrepancy for unsupervised domain adaptation. *CoRR*, abs/1903.04064, 2019.

[36] Seunghun Lee, Sunghyun Cho, and Sunghoon Im. Dranet: Disentangling representation and adaptation networks for unsupervised cross-domain adaptation. *CoRR*, abs/2103.13447, 2021.

[37] Jizhizi Li, Jing Zhang, Stephen J. Maybank, and Dacheng Tao. End-to-end animal image matting. *CoRR*, abs/2010.16188, 2020.

[38] Zewen Li, Wenjie Yang, Shouheng Peng, and Fan Liu. A survey of convolutional neural networks: Analysis, applications, and prospects. *CoRR*, abs/2004.02806, 2020.

[39] Jie Liu, Kechen Song, Mingzheng Feng, Yunhui Yan, Zhibiao Tu, and Liu Zhu. Semi-supervised anomaly detection with dual prototypes autoencoder for industrial surface inspection. *Optics and Lasers in Engineering*, 136:106324, 2021.

[40] Kun Liu, Ying Yang, Xiaosong Yang, Jingkai Wang, Weipeng Liu, and Haiyong Chen. Multi-level joint distributed alignment-based domain adaptation for cross-scenario strip defect recognition. *Journal of Intelligent Manufacturing*, pages 1–14, 2024.

[41] Yingzhou Lu, Minjie Shen, Huazheng Wang, Xiao Wang, Capucine van Rechem, and Wenqi Wei. Machine learning for synthetic data generation: A review, 2024.

[42] C Mares, B Barrientos, and A Blanco. Measurement of transient deformation by color encoding. *Optics express*, 19(25):25712–25722, 2011.

[43] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *CoRR*, abs/1802.05957, 2018.

[44] Francisco J. Moreno-Barea, Fiammetta Strazzera, José M. Jerez, Daniel Urda, and Leonardo Franco. Forward noise adjustment scheme for data augmentation. *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 728–734, 2018.

[45] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning, 2018.

[46] OpenCV. Morphological transformations. https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html, 2024. Accessed: May 16, 2024.

[47] Arianna Patrizi, Giorgio Gambosi, and Fabio Massimo Zanzotto. Data augmentation using background replacement for automated sorting of littered waste. *Journal of Imaging*, 7(8), 2021.

[48] Thomas Pyzdek and Paul A Keller. *Quality engineering handbook*. CRC Press, 2003.

[49] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.

[50] Jean-Francois Rajotte, Robert Bergen, David L. Buckeridge, Khaled El Emam, Raymond Ng, and Elissa Strome. Synthetic data as an enabler for machine learning applications in medicine. *iScience*, 25(11):105331, 2022.

[51] P Ramesh Kumar and A Vijaya. Naïve bayes machine learning model for image classification to assess the level of deformation of thin components. *Materials Today: Proceedings*, 68:2265–2274, 2022. 4th International Conference on Advances in Mechanical Engineering.

[52] Maoqi Ran, Baoping Tang, Peng Sun, Qikang Li, and Tielin Shi. A gradient aligned domain adversarial network for unsupervised intelligent fault diagnosis of gearboxes. *ISA Transactions*, 2024.

[53] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis, 2016.

[54] Preeti Saini and Mr Rohit Anand. Identification of defects in plastic gears using image processing and computer vision: A review. *International Journal of Engineering Research*, 3(2):94–99, 2014.

[55] Fátima A Saiz, Garazi Alfaro, Iñigo Barandiaran, and Manuel Graña. Generative adversarial networks to improve the robustness of visual defect segmentation by semantic networks in manufacturing components. *Applied Sciences*, 11(14):6368, 2021.

[56] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training gans. In D. Lee,

M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[57] Rob BN Scharff, Rens M Doornbusch, Xander L Klootwijk, Ajinkya A Doshi, Eugeni L Doubrovski, Jun Wu, Jo MP Geraedts, and Charlie CL Wang. Color-based sensing of bending deformation on soft robots. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4181–4187. IEEE, 2018.

[58] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.

[59] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Josh Susskind, Wenda Wang, and Russ Webb. Learning from simulated and unsupervised images through adversarial training, 2017.

[60] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[61] Benjamin Staar, Michael Lütjen, and Michael Freitag. Anomaly detection with convolutional neural networks for industrial surface inspection. *Procedia CIRP*, 79:484–489, 2019.

[62] Domen Tabernik, Samo Šela, Jure Skvarč, and Danijel Skočaj. Segmentation-based deep-learning approach for surface-defect detection. *Journal of Intelligent Manufacturing*, 31(3):759–776, 2020.

[63] Muhammad Talha, Sheikh Faisal Rashid, Zain Iftikhar, Muhammad Touseef Afzal, and Liu Ying. Transferable learning architecture for scalable visual quality inspection. In *2022 2nd International Conference on Artificial Intelligence (ICAI)*, pages 26–32, 2022.

[64] Yunchao Tang, Lijuan Li, Chenglin Wang, Mingyou Chen, Wenxian Feng, Xiangjun Zou, and Kuangyu Huang. Real-time detection of surface deformation and strain in recycled aggregate concrete-filled steel tubular columns via four-ocular vision. *Robotics and Computer-Integrated Manufacturing*, 59:36–46, 2019.

[65] Apostolia Tsirikoglou, Gabriel Eilertsen, and Jonas Unger. A survey of image synthesis methods for visual machine learning. In *Computer Graphics Forum*, volume 39, pages 426–451. Wiley Online Library, 2020.

[66] 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.

[67] Hidefumi Wakamatsu, Kousaku Takahashi, and Shinichi Hirai. Dynamic modeling of linear object deformation based on differential geometry coordinates. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 1028–1033. IEEE, 2005.

[68] Xing Wei, Zhaoxin Ji, Fan Yang, Chong Zhao, Bin Wen, and Yang Lu. Self-training domain adaptation via weight transmission between generators. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3280–3284. IEEE, 2024.

[69] Jing Xiang, Guitao Cao, Xinyue Zhang, Hanxiu Zhang, Chunwei Wu, and Hong Wang. Discriminative feature mining and alignment for unsupervised domain adaptation. In *2023 International Joint Conference on Neural Networks (IJCNN)*, pages 01–08. IEEE, 2023.

[70] Mei Xiao, Yu Liu, Lei Zhang, Wenjian Jia, and Shi Dong. Extraction and expression of deformation features for wear images from four-ball friction test: For classification purpose. *Proceedings of the Institution of Mechanical Engineers, Part J: Journal of Engineering Tribology*, 237(2):268–275, 2023.

[71] Zhihong Yan, Feitao Hu, Jing Fang, and Jianpeng Cheng. Multi-line laser structured light fast visual positioning system with assist of tof and cad. *Optik*, 269:169923, 2022.

[72] Shengqing Yao, Yuming Chen, Yanfang Zhang, Zhizhong Xiao, and Jiaojiao Ni. Shared wasserstein adversarial domain adaption. *Multimedia Tools and Applications*, pages 1–11, 2024.

[73] Zhiwei Zhao, Yingguang Li, Changqing Liu, and James Gao. On-line part deformation prediction based on deep learning. *Journal of Intelligent Manufacturing*, 31:561–574, 2020.

[74] Stephan Zheng, Yang Song, Thomas Leung, and Ian Goodfellow. Improving the robustness of deep neural networks via stability training. In *Proceedings of*

the ieee conference on computer vision and pattern recognition, pages 4480–4488, 2016.

[75] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation, 2017.

[76] 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.

[77] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2020.