TECHNISCHE UNIVERSITÄT CHEMNITZ

Faculty of Electrical Engineering and Information Technology
Professorship of Digital Signal Processing and Circuit Technology

Master Thesis

Using Neural Networks for the Domain Adaptation of Synthetic Generated Document Images

Giriraj Sukumar Pawar

Submitted in Fulfilment of the Requirements for the Academic Degree M.Sc.

Chemnitz, July 20, 2021

Supervisor: Prof. Dr.-Ing. Gangolf Hirtz

Advisors: Tobias Scheck M.Sc., Paul Fischer M.Sc.

Pawar, Giriraj Sukumar
Matriculation Number: 551205
Using Neural Networks for the Domain Adaptation of Synthetic Generated Document Images
Master Thesis, Faculty of Electrical Engineering and Information Technology,
Professorship of Digital Signal Processing and Circuit Technology,
Technische Universität Chemnitz, July 2021.

Abstract

Neural networks have improved significantly in past decades. They are competent to solve complex problems in the field of deep learning. Also, they are capable to handle a large amount of data. However, the training of neural networks requires a significantly large amount of annotated data, which is not always possible. It is inevitable to machine learning engineers have to generate synthetic data. Nevertheless, the neural networks trained on synthetic data will not able to perform or generalize well on real data. In recent years, an effective technique named domain adaptation has evolved to address the problem of lack of annotated data. The domain adaptation technique can transform data from one domain to another domain. For example, domain adaptation techniques like image-to-image translation can be used to transform images of zebras into images of horses and vice-versa. In this thesis, the image-to-image translation application is implemented using Cycle-Consistent Adversarial Network (CycleGAN). CycleGAN is evolved variant of Generative Adversarial Network (GAN). It is an unsupervised image-to-image translation method which learns to transform an image from a source domain to a target domain in the absence of paired examples and annotated data. The objective of the thesis is to improve document image classification and reduce the scarcity of annotated data. This application attempts to close the domain gap between synthetic data distribution and real data distribution by generating realistic document images from synthetic document images using CycleGAN. These realistic document images can be used to train a classifier to classify real document images further. This process accelerates the process of annotating new real document images and the scarcity of annotated data can be reduced. In this thesis, several experiments are performed to understand the domain gap between data distributions. Experimental results show generated data distribution matched comparably better to real data distribution than synthetic data distribution and faxified data distribution. With the obtained results during the experiments, it can be said a large number of realistic document images can be generated using CycleGAN to resolve the problem of scarcity of data in the target domain to improve the performance of document image classification models. This thesis only discusses a CycleGAN to solve the problem defined in this thesis. The rest of the methods and comparisons with them are left for future work. Also, CycleGAN can be used for generating realistic images in many tasks like image classification, segmentation, object detection, reconstruction, etc. The aim of this thesis is limited to improve the document image classification due to time constraints.

Keywords: CycleGAN, GAN, Domain Gap, Domain Adaptation, Image-to-Image Translation, Data Distributions.

Acknowledgments

I would like to express my gratitude to my advisors, Tobias Scheck, Paul Fischer, Thaddäus Strobel, Dr. Martin Voigt, and Clemens Reinhardt, who helped me throughout this project. I would like to extend a special thanks to my parents for their enormous support.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

GAN	Generative Adversarial Network
CNN	Convolutional Neural Network
cGAN	Conditional Adversarial Network
LSGAN	Least Squares Generative Adversarial Network
CycleGAN	Cycle-Consistent Adversarial Network
GT	Ground Truth
DIBCO	Document Image Binarization Competition
PSNR	Peak Signal-to-Noise Ratio
ResNet	Residual Network
AMT	Amazon Mechanical Turk
CUT	Contrastive Unpaired Translation
MUNIT	Multimodal Unsupervised Image-to-image Translation
DRIT	Diverse Image-to-Image Translation
GCGAN	Geometry-Consistent Generative Adversarial Networks
FastCUT	Fast Contrastive Unpaired Translation
FID	Fréchet Inception Distance
HTR	Handwritten Text Recognition
OCR	Optical Character Recognition
WGAN	Wasserstein GAN
ML	Machine Learning
DL	Deep Learning
ANN	Artificial Neural Network
AI	Artificial Intelligence
MNIST	Modified National Institute of Standards and Technology database
SVHN	Street View House Numbers
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
COCO	Common Objects in Context
1D	One-dimensional
2D	Two-dimensional

List of Symbols

G Generator
D Discriminator

1. Introduction

1.1 Overview

Artificial Intelligence (AI) has been a game-changer in the computer science domain and has evolved tremendously over the years. AI has a presence in many sectors like Healthcare [2], Autonomous Vehicles [3], Robotics [4], Space Exploration [5], and Computer Vision [6]. This is largely due to the research in Machine Learning (ML) and Deep Learning (DL). Machine Learning is a subdomain of Artificial Intelligence. Machine learning is an art of programming machines, so they can learn from data without being explicitly programmed. Machine learning is used create many AI applications, where it is difficult or unfeasible to develop traditional algorithms to perform the needed tasks. Although Machine Learning and Deep learning domains fall under the category of Artificial Intelligence, there are some important differences between them (figure 1.1). First, deep learning is subdomain of machine learning. Second, deep learning algorithms are powered by Artificial Neural Networks (ANNs), and third, they require less human intervention while extracting features from the data compared to deep learning. Now let us have a look into deep learning and try to understand how it is different from deep learning.

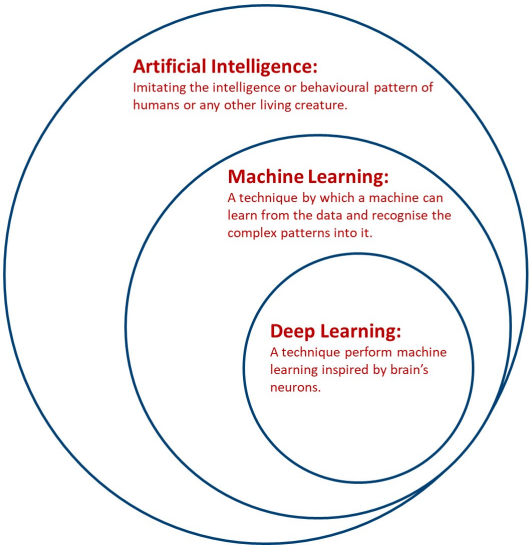


Figure 1.1: Relationship between Artificial Intelligence, Machine Learning, and Deep Learning.

The concept of deep learning was invented back in the 1950s but was largely ignored until the 1980s and 1990s. however, with the efficient, fast computation hardware, and abundant data in this decade it has become a popular research topic among many AI research institutions, organizations, and startups. deep learning is inspired by the biological network of neurons present inside the brain. Deep learning algorithms learn to discover meaningful, complex patterns in the digital representation of data, like sounds and images. To achieve this, deep learning uses a multi-layered structure of algorithms called Artificial Neural Networks (ANNs). ANNs are the heart of deep learning. They are flexible, efficient, and scalable, and suitable for vast and highly complex deep learning tasks like classifying billions of images (e.g., Google Images, Instagram, and Facebook), object detection (e.g., Tesla's Self-driving Cars), improving speech recognition systems (e.g., Apple's Siri, Amazon's Alexa, and Google Assistant), defense systems (Israel's Iron Dome, U.S.A's Patriot Missile System), and recommending the best videos to watch to hundreds of millions of users every day (e.g., YouTube).

Neural Networks are capable enough to solve complex problems by extracting features, recognizing patterns in data efficiently. However, there are certain challenges while training the neural

networks. In short, since usually the task is to select a deep learning algorithm and train it on data, majorly two things that can cause difficulties are bad algorithm and bad data. Let us start with some examples of bad data. The insufficient quantity of training data one of the major problems that occurred while training deep learning models; it takes a lot of data for most deep learning algorithms to work properly. The second is the poor quality data; if the training data is erroneous, noisy, full of outliers will make a model hard to detect patterns in the data. Hence the model will not perform well. The third is the irrelevant features; the model will only learn if the training data has more relevant features than irrelevant features. Hence, it is often worth putting effort and spend time cleaning up training data. Most of the Machine Leaning Engineers spend significant amount of time doing the same.

Now that after some examples of bad data, let's look at some of the examples of bad algorithms. The overfitting of training data; means the model performs well over the training data but generalizes well over unseen test data. Overfitting happens when the model is very complex compared to the amount and noisiness of the training data. The method used to constrain a model to make it simpler and decrease the risk of overfitting is called regularization [7]. Regularization is one of the solutions provided to generalize a model better to the new examples. The second is underfitting the training data; it is the opposite of overfitting, which means the model is too simple to learn the underlying structure of data. Selecting a less complex model, feeding better features during training can solve the problem of underfitting [8]. While training any deep learning model it is required to have good quality and enough quantity of data. But in many cases, it is very difficult to have data that is labeled and annotated. The training of the deep learning model is highly influenced by the quality and quantity of the data that has been used for the training of the model. But in many cases, it is very difficult to have data that is labeled and annotated.

In the following sections, the motivation behind this thesis discussed in Section 1.2 and the problem statement is discussed in Section 1.3. The objectives of this thesis discussed in Section 1.4. The thesis structure and limitations are discussed in Section 1.5. Finally, the terminologies used in this thesis are defined in Section 1.6.

1.2 Motivation

In recent years, deep learning methods have shown great effectiveness in many fields, including natural language processing and computer vision. However, such kinds of methods are still limited by a poor generalization due to the insufficient quantity of training data [9]. Annotated data are scarce when it comes to developing deep learning models for computer vision applications. The performance of such deep learning models can be improved with the introduction of a large amount of annotated data. Though, it is hard to obtain, due to the high cost of data annotation, specifically in the case of numerous variants of data. To overcome the obstacle of the scarcity of annotated data, there are some methods available to tackle this problem. The popular methods are Active Learning [10], Data Augmentation [11], Transfer Learning [12], and Domain Adaptation [13].

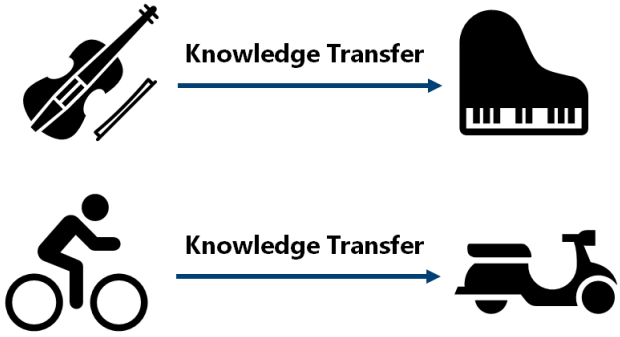


Figure 1.2: Simple examples of transfer learning.

This thesis aims to solve the data scarcity problem using the domain adaptation method. Domain adaptation is a subcategory of transfer learning in which the task is to transfer the knowledge from the source domain to the target domain. Before discussing domain adaptation, let us have a look into transfer learning. Transfer learning is the ability of a system to recognize and apply the knowledge learned from one task to another task [12]. For example, if a person has mastered one musical instrument like the violin can learn piano faster compared to others, as the knowledge of one musical instrument can be applied while learning another musical instrument. Figure 1.2 show intuitive example of transfer learning. Transfer learning inspired by human behavior, humans beings are capable to transfer knowledge from one domain to another domain. The transfer learning grasps knowledge from the source domain to improve the learning performance in the target domain so the number of labeled samples required in the target domain can be reduced. It is important to mention, transfer learning is effective only if the source domain and target domain are related. For example, learning bicycles will not help to learn piano faster. Qiang Yang et al. [14] have performed survey on transfer learning, more information about the transfer learning can found in their research paper.

The difference between traditional machine learning and domain adaptation is shown in figure 1.4. When the source and target domains are the same and learning tasks also the same, then such a learning problem becomes a traditional machine learning problem [14]. When the source and target domains are different but related, and learning tasks are the same then, such a learning problem becomes a domain adaptation problem [14]. In domain adaptation, source and target domains have the same feature space but different distributions in contrast to transfer learning, which includes cases where the target domain's feature space is different from the source domain's feature space. Domain adaptation is distinguished depending upon the similarity or dissimilarity of feature space and availability of annotated data in the source domain and target domains. The domain adaptation has two categories, if the feature space is the same between the source domain and target domain is called homogeneous domain adaptation. If the feature space is different between the source and target domain is called heterogeneous domain adaptation. Further homogeneous and heterogeneous domain adaptation divided into three types of domain adaptation, supervised domain adaptation, semi-supervised domain adaptation, unsupervised domain adaptation. In the supervised domain adaptation, the samples in source and target domains are labeled. Semi-supervised domain adaptation has a small set of samples labeled in the target domain. And, in unsupervised domain adaptation, the samples in the source and target domain are not labeled [14]. A simple example of domain adaptation of Street View House Numbers (SVHN) transformed into Handwritten Digits shown in figure 1.3.

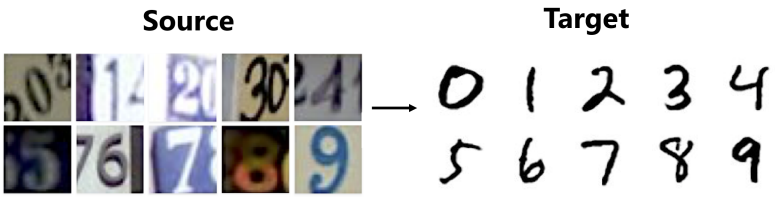


Figure 1.3: Simple example of domain adaptation from SVHN dataset [15] to MNIST Dataset[3]for the digit recognition task.[4]

To understand how domain adaptation works, let's have a simple example. Consider the source domain represented by the SVHN dataset. It is a collection of house number images, and the target domain represented by the MNIST dataset, which is a collection of handwritten digit images. When the CNN is trained and evaluated on the source domain SVHN dataset for the task of identifying numbers it will achieve good accuracy. However, the same classifier will perform worst when evaluated theMNIST dataset. This performance gap occurs due to differences between the domain data distribution. The images in the SVHN dataset consist of different fonts, blur, noise, and different backgrounds. But the images in the MNIST dataset contain a clean background and handwritten

[1] `http://yann.lecun.com/exdb/mnist/` last access: 31.03.2021

[2] `https://machinelearning.apple.com/research/bridging-the-domain-gap-for-neural-models` last access: 07.06.2021

strokes. Now consider images are scarce in the target domain. Only a small amount of the target domain images are available, which are unlabeled. As we know training a classifier using a smaller amount of data leads to underfitting and eventually leads to the worst performance on unseen data. Hence, to create a sufficient amount of data, the domain adaptation model is trained. It learns to transfer the underlying knowledge from the source domain to the target domain. In this case, labeled data is available in the source domain, and unlabeled data available in the target domain. Such a setup is called unsupervised domain adaptation because the model learns to transform images from one domain to another in the absence of labeled data in the target domain, and without taking much help from the labeled data from the source domain during the learning process. Using this domain adaptation model, large amount of annotated data can be created in the target domain by transforming source domain images into target domain images. Once a sufficient amount of images present in the target domain, the task to identify numbers in the target domain can be improved significantly. Nowadays, the domain adaptation technique is widely used in the field of Handwritten Text Recognition (HTR) [16], Image Classification [14], Style Transfer [17], and Optical Character Recognition (OCR) [18] to solve the problem of scarcity of data. To make these applications robust and efficient comes under a very big scope. The scope of this thesis is limited to image classification tasks.

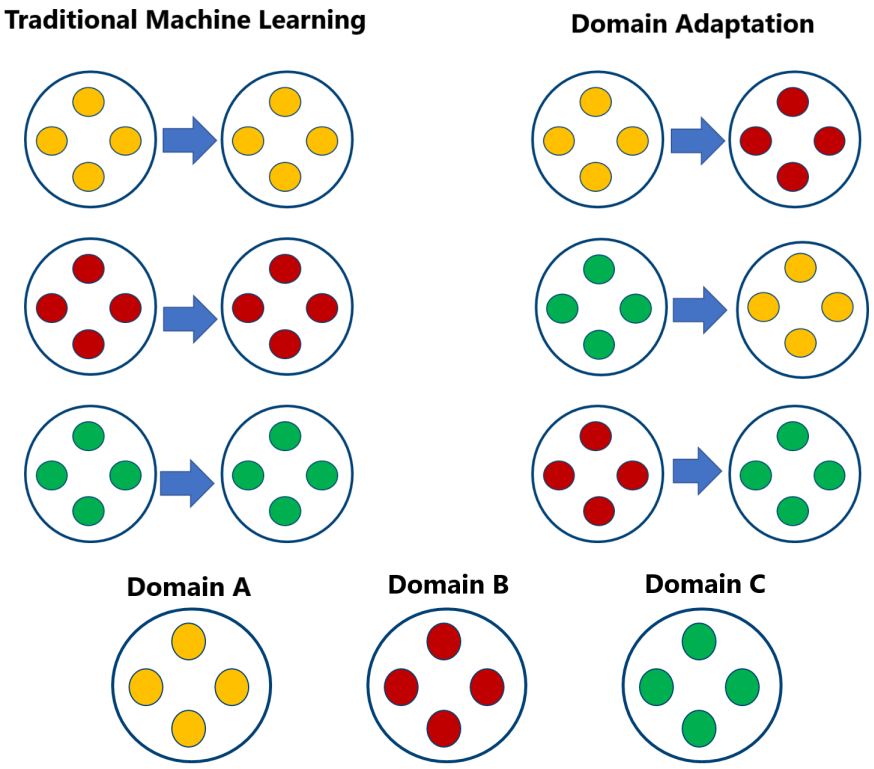


Figure 1.4: Difference between traditional machine learning and domain adaptation.

1.3 Problem Statement

As mentioned earlier, annotated data are scarce when it comes to the training of neural networks, and labeling a large set of data is a costly and tedious job. For example, real document images (figure A.4) with different types of handwriting. In such cases, machine learning engineers have to inevitably generate synthetic data. However, deep learning models trained using synthetic data will not generalize well on real data [9] (figure 1.5). The synthetic data lacks realism [9]. It does not possess a similar noise distribution as real data [9]. Hence, in the last two decades, numerous domain adaptation methodologies have been introduced [18]. Such methods are used to transform synthetic data into realistic data by reducing the divergence between the distribution of real data and the distribution of synthetic data. In this thesis, a domain adaptation application is developed

using CycleGAN [19] to reduce the domain gap between synthetic data distribution and real data distribution. The CycleGAN is an extended variant of Generative Adversarial Network (GAN) [20]. This application is designed in consultation with, ML developers at Elevait Deutschland GmbH, a Germany-based company that develops AI applications for business use-cases. Elevait is widely contributing in the field of Cognitive Business Robotics [21] to automate document processing. Elevait has developed state-of-the-art HTR and OCR tools to process documents and extract information from documents. To make those systems robust and efficient a large number of document images are required. The idea is to create a large number of synthetic document images to have a large quantity of annotated data. However, the synthetic document images will not generalize well when they have to process real document images because the real document images have different noise distribution. Furthermore, artifacts such as salt-and-pepper, background noise, blur due to camera motion or shake, watermarks, stains, wrinkles, and fading text are often introduced during the scanning process. Hence this application is used to transform synthetic document images into realistic document images. This application is capable to capture the noise distribution of real documents and transform an image from the source domain to the target domain. Such a kind of transformation is called image-to-image translation.

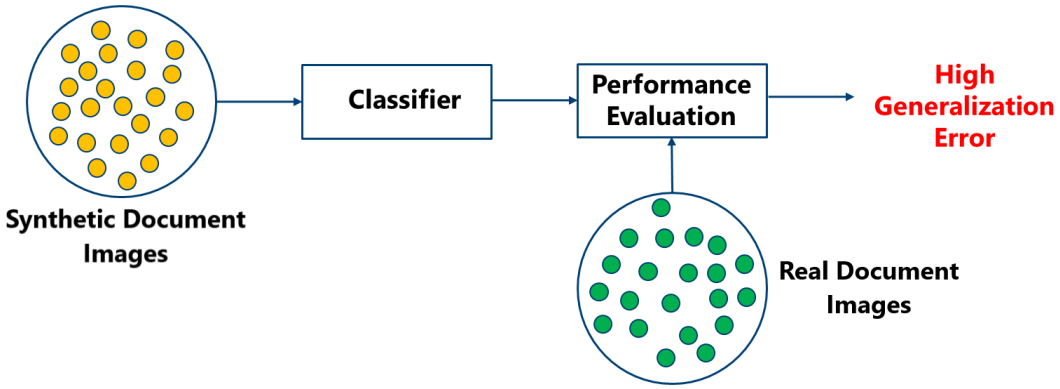


Figure 1.5: Illustration of the problem this thesis aims to solve.

The synthetic document images are created using unfilled form images (figure A.3) and handwritten crops (figure 5.1) retrieved from handwriting datasets like MNIST (figure A.1) or any other datasets. These unfilled form images contain fields like are customer number, customer name, and other customer information. These fields are represented by using bounding boxes. The bounding boxes are annotated using Common Objects in Context (COCO) annotations [22]. In this thesis, the handwritten crops are inserted over the unfilled form images to generate numerous synthetic document images. As mentioned earlier deep learning models trained using synthetic document images will not generalize well on real document images. In this thesis, the handwritten crops are inserted over the unfilled form images to generate numerous synthetic document images. As mentioned earlier deep learning models trained using synthetic document images will not generalize well on real document images. Further, the image-to-image translation application is developed using CycleGAN to transform synthetic document images into realistic document images. ultimately, to reduce the domain gap between synthetic data distribution and real data distribution.

A large number of realistic document images can be generated to tackle the problem of data scarcity in the target domain. As we already know that real document images are scarce and labeling images is a tedious and costly job. Our image-to-image translation application transforms synthetic document images to realistic document images to solve the following problems. First, We have labeled synthetic document images in the source domain, ultimately we get labeled, transformed realistic document images in the target domain. So the problem of labeling, annotations, and scarcity is solved. Second, Consider we have a huge chunk of real document images which are not labeled. If the simple classifier is trained on realistic document images. Later, these chuck of real document images can be automatically classified with ease. This will save data annotation and data collection efforts. Further, making image classification tools in the target domain robust and efficient even in the absence of real data. The figure 1.5 describes if the classifier is trained using

synthetic document images, it won't generalize well on the unseen real document images. Further, it leads to high generalization error, hence represented in dark red color for better understanding. In figure 1.6 Data distribution in light green color represents realistic data which is generated by the domain adaptation method. Dark green represents the real data. The proposed solution is to use domain adaptation methods like the image-to-image translation to reduce the divergence between data distributions. Further, the classifier is trained using realistic document images, it generalizes better on the unseen real document images compared to the classifier are trained using synthetic document images, hence low generalization error represented in light red color for better understanding compared to figure 1.5.

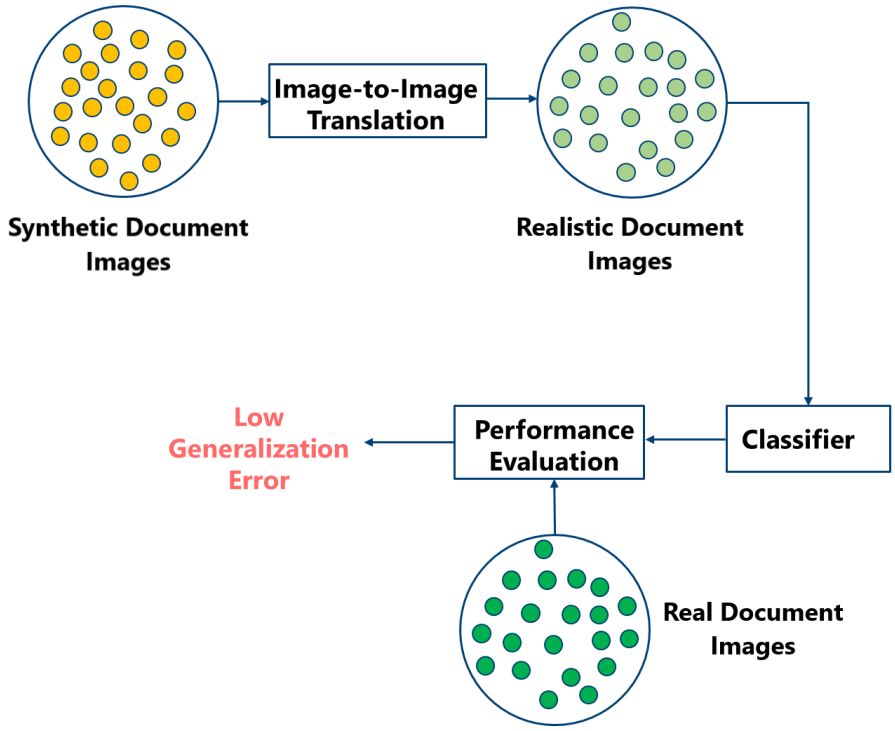


Figure 1.6: Illustration of the proposed solution to reduce the domain gap between synthetic document images and real document images.

1.4 Thesis Objectives

Every research work comes with definite objectives to achieve. The aim of the thesis is develop a image-to-image translation application to perform domain adaptation, and close and analyze the domain gap between synthetic data distribution image and real data distribution.

- Objective 1 is to perform a literature survey in which different types of image-to-image translation methods should be discussed. Also, the theoretical comparison between existing methods must be presented, to finally, deciding a methodology to solve the problem statement.
- Objective 2 is to create and collect datasets to train CycleGAN, by keeping small testing datasets of real images aside to evaluate models.
- Objective 3 is to proceed with the design, implementation, and training of CycleGAN and classifiers.
- Objective 4 is to perform experiments, to determine the quality of images generated by the CycleGAN and understand the domain gap between synthetic data distribution and real data distribution.

- Objective 5 is to document all the information regarding the thesis, for example, chosen methodology, fundamentals required to understand the work, experiments, results, limitations, conclusion, and future work.

1.5 Thesis Limitations and Structure

The domain adaptation field is promoting incremental findings. Hence, this thesis has its limitations due to time deadlines. This image-to-image application is implemented only using CycleGANs. The implementation and comparison with other methodologies are placed apart for future work. The scope of this thesis is limited to the improvement of document image classification by increasing the quality and quantity of labeled data and reducing the domain gap between real data distribution and CycleGAN generated data distribution. The rest of this thesis is organized as follows. Chapter 2 briefly reviews related works of numerous GANs variants. Chapter 3 discusses the about GANs and Convolution Neural Networks (CNNs). The decided method is described in Chapter 4, the implementation and experimental results are presented in Chapter 5. Finally, this work has been concluded in Chapter 7 along with the future work and the limitations.

1.6 Terminology

In this thesis for simplicity and readability, many terms are explained beforehand for better understanding and to avoid confusion. The following terms described in this thesis are consistently used throughout this thesis. The term “Artificial Neural Networks (ANNs)” will be used as “Neural Networks” interchangeably.

2. Related Works

There have been numerous amounts of research in the field of domain adaptation. Several variants of GANs are evolved over the years to resolve various problems. Especially the image-to-image translation methods are improved significantly. This chapter aims to discuss different image-to-image translation methods. Also, a brief comparison upon existing methods carried out in section 2.2. Lastly, section 2.3 concludes the motivation behind choosing a particular approach.

2.1 Literature Survey

Ian J. Goodfellow et al.[20] proposed a framework of GANs in which two models are simultaneously trained. A generative model that captures the data distribution, and a discriminative model that estimates the probability that a sample came from the training data rather than a generative model. The training procedure for a generative model is to maximize the probability of a discriminative model making a mistake. Basically, the generator learns to generate plausible data. The discriminator learns to distinguish the generator's fake data from real data. The discriminator penalizes the generator for producing implausible results. When training begins, the generator produces fake data, and the discriminator quickly learns to tell that it's fake and the generator penalized to produce plausible results. As training progresses, the generator gets closer to producing output that can fool the discriminator. Finally, if generator training goes well, the discriminator gets worse at telling the difference between real and fake. At the end of the training, eventually, we have a generator model which produces plausible results which are similar to real data. Authors have trained GAN an a range of datasets including MNIST [23], the Toronto Face Database (TFD) [24], and CIFAR-10 [25]. Also compared against already existing methods like DBN [26], Stacked CAE [26], and Deep GSN [27] . The authors do not claim that the samples generated by GANs are better than samples generated by already existed methods. Authors believe that these samples are at least competitive with the better generative models in the literature and highlight the potential of the generative adversarial framework. The advantage of using GANs is primarily computational. Adversarial models may also gain some statistical advantage from the generator network not being updated directly with data examples, but only with gradients flowing through the discriminator using backpropagation. Special care should be taken during training the GANs, the generator must not be trained too much without updating the discriminator, to avoid the Helvetica Scenario [28] in which the generator collapses to produce the same output (or a small set of outputs) over and over again. Usually, GANs should produce a wide variety of outputs. The Helvetica Scenario is also called Mode Collapse [29].

Xudong Mao et al.[31] proposed another variant of GANs called Least Squares Generative Adversarial Networks (LSGANs). The Regular GANs hypothesize the discriminator as a classifier with the sigmoid cross-entropy loss function. However, they found that the cross-entropy loss function may lead to the vanishing gradients problem during the learning process. To overcome such a problem, the authors proposed the LSGANs which adopts the least-squares loss function for the discriminator. The least-squares loss function penalizes the fake samples and forces the generator to generate samples toward the decision boundary. The authors evaluated LSGANs using two datasets LSUN [32] and HWDB1.0 (Handwritten Chinese Character Dataset) [33]. When trained on the LSUN dataset [32] they observed, the images generated by LSGANs are of better quality than the ones generated by the two baseline methods, DCGANs [34] and EBGANs [35]. Also, when trained on the Handwritten Chinese Character Dataset, the generated characters were readable and clear. Another experiment was conducted to evaluate the stability of LSGAN on a Gaussian mixture distribution dataset, which is designed in literature [36]. They train LSGANs and regular GAN on a 2D mixture of 8 Gaussian datasets using a simple architecture, where both the generator and the discriminator contain three fully-connected layers. It is observed that regular GANs suffer from mode collapse. GANs generate samples around a single valid mode of the data distribution. But LSGANs learn

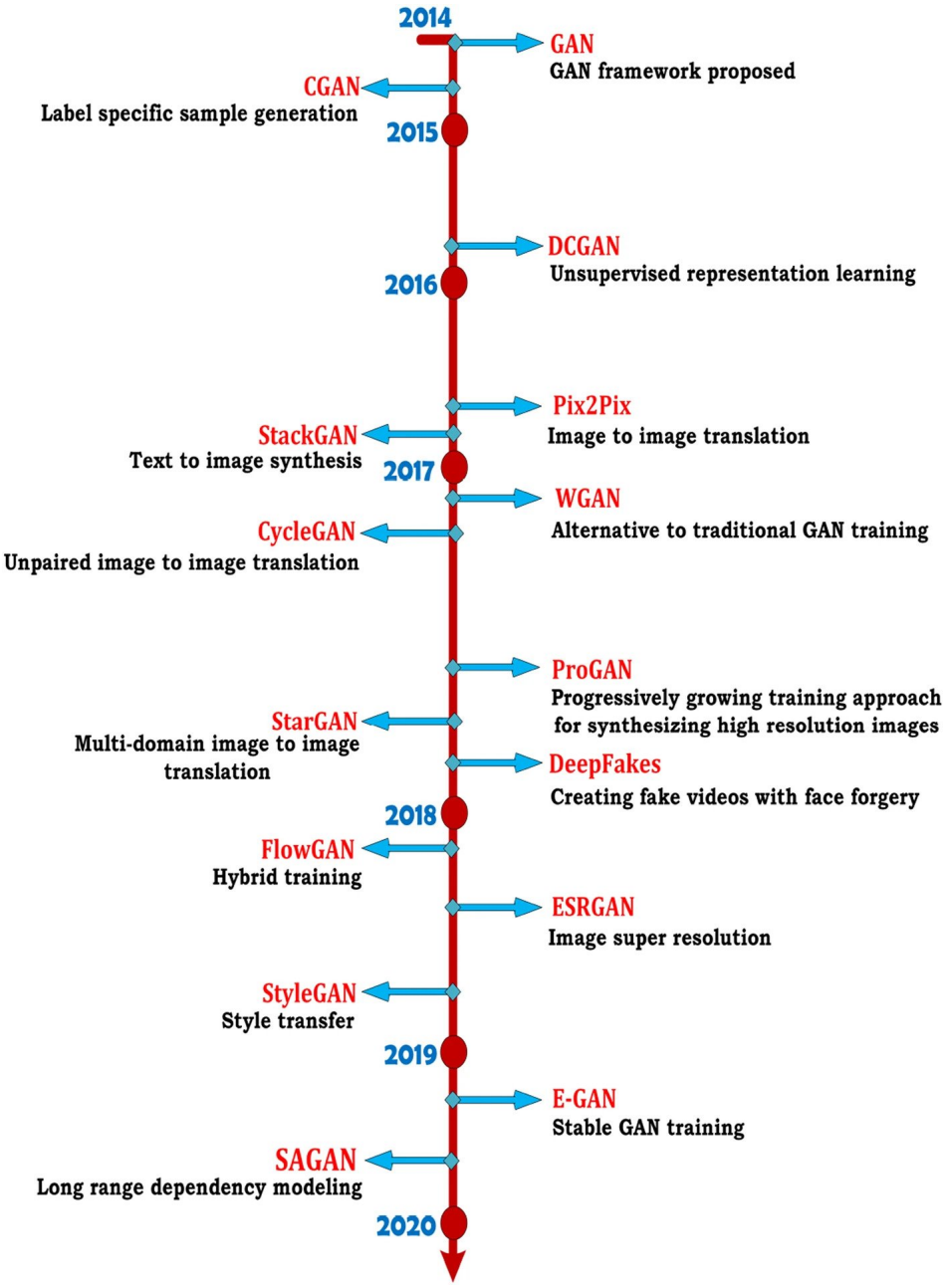


Figure 2.1: Evolution of GANs Over the Years [30].

the Gaussian mixture distribution successfully. In this paper, numerous comparison experiments for evaluating the stability are conducted and the results demonstrate that LSGANs can generate higher quality images than regular GANs, DCGANs [34], and EBGANs [35] and perform more stable than regular GANs during the learning process.

Phillip Isola et al.[37] proposed cGANs as a generic solution to numerous image-to-image translation problems. The authors wanted to provide a single solution for multiple types of image-to-image translation problems. For example, synthesizing photos from label maps [38], reconstructing objects from edge maps [39] [40] , and colorizing images. cGANs not only learn the mapping from input image to output image, but also learn a loss function to train this mapping. This makes it possible to apply the same generic approach to problems that traditionally would require very different loss formulations. Authors implemented cGAN and released it as the pix2pix software to solve distinct image-to-image translation problems. This software is popular among a large number of internet users, many of them are artists, because of its wide applicability and ease of adoption without the need for parameter tweaking. In cGAN the generator uses U-Net-based architecture [41], the U-Net is an encoder-decoder with skip connections between mirrored layers in the encoder and decoder stacks. The discriminator uses a convolutional PatchGAN classifier [42], which only penalizes structure at the scale of image patches. Unlike unconditional GANs, both the generator and discriminator observe the input images. Authors performed multiple experiments during ablation studies using evaluation metrics like, Amazon Mechanical Turk (AMT) perceptual study, FCN-Score, and semantic segmentation metrics. They found that L1 distance loss encourages less blurring compared to the L2 distance loss. Also combining L1 Loss and cGAN (L1 + cGAN) generates better results compared to combining Unconditional GAN and L1 Loss ((L1 + GAN)). The cGAN appears to be more effective on the problem where the output is highly detailed or photographic. The pix2pix software code is available at GitHub. In figure 2.2 the mapping from edges →photo transformation illustrated, and both generator and discriminator are conditioned on auxiliary information like input edge map.

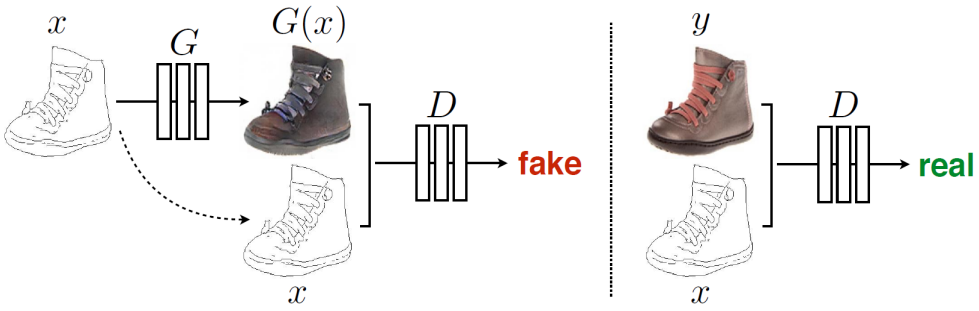


Figure 2.2: Illustration of training a cGAN to map edges →photo transformation. Unlike unconditional GANs both discriminator and generator observe the input edge map [37].

Taesung Park et al.[43] proposed a framework for encouraging content preservation in unpaired image-to-image translation problems by maximizing the mutual information between input and output with patchwise contrastive learning [44]. In the patchwise contrastive learning for image-to-image translation, a generated output patch should appear closer to its corresponding input patch in comparison to other random patches present in the same input. To achieve patchwise contrastive learning, drawing patches internally from within the input image, rather than externally from other images in the dataset, forces the patches to better preserve the content of the input. This method requires neither a memory bank nor specialized architecture. The authors demonstrated that the framework enables one-sided translation in the unpaired image-to-image translation setting while improving quality, consuming less memory, and reducing training time. They call this approach Contrastive Unpaired Translation (CUT). Since contrastive representation is formulated within the same image, this method can even be trained on single images, where each domain is having only a single image. The several prior methods like CycleGAN [19], Multimodal Unsupervised Image-to-image Translation (MUNIT) [45], Diverse Image-to-Image Translation (DRIT) [46], and Geometry-Consistent Generative Adversarial Networks (GCGAN) [47] were unable to achieve signif-

icant results compared to the CUT method, on other hand, it often produced higher quality images and more accurate generations with light footprint in terms of training speed and GPU memory usage. Since CUT method is one-sided, it is memory efficient and faster compared to prior baselines. The evaluation metrics like Fréchet Inception Distance (FID) [48] and semantic segmentation scores are used to compare the quality of generated images using CUT method. Furthermore, the authors also introduced faster and lighter variant Fast Contrastive Unpaired Translation (FastCUT). FastCUT also produces competitive results with even lighter computation cost of training. The code and models for CUT are available at GitHub.

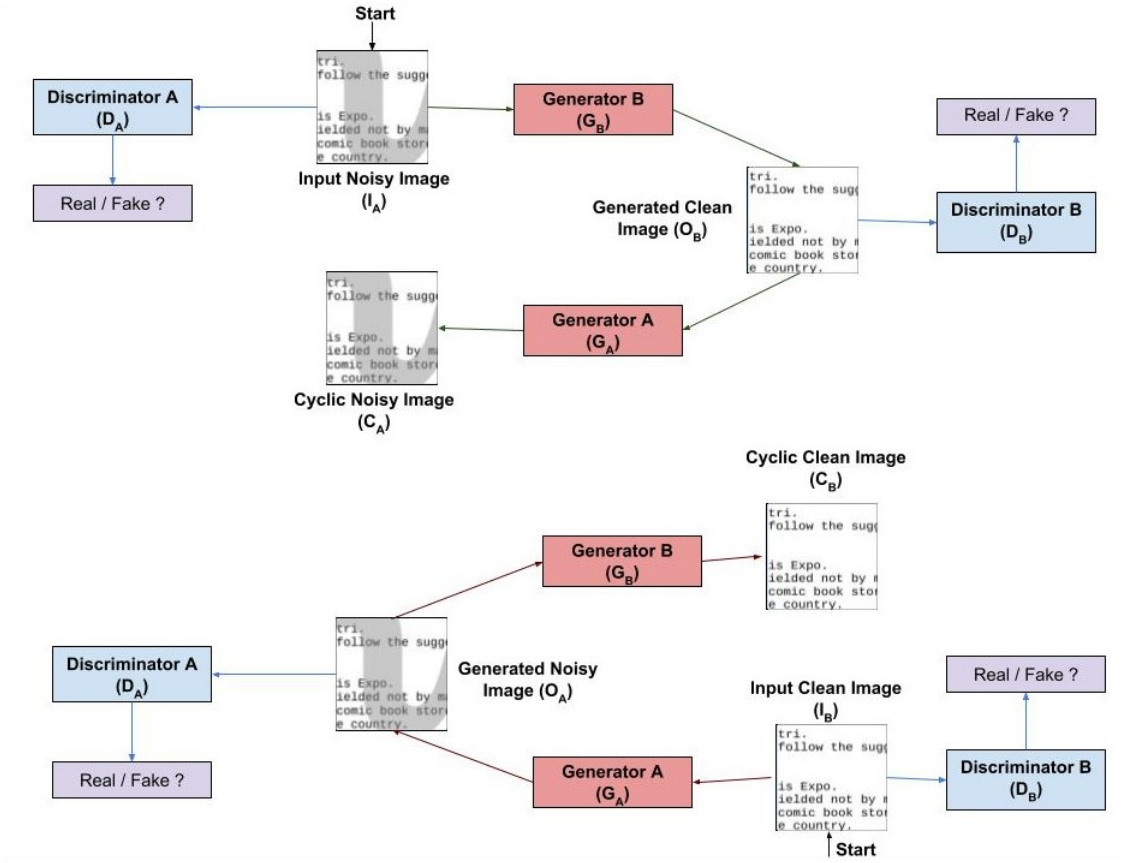


Figure 2.3: Illustration of CycleGAN transforming the noisy document images into clean document images. The generators G_A and G_B are responsible for the mapping of noisy images to clean images using cycle-consistency loss [19]. And the two discriminators D_A and D_B rejects samples generated by D_A and D_B acting like an adversary [49].

Monika Sharma et al.[49] is addressing a problem in the scanning process that often results in the introduction of salt and pepper noise, blur due to camera motion, or shake, water markings, coffee stains, wrinkles, or faded data. These artifacts pose many readability challenges to current text recognition algorithms and significantly degrade their performance. So, the existing denoising techniques require a dataset comprising of noisy documents paired with cleaned versions of the same document. However, very often in the real world, such a paired dataset is not available to train a model to generate clean documents from noisy versions. Hence, the authors proposed to use CycleGAN because it is known to learn a mapping between the distributions in the absence of paired training dataset. Using CycleGAN, noisy document images transformed into denoised and clean document images to achieve image-to-image translation. They have compared the performance of CycleGAN for document cleaning tasks with a cGAN by training them over the same dataset. The only difference was CycleGAN trained using unpaired images and cGAN trained using the paired images. They have used Peak Signal-to-Noise Ratio (PSNR)[1] as a evaluation metric to evaluate the quality of transformed denoised images. Several experiments were performed on 4 separate

[1]Peak Signal-to-Noise Ratio: `http://www.ni.com/white-paper/13306/en/` last access: 31.03.2021.

document public document datasets, one each for background noise removal, deblurring, watermark removal, and defading. Finally, they illustrate that CycleGAN learns a more robust mapping from the space of noisy to clean documents compared to cGAN. The complete architecture of CycleGAN for denoising documents illustrated in figure 2.3.

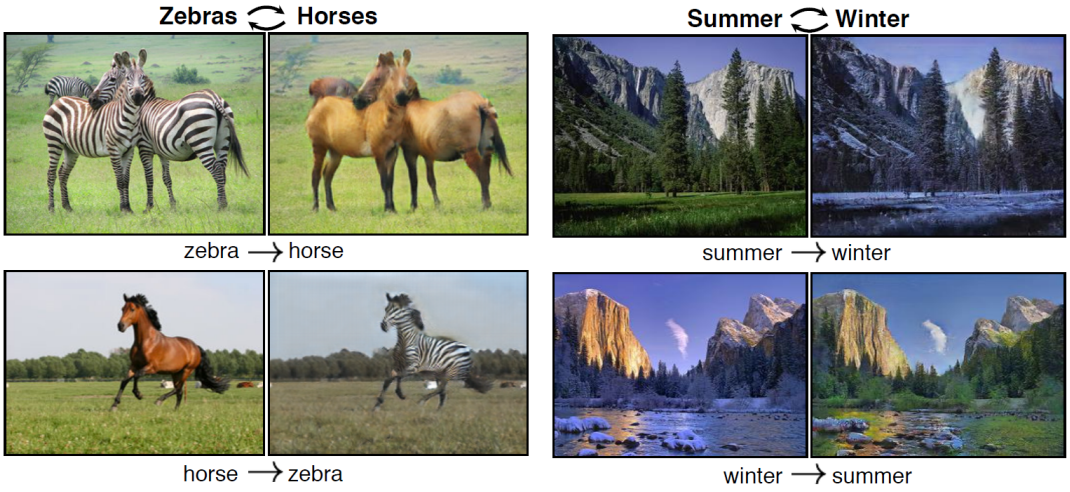


Figure 2.4: Illustration of CycleGAN transforming an image from one into the other and vice versa. For example, zebra image transformed into horse image, and vice versa. The view of Yosemite mountains in summer transformed into winter, and vice versa [19].

Chris Tensmeyer et al.[9] realized solving binarization tasks using deep learning models is very challenging. It is due to the lack of large quantities of labeled data available to train such models. They also mention there have been efforts to create synthetic data for binarization using image processing techniques but, they generally lack realism. In this paper, the authors proposed a method to produce realistic synthetic data using an adversarially trained image translation model. They extended the popular CycleGAN model to be conditioned on the ground truth binarization mask as it translates images from the domain of synthetic images to the domain of real images. They have found that modifying the discriminator to condition on the binarization Ground Truth (GT) leads to increased realism and better agreement between the GT and the produced image. They called the proposed model DGT-CycleGAN. Also shown DGT-CycleGAN model produces more realistic synthetic data. They validated their approach by pretraining deep networks on realistic synthetic datasets generated by DGT-CycleGAN, CycleGAN, and image compositing. They evaluate both pretrained only and finetuned models on each of the Document Image Binarization Competition (DIBCO) datasets. They have concluded that pretraining deep neural networks on the more realistic synthetic data generated using DGT-CycleGAN lead to better predictive performance both before and after finetuning on real data.

Jun-Yan Zhu et al.[19] proposed a modified version of GAN which is the state-of-the-art method for the image-to-image translation that can transform the images from source domain X to target domain Y in the absence of paired training dataset. This method can learn to capture special characteristics of one image collection and figuring out how these characteristics could be translated into the other image collection, all in the absence of any paired training examples. This modified version of GAN is called CycleGAN. In this method, the goal is to learn a mapping $G : X \rightarrow Y$ such that the distribution of images $G(X)$ is indistinguishable from the distribution Y using an adversarial loss. Because this mapping is highly under-constrained and coupled with an inverse mapping $F : Y \rightarrow X$ to introduce a cycle consistency loss to enforce $F(G(X)) \approx X$ and vice versa. Along with an adversarial loss and cycle consistency loss, this work also introduces identity mapping loss which helps to preserve the colour of input images. Authors considered evaluation metrics like AMT perceptual studies, FCN-score [37], and semantic segmentation metrics to compare the quality of generated images against other baseline. The several prior methods like Bi-GAN/ALI [[50], [51]], CoGAN [52], SimGAN [53] were unable to achieve compelling results. The CycleGAN method, on

other hand, can produce images that are often of similar quality to the fully supervised pix2pix [37]. Authors provide both PyTorch and Torch implementations. In figure 2.4 examples of CycleGAN transforming an image from one into the other and vice versa illustrated.

2.2 Discussion

Over the years, GANs have evolved to solve different kinds of problems. The evolution of the GANs is illustrated in the figure 2.1 using a timeline diagram. GANs suffer from unstable training, vanishing gradients problem, and mode collapse. Consequently, Martin Arjovsky et al.[54] proposed Wasserstein GAN (WGAN) to solve problems with GANs. WGANs improve the stability of learning, get rid of problems like mode collapse, and provide meaningful learning curves useful for debugging and hyperparameter searches. Although the implementation of WGAN is straightforward, the theory behind it is heavy and requires some hack, for example, Weight Clipping [55]. Hence, Xudong Mao et al.[31] proposed a simple and more intuitive method compared to WGAN, called LSGAN. First, LSGANs are able to generate higher quality images than regular GANs. Second, LSGANs perform more stable during the learning process. Moreover, Jun-Yan Zhu et al.[19] proposed CycleGAN. It is an unsupervised image-to-image translation approach. Compared to GANs, CycleGANs can deal more meticulously with the problems like unstable training, vanishing gradients problems, and mode collapse. Also, they described that CycleGAN has outperformed existing baselines as Bi-GAN/ALI [[50], [51]], CoGAN[52], and SimGAN [53]. Furthermore, Monika Sharma et al.[49] proclaimed CycleGAN can transform noisy document images into denoised and clean document images to achieve image-to-image translation. They have also compared the performance of the developed image-to-image application with cGAN and demonstrated CycleGAN had outperformed cGAN. Chris Tensmeyer et al. [9] proposed DGT-CycleGAN, a modified version of CycleGAN, which is adequate to translate images from the domain of synthetic images to the domain of real images to solve binarization tasks. Moreover, Taesung Park et al. [43] proposed CUT. They have demonstrated that CUT is a better, faster, and memory-efficient approach to perform unsupervised image-to-image translation. It has outperformed CycleGAN [19], MUNIT [45], DRIT [46], and GCGAN [47]. However, due to time constraints, multiple available references, and code repositories, CycleGAN has been a choice for this thesis and research to perform unsupervised image-to-image translation.

2.3 Conclusion

The image-to-image translation is a class of computer vision and computer graphics problems. In which the goal is to learn the mapping between a source image and target image using a training set of aligned image pairs. Although, for many tasks, aligned or paired training data will not be available. This thesis is attempting to close a domain gap between synthetic document images and real document images in the absence of paired training data. Collecting and annotating paired document images is gruelling, time-consuming, and costly. The thesis aims to develop an image-to-image translation application to transform synthetic document images into realistic document images to perform domain adaptation, by closing the gap between synthetic data distribution and real data distribution. Ultimately, a large amount of realistic annotated set of document images can be generated using this application. Furthermore, they can be used to improve document image classification. Also, in case, if a classifier trained using such generated realistic document images, it can fasten the process of labeling the new unlabeled real document images. As per the above literature survey [19] [49] and problem statement, CycleGAN could be a remarkable approach to transform synthetic document images into realistic document images in the absence of paired training data. Hence, in this thesis, the image-to-image translation application is realized using CycleGAN.

3. Fundamentals

This chapter aims to develop a better understanding of fundamental concepts required to understand this thesis. It discusses the mathematics and working principle of the GANs. It also discusses CNNs and its layers. The formulation and architecture of GANs is explained in Section 3.1. In Section 3.2, layers like convolution layer, activation layer, pooling layer, and fully connected layer have briefly explained.

3.1 Generative Adversarial Networks (GANs)

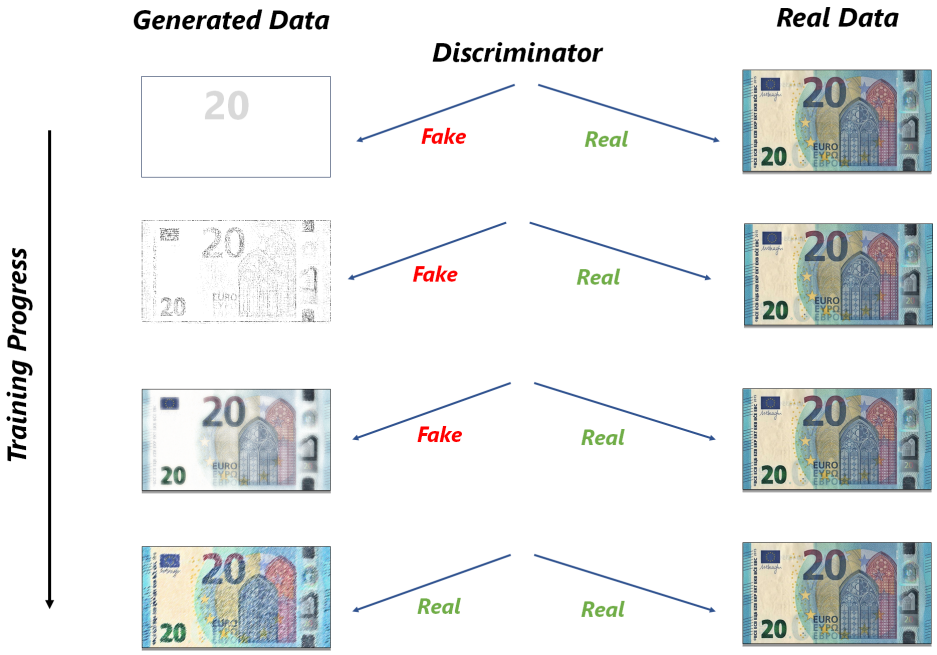


Figure 3.1: Intuitive example of GAN training progress.

Ian J. Goodfellow et al.[20] proposed GANs. It has two models in its architecture, one is generator, and another is discriminator. Authors mentioned both can be implemented using multilayer networks. Normally, the term multilayer networks indicated for CNNs or ANNs. Let's try to understand the function of GANs intuitively. Consider generator is a forger who creates fake images of currency, with the intention of making them realistic as much as possible. Furthermore, the discriminator is an expert, which receives both forged and real images, and distinguishes between real and forged images. The discriminator has access to images generated by the generator and real images present in the dataset. The generator generates fake images without having access to the real images using random noise distribution. Both generator and discriminator are trained simultaneously. The discriminator learns by the error signal provided by the ground truth of distinguishing whether the image came from the dataset of real images or from the generator. The generator learns using the same error signal given by the discriminator to produce a better quality of forged or fake images. It is a setup where both generator and discriminator are competing with each other.

In figure 3.1 an intuitive example of GAN training process is displayed. During the initial stage of the training, images produced by the generator are easily distinguishable by the discriminator as fake images. After a certain number of iterations, the quality images generated by the generator

increases by the feedback error signal given by the discriminator. Also, the discriminator gets smarter and smarter to distinguish fake images and real images, like mentioned earlier both models are trained simultaneously. But at a certain point generator starts to produce realistic images which the discriminator classifies as real. In figure 3.2, we can see GAN in action. Let us have a look into the GAN's core architecture. As described earlier, the task of the GAN is to generate fake samples. Hence, the training dataset has a set of real data samples, from which the generator learns to create fake samples that are similar to the real samples. This set of real data samples serves input X to the discriminator D. The random noise vector Z retrieved from the random noise distribution serves as an input to the generator to produce fake samples. The generator G takes Z as an input and outputs a fake sample X'. Its goal is to create fake samples that are indistinguishable from the real samples from the training dataset. The discriminator D takes input from real data sample X that are present in the training dataset and fake samples X' generated from the generator G. For each sample, it determines and outputs the probability of the sample that is real. For every discriminator's prediction, The classification error is backpropagated to update both generator and discriminator during iterative training. The discriminator's weights are updated to maximize classification accuracy which means, maximizing the probability of correct prediction X as real and X' as fake. The generator's weights are updated to maximize the probability that the discriminator misclassifies X' as real.

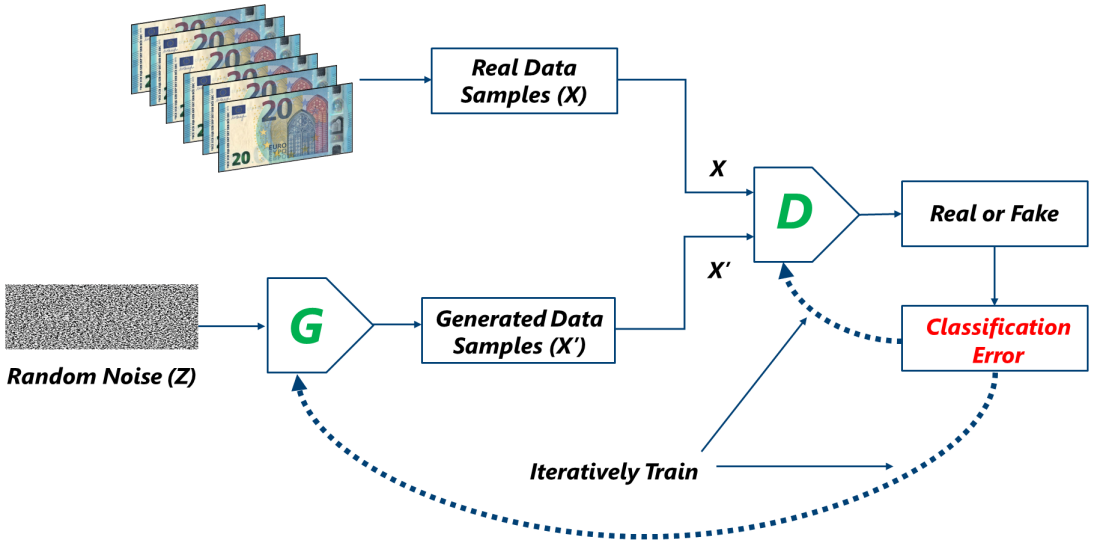


Figure 3.2: Overview of core GAN architecture along with the generator's and discriminator's inputs, outputs, and their interactions.

Now let's try to understand the mathematics behind the GANs. For learning the generator's distribution p_g over data x, the input noise variable defined as $p_z(z)$. The generator G and discriminator D are the differentiable functions represented by multilayer networks with parameters θ_g and θ_d respectively. The mapping function between from some representation space, called, the latent space to the space of data, represented by $G(z;\theta_g)$. The $D(x;\theta_d)$ is a mapping function that maps data x to the probability that it came from the real data distribution rather than generator distribution p_g. Basically, $D(x)$ represents the likelihood of x came from the data rather thatm p_g. $D(x;\theta_d)$ outputs a single scalar value between $[0,1]$. D is trained to maximize the probability of correctly labeling both training examples and data generated by the G. Usually, when the generator is training, the discriminator does not train and vice versa. This means for a fixed generator G, the discriminator is trained to classify data as either being from the real data distribution (real, probability close to one) or from the fixed generator(fake, probability close to zero). After certain iterators of the training, when the discriminator is optimal, it can be frozen. Following, the generator G will be continued to be trained to lower the accuracy of the discriminator. After training, if the generator can match the real data distribution, then the discriminator maximally confused, will predict 0.5 probability for its inputs. In practice, the discriminator is may not be trained till it is optimal [56]. Furthermore, simultaneously G is being trained to minimize $log(1 - D(G(z))$. Simply

put, D and G play the two-player minimax game with the objective function $\mathcal{L}_{GAN}(G, D)$:

$$\min_G \max_D \mathcal{L}_{GAN}(D, G) = E_{x \sim p_{data}(x)}[logD(x)] + E_{z \sim p_z(z)}[log(1 - D(G(z)))]. \tag{3.1}$$

Authors mentioned, in practice, the equation 3.1 may not give enough gradient for G to learn well. During Early in learning phase, when G is poor, D can reject samples with high confidence because they are clearly distinct from the training data. In this case, term $log(1 - D(G(z)))$ in equation 3.1 saturates very quickly. Hence, we can train G to maximize $logD(G(z))$ rather than training G to minimize $log(1 - D(G(z)))$. This objective function leadsto the same fixed point of the dynamics of G and D by providing much stronger gradients early in learning [20].

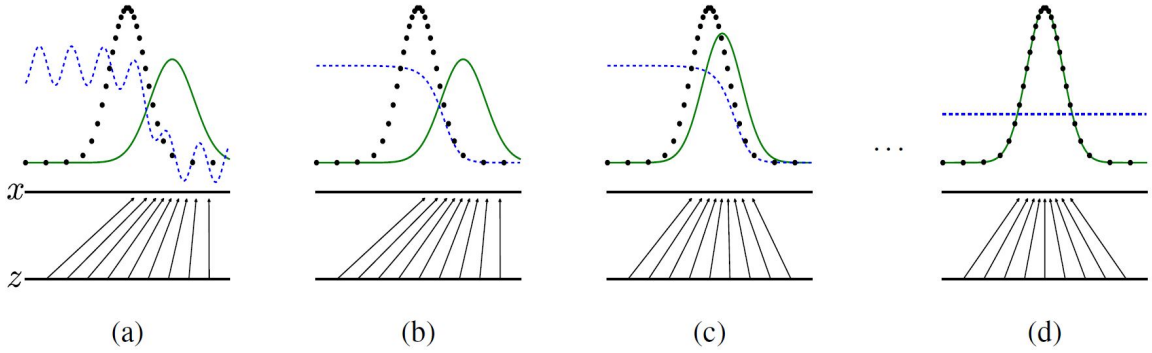


Figure 3.3: Illustration of GANs converging to match generated data distribution p_g to real data distribution p_{data} [20].

In figure 3.3, GANs converging to match generated data distribution p_g to real data distribution p_{data}. While training GANs, discriminative distribution (D, blue, dashed line) is simultaneously updated, so it will discriminate samples from the real data distribution (black, dotted line) p_x from those of the generated data distribution p_g (G, green, solid line). The lower horizontal line represents the domain noise distribution p_z from which z is sampled uniformly. The horizontal line above represents the part of the domain of real data x. The upward arrows depict the mapping of $x = G(z)$, which creates the non-uniform generated data distribution p_g on transformed samples. (a) Consider, generator G and discriminator D are on the verge of convergence, p_g is almost similar to p_{data} and D is a partially accurate classifier. (b) Slowly, discriminator D trained further to distinguish real data samples from generated data samples. For a fixed generator, there is an optimal discriminator, $D^*(x) = \frac{p_{data}(x)}{p_{data}(x)+p_g(x)}$. (c) The generator G is updated using gradients that are backpropagated from discriminator D and makes $G(z)$ to flow to regions that are more likely to be classified as real data. The parameters of the generator are updated, while the parameters of the discriminator are fixed and vice versa. (d) After several steps of training, the generator, G, is optimal where $p_g = p_{data}$, which is equivalent to the optimal discriminator D predicting 0.5 for all samples drawn from x, at this stage discriminator D is maximally confused and will not able to distinguish real data samples from generated data samples, i.e. $D(x) = \frac{1}{2}$ [20].

3.1.1 GAN Training

This section presents the algorithm of the GAN and in the figure A.2, we can see the pictorial representation of the algorithm. The GAN training algorithm is divided into two sections. These two sections are discriminator training and generator training. In figure A.2 shows the same GAN network in different stages of the training process at distinct time points. For example, while training generator, the discriminator is not training and vice versa. Hence, the training dataset of real data samples is grayed out or disabled in the section when the generator is getting trained. While training discriminator both generated samples from the generator and real samples are used as an input. In the following sections describe generator's and discriminator's architecture and their training process. Further, the GAN training algorithm has been described in the form of pseudo code.

The Discriminator Architecture

The discriminator in a GAN is a binary classifier. It attempts to classify real data samples from fake data samples generated by the generator. It outputs the probability of input being a real data sample. The discriminator can be implemented using a multilayer neural network which is suitable to the kind of data it's classifying. If the discriminator is classifying images the CNN could be a good choice [34]. The discriminator's training data come from the two resources, training dataset or collection of real data samples and fake data samples generated by the generator. During GAN training, the discriminator is trained by both real data samples and fake data samples generated by the generator. The discriminator is penalized for misclassifying a real data sample as fake or a fake data sample as real. Such classification error is called discriminator loss. The discriminator loss is backpropagated to update the weights of the discriminator network. The training process of the discriminator D using backpropagation illustrated in figure 3.4.

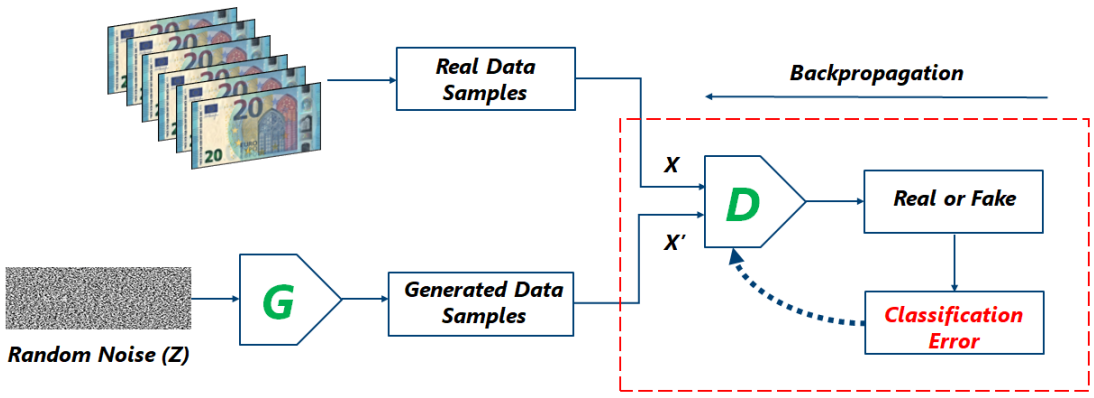


Figure 3.4: Illustration of the training of the discriminator D using backpropagation.

The Generator Architecture

The generator produces fake samples using random noise vectors sampled from the uniform random noise distribution. As described earlier, the discriminator's weights are frozen while training the generator. This means the discriminator is not training while the generator is generating fake samples and vice versa. The generator can be implemented using a multilayer neural network which is suitable for the kind of data it's generating. If the generator is generating images using the random noise distribution, the CNN could be a good choice [34]. The generator aims to produce fake samples

that are as realistic as possible. But when the generator for fails to fool the discriminator or when the discriminator classifies the generated sample as fake, then it is penalized with classification error. Such classification error is called generator loss. The generator loss is backpropagated to update the weights of the generator. The training process of the generator G using backpropagation illustrated in figure 3.5.

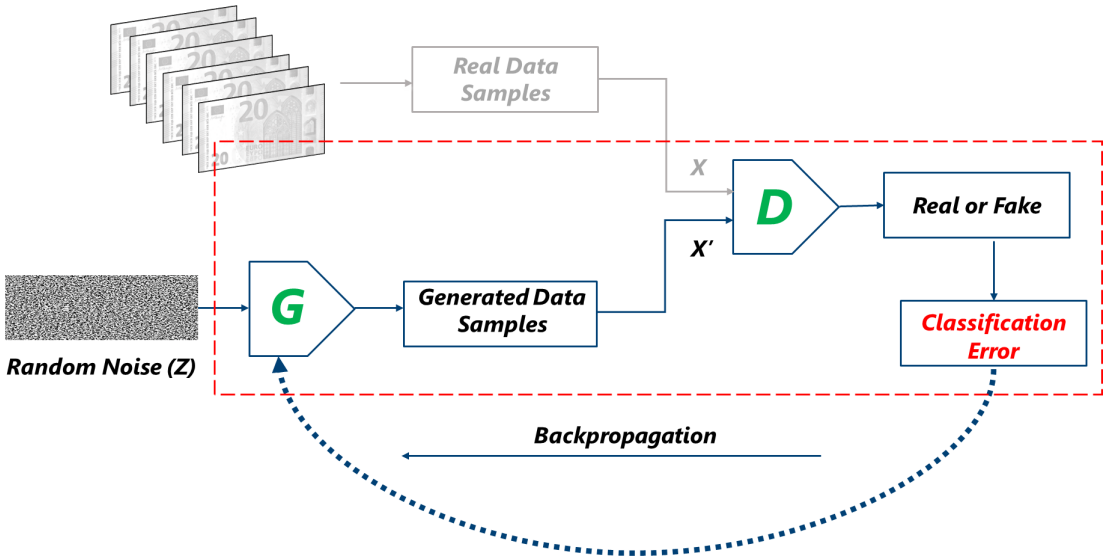


Figure 3.5: Illustration of the training of the generator G using backpropagation.

GAN Training Algorithm

for *each training iteration* **do**

1. Train the Discriminator:

a) Get a random real data sample X from the training dataset.

b) Get a random noise vector Z from the random noise distribution, pass it thorough the Generator and create a fake sample X'.

c) Classify X and X' using the Discriminator.

d) Backpropagate the calculated classification error to update the Discriminator's weights to *minimize* classification error.

2. Train the Generator:

a) Get a random noise vector Z from the random noise distribution, pass it thorough the Generator and create a fake sample X'.

b) Classify X' using the Discriminator.

c) Backpropagate the calculated classification error to update the Generator's weights to *maximize* the Discriminator's error.

end

Algorithm 1: GAN Training Algorithm [1].

3.2 Convolution Neural Networks

In recent years in the field of machine learning, drastic improvements have occurred to increase the performance of machine learning models. Also, numerous deep learning techniques like ANNs are evolved significantly. These biologically inspired neural networks were able to exceed the performance compared to the traditional machine learning techniques to solve complex problems [57]. The most successful image-driven pattern recognition technique among ANNs is CNNs [57]. Nowadays, CNNs are used to solve difficult image recognition, image classification, and object detection tasks. The CNNs have been a popular method because of its extraordinary results at object recognition competition known as the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012 for solving computer vision tasks. CNN is a kind of deep learning technique for processing data that has a grid pattern, for example, images. CNNs are inspired by the early findings in the study of biological vision, especially, organization of the animal visual cortex [[58], [59], [8]] and designed to automatically and adaptively learn spatial hierarchies of features, from low-level to high-level patterns [8]. It is composed of various building blocks, such as convolution layers, pooling layers, and fully connected layers. As shown in figure 3.6, the first two layers are convolution and pooling layers. They perform feature extraction. whereas the third, a fully connected layer, maps the extracted features into the final output, such as classification [60]. Let us have a look at each layer in the following sections.

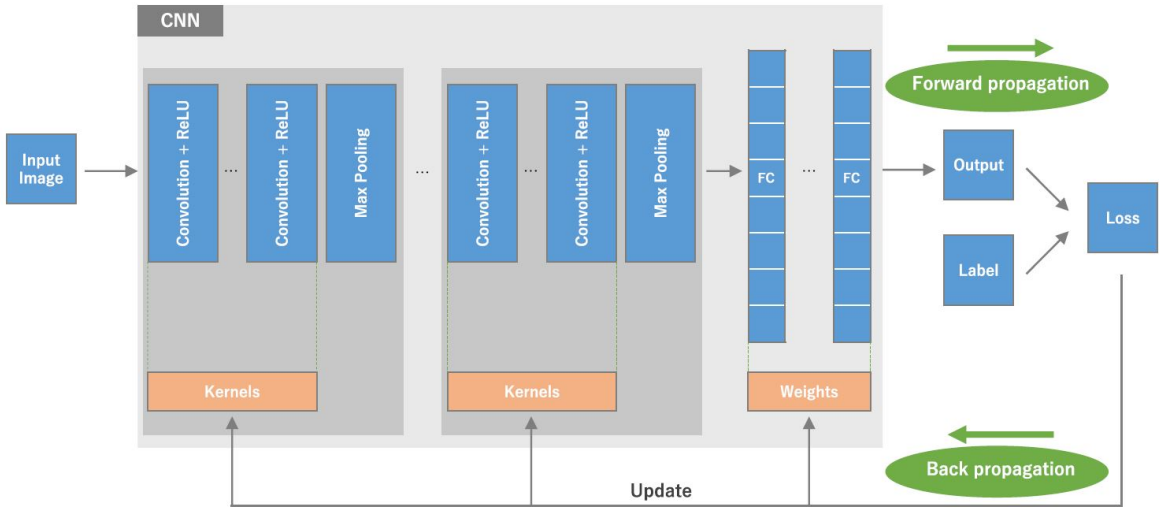


Figure 3.6: Overview of CNN architecture and its training process. CNNs is a combination of several building blocks like convolution layers, pooling layers, and fully connected layers. These building blocks are stacked upon each other. The CNNs is trained using the training dataset, the input data is fed in the forward direction through the network. The process of feeding data in the forward direction to the CNN is called forward propagation. Each layer accepts the input data, processes it as per the activation function like ReLU (Rectified Linear Unit) (figure 3.11), and passes it to the successive layer. Using a loss function through forward propagation on a training dataset, a model's performance under particular kernels and weights is calculated. The learnable parameters, for example, weights and kernels, are updated as per the loss value through backpropagation using a gradient descent optimization algorithm [61].

3.2.1 Convolution Layer

The convolution layer is a basic component of the CNN architecture that performs feature extraction. It is typically a combination of linear and nonlinear operations like convolution operation and activation function. It is one of the core building blocks of CNNs. Also, it is responsible for most of the heavy computations. A convolution layer plays an important role in CNN, it performs mathe-

matical operations like convolution, a specialized type of linear operation. The digital images store pixel values in a Two-dimensional (2D) grid, i.e., an array of numbers as shown in figure 3.8. The small grid of parameters called the kernel, an optimizable feature extractor, is applied at each image position [60], which makes CNNs highly efficient for image processing, feature extraction, since a feature could occur at any location in the image. The layers are connected, one layer feeds its output to the next layer. The extracted features can hierarchically and progressively become more complex [60]. Training is the process of optimizing parameters such as kernels which minimize the difference between outputs and ground truth labels through an optimization algorithm called backpropagation [62] and gradient descent [61], among others.

For feature extraction linear operation, convolution is used, where a small array of numbers, called a kernel, is applied across the input, which is an array of numbers, called a tensor. An element-wise product between kernel's each element and the input tensor is calculated at each location of the tensor and summed to obtain the output value in the corresponding position of the output tensor, called a feature map (figure 3.8a-c) [60]. This procedure is repeated by applying multiple kernels to create an arbitrary number of feature maps, which describe different characteristics of the input tensors [60]. The different kernels can be considered as different feature extractors. Two important hyperparameters that represent the convolution operation are the size and number of kernels. The size is typically 3×3, but sometimes 5×5 or 7×7, depends on the requirement. The number of kernels is arbitrary, and determines the depth of output feature maps. The above-mentioned convolution operation prevents the center of each kernel from overlapping the input tensor's outermost element. And the output feature map's height and width are reduced compared to the input tensor. To address this issue, the padding, predominantly zero-padding technique is used where rows and columns of zeros are added on each side of the input tensor, to fit the center of a kernel on the outermost element and keep the same spatial dimension through the convolution operation (figure 3.7). Modern CNN architectures normally employ zero padding to retain spatial dimensions to apply more further layers. Each successive feature map would get smaller after the successive convolution operation without zero padding. A stride is the distance between two successive kernel positions, and it also defines the convolution operation. A stride of 1 is the most common choice; however, a stride greater than 1 is sometimes used to achieve feature map downsampling. A pooling operation, as defined in Section 3.2.2, is an alternative technique for downsampling.

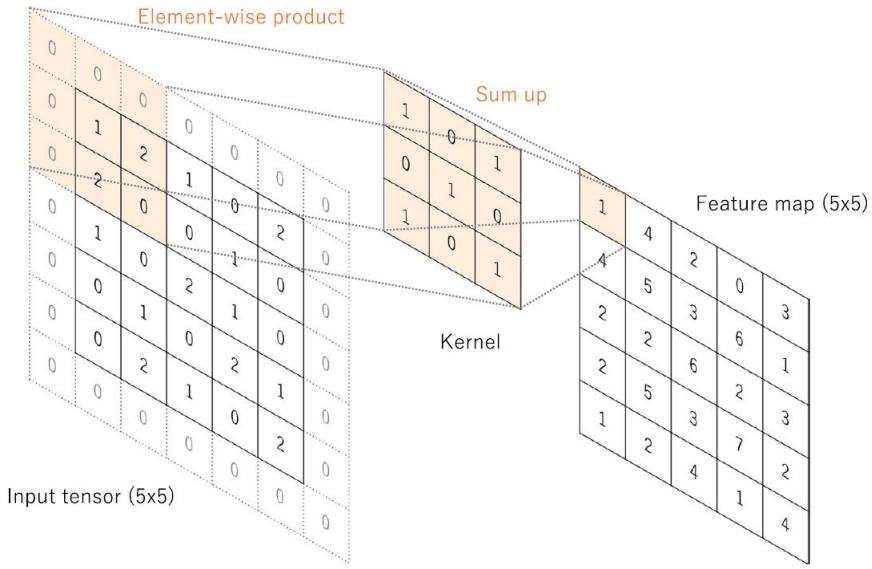


Figure 3.7: Illustration of a convolution operation with zero padding to retain spatial dimensions. Note that an input dimension of 5×5 is retained in the generated output feature map. In this example, kernel size is set to 3×3, and stride is set to 1 [60].

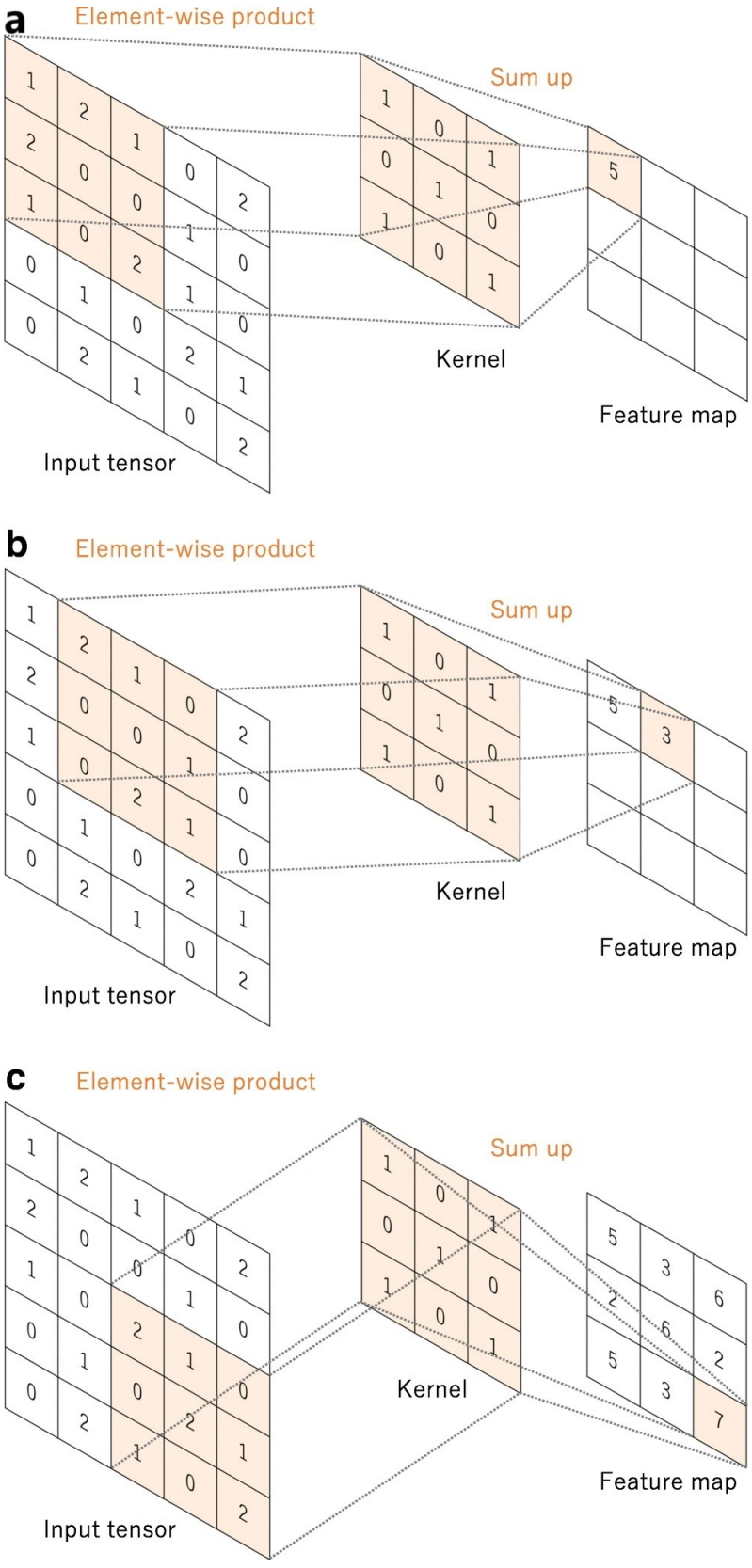


Figure 3.8: **a**–**c** An illustration of convolution operation with no padding. In this example, kernel size is set to 3×3, and stride is set to 1. A kernel is applied across the input tensor, and an element-wise product between each element of the kernel and the input tensor is calculated at each location and summed to obtain the output value in the corresponding position of the output tensor, called a feature map [60].

Activation Functions

ANNs are inspired by biological neural networks present in the animal brain. Before constructing any ANN, it is important to understand the artificial neuron model. In the figure 3.10 schematic diagram of an artificial neuron is illustrated. A biological neuron gets excited when other neurons with different weights send electrical signals to it. The value of the electrical signal should be big enough to excite the neuron. Otherwise, it will be in an inactive state. In the figure 3.3 schematic diagram of an artificial neuron is illustrated. Where $\{X_1, X_2, X_3, ..., X_n\}$ are the inputs to the artificial neuron, $\{W_1, W_2, W_3, ..., W_n\}$ are the weights corresponding to the inputs. b is the bias. The summation symbol represents the addition unit. The addition unit gets the linear weight sum Z of the inputs and bias. Dot product between inputs and weights performed before addition. If $X = [X_1, X_2, X_3, ..., X_n] \in R^n$ and $W = [W_1, W_2, W_3, ..., W_n] \in R^n$, then output Z is represented as

$$Z = XW^\top + b. \tag{3.2}$$

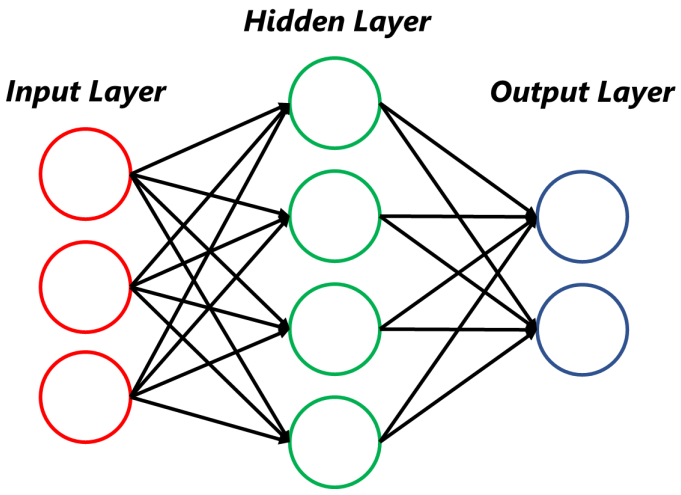


Figure 3.9: Simple Neural Network.

The function f is activation function. It is used to excite the response state of biological neurons and obtain output Y

$$Y = f(Z). \tag{3.3}$$

The data is fed to the input layer of the neural network, the input undergoes the linear operation. Later, activation functions are applied to it in the hidden layers, and output is produced. In neural networks the hidden layer lies between input layer and output layer. In figure 3.9 simple neural network is illustrated. The activation function is applied to the outputs of a linear operation like convolution. The activation functions important while constructing any neural network. They are mathematical functions attached to the neurons. Also, they tell which of the neuron is excited or triggered in each layer of neural networks. The purpose of the activation function is to introduce non-linearity in the neural network.

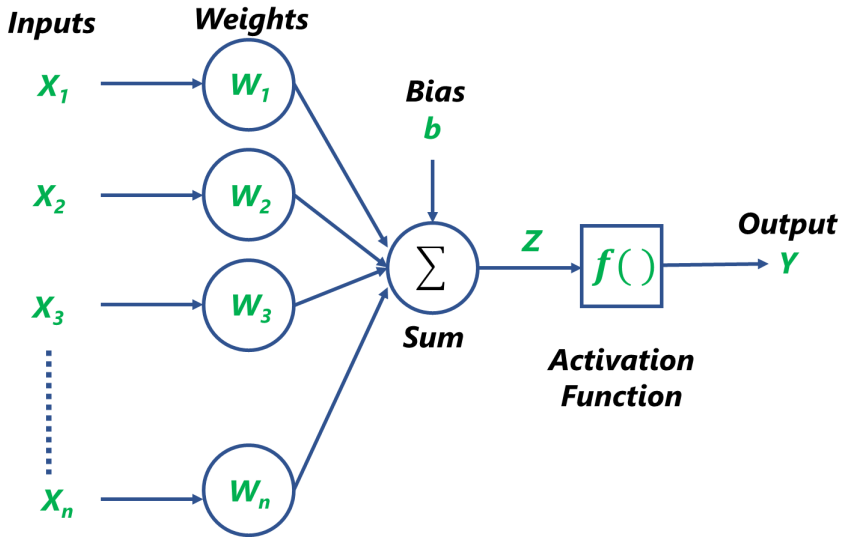


Figure 3.10: Illustration of Artificial Neuron.

The equation 3.2 performs a linear operation. This linear function gets repeated every hidden layer of the neural network. A combination of such linear functions also a linear function, equivalent to all the hidden layers collapsed into a single linear function performing linear regression. Hence, all the hidden layers become useless. Even if linear functions are easy to use, but they fail to learn complex patterns present in the data like images, speech, and videos. Hence, nonlinear activation functions are used, while constructing neural networks. The nonlinear activation functions are differentiable, and they make gradient descent optimization (backpropagation) possible. Backpropagation minimizes the error and enhances the accuracy and performance of the neural network. The nonlinear functions like the sigmoid and hyperbolic tangent (tanh) functions were used previously because they are mathematical representations of biological neuron behavior. The rectified linear unit is now the most commonly used nonlinear activation function (ReLU) , which simply computes the function: $f(x) = \max(0, x)$ (figure 3.11) [[63], [64], [[65]], [[66]], [67]].

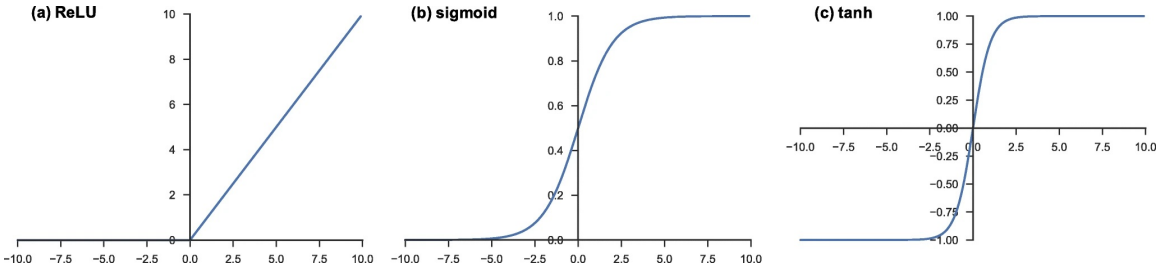


Figure 3.11: Most common nonlinear activation functions used while constructing Neural Networks: **a)** rectified linear unit (ReLU), **b)** sigmoid, and **c)** hyperbolic tangent (tanh) [60].

3.2.2 Pooling Layer

A pooling layer performs downsampling on feature maps to gradually reducing the spatial dimensionality, which leads to the reduction of the number of parameters and computational complexity of the neural network and controlling overfitting. Downsampling introduces translation invariance to minor shifts and distortions [68]. Downsampling of feature maps can be accomplished by using convolution layers by increasing the stride of the convolution operation across the image. But the robust and common approach of downsampling is to use a pooling layer [68]. Similar to convolution

operations, it used hyperparameters like filter size, stride, and padding. Also, it is common to periodically insert a pooling layer in between successive convolution layers while constructing neural networks. There two common types of pooling methods one is max pooling the other is average pooling. The max-pooling considers the most activated (maximum value) value in each input patch of the feature map. The average pooling averages values in each input patch of the feature map.

Max Pooling

Max pooling is the most common and popular type of pooling operation. The max-pooling operation is independently operated at every depth slice of the input and resized spatially. Commonly, in max-pooling filters of size 2×2 applied with a stride of 2. The max-pooling operation extracts small patches of given filter size from the input feature maps and outputs the maximum value in every patch discarding others (figure 3.12) [68]. The spatial dimension of feature maps is reduced by a factor of two by discarding 75% of its activations. The height and width dimension of the features maps is reduced but the depth dimension remains unchanged. Pooling operations with larger filter sizes are too destructive. It's worth noting that max-pooling layers do not have learnable parameters.

Global Average Pooling

There is another type of pooling operation called global average pooling. It performs an extreme type of downsampling. It downsamples a feature map of size height × width × depth into a $1 \times 1 \times$ depth array by averaging all the elements present in the feature map by keeping depth dimension unchanged. This operation is applied only once before fully connected layers. There are some benefits of using global average pooling: a) It reduces the number of learnable parameters. b) allows the CNN to consider inputs of variable size [60] [69].

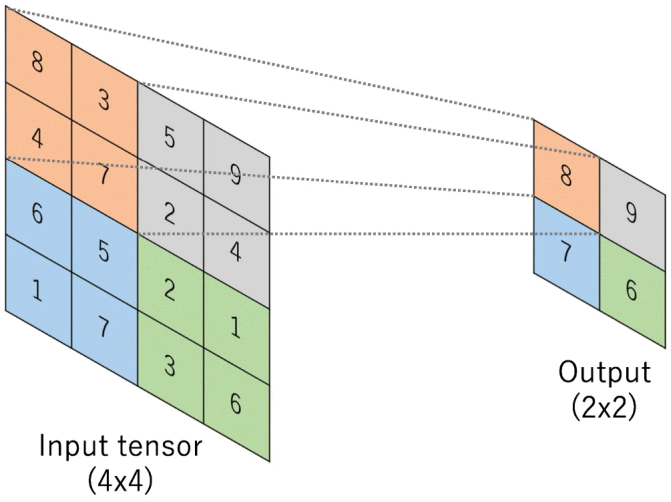


Figure 3.12: Illustration of max pooling operation with a filter size of 2×2. No padding, and a stride of 2. Max pooling operation extracts 2×2 patches from the input tensors, outputs the maximum value in each patch, and discards the rest of the other values, which results in the spatial dimension of an input tensor downsampled by a factor of 2 [60].

3.2.3 Fully connected layer

In neural networks, fully connected layers are those layers where all the inputs from one layer are connected to every neuron of the next layer. The final output feature map of convolution layers or pooling layers is flattened and the output is transformed into a One-dimensional (1D) vector or array of numbers. The flattened output is connected to one or more fully connected layers. The

fully connected layers are also called dense layers, where every input is connected to every output by a learnable weight. A fully connected layer performs multiplication between the input and weight matrix and then adds a bias vector. Each fully connected layer is followed by a nonlinear activation function, for example, ReLU. As already described, the nonlinear activation function helps neurons learn complex patterns. The equation 3.3 represents the function of a fully connected layer. The features extracted using convolution layers and downsampled by pooling layers are mapped by a subset of fully connected layers to the final output of the neural network, for example, probabilities for each class in the classification tasks. Most of the time number of output nodes in the final fully connected layer equals the number of classes. The final fully connected layer's activation function is usually different than others. Each task chooses a suitable activation function. The sigmoid activation function is used in binary classification tasks. For example, logistic regression models the probabilities for binary classification tasks using the sigmoid activation function. The softmax activation function is widely used in multiclass classification tasks. It normalizes output real values from the last fully connected layer to target class probabilities, ranges between 0 and 1, and all values sum to 1 [60].

4. Methodology

Next, the CycleGAN, is presented with its loss functions, in Section 4.2.

4.1 Proposed Approach

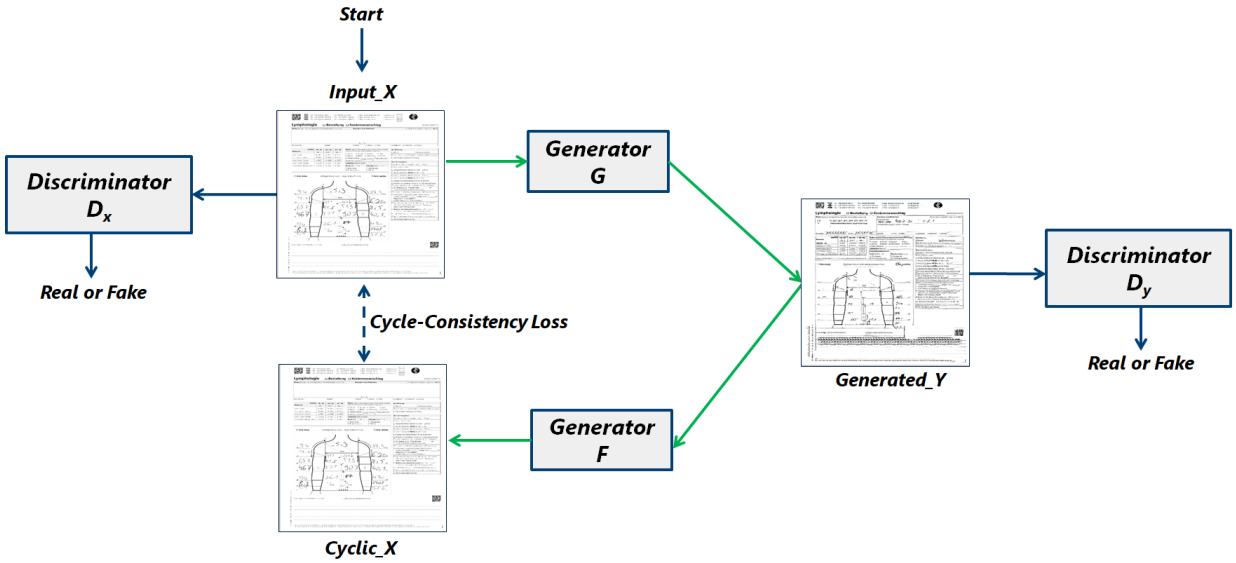


Figure 4.1:

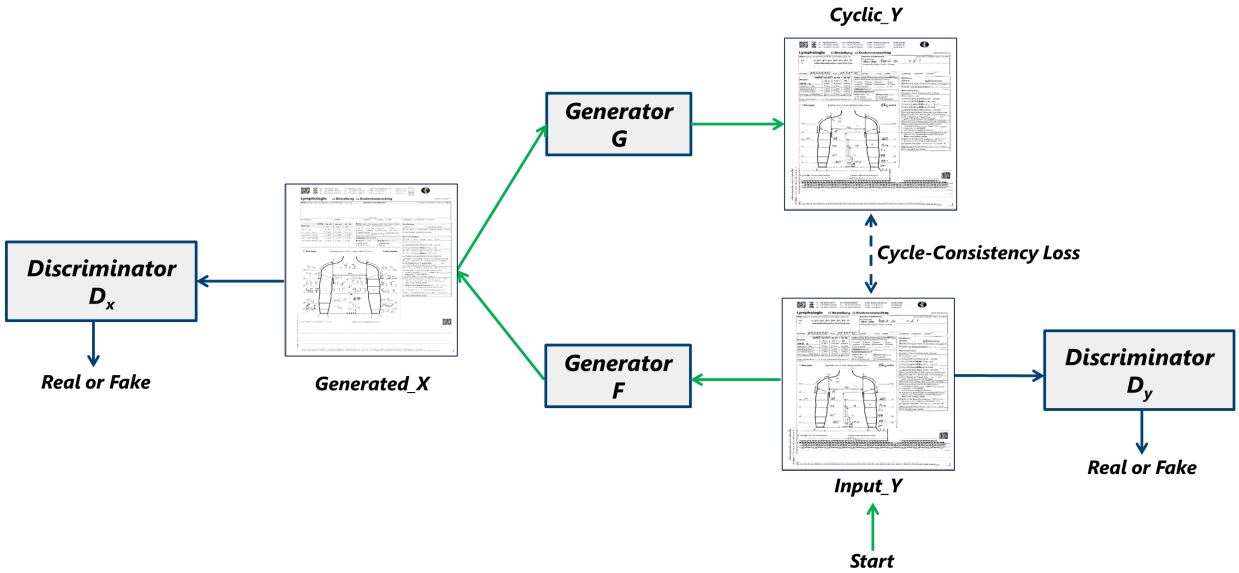


Figure 4.2:

4.2 Cycle-Consistent Adversarial Networks

4.2.1 Formulation

The aim is to learn mapping functions between two domains X and Y. The Domain X represents synthetic document images distribution and domain Y represents real document images distribution. Given training samples $\{x_i\}_{i=1}^{N}$ where $x_i \in X$ and $\{y_j\}_{j=1}^{M}$ where $y_j \in Y$. We denote the data distribution $x \sim p_{data}(x)$ and $y \sim p_{data}(y)$. As illustrated in Figure 4.3, the model includes two mappings $G : X \rightarrow Y$ and $F : Y \rightarrow X$. In addition, we introduce two adversarial discriminators D_X and D_Y , where D_X aims to distinguish between images $\{x\}$ and translated images $\{F(y)\}$; in the same way, D_Y aims to discriminate between $\{y\}$ and $\{G(x)\}$. Our objective contains three types of terms: least-square losses (LSGAN) [31] for matching the distribution of generated images to the data distribution in the target domain; cycle consistency losses to prevent the learned mappings G and F from contradicting each other; and identity mapping loss to preserve the color of the input images.

4.2.2 Least-Square Loss

Viewing the discriminator as a classifier, regular GANs adopt the sigmoid cross-entropy loss function. As stated in Section 2.2, when updating the generator, this loss function will cause the problem of vanishing gradients for the samples that are on the correct side of the decision boundary, but are still far from the real data. Also to stabilize the model training procedure. First, $\mathcal{L}_{GAN}$ (equation 3.1), the negative log-likelihood objective replaced by a least-squares loss (LSGAN) [31]. This loss is more stable during training and generates higher quality results. The least-square loss is used to optimize the generator and discriminator adversarially. We use the a-b coding scheme for the discriminator, where a and b are the labels for fake data and real data, respectively. Then the modified objective functions using least-squares loss can be defined as follows:

$$\min_D \mathcal{L}_{LSGAN}(D_Y) = \frac{1}{2}\,\mathbb{E}_{y\sim p_{data}(y)}\,[\|(D(y)-b)\|_2] + \frac{1}{2}\,\mathbb{E}_{x\sim p_{data}(x)}\,[\|(D(G(x))-a)\|_2] \tag{4.1}$$

$$\min_G \mathcal{L}_{LSGAN}(G) = \mathbb{E}_{x\sim p_{data}(x)}\,[\|(D(G(x))-c)\|_2] \tag{4.2}$$

$$\mathcal{L}_{LSGAN}(G, D_Y, X, Y) = \min_D \mathcal{L}_{LSGAN}(D_Y) + \min_G \mathcal{L}_{LSGAN}(G) \tag{4.3}$$

$$\min_D \mathcal{L}_{LSGAN}(D_X) = \frac{1}{2}\,\mathbb{E}_{x\sim p_{data}(x)}\,[\|(D(x)-b)\|_2] + \frac{1}{2}\,\mathbb{E}_{y\sim p_{data}(y)}\,[\|(D(F(y))-a)\|_2] \tag{4.4}$$

$$\min_G \mathcal{L}_{LSGAN}(F) = \mathbb{E}_{y\sim p_{data}(y)}\,[\|(D(F(y))-c)\|_2] \tag{4.5}$$

$$\mathcal{L}_{LSGAN}(F, D_X, Y, X) = \min_D \mathcal{L}_{LSGAN}(D_X) + \min_G \mathcal{L}_{LSGAN}(F) \tag{4.6}$$

where c denotes the value that G wants D to believe for fake data. Basically, $a = 0$, $b = 1$, and $c = 1$.

4.2.3 Cycle Consistency Loss

Adversarial training can, in theory, learn mappings G and F that produce outputs identically distributed as target domains Y and X respectively. However, with large enough capacity, a network can map the same set of input images to any random permutation of images in the target domain, where any of the learned mappings can induce an output distribution that matches the target distribution. Thus, adversarial losses alone cannot guarantee that the learned function can map an individual input x_i to a desired output y_i. To further reduce the space of possible mapping functions, we argue that the learned mapping functions should be cycle-consistent: as shown in Figure 4.3 (b), for each image x from domain X, the image translation cycle should be able to bring x back to the original image, i.e., $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$. We call this forward cycle consistency. Similarly, as illustrated in Figure 4.3 (c), for each image y from domain Y, G and F should also satisfy backward cycle consistency: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$. We incentivize this behavior using a cycle consistency loss:

$$\mathcal{L}_{cyc}(G, F) = \mathbb{E}_{x \sim p_{data}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{data}(y)}[\|G(F(y)) - y\|_1]. \tag{4.7}$$

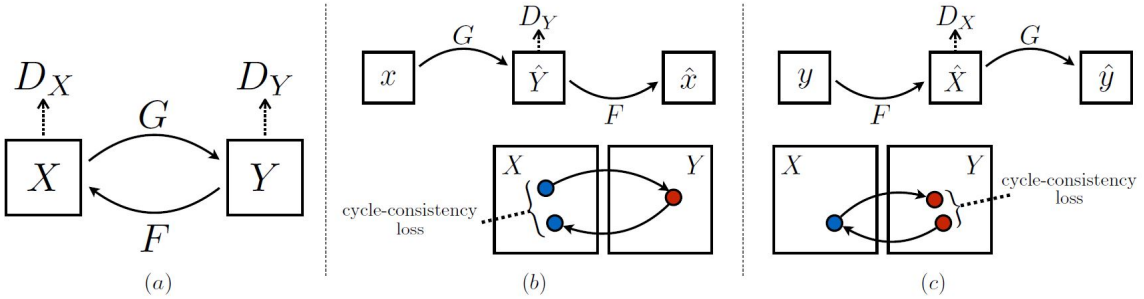


Figure 4.3: (a) CycleGAN model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs indistinguishable from domain Y, and vice versa for D_X and F. To further regularize the mappings, we introduce two cycle consistency losses that capture the intuition that if we translate from one domain to the other and back again we should arrive at where we started: (b) forward cycle-consistency loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$ [19].

4.2.4 Identity Mapping Loss

It is helpful to introduce an additional loss to encourage the mapping to preserve color composition between the input and output. In particular, we adopt the technique of [70] and regularize the generator to be near an identity mapping when real samples of the target domain are provided as the input to the generator:

$$\mathcal{L}_{identity}(G, F) = \mathbb{E}_{y \sim p_{data}(y)}[\|(F(y) - y\|_1] + \mathbb{E}_{x \sim p_{data}(x)}[\|G(x) - x\|_1]. \tag{4.8}$$

4.2.5 Full Objective

The full objective is:

$$\begin{aligned}\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{LSGAN}(G, D_Y, X, Y) \; + \; \mathcal{L}_{LSGAN}(F, D_X, Y, X) \; + \\ \lambda_{identity} \; \mathcal{L}_{identity}(G, F) \; + \; \lambda_{cyc} \; \mathcal{L}_{cyc}(G, F),\end{aligned} \tag{4.9}$$

where λ_{cyc} and $\lambda_{identity}$ control the relative importance of the two objectives.

5. Implementation

This chapter discusses the implementation of the CycleGAN and Classifiers. In this thesis, classifiers are used to determine the domain gap between the distributions. Also, they are used to evaluate the quality of images generated by the CycleGAN. How the classifiers are being used to analyze the domain gap between distributions will be discussed in-depth in chapter Evaluation 6. In section 5.2 dataset preparation is described. The architecture of CycleGAN and Classifiers discussed in section 5.3. The training details of CycleGAN and Classifiers described in

5.1 Training Details

. The experiments are visualized using Tensorboard[1]. TensorBoard is a tool that provides the visualization needed for machine learning research and experiments. The neural networks implemented in this thesis using Python, Keras APIs, and TensorFlow library[71]. The reference code for the CycleGAN is available here. All of the neural networks are trained upon GPUs (Graphics Processing Units) like Nvidia Tesla T4 and Tesla V100-SXM2.

5.2 Dataset Preparation

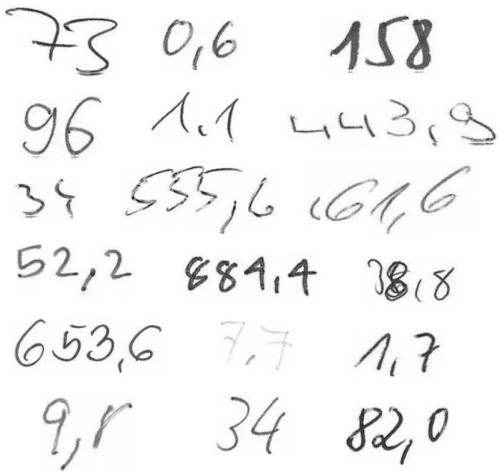


Figure 5.1: Examples of Handwritting Crops from Handwritting Number Dataset. (Figure reproduced from elevait GmbH & Co. KG with permission.)

The dataset preparation is one of the vital aspects of training any neural network. Bad quality data leads to a poor generalization of neural networks. There are ten types of documents that were considered to work with this image-to-image translation application. Hence, the CycleGAN is trained using a stash of synthetic document images and real document images. Around 100,000 synthetic document images in the source domain and the same number of real document images in the target domain. The synthetic document images are generated using unfilled form image (figure A.3) and handwritten crops (figure 5.1). The process of inserting handwritten crops on empty templates can be visualized in figure 5.2. Each unfilled form image is filled with the help of provided

[1] `https://www.tensorflow.org/tensorboard` last access: 04.05.2021

bounding box annotations [72]. For each class of unfilled form image 10,000, synthetic document images are created. As mentioned earlier, 100,000 synthetic document images are created in total. The same created 100,000 synthetic document images are used while training CycleGAN. Just they are stashed at the same location collectively. The created 100,000 synthetic document images were also faxified and a faxified dataset of 100,000 images is created. It has the same structure as synthetic document images, 10 classes, and each class has 10,000 images. The faxification process uses several image transformations to make a clean gray-scale image look like it was sent via fax. A sample faxified image can be seen in figure A.5. The faxification process is described briefly in Section 6.2.5. Also, the faxification can be visualized in figure 6.3. In the table 5.1 the number of samples in each dataset is mentioned. For testing, around 1162 annotated real document images are used. This testing dataset is used to evaluate the performance of the classifiers trained upon different data distributions like synthetic document images, faxified document images, and CycleGAN generated document images. Basically, testing dataset is significant to understand the domain gap between real data distribution and remaining data distributions. In the table 5.2 the number of samples in each class in the test dataset is mentioned. The testing dataset is unbalanced. The datasets used in this thesis can not be cited or published because they are not open for public use.

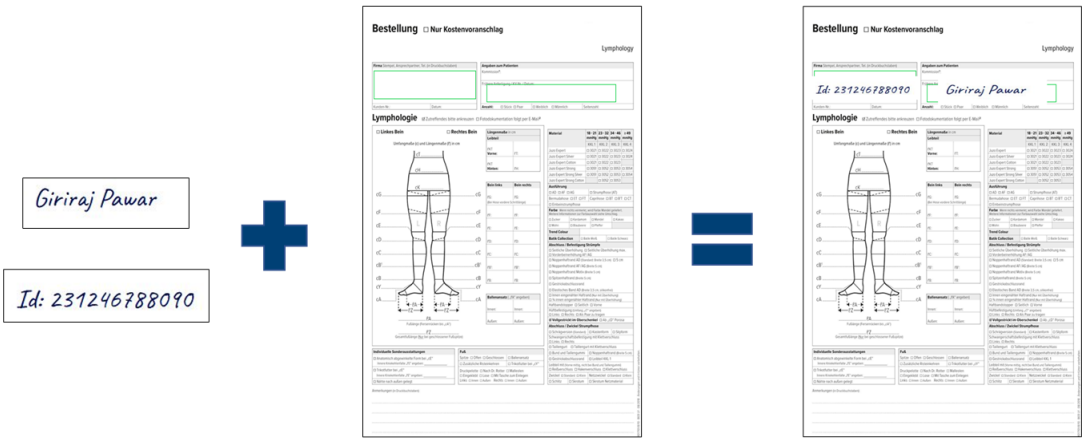


Figure 5.2: Inserting Handwritten Crops in Empty Form Templates.

Datasets	Size (Number of Images)
Synthetic Document Images	100,000
Real Document Images	100,000
Faxified Document Images	100,000
Annotated Real Document Images (Used for testing)	1162

Table 5.1: Size of Datasets used for training CycleGAN and Classifiers.

Classes	Size (Number of Images)
DE_LY_Arm_2020-01	44
DE_LY_Bein_2018-08	47
DE_LY_Bein_2019-01	50
DE_LY_Bein_2019-07	60
DE_LY_Bein_2020-01	624
DE_LY_Bein_2020-03	128
DE_LY_Hand_2020-01	16
DE_PH_Bein_2018-09	22
DE_PH_Bein_2019-02	28
DE_PH_Bein_2020-01	143

Table 5.2: Number of Images in each Class of Annotated Real Document Images Dataset.

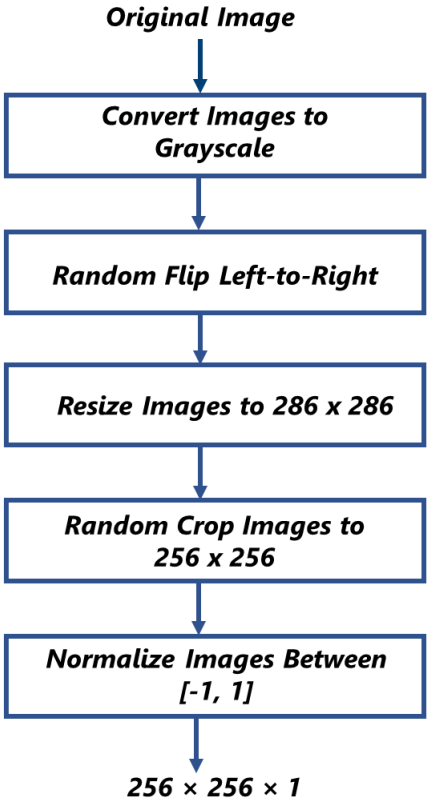


Figure 5.3: Steps involved in preprocessing of training images of CycleGAN.

5.3 Network Architecture

5.3.1 CycleGAN

Johnson et al.[17] proposed the architecture of CycleGAN. In which the generator has three sequences of blocks one is downsampling, transformation, and upsampling. The sequence of 2 downsampling convolutional blocks encode the $256 \times 256 \times 1$ grayscale input image, 9 Residual Network (ResNet) convolutional blocks to transform the image, and a 2 upsampling convolutional blocks to generate the output image of the same dimension as the input image. The reason behind using residual blocks is it resolves the vanishing gradient problem in deep neural networks. The discriminator classifier network is designed using PatchGAN architecture [37] [42]. The PatchGAN discriminator is simply a CNN. The major difference between the PatchGAN discriminator and General GAN discriminator is, GAN discriminator maps input image to the scalar output, which represents image being real to fake. But, the PatchGAN discriminator maps the input image to $N \times N$ array of outputs, where each element in an output array represent a patch in an input image being real or fake. Basically, the PatchGAN discriminator penalizes structure at the scale of local image patches and attempts to classify if each $M \times M$ patch in an image is real or fake.

Johnson et al. [17] have provided naming conventions to define the architecture of generator and discriminator used in CycleGAN. `c7s1-k` denotes a 7×7 Convolution-InstanceNormlization-ReLU layer with k filters and stride 1. The downsampling block `dk` denoted by a 3×3 Convolution-InstanceNormlization-ReLU layer with k filters and stride 2. To reduce artifacts reflection padding is used. `Rk` denotes a single residual block that has two 3×3 convolutional layers with the same number of filters k on both layers and stride 1. The upsampling block `uk` denoted a 3×3 TransposedConvolution-InstanceNormlization-ReLU layer with k filters and stride 2. The complete generator network with 9 residual blocks can be decribed as: `c7s1-64, d128, d256, R256, R256, R256, R256, R256, R256, R256, R256, u128, u64, c7s1-1`. The last

layer `c7s1-1` denotes a 7×7 Convolution layer with 1 filter and stride 1. Next, the final output is followed by a tanh activation function (figure 3.11). All of the layers in the generator can be seen in figure A.7.

Operation Layer	Number of Filters	Size of Each Filter	F1-score	Support

Table 5.3:

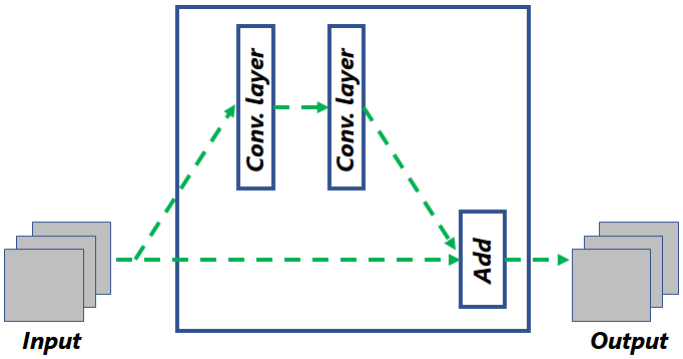


Figure 5.4: Illustration of ResNet Blocks in CycleGAN Generator Architecture.

The discriminator uses 70×70 PatchGAN classfier architecture [37]. It is also called a Markovian discriminator [42]. `Ck` denotes a 4×4 Convolution-InstanceNormalization-LeakyReLU layer with k filters and stride 2. Leaky ReLUs with a slope of 0.2 are used. Instance Normalization is not used for the first `C64` layer. After the last layer `C512`, the convolution operation is applied with filter 1 to produce an output of depth 1 using 4×4 kernel and stride 1. The discriminator network can be described as: `C64-C128-C256-C512`. All of the layers in the discriminator can be seen in figure A.13. For more information on the architectures of generators and discriminators, this Github repository can be referred to.

Operation Layer	Number of Filters	Size of Each Filter

Table 5.4:

5.3.2 Classifier

The classifier used to determined the domain gap between data distributions is a CNN. The classifier architecture is simplistic (figure **??**). It has just Two Convolution Layers, One Max Pooling Layer, and One Dropout Layer. The architecture of classifier can be preciously visualised in figure A.6.

Operation Layer	Number of Filters	Size of Each Filter

Table 5.5:

5.4 Training Details

5.4.1 CycleGAN

One of the major challenges of the implementation part was designing an efficient input pipeline. The developed image-to-image translation application and classifiers are trained using 100,000 images. Loading such a large dataset is a tedious and time-consuming job but TensorFlow has provided wonderful APIs like tf.data to load large dataset spontaneously. To learn more about how to load large dataset efficiently in TensorFlow refer this Tutorial. Two techniques from recent works are applied to stabilize CycleGAN model training procedure. First, for (equation 1), we replace the negative log likelihood objective by a least-squares loss [

The CycleGAN is trained from scratch, with a learning rate of 0.0002. In practice, the objective is divided by 2 while optimizing discriminator (equations 4.1 and 4.4), which slows down the rate at which discriminator learns, relative to the rate of generator. Weights are initialized from a Gaussian distribution with mean (μ) 0 and standard deviation (σ) 0.02. Images used for the training CycleGAN are converted into grayscale. Also, random mirroring and random jittering is applied. In random mirroring, the image is randomly flipped horizontally i.e. left to right. In random jittering, the image is resized to 286×286 and then randomly cropped to 256×256. As mentioned in the paper [19], random jittering and mirroring applied to the training dataset. These are some of the image augmentation techniques that avoids overfitting. These are some of the image augmentation techniques that avoid overfitting. Lastly, the images are normalized in the range of $[-1, 1]$.

The CycleGAN is trained for 20 epochs and all the individual models like Forward Generator G, Backward Generator F, Discriminator D_Y, and Discriminator D_X are saved as checkpoints at the end of every epoch.

5.4.2 Classifier

The images used to train and test the classifiers were also applied with random mirroring and random jittering after converting the image to grayscale. The size of the images is $256 \times 256 \times 1$. Also, the images are normalized in the range of $[-1, 1]$. The classifier architecture can be visualized in figure **??**. The pre-processing process can be seen in figure **??**. The classifier trained using 80% data and 20% data used for validation. Subsequently, the performance of the classifiers evaluated using real test document images dataset.

6. Evaluation

To evaluate the quality of images generated by the CycleGAN, a classifier is trained on the CycleGAN generated data and its accuracy on a real dtest dataset is used as a metric to measure how well the CycleGAN model distribution matches the real data distribution. Basically, The classification capability of the trained classifier is used as an objective measure to assess the quality of images generated by CycleGAN. Also, the classifier is trained on the synthetic document images and it's accuracy evaluated in

Classification report and metrics

6.1 Evaluation Metrics

6.1.1 Accuracy

6.1.2 f1-score

6.2 Experiments

In this thesis, many experiments were performed to understand the domain gap between data distributions. Three data distributions are considered for the experiments. Each data distribution represents a different domain. The synthetic document images represent the synthetic data distribution. The faxified document images represent faxified data distribution. The CycleGAN generated images represent CycleGAN generated data distribution. The goal of the experiments is to illustrate and analyse the domain gap between the real data distribution and above mentioned three data distributions. Now let us explain steps involved while performing experiments. First, the classifier(figure **??**) is trained on synthetic document images and its performance is evaluated on annotated real document images. Second, the new classifier with the same architecture is trained on faxified document images and its performance is evaluated on annotated real document images. Third, the CycleGAN is trained using synthetic document images and real document images and at the end of every epoch, the checkpoint is saved. Once training is finished all saved checkpoints are loaded. The forward Generator G is loaded and 100,000 synthetic document images are transformed into 100,000 realistic document images. The realistic document images are also referenced interchangeably as CycleGAN generated images. Forward Generator G transforms images from the source domain to the target domain where synthetic document images represent the source domain and real document images represent the target domain. Next, 100,000 CycleGAN generated document images are used to train another new classifier with the same architecture as the previous, and its performance is evaluated on annotated real document images. The experiment aims to understand the quality of the images generated by CycleGAN. Especially, how efficiently CycleGAN was able to close the domain gap between synthetic data distribution and real data distribution. The evaluation metrics like accuracy, weighted average F1 score, and macro average F1 score [73] are used to understand the contrast between the performance of the classifiers on annotated real document images.

6.2.1 Experiment Steps

1. Train a classifier on synthetic document images and evaluate its classification performance on the annotated real document images.
2. Train a classifier on faxified document images and evaluate its classification performance on

the annotated real document images.

3. Train a CycleGAN.
4. Generate document images using trained CycleGAN.
5. Train a classifier on CycleGAN generated document images and evaluate its classification performance on the annotated real document images.
6. Compare the classification performance of the above three classifiers on the annotated real document images to illustrate the domain gap between real data distribution and three distributions synthetic data, faxified data, and CycleGAN generated data.

6.2.2 CycleGAN Training

The CycleGAN consists of two generators G and F and two discriminators D_X and D_Y. Domain X represents Source Domain and Domain Y represents Target Domain. The aim is to transform Synthetic Document Images into Realistic Document Images. Hence, The Synthetic Document Images represent the Source Domain and Real Document Images represent the Target Domain.

6.2.3 Training a Classifier on Synthetic Document Images

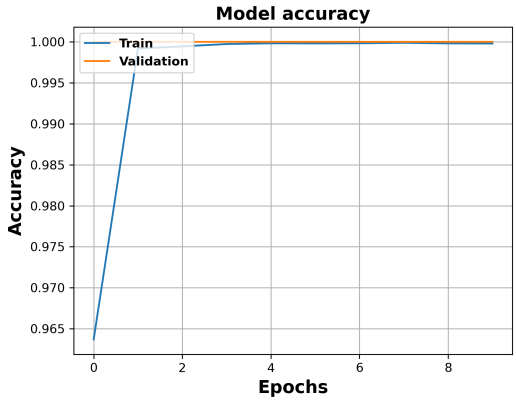


Figure 6.1: Epoch vs Accuracy. Training the Classifier on Synthetic Document Images.

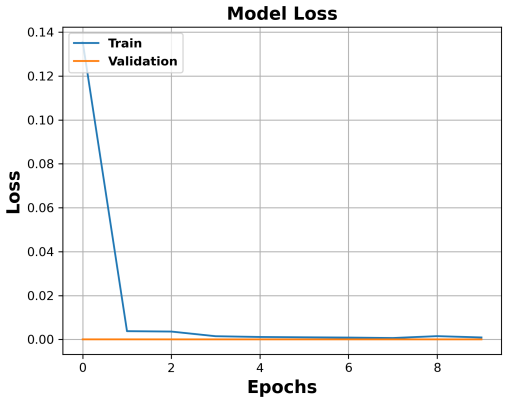


Figure 6.2: Epoch vs Loss Plot. Training the Classifier on Synthetic Document Images.

	Precision	Recall	F1-score	Support
DE_LY_Arm_2020-01	0.84	0.48	0.61	44
DE_LY_Bein_2018-08	0.07	0.49	0.12	47
DE_LY_Bein_2019-01	0.23	0.22	0.23	50
DE_LY_Bein_2019-07	0.22	0.43	0.29	60
DE_LY_Bein_2020-01	0.86	0.23	0.36	624
DE_LY_Bein_2020-03	0.23	0.19	0.21	128
DE_LY_Hand_2020-01	0.92	0.69	0.79	16
DE_PH_Bein_2018-09	0.16	0.23	0.19	22
DE_PH_Bein_2019-02	0.10	0.21	0.14	28
DE_PH_Bein_2020-01	0.28	0.51	0.36	143
Accuracy			**0.30**	1162
Macro Average	0.39	0.37	0.33	1162
Weighted Average	0.59	0.30	0.33	1162

Table 6.1: Classification Report. The Classifier is trained On Synthetic Document Images, its Classification Performance Evaluated on the Annotated Real Document Images.

6.2.4 Training a Classifier on CycleGAN Generated Document Images

6.2.5 Training a Classifier on Faxified Document Images

The faxification process mimics the way the fax machine works. It transforms the images in such a way like it was sent via fax. Whi

Usually, fax machines use a horizontal resolution of 1728 pixels and transmit only black-and-white images, which might be dirty and are generally also not aligned perfectly. The faxification process is not deterministic, involves randomness during the process of faxification of the images. It uses several image transformations like gamma, brightness, rotations, resizing, rescaling, binarization, adding noise, adding verticle lines, and conversion to a grayscale image. In figure 6.4, it is visible a snippet from the synthetic document image that has transformed randomly when it has been through the faxification process.

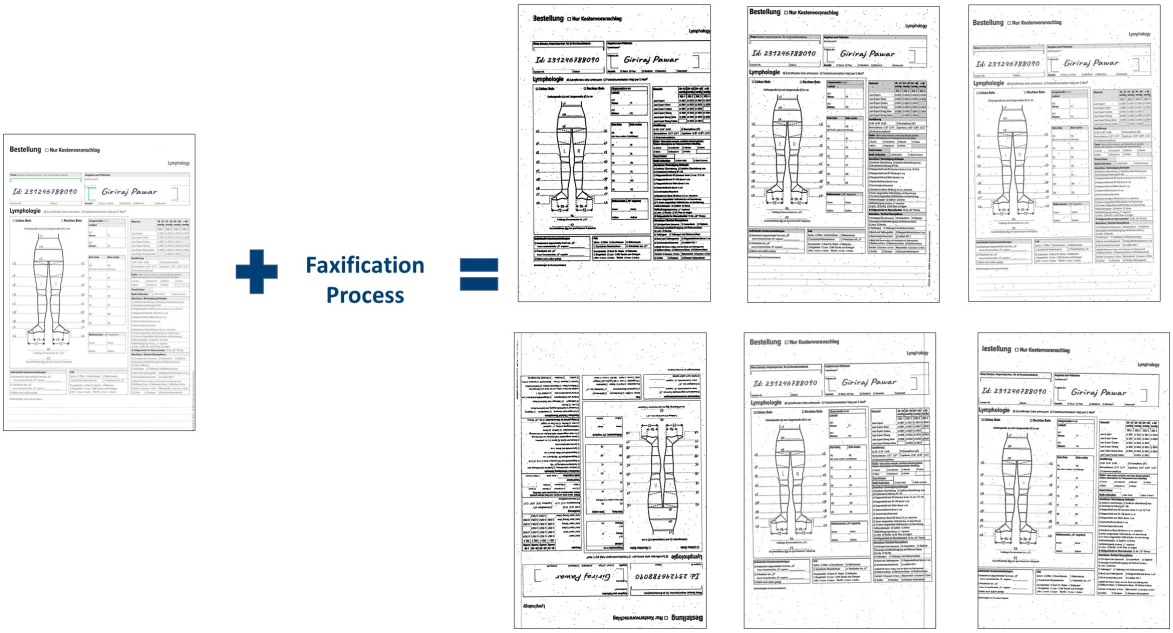


Figure 6.3: Illustration of Faxification Process Applied on Synthetic Document Images.

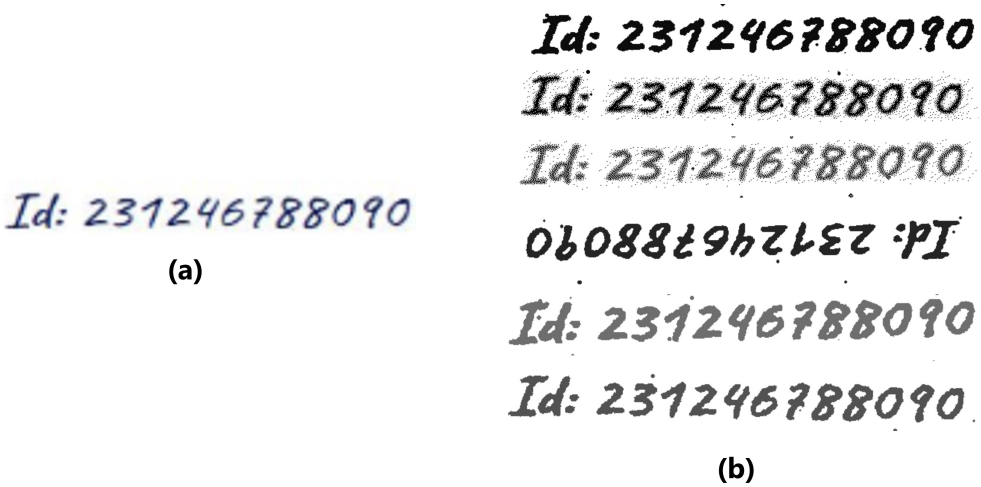


Figure 6.4: Illustration of faxified images to conclude that faxification process is a random process, the input images are faxified randomaly to create distinct output. For example, snippets shown in (b) for a single type of image snippet in (a) are distinct and random. (a) Snippet from the synthetic document image. (b) Snippets from the faxified document images.

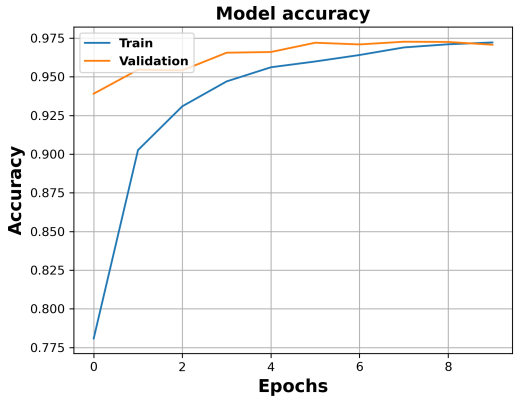


Figure 6.5: Epoch vs Accuracy Plot. Training the Classifier on Faxified Document Images.

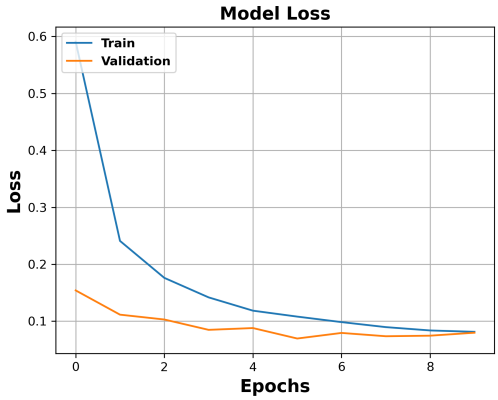


Figure 6.6: Epoch vs Loss Plot. Training the Classifier on Faxified Document Images.

	Precision	Recall	F1-score	Support
DE_LY_Arm_2020-01	1.00	0.95	0.98	44
DE_LY_Bein_2018-08	1.00	0.28	0.43	47
DE_LY_Bein_2019-01	0.46	0.26	0.33	50
DE_LY_Bein_2019-07	0.47	1.00	0.63	60
DE_LY_Bein_2020-01	0.80	0.03	0.05	624
DE_LY_Bein_2020-03	0.17	0.97	0.29	128
DE_LY_Hand_2020-01	0.67	1.00	0.80	16
DE_PH_Bein_2018-09	0.56	0.41	0.47	22
DE_PH_Bein_2019-02	0.71	0.61	0.65	28
DE_PH_Bein_2020-01	0.99	0.98	0.99	143
Accuracy			0.39	1162
Macro Average	0.68	0.65	0.56	1162
Weighted Average	0.73	0.39	0.32	1162

Table 6.2: Classification Report. The Classifier is trained On Faxified Document Images, its Classification Performance Evaluated on the Annotated Real Document Images.

6.3 Results

6.3.1 Qualitative Results

6.3.2 Quantitative Results

6.3.3 Overview of Domain Gap between Distribution

7. Conclusion and Future Work

7.1 Conclusion

This method of unsupervised domain adaptation helps improve the performance of machine learning models in the presence of a domain shift. It enables training of models that are performant in diverse scenarios, by lowering the cost of data capture and annotation required to excel in areas where ground truth data is scarce or hard to collect.

7.2 Future Works

Neural networks are a breakthrough technique in the advancement of modern machine learning systems. However, despite the exceptional learning capacity and improved generalizability, these neural networkd still suffer from poor transferability. This is the challenge of domain adaptation — a transformation in the relationship between data collected across different domains

A. Appendix

A.1 MNIST Handwritten Numbers Dataset

Figure A.1: Examples of Handwritten Numbers from the MNIST Dataset.[1]

A.2 GAN Training

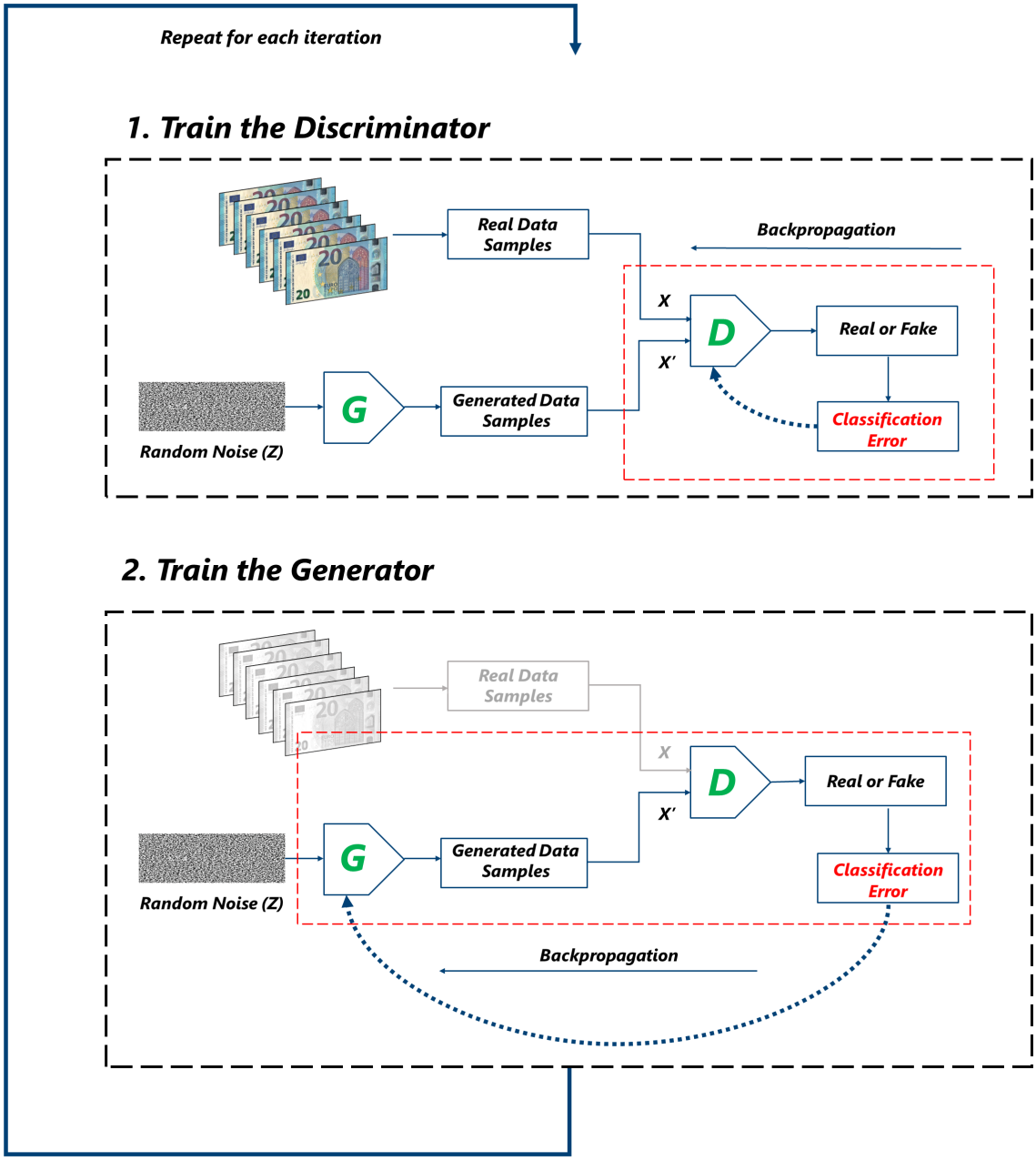


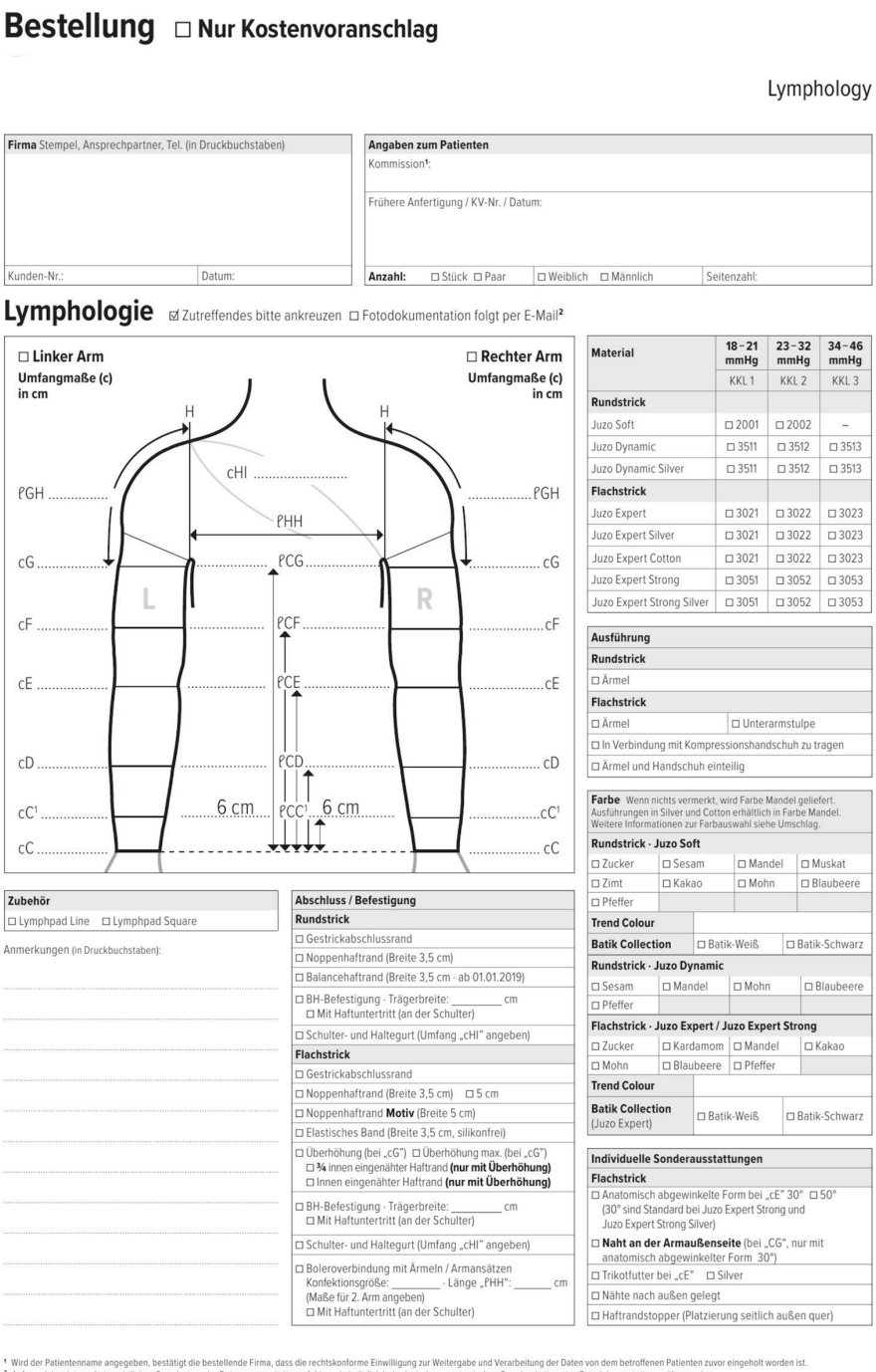
Figure A.2: Illustration of training of the GAN as per the algorithm.

[1] `http://yann.lecun.com/exdb/mnist/` last access: 31.03.2021

A.3 CycleGAN Training

A.4 Confusion Matrices

A.5 Examples of Document Images



Bestellung ☐ Nur Kostenvoranschlag

Lymphology

Firma Stempel, Ansprechpartner, Tel. (in Druckbuchstaben)

Kunden-Nr.: Datum:

Angaben zum Patienten

Kommission[1]:

Frühere Anfertigung / KV-Nr. / Datum:

Anzahl: ☐ Stück ☐ Paar ☐ Weiblich ☐ Männlich Seitenzahl:

Lymphologie ☑ Zutreffendes bitte ankreuzen ☐ Fotodokumentation folgt per E-Mail[2]

☐ Linker Arm
Umfangmaße (c) in cm

☐ Rechter Arm
Umfangmaße (c) in cm

H H cHI ℘GH ℘GH ℘HH cG ℘CG cG L R cF ℘CF cF cE ℘CE cE cD ℘CD cD cC¹ 6 cm ℘CC¹ 6 cm cC¹ cC cC

Material	18–21 mmHg KKL 1	23–32 mmHg KKL 2	34–46 mmHg KKL 3
Rundstrick			
Juzo Soft	☐ 2001	☐ 2002	–
Juzo Dynamic	☐ 3511	☐ 3512	☐ 3513
Juzo Dynamic Silver	☐ 3511	☐ 3512	☐ 3513
Flachstrick			
Juzo Expert	☐ 3021	☐ 3022	☐ 3023
Juzo Expert Silver	☐ 3021	☐ 3022	☐ 3023
Juzo Expert Cotton	☐ 3021	☐ 3022	☐ 3023
Juzo Expert Strong	☐ 3051	☐ 3052	☐ 3053
Juzo Expert Strong Silver	☐ 3051	☐ 3052	☐ 3053

Ausführung

Rundstrick

☐ Ärmel

Flachstrick

☐ Ärmel ☐ Unterarmstulpe

☐ In Verbindung mit Kompressionshandschuh zu tragen

☐ Ärmel und Handschuh einteilig

Farbe Wenn nichts vermerkt, wird Farbe Mandel geliefert. Ausführungen in Silver und Cotton erhältlich in Farbe Mandel. Weitere Informationen zur Farbauswahl siehe Umschlag.

Rundstrick · Juzo Soft

☐ Zucker ☐ Sesam ☐ Mandel ☐ Muskat
☐ Zimt ☐ Kakao ☐ Mohn ☐ Blaubeere
☐ Pfeffer

Trend Colour

Batik Collection ☐ Batik-Weiß ☐ Batik-Schwarz

Rundstrick · Juzo Dynamic

☐ Sesam ☐ Mandel ☐ Mohn ☐ Blaubeere
☐ Pfeffer

Flachstrick · Juzo Expert / Juzo Expert Strong

☐ Zucker ☐ Kardamom ☐ Mandel ☐ Kakao
☐ Mohn ☐ Blaubeere ☐ Pfeffer

Trend Colour

Batik Collection (Juzo Expert) ☐ Batik-Weiß ☐ Batik-Schwarz

Zubehör

☐ Lymphpad Line ☐ Lymphpad Square

Anmerkungen (in Druckbuchstaben):

Abschluss / Befestigung

Rundstrick

☐ Gestrickabschlussrand
☐ Noppenhaftrand (Breite 3,5 cm)
☐ Balancehaftrand (Breite 3,5 cm · ab 01.01.2019)
☐ BH-Befestigung · Trägerbreite: _____ cm
☐ Mit Haftuntertritt (an der Schulter)
☐ Schulter- und Haltegurt (Umfang „cHI" angeben)

Flachstrick

☐ Gestrickabschlussrand
☐ Noppenhaftrand (Breite 3,5 cm) ☐ 5 cm
☐ Noppenhaftrand **Motiv** (Breite 5 cm)
☐ Elastisches Band (Breite 3,5 cm, silikonfrei)
☐ Überhöhung (bei „cG") ☐ Überhöhung max. (bei „cG")
☐ ¾ innen eingenähter Haftrand **(nur mit Überhöhung)**
☐ Innen eingenähter Haftrand **(nur mit Überhöhung)**
☐ BH-Befestigung · Trägerbreite: _____ cm
☐ Mit Haftuntertritt (an der Schulter)
☐ Schulter- und Haltegurt (Umfang „cHI" angeben)
☐ Boleroverbindung mit Ärmeln / Armansätzen
Konfektionsgröße: _____ · Länge „℘HH": _____ cm
(Maße für 2. Arm angeben)
☐ Mit Haftuntertritt (an der Schulter)

Individuelle Sonderausstattungen

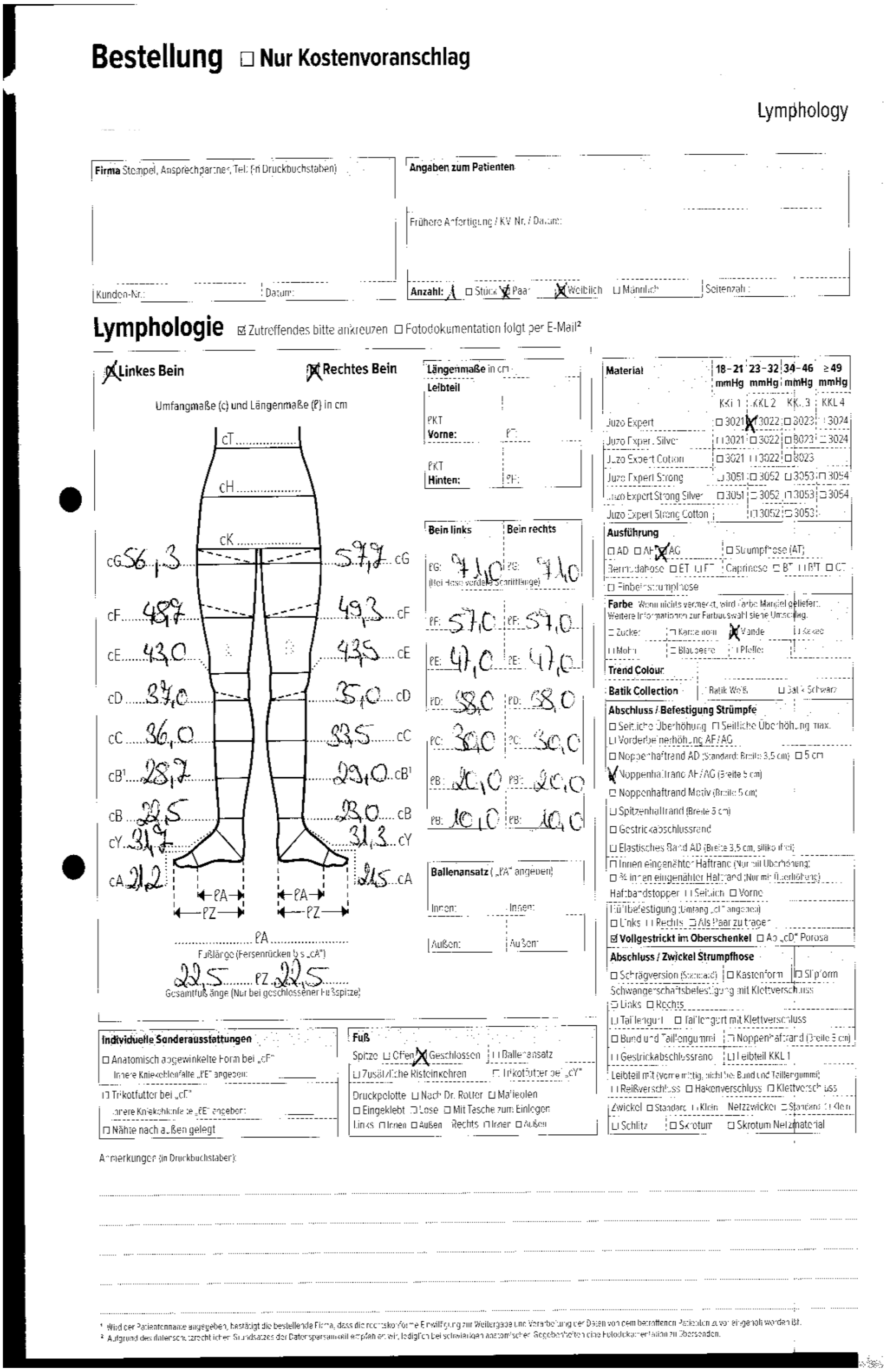
Flachstrick

☐ Anatomisch abgewinkelte Form bei „cE" 30° ☐ 50°
(30° sind Standard bei Juzo Expert Strong und Juzo Expert Strong Silver)
☐ **Naht an der Armaußenseite** (bei „CG", nur mit anatomisch abgewinkelter Form 30°)
☐ Trikotfutter bei „cE" ☐ Silver
☐ Nähte nach außen gelegt
☐ Haftrandstopper (Platzierung seitlich außen quer)

DEU · 1407301/78 · 8650 LY · 09/2018 · Änderungen und Irrtümer vorbehalten.

[1] Wird der Patientenname angegeben, bestätigt die bestellende Firma, dass die rechtskonforme Einwilligung zur Weitergabe und Verarbeitung der Daten von dem betroffenen Patienten zuvor eingeholt worden ist.
[2] Aufgrund des datenschutzrechtlichen Grundsatzes der Datensparsamkeit empfehlen wir, lediglich bei schwierigen anatomischen Gegebenheiten eine Fotodokumentation zu übersenden.

Figure A.3: Examples of unfilled form image.



Bestellung ☐ **Nur Kostenvoranschlag**

Lymphology

Firma Stempel, Ansprechpartner, Tel. (in Druckbuchstaben)

Angaben zum Patienten

Frühere Anfertigung / KV Nr. / Datum:

Kunden-Nr.: Datum:

Anzahl: ☐ Stück ☒ Paar ☒ Weiblich ☐ Männlich Seitenzahl:

Lymphologie ☒ Zutreffendes bitte ankreuzen ☐ Fotodokumentation folgt per E-Mail[2]

☒ **Linkes Bein** ☒ **Rechtes Bein**

Umfangmaße (c) und Längenmaße (ℓ) in cm

	Linkes Bein	Rechtes Bein
cG	56,3	57,7
cF	48,7	49,3
cE	43,0	43,5
cD	37,0	35,0
cC	36,0	33,5
cB[1]	28,7	29,0
cB	22,5	23,0
cY	31,7	31,3
cA	21,2	21,5

Fußlänge (Fersenrücken bis „cA")

Gesamtfußlänge (Nur bei geschlossener Fußspitze): 22,5 ℓZ 22,5

Längenmaße in cm — **Leibteil**: ℓKT Vorne:, ℓKT Hinten:

Bein links	Bein rechts
ℓG: 71,0	ℓG: 71,0
ℓF: 57,0	ℓF: 57,0
ℓE: 47,0	ℓE: 47,0
ℓD: 38,0	ℓD: 38,0
ℓC: 30,0	ℓC: 30,0
ℓB: 20,0	ℓB: 20,0
ℓB: 10,0	ℓB: 10,0

Ballenansatz („ℓA" angeben): Innen: Außen:

Material 18–21 mmHg KKL 1 | 23–32 mmHg KKL 2 | 34–46 mmHg KKL 3 | ≥49 mmHg KKL 4

Juzo Expert ☐ 3021 ☒ 3022 ☐ 3023 ☐ 3024
Juzo Expert Silver ☐ 3021 ☐ 3022 ☐ 3023 ☐ 3024
Juzo Expert Cotton ☐ 3021 ☐ 3022 ☐ 3023
Juzo Expert Strong ☐ 3051 ☐ 3052 ☐ 3053 ☐ 3054
Juzo Expert Strong Silver ☐ 3051 ☐ 3052 ☐ 3053 ☐ 3054
Juzo Expert Strong Cotton ☐ 3052 ☐ 3053

Ausführung ☐ AD ☐ AF ☒ AG ☐ Stumpfhose (AT) Bermudahose ☐ ET ☐ ET Caprihose ☐ BT ☐ BT ☐ CT ☐ Einbeinstrumpfhose

Farbe Wenn nichts vermerkt, wird Farbe Mandel geliefert. Weitere Informationen zur Farbauswahl siehe Umschlag. ☐ Zucker ☐ Karamell ☒ Mandel ☐ Sesam ☐ Mohn ☐ Blaubeere ☐ Pfeffer

Trend Colour

Batik Collection ☐ Batik Weiß ☐ Batik Schwarz

Abschluss / Befestigung Strümpfe ☐ Seitliche Überhöhung ☐ Seitliche Überhöhung max. ☐ Vorderbeinerhöhung AF/AG ☐ Noppenhaftrand AD (Standard: Breite 3,5 cm) ☐ 5 cm ☒ Noppenhaftrand AF/AG (Breite 5 cm) ☐ Noppenhaftrand Motiv (Breite 5 cm) ☐ Spitzenhaftrand (Breite 3 cm) ☐ Gestrickabschlussrand ☐ Elastisches Band AD (Breite 3,5 cm, silikonfrei) ☐ Innen eingenähter Haftrand ☐ ¾ innen eingenähter Haftrand Haftbandstopper ☐ Seitlich ☐ Vorne Hüftbefestigung (Umfang „cT" angeben) ☐ Links ☐ Rechts ☐ Als Paar zu tragen ☒ **Vollgestrickt im Oberschenkel** ☐ Ab „cD" Porosa

Abschluss / Zwickel Strumpfhose ☐ Schrägversion (Standard) ☐ Kastenform ☐ Slipform Schwangerschaftsbefestigung mit Klettverschluss ☐ Links ☐ Rechts ☐ Taillengurt ☐ Taillengurt mit Klettverschluss ☐ Bund und Taillengummi ☐ Noppenhaftrand (Breite 5 cm) ☐ Gestrickabschlussrand ☐ Leibteil KKL 1 Leibteil mit (vorne mittig, nicht bei Bund und Taillengummi) ☐ Reißverschluss ☐ Hakenverschluss ☐ Klettverschluss Zwickel ☐ Standard ☐ Klein Netzzwickel ☐ Standard ☐ Klein ☐ Schlitz ☐ Skrotum ☐ Skrotum Netzmaterial

Individuelle Sonderausstattungen ☐ Anatomisch angewinkelte Form bei „cF" Innere Kniekehlenfalte „ℓE" angeben: ☐ Trikotfutter bei „cF" Innere Kniekehlenfalte „ℓE" angeben: ☐ Nähte nach außen gelegt

Fuß Spitze ☐ Offen ☒ Geschlossen ☐ Ballenansatz ☐ Zusätzliche Ristelnkehren ☐ Trikotfutter bei „cY" Druckpelotte ☐ Nach Dr. Rotter ☐ Malleolen ☐ Eingeklebt ☐ Lose ☐ Mit Tasche zum Einlegen Links ☐ Innen ☐ Außen Rechts ☐ Innen ☐ Außen

Anmerkungen (in Druckbuchstaben):

Figure A.4: Example of real document image.

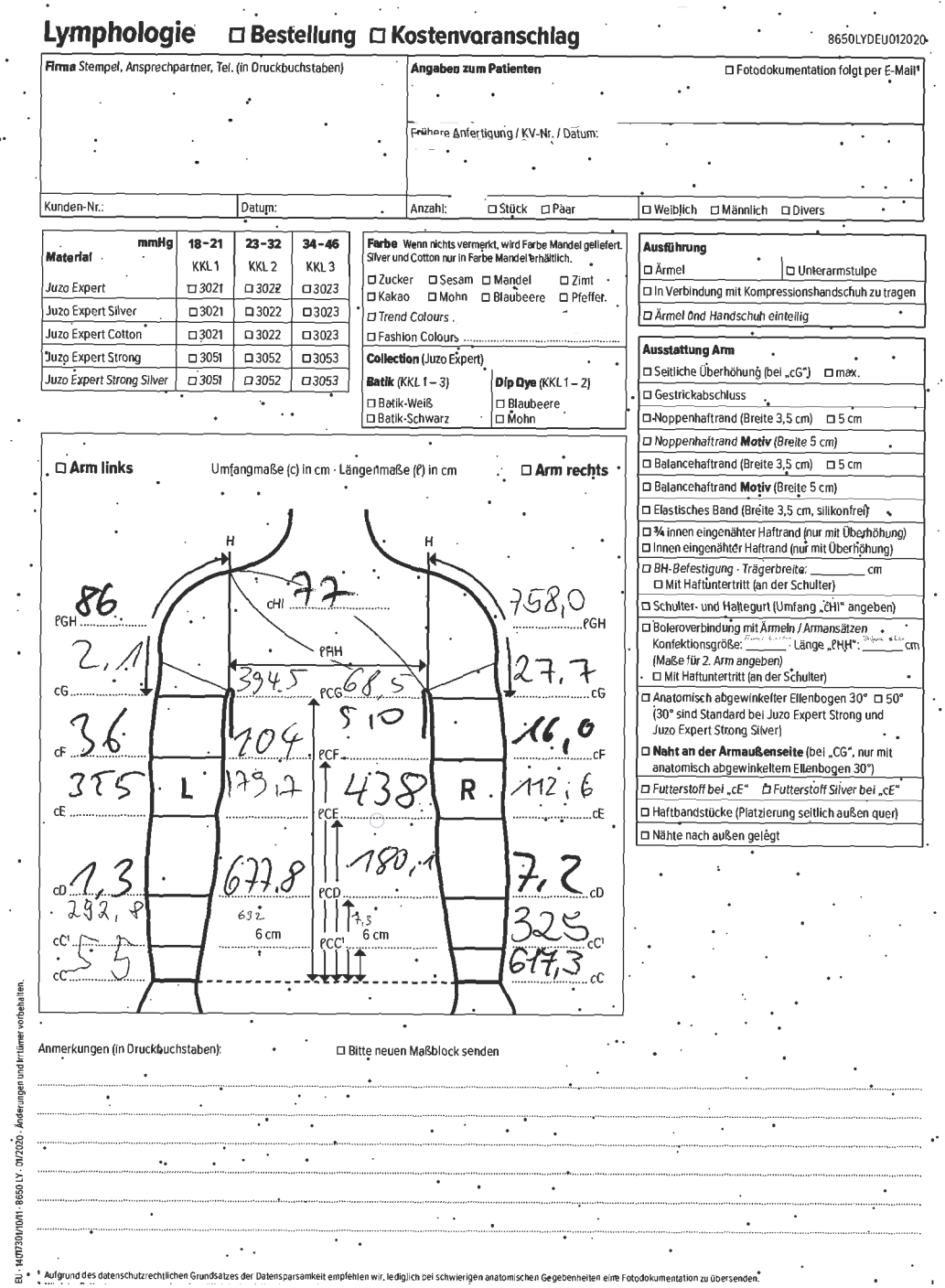


Figure A.5: Examples of faxified document image.

A.6 Classifier Architecture Diagram

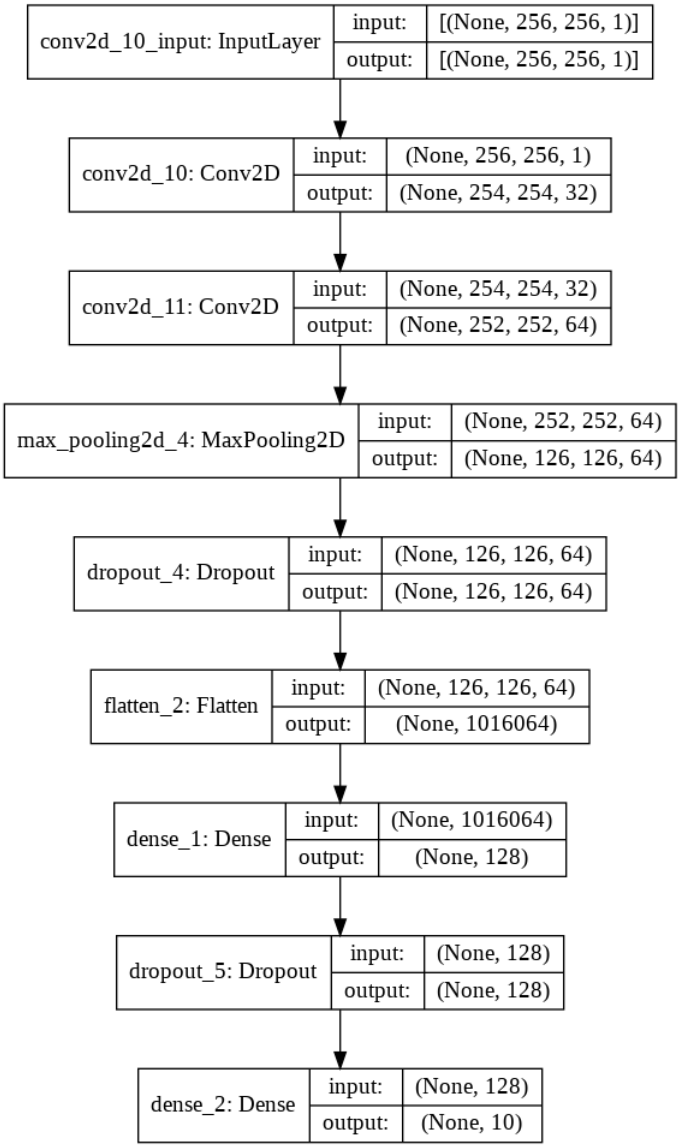


Figure A.6: Classifier Architecture Diagram.

A.7 Generator Architecture Diagram

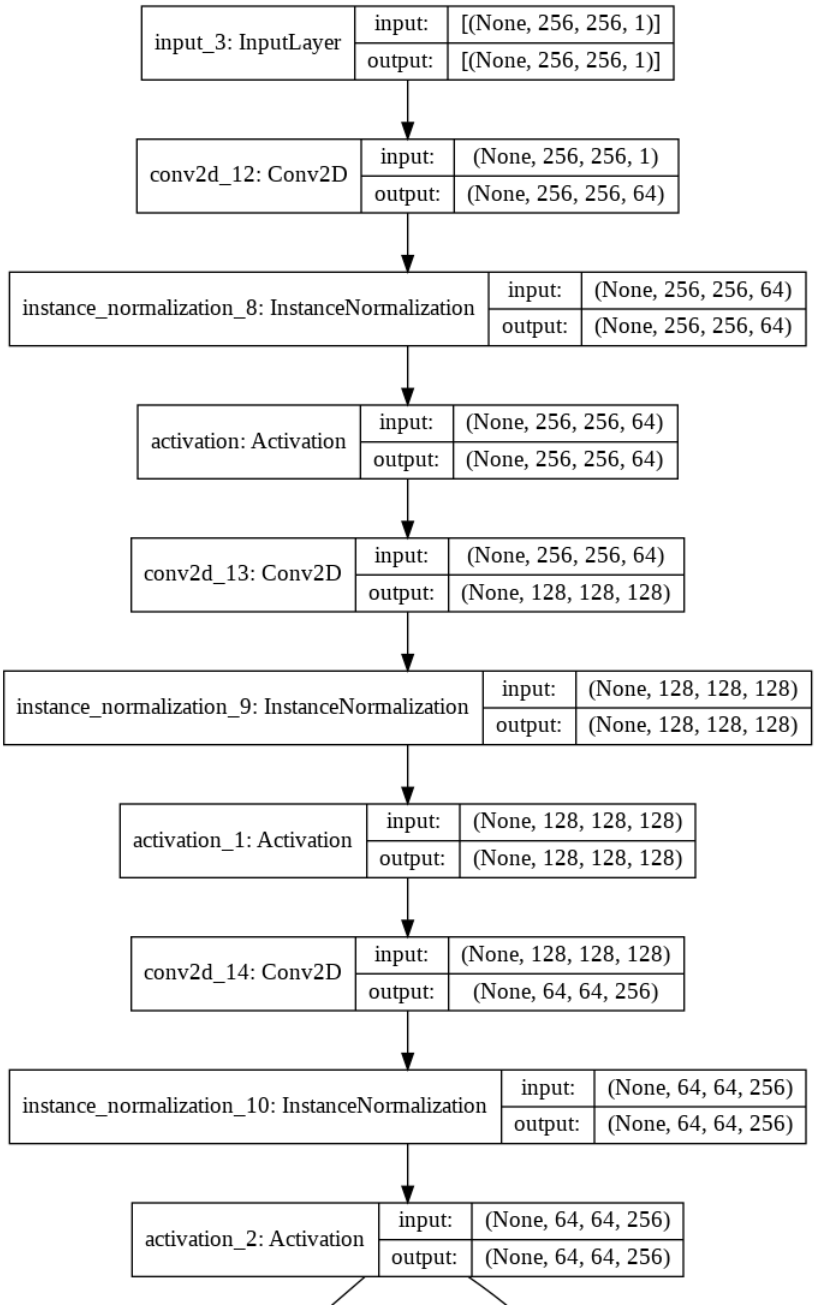


Figure A.7: Generator Architecture Diagram. Continue to Next Page.

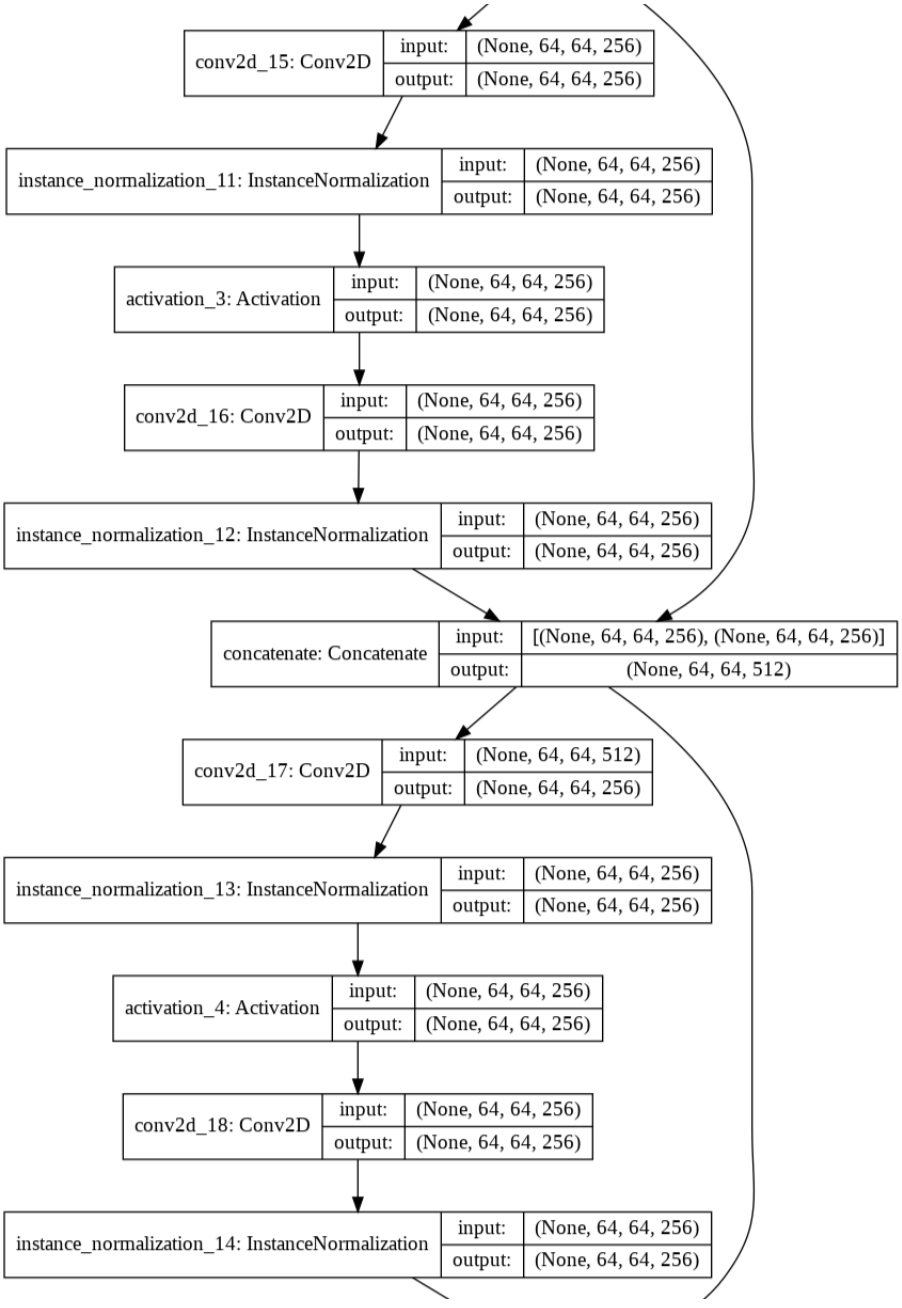


Figure A.8: Generator Architecture Diagram. Continue to Next Page.

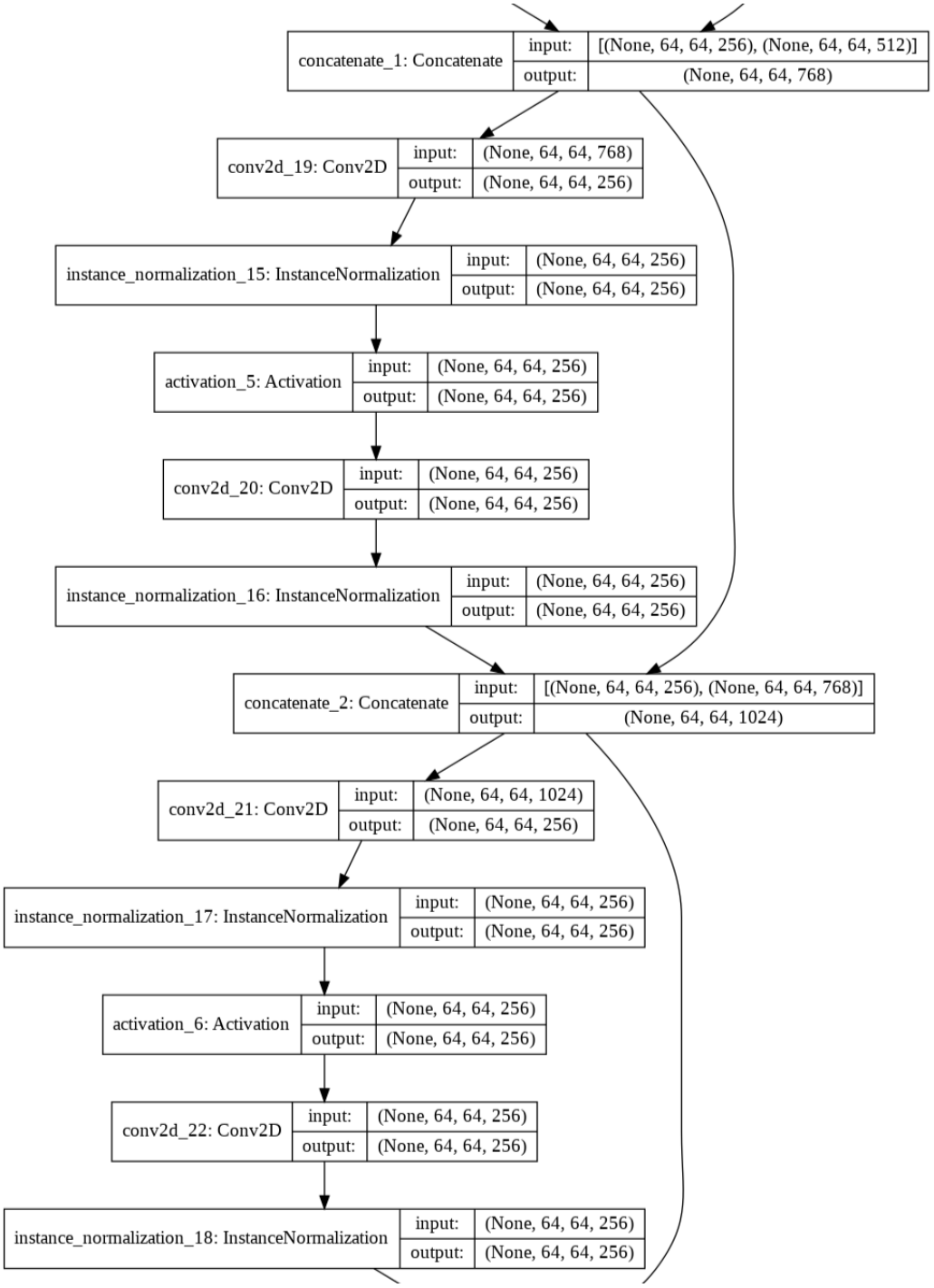


Figure A.9: Generator Architecture Diagram. Continue to Next Page.

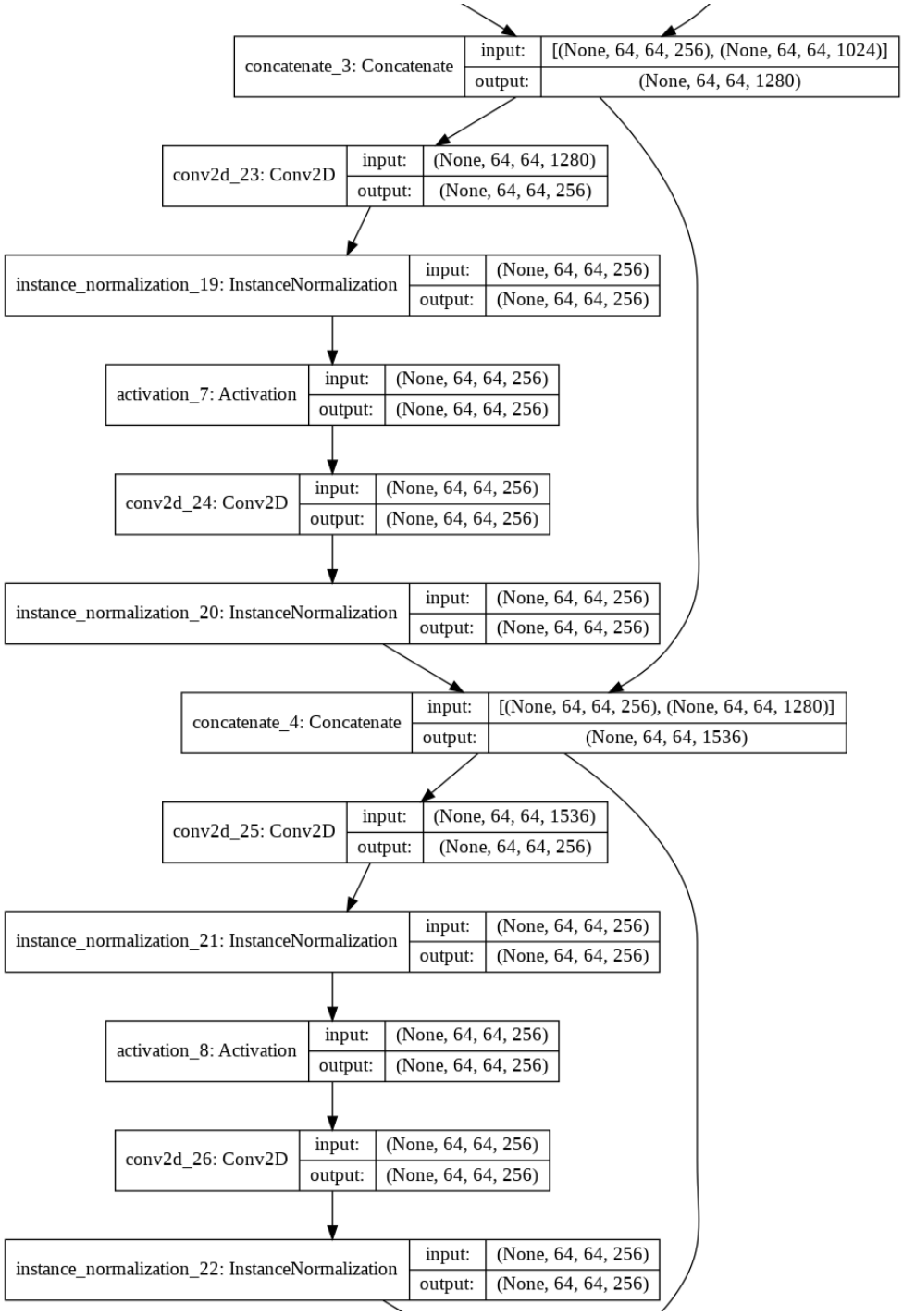


Figure A.10: Generator Architecture Diagram. Continue to Next Page.

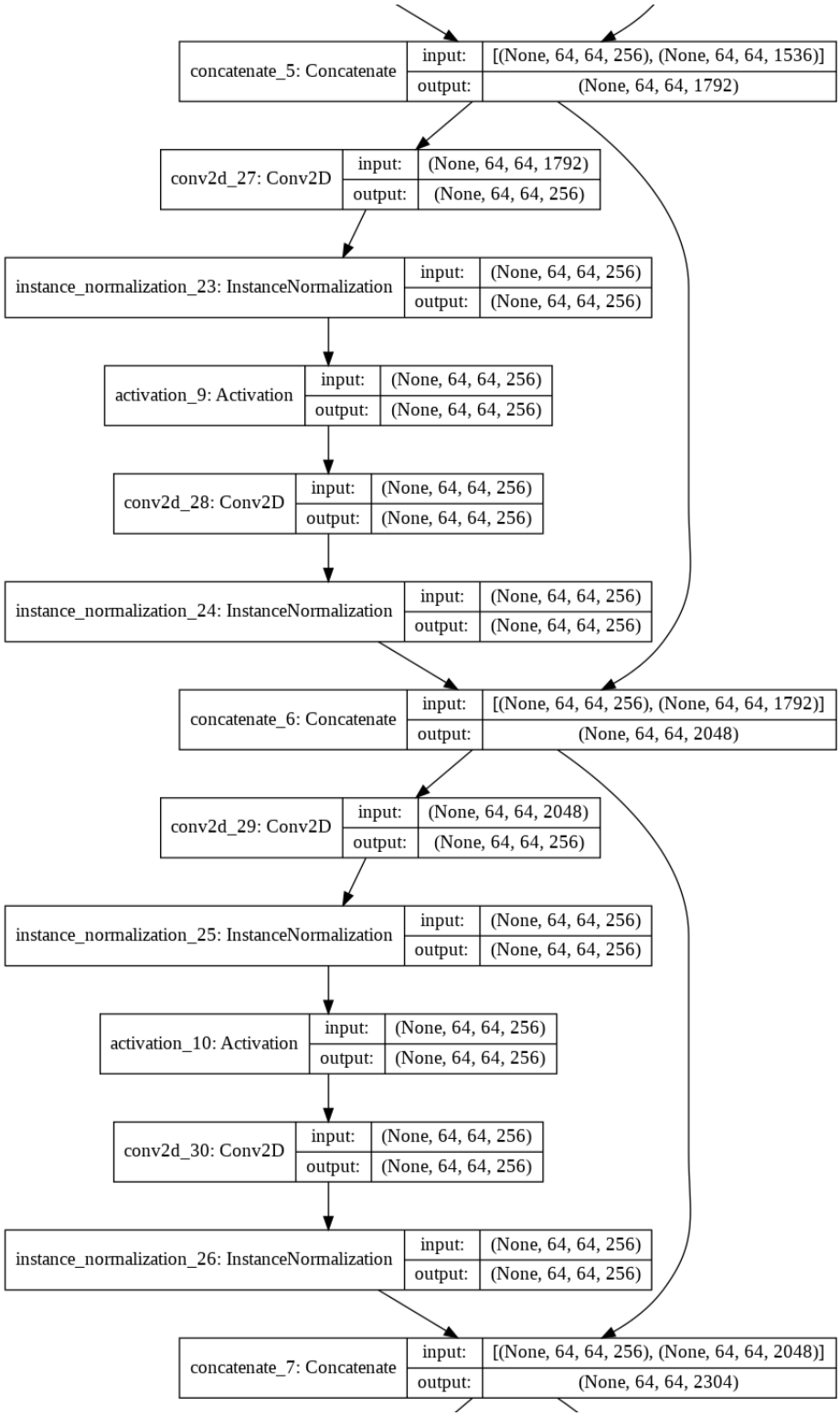


Figure A.11: Generator Architecture Diagram. Ends Here.

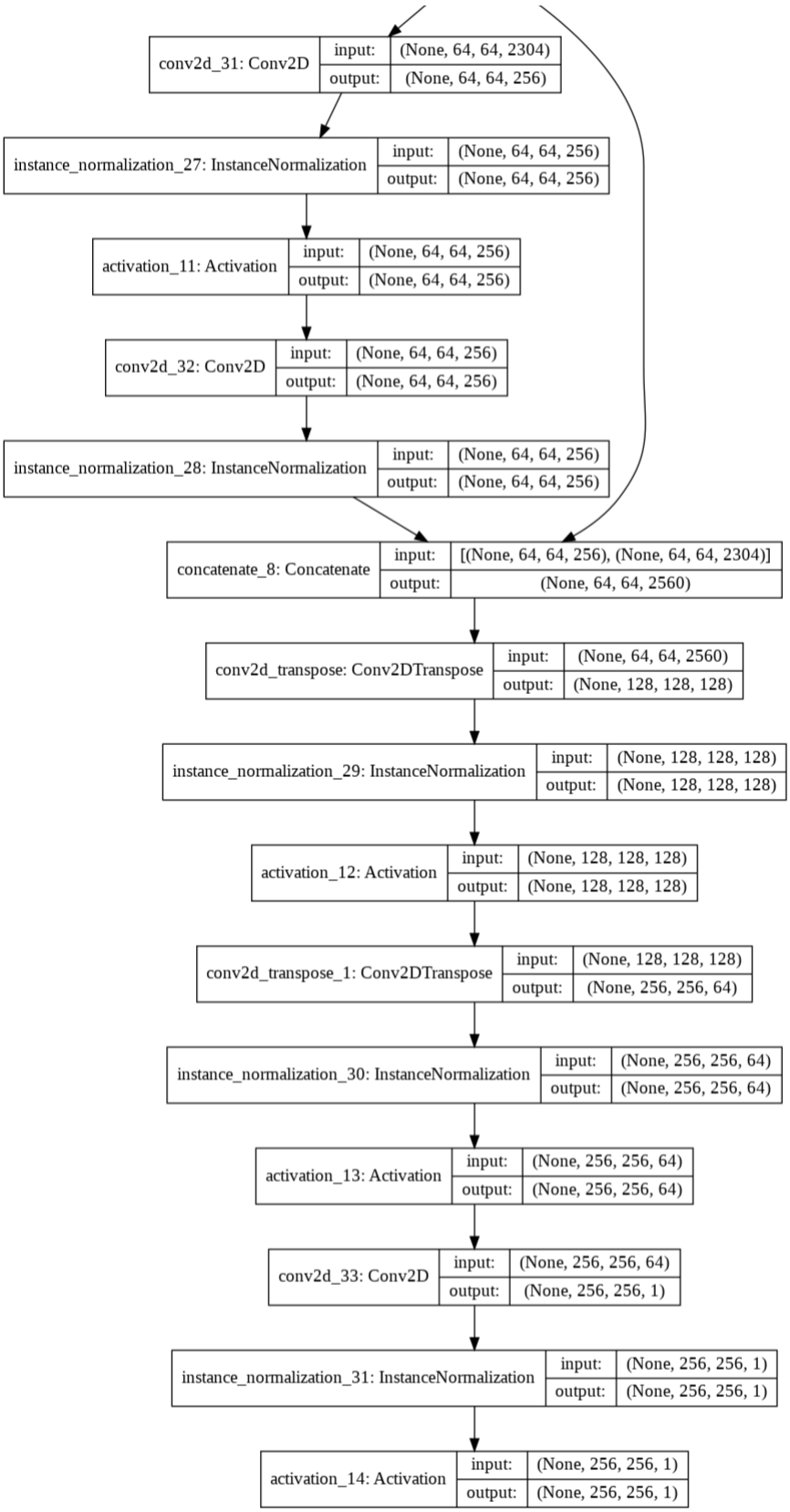


Figure A.12: Generator Architecture Diagram. Ends Here.

A.8 Discriminator Architecture Diagram

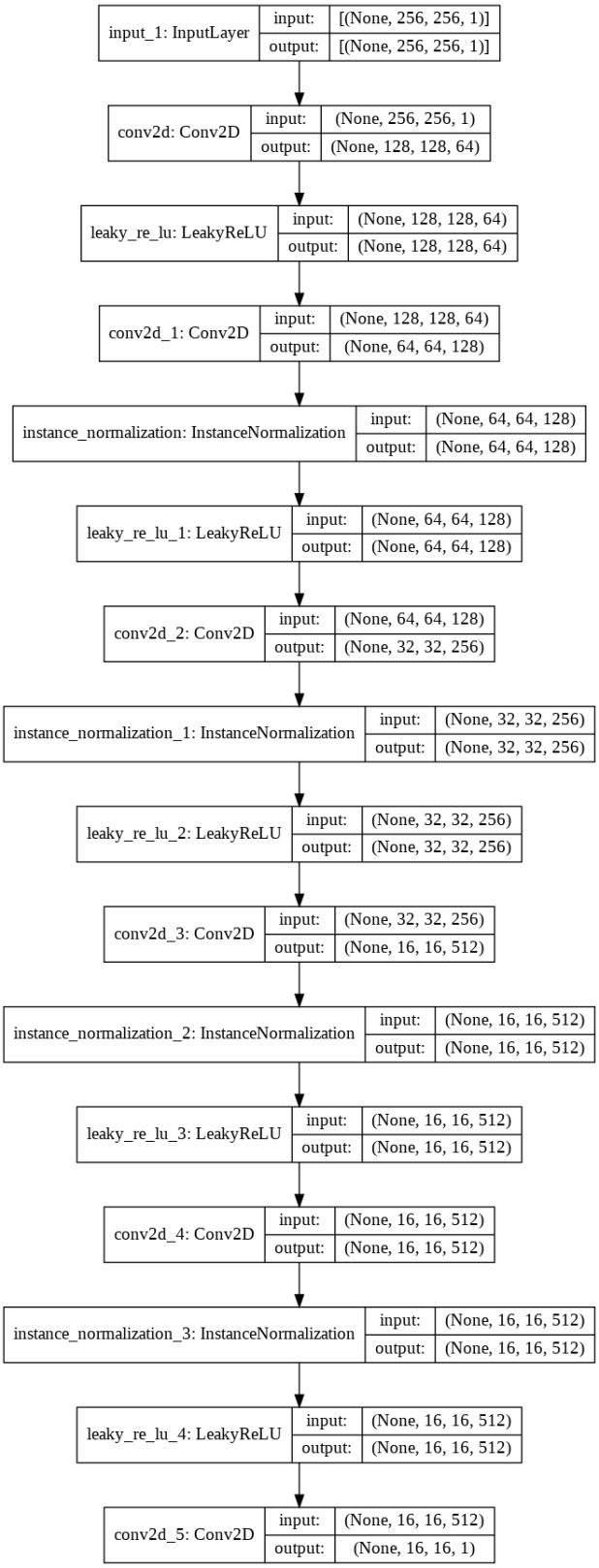


Figure A.13: Discriminator Architecture Diagram.

Bibliography

[1] J. Langr and V. Bok. *GANs in Action: Deep learning with Generative Adversarial Networks*. Manning Publications, 2019.

[2] Kun-Hsing Yu, Andrew L. Beam, and Isaac S. Kohane. Artificial intelligence in healthcare. *Nature Biomedical Engineering*, 2(10):719–731, 2018.

[3] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A survey of autonomous driving: Common practices and emerging technologies. *IEEE Access*, 8:58443–58469, 2020.

[4] Michael Brady. Artificial intelligence and robotics. In Michael Brady, Lester A. Gerhardt, and Harold F. Davidson, editors, *Robotics and Artificial Intelligence*, pages 47–63, Berlin, Heidelberg, 1984. Springer Berlin Heidelberg.

[5] Daniela Girimonte and Dario Izzo. *Artificial Intelligence for Space Applications*, pages 235–253. Springer London, London, 2007.

[6] Aboul-Ella Hassanien, Ahmad Taher Azar, Tarek Gaber, Diego Oliva, and Fahmy M. Tolba, editors. *Proceedings of the International Conference on Artificial Intelligence and Computer Vision (AICV2020)*. Springer International Publishing, 2020.

[7] Jan Kukačka, Vladimir Golkov, and Daniel Cremers. Regularization for deep learning: A taxonomy, 2017.

[8] Aurlien Gron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 1st edition, 2017.

[9] C. Tensmeyer, M. Brodie, D. Saunders, and T. Martinez. Generating realistic binarization data with generative adversarial networks. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 172–177, 2019.

[10] Patrick Hemmer, Niklas Kühl, and Jakob Schöffer. Deal: Deep evidential active learning for image classification, 2020.

[11] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, 2019.

[12] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning, 2020.

[13] Ievgen Redko, Emilie Morvant, Amaury Habrard, Marc Sebban, and Younès Bennani. A survey on domain adaptation theory: learning bounds and theoretical guarantees, 2020.

[14] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[15] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[16] Lei Kang, Marcal Rusinol, Alicia Fornes, Pau Riba, and Mauricio Villegas. Unsupervised adaptation for synthetic-to-real handwritten word recognition. *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, Mar 2020.

[17] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution, 2016.

[18] Q. A. Bui, D. Mollard, and S. Tabbone. Automatic synthetic document image generation using generative adversarial networks: Application in mobile-captured document analysis. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 393–400, 2019.

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

[20] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[21] Giorgio Metta and Angelo Cangelosi. *Cognitive Robotics*, pages 613–616. Springer US, Boston, MA, 2012.

[22] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 740–755, Cham, 2014. Springer International Publishing.

[23] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[24] Joshua Susskind, Adam Anderson, and Geoffrey E Hinton. The toronto face dataset. Technical report, Technical Report UTML TR 2010-001, U. Toronto, 2010.

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

[26] Yoshua Bengio, Grégoire Mesnil, Yann Dauphin, and Salah Rifai. Better mixing via deep representations, 2012.

[27] Yoshua Bengio, Eric Thibodeau-Laufer, Guillaume Alain, and Jason Yosinski. Deep generative stochastic networks trainable by backprop, 2014.

[28] P Manisha and Sujit Gujar. Generative adversarial networks (gans): What it can generate and what it cannot?, 2019.

[29] Hoang Thanh-Tung and Truyen Tran. On catastrophic forgetting and mode collapse in generative adversarial networks, 2020.

[30] M. R. Pavan Kumar and Prabhu Jayagopal. Generative adversarial networks: a survey on applications and challenges. *International Journal of Multimedia Information Retrieval*, 10(1):1–24, 2021.

[31] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks, 2017.

[32] Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop, 2016.

[33] C. Liu, F. Yin, Q. Wang, and D. Wang. Icdar 2011 chinese handwriting recognition competition. In *2011 International Conference on Document Analysis and Recognition*, pages 1464–1469, 2011.

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

[35] Junbo Zhao, Michael Mathieu, and Yann LeCun. Energy-based generative adversarial network, 2017.

[36] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks, 2017.

[37] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2018.

[38] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding, 2016.

[39] Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A. Efros. Generative visual manipulation on the natural image manifold, 2018.

[40] A. Yu and K. Grauman. Fine-grained visual comparisons with local learning. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 192–199, 2014.

[41] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[42] Chuan Li and Michael Wand. Precomputed real-time texture synthesis with markovian generative adversarial networks, 2016.

[43] Taesung Park, Alexei A. Efros, Richard Zhang, and Jun-Yan Zhu. Contrastive learning for unpaired image-to-image translation, 2020.

[44] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding, 2019.

[45] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. Unsupervised image-to-image translation networks, 2018.

[46] Hsin-Ying Lee, Hung-Yu Tseng, Qi Mao, Jia-Bin Huang, Yu-Ding Lu, Maneesh Singh, and Ming-Hsuan Yang. Drit++: Diverse image-to-image translation via disentangled representations, 2019.

[47] Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich, Kun Zhang, and Dacheng Tao. Geometry-consistent generative adversarial networks for one-sided unsupervised domain mapping, 2018.

[48] 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, 2018.

[49] Monika Sharma, Abhishek Verma, and Lovekesh Vig. Learning to clean: A gan perspective, 2019.

[50] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning, 2017.

[51] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. Adversarially learned inference, 2017.

[52] Ming-Yu Liu and Oncel Tuzel. Coupled generative adversarial networks, 2016.

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

[54] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017.

[55] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans, 2017.

[56] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1):53–65, Jan 2018.

[57] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015.

[58] D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243, 1968.

[59] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980.

[60] Rikiya Yamashita, Mizuho Nishio, Richard Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, 9, 06 2018.

[61] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

[62] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

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

[64] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[65] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress.

[66] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions, 2017.

[67] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. JMLR Workshop and Conference Proceedings.

[68] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning (adaptive computation and machine learning series). *Cambridge Massachusetts*, pages 321–359, 2017.

[69] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network, 2014.

[70] Yaniv Taigman, Adam Polyak, and Lior Wolf. Unsupervised cross-domain image generation, 2016.

[71] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[72] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[73] Zachary Chase Lipton, Charles Elkan, and Balakrishnan Narayanaswamy. Thresholding classifiers to maximize f1 score, 2014.