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

Pawar, Giriraj Sukumar

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

Chemnitz, September 30, 2021

Supervisor: Prof. Dr.-Ing. Gangolf Hirtz

Advisors: Dr.-Ing. Ana Cecilia Perez Grassi, M.Sc. Tobias Scheck

External Advisors: Dr.-Ing. Martin Voigt, M.Sc. Paul Fischer

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, September 2021.

Abstract

Neural networks have improved significantly in past decades. They are competent to solve complex problems in the field of deep learning, and they are capable to manage a large amount of complex data like images, videos, and sound. However, the training of neural networks requires a significantly large amount of annotated data, which is not always possible. Machine learning engineers inevitably have to generate synthetic data. Although, 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 scarcity of annotated data. The domain adaptation technique can transform data from the source domain to the target 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. This thesis proposes an image-to-image translation application that aims to close the domain gap between synthetic data distribution and real data distribution using Cycle-Consistent Adversarial Networks (CycleGANs). The proposed application is used to transform synthetic document images into realistic document images, to overcome the scarcity of annotated real document images. In addition, these generated real document images are used to train a classifier to classify similar unlabeled real document images, thereby accelerating the process of labeling images in an unsupervised and automated manner. Experimental results show the generated realistic document images are qualitatively convincing and need improvement quantitatively. Such preliminary results show that CycleGAN can solve the problem of data scarcity by generating high-quality images in the target domain. The purpose of this thesis is limited to improving the classification of real document images. Once the rich data is generated in the target domain, the performance of the real document image classifier can eventually be improved. The work of this thesis is limited to the study of CycleGAN as an image-to-image conversion method without paired training data. The remaining methods and comparisons with them are left for future work. In the future, CycleGAN can be used to generate realistic images in many tasks, such as handwriting recognition, image classification, image segmentation, and object detection.

Keywords: CycleGAN, Generative Adversarial Network (GAN), Domain Gap, Domain Adaptation, Image-to-Image Translation, Data Distributions.

Acknowledgments

I would like to express my gratitude to my advisors, Dr.-Ing. Gangolf Hirtz, Dr.-Ing. Ana Cecilia Perez Grassi, Dr.-Ing. Martin Voigt, Tobias Scheck, Paul Fischer, Thaddäus Strobel, and Clemens Reinhardt, who helped me throughout this project. Also, I would like to extend a special thanks to my parents and my friends for their enormous support. I would like to dedicate this thesis to my teacher Subhash K. U. for teaching me how to program, may his soul rest in peace and god gives strength to his family.

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
DCGAN	Deep Convolutional Generative Adversarial Network
EBGAN	Energy-based Generative Adversarial Network
CycleGAN	Cycle-Consistent Adversarial Network
ACL-GAN	Adversarial Consistency Loss Generative Adversarial Network
DBN	Deep Belief Network
Stacked-CAE	Stacked Convolutional Autoencoder
Deep-GSN	Deep Generative Stochastic Networks
GT	Ground Truth
DIBCO	Document Image Binarization Competition
PSNR	Peak Signal-to-Noise Ratio
ResNet	Residual Network
CUT	Contrastive Unpaired Image-to-Image Translation
MUNIT	Multimodal Unsupervised Image-to-image Translation
DRIT	Diverse Image-to-Image Translation
VAE	Variational Autoencoder
GCGAN	Geometry-Consistent Generative Adversarial Networks
FastCUT	Fast Contrastive Unpaired Translation
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
ReLU	Rectified Linear Unit
1D	One-dimensional
2D	Two-dimensional

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[2]. AI has a presence in many sectors like Healthcare[3], Autonomous Vehicles[4], Robotics[5], Space Exploration[6], and Computer Vision[7]. 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 machine learning.

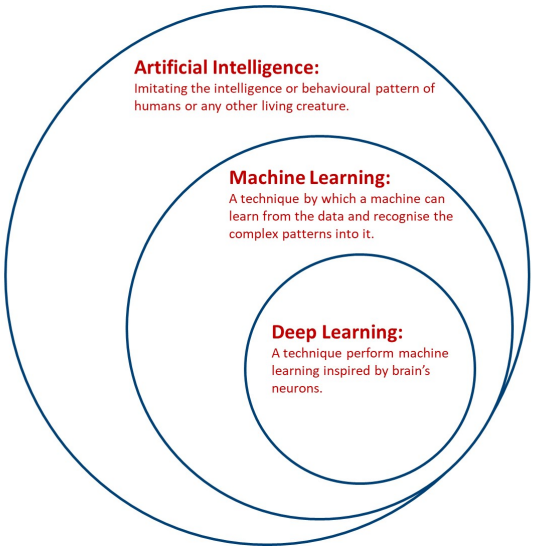


Figure 1.1: Relationship between Artificial Intelligence, Machine Learning, and Deep Learning.

The concept of deep learning was invented as early as the 1950s, although it was largely ignored until the 1980s and 1990s. However, with the efficient, fast computation hardware, and abundant data since 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. The two main things that can cause difficulties are bad algorithms and bad data. Let's start with some examples of bad data. Insufficient training data is one of the main problems that may arise when training deep learning models. A large amount of training data is required for most deep learning models to work properly. The second is the poor quality data. If the training data is wrong, noisy, and full of outliers, it will make it difficult for the model 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.

Now, that after some examples of bad data, let's look at some of the examples of bad algorithms. Overfitting and underfitting are one of the main problems of deep learning. "Overfitting happens when the model is too complex relative to the amount and noisiness of the training data"[8]. In the overfitting, model learns training data including noise to the extent, it negatively impacts the performance of the model, leading to higher generalization error on unseen data. One of methods used to decrease the risk of overfitting is regularization[9]. Regularization is one of the solutions provided to generalize a model better to the new examples. "Underfitting is the opposite of overfitting, it occurs when a model is too simple to learn the underlying structure of the data"[8]. In the case of the underfitting model, it neither learns training data nor generalize to the unseen data. The problem of underfitting is resolved "by choosing a powerful model with more parameters and providing better features during the model learning process"[8]. 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. But in many cases, data is scarce and it is very difficult to have data that is labeled and annotated.

1.2 Motivation

Deep learning methods have many effective applications in several fields, including natural language processing and computer vision. However, these methods still are limited by poor generalization due to the insufficient quantity of training data[10]. 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. However, due to the high cost of data annotation, it is difficult to obtain a large set of annotated training data, especially when there are many classes 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[11], Data Augmentation[12], Transfer Learning[13], and Domain Adaptation[14].

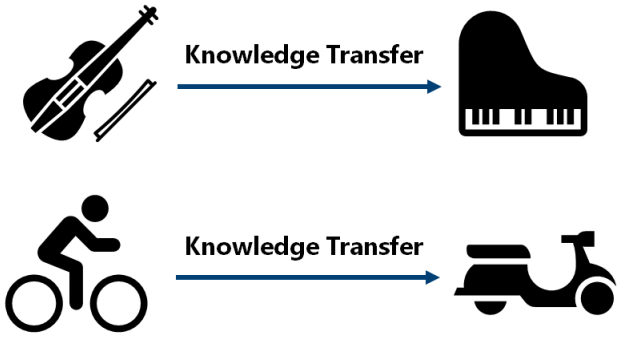


Figure 1.2: Simple examples of transfer learning.

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. "Transfer learning is the ability of a system to recognize and apply the knowledge learned from one task to another task"[13]. 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 shows an 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.[15] have performed survey on transfer learning, more information about the transfer learning can found in their research paper.

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[15]. 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[15]. 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[15]. 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, and unsupervised domain adaptation. In the supervised domain adaptation, the samples in the target domains are labeled. Semi-supervised domain adaptation has a small set of labeled and unlabeled samples in the target domain. And, in unsupervised domain adaptation, the samples in the target domain are not labeled[15]. A simple example of domain adaptation of SVHN transformed into handwritten digits shown in figure 1.3. The difference between traditional machine learning and domain adaptation is shown in figure 1.5.

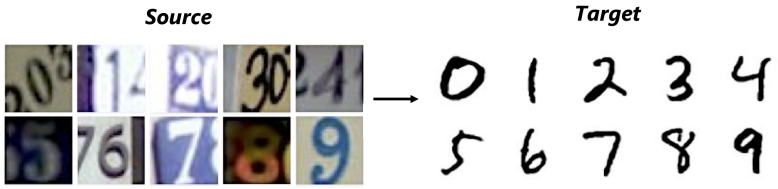


Figure 1.3: Simple example of domain adaptation from SVHN dataset[16] to MNIST Dataset[1] for the digit recognition task.[2]

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 is represented by the MNIST dataset, which is a collection of handwritten digits images. When the CNN is trained and evaluated on the source domain SVHN dataset for the task of identifying numbers, it will achieve high 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 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

[1] `http://yann.lecun.com/exdb/mnist/` last access: September 30, 2021

[2] `https://machinelearning.apple.com/research/bridging-the-domain-gap-for-neural-models` last access: September 30, 2021

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)[17], Image Classification[15], Style Transfer[18], and Optical Character Recognition (OCR)[19] to solve the problem of scarcity of data. Making such applications robust, accurate, and efficient is a big task, which requires years of research. The scope of this thesis is limited to improving document image classification using the proposed domain adaptation technique.

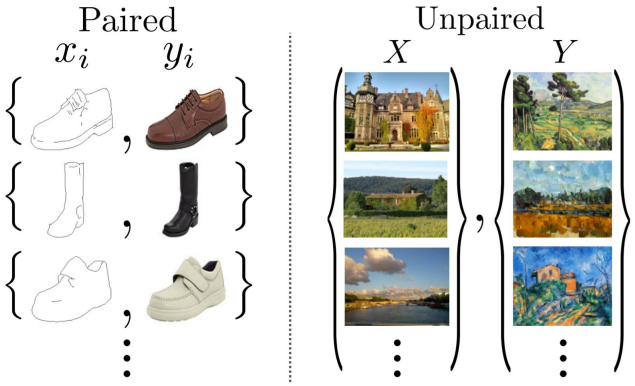


Figure 1.4: The paired training data set consists of training examples, in which there is a correspondence between the source domain and the target domain (Left, Edges ↔ Images). But in the unpaired training data set, there is no correspondence between the source domain and the target domain (Right, Photographs ↔ Painitings)[20].

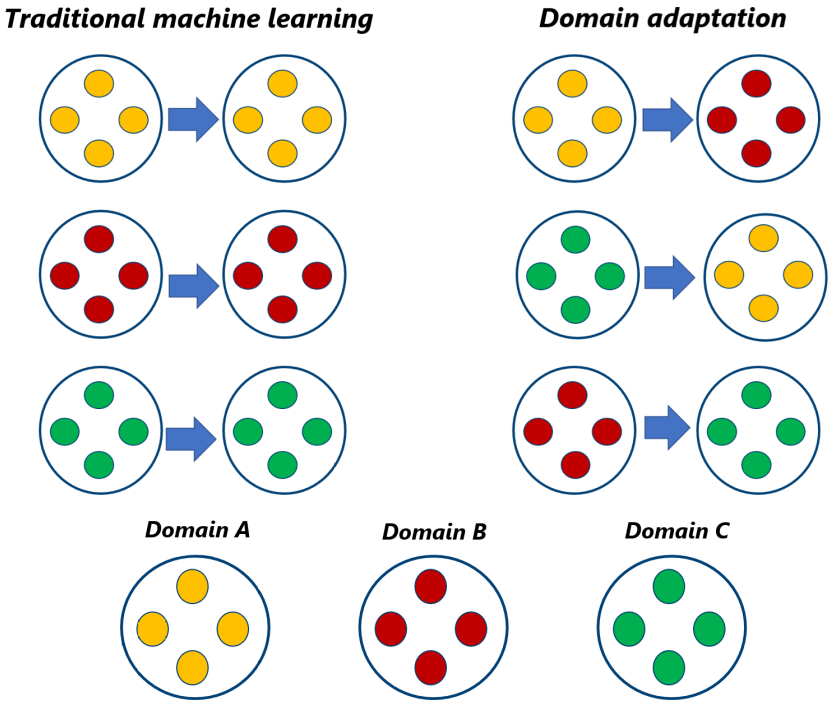


Figure 1.5: Difference between traditional machine learning and domain adaptation.

1.3 Problem Statement and Proposed Solution

As described earlier, a large amount of data is required to train a neural network, but annotated data is scarce, and data labeling is a costly and tedious job. For example, in this thesis, it is being addressed that collecting several and distinct real document images (figure A.9) with different types of handwriting is expensive and difficult job. In such cases, machine learning engineers have to inevitably generate synthetic document images. However, deep learning models trained using synthetic document images will not generalize well on unseen real document images[10] (figure 1.6). Because, the synthetic document images lacks realism[10]. They don't possess a similar noise distribution, characteristic, and artifacts as real document images[10]. Hence, in the last two decades, numerous domain adaptation methodologies have been introduced to perform image-to-image translation[19]. Image-to-image translation is a method of transforming images from source to a target domain by learning the underlying characteristic of target domain images. It is used to transform synthetic images into realistic images by reducing the divergence or gap between the distribution of real data and the distribution of synthetic data. In this thesis, an image-to-image translation application is developed using CycleGAN to reduce the domain gap between synthetic data distribution and real data distribution[20]. The CycleGAN is an extended variant of Generative Adversarial Network (GAN)[21]. It is a method to perform unpaired image-to-image translation, in which the model is trained using the collection of images from the source and target domain that are not related to each other[3].

The proposed image-to-image translation application is developed 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[22] to automate document processing. Elevait has developed state-of-the-art HTR and OCR tools to process documents and extract information from them. 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, as mentioned earlier, the synthetic document images will not generalize well when they have to process real document images because real document images have different characteristics. Furthermore, real document images possess artifacts such as salt-and-pepper, background noise, blur due to camera motion or shake, watermarks, stains, wrinkles, and fading text, introduced during the scanning process[23]. To address this problem, in this thesis, the image-to-image translation application is developed to transform synthetic document images into realistic document images to reduce the domain gap between real data distribution and synthetic data distribution. In this way, a large number of realistic document images can be generated to reduce the scarcity of annotated real document images in the target domain.

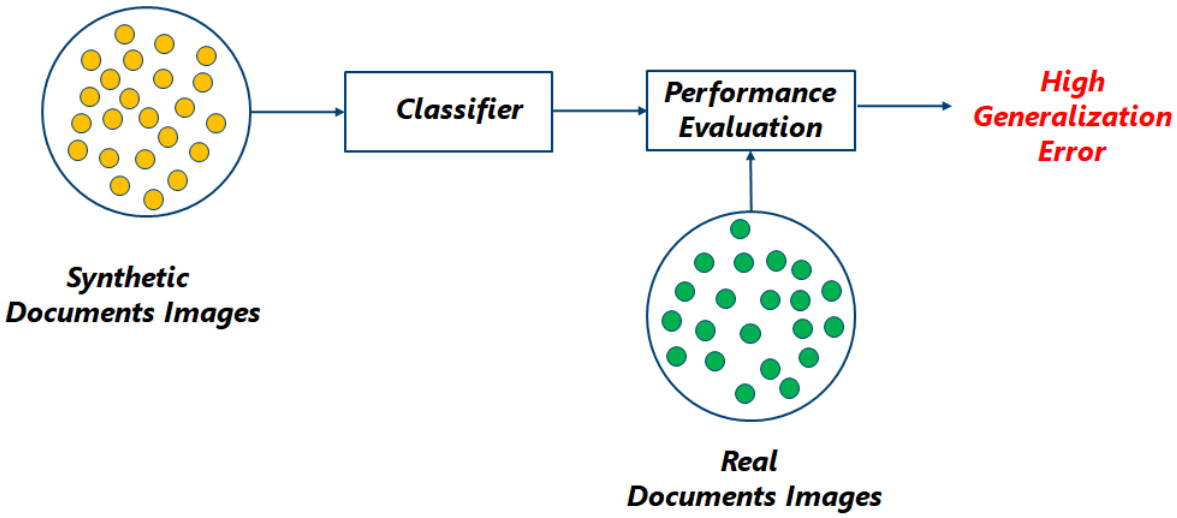


Figure 1.6: An illustration of the problem to be solved in this thesis.

[3] `https://machinelearningmastery.com/what-is-cyclegan/` last access: September 30, 2021

The synthetic document images are created using templates (figure A.8) and handwritten crops (figure 5.1) retrieved from handwriting datasets like MNIST (figure A.1) or any other datasets. These templates 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[24]. The handwritten crops are inserted over the empty templates to generate numerous synthetic document images. The proposed image-to-image translation application solves the following problems. First, labeled synthetic document images are available in the source domain, ultimately, using the proposed application the collection of synthetic document images are transformed into realistic document images in the target domain. So the problem of labeling, annotations, and scarcity is solved. Second, consider, a huge chunk of real document images 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 1.6 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 1.7 Data distribution in light green color represents realistic document images which is generated by the proposed image-to-image translation application. Dark green represents the real document images. Further, the classifier is trained using realistic document images, 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 when compared to figure 1.6.

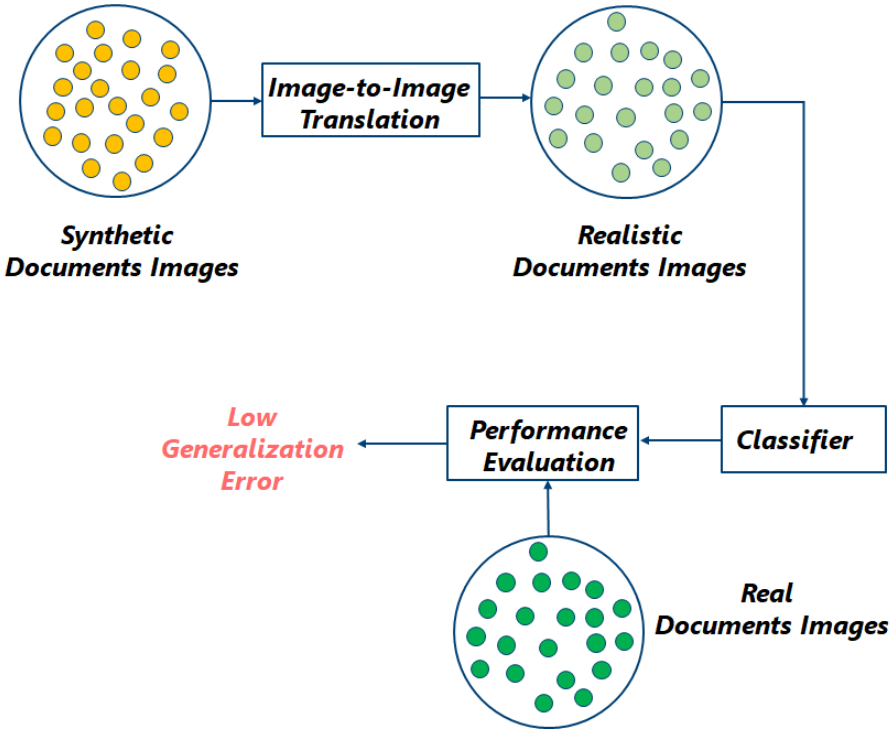


Figure 1.7: An illustration of the solution proposed to reduce the domain gap between the synthetic document images and the real document images.

1.4 Thesis Objectives

The goal of this thesis is to close the domain gap between synthetic data distribution and real data distribution by developing an image-to-image translation application using CycleGAN, which is used to transform synthetic document images into realistic document images. In addition, in this thesis, experiments are conducted to understand the domain gap between data distributions. The main objectives of this thesis are listed below:

1. Conduct a literature survey, in which different types of image-to-image translation methods are discussed and briefly explained. In addition, also discuss a theoretical comparison between existing methods, which lead to the reason for choosing CycleGAN to solve the problem statement.
2. Create a dataset of synthetic document images, faxified document images, and real document images. In addition, a testing dataset of annotated real document images is collected for evaluation of the models.
3. Implement proposed image-to-image translation application using CycleGAN. Next, train the application model using a dataset of synthetic document images and real document images. A dataset of synthetic document images represents the source domain, and a dataset of real document images represents the target domain.
4. Train three different classifiers on different data distributions (Synthetic document images, Faxified document images, and CycleGAN generated document images) and evaluate their performance on unseen annotated real document images (testing dataset) and analyze the domain gap between these distributions and real data distribution, using metrics like accuracy, macro average F1-score, and weighted average F1-score. These experiments are listed briefly below:
   (a) Train a classifier using synthetic document images and evaluate its performance over testing dataset.
   (b) Train a classifier using faxified document images and evaluate its performance over testing dataset.
   (c) Train a classifier using CycleGAN generated document images and evaluate its performance over testing dataset. This experiment reveals how well the CycleGAN generated document images are generalizing to the real document images, eventually determining the quality of images generated by the proposed image-to-image translation application.
5. Record all details regarding this thesis, for example, literature survey, selected methodology, fundamentals required to understand this thesis, implementation, experiments performed, obtained results, analysis of results, limitations and scope of the research, 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 constraints. The solution of the defined problem and analysis performed in this thesis investigated only using CycleGAN. The comparison with other methodologies is kept apart for future research. The scope of this thesis is limited to the improvement of real document images classification by increasing the quality and quantity of annotated images in the target domain using the proposed image-to-image translation application. The thesis is organized as follows. Chapter 2 describes related literature and existing methods to solve the problem. Chapter 3 discusses essential topics GANs and CNNs. Chapter 4 describes the method and loss functions used to solve the defined problem. Chapter 5 describes the details about the dataset, the architecture of the neural networks implemented in this thesis. Chapter 6 illustrates the evaluation metrics, experiments performed, and obtained results. Chapter 7 concludes the research and results analysis along with the future work and the limitations of the thesis.

2. Related Works

In the last decades, the field of computer vision and computer graphics has made a tremendous amount of progress, especially in the field of domain adaptation. The image-to-image translation is a classic example of domain adaptation, in which the goal is to learn the mapping between a source image and a target image using a training set of aligned image pairs. GANs were among the initial methods used to perform image-to-image translation[25][21]. But, GANs suffers from unstable learning and mode collapse[26]. Also, for many tasks, aligned or paired training data will not be available, and collecting and annotating paired document images is tedious, difficult, time-consuming, and costly. Hence, over the years, several stable supervised and unsupervised image-to-image translation methods are developed. This chapter discusses some of those popular image-to-image translation methods. Also, a brief theoretical comparison of these existing methods is discussed. The motivation behind selecting the approach to solve the addressed problem statement of this thesis is summarized in section 2.2.

2.1 Literature Survey

Ian J. Goodfellow et al.[21] proposed a framework of GANs in which two models are simultaneously trained. "A generative model is trained to learn the data distribution, and a discriminative model that predicts the probability that a sample came from the training data rather than a generative model. The idea of training a generative model is to generate realistic data and maximize the probability of a discriminative model making a mistake. The generator learns to generate realistic data, and the discriminator learns to distinguish real data from the generator's fake data. The generator is penalized by the discriminator for producing non-credible results. As training starts, the generator begins to create fake data, and the discriminator quickly learns to classify that it's fake and the generator is penalized to improve and produce credible results. The generator gets closer to producing output samples that can fool the discriminator as training progresses. Finally, if generator training goes well, the discriminator becomes worse at distinguishing between real and fake samples. At the end of the training, ultimately, we have a generator model which produces credible results which are similar to real data"[21]. Authors have trained GAN on a range of datasets including MNIST[27], the Toronto Face Database (TFD)[28], and CIFAR-10[29], also compared against already existing methods like Deep Belief Network (DBN)[30], Stacked Convolutional Autoencoder (Stacked-CAE)[30], and Deep Generative Stochastic Networks (Deep-GSN)[31]. The authors do not claim that "the samples generated by GANs are better than samples generated by already existed methods"[21]. Authors believe "at least the obtained results are competitive when compared with the better generative models in the literature, which highlights the potential of the generative adversarial framework"[21]. "The first and foremost advantage of GANs is computation when compared to other generative models. Adversarial models may have added statistical advantage from the generator network being updated by the gradients flowing through the discriminator using backpropagation, not being updated directly with data examples"[21]. Backpropagation is an algorithm to calculate gradients of the loss function to update weights of the neurons in the neural network[2]. Authors metioned, to prevent the Helvetica Scenario[32], special care must be taken while training the GANs. The generator must not be trained too much without updating the discriminator, training of both models should happen simultaneously. The discriminator must be updated well to provide maximum error to update the generator to produce high-quality images. In the Helvetica Scenario, the generator collapses, repeatedly producing the same output or a small set of outputs. Usually, GANs should produce a wide variety of outputs. The Helvetica Scenario is also called Mode Collapse[26].

Xudong Mao et al.[34] proposed another variant of GANs called Least Squares Generative Adversarial Networks (LSGANs). "The discriminator is hypothesized to be a classifier with the sigmoid cross-entropy loss function in regular GANs. They realized, however, that the cross-entropy loss func-

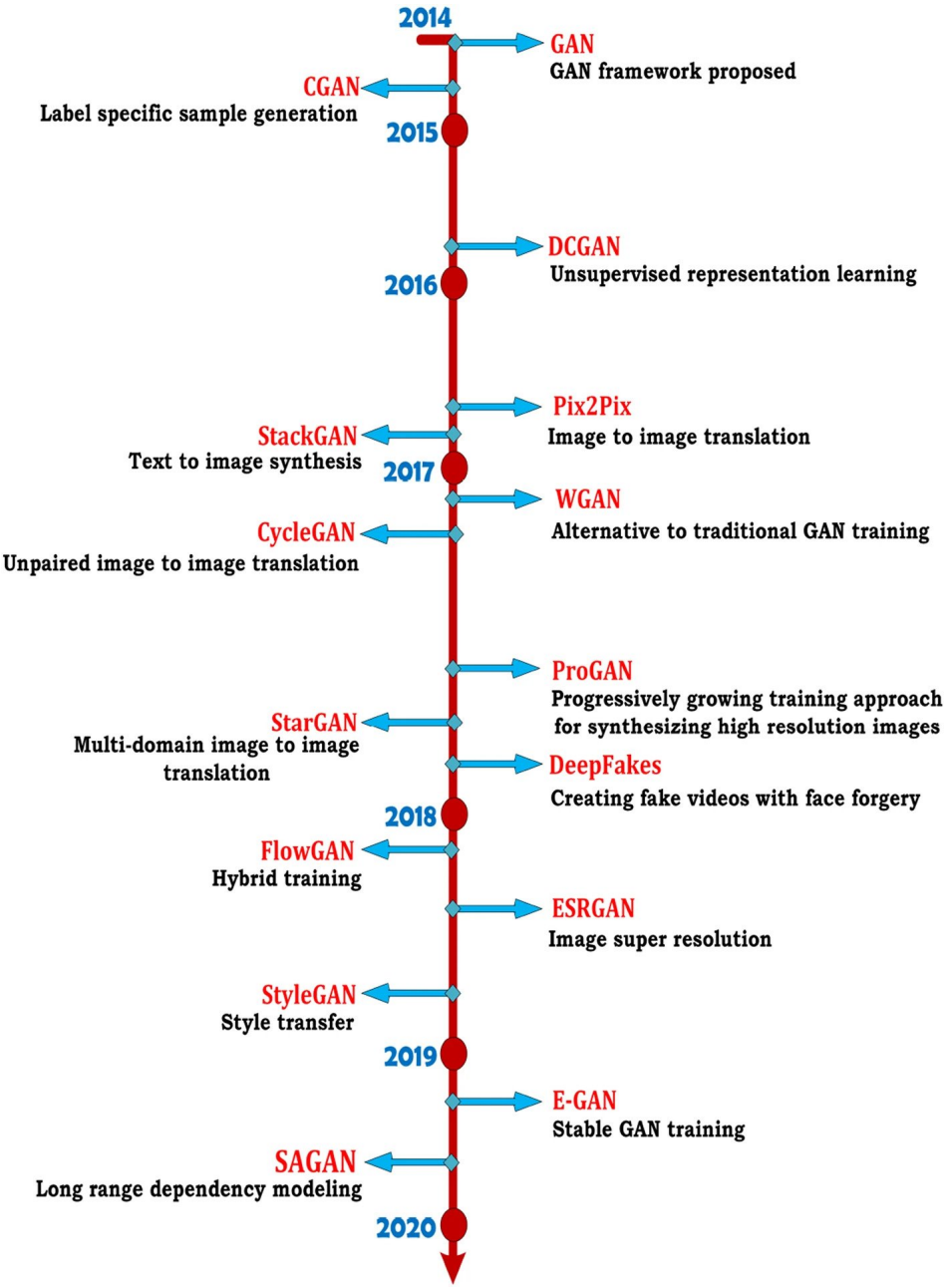


Figure 2.1: Evolution of GANs Over the Years[33].

tion can cause vanishing gradients during the learning process"[34]. Hence, the authors presented the LSGANs, which uses the least-squares loss function for the discriminator to solve this problem. "The fake samples are penalized by the least-squares loss function, which forces the generator to generate samples closer towards to the decision boundary"[34]. The authors evaluated LSGANs using two datasets LSUN[35] and HWDB1.0 (Handwritten Chinese Character Dataset)[36]. When trained on the LSUN dataset[35] they observed, the images generated by LSGANs are of better quality than the ones generated by the two baseline methods, Deep Convolutional Generative Adversarial Networks (DCGANs)[37] and EBGANs[38]. 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 LSGANs on a Gaussian mixture distribution dataset, which is designed in literature by Luke Metz et al.[39]. 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 the Gaussian mixture distribution successfully"[34]. Further, in this paper, for evaluating the stability, numerous comparison experiments are conducted and the results demonstrate that LSGANs can generate higher quality images than regular GANs, DCGANs[37], and Energy-based Generative Adversarial Networks (EBGANs)[38] and the learning process is stable compared to the regular GANs.

Phillip Isola et al.[40] 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[41], reconstructing objects from edge maps[42][43], and colorizing images[44]. "cGAN learns the mapping from the input image to output image along with a loss function to train this mapping. This allows using the same generic method to the distinct problems, that traditionally would require very different loss formulations"[40]. "In cGAN the generator uses U-Net-based architecture, the U-Net is an encoder-decoder with skip connections between mirrored layers in the encoder and decoder stacks[45]. The discriminator uses a convolutional PatchGAN classifier, which only penalizes structure at the scale of image patches[46]. Also, unlike unconditional GANs, both the generator and discriminator observe the input images"[40]. Authors found "L1 loss encourages less blurring compared to the L2 loss"[40]. 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"[40]. Authors have released cGAN as the pix2pix software and mentioned, cGAN has widespread applications, and it can be adopted without ease to solve different image-to-image translation problems without significant parameter modifications. 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 additional information like input edge map[40].

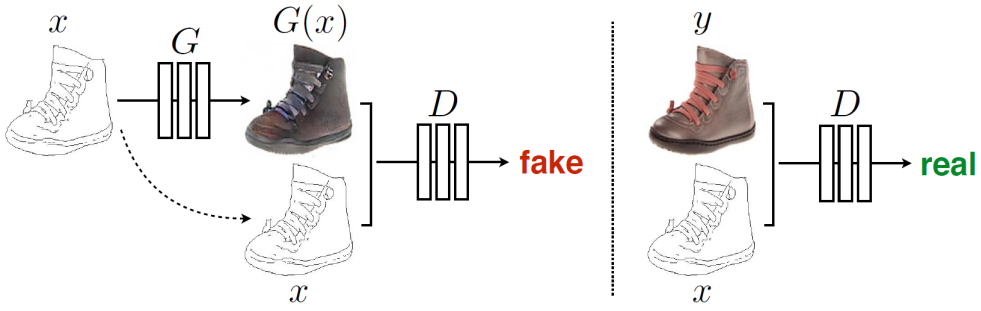


Figure 2.2: An illustration of the training a cGAN to map edges → photo transformation. In cGANs, both discriminator and generator observe the input edge map, unlike unconditional GANs[40].

Taesung Park et al.[47] proposed Contrastive Unpaired Image-to-Image Translation (CUT) for encouraging content preservation in unpaired image-to-image translation problems by maximizing the mutual information between input and output with patchwise contrastive learning[48]. Authors stated, "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"[47]. They demonstrated, "the framework enables one-sided translation in the unpaired image-to-image translation setting while improving quality, consuming less memory, and reducing training time"[47]. The prime advantage of this method is it does not require lots of memory[49][50] or specialized architecture[51][52]. The contrastive representation is formulated within the same image, hence this method can even be trained on single images, where each domain is having only a single image[47]. The several prior methods like CycleGAN[20], Multimodal Unsupervised Image-to-image Translation (MUNIT)[53], Diverse Image-to-Image Translation (DRIT)[54], and Geometry-Consistent Generative Adversarial Networks (GCGAN)[55] were unable to achieve significant 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. 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.

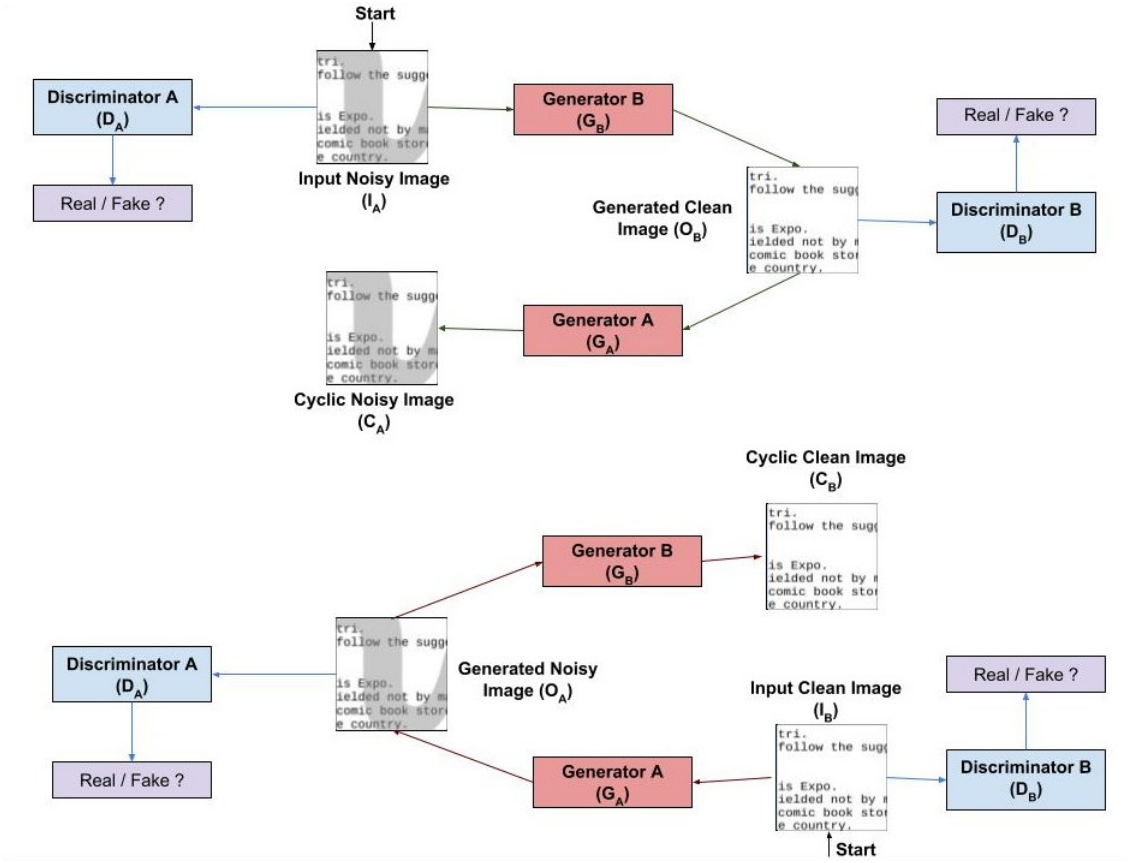


Figure 2.3: An illustration of CycleGAN transforming the noisy document images into clean document images, and vice versa. The generators G_A and G_B are responsible for the mapping of noisy images to clean images using cycle-consistency loss[20]. The two discriminators D_A and D_B rejects samples generated by G_A and G_B respectively acting as an adversary[23].

Monika Sharma et al.[23] are addressing "a problem in the scanning process that often results in the introduction of blur due to camera motion, salt and pepper noise, or water markings, shake, wrinkles, coffee stains, or faded data. These artifacts pose many readability challenges to current text recognition algorithms and significantly degrade their performance"[23]. "The existing denoising techniques require a paired training dataset consisting of noisy documents paired with cleaned versions of the same document. However, in the real world, to generate clean documents from noisy versions, such a paired training dataset is not available to train a model"[20][23]. Hence, the authors

used CycleGAN to transform noisy document images to clean document images and vice versa[20]. "CycleGAN is a method to learn a mapping between two data distributions in the absence of paired training dataset"[20][23]. They have compared the performance of CycleGAN for document cleaning tasks with 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 public document datasets, one each for deblurring, watermark removal, background noise removal, and defading. Finally, they concluded "CycleGAN learns a more robust mapping from the space of noisy to clean documents compared to cGAN"[23]. The complete architecture of CycleGAN for denoising documents illustrated in figure 2.3.

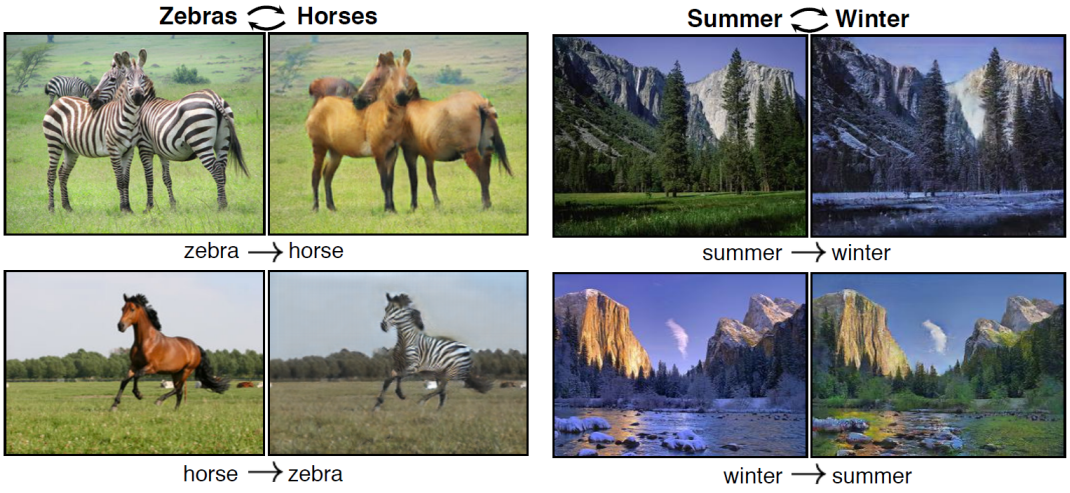


Figure 2.4: An illustration of CycleGAN transforming an image from one into the other and vice versa. For example, zebra image transformed into horse image, and vice versa. The view of Yosemite mountains in summer transformed into winter, and vice versa[20].

Jun-Yan Zhu et al.[20] noticed the scarcity of paired training data to perform image-to-image translation. Hence, they proposed CycleGAN as an image-to-image translation method that can transform the images from source domain X to target domain Y in the absence of paired training dataset. "This method learns to capture underlying characteristics of the collection of images in the source domain and figuring out how these characteristics could be transformed into the collection of images in the target domain, all in the absence of any paired training examples"[20]. In this method, "the goal is to learn a mapping $G : X \rightarrow Y$ so that the generated distribution of images $G(X)$ matches the target distribution Y using an adversarial loss. However, the adversarial losses alone are not sufficient to map the images from the source domain to the target domain. Hence, authors introduced inverse mapping $F : Y \rightarrow X$ to introduce a cycle-consistency loss to enforce $F(G(X)) \approx X$, and vice versa to regularize the output and keep it highly under-constrained"[20]. "The cycle-consistency property of an image-to-image translation can be understood by a simple example, if we translate a sentence of English to French, again back from French to English, one should arrive back to the original sentence. In this manner, CycleGAN learns to produce images within the context instead of generating random images"[2]. Authors investigated several prior methods like Bi-GAN/ALI[56][57], CoGAN[58], SimGAN[59] 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[40]. 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.

Chris Tensmeyer et al.[10] realized solving binarization tasks using deep learning models is very challenging. It is due to the scarcity of large quantities of labeled data available to train such models. There have been efforts to create synthetic data for binarization using image processing

[1]Peak Signal-to-Noise Ratio: `http://www.ni.com/white-paper/13306/en/` last access: September 30, 2021.
[2]`https://machinelearningmastery.com/what-is-cyclegan/` last access: September 30, 2021

techniques but, they generally lack realism[10]. Hence, authors proposed DGT-CycleGAN to produce realistic synthetic data using an adversarially trained image-to-image translation model[10]. Authors modified "the CycleGAN model to be conditioned on the ground truth binarization mask as it transforms images from the source domain of synthetic images to the target domain of real images"[10]. They found 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[10]. Authors pretrained deep learning models on realistic synthetic datasets generated by CycleGAN, DGT-CycleGAN, and image compositing[10]. They found pretraining a model on realistic synthetic data generated by the DGT-CycleGAN model achieves the best performance for solving binarization tasks compared to data generated by CycleGAN and image compositing[10]. Also, they evaluated "both pretrained only and finetuned models on each of the Document Image Binarization Competition (DIBCO) datasets[3]to measure their performance"[10]. "Finetuning a model on DIBCO dataset after pretraining on the realistic synthetic images generated by the DGT-CycleGAN has lead to a 13% reduction of F-measure error compared to no pretraining and to 6% error reduction compared to pretraining on non-realistic synthetic data"[10]. Authors concluded, pretraining deep neural networks on the realistic synthetic data generated using DGT-CycleGAN lead to better predictive performance both before and after finetuning on real data.

2.2 Summary

Over the years, GANs have evolved to solve different kinds of problems(figure 2.1). General GANs suffer from unstable training, vanishing gradients problem, and mode collapse. Therefore, Martin Arjovsky et al.[60] proposed Wasserstein GANs (WGANs) to solve problems with GANs. WGANs solve problems like mode collapse, improve learning stability, and provide meaningful learning curves beneficial for hyperparameter searches and debugging[60]. Although the implementation of WGAN is straightforward, the theory behind it is difficult and requires modifications, for example, implementation of Weight Clipping[61]. Hence, Xudong Mao et al.[34] proposed a simple and more intuitive method compared to WGAN, called LSGAN. First, LSGANs generate higher quality images than GANs[34]. Second, its learning process is stable compared to GANs[34]. Monika Sharma et al.[23] demonstrated CycleGAN can transform noisy document images into denoised and clean document images. They demonstrated CycleGAN transforms noisy document images into clean document images superior to cGAN. Chris Tensmeyer et al.[10] 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. Taesung Park et al.[47] proposed CUT. They have demonstrated that CUT is a better, faster, and memory-efficient approach to perform unsupervised image-to-image translation. Further, Jun-Yan Zhu et al.[20] proposed an unsupervised image-to-image translation approach CycleGAN to address the problem of scarcity of paired training data while training image-to-image translation applications. Compared to GANs, CycleGANs also can deal more meticulously with the problems like unstable training, vanishing gradients problems, and mode collapse. In this thesis, the image-to-image translation model is developed using LSGAN and CycleGAN. The generators and discriminators are optimized using cycle-consistency loss[20] and least-square loss[34]. The proposed method is simple and capable to solve problems in general GANs. Also, it learns a function of image-to-image translation in the absence of paired training data using cycle-consistency property[20].

[3]DIBCO 2019: `https://vc.ee.duth.gr/dibco2019/` last access: September 30, 2021

3. Fundamentals

This chapter intends to develop a better understanding of fundamental concepts required to understand this thesis. Section 3.1 describes architecture of GANs. Section 3.2 describes neural network layers, namely, convolution layer, activation layer, pooling layer, and fully connected layer.

3.1 Generative Adversarial Networks (GANs)

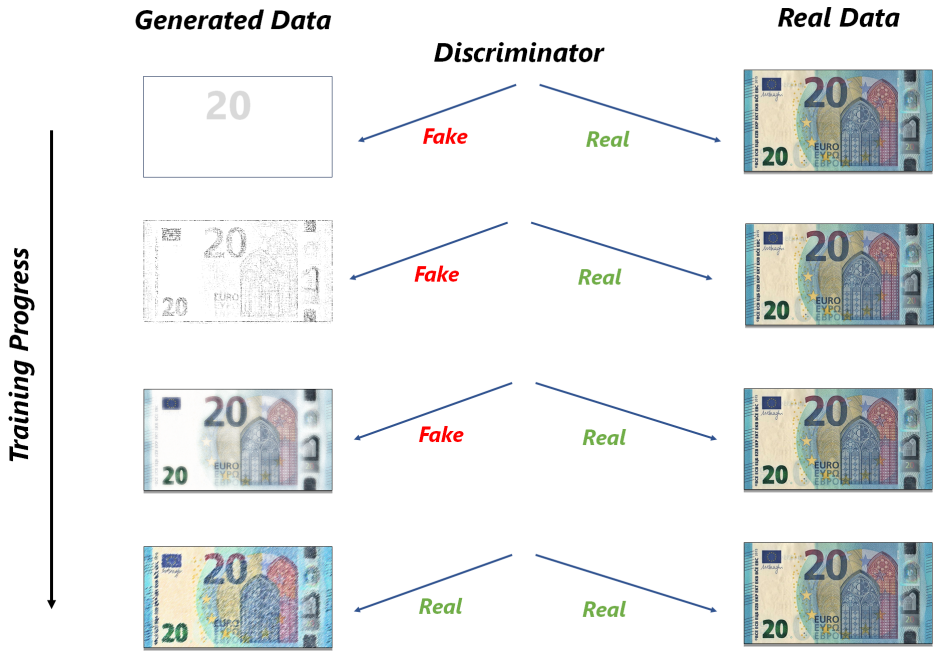


Figure 3.1: An intuitive example of GAN training progress.

GANs have two models in their architecture first is the generator, and the second is the discriminator, and both models can be implemented using multilayer perceptrons[21], the multilayer perceptron can loosely be indicated by ANNs. Now, 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 given after misclassifying real images as fake and fake images as real. The generator learns by the error signal given by the discriminator when the discriminator successfully classifies images produced by the generator as fake. To produce a better quality of forged or fake images, the generator model is updated using this error signal given by the discriminator. 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 illustrated. In the initial stages of the training, the images generated by the generator are easily distinguished by the discriminator as fake images. After a certain number of iterations, the quality of the images generated by the generator increases by updating itself using the feedback error signal given by the discriminator and finally it classifies them as real.

Now, let's have a look into the GAN's architecture. As described earlier, the task of the GAN is to generate fake samples. The training dataset of real data samples serves as an 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. GAN in action is illustrated in figure 3.2.

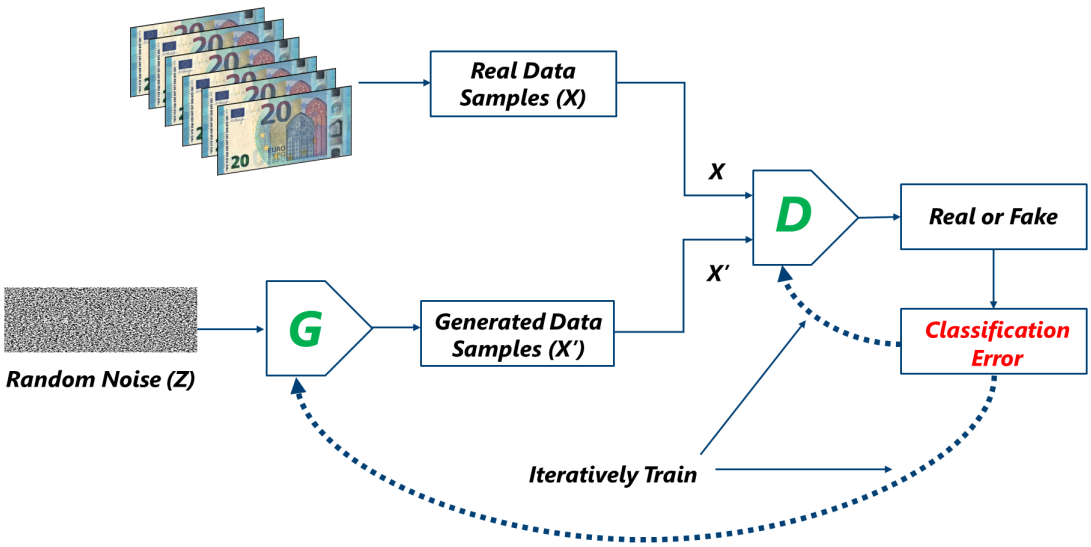


Figure 3.2: An illustration of GAN architecture along with the generator's and discriminator's inputs, outputs, and their interactions.

Now let's try to understand the mathematics behind 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 perceptrons with parameters θ_g and θ_d respectively. The mapping function 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. For a fixed 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 a certain number of training iterations, when the discriminator is optimal, it can be frozen and the generator G will be continue to learn by the error provided by discriminator to produce plausible data. At the end of the 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[62]. Mathematically 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[21]. Hence, training G to maximize $logD(G(z))$ rather than training G to minimize $log(1 - D(G(z)))$. "This objective function leads to the same fixed point of the dynamics of G and D by providing much stronger gradients early in learning"[21].

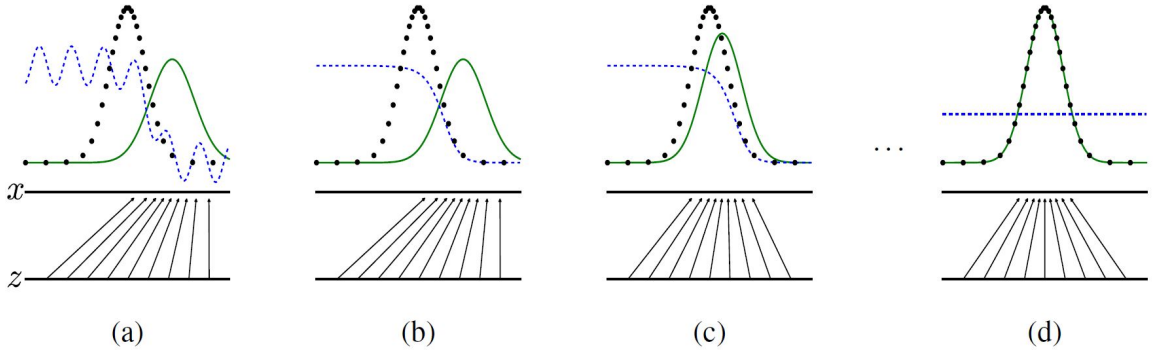


Figure 3.3: An illustration of GANs converging to match generated data distribution p_g to real data distribution p_{data}[21].

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 of 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, G and D are on the verge of convergence, p_g is almost similar to p_{data} and D is a partially accurate classifier. (b) Slowly, D trained further to distinguish real data samples from generated data samples. For a fixed G, there is an optimal discriminator, $D^*(x) = \frac{p_{data}(x)}{p_{data}(x)+p_g(x)}$. (c) The G is updated using gradients that are backpropagated from D and makes $G(z)$ to flow to regions that are more likely to be classified as real data, the parameters of the G are updated, while the parameters of the D are fixed. (d) After several steps of training, G is optimal where $p_g = p_{data}$, which is equivalent to the optimal D predicting 0.5 for all samples drawn from x, at this stage D is maximally confused and will not able to distinguish real data samples from generated data samples, i.e. $D(x) = \frac{1}{2}$[21].

3.1.1 GAN Training

This section discusses the algorithm of the GAN. In the figure A.4, the algorithm of the GAN is pictorially illustrated. The GAN training algorithm is divided into two sections. These two sections are discriminator training and generator training. In figure A.4 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 in the section where the generator training is illustrated. While training discriminator both generated samples from the generator and real samples are used as an input. In the following sections generator's and discriminator's architecture and their training process is described. Further, the GAN training algorithm has been described in the form of pseudo code.

Discriminator Architecture

In GANs the discriminator 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.

In computer vision tasks, the discriminator classifies images, hence, it is implemented using CNN[37]. 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. 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.

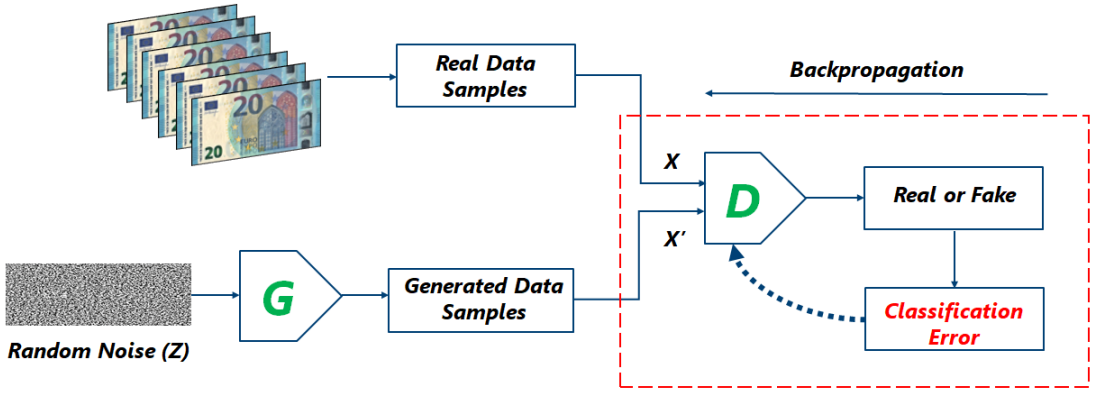


Figure 3.4: An illustration of the training of the discriminator D using backpropagation.

Generator Architecture

The generator produces fake samples using random noise vectors sampled from the uniform random noise distribution. In computer vision tasks, the generator produces fake images, hence, it is implemented using CNN[37]. 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 samples 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.

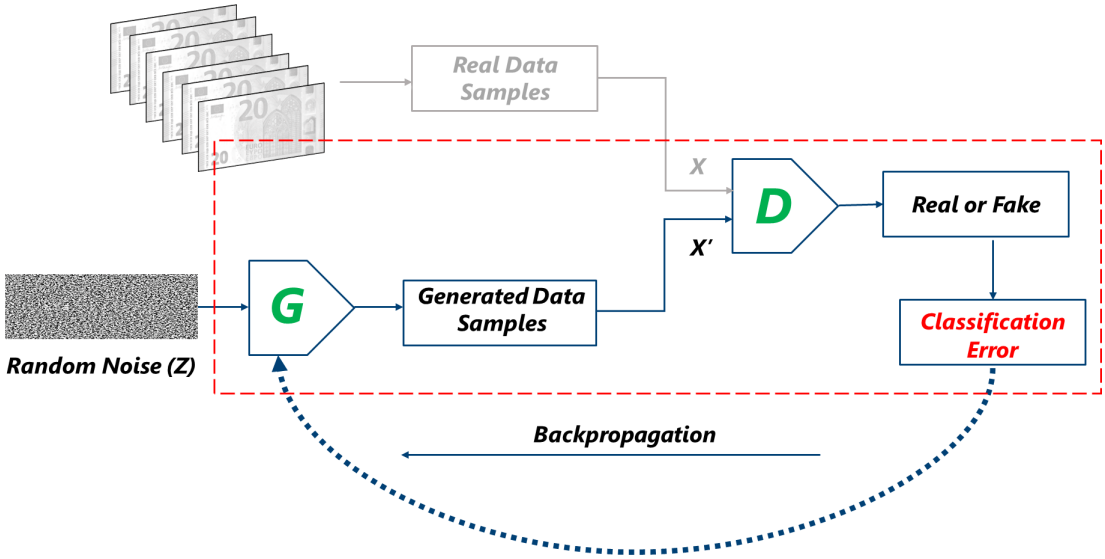


Figure 3.5: An illustration of the training of the generator G using backpropagation.

GAN Training Algorithm

for *each training iteration* **do**

1. Train the Discriminator:

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

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

c) Classify X and X' using the discriminator D.

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

2. Train the Generator:

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

b) Classify X' using the discriminator D.

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 deep learning is evolved with the rise of ANNs. ANNs are biologically inspired and able to solve complex problems[63]. CNNs is the most successful image-driven pattern recognition technique among ANNs[63]. 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 and videos. CNNs are inspired by the early findings in the study of biological vision, especially, organization of the animal visual cortex[64][65][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, and they perform feature extraction. Whereas, the third, a fully connected layer, maps the extracted features into the final output, such as classification[66]. Let's have a look at each layer in the following sections.

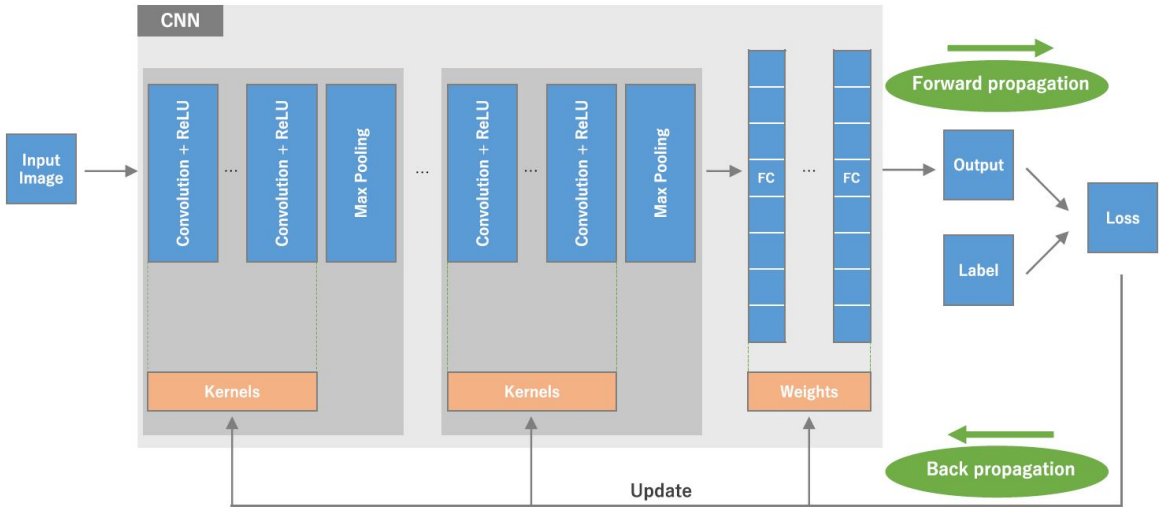


Figure 3.6: An illustration of CNN architecture and its training process. CNN is a combination of several building blocks like convolution layers, pooling layers, and fully connected layers. These building blocks are stacked upon each other. The CNN is trained using the training dataset, the input data 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 Rectified Linear Unit (ReLU)(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[67].

3.2.1 Convolution Layer

The convolution layer is a basic component of the CNN 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 by performing mathematical 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, which makes CNNs highly efficient for image processing, feature extraction, since a feature could occur at any location in the image[66]. The layers are connected, one layer feeds its output to the next layer. The extracted features can hierarchically and progressively become more complex[66].

Training is the process of optimizing parameters such as kernels which minimize the difference between outputs and ground truth labels through an optimization algorithm backpropagation[68] and gradient descent[67][66].

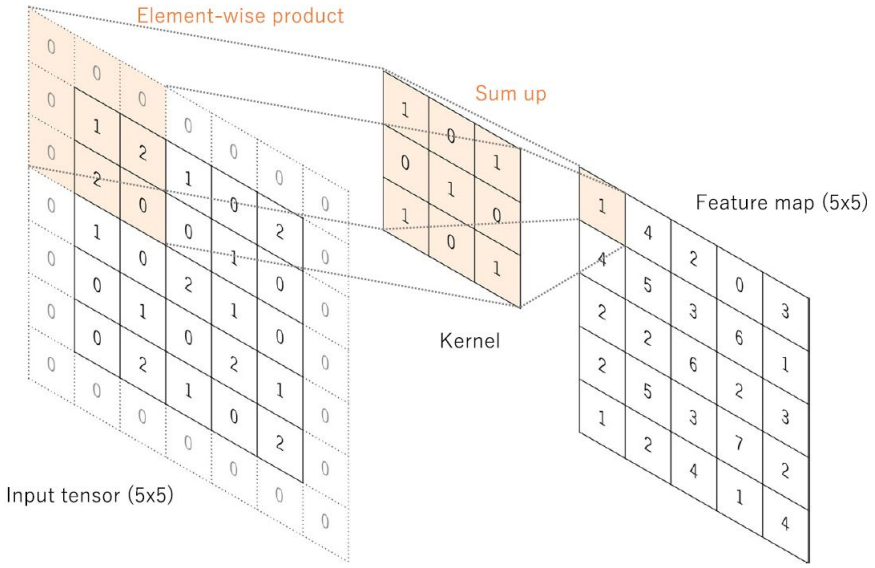


Figure 3.7: Illustration of a convolution operation with zero padding to retain spatial dimensions. Note that an input dimension of 5×5 is retained in the generated output feature map. In this example, kernel size is set to 3×3, and stride is set to 1[66].

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)[66]. This procedure is repeated by applying multiple kernels to create an arbitrary number of feature maps, which describe different characteristics of the input tensors[66]. 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. Hence, 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 is an alternative technique for downsampling. It is described briefly in section 3.2.2.

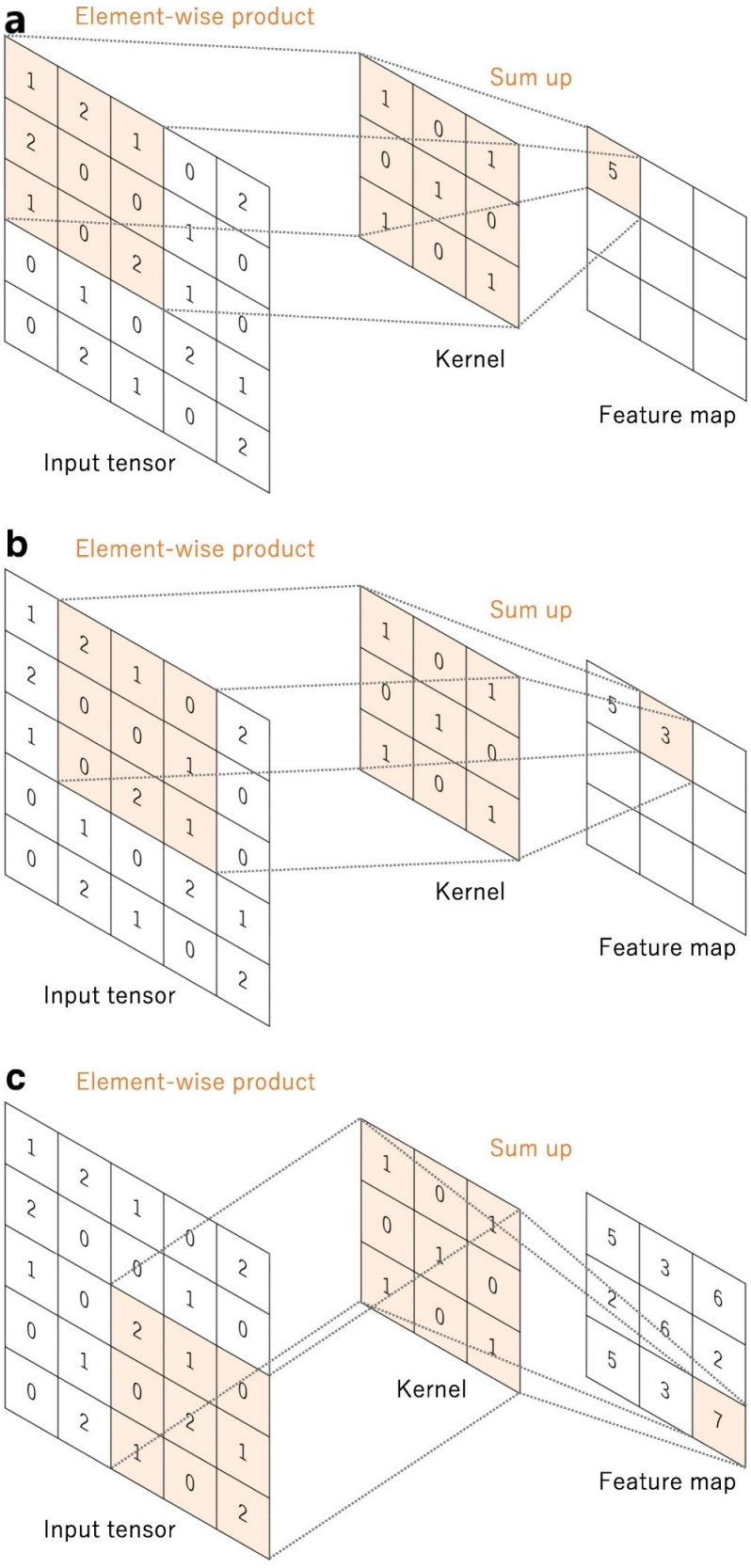


Figure 3.8: a–c An illustration of convolution operation with no padding. In this example, kernel size is set to 3×3, and stride is set to 1. A kernel is applied across the input tensor, and an element-wise product between each element of the kernel and the input tensor is calculated at each location and summed to obtain the output value in the corresponding position of the output tensor, called a feature map[66].

Activation Functions

The artificial neuron is a mathematical function inspired by the neurons inside the brain, and they are the basis of ANNs. The artificial neurons receive one or more weighted inputs and bias, and their sum is passed through an activation function. Schematic diagram of an artificial neuron is illustrated in the figure 3.3, 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, and b is the bias. The summation symbol represents the addition unit. Inputs are fed to the neuron, they perform a linear transformation on these inputs using the weights and biases. 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

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

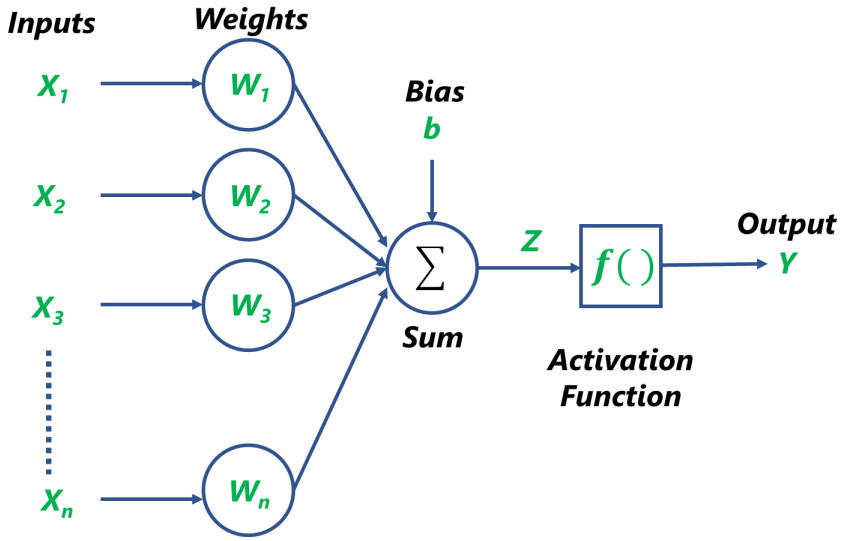


Figure 3.9: An illustration of artificial neuron.

The function f is the activation function applied on Z.

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

The data is fed to the input layer of the neurons in neural network, undergoes the linear operation, later, activation functions are applied in the hidden layers, and output is produced. The produced output the propagated forward to next hidden layers, this is movement is called forward propagation[1]. In neural networks the hidden layer lies between input layer and output layer. A simple neural network is illustrated in figure 3.10. The main purpose of the activation function is to introduce non-linearity in the neural network. The 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. The neural networks learn from the errors calculated at the output layer, a differentiable nonlinear activation function is required to perform backpropagation to compute loss with respect to the weights of the neurons and optimize them using gradient descent optimization algorithm or any other optimization algorithm.

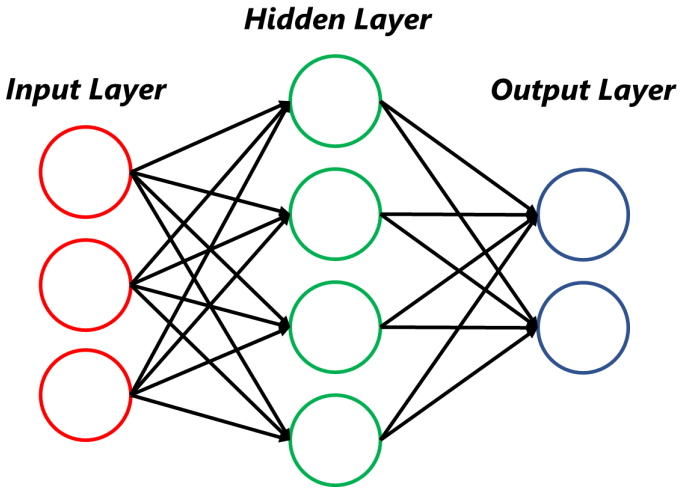


Figure 3.10: Simple neural network

The backpropagation minimizes the error, updates the weights of neurons, and enhances the accuracy and performance of the neural network. The nonlinear functions, sigmoid, hyperbolic tangent(tanh) functions ware mathematical representations of biological neuron behavior. The ReLU is now the most commonly used nonlinear activation function, it computes the function: $f(x) = \max(0, x)$. These activation functions along with their plots are illustrated in figure 3.11[69][70].

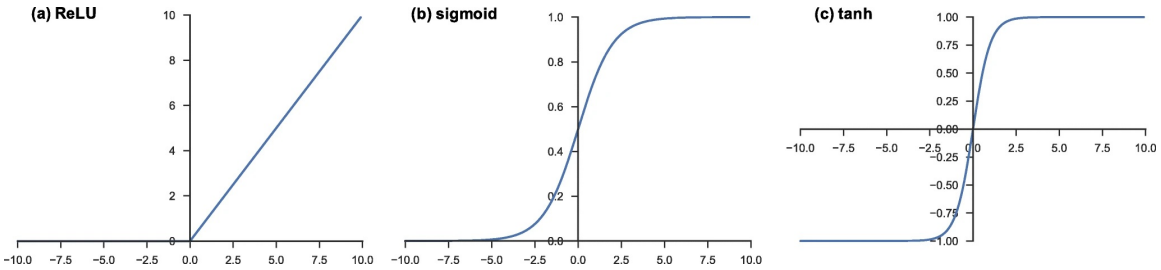


Figure 3.11: Most common nonlinear activation functions used while constructing neural networks: **a)** ReLU, **b)** sigmoid, and **c)** hyperbolic tangent (tanh)[66].

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[2]. 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[2]. Similar to convolution operations, it uses 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 in each input patch of the feature map. The average pooling averages values in each input patch of the feature map[71].

[1] `https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/` last access: September 30, 2021

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, max-pooling filters are of size 2×2 and 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)[2]. The spatial dimension of feature maps is reduced by a factor of two by discarding 75% of its activations[72]. 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.

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[66][73]. c) It is designed to replace fully connected layers, the idea is to generate one feature map for each class, in the classification task, as they provide robust spatial information, and the resulting vector can easily be fed to the softmax layer[2].

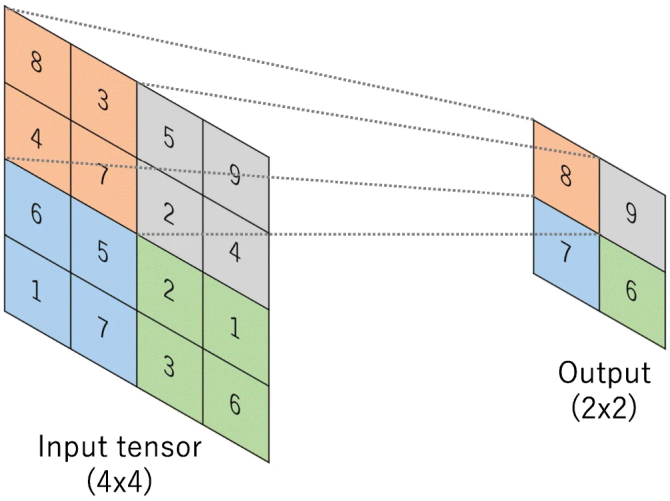


Figure 3.12: Illustration of max pooling operation with a filter size of 2×2. No padding, and a stride of 2. Max pooling operation extracts 2×2 patches from the input tensors, outputs the maximum value in each patch, and discards the rest of the other values, which results in the spatial dimension of an input tensor downsampled by a factor of 2[66].

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. 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 then adds a bias vector. Equation 3.3 represents the function of each neuron in a fully connected layer. Each fully connected layer is followed by a nonlinear activation function, for example, ReLU. As already

[2] `https://paperswithcode.com/method/global-average-pooling/` last access: September 30, 2021

described, the nonlinear activation function helps neurons learn complex patterns. 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. Often the number of output nodes in the final fully connected layer equals the number of classes. The last 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[66].

4. Methodology

This chapter describes the unpaired image-to-image translation method CycleGAN. In this thesis, CycleGAN is used to implement an image-to-image translation application. This application transforms synthetic document images into realistic document images. The application model is optimized using cycle-consistency loss, least-square loss, and identity mapping loss. These loss functions are described in this chapter. Section 4.2 presents the algorithm of CycleGAN.

4.1 Cycle-Consistent Adversarial Networks

In CycleGAN aim is to learn the mapping function between two domains X and Y, and vice versa. The distribution of synthetic document images represents source domain X and the distribution of real document images represents target domain Y. The synthetic document images created using empty form templates and handwritten crops(figure 5.2). For a given training samples $\{x_i\}_{i=1}^{N}$ where $x_i \in X$ and $\{y_j\}_{j=1}^{M}$ where $y_j \in Y$. The synthetic data distribution represented as $x \sim p_{data}(x)$ and real data distribution represented as $y \sim p_{data}(y)$. The model includes two mappings functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$ as illustrated in figure 4.2, they are the generators. Along with generators, two adversarial discriminators D_X and D_Y are introduced. D_X learns to distinguish images $\{x\}$ from translated images $\{F(y)\}$. In the same way, D_Y learns to distinguish images $\{y\}$ from $\{G(x)\}$. Now let's try to understand the architecture of CycleGAN using pictorial representation.

The architecture of the CycleGAN is illustrated in figure 4.1. Figure (a) depicts the flow of transformation from synthetic document images to real document images. Figure (b) depicts the flow of transformation from real document images to synthetic document images. Figure 4.1 illustrates CycleGAN has two generators G and F, two discriminators D_X and D_Y. (a) The discriminator D_Y takes input *Input_Y* from the target domain Y and fake image *Generated_Y* generated by the generator G. The discriminator D_Y learns to distinguish target domain images from the fake images generated by the G, and penalizes itself in the event of misclassification. When the discriminator D_Y determines the image generated by the generator G is fake. It gives feedback to generator G through backpropagation to update its weights to produce good quality images. Next, to achieve cycle-consistency the fake image *Generated_Y* passed thorough generator F to produce image *Cyclic_X* back in the source domain X, and the cycle-consistency loss is minimized between *Input_X* and *Cyclic_X*. (b) The discriminator D_X takes input *Input_X* from the source domain X and fake image *Generated_X* generated by the generator F. The discriminator D_X learns to distinguish source domain images from fake images generated by the generator F, and penalizes itself in the event of misclassification. When the discriminator D_X determines the image generated by generator F is fake. It gives feedback to generator F through backpropagation to update its weights to produce good quality images. Next, to achieve cycle-consistency the fake image *Generated_X* passed thorough generator G to produce image *Cyclic_Y* back in the target domain Y, and the cycle-consistency loss is minimized between *Input_Y* and *Cyclic_Y*. The transformation flow of synthetic document images into real document images, and vice versa (marked in green dotted arrows) by minimizing the cycle-consistency loss (marked in green dotted arrows) between the input image and cyclically generated image.

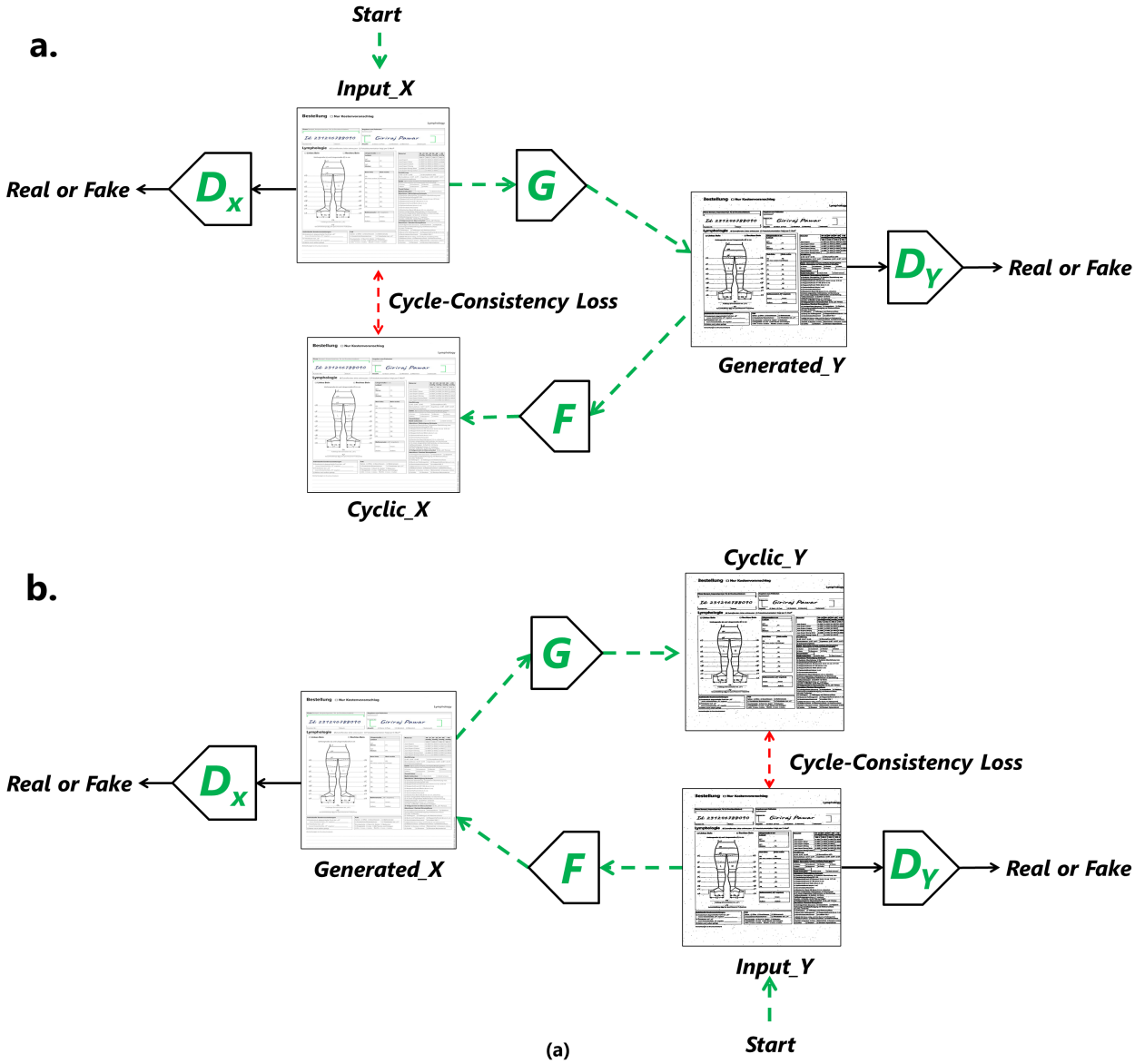


Figure 4.1: An illustration of the proposed image-to-image translation application using CycleGAN for transforming synthetic document images into real document images, and vice versa. It consists of two generators, G and F which map synthetic document images to real document images and real document images to synthetic document images, respectively minimizing cycle-consistency loss[20]. It also contains two discriminators D_X and D_Y which acts as the adversary and reject images generated by generators.

4.1.1 Least-Square Loss

In GANs the discriminator is a binary classifier that adopts the sigmoid cross-entropy loss function. As discussed in chapter 2, while updating the generator, the sigmoid cross-entropy loss function causes the vanishing gradients problem for the samples that are on the correct side of the decision boundary but are still far from the real data. Also, the sigmoid cross-entropy loss function causes difficulty to stabilize the model training procedure[34]. First, $\mathcal{L}_{GAN}$ (equation 3.1), the negative log-likelihood objective replaced by a least-squares loss functions[34]. The least-squares loss is more stable during training and generates higher quality results[34]. It is used to optimize the generator and discriminator adversarially. In the below equations, a, b, and c are the coding scheme for the equations of the discriminator. Where a and b are the labels for fake data and real data, respectively. c denotes the value that G wants D to believe for fake data. Basically, $a = 0$, $b = 1$, and $c = 1$. That means fake data is represented by 0, and real data is represented by 1. Then the improved objective

functions using least-squares loss can be defined as follows:

$$\min_{D} \mathcal{L}_{ls}(D_Y) = \frac{1}{2}\ \mathbb{E}_{y\sim p_{data}(y)}\ (D_Y(y)-b)^2 + \frac{1}{2}\ \mathbb{E}_{x\sim p_{data}(x)}\ (D_Y(G(x))-a)^2 \tag{4.1}$$

$$\min_{G} \mathcal{L}_{ls}(G) = \mathbb{E}_{x\sim p_{data}(x)}\ (D_Y(G(x))-c)^2] \tag{4.2}$$

$$\mathcal{L}_{ls}(G, D_Y, X, Y) = \min_{D} \mathcal{L}_{ls}(D_Y) + \min_{G} \mathcal{L}_{ls}(G) \tag{4.3}$$

$$\min_{D} \mathcal{L}_{ls}(D_X) = \frac{1}{2}\ \mathbb{E}_{x\sim p_{data}(x)}\ (D_X(x)-b)^2 + \frac{1}{2}\ \mathbb{E}_{y\sim p_{data}(y)}\ (D_X(F(y))-a)^2 \tag{4.4}$$

$$\min_{F} \mathcal{L}_{ls}(F) = \mathbb{E}_{y\sim p_{data}(y)}\ (D_X(F(y))-c)^2 \tag{4.5}$$

$$\mathcal{L}_{ls}(F, D_X, Y, X) = \min_{D} \mathcal{L}_{ls}(D_X) + \min_{F} \mathcal{L}_{ls}(F) \tag{4.6}$$

4.1.2 Cycle-Consistency Loss

The cycle-consistency loss function is the basis of CycleGAN. Theoretically, adversarial training can learn mappings that produce outputs identically distributed as target domains Y and X, respectively[74]. However, with a large amount of capacity, a neural network maps the same set of input images to any random permutations of images in the target domain, where any of the learned mappings can produce an output distribution that matches the target distribution[74]. Hence, depending just on adversarial losses alone cannot promise that the learned mapping function can map an individual input from the source domain x_i to a desired output y_j in the target domain. To further narrow down and reduce the space of possible outcomes using mapping functions, attempt has been made to argue that learned mapping functions should be cycle-consistent. The idea of being cycle-consistent can be understood by a simple example for translation a sentence in languages. For example, if an English sentence is translated into German and later from German to English, the outcome should be an original English sentence from where we have started.

The figure 4.2, (a) CycleGAN model with two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and respective adversarial discriminators D_Y and D_X is illustrated. (b) for each image x from source domain X, the image translation cycle should be able to bring x back to the original image, i.e., $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$. This is called forward cycle-consistency. (c) for each image y from target domain Y, the image translation cycle should be able to bring y back to the original image, i.e., $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$. This is called backward cycle-consistency. G and F both should satisfy cycle-consistency. The cycle-consistency loss in the form of a mathematical equation:

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

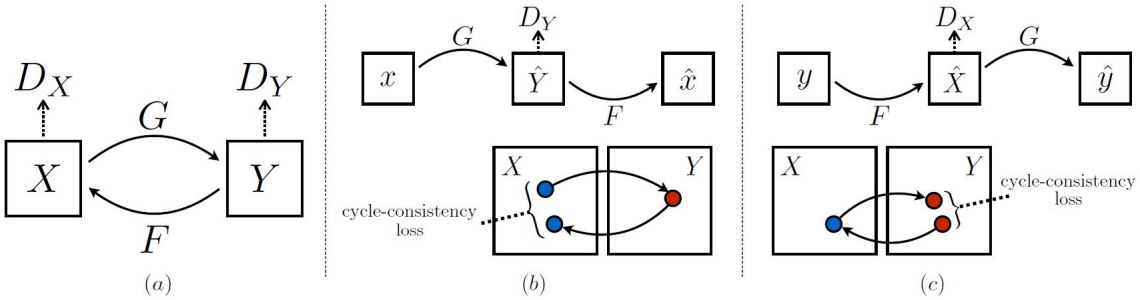


Figure 4.2: An illustration of CycleGAN model with its mapping functions, respective discriminators, and forward and backward consistency loss[20].

4.1.3 Identity Mapping Loss

The identity mapping loss encourages the mapping to preserve color composition between the input and output[75]. It regularizes the generator G to be near an identity mapping when real samples of the target domain are provided as the input to the generator G. Also, the identity mapping loss regularizes the generator F to be near an identity mapping when synthetic samples of the source domain are provided as the input to the generator F. It means provided an input to the generator from the same domain in which it is transforming into, the output should be identical to the input. The identity mapping loss in the form of a mathematical equation:

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

4.1.4 Complete Objective Function

Combining equations 4.6, 4.7, and 4.8 the final objective is achieved. The model is optimized using least-squares loss, cycle-consistency loss, and identity mapping loss. In equation 4.9, the terms λ_{cyc} and $\lambda_{identity}$ control the relative importance of the term in the objective. The complete objective function in the form of a mathematical equation:

$$\mathcal{L}(G,F,D_X,D_Y) = \mathcal{L}_{ls}(G,D_Y,X,Y) \;+\; \mathcal{L}_{ls}(F,D_X,Y,X) \;+\; \lambda_{ide}\;\mathcal{L}_{ide}(G,F) \;+\; \lambda_{cyc}\;\mathcal{L}_{cyc}(G,F). \tag{4.9}$$

4.2 CycleGAN Training Algorithm

for *each training iteration* **do**

Draw a random sample x_i from source domain X where $\{x_i\}_{i=1}^{N}$ and $x_i \in X$

Draw a random sample y_j from target domain Y where $\{y_j\}_{j=1}^{M}$ and $y_j \in Y$

1. Train the Discriminators:

a) Compute the discriminator loss on real and fake images:

$$\mathcal{L}_{real,fake}^{(D_X)} = \frac{1}{2}\,(D_X(x_i)-1)^2 + \frac{1}{2}\,(D_X(F(y_j)))^2$$

$$\mathcal{L}_{real,fake}^{(D_Y)} = \frac{1}{2}\,(D_Y(y_j)-1)^2 + \frac{1}{2}\,(D_Y(G(x_i)))^2$$

c) Update the discriminators

2. Train the Generators:

a) Compute the cycle-consistency losses:

$$\mathcal{L}_{cyc}^{(X\to Y\to X)} = F(G(x_i)) - x_i$$

$$\mathcal{L}_{cyc}^{(Y\to X\to Y)} = G(F(y_j)) - y_j$$

b) Compute the identity mapping losses:

$$\mathcal{L}_{ide}^{(X\to X)} = G(x_i) - x_i$$

$$\mathcal{L}_{ide}^{(Y\to Y)} = F(y_j) - y_j$$

c) Compute the Generator G loss:

$$\mathcal{L}_G = (D_Y(G(x_i))-1)^2 + \mathcal{L}_{cyc}^{(Y\to X\to Y)} + \mathcal{L}_{ide}^{(Y\to Y)}$$

Compute the Generator F loss:

$$\mathcal{L}_F = (D_X(F(y_j))-1)^2 + \mathcal{L}_{cyc}^{(X\to Y\to X)} + \mathcal{L}_{ide}^{(X\to X)}$$

d) Update the generators

end

Algorithm 2: CycleGAN training algorithm.

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 and to evaluate the quality of images generated by the CycleGAN. How the classifiers are used to analyze the domain gap between distributions will be discussed in chapter Evaluation 6. In section 5.1 dataset preparation is described. The architecture of CycleGAN and classifiers discussed in section 5.2. The training details of CycleGAN and classifiers described in 5.3. 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[76]. 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.

5.1 Dataset Preparation

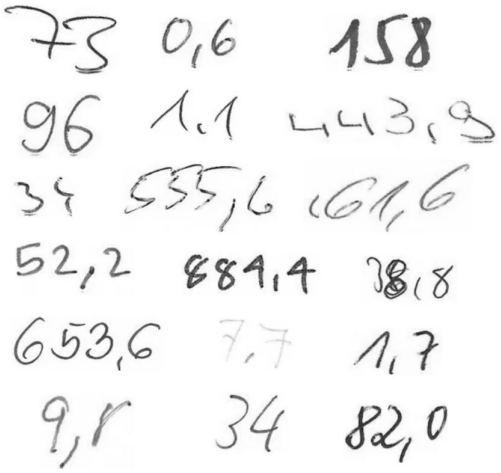


Figure 5.1: Examples of handwritting crops from the handwritting number dataset (figure reproduced from elevait GmbH & Co. KG with permission).

The dataset preparation is one of the vital step before training any neural network. Bad quality data leads to a poor generalization of neural networks. Ten types of documents that were considered to work with this image-to-image translation application. The CycleGAN is trained using a stash of synthetic document images and real document images stored in a memory. There are 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 templates (figure A.8) and handwritten crops (figure 5.1). The process of inserting handwritten crops on empty templates can be visualized in figure 5.2. Each template is filled with the help of provided bounding box annotations[77]. For each class of template 10,000, synthetic document images are created. The dataset of 100,000 synthetic document images is formed. These synthetic document images are used while training CycleGAN, just they are stashed at the single location collectively. The created 100,000 synthetic document images were also faxified and a faxified dataset of 100,000 images is created. It has 10 classes, and each class has 10,000 images, similar to the synthetic document images dataset. The faxification process uses several image transformations to make a clean gray-

[1] `https://www.tensorflow.org/tensorboard` last access: September 30, 2021

scale image look like it was sent via fax. A sample faxified image can be seen in figure A.10. The faxification process is described briefly in section 6.2.3. Also, the faxification process 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.

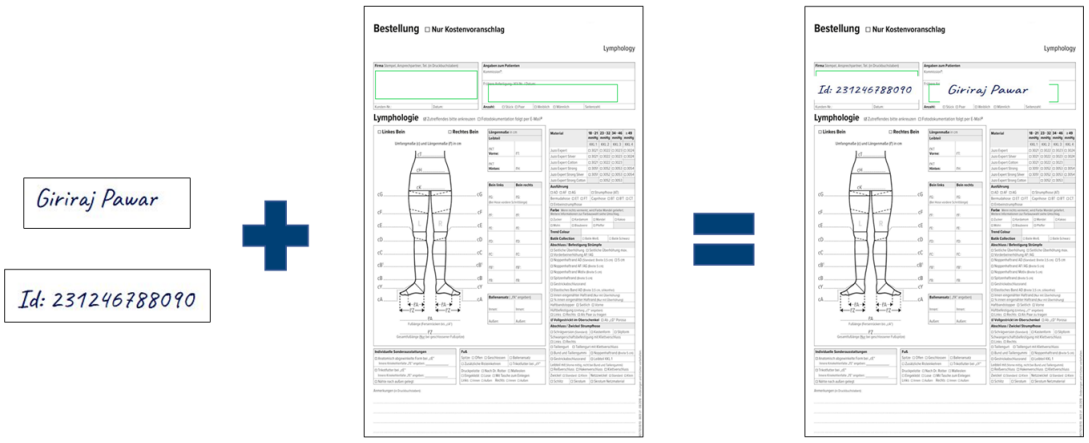


Figure 5.2: Inserting handwritten crops on empty form templates (figure reproduced from elevait GmbH & Co. KG with permission).

The templates are unfilled form images. As described before, in this thesis, ten classes of templates are chosen for creating document images datasets like synthetic and faxified document images datasets. The 8 out of 10 templates are a type of "Bein". These templates are very similar to each other apart from minor differences. Examples of "Bein" templates are shown in figure 5.3. The other two classes of templates are of type "Hand" and "Arm". In figure 5.4, (b) is the template of "Bein", (a) is the template of "Arm", and (c) is the template of "Hand". The list of classes or templates is listed in the table 5.2.

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.

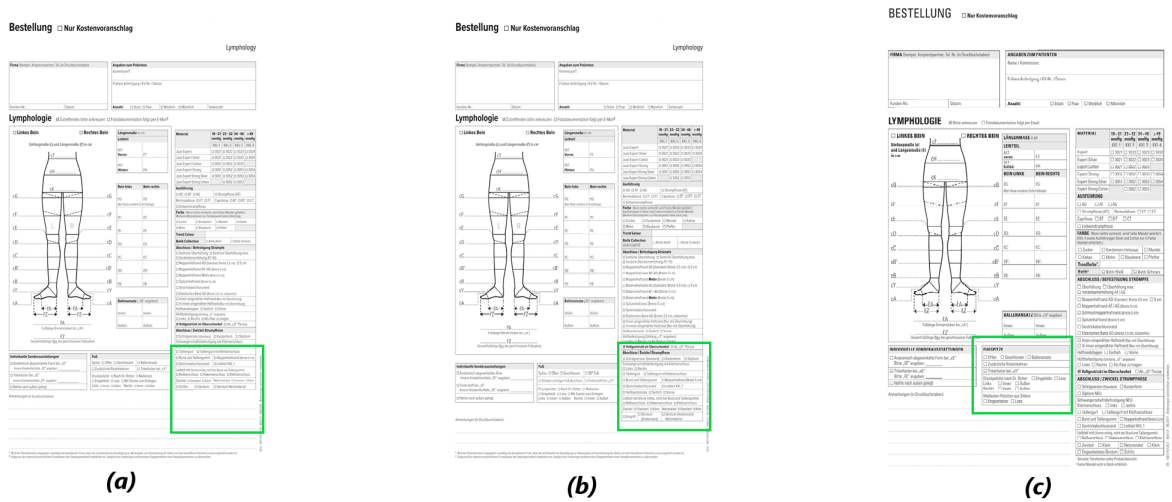


Figure 5.3: Templates with minor differences (figure reproduced from elevait GmbH & Co. KG with permission).

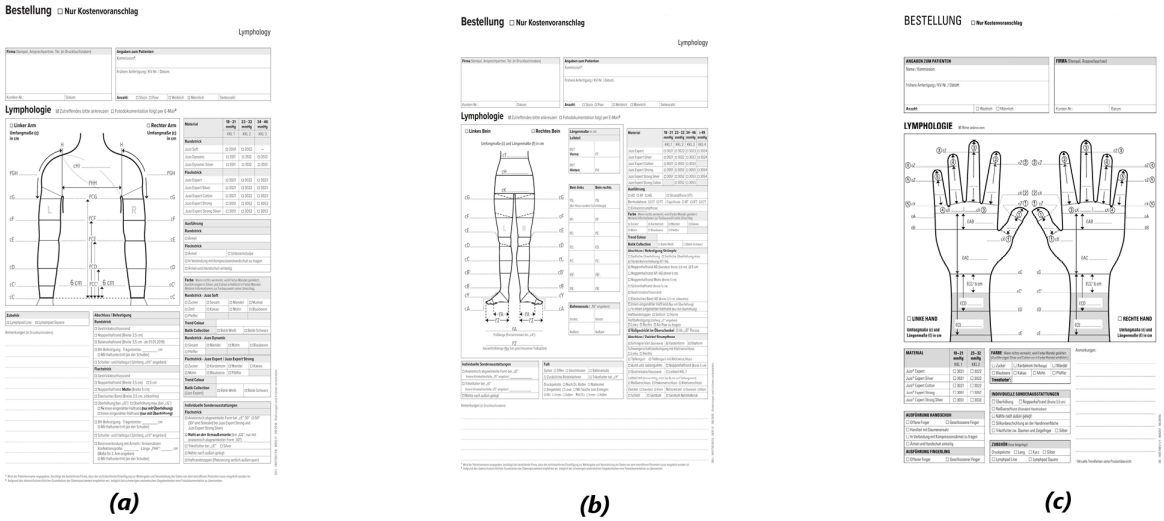


Figure 5.4: Templates with major differences (figure reproduced from elevait GmbH & Co. KG with permission).

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 (testing dataset).

5.2 Network Architecture

5.2.1 CycleGAN

Johnson et al.[18] 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 2 upsampling convolutional blocks to generate the output image of the same dimension as the input image. The reason behind using residual blocks is, they resolve the vanishing gradient problem in deep neural networks. The discriminator classifier network is designed using PatchGAN architecture[40][46]. 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.[18] 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` is 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 convolution 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 described 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.12. The architecture of the generator is illustrated in table 5.3. Also, in the figure 5.5, the architecture of ResNet block in the generator network is illustrated. In which, at every ResNet block, the output from the previous layer is passed through two convolution layers(each with `c3s1-256`). Further, it concatenated with the convolution layer's output and forwarded to the following layers.

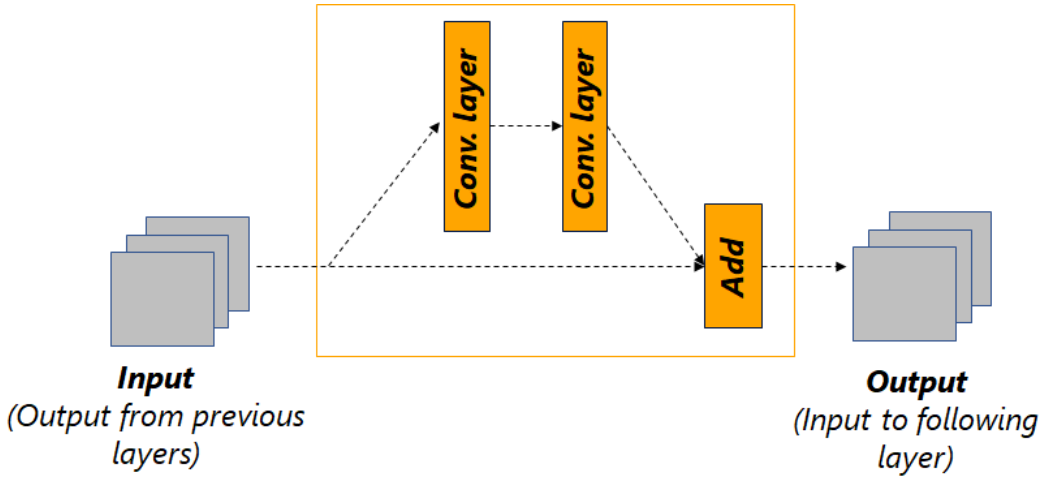


Figure 5.5: Illustration of ResNet blocks in CycleGAN generator architecture.

Operation Layer	Number of Filters/Units	Size of Each Filter	Stride Value
Input Image $(256 \times 256 \times 1)$	-	-	-
Convolution Layer Instance Normalization ReLU	64	7×7	1×1
Convolution Layer Instance Normalization ReLU	128	3×3	2×2
Convolution Layer Instance Normalization ReLU	256	3×3	2×2
9 Residual Blocks 2 Convolution Layers *(each with)* Instance Normalization ReLU	256	3×3	1×1
Transposed Convolution Layer Instance Normalization ReLU	128	3×3	2×2
Transposed Convolution Layer Instance Normalization ReLU	64	3×3	2×2
Convolution Layer tanh	1	7×7	1×1
Output $(256 \times 256 \times 1)$	-	-	-

Table 5.3: Generator architecture

The discriminator uses 70×70 PatchGAN classfier architecture[40]. It is also called a Markovian discriminator[46]. The L1 and L2 loss functions produce blurry results while solving image generation problems[78]. These losses fail to encourage high-frequency crispness[78]. To model high frequencies, more attention is given to the structure in local image patches[40]. Therefore Markovian discriminator is called the PatchGAN discriminator. `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-C512-C1`. All of the layers in the discriminator can be seen in figure A.18. The architecture of the classifier is illustrated in table 5.4.

Operation Layer	Number of Filters/Units	Size of Each Filter	Stride Value
Input Image $(256 \times 256 \times 1)$	-	-	-
Convolution Layer Instance Normalization LeakyReLU (0.2)	64	4×4	2×2
Convolution Layer Instance Normalization LeakyReLU (0.2)	128	4×4	2×2
Convolution Layer Instance Normalization LeakyReLU (0.2)	256	4×4	2×2
Convolution Layer Instance Normalization LeakyReLU (0.2)	512	4×4	1×1
Convolution Layer Instance Normalization LeakyReLU (0.2)	512	4×4	1×1
Output $(16 \times 16 \times 1)$	-	-	-

Table 5.4: Discriminator architecture

For more information on the architectures of generators and discriminators, Github repository[2] can be referred to.

5.2.2 Classifier

In this thesis, the classifiers are used to analyze the domain gap between real data distribution and other data distributions like synthetic data distribution, faxified data distribution, and CycleGAN generated data distribution. Three separate classifiers are trained upon synthetic data distribution faxified data distribution, and CycleGAN generated data distribution respectively. Their classification performance was evaluated on annotated real document images (test dataset) using metrics like weighted average F1-score, macro average F1-score, and accuracy to investigate the domain gap between real data distribution and mentioned three data distributions. Chapter Evaluation 6 discusses more about evaluation metrics. The classifier architecture is simple and easy to implement. It has two convolution layers, one max-pooling layer, and one dropout layer. The architecture of the classifier is illustrated in table 5.5. Also, all the layers in the classifier can be seen in figure A.11.

[2] `https://github.com/junyanz/CycleGAN` last access: September 30, 2021

Operation Layer	Number of Filters/Units	Size of Each Filter	Stride Value
Input Image $(256 \times 256 \times 1)$	-	-	
Convolution Layer ReLU	32	3×3	1×1
Convolution Layer ReLU	64	3×3	1×1
Max Pooling Layer	-	2×2	2×2
Dropout Layer (0.25)	-	-	-
Flatten Layer	-	-	-
Dense Layer ReLU	128	-	-
Dropout Layer (0.5)	-	-	-
Dense Layer Softmax	10	-	-
Output (1×10)	-	-	-

Table 5.5: Classifier architecture

5.3 Training Details

5.3.1 CycleGAN

Designing an efficient and fast input pipeline is challenging. The proposed 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 tf.data APIs to load large datasets spontaneously. To learn more about how to load large datasets efficiently in TensorFlow refer to this tutorial. If the source domain is X and the target domain is Y. The first generator G is called the forward generator, which transforms images from $X \rightarrow Y$, and D_Y acts as a discriminator for the generator G. The second generator F is called the backward generator, which transforms images from $Y \rightarrow X$, and D_X acts as a discriminator for the generator F. For the stable training of the CycleGAN model, the general GAN objective function 3.1 has been modified. In which, the negative log-likelihood objective is replaced by least-squares loss[34]. The GAN loss function $\mathcal{L}_{ls}(G, D_Y, X, Y)$ of the forward generator G minimizes $\mathbb{E}_{x \sim p_{data}(x)} \ (D_Y(G(x)) - 1)^2$ and discriminator D_Y minimizes $\frac{1}{2} \ \mathbb{E}_{y \sim p_{data}(y)} \ (D_Y(y) - 1)^2 + \frac{1}{2} \ \mathbb{E}_{x \sim p_{data}(x)} \ (D_Y(G(x)))^2$. Similarly, the GAN loss function $\mathcal{L}_{ls}(F, D_X, Y, X)$ of the backward generator F minimizes $\mathbb{E}_{y \sim p_{data}(y)} \ (D_X(F(y)) - 1)^2$ and discriminator D_X minimizes $\frac{1}{2} \ \mathbb{E}_{x \sim p_{data}(x)} \ (D_X(x) - 1)^2 + \frac{1}{2} \ \mathbb{E}_{y \sim p_{data}(y)} \ (D_X(F(y)))^2$.

The CycleGAN is trained with a learning rate of 0.0002. The weights neural networks(generators and discriminators) in the CycleGAN model are initialized by a Gaussian distribution with mean (μ) 0 and standard deviation (σ) 0.02. In the final objective function 4.9, λ_{cyc} is set to 10 and $\lambda_{identity}$ is set to 0.5, which means importance of cycle-consistency loss is higher than identity mapping loss in the final objective function. The weights of the neural networks of CycleGAN model are optimized by ADAM solver, which is a method for stochastic optimization[79]. Also, while constructing generators and discriminators the instance normalization layers[80] are used, and gamma weight is initialized by Gaussian distribution with mean (μ) 0 and standard deviation (σ) 0.02.

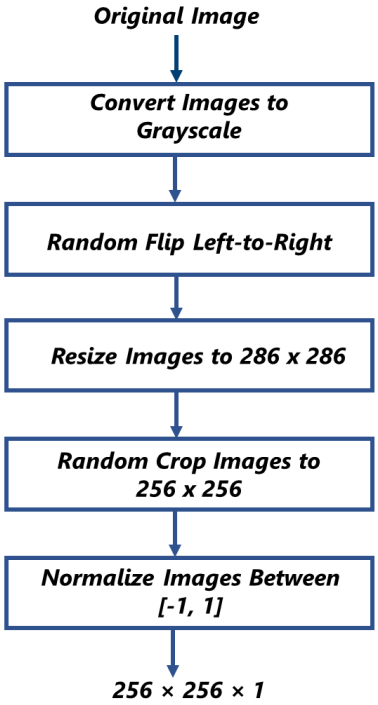


Figure 5.6: Steps involved in preprocessing of training images of CycleGAN.

The CycleGAN model is trained for 20 epochs with batch size 1. The complete CycleGAN model consists of forward generator G, backward generator F, discriminator D_Y, and discriminator D_X. The training checkpoints[3] are saved at the end of every epoch. The stack of 100,000 synthetic document images (Domain X) and 100,0000 real document images (Domain Y) are used to train the CycleGAN model. The training images are preprocessed, and the preprocessing process is illustrated in figure 5.6. Initially, the training images are converted into grayscale. Next, random mirroring is applied, in which the image is randomly flipped horizontally from left to right. Further, random mirroring is applied, in which the image is resized to 286×286 and then randomly cropped to 256×256. Random jittering and mirroring are image augmentation techniques that avoid overfitting[20]. Finally, the images are normalized in the range of $[-1, 1]$.

5.3.2 Classifier

Three separate classifiers are trained, first on synthetic document images, second on faxified document images, and third on CycleGAN generated document images. The synthetic and faxified document images are preprocessed. The preprocessing process of these images is illustrated in figure 5.7. Initially, the training images are converted into grayscale. Next, resized to 256×256 and later normalized between $[-1, 1]$. The CycleGAN generated document images are need not be preprocessed, as the images generated by the generator G ($G : X \rightarrow Y$) are of 256×256 dimension and already normalized between $[-1, 1]$. Because at the final layer of the generator, the tanh activation function is used, this can be seen in the architecture of the generator in table 5.3. All three classifiers are trained for 10 epochs with 100,000 images and evaluated using 1162 annotated real document images (testing dataset).

[3] `https://www.tensorflow.org/guide/checkpoint` last access: September 30, 2021

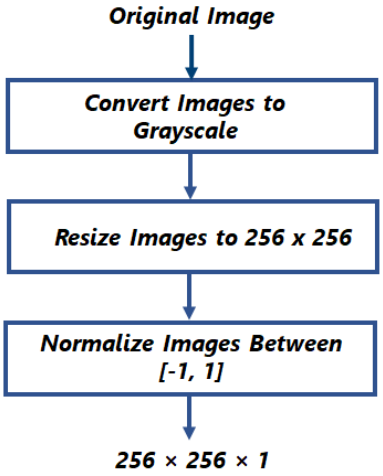


Figure 5.7: Steps involved in preprocessing of training images (Only synthetic and faxified document images) of classifiers.

6. Experiments and Evaluation

This chapter discusses the experiments conducted in this thesis, evaluation metrics, and results. The evaluation metrics like accuracy, precision, recall, confusion matrix, and F1-score are discussed in section 6.1. These metrics are used to evaluate the domain gap between the distributions and quality of CycleGAN generated document images. In section 6.2 conducted experiments are explained along with training plots, confusion matrices, and classification reports. Finally, in section 6.3, quantitative and qualitative results are discussed thoroughly along with the typical failure cases.

6.1 Evaluation Metrics

In machine learning, there are numerous performance metrics to evaluate neural networks. Each evaluates different aspects of neural network performance. Hence, It is needed to have a specific set of performance metrics for a particular problem solved using neural networks. It's important to evaluate the performance of the neural network after training, using testing data, to determine its actual performance or generalization error on unseen data. In this thesis, the performance of classifiers trained upon different data distributions determined using annotated real document images (test dataset). The popular classifier performance evaluation metrics are accuracy, precision, recall, confusion matrix, and F1-score[81].

The testing dataset used to evaluate the classifiers is unbalanced, hence metrics like weighted average and macro average F1-scores are essential for the performance comparison of the classifiers trained on different data distributions. The accuracy is the most common metric used to evaluate classifiers. It is the ratio of accurately classified data items to the total number of observations (equation 6.3). The precision is "how many selected items are relevant"[1]. "To put it another way, out of the observations that an algorithm has predicted to be positive, how many of them are actually positive"[82]. The precision is the ratio of the number of true positives divided by the sum of the true positive and false positives (equations 6.2). The recall is "how many relevant items are selected"[1]. "In fact, out of the observations that are actually positive, how many of them have been predicted by the algorithm"[82]. The recall is the ratio of the number of true positives divided by the sum of true positives and false negatives. In below equations 6.3, 6.2, and 6.1 the TP means true positives, TN means true negatives, FP means false positives, and FN means false negatives.

$$Recall = \frac{TP}{TP + FN} \tag{6.1}$$

$$Precision = \frac{TP}{TP + FP} \tag{6.2}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.3}$$

The F1-score is the harmonic mean of the precision and recall (equation 6.4). The weighted average F1-score is determined by first calculating the F1-score of each class separately and each multiplied by the weight (the number of true instances for each class) and finally added together, hence favoring the majority class. The equation 6.6 represents the weighted average F1-score. The

[1] `https://en.wikipedia.org/wiki/Precision_and_recall` last access: September 30, 2021

macro average F1-score computes the unweighted mean of separate F1-score of each class. The macro average F1-score does not take label inbalance into account. This leads to bigger penalization when the classifier does not perform well on minority classes. The equation 6.5 represents the macro average F1-score. In equations 6.6 and 6.5, N represents number of classes. More information about accuracy, precision, recall, and F1-score can be found here.

$$F1\text{-}score = \frac{2 \times precision \times recall}{precision + recall} \tag{6.4}$$

$$Macro\ average\ F1\text{-}score = \frac{F1_{class1} + F1_{class2} + ... + F1_{classN}}{N} \tag{6.5}$$

$$Weighted\ average\ F1\text{-}score = \frac{F1_{class1} \times W_1 + F1_{class2} \times W_2 + ... + F1_{classN} \times W_N}{N} \tag{6.6}$$

The confusion matrix is the most intuitive metric to determine the accuracy of the classifiers. It is a table that describes how well a classifier performs on a test dataset that is labeled or annotated. It is useful when the classifier has to classify more than two classes. The instances of the true class are represented at each row of the confusion matrix whereas the instances of predicted class probabilities are represented by each column or vice versa. In this thesis, the confusion matrix is extensively used to determine the performance of the classifiers trained on different data distributions to analyze domain gap and quality of CycleGAN generated document images. More information about the confusion matrix can be found here.

6.2 Experiments

In this thesis, three data distributions are considered for the experiments. The synthetic document images represent the synthetic data distribution, the faxified document images represent faxified data distribution, and CycleGAN generated images represent CycleGAN generated data distribution. Experiments are performed to analyze the domain gap between these data distributions and real data distribution, represented by real document images. Also, the experiment performed to determine the quality of CycleGAN generated document images compared to the real document. The experiment is conducted to analyze the domain gap between CycleGAN generated data distribution and real data distribution, determines the quality of CycleGAN generated document images, revealing how much they are close to the real data distribution.

Now let's describe the conducted experiments briefly, a) Train a classifier on synthetic document images, and its performance is evaluated on testing dataset (Annotated real document images, table 5.2). b) Another classifier with the same architecture is trained on faxified document images, and its performance is evaluated on the testing dataset. c) The CycleGAN is trained for 20 epochs, using synthetic document images (Source domain X) and real document images (Target domain Y), and the checkpoint is saved at the end of every epoch. Once CycleGAN training is finished, the saved model is loaded to generate images or transform synthetic document images into realistic document images. From the saved CycleGAN model, generator G, retrieved, as it transforms synthetic document images into realistic document images (Source domain X to Target domain Y, $G : X \rightarrow Y$). 100,000 synthetic document images transformed into realistic document images, generating a dataset of 100,000 CycleGAN generated document images, with 10 classes, and each class has 10000 images. d) Next, another classifier, with same architecture is trained on the CycleGAN generated document images, and its performance is evaluated on testing dataset. The experiment determines the quality of the images generated by CycleGAN. It indicates how well CycleGAN was able to close the domain gap between synthetic data distribution and real data distribution. To put it another way, how well the CycleGAN generated data distribution matched the real data distribution. As a whole, how similar CycleGAN generated document images are to the real document images. In this thesis,

the evaluation metrics like accuracy, weighted average F1-score, and macro average F1-score[83] are used to determine the performance of classifiers on testing dataset. The architecture of the classifier is illustrated in table 5.5.

6.2.1 Experiment Steps

1. Train a classifier using synthetic document images and evaluate its performance over testing dataset.
2. Train a classifier using faxified document images and evaluate its performance over testing dataset.
3. Train CycleGAN using synthetic document images and real document images.
4. Generate realistic document images using generator G retrieved from the trained CycleGAN model.
5. Train a classifier using CycleGAN generated document images and evaluate its performance over testing dataset. This experiment reveals how well the CycleGAN generated document images are generalizing to the real document images, eventually determining the quality of images generated by CycleGAN.
6. Compare the classification performance of the above three classifiers (trained on 3 different data distributions) on testing dataset, and determine which distribution is closer to the real data distribution.

6.2.2 Training a Classifier on Synthetic Document Images

The dataset of synthetic document images is created using templates (figure of sample A.8) and handwritten crops (figure of sample 5.1) using the process mentioned in figure 5.2. This dataset consists of 100,000 document images, 10 classes, and each class has 10000 images. In this experiment, a classifier is trained on synthetic document images dataset. The performance of this classifier is evaluated on testing dataset (Annotated real document images) to understand the domain gap between real data distribution and synthetic data distribution. The classification report in table 6.1 states that the accuracy of this classifier on real document images is 25%, macro average F1-score is 27%, and weighted average F1-score is 31%. The obtained results conclude that there is a huge gap between real data distribution and synthetic data distribution. The confusion matrix is illustrated in figure A.6. In the confusion matrix, it is completely visible that most of the images from the testing dataset were classified wrongly. The epochs against accuracy and epochs against loss plots while training classifier on synthetic document images is illustrated in figure 6.1 and 6.2 respectively.

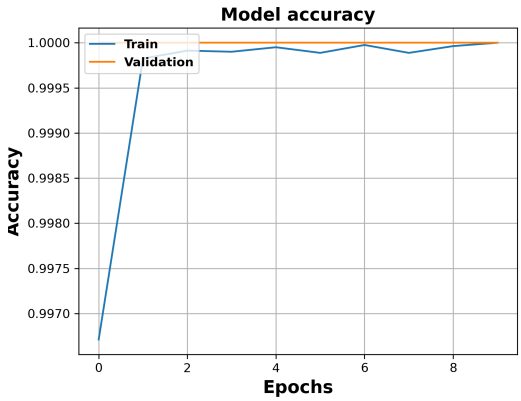


Figure 6.1: Epochs vs. Accuracy plot while training a classifier on synthetic document images.

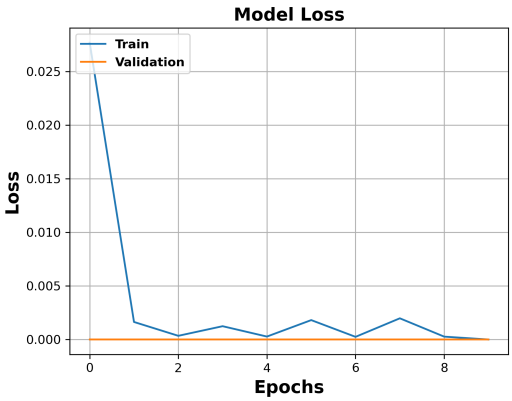


Figure 6.2: Epochs vs. Loss plot while training a classifier on synthetic document images.

	Precision	Recall	F1-score	Support
DE_LY_Arm_2020-01	0.65	0.50	0.56	44
DE_LY_Bein_2018-08	0.50	0.02	0.04	47
DE_LY_Bein_2019-01	0.06	0.90	0.11	50
DE_LY_Bein_2019-07	0.36	0.20	0.26	60
DE_LY_Bein_2020-01	0.96	0.21	0.34	624
DE_LY_Bein_2020-03	0.24	0.25	0.24	128
DE_LY_Hand_2020-01	0.70	0.44	0.54	16
DE_PH_Bein_2018-09	0.10	0.14	0.12	22
DE_PH_Bein_2019-02	0.12	0.04	0.06	28
DE_PH_Bein_2020-01	0.95	0.51	0.26	143
Accuracy			**0.25**	1162
Macro average	0.46	0.29	**0.27**	1162
Weighted average	0.74	0.25	**0.31**	1162

Table 6.1: Classification report, evaluating a classifier on the testing dataset after training with synthetic document images.

6.2.3 Training a Classifier on Faxified Document Images

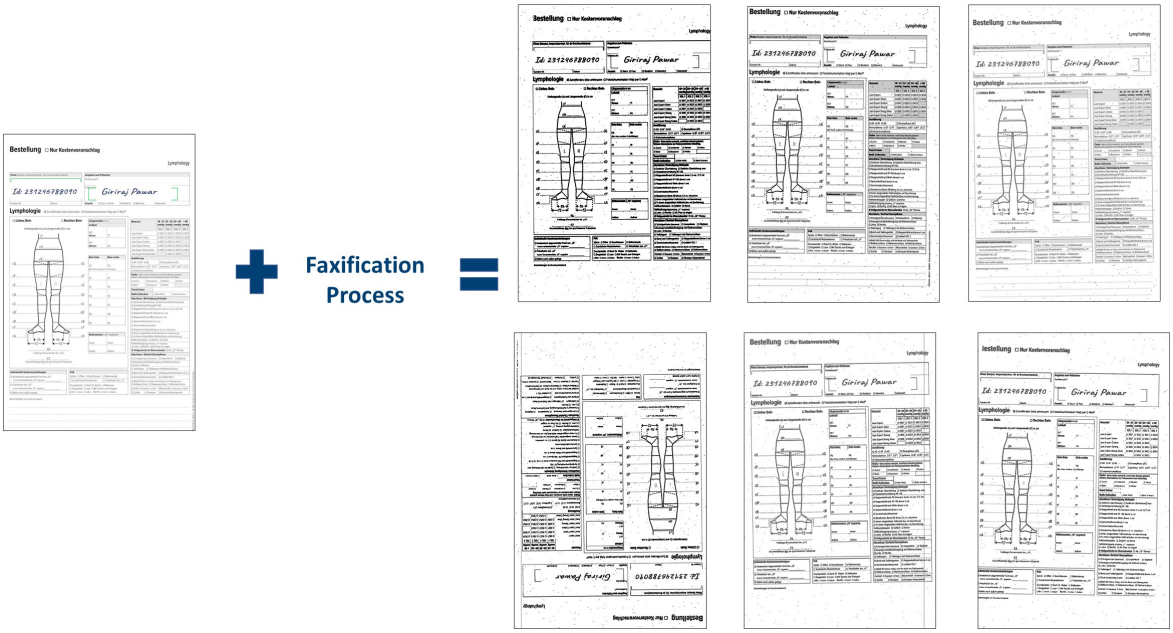


Figure 6.3: Illustration of faxification process applied on synthetic document images (figure reproduced from elevait GmbH & Co. KG with permission).

The faxification process mimics the way the fax machine works. Usually, fax machines transmit only black-and-white images, which might be dirty and are generally also not aligned perfectly. This leads to several common artifacts being introduced into transferred images, so faxification process attempts to mimic those introductions of artifacts into images. The faxification process transforms clean gray-scale synthetic document images in such a way like it was sent via fax. The faxification process is not deterministic, involves randomness during the process of faxification of the images. It uses several image transformations like gamma transformation, brightness transformation, 180-degree rotations, resizing, rescaling, binarization, adding noise, adding verticle lines, and conversion

to a grayscale image. The faxification process can be visualized in figure 6.3. In figure 6.4, it is visible a snippet from the synthetic document image that has transformed randomly into different image transformations when it has been through the faxification process. The faxification process is implemented in Python using a traditional programming approach.

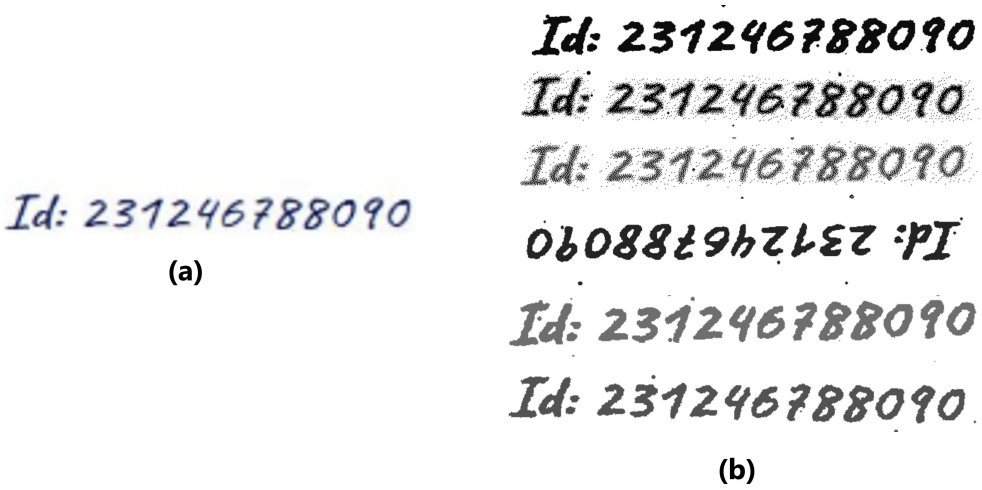


Figure 6.4: Illustration of faxified document images to conclude that faxification process is a random process, the input images are faxified randomly to create distinct and random output. For example, for a snippet in a synthetic document image (a), snippets of faxified document images shown in (b)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

The faxified document images dataset is created by transforming synthetic document images using the faxification process (figure 6.3). In this experiment, the classifier is trained on a faxified document images dataset. The performance of this classifier is evaluated on the testing dataset to understand the domain gap between real data distribution and faxified data distribution. The classification report in table 6.2 states that the accuracy of this classifier on testing dataset is 43%, macro average F1-score is 58%, and weighted average F1-score is 43%. The results conclude that the faxified data distribution matched the real data distribution better than the synthetic data distribution. The confusion matrix is illustrated in figure A.7. In the confusion matrix, it's visible that document images of class DE_LY_Bein_2020-01 are wrongly classified as DE_LY_Bein_2020-03. The rest of the document images are classified satisfactorily. The epochs against accuracy and epochs against loss plots while training classifier on synthetic document images is illustrated in figure 6.5 and 6.6 respectively.

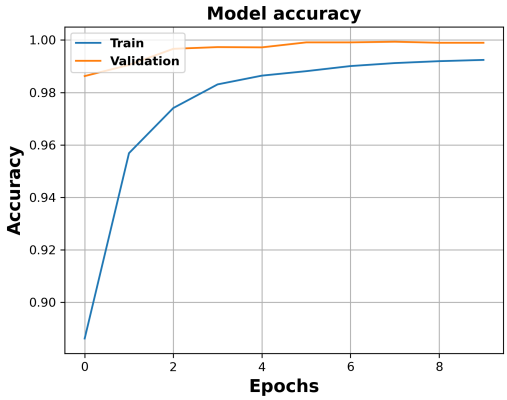


Figure 6.5: Epoch vs. Accuracy plot while training a classifier on faxified document images.

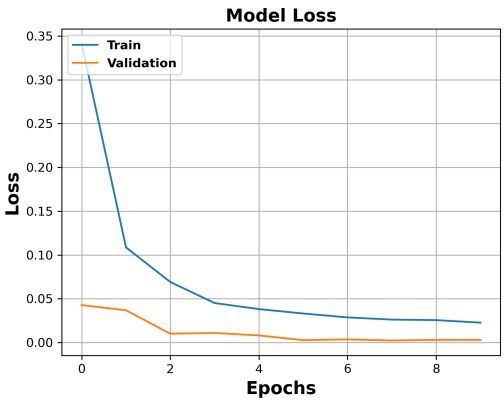


Figure 6.6: Epoch vs Loss Plot.

	Precision	Recall	F1-score	Support
DE_LY_Arm_2020-01	0.97	0.75	0.85	44
DE_LY_Bein_2018-08	1.00	0.53	0.69	47
DE_LY_Bein_2019-01	0.74	0.34	0.47	50
DE_LY_Bein_2019-07	0.53	1.00	0.69	60
DE_LY_Bein_2020-01	1.00	0.17	0.29	624
DE_LY_Bein_2020-03	0.19	0.98	0.32	128
DE_LY_Hand_2020-01	0.21	1.00	0.34	16
DE_PH_Bein_2018-09	0.68	0.68	0.68	22
DE_PH_Bein_2019-02	0.78	0.64	0.71	28
DE_PH_Bein_2020-01	1.00	0.59	0.74	143
Accuracy			**0.43**	1162
Macro average	0.71	0.67	**0.58**	1162
Weighted average	0.85	0.43	**0.43**	1162

Table 6.2: Classification report, evaluating a classifier on the testing dataset after training with faxified document images.

6.2.4 CycleGAN Training

Random input images x_i and y_i are retrieved from source domain X and target domain Y. The fake image generated using generator G is *Generated_Y* and the fake image generated using generator by F is *Generated_X*. As we described earlier, G is a mapping function to transform an image from source domain X to target domain Y, mathematically it is $G : X \rightarrow Y$, and F is vice versa $F : Y \rightarrow X$. The generator G is called the forward generator, and F is called the backward generator. The discriminator loss for D_X and D_Y on real and fake images is computed. The computed discriminator loss is used to update the weights of both discriminators. Next, cycle-consistency loss is calculated by transforming the source domain image into the target domain, back again into the source domain. It is called forward cycle-consistency loss. It is calculated by determining the difference between the original image in the source domain and cycled image back in the source domain. If you refer to figure 4.1, the forward cycle-consistency loss is the difference between *Input_X* and *Cyclic_X*. The backward cycle-consistency loss is calculated by transforming the target domain image into the source domain, back again into the target domain. In figure 4.1, the backward cycle-consistency loss is the difference between *Input_Y* and *Cyclic_Y*. The cycle-consistency loss forces generating an image that is context instead of generating random images.

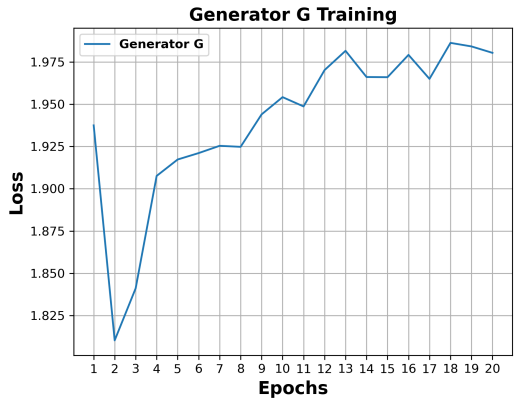


Figure 6.7: Generator G training Epochs vs. Loss plot.

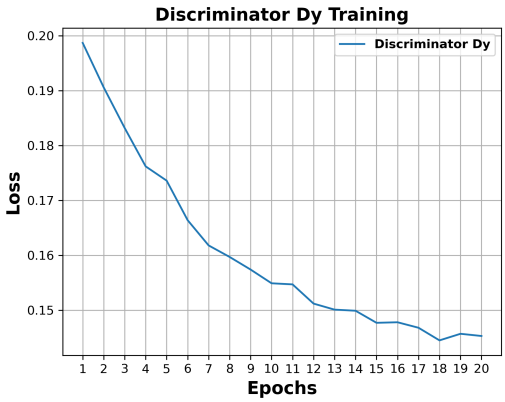


Figure 6.8: Discriminator D_Y training Epochs vs. Loss plot.

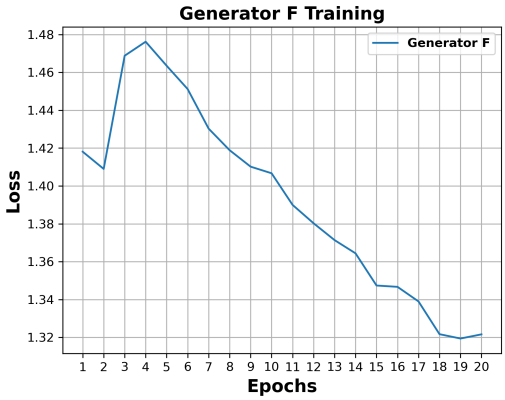


Figure 6.9: Generator F training Epochs vs. Loss plot.

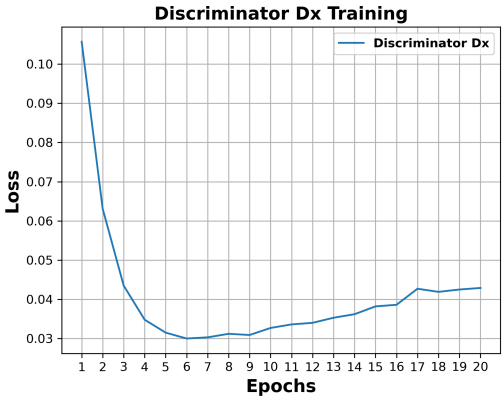


Figure 6.10: Discriminator D_X training Epochs vs. Loss plot.

Further, identity mapping loss is calculated to preserve the color of images after domain translation using generators G and F. The generator G transforms the image from domain X to the domain Y. When the input y_i is given to generator G from the target domain Y, it should produce an identical image. The identity mapping loss for generator G is the difference between the original image y_i, and the transformed image $Same\_Y$ when y_i is used as an input for G. And, the identity mapping loss for generator F is the difference between input image x_i and the transformed image $Same\_X$ when x_i is used as an input for F. The generator loss for generators G and F is calculated by the feedback given by their respective discriminators D_Y and D_X. Combining generator loss, cycle-consistency loss, and identity mapping loss, the total generator loss is calculated separately for generators G and F, and their weights are updated to produce quality images. The epochs against loss plot while training generators G and F are shown in figure 6.7 and 6.9 respectively. Also, the epochs against loss plot while training discriminators D_X and D_Y are shown in figure 6.10 and 6.8 respectively. To understand the algorithm of CycleGAN in-depth, refer to the pseudo-code from 2. The complete architecture of the proposed image-to-image translation application using CycleGAN is illustrated in figure 4.1. For more CycleGAN training details refer section 5.3.1.

6.2.5 Training a Classifier on CycleGAN Generated Document Images

Once CycleGAN is trained, the generator G is retrieved from the saved CycleGAN model to transform synthetic document images to realistic document images. 100,000 synthetic document images are transformed into realistic document images. The classifier is trained using these transformed 100,000 realistic document images. The performance of this classifier is evaluated on the testing dataset to understand the domain gap between real data distribution and CycleGAN generated data distribution. The purpose of this experiment is to infer how much the CycleGAN generated data distribution matches the real data distribution. The quality of generated realistic document images can be determined using this experiment. The classification report in table 6.3 states that the accuracy of this classifier on the testing dataset is 27%, macro average F1-score is 34%, and weighted average F1-score is 25%. The results conclude that the faxified data distribution matched the real data distribution better than the CycleGAN generated data distribution. The confusion matrix is illustrated in figure A.7. In the confusion matrix, the document images DE_LY_Arm_2020-01, DE_LY_Bein_2019-07, DE_LY_Hand_2020-01, and DE_PH_Bein_2020-01 are classified satisfactorily. Document images DE_LY_Bein_2020-01 and DE_LY_Bein_2020-03 are wrongly classified as DE_LY_Bein_2019-07. Document images DE_PH_Bein_2018-09 and DE_PH_Bein_2019-02 are wrongly classified as DE_PH_Bein_2019-02. In section 6.3.2, qualitative results of the image-to-image translation are illustrated along with failure cases. The epochs against accuracy and epochs against loss plots while training classifier on CycleGAN generated document images is illustrated in figure 6.11 and 6.12 respectively.

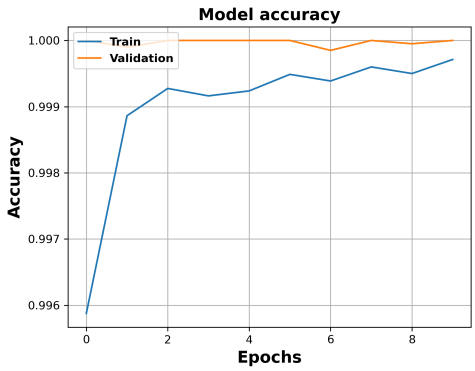


Figure 6.11: Epochs vs. Accuracy plot while training a classifier on CycleGAN generated document images.

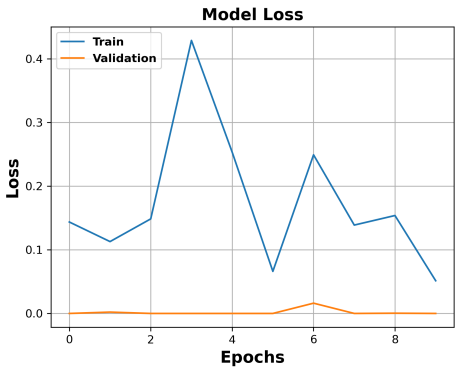


Figure 6.12: Epochs vs. Loss plot while training a classifier on CycleGAN generated document images.

	Precision	Recall	F1-score	Support
DE_LY_Arm_2020-01	0.53	0.75	0.62	44
DE_LY_Bein_2018-08	0.21	0.28	0.24	47
DE_LY_Bein_2019-01	0.16	0.14	0.15	50
DE_LY_Bein_2019-07	0.11	0.83	0.19	60
DE_LY_Bein_2020-01	0.72	0.07	0.12	624
DE_LY_Bein_2020-03	0.16	0.34	0.22	128
DE_LY_Hand_2020-01	0.80	0.75	0.77	16
DE_PH_Bein_2018-09	0.07	0.05	0.06	22
DE_PH_Bein_2019-02	0.33	0.46	0.39	28
DE_PH_Bein_2020-01	0.70	0.67	0.68	143
Accuracy			**0.27**	1162
Macro average	0.38	0.43	**0.34**	1162
Weighted average	0.55	0.27	**0.25**	1162

Table 6.3: Classification report, evaluating a classifier on the testing dataset after training with CycleGAN generated document images.

6.3 Results

In this chapter, the qualitative and quantitative results are discussed. The qualitative results are illustrated in section 6.3.2, to understand the visual quality of CycleGAN generated document images. Some sample synthetic document images transformed into realistic document images by the proposed image-to-image translation application are illustrated in figure 6.14. The typical failure cases of the image-to-image transformation are illustrated in section 6.3.3. Figure 6.15 illustrates the proposed image-to-image translation application using CycleGAN has failed to transform handwritten crops present in the synthetic document images. Figure 6.16 illustrates the transformed image has undesired noisy artifacts. Figure 6.17 illustrates transformed image has a dark border and noisy artifacts. Figure 6.18 illustrates transformed image has random artifacts that are not present in the synthetic document image.

The quantitative results are illustrated using a table and bar plot in section 6.3.1. In which, the performance of the classifier trained using different distributions is evaluated, and compared using the testing dataset (Annotated real document images), to analyze the domain gap between these distributions and real data distribution, and to determine the quality of CycleGAN generated

document images. The weighted average F1-score states, 25% real document images were correctly when a classifier is trained using CycleGAN generated document images. 43% real document images were correctly when a classifier is trained using faxified document images. Results infer quantitatively CycleGAN generated data distribution could not match real data distribution, but the faxified data distribution created using the faxification process was satisfactorily matched to the real data distribution, higher compared to CycleGAN generated data distribution. The initial qualitative results look very hopeful for further research (figure 6.3.2). The proposed image-to-image translation application can be improved further in the future using different methods and loss functions. The future work and conclusion discussed in the chapter 7.

6.3.1 Quantitative Results

Data distributions	Accuracy	Weighted average F1-score	Macro average F1-score
Synthetic document images	25%	31%	27%
CycleGAN generated document images	27%	25%	34%
Faxified document images	43%	43%	58%

Table 6.4: The accuracies and F1-scores when the classifiers trained on different data distributions and evaluated on testing dataset (Annotated real document images).

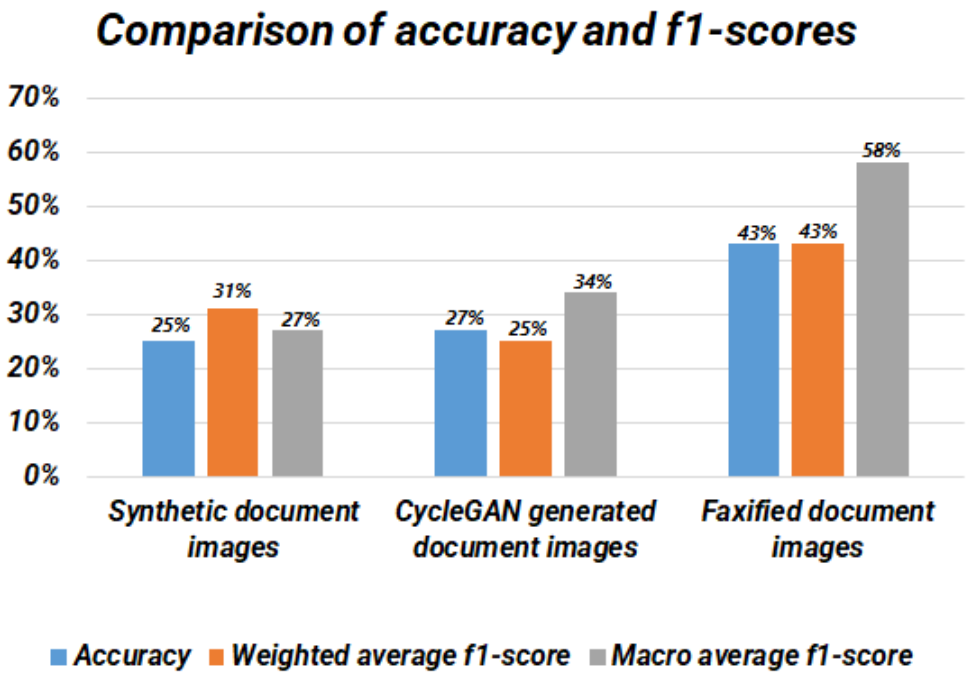


Figure 6.13: Comparison of accuracies and F1-scores, when the classifiers trained on different data distributions and evaluated on testing dataset (Annotated real document images).

6.3.2 Qualitative Results

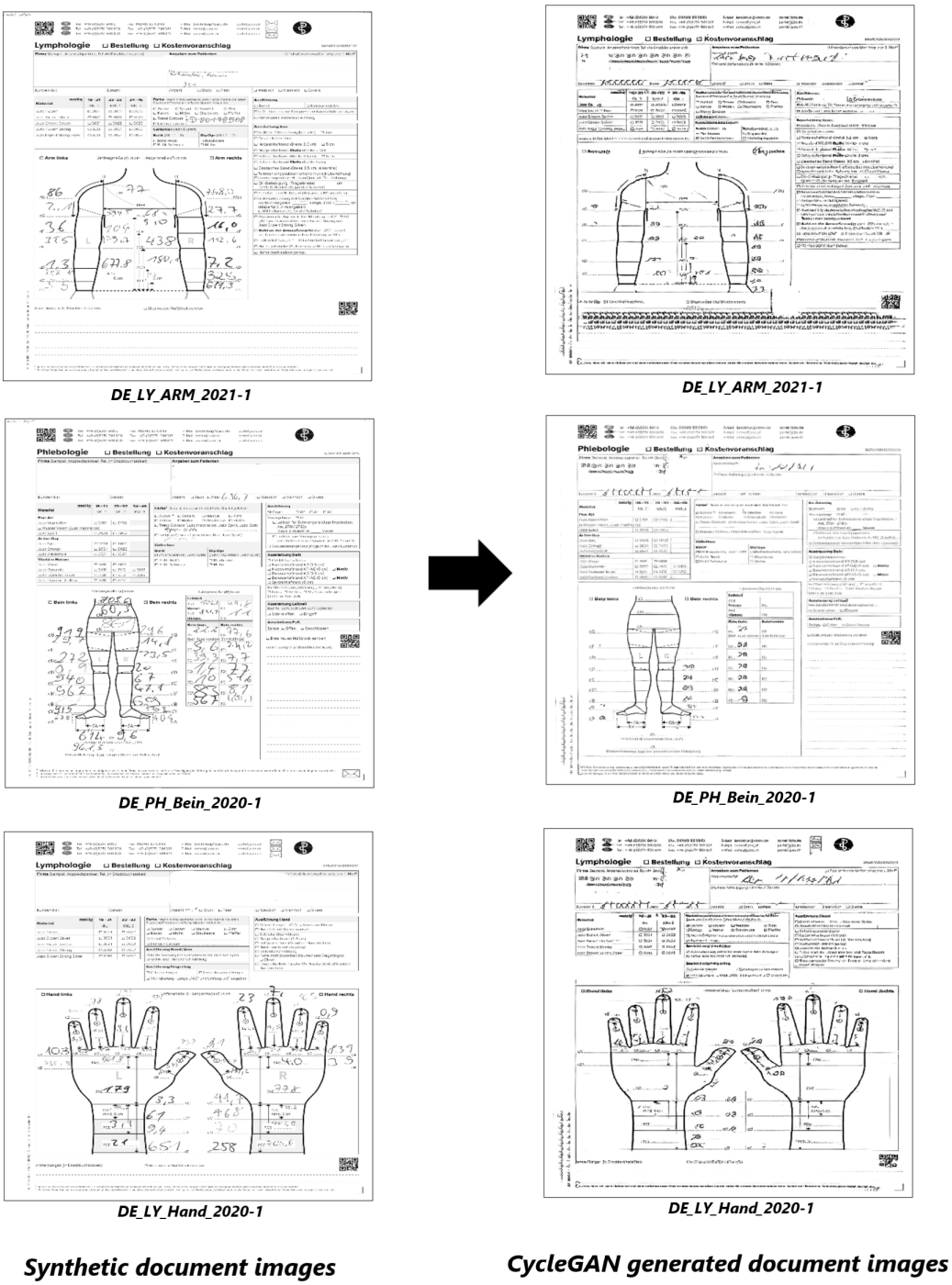


Figure 6.14: Synthetic document images transformed into realistic document images by the image-to-image translation application (figure reproduced from elevait GmbH & Co. KG with permission).

6.3.3 Failure Cases

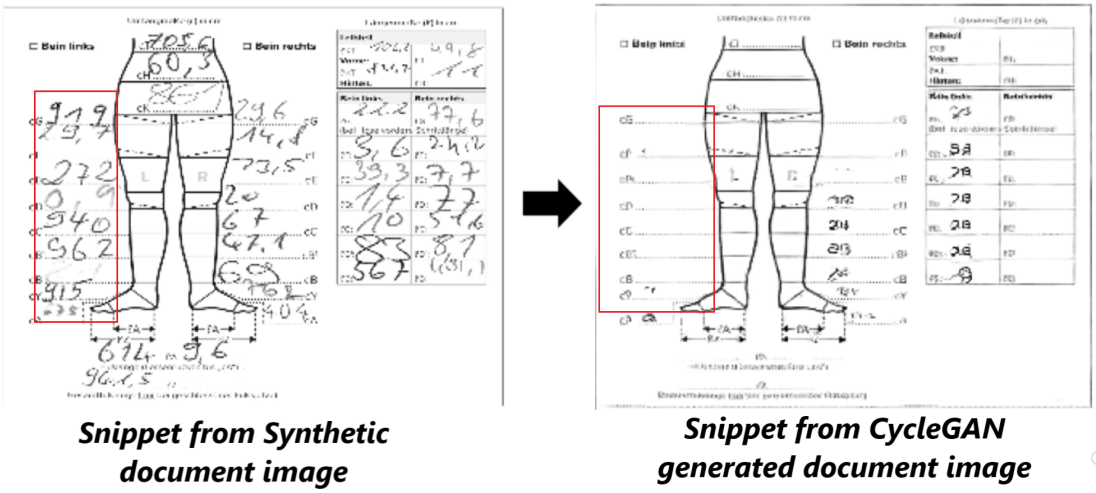


Figure 6.15: The snippet from CycleGAN generated document image illustrates that the proposed image-to-image translation application did not transform or reconstruct handwritten crops in the target domain from the synthetic document image in the source domain (figure reproduced from elevait GmbH & Co. KG with permission).

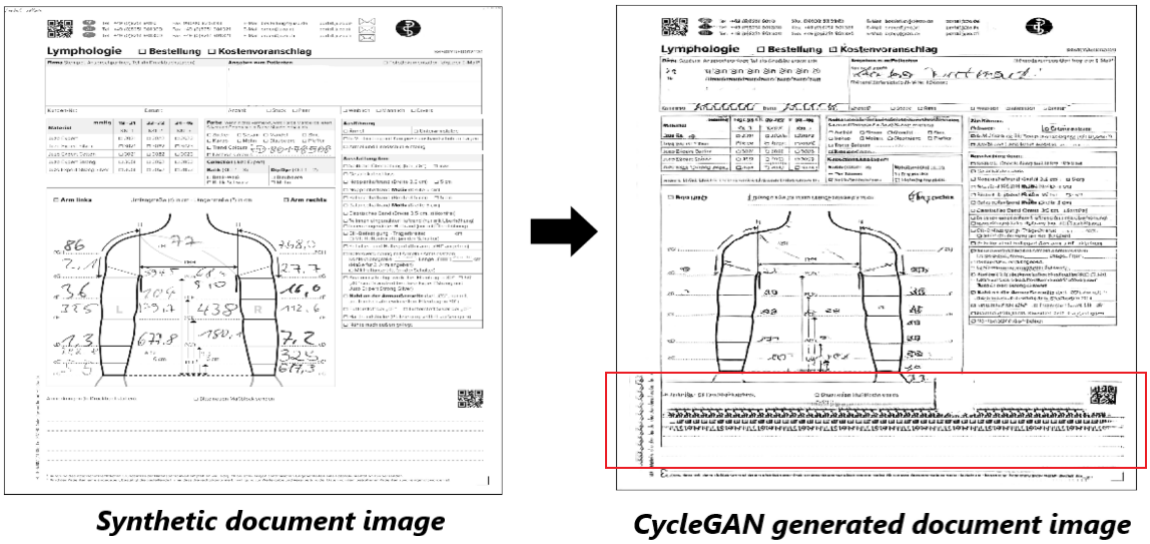


Figure 6.16: The CycleGAN generated document image consists of noisy artifacts that are unrealistic (figure reproduced from elevait GmbH & Co. KG with permission).

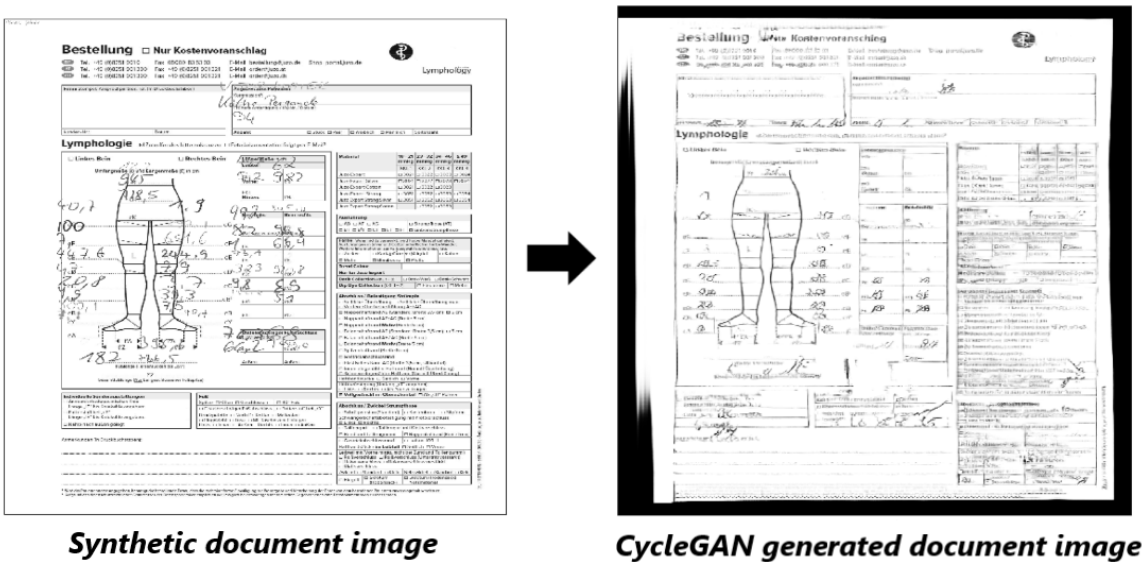


Figure 6.17: The CycleGAN generated document image consists of dark border (figure reproduced from elevait GmbH & Co. KG with permission).

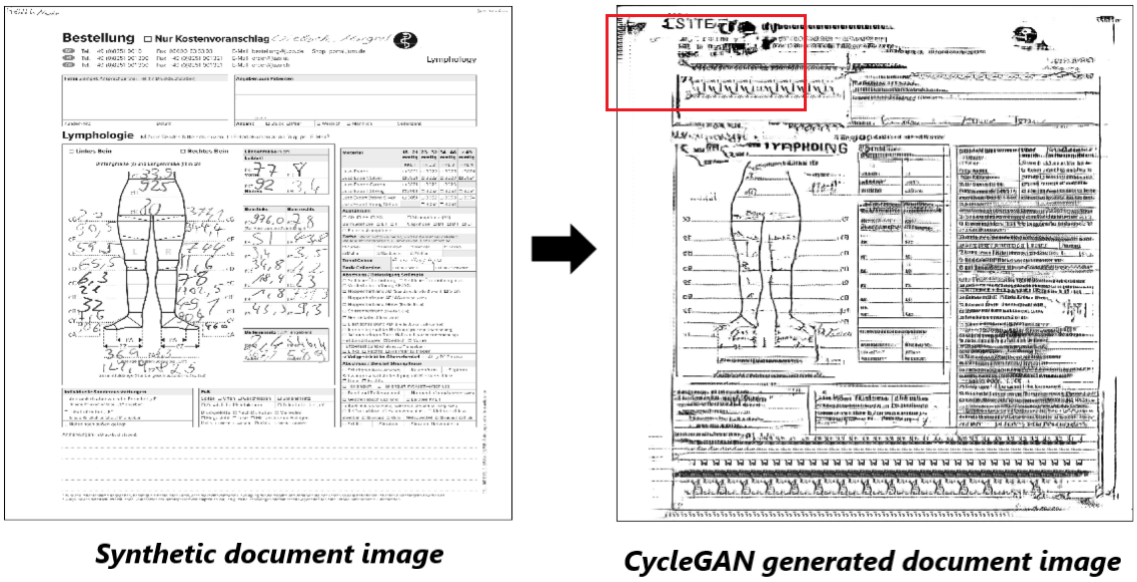


Figure 6.18: The CycleGAN generated document image consists of noisy artifacts that are unrealistic and the visual quality of the generated image is not good. There are some artifacts (marked in red box) appear in the generated images that are not present in the synthetic image (figure reproduced from elevait GmbH & Co. KG with permission).

7. Conclusion and Future Work

In this chapter, the overview of this thesis is described by outlining the problem statement and the proposed solution in section 7.1. The results of the experiments conducted by the proposed image-to-image translation application to reduce the domain gap between synthetic data distribution and real data distribution are concluded in section 7.2. Finally, in section 7.3 the limitations and future work are discussed.

7.1 Overview

Since the last decades, many success stories about deep learning have been written. Also, it has evolved tremendously due to the introduction of neural networks, which learn complex patterns present in videos, images, and speeches. Nowadays, neural networks are widely used in autonomous vehicles, language translations, object detection, face recognition, speech recognition, and many other sophisticated applications. However, there are limitations to deep learning models. In traditional deep learning models, neural networks are trained and tested on similar data distribution or in the same domain to achieve the final objective by minimizing the error. In real-world scenarios, the data could arrive from different feature spaces, data distributions, and different domains. In such situations, neural networks don't generalize well to the data arrived from new data distribution or domain; for example, the model to detect pedestrian trained images of a pedestrian in summer might fail to detect pedestrians in winter. This problem of performance degradation causes because domain shift[84].

A large amount of training data is required to train neural networks to achieve low generalization error and high performance. However, training data are scarce and it is a very difficult job to collect and annotate the data. Hence, machine learning engineers required to create synthetic data, but neural networks trained using synthetic data will not generalize well on real data. So, over the year several domain adaption methodologies are developed by machine learning researchers to solve the problem of domain shift by closing the domain gap between two different domains[84]. One of such domain adaptation methodologies is image-to-image translation, in which images from the source domain are transformed into the target domain by a function that learns the underlying characteristics extracted from the collection of images in both domains. For example, transforming the image of a zebra from the source domain into the image of a horse in the target domain, and vice versa.

7.2 Conclusion

This thesis aims to solve the problem of data scarcity of real document images in the target domain using an image-to-image translation application. The proposed image-to-image translation application is developed using CycleGAN to transform synthetic document images into realistic document images to close the domain gap between synthetic data distribution and real data distribution. The synthetic document images created using templates and handwritten crops are transformed into realistic document images to tackle the problem of data scarcity in the target domain further to improve the classifier of real document images. The quality of images generated using CycleGAN is computed on the basis of how a classifier trained using images generated by the CycleGAN generalizes to the testing dataset of annotated real document images.

In this thesis, CycleGAN is trained using form document images. The form document images consist of minute artifacts like check boxes, small fields, and different handwritings, which increases the complexity in the learning of neural networks. Also, most of the images used for generating

realistic document images are of type "Bein", and they are very similar to each other. Small artifact transformation failure has caused the wrong classification because learning such distinct minor features present in training datasets by CycleGAN is very challenging. The images of the type "Arm" and "Hand" are very distinct from the image type "'Bein". Hence, they are classified significantly well A.6. Results conclude quantitatively the images generated using CycleGAN did not match well to the real data distribution. Further, experiments conducted with dataset of synthetic document images, faxified document images to understand domain gap between these distributions and real data distribution. The quantitative results show the faxified data distribution has matched real data distribution significantly well compared to synthetic data distribution and CycleGAN generated data distribution.

Qualitatively the images generated using CycleGAN are promising for further research (figure 6.3.2). Though qualitative results are promising, some typical failure cases have been identified, and it requires further attention in the future to improve the quality of CycleGAN generated images. Failure cases like the transformation of handwriting crops from synthetic document images to realistic document images are failed (figure 6.15), and the generated realistic document images have unrealistic noise, dark borders, and unrealistic artifacts (figures 6.16, 6.17, and 6.18)). In the next section 7.3, a discussion about improving the proposed image-to-image translation application is discussed, along with thesis limitations.

7.3 Future Work

The proposed image-to-image translation application is implemented only using CycleGANs. Comparison between other methodologies and chosen methodology CycleGAN to transform synthetic document images into realistic document images is left for future research. For example CUT[47], FastCUT[47], Adversarial Consistency Loss Generative Adversarial Network (ACL-GAN)[85], Variational Autoencoder (VAE)[86], and DCGAN[37] are recommended for future research. GANs are a great success in generating realistic images, but the training process is not easy, the training is slow and unstable. GANs are hard to train. It has problems like mode collapse, vanishing gradients, lack of proper evaluation metric, and hard to converge. The most important problem with GANs is there no proper evaluation metric, without a good evaluation metric, it is like working in darkness. GANs have complex object functions observing training progress is difficult, there is no signal to where to stop the training, also, there no single common and good performance indicator to evaluate numerous types of GANs. In future, finding a proper performance indicator for GANs can a part of research.

In this thesis, CycleGAN's generators and discriminators are optimized using least-square loss function[34] to avoid mode collapse and vanishing gradient problems. The quality of generated images possibly would have been better if generators and discriminators of CycleGAN would have optimized using the Wasserstein metric. Wasserstein distance metric is popularly used in WGANs. WGANs improve GAN's training by adopting a smooth Wasserstein distance metric for measuring the distance between two probability distributions[60]. Shrivastava et al.[59] proposed a strategy for stable training. In which the discriminators are updated using a history of previously generated images instead of the ones produced recently by the generators. An image buffer of maintained to store 50 previously generated images. In the future, this strategy can be used to improve stability during the training and reduce model oscillations to produce better results[59]. Qualitative results show the handwritten crops from synthetic document images were not reconstructed or transformed in the generated images (figure 6.15). In the future, this thesis work can be extended to solve this problem, so the generated images are used to improve the handwriting recognition models. Training CycleGAN sequentially is consumed numerous hours. Hence in the future, distributed training across multiple GPUs would be preferred while training the CycleGANs.

Data analysis and data cleaning are the essential steps taken before training any neural network. The dataset used for the training the CycleGAN consists of synthetic document images dataset and real document images dataset. The dataset of synthetic document images is equally distributed. Every selected class has the same number of samples. The dataset of real document images had random real images from different classes, which are unknown and disorganized. It is just a collection large number of real document images. In the future, to obtain better results the real document

images dataset can be organized similarly to the synthetic document images dataset by performing data analysis and data cleaning at the initial stages of the research.

A. Appendix

A.1 MNIST Handwritten Numbers Dataset

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

A.2 CycleGAN Models Training

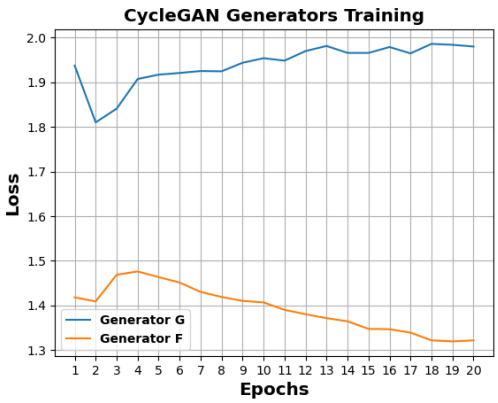


Figure A.2: CycleGAN generators G and F training epochs vs loss plot.

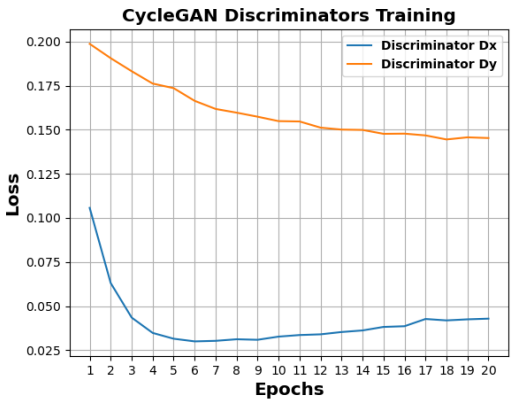


Figure A.3: CycleGAN discriminators D_X and D_Y training epochs vs loss plot.

[1] `http://yann.lecun.com/exdb/mnist/` last access: September 30, 2021

A.3 GAN Training

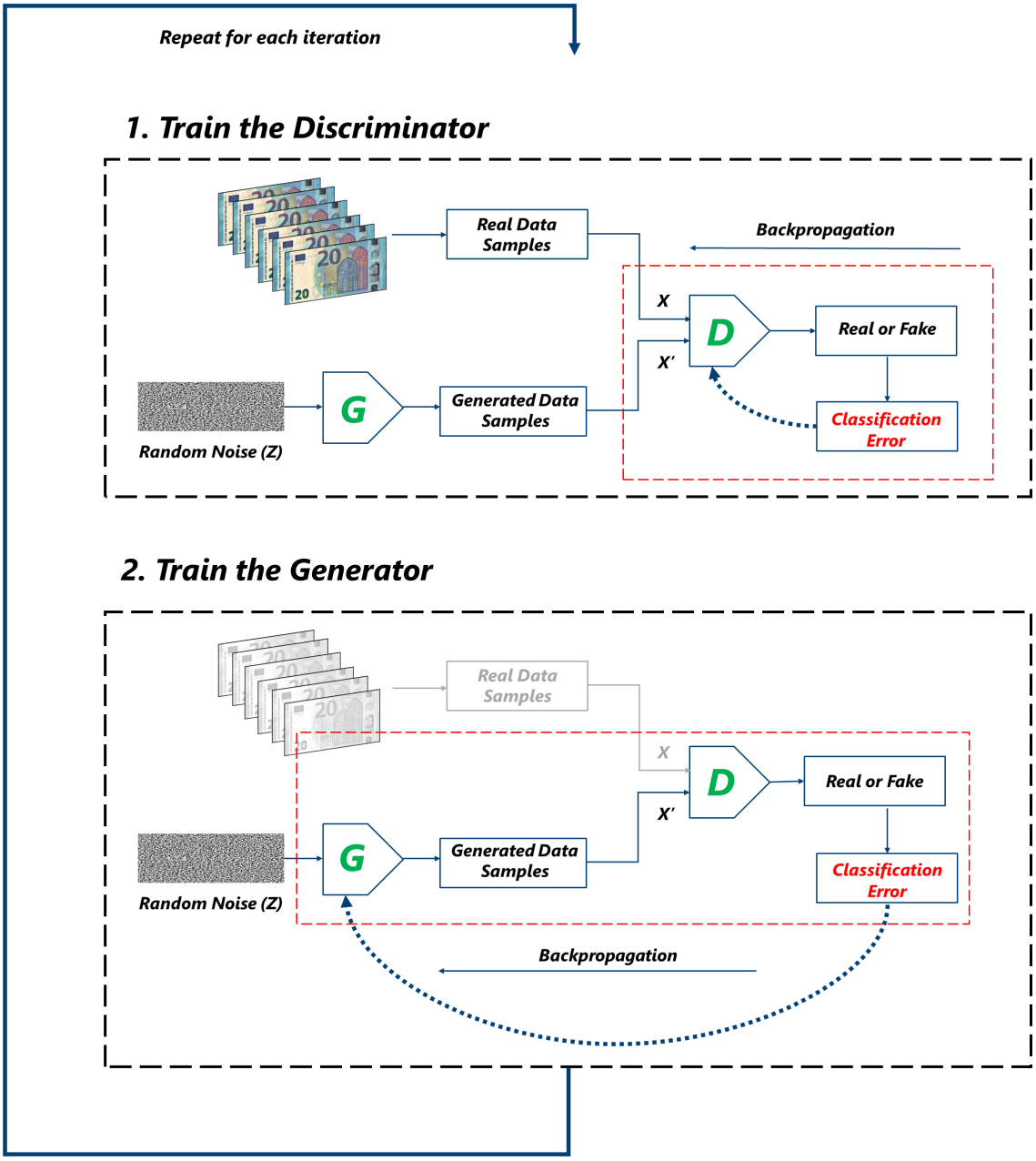


Figure A.4: Illustration of training of the GAN as per the algorithm.

A.4 Confusion Matrices

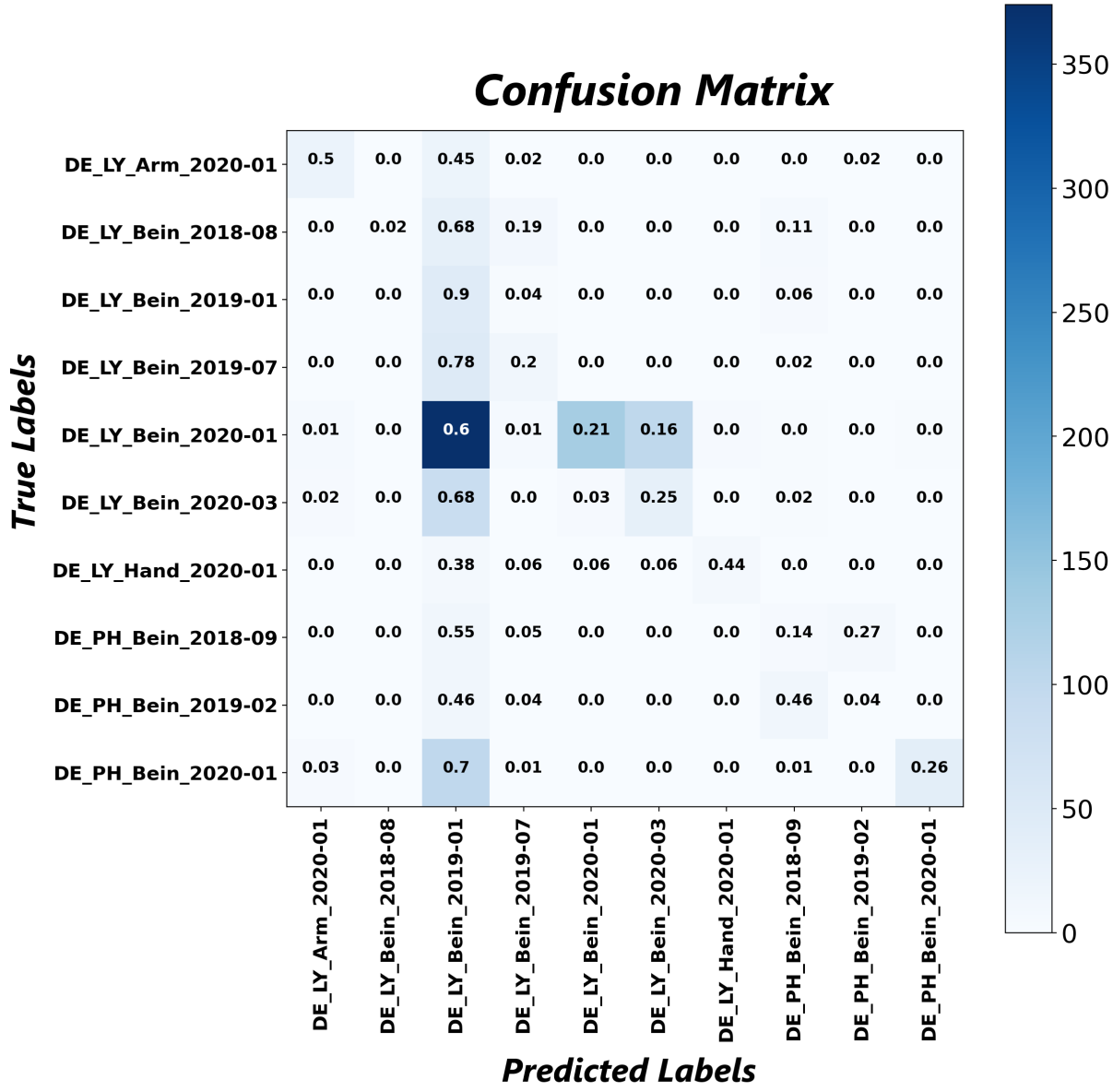


Figure A.5: Confusion matrix to analyze the performance of the classifier trained on synthetic document images and evaluated using real annotated document images (test dataset).

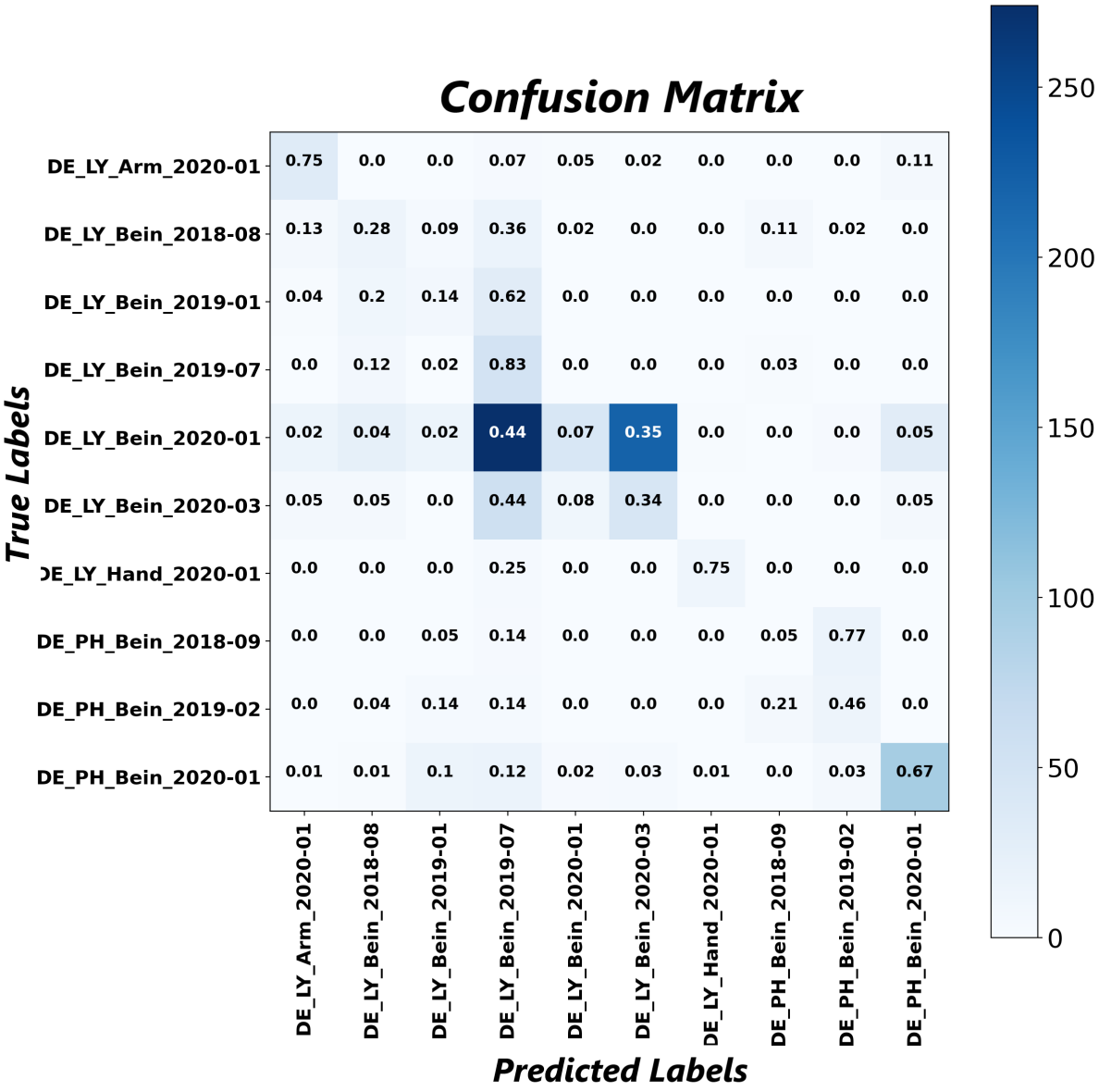


Figure A.6: Confusion matrix to analyze the performance of the classifier trained on CycleGAN generated document images and evaluated using real annotated document images (test dataset).

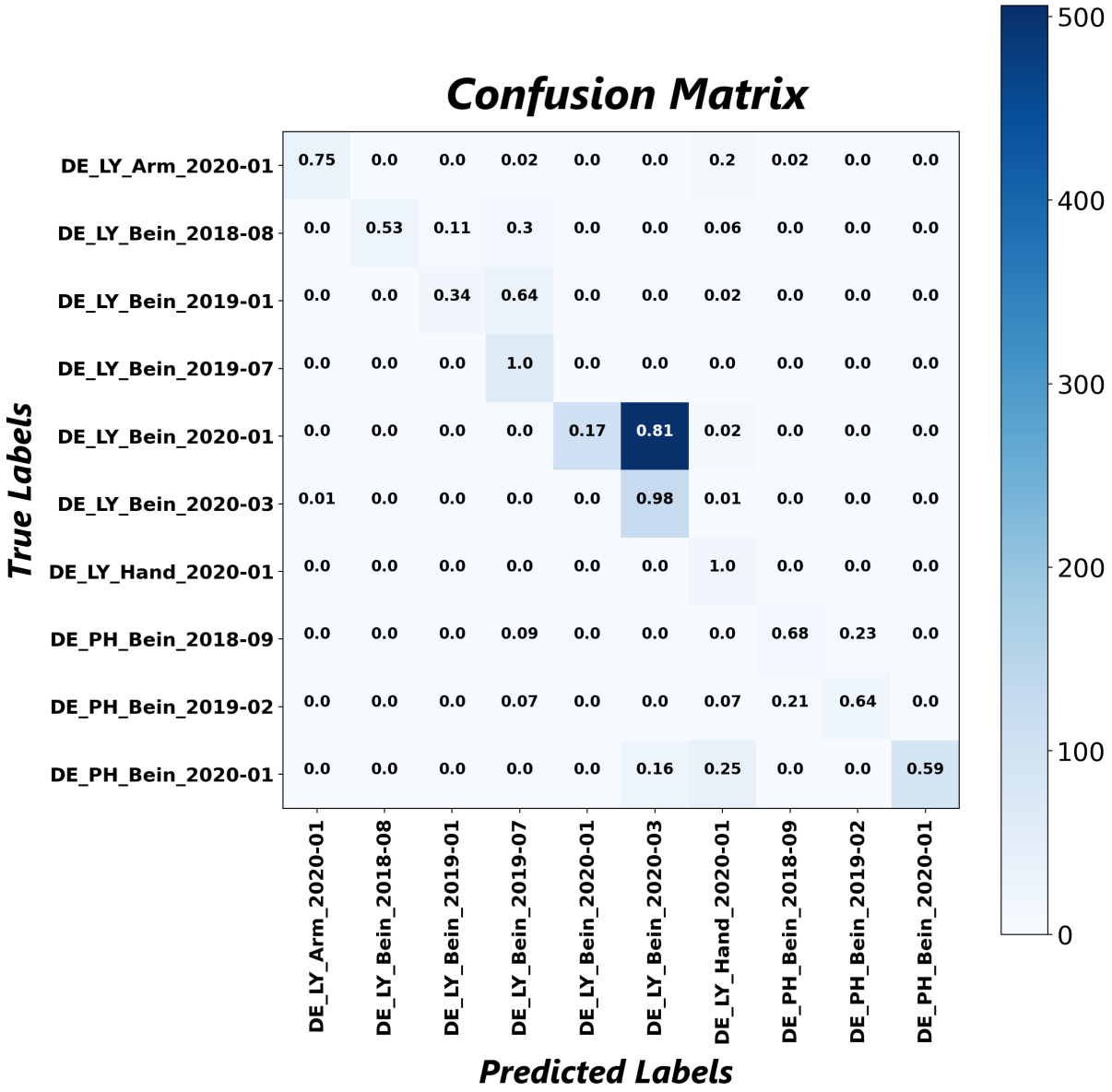
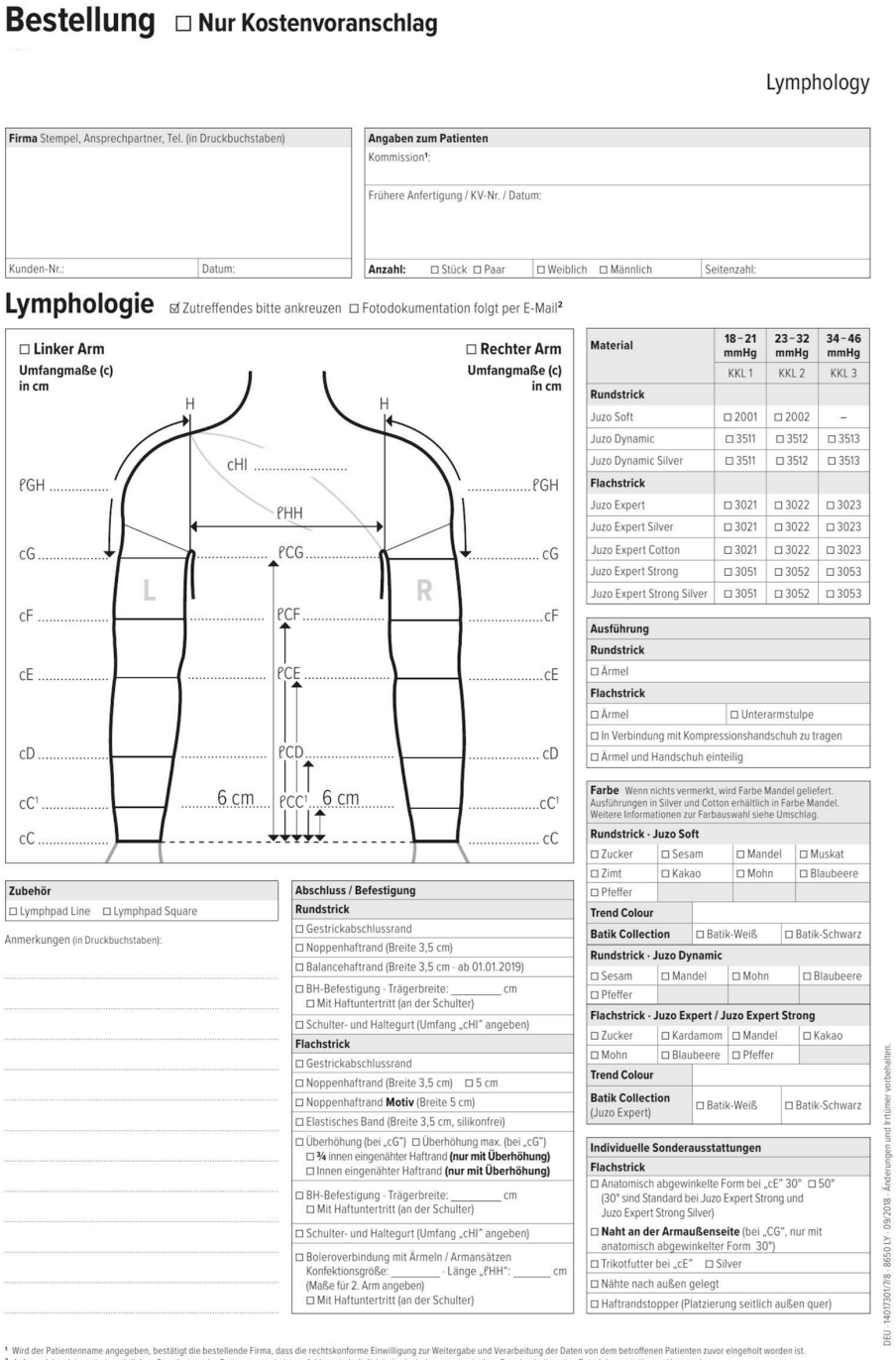


Figure A.7: Confusion matrix to analyze the performance of the classifier trained on faxified document images and evaluated using real annotated document images (test dataset).

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

Material	**18–21 mmHg**	**23–32 mmHg**	**34–46 mmHg**
	KKL 1	KKL 2	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):

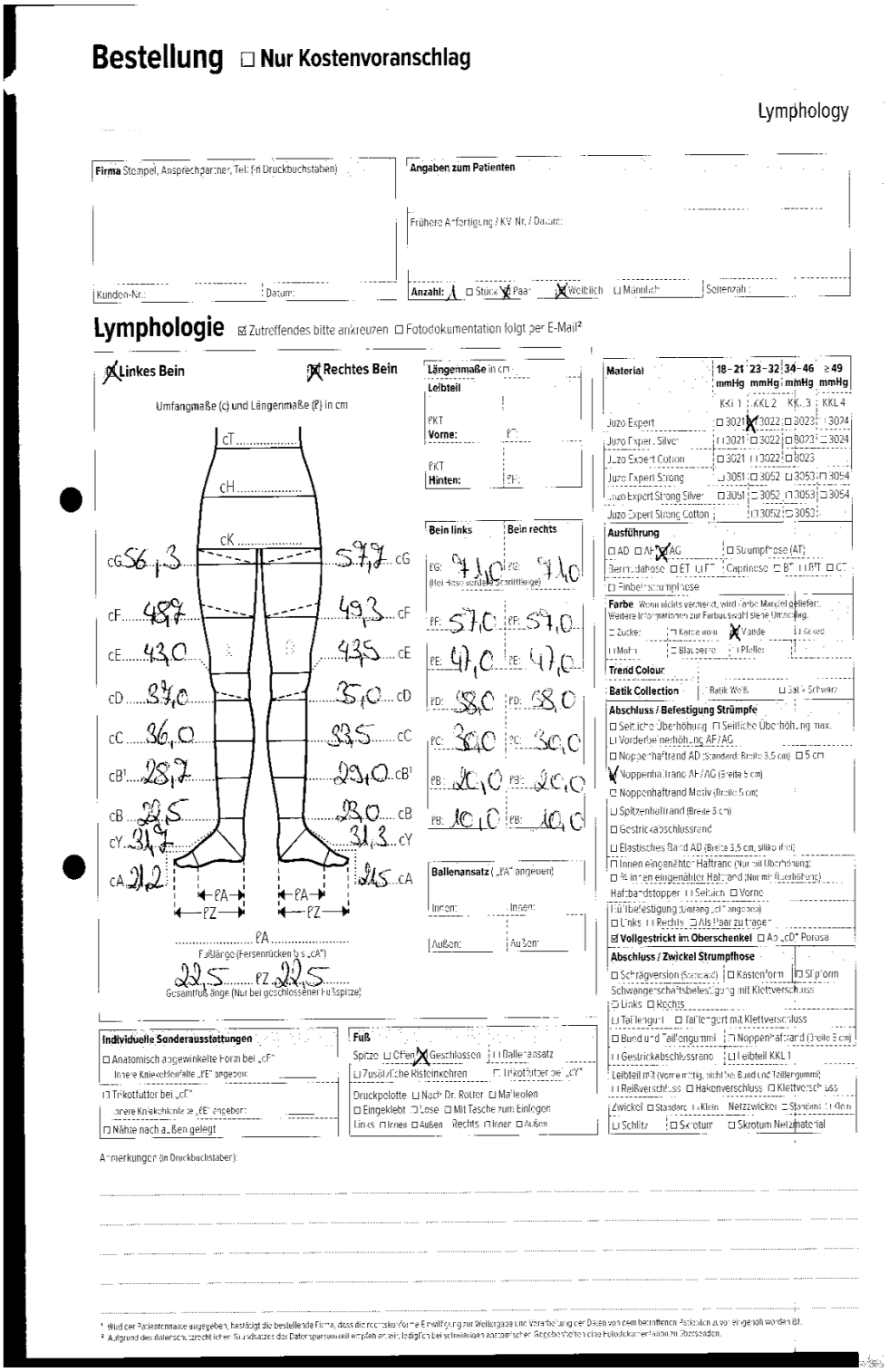
Abschluss / Befestigung
Rundstrick
☐ Gestrickabschlussrand
☐ Noppenhaftrand (Breite 3,5 cm)
☐ Balancehaftrand (Breite 3,5 cm · ab 01.01.2019)
☐ BH-Befestigung · Trägerbreite: _____ cm ☐ Mit Haftuntertritt (an der Schulter)
☐ Schulter- und Haltegurt (Umfang „cHI" angeben)
Flachstrick
☐ Gestrickabschlussrand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

[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.

DEU · 1407/2017/78 · 8650 LY · 09/2018 · Änderungen und Irrtümer vorbehalten.

Figure A.8: Examples of template (figure reproduced from elevait GmbH & Co. KG with permission).



Bestellung ☐ Nur Kostenvoranschlag

Lymphology

Lymphologie ☒ Zutreffendes bitte ankreuzen ☐ Fotodokumentation folgt per E-Mail²

☒ Linkes Bein ☒ Rechtes Bein

Umfangmaße (c) und Längenmaße (ℓ) in cm

Individuelle Sonderausstattungen

Fuß

Material

Ausführung

Farbe

Trend Colour

Batik Collection

Abschluss / Befestigung Strümpfe

Abschluss / Zwickel Strumpfhose

Anmerkungen (in Druckbuchstaben):

¹ Wird der Patientenname angegeben, bestätigt die bestellende Firma, dass die notwendige Einwilligung zur Weitergabe und Verarbeitung der Daten von dem betroffenen Patienten vorliegt.

² Aufgrund des äußerst geringen Strahlenrisikos der Datenspeicherung wird empfohlen, lediglich bei schwierigen anatomischen Gegebenheiten eine Fotodokumentation zu überspielen.

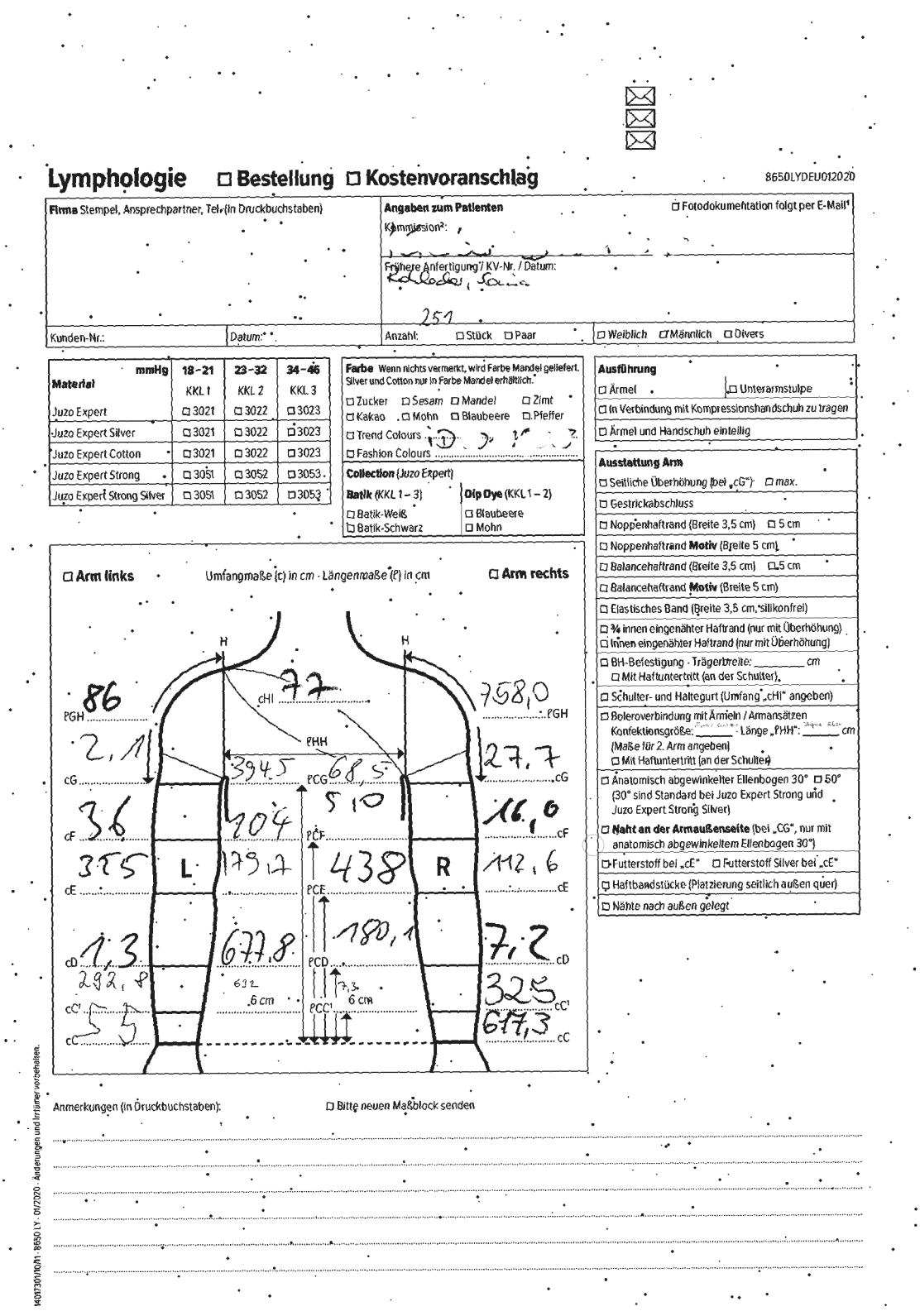


Figure A.10: Examples of faxified document image (figure reproduced from elevait GmbH & Co. KG with permission).

A.6 Classifier Architecture Diagram

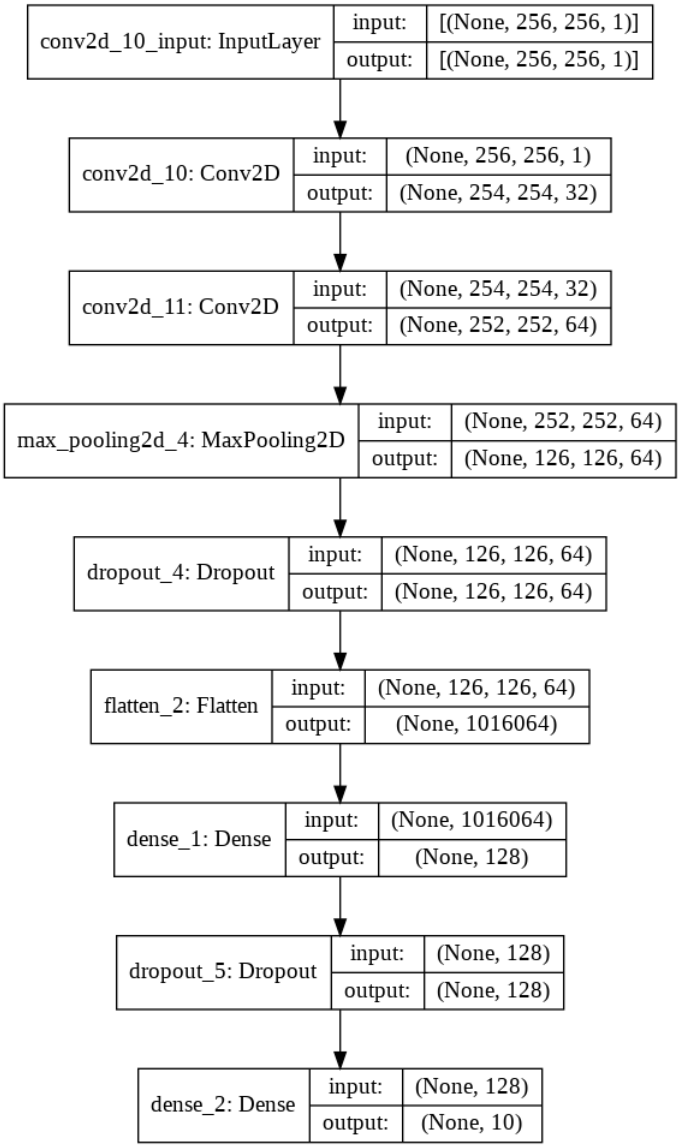


Figure A.11: Classifier model summary.

A.7 Generator Model Summary

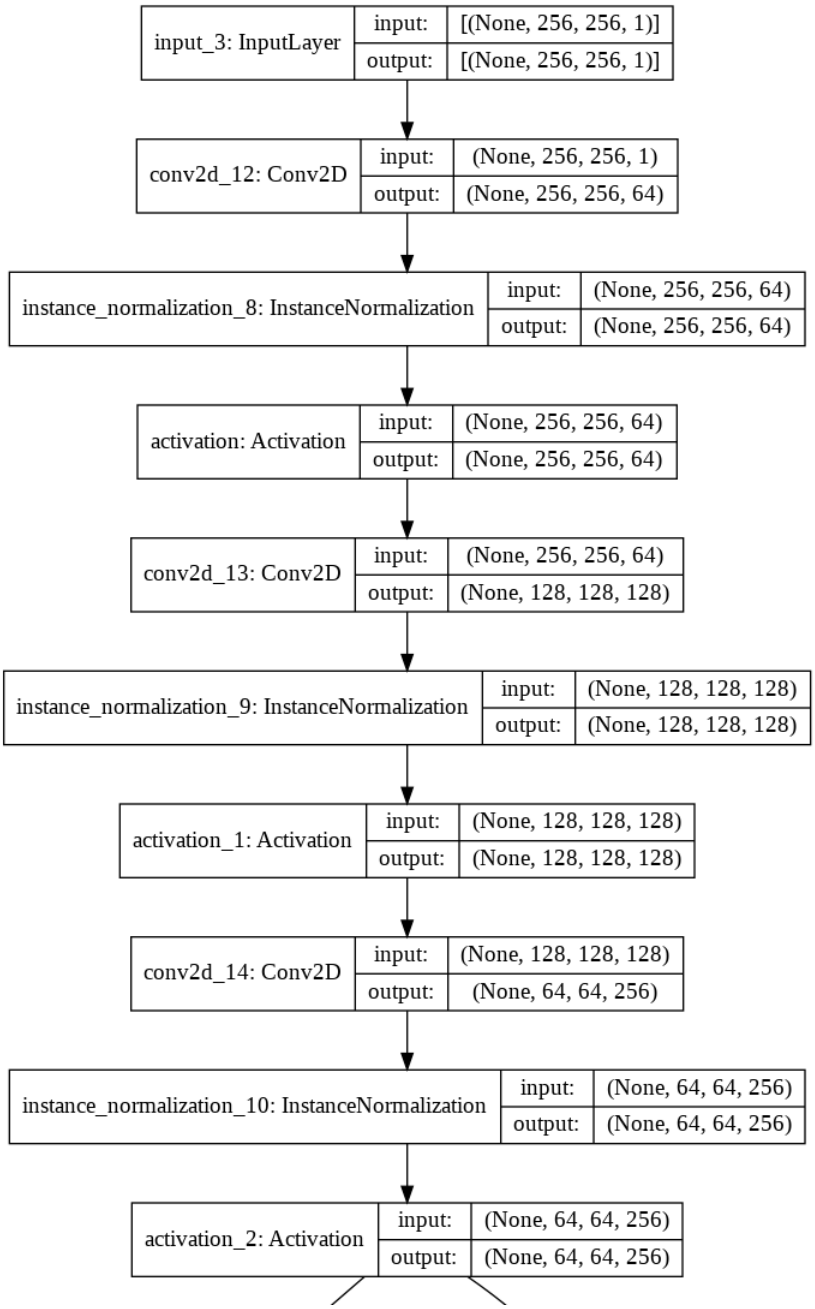


Figure A.12: Generator model summary. Continue to next page.

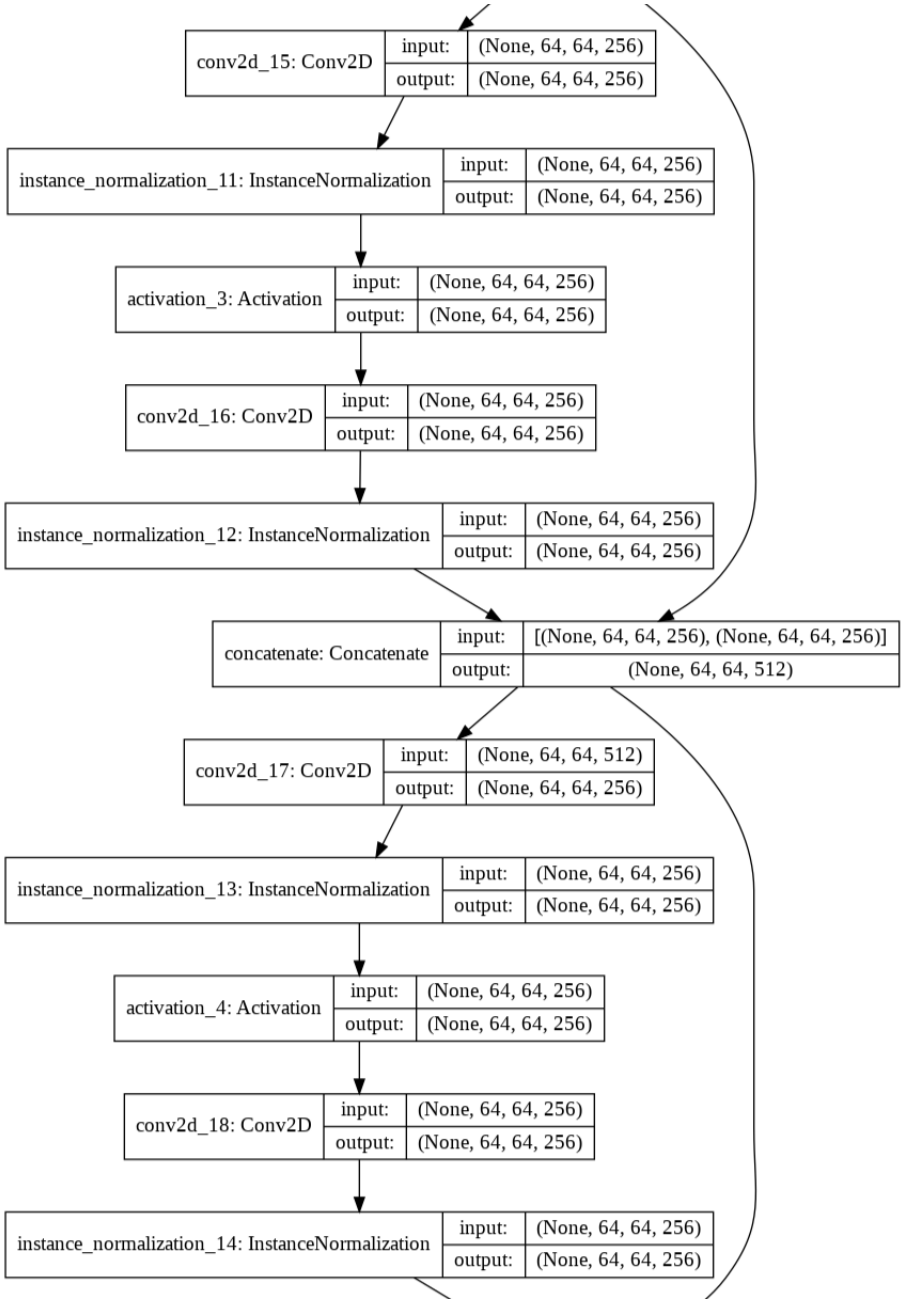


Figure A.13: Generator model summary. Continue to next page.

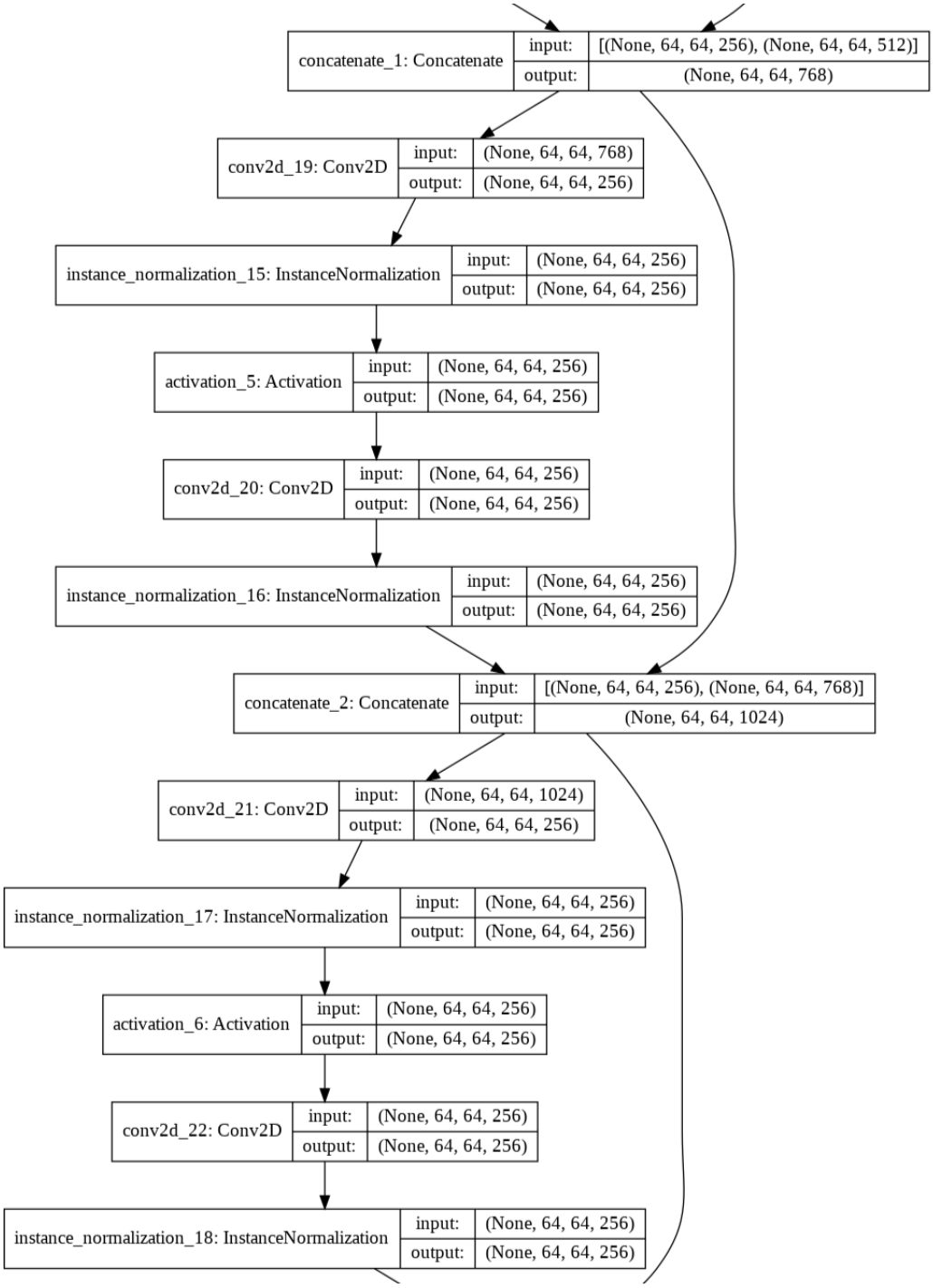


Figure A.14: Generator model summary. Continue to next page.

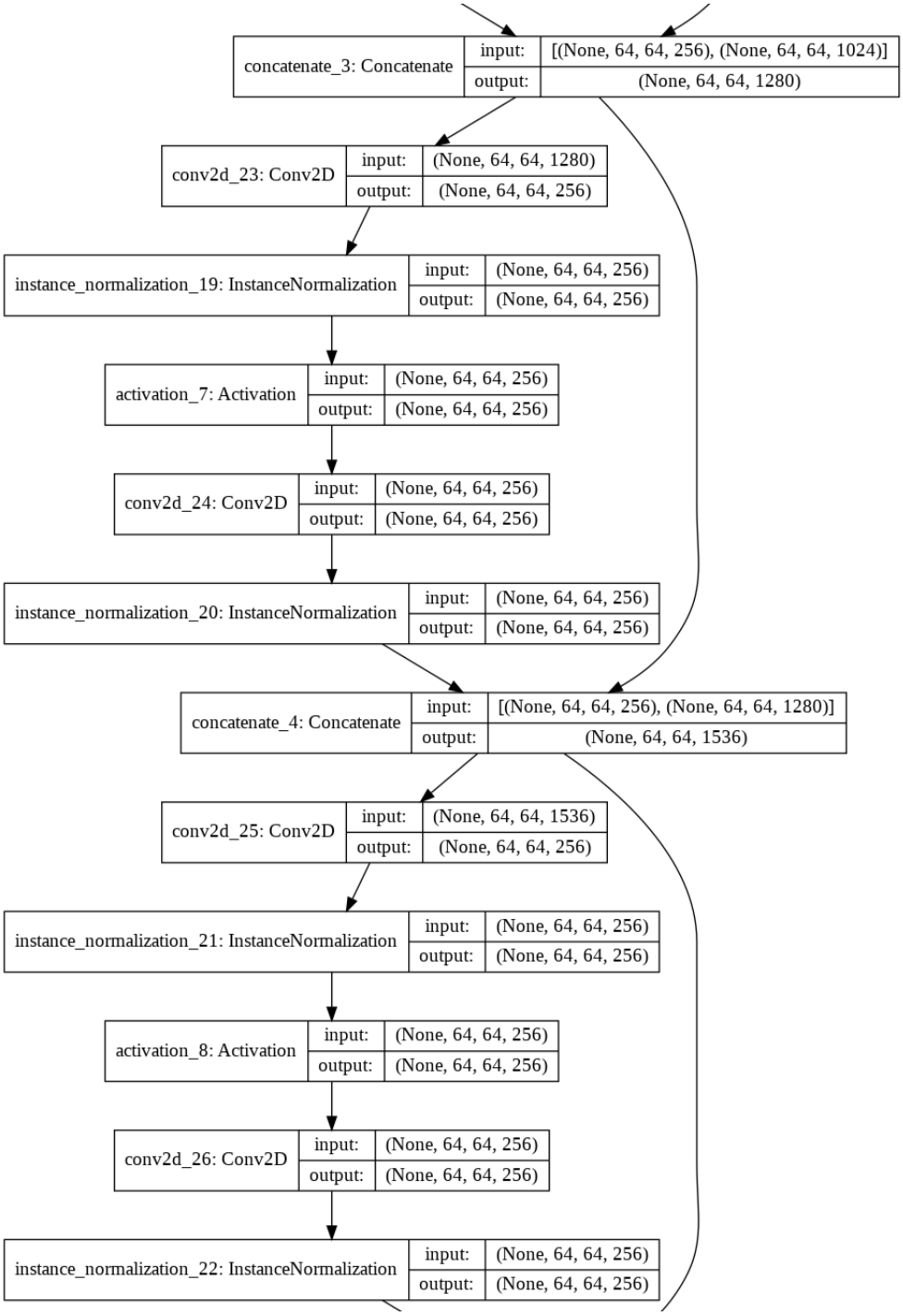


Figure A.15: Generator model summary. Continue to next page.

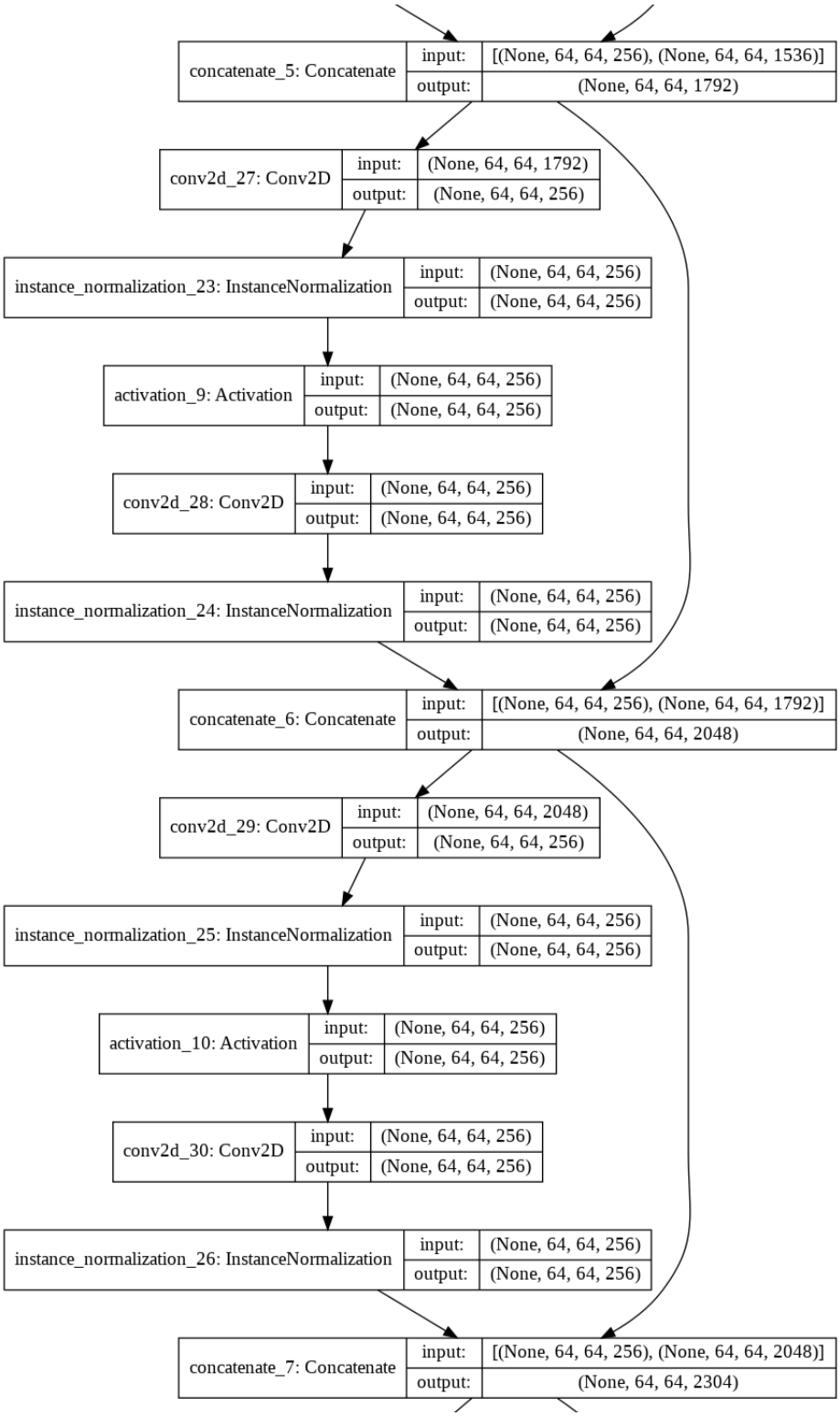


Figure A.16: Generator model summary. Continue to next page.

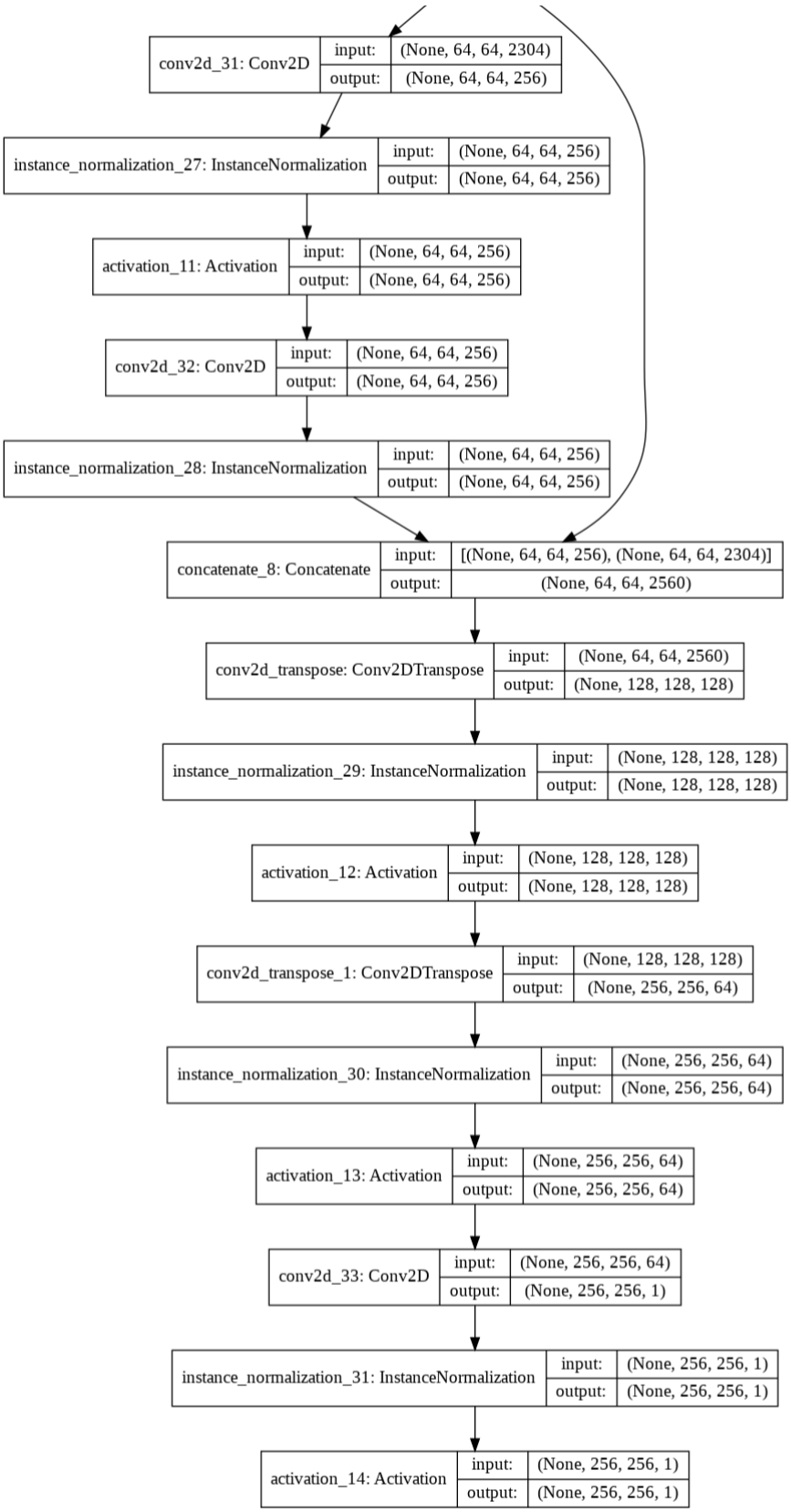


Figure A.17: Generator model summary. Ends here.

A.8 Discriminator Model Summary

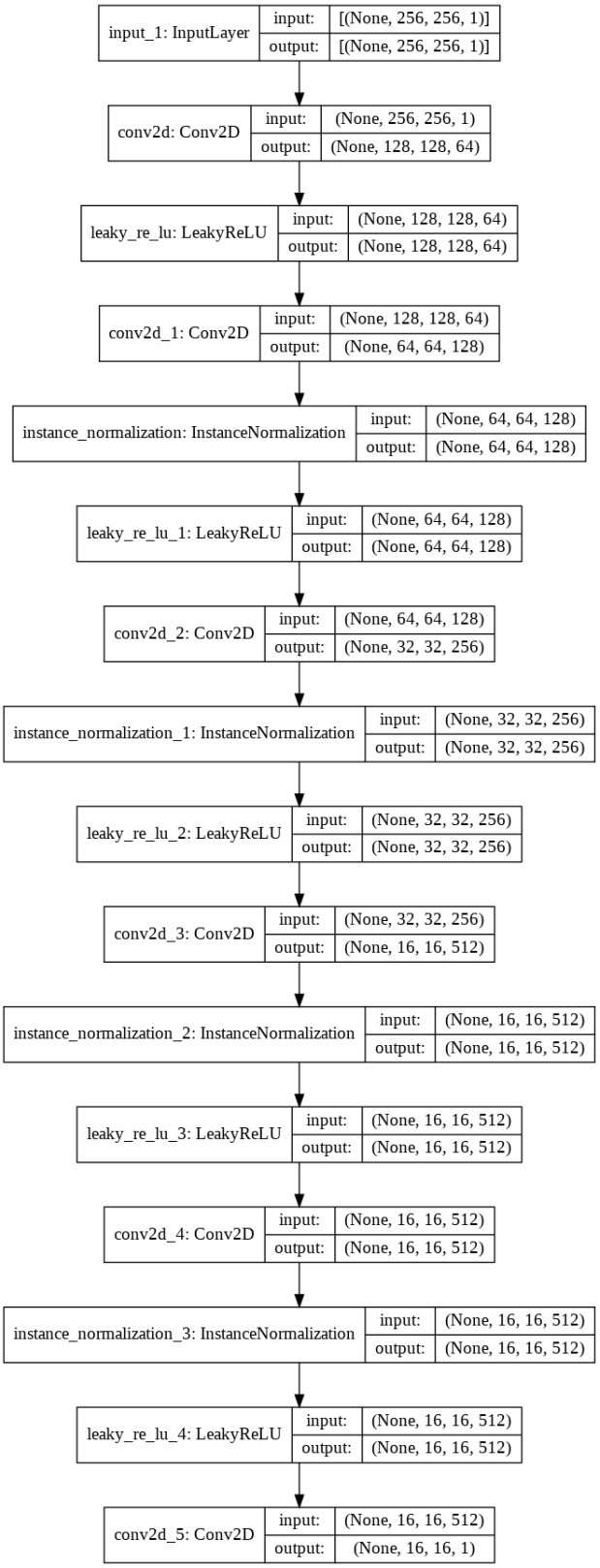


Figure A.18: Discriminator model summary.

Bibliography

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

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

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

[4] 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.

[5] 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.

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

[7] 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.

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

[10] 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.

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

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

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

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

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

[16] 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.

[17] 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.

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

[19] 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.

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

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

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

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

[24] 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.

[25] Yingxue Pang, Jianxin Lin, Tao Qin, and Zhibo Chen. Image-to-image translation: Methods and applications, 2021.

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

[27] 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.

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

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

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

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

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

[33] 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.

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

[35] 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.

[36] 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.

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

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

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

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

[41] 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.

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

[43] 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.

[44] Wesley Peisch, Luca Pistor, and Samarpreet Singh Pandher. Colorizing grayscale paintings using cgans, 2021.

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

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

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

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

[49] Zhirong Wu, Yuanjun Xiong, Stella X. Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance discrimination. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3733–3742, 2018.

[50] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning, 2020.

[51] Olivier J. Hénaff, Aravind Srinivas, Jeffrey De Fauw, Ali Razavi, Carl Doersch, S. M. Ali Eslami, and Aaron van den Oord. Data-efficient image recognition with contrastive predictive coding, 2020.

[52] Philip Bachman, R Devon Hjelm, and William Buchwalter. Learning representations by maximizing mutual information across views, 2019.

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

[54] 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.

[55] 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.

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

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

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

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

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

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

[62] 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.

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

[64] 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.

[65] 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.

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

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

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

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

[70] 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.

[71] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In Konstantinos Diamantaras, Wlodek Duch, and Lazaros S. Iliadis, editors, *Artificial Neural Networks – ICANN 2010*, pages 92–101, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[72] Ashwani Kumar. Ordinal pooling networks: For preserving information over shrinking feature maps, 2018.

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

[74] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks, 2017.

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

[76] 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.

[77] 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.

[78] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network, 2017.

[79] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[80] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization, 2017.

[81] David M. W. Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation, 2020.

[82] Meysam Vakili, Mohammad Ghamsari, and Masoumeh Rezaei. Performance analysis and comparison of machine and deep learning algorithms for iot data classification, 2020.

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

[84] Abolfazl Farahani, Sahar Voghoei, Khaled Rasheed, and Hamid R. Arabnia. A brief review of domain adaptation, 2020.

[85] Yihao Zhao, Ruihai Wu, and Hao Dong. Unpaired image-to-image translation using adversarial consistency loss, 2021.

[86] Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

TECHNISCHE UNIVERSITÄT CHEMNITZ

Studentenservice – Zentrales Prüfungsamt

Selbstständigkeitserklärung

Name: Pawar Vorname: Giriraj Sukumar geb. am: 13.04.1993 Matr.-Nr.: 551205	**Bitte beachten:** 1. Bitte binden Sie dieses Blatt am Ende Ihrer Arbeit ein.

Selbstständigkeitserklärung*

Ich erkläre gegenüber der Technischen Universität Chemnitz, dass ich die vorliegende Masterarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe.

Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch nicht als Prüfungsleistung eingereicht und ist auch noch nicht veröffentlicht.

Datum: 31.08.2021

Unterschrift: .....

* Statement of Authorship

I hereby certify to the Technische Universität Chemnitz that this thesis is all my own work and uses no external material other than that acknowledged in the text.

This work contains no plagiarism and all sentences or passages directly quoted from other people's work or including content derived from such work have been specifically credited to the authors and sources.

This paper has neither been submitted in the same or a similar form to any other examiner nor for the award of any other degree, nor has it previously been published.