TECHNISCHE UNIVERSITÄT CHEMNITZ

Faculty of Electrical Engineering and Information Technology

Professorship of Digital Signal Processing and Circuit Technology

Master Thesis

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

Giriraj Sukumar Pawar

Chemnitz, May 7, 2021

Supervisor: Prof. Dr.-Ing. Gangolf Hirtz
Advisor: Tobias Scheck M.Sc. and Paul Fischer M.Sc.

Pawar, Giriraj Sukumar
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, May 2021.

Abstract

This thesis presents a method for unsupervised domain adaptation
Keywords:

Acknowledgements

Contents

List of Figures

List of Tables

List of Abbreviations

GAN	Generative Adversarial Network
CNN	Convolutional Neural Network
cGAN	Conditional Adversarial Network
LSGAN	Least Squares Generative Adversarial Network
CycleGAN	Cycle-Consistent Adversarial Network
GT	Ground Truth
DIBCO	Document Image Binarization Competition
PSNR	Peak Signal-to-Noise Ratio
ResNet	Residual Network
AMT	Amazon Mechanical Turk
CUT	Contrastive Unpaired Translation
MUNIT	Multimodal Unsupervised Image-to-image Translation
DRIT	Diverse Image-to-Image Translation
GCGAN	Geometry-Consistent Generative Adversarial Networks
FastCUT	Fast Contrastive Unpaired Translation
FID	Fréchet Inception Distance
HTR	Handwritten Text Recognition
OCR	Optical Character Recognition
WGAN	Wasserstein GAN

1. Introduction

In recent years, deep learning methods have shown great effectiveness in many fields, including natural language processing and computer vision. However, such kinds of methods are still limited by a poor generalization due to the insufficient quantity of training data [TBSM19]. Annotated data are scarce when it comes to developing deep learning models for computer vision applications. The performance of such deep learning models can be improved with the introduction of a large amount of annotated data. Though, it is hard to obtain, due to the high cost of data annotation, specifically in the case of numerous variants of data. To overcome the obstacle of the scarcity of annotated data, there are some methods available to tackle this problem. The popular methods are Active Learning [HKS20], Data Augmentation [SK19], Transfer Learning [ZQD+20], and Domain Adaptation [RMH+20].

1.1 Motivation

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. Nowadays, the domain adaptation technique is widely used in the field of Handwritten Text Recognition (HTR) and Optical Character Recognition (OCR). To make the HTR and OCR systems robust and efficient requires a large quantity of diverse data. As mentioned earlier there is a scarcity of annotated data. For example, filled forms (figure A.2) with different types of handwriting. In such cases, machine learning engineers have to inevitably generate synthetic data. However, deep learning models trained using synthetic data will not generalise well on real data [TBSM19]. The synthetic data lacks realism. It does not possess a similar noise distribution as real data. Hence, in the last two decades, numerous domain adaptation methodologies have been introduced. Such methods are used to transform synthetic data into realistic data by reducing the divergence between the distribution of real data and the distribution of synthetic data.

1.2 Problem Statement

In this thesis, a domain adaptation application is developed using CycleGAN [ZPIE20] to reduce the domain gap between synthetic images and real images. The CycleGAN is an extended variant of Generative Adversarial Network (GAN) [GPAM+14]. This application is designed in consultation with, ML developers at AI4BD, a Germany-based company that develops AI applications for business use-cases. AI4BD is widely contributing in the field of Cognitive Business Robotics [MC12] to automate document processing. AI4BD has developed state-of-the-art HTR and OCR tools to process documents and extract information from documents. To make those systems robust and efficient a large number of document images are required. The idea is to create a large number of synthetic document images to have a large quantity of annotated data. However, the synthetic document images will not generalise well when they have to process real document images because the real document images have different noise distribution. Also, The scanning process often results in the introduction of artifacts such as salt-and-pepper, background noise, blur due to camera motion or shake, watermarks, stains, wrinkles, and the faded text. Hence this application is used to transform synthetic document images into realistic document images. This application is capable to capture the noise distribution of real documents and transform an image from the source domain to the target domain. Such a kind of transformation is called image-to-image translation.

This image-to-image translation application trades with synthetically generated document images. The synthetic document images are created using empty template images and handwritten crops (figure 5.2) retrieved from handwriting datasets like MNIST (figure 5.4) or any other datasets. Handwritten crops are inserted over the empty template images to generate numerous synthetic document images. As mentioned earlier deep learning models trained using synthetic document images

will not generalise well on real document images. The objective of this thesis is to use CycleGAN and develop an image-to-image translation application to reduce the domain gap between synthetic document images and real document images. Because CycleGANs are the state-of-the-art for unpaired image-to-image translation. In the absence of paired document images, the CycleGAN can transform synthetic document images into realistic document images. A large number of realistic document images can be generated. This will save data annotation and data collection efforts. Further, these realistic documents can be used to train deep learning models and make the HTR and OCR tools robust and efficient.

1.3 Thesis Structure and Limitations

The domain adaptation field is promoting incremental findings. Hence, this thesis has its limitations due to time deadlines. This image-to-image application is implemented only using CycleGANs. The implementation and comparison with other methodologies are placed apart for future work. The handwriting datasets and document images datasets are can not be disclosed or cited due to data privacy concerns and copyright issues. The rest of this thesis is organized as follows. Chapter 2 briefly reviews related works of numerous GANs variants. Chapter 3 discusses the working principle of GANs and Convolution Neural Networks (CNNs). The decided approach is described in Chapter 4, the implementation and experimental results are presented in Chapter 5. Finally, this work has been concluded in Chapter 6 along with the future work and the limitations.

2. Related Works

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

2.1 Literature Survey

Ian J. Goodfellow et al. [GPAM+14] proposed a framework of GANs in which two models are simultaneously trained. A generative model that captures the data distribution, and a discriminative model that estimates the probability that a sample came from the training data rather than a generative model. The training procedure for a generative model is to maximize the probability of a discriminative model making a mistake. Basically, the generator learns to generate plausible data. The discriminator learns to distinguish the generator's fake data from real data. The discriminator penalizes the generator for producing implausible results. When training begins, the generator produces fake data, and the discriminator quickly learns to tell that it's fake and the generator penalized to produce plausible results. As training progresses, the generator gets closer to producing output that can fool the discriminator. Finally, if generator training goes well, the discriminator gets worse at telling the difference between real and fake. At the end of the training, eventually, we have a generator model which produces plausible results which are similar to real data. Authors have trained GAN an a range of datasets including MNIST [LBBH98], the Toronto Face Database (TFD) [SAH10], and CIFAR-10 [KH+09]. Also compared against already existing methods like DBN [BMDR12], Stacked CAE [BMDR12], and Deep GSN [BTLAY14] . The authors do not claim that the samples generated by GANs are better than samples generated by already existed methods. Authors believe that these samples are at least competitive with the better generative models in the literature and highlight the potential of the generative adversarial framework. The advantage of using GANs is primarily computational. Adversarial models may also gain some statistical advantage from the generator network not being updated directly with data examples, but only with gradients flowing through the discriminator using backpropagation. Special care should be taken during training the GANs, the generator must not be trained too much without updating the discriminator, to avoid the Helvetica Scenario [MG19] in which the generator collapses to produce the same output (or a small set of outputs) over and over again. Usually, GANs should produce a wide variety of outputs. The Helvetica Scenario is also called Mode Collapse [TTT20].

Xudong Mao et al. [MLX+17] proposed another variant of GANs called Least Squares Generative Adversarial Networks (LSGANs). The Regular GANs hypothesize the discriminator as a classifier with the sigmoid cross-entropy loss function. However, they found that the cross-entropy loss function may lead to the vanishing gradients problem during the learning process. To overcome such a problem, the authors proposed the LSGANs which adopts the least-squares loss function for the discriminator. The least-squares loss function penalizes the fake samples and forces the generator to generate samples toward the decision boundary. The authors evaluated LSGANs using two datasets LSUN [YSZ+16] and HWDB1.0 (Handwritten Chinese Character Dataset) [LYWW11]. When trained on the LSUN dataset [YSZ+16] they observed, the images generated by LSGANs are of better quality than the ones generated by the two baseline methods, DCGANs [RMC16] and EBGANs [ZML17] . Also, when trained on the Handwritten Chinese Character Dataset, the generated characters were readable and clear. Another experiment was conducted to evaluate the stability of LSGAN on a Gaussian mixture distribution dataset, which is designed in literature [MPPSD17]. 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. In this paper, numerous comparison experiments for evaluating the stability are conducted and the results demonstrate that LSGANs can generate higher quality images than regular GANs, DCGANs [RMC16], and EBGANs [ZML17] and perform more stable than regular GANs during the learning process.

Phillip Isola et al. [IZZE18] proposed Conditional Adversarial Networks (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 [COR+16], reconstructing objects from edge maps [ZKSE18] [YG14] , and colorizing images. cGANs not only learn the mapping from input image to output image, but also learn a loss function to train this mapping. This makes it possible to apply the same generic approach to problems that traditionally would require very different loss formulations. Authors implemented cGAN and released it as the pix2pix software to solve distinct image-to-image translation problems. This software is popular among a large number of internet users, many of them are artists, because of its wide applicability and ease of adoption without the need for parameter tweaking. In cGAN the generator uses U-Net-based architecture [RFB15], the U-Net is an encoder-decoder with skip connections between mirrored layers in the encoder and decoder stacks. The discriminator uses a convolutional PatchGAN classifier [LW16], which only penalizes structure at the scale of image patches. Unlike unconditional GANs, both the generator and discriminator observe the input images. Authors performed multiple experiments during ablation studies using evaluation metrics like, Amazon Mechanical Turk (AMT) perceptual study, FCN-Score, and semantic segmentation metrics. They found that L1 distance loss encourages less blurring compared to the L2 distance loss. Also combining L1 Loss and cGAN (L1 + cGAN) generates better results compared to combining Unconditional GAN and L1 Loss ((L1 + GAN)). The cGAN appears to be more effective on the problem where the output is highly detailed or photographic. The pix2pix software code is available at GitHub.

Taesung Park et al. [PEZZ20] proposed a framework for encouraging content preservation in unpaired image-to-image translation problems by maximizing the mutual information between input and output with patchwise contrastive learning [vdOLV19]. In the patchwise contrastive learning for image-to-image translation, a generated output patch should appear closer to its corresponding input patch in comparison to other random patches present in the same input. To achieve patchwise contrastive learning, drawing patches internally from within the input image, rather than externally from other images in the dataset, forces the patches to better preserve the content of the input. This method requires neither a memory bank nor specialized architecture. The authors demonstrated that the framework enables one-sided translation in the unpaired image-to-image translation setting while improving quality, consuming less memory, and reducing training time. They call this approach Contrastive Unpaired Translation (CUT). Since contrastive representation is formulated within the same image, this method can even be trained on single images, where each domain is having only a single image. The several prior methods like CycleGAN [ZPIE20], Multimodal Unsupervised Image-to-image Translation (MUNIT) [LBK18], Diverse Image-to-Image Translation (DRIT) [LTM+19], and Geometry-Consistent Generative Adversarial Networks (GCGAN) [FGW+18] 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. The evaluation metrics like Fréchet Inception Distance (FID) [HRU+18] and semantic segmentation scores are used to compare the quality of generated images using CUT method. Furthermore, the authors also introduced faster and lighter variant Fast Contrastive Unpaired Translation (FastCUT). FastCUT also produces competitive results with even lighter computation cost of training. The code and models for CUT are available at GitHub.

Monika Sharma et al. [SVV19] is addressing a problem in the scanning process that often results in the introduction of salt and pepper noise, blur due to camera motion, or shake, water markings, coffee stains, wrinkles, or faded data. These artifacts pose many readability challenges to current text recognition algorithms and significantly degrade their performance. So, the existing denoising techniques require a dataset comprising of noisy documents paired with cleaned versions of the same document. However, very often in the real world, such a paired dataset is not available to train a model to generate clean documents from noisy versions. Hence, the authors proposed to use CycleGAN because it is known to learn a mapping between the distributions in the absence of

paired training dataset. Using CycleGAN, noisy document images transformed into denoised and clean document images to achieve image-to-image translation. They have compared the performance of CycleGAN for document cleaning tasks with a cGAN by training them over the same dataset. The only difference was CycleGAN trained using unpaired images and cGAN trained using the paired images. They have used Peak Signal-to-Noise Ratio (PSNR)[1] as a evaluation metric to evaluate the quality of transformed denoised images. Several experiments were performed on 4 separate document public document datasets, one each for background noise removal, deblurring, watermark removal, and defading. Finally, they illustrate that CycleGAN learns a more robust mapping from the space of noisy to clean documents compared to cGAN.

Chris Tensmeyer et al. [TBSM19] realized solving binarization tasks using deep learning models is very challenging. It is due to the lack of large quantities of labeled data available to train such models. They also mention there have been efforts to create synthetic data for binarization using image processing techniques but, they generally lack realism. In this paper, the authors proposed a method to produce realistic synthetic data using an adversarially trained image translation model. They extended the popular CycleGAN model to be conditioned on the ground truth binarization mask as it translates images from the domain of synthetic images to the domain of real images. They have found that modifying the discriminator to condition on the binarization Ground Truth (GT) leads to increased realism and better agreement between the GT and the produced image. They called the proposed model DGT-CycleGAN. Also shown DGT-CycleGAN model produces more realistic synthetic data. They validated their approach by pretraining deep networks on realistic synthetic datasets generated by DGT-CycleGAN, CycleGAN, and image compositing. They evaluate both pretrained only and finetuned models on each of the Document Image Binarization Competition (DIBCO) datasets. They have concluded that pretraining deep neural networks on the more realistic synthetic data generated using DGT-CycleGAN lead to better predictive performance both before and after finetuning on real data.

Jun-Yan Zhu et al. [ZPIE20] proposed a modified version of GAN which is the state-of-the-art method for the image-to-image translation that can transform the images from source domain X to target domain Y in the absence of paired training dataset. This method can learn to capture special characteristics of one image collection and figuring out how these characteristics could be translated into the other image collection, all in the absence of any paired training examples. This modified version of GAN is called CycleGAN. In this method, the goal is to learn a mapping $G : X \rightarrow Y$ such that the distribution of images $G(X)$ is indistinguishable from the distribution Y using an adversarial loss. Because this mapping is highly under-constrained and coupled with an inverse mapping $F : Y \rightarrow X$ to introduce a cycle consistency loss to enforce $F(G(X)) \approx X$ and vice versa. Along with an adversarial loss and cycle consistency loss, this work also introduces identity mapping loss which helps to preserve the colour of input images. Authors considered evaluation metrics like AMT perceptual studies, FCN-score [IZZE18], and semantic segmentation metrics to compare the quality of generated images against other baseline. The several prior methods like Bi-GAN/ALI [[DKD17], [DBP$^+$17]], CoGAN [LT16], SimGAN [SPT$^+$17] 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 [IZZE18]. Authors provide both PyTorch and Torch implementations.

2.2 Discussion

GANs suffer from unstable training, vanishing gradients problem, and mode collapse. Consequently, Martin Arjovsky et al. [ACB17] proposed Wasserstein GAN (WGAN) to solve problems with GANs. WGANs improve the stability of learning, get rid of problems like mode collapse, and provide meaningful learning curves useful for debugging and hyperparameter searches. Although the implementation of WGAN is straightforward, the theory behind it is heavy and requires some hack, for example, Weight Clipping [GAA$^+$17]. Hence, Xudong Mao et al. [MLX$^+$17] proposed a simple and more intuitive method compared to WGAN, called LSGAN. First, LSGANs are able to generate higher quality images than regular GANs. Second, LSGANs perform more stable during the learning process. Moreover, Jun-Yan Zhu et al. [ZPIE20] proposed CycleGAN. It is an unsupervised image-to-image translation approach. Compared to GANs, CycleGANs can deal more

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

meticulously with the problems like unstable training, vanishing gradients problems, and mode collapse. Also, they described that CycleGAN has outperformed existing baselines as Bi-GAN/ALI [[DKD17], [DBP+17]], CoGAN[LT16], and SimGAN [SPT+17]. Furthermore, Monika Sharma et al. [SVV19] proclaimed CycleGAN can transform noisy document images into denoised and clean document images to achieve image-to-image translation. They have also compared the performance of the developed image-to-image application with cGAN and demonstrated CycleGAN had outperformed cGAN. Chris Tensmeyer et al. [TBSM19] proposed DGT-CycleGAN, a modified version of CycleGAN, which is adequate to translate images from the domain of synthetic images to the domain of real images to solve binarization tasks. Moreover, Taesung Park et al. [PEZZ20] proposed CUT. They have demonstrated that CUT is a better, faster, and memory-efficient approach to perform unsupervised image-to-image translation. It has outperformed CycleGAN [ZPIE20], MUNIT [LBK18], DRIT [LTM+19], and GCGAN [FGW+18]. However, due to time constraints, multiple available references, and code repositories, CycleGAN has been a choice for this thesis and research to perform unsupervised image-to-image translation. In this thesis, CycleGAN is combined with LSGAN, in which the discriminator model is updated using a least-squares loss (L2 Loss). Discussion about the loss functions has been carried out in-depth in Chapter 4.

2.3 Conclusion

The image-to-image translation is a class of computer vision and computer graphics problems. In which the goal is to learn the mapping between a source image and target image using a training set of aligned image pairs. Although, for many tasks, aligned or paired training data will not be available. This thesis is striving to close a domain gap between synthetic document images and real images in the absence of paired training data. Collecting and annotating paired document images is gruelling, time-consuming, and costly. The thesis aims to develop an image-to-image translation application to transform synthetic document images into realistic document images. Ultimately, a large amount of realistic annotated set of document images can be generated using this application. As per the above literature survey and problem statement, CycleGAN is a remarkable approach to transform synthetic document images into realistic document images in the absence of paired training data. Hence, in this thesis, the image-to-image translation application is realized using CycleGAN.

3. Fundamentals

This chapter aims to develop a better understanding of fundamental concepts required to understand the thesis. It discusses two significant concepts, the working principle of GANs and CNNs. The basic architecture of the generator and discriminator are briefly explained in Section 3.1. The multiple layers like convolution layer, activation layer, pooling layer, and fully connected layer have briefly explained in Section 3.2.

3.1 Working Principle of GANs

The GAN has two parts one is a generator and the other is a discriminator. The generator learns to generate plausible data. The generated instances become negative training examples for the discriminator. The discriminator learns to distinguish the generator's fake data from real data. The discriminator penalizes the generator for producing implausible results. When training begins, the generator produces fake data, and the discriminator quickly learns to tell that it's fake. As training progresses, the generator gets closer to producing output that can fool the discriminator. Finally, if generator training goes well, the discriminator gets worse at telling the difference between real and fake. It starts to classify fake data as real, and its accuracy decreases. Eventually generator will start producing plausible data. Both the generator and the discriminator are neural networks. The generator output is connected directly to the discriminator input. Through backpropagation, the discriminator's classification provides a signal that the generator uses to update its weights.

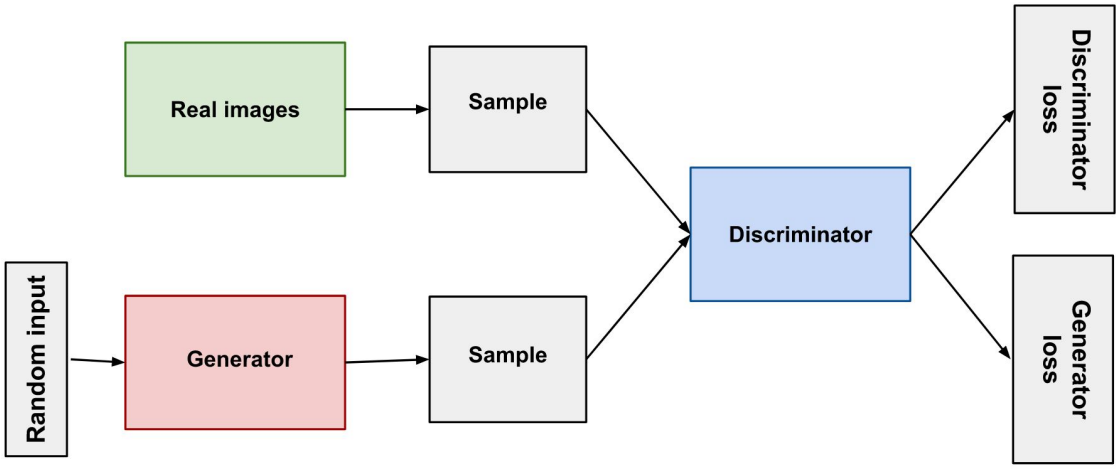


Figure 3.1: Overview of GAN Structure.[1]

3.1.1 The Discriminator

The discriminator in a GAN is simply a classifier. It tries to distinguish real data from the data created by the generator. It could use any network architecture appropriate to the type of data it's classifying. The discriminator's training data comes from two sources: a) Real Data Instances. The discriminator uses these instances as positive examples during training. b) Fake Data Instances created by the generator. The discriminator uses these instances as negative examples during training.

In figure 3.2, the two "Sample" boxes represent these two data sources feeding into the discriminator. When the discriminator is training, the generator does not train. Its weights remain constant while it produces examples for the discriminator to train on. The discriminator connects to two loss functions. The discriminator ignores the generator loss and just uses the discriminator loss during its training. The generator loss is used during generator training. The next Section 3.1.2 describes why the generator loss connects to the discriminator. Throughout discriminator training: a) The discriminator classifies both real data and fake data from the generator. b) The discriminator loss penalizes the discriminator for misclassifying a real instance as fake or a fake instance as real. c) The discriminator updates its weights through backpropagation from the discriminator loss through the discriminator network.

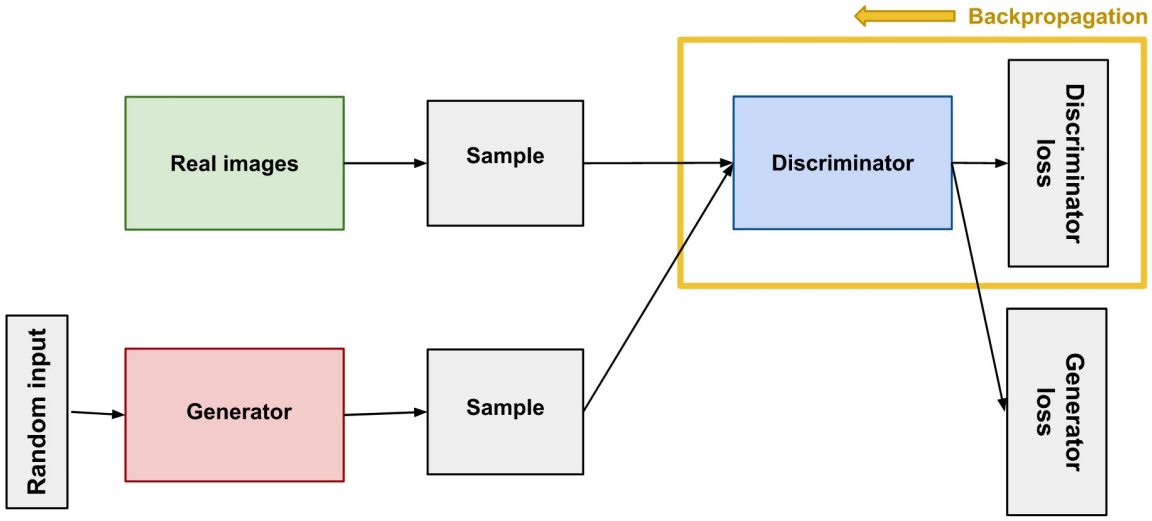


Figure 3.2: Backpropagation in Discriminator Training.[1]

3.1.2 The Generator

The generator part of a GAN learns to create fake data by incorporating feedback from the discriminator. It learns to make the discriminator classify its output as real. Generator training requires tighter integration between the generator and the discriminator than discriminator training requires. The portion of the GAN that trains the generator includes: a) Random input. b) Generator network, which transforms the random input into a data instance. c) Discriminator network, which classifies the generated data. d) Discriminator output. e) Generator loss, which penalizes the generator for failing to fool the discriminator.

Neural networks need some form of input data, like an instance for classification or to predict about. But what to use as an input for a network that outputs entirely new data instances? In its most basic form, a GAN takes random noise as its input. The generator then transforms this noise into a meaningful output. By introducing noise, GAN can produce a wide variety of data, sampling from different places in the target distribution. Experiments suggest that the distribution of the noise doesn't matter much, so it is possible choose something easy to sample from, as a uniform distribution. For convenience, space from which the noise is sampled is usually of a smaller dimension than the dimensionality of the output space.

To train a neural network, the net's weights altered to reduce the error or loss of its output. In GAN, however, the generator is not directly connected to the loss that we're trying to affect. The generator feeds into the discriminator net, and the discriminator produces the output we're trying to affect. The generator loss penalizes the generator for producing a sample that the discriminator network classifies as fake. This extra chunk of network must be included in backpropagation. Backpropagation adjusts each weight in the right direction by calculating the weight's impact on the output — how the output would change if you changed the weight. But the impact of a generator

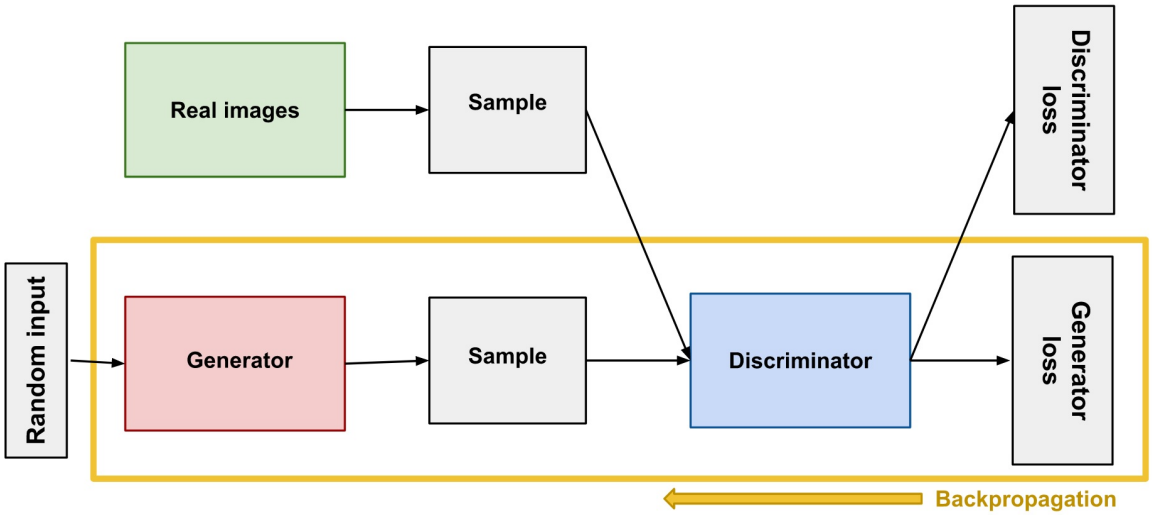


Figure 3.3: Backpropagation in Generator Training.[1]

weight depends on the impact of the discriminator weights it feeds into. So backpropagation starts at the output and flows back through the discriminator into the generator. At the same time, we don't want the discriminator to change during generator training. Trying to hit a moving target would make a hard problem even harder for the generator. The generator is trained with the following procedure: 1) Sample random noise. 2) Produce generator output from sampled random noise. 3) Get discriminator "Real" or "Fake" classification for generator output. 4) Calculate loss from discriminator classification. 5) Backpropagate through both the discriminator and generator to obtain gradients. 6) Use gradients to change only the generator weights.

3.2 Convolution Neural Networks

A tremendous interest in deep learning has emerged in recent years [LBH15]. The most established algorithm among various deep learning models is CNNs, a class of artificial neural networks that has been a dominant method in computer vision tasks since the astonishing results were shared on the object recognition competition known as the ImageNet Large Scale Visual Recognition Competition (ILSVRC) in 2012 [[RDS+15], [KSH12a]]. CNN is a type of deep learning model for processing data that has a grid pattern, such as images, which is inspired by the organization of animal visual cortex [[HW68], [Fuk80]] and designed to automatically and adaptively learn spatial hierarchies of features, from low-level to high-level patterns. It is composed of multiple building blocks, such as convolution layers[2], pooling layers, and fully connected layers. The first two, convolution and pooling layers, perform feature extraction, whereas the third, a fully connected layer, maps the extracted features into final output, such as classification. Let us have a look into each layer in coming sections.

3.2.1 Convolution Layer

The convolution layer is a fundamental component of the CNN architecture that performs feature extraction, which typically consists of a combination of linear and nonlinear operations, i.e., convolution operation and activation function. It is the core building block of a CNNs that does most of the computational heavy lifting. A convolution layer plays a key role in CNN, which is composed of

[1] `https://developers.google.com/machine-learning/gan/training/` last access: 14.04.2021

[2] A convolution is a mathematical operation that slides one function over another and measures the integral of their pointwise multiplication. It has deep connections with the Fourier transform and the Laplace transform, and is heavily used in signal processing. Convolutional layers actually use cross-correlations, which are very similar to convolutions (see `https://homl.info/76` for more details) last access: 31.03.2021.

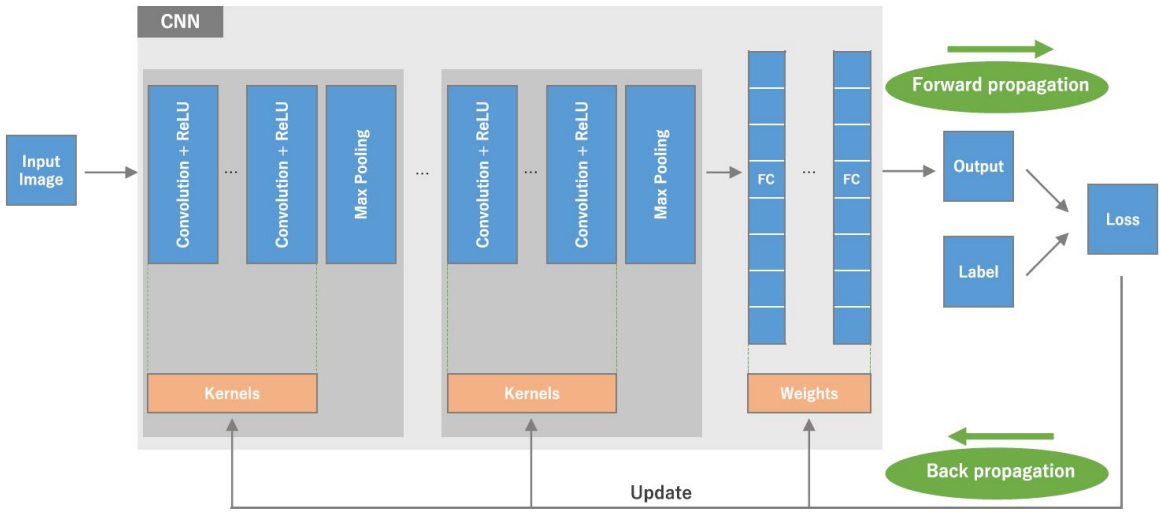


Figure 3.4: An overview of a CNN architecture and the training process. CNNs are composed of a stacking of several building blocks: convolution layers, pooling layers (e.g., max pooling), and fully connected (FC) layers. A model's performance under particular kernels and weights is calculated with a loss function through forward propagation on a training dataset, and learnable parameters, i.e., kernels and weights, are updated according to the loss value through backpropagation with gradient descent optimization algorithm. The ReLU notation stands for rectified linear unit. It is an activation function [YNDT18].

a stack of mathematical operations, such as convolution, a specialized type of linear operation. In digital images, pixel values are stored in a two-dimensional (2D) grid, i.e., an array of numbers as shown in figure 3.5, and a small grid of parameters called kernel, an optimizable feature extractor, is applied at each image position, which makes CNNs highly efficient for image processing, since a feature may occur anywhere in the image. As one layer feeds its output into the next layer, extracted features can hierarchically and progressively become more complex. The process of optimizing parameters such as kernels is called training, which is performed so as to minimize the difference between outputs and ground truth labels through an optimization algorithm called backpropagation [GBC16] and gradient descent [Rud17], among others.

Convolution

Convolution is a specialized type of linear operation used for feature extraction, 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 each element of the kernel 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 3.5a-c). This procedure is repeated applying multiple kernels to form an arbitrary number of feature maps, which represent different characteristics of the input tensors; different kernels can, thus, be considered as different feature extractors. Two key hyperparameters that define the convolution operation are size and number of kernels. The former is typically 3×3, but sometimes 5×5 or 7×7. The latter is arbitrary, and determines the depth of output feature maps. The convolution operation described above does not allow the center of each kernel to overlap the outermost element of the input tensor, and reduces the height and width of the output feature map compared to the input tensor. Padding, typically zero padding, is a technique to address this issue, where rows and columns of zeros are added on each side of the input tensor, so as to fit the center of a kernel on the outermost element and keep the same in-plane dimension through the convolution operation (figure 3.6). Modern CNN architectures usually employ zero padding to retain in-plane dimensions in order to apply more layers. Without zero padding, each successive feature map would get smaller after the convolution operation. The distance between two successive kernel positions is called a stride, which also defines the convolution operation. The common choice

Task	Last layer activation function
Binary classification	Sigmoid
Multiclass single-class classification	Softmax
Multiclass multiclass classification	Sigmoid
Regression to continuous values	Identity

Table 3.1: A list of commonly applied last layer activation functions for various tasks.

of a stride is 1; however, a stride larger than 1 is sometimes used in order to achieve downsampling of the feature maps. An alternative technique to perform downsampling is a pooling operation, as described below in Section 3.2.2.

Nonlinear Activation Function

The outputs of a linear operation such as convolution are then passed through a nonlinear activation function. Although smooth nonlinear functions, such as sigmoid or hyperbolic tangent (tanh) function, were used previously because they are mathematical representations of a biological neuron behavior, the most common nonlinear activation function used presently is the rectified linear unit (ReLU), which simply computes the function: $f(x) = \max(0, x)$ (figure 3.7) [[LBH15], [KSH12b], [[NH10]], [[RZL17]], [GBB11]].

3.2.2 Pooling Layer

A pooling layer provides a typical downsampling operation which reduces the in-plane dimensionality of the feature maps in order to introduce a translation invariance to small shifts and distortions, and decrease the number of subsequent learnable parameters. It is of note that there is no learnable parameter in any of the pooling layers, whereas filter size, stride, and padding are hyperparameters in pooling operations, similar to convolution operations.

Max Pooling

The most popular form of pooling operation is max pooling, which extracts patches from the input feature maps, outputs the maximum value in each patch, and discards all the other values (figure 3.8). A max pooling with a filter of size 2×2 with a stride of 2 is commonly used in practice. This downsamples the in-plane dimension of feature maps by a factor of 2. Unlike height and width, the depth dimension of feature maps remains unchanged.

Global Average Pooling

Another pooling operation worth noting is a global average pooling [LCY14]. A global average pooling performs an extreme type of downsampling, where a feature map with size of height $\times$ width is downsampled into a 1×1 array by simply taking the average of all the elements in each feature map, whereas the depth of feature maps is retained. This operation is typically applied only once before the fully connected layers. The advantages of applying global average pooling are as follows: (1) reduces the number of learnable parameters and (2) enables the CNN to accept inputs of variable size.

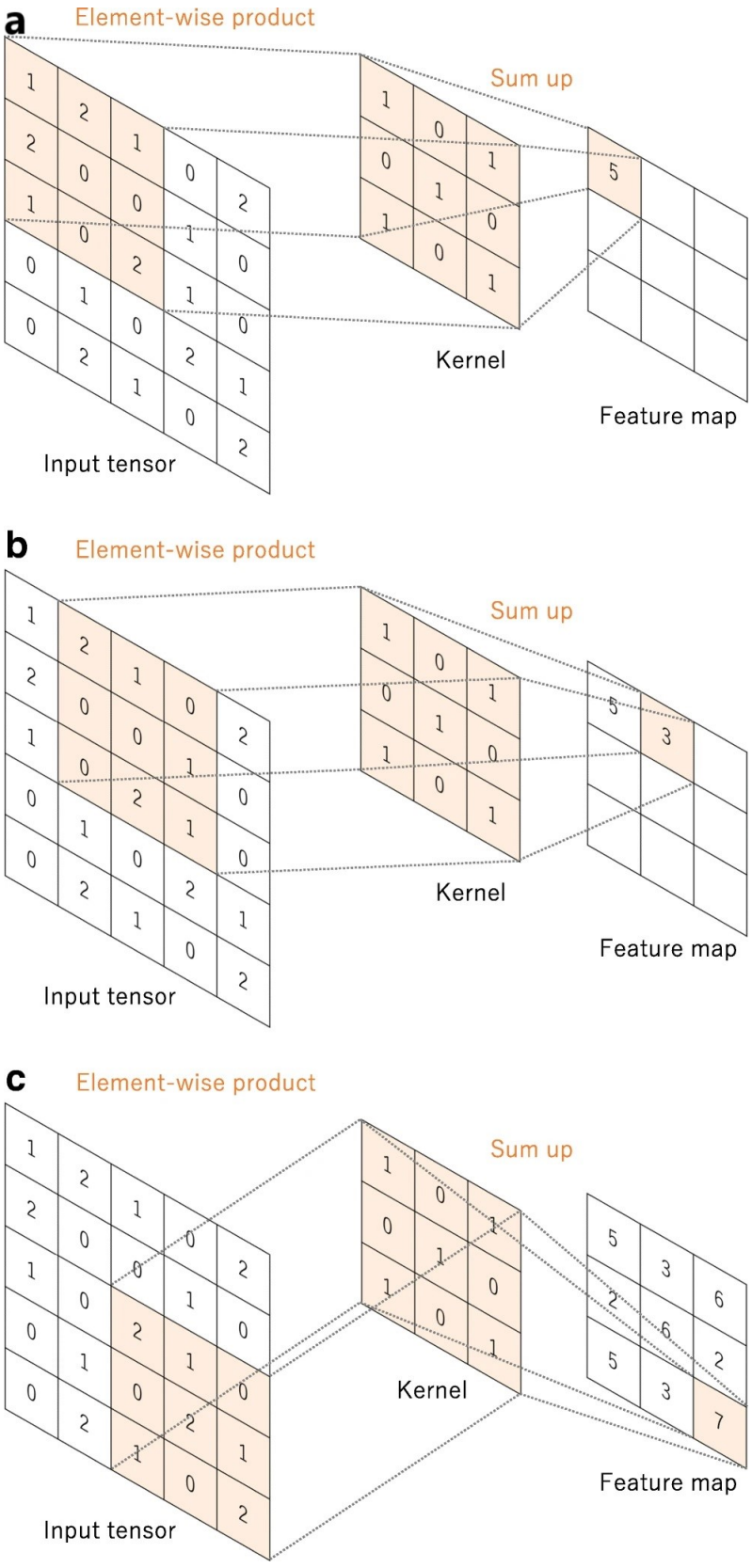


Figure 3.5: **a**–**c** An example of convolution operation with a kernel size of 3×3, no padding, and a stride of 1. A kernel is applied across the input tensor, and an element-wise product between each element of the kernel and the input tensor is calculated at each location and summed to obtain the output value in the corresponding position of the output tensor, called a feature map [YNDT18].

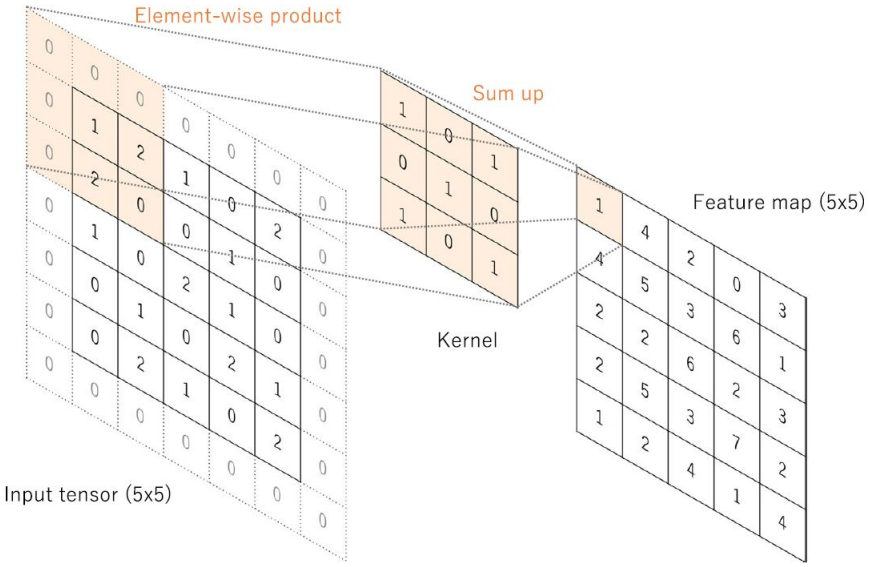


Figure 3.6: A convolution operation with zero padding so as to retain in-plane dimensions. Note that an input dimension of 5×5 is kept in the output feature map. In this example, a kernel size and a stride are set as 3×3 and 1, respectively [YNDT18].

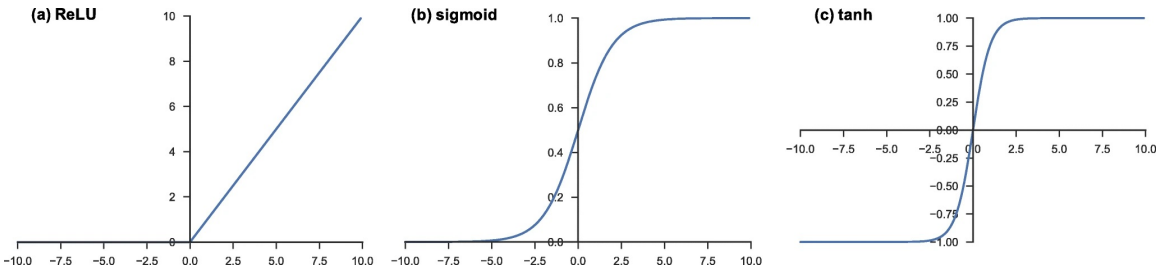


Figure 3.7: Activation functions commonly applied to neural networks: **a)** rectified linear unit (ReLU), **b)** sigmoid, and **c)** hyperbolic tangent (tanh) [YNDT18].

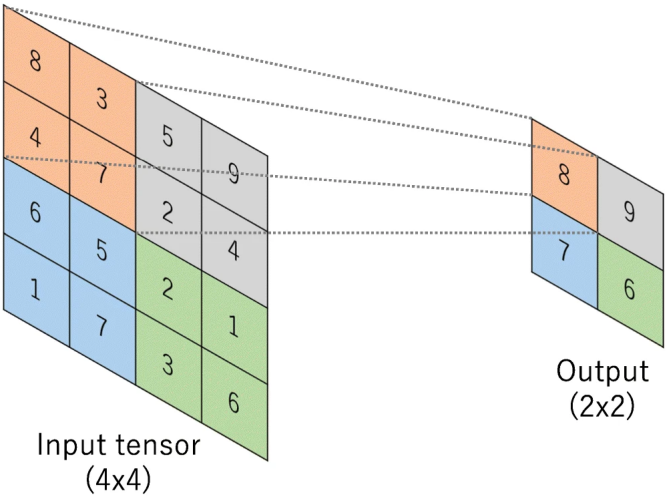


Figure 3.8: An example of max pooling operation with a filter size of 2×2, no padding, and a stride of 2, which extracts 2×2 patches from the input tensors, outputs the maximum value in each patch, and discards all the other values, resulting in downsampling the in-plane dimension of an input tensor by a factor of 2 [YNDT18].

3.2.3 Fully connected layer

The output feature maps of the final convolution or pooling layer is typically flattened, i.e., transformed into a one-dimensional (1D) array of numbers (or vector), and connected to one or more fully connected layers, also known as dense layers, in which every input is connected to every output by a learnable weight. Once the features extracted by the convolution layers and downsampled by the pooling layers are created, they are mapped by a subset of fully connected layers to the final outputs of the network, such as the probabilities for each class in classification tasks. The final fully connected layer typically has the same number of output nodes as the number of classes. Each fully connected layer is followed by a nonlinear function, such as ReLU, as described above. The activation function applied to the last fully connected layer is usually different from the others. An appropriate activation function needs to be selected according to each task. An activation function applied to the multiclass classification task is a softmax function which normalizes output real values from the last fully connected layer to target class probabilities, where each value ranges between 0 and 1 and all values sum to 1. Typical choices of the last layer activation function for various types of tasks are summarized in Table 3.1.

4. Methodology

In this chapter, first, the formulation of GAN is briefly reviewed in Section 4.1. Next, the CycleGAN, is presented with its loss functions, in Section 4.2.

4.1 Generative Adversarial Networks

Ian J. Goodfellow et al. [GPAM+14] described the adversarial modeling framework as most simple to apply when the models are both artificial neural networks. To learn the generator's distribution p_g over data x, a prior on input noise variables defined $p_z(z)$, then represent a mapping to data space as $G(z;\theta_g)$, where G is a differentiable function represented by a neural network with parameters θ_g. There also a second neural network $D(x;\theta_d)$ that outputs a single scalar. $D(x)$ represents the probability that x came from the data rather than p_g. D is trained to maximize the probability of assigning the correct label to both training examples and samples from G. Simultaneously G is trained to minimize $log(1 - D(G(z)))$. In other words, D and G play the following two-player minimax game with objective function $\mathcal{L}_{GAN}(G, D)$:

$$\min_G \max_D \mathcal{L}_{GAN}(D, G) = E_{x\sim p_{data}(x)}[logD(x)] + E_{z\sim p_z(z)}[log(1 - D(G(z)))]. \tag{4.1}$$

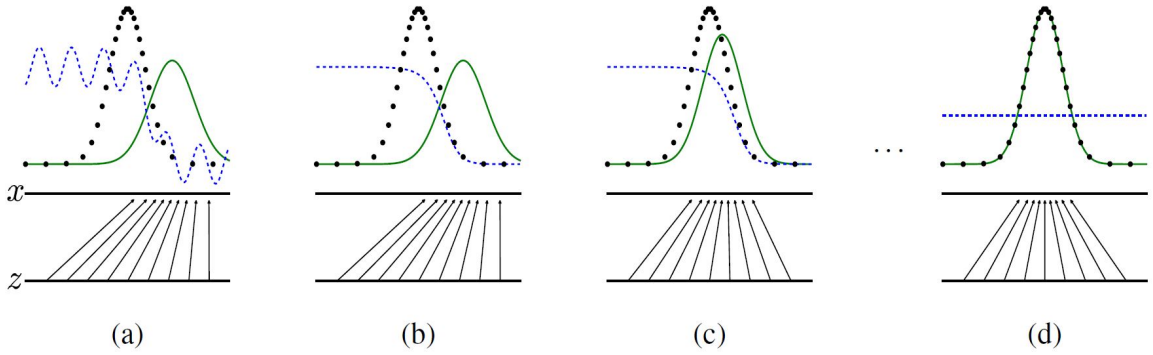


Figure 4.1: Generative adversarial networks are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data **g**enerating distribution (black, dotted line) p_x from those of the generative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which z is sampled, in this case uniformly. The horizontal line above is part of the domain of x. The upward arrows show how the mapping $x = G(z)$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g. (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(x) = \frac{p_{data}(x)}{p_{data}(x)+p_g(x)}$. (c) After an update to G, gradient of D has guided $G(z)$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{data}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(x) = \frac{1}{2}$ [GPAM+14].

In practice, equation 4.1 may not provide sufficient gradient for G to learn well. Early in learning, when G is poor, D can reject samples with high confidence because they are clearly different from the training data. In this case, $log(1 - D(G(z)))$ saturates. Rather than training G to minimize

$log(1 - D(G(z)))$ we can train G to maximize $logD(G(z))$. This objective function results in the same fixed point of the dynamics of G and D but provides much stronger gradients early in learning.

4.2 Cycle-Consistent Adversarial Networks

4.2.1 Formulation

The aim is to learn mapping functions between two domains X and Y. The Domain X represents synthetic document images distribution and domain Y represents real document images distribution. Given training samples $\{x_i\}_{i=1}^{N}$ where $x_i \in X$ and $\{y_j\}_{j=1}^{M}$ where $y_j \in Y$. We denote the data distribution $x \sim p_{data}(x)$ and $y \sim p_{data}(y)$. As illustrated in Figure 4.2, the model includes two mappings $G : X \rightarrow Y$ and $F : Y \rightarrow X$. In addition, we introduce two adversarial discriminators D_X and D_Y , where D_X aims to distinguish between images $\{x\}$ and translated images $\{F(y)\}$; in the same way, D_Y aims to discriminate between $\{y\}$ and $\{G(x)\}$. Our objective contains three types of terms: least-square losses (LSGAN) [MLX+17] for matching the distribution of generated images to the data distribution in the target domain; cycle consistency losses to prevent the learned mappings G and F from contradicting each other; and identity mapping loss to preserve the color of the input images.

4.2.2 Least-Square Loss

Viewing the discriminator as a classifier, regular GANs adopt the sigmoid cross-entropy loss function. As stated in Section 2.2, when updating the generator, this loss function will cause the problem of vanishing gradients for the samples that are on the correct side of the decision boundary, but are still far from the real data. Also to stabilize the model training procedure. First, $\mathcal{L}_{GAN}$ (equation 4.1), the negative log-likelihood objective replaced by a least-squares loss (LSGAN) [MLX+17]. This loss is more stable during training and generates higher quality results. The least-square loss is used to optimize the generator and discriminator adversarially. We use the a-b coding scheme for the discriminator, where a and b are the labels for fake data and real data, respectively. Then the modified objective functions using least-squares loss can be defined as follows:

$$\min_D \mathcal{L}_{LSGAN}(D_Y) = \frac{1}{2}\,\mathbb{E}_{y\sim p_{data}(y)}\,[\|(D(y)-b)\|_2] + \frac{1}{2}\,\mathbb{E}_{x\sim p_{data}(x)}\,[\|(D(G(x))-a)\|_2] \tag{4.2}$$

$$\min_G \mathcal{L}_{LSGAN}(G) = \mathbb{E}_{x\sim p_{data}(x)}\,[\|(D(G(x))-c)\|_2] \tag{4.3}$$

$$\mathcal{L}_{LSGAN}(G, D_Y, X, Y) = \min_D \mathcal{L}_{LSGAN}(D_Y) + \min_G \mathcal{L}_{LSGAN}(G) \tag{4.4}$$

$$\min_D \mathcal{L}_{LSGAN}(D_X) = \frac{1}{2}\,\mathbb{E}_{x\sim p_{data}(x)}\,[\|(D(x)-b)\|_2] + \frac{1}{2}\,\mathbb{E}_{y\sim p_{data}(y)}\,[\|(D(F(y))-a)\|_2] \tag{4.5}$$

$$\min_G \mathcal{L}_{LSGAN}(F) = \mathbb{E}_{y\sim p_{data}(y)}\,[\|(D(F(y))-c)\|_2] \tag{4.6}$$

$$\mathcal{L}_{LSGAN}(F, D_X, Y, X) = \min_D \mathcal{L}_{LSGAN}(D_X) + \min_G \mathcal{L}_{LSGAN}(F) \tag{4.7}$$

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

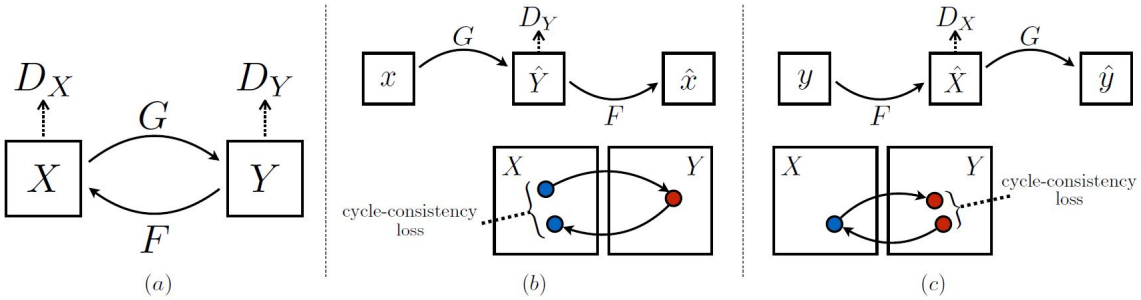


Figure 4.2: (a) CycleGAN model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs indistinguishable from domain Y, and vice versa for D_X and F. To further regularize the mappings, we introduce two cycle consistency losses that capture the intuition that if we translate from one domain to the other and back again we should arrive at where we started: (b) forward cycle-consistency loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$ [ZPIE20].

4.2.3 Cycle Consistency Loss

Adversarial training can, in theory, learn mappings G and F that produce outputs identically distributed as target domains Y and X respectively. However, with large enough capacity, a network can map the same set of input images to any random permutation of images in the target domain, where any of the learned mappings can induce an output distribution that matches the target distribution. Thus, adversarial losses alone cannot guarantee that the learned function can map an individual input x_i to a desired output y_i. To further reduce the space of possible mapping functions, we argue that the learned mapping functions should be cycle-consistent: as shown in Figure 4.2 (b), for each image x from domain X, the image translation cycle should be able to bring x back to the original image, i.e., $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$. We call this forward cycle consistency. Similarly, as illustrated in Figure 4.2 (c), for each image y from domain Y, G and F should also satisfy backward cycle consistency: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$. We incentivize this behavior using a cycle consistency loss:

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

4.2.4 Identity Mapping Loss

It is helpful to introduce an additional loss to encourage the mapping to preserve color composition between the input and output. In particular, we adopt the technique of [TPW16] and regularize the generator to be near an identity mapping when real samples of the target domain are provided as the input to the generator:

$$\mathcal{L}_{identity}(G, F) = \mathbb{E}_{y \sim p_{data}(y)}[\|(F(y) - y\|_1] + \mathbb{E}_{x \sim p_{data}(x)}[\|G(x) - x\|_1]. \tag{4.9}$$

4.2.5 Full Objective

The full objective is:

$$\begin{aligned} \mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{LSGAN}(G, D_Y, X, Y) \; + \; \mathcal{L}_{LSGAN}(F, D_X, Y, X) \; + \\ \lambda_{identity} \; \mathcal{L}_{identity}(G, F) \; + \; \lambda_{cyc} \; \mathcal{L}_{cyc}(G, F), \end{aligned} \tag{4.10}$$

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

5. Implementation and Evaluation

The proposed image-to-image translation application is implemented using CycleGAN. The quality of images generated by the CycleGAN is assessed by a classifier that is trained on the same generated images and tested on real images. The classification accuracy of the classifier on real images is the metric to evaluate the quality of generated images. The evaluation of images generated by CycleGAN is described thoroughly in section 5.8. In this thesis, numerous experiments were performed to understand the domain gap[1] between real document images, faxified document images, and CycleGAN generated document images. The experiments are visualized using Tensorboard[2]. TensorBoard provides the visualization and tooling needed for machine learning experimentation. The CycleGAN and Classifier are implemented in Python and using the TensorFlow library[AAB+15]. The 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

The dataset preparation is one of the vital aspects of training any neural network. Bad quality data leads to a poor generalization of neural networks. There are around ten types of documents that are considered to work with this image-to-image translation application. Hence, the CycleGAN is trained using a stash of synthetic document images and real document images. Around 100,000 synthetic document images in the source domain and the same number of real document images in the target domain. The synthetic document images are generated using unfilled form templates (figure A.1) and handwritten crops (figure 5.4 and 5.2). The process of inserting handwritten crops on empty templates can be visualized in figure 5.1. Each empty form template is filled with the help of provided bounding box annotations. For each class of template 10,000, synthetic document images are created. As mentioned earlier, 100,000 synthetic document images are created in total. The same created 100,000 synthetic document images are used while training CycleGAN. Just they are stashed at the same location collectively. The created 100,000 synthetic document images also faxified and a faxified dataset of 100,000 images is created. It has the same structure as synthetic document images, 10 classes, and each class has 10,000 images. The faxification process uses several image transformations to make a clean gray-scale image look like it was sent via fax. A sample faxified image can be seen in figure A.3. The faxification process described briefly in Section 5.6.2. Also, the faxification can be visualised in figure 5.9. For testing, around 1162 annotated real document images are used. The testing dataset is unbalanced. The datasets used in this thesis can not be cited because they are not open for public use.

[1] `https://machinelearning.apple.com/research/bridging-the-domain-gap-for-neural-models` last access: 04.05.2021

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

Datasets	Size (Number of Images)
Synthetic Document Images	100,000
Real Document Images	100,000
Faxified Document Images	100,000
Test Document Images	1162

Table 5.1: Datasets used for training CycleGAN and Classifier.

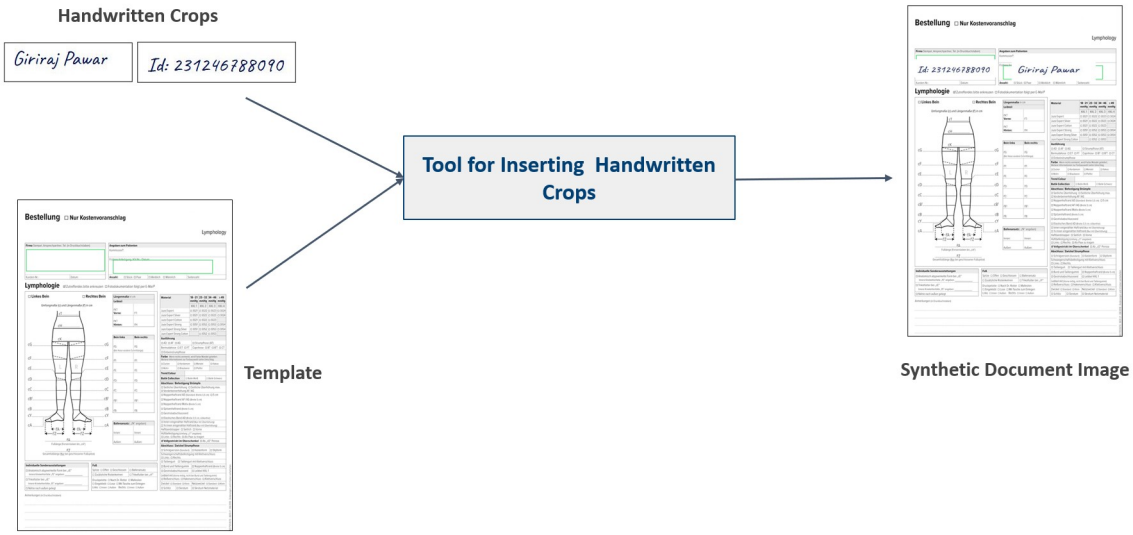


Figure 5.1: Inserting Handwritten Crops in Empty Form Templates.

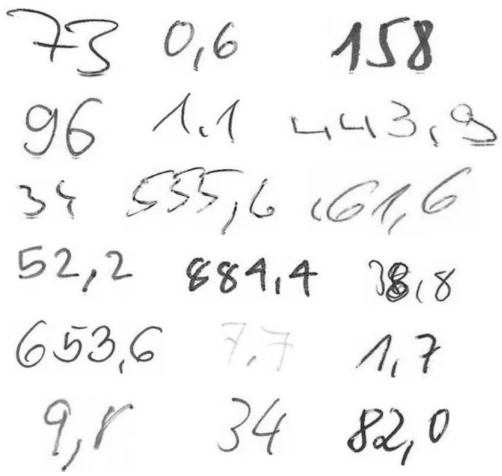


Figure 5.2: Examples of Handwritten Number Dataset ©AI4BD GmbH.

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 Test Document Images Dataset. This Dataset is used to Evaluate the Performance of the Classifier trained upon different data distributions.

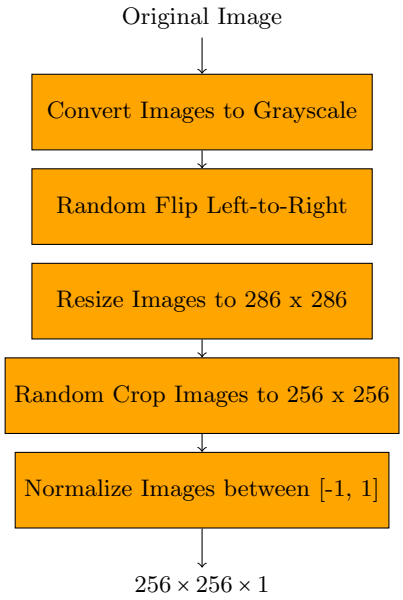


Figure 5.3: Pre-processing Process of Images.

5.2 Training Details

5.2.1 CycleGAN

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

The CycleGAN is trained from scratch, with a learning rate of 0.0002. In practice, the objective is divided by 2 while optimizing discriminator (equations 4.2 and 4.5), which slows down the rate at which discriminator learns, relative to the rate of generator. Weights are initialized from a Gaussian distribution with mean (μ) 0 and standard deviation (σ) 0.02. Images used for the training CycleGAN are converted into grayscale. Also, random mirroring and random jittering is applied. In random mirroring, the image is randomly flipped horizontally i.e. left to right. In random jittering, the image is resized to 286×286 and then randomly cropped to 256×256. As mentioned in the paper [ZPIE20], random jittering and mirroring applied to the training dataset. These are some of the image augmentation techniques that avoids overfitting. These are some of the image augmentation techniques that avoid overfitting. Lastly, the images are normalized in the range of $[-1, 1]$.

5.2.2 Classifier

The images used to train and test the classifiers were also applied with random mirroring and random jittering after converting the image to grayscale. The size of the images is $256 \times 256 \times 1$. Also, the images are normalized in the range of $[-1, 1]$. The classifier architecture can be visualized in figure 5.5. The pre-processing process can be seen in figure 5.3. The classifier trained using 80% data and 20% data used for validation. Subsequently, the performance of the classifiers evaluated using real test document images dataset.

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

Figure 5.4: Examples of Handwritten Numbers from the MNIST Dataset.[3]

5.3 Network Architecture

5.3.1 CycleGAN

The architecture of CycleGAN is adapted from Johnson et al. [JAFF16]. The generator network is implemented using a sequence of downsampling convolutional blocks to encode the $256 \times 256 \times 1$ grayscale input image, 9 Residual Network (ResNet) convolutional blocks to transform the image, and a number of upsampling convolutional blocks to generate the output image of the same dimension as the input image. The reason behind using residual blocks is it resolves the vanishing gradient problem in deep neural networks. The discriminator networks uses PatchGAN [IZZE18]. In PatchGAN, after feeding one input image to the network, it gives you the probabilities of two things: either real or fake, but not in scalar output indeed, it used the $N \times N$ output vector. Here $N \times N$ can be different depending on the dimension of an input image. The naming convention used in the Johnson et al.'s [JAFF16] Github repository. The architecture of discriminator and generator can be preciously visualised in figures A.11 and A.5.

Let `c7s1-k` denote a 7×7 Convolution-InstanceNorm-ReLU layer with k filters and stride 1. `dk` denotes a 3×3 Convolution-InstanceNorm-ReLU layer with k filters and stride 2. Reflection padding was used to reduce artifacts. `Rk` denotes a residual block that contains two 3×3 convolutional layers with the same number of filters on both layer. `uk` denotes a 3×3 fractional-strided-Convolution-InstanceNorm-ReLU layer with k filters and stride $\frac{1}{2}$. The generator network with 9 residual blocks consists of: `c7s1-64, d128, d256, R256, R256, R256, R256, R256, R256, R256, R256, u128, u64, c7s1-1`. followed by a Tanh function (figure 3.7). The discriminator uses 70×70 PatchGAN [IZZE18], which is also called as Markovian discriminator [IZZE18]. Let `Ck` denote a 4×4 Convolution-InstanceNorm-LeakyReLU layer with k filters and stride 2. After the last layer, a convolution is applied to produce a 1-dimensional output. We do not use InstanceNorm for the first C64 layer. We use leaky ReLUs with a slope of 0.2. The discriminator architecture is: `C64-C128-C256-C512`.

5.3.2 Classifier

The classifier used to determined the domain gap between data distributions is a CNN. The classifier architecture is simplistic (figure 5.5). It has just Two Convolution Layers, One Max Pooling Layer, and One Dropout Layer. The architecture of classifier can be preciously visualised in figure A.4. While conducting the experiments, ResNet-50 [HZRS15] architecture also has been considered. The architecture of ResNet-50 classifier can be viewed in figure 5.6.

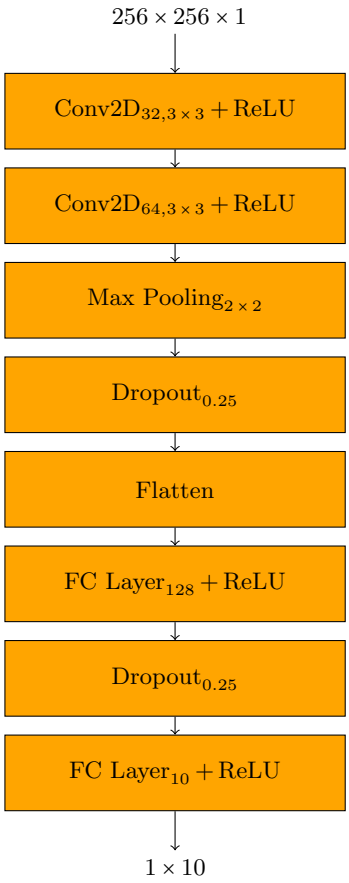


Figure 5.5: Classifier Architecture Blocks.

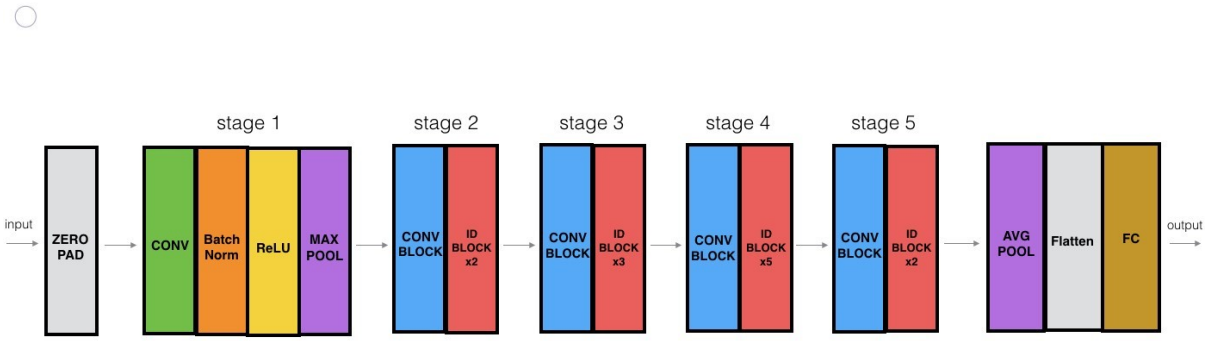


Figure 5.6: ResNet-50 Classifier Architecture.[4]

[4] `https://github.com/priya-dwivedi/Deep-Learning/blob/master/resnet_keras/Residual_Networks_yourself.ipynb` last access: 04.05.2021

5.4 Experiment Steps

1. Train a classifier on synthetic document images and evaluate accuracy on the test document images.
2. Train a classifier on faxified document images and evaluate accuracy on the test document images.
3. Train a CycleGAN and save the model.
4. Load the saved CycleGAN model and generate document images using 100,000 synthetic document images as input.
5. Train a classifier on generated document images and evaluate its performance on test document images.
6. Compare the performance of classifiers and analyse whether generated data distribution matches real data distribution.

5.5 Experiments

5.6 Training CycleGAN

The CycleGAN consists of two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$ (figure 4.2). Here indicates CycleGAN has Two generators G and F. The Generator G transforms images from Source Domain (Synthetic Document Images) to Target Domain (Real Document Images). The Generator F transforms images from Target Domain (Real Document Images) into Source Domain (Synthetic Document Images).

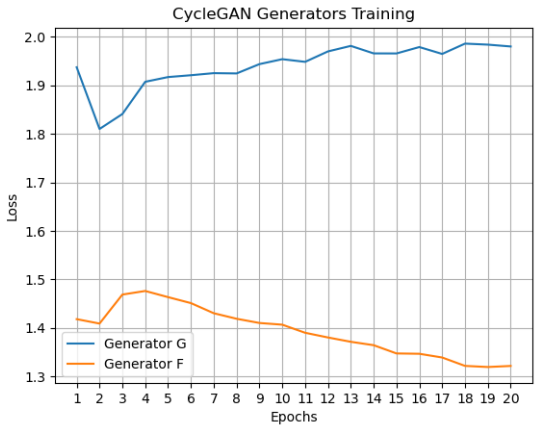


Figure 5.7:

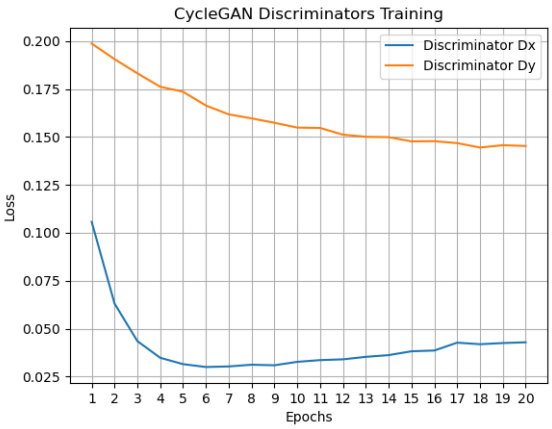


Figure 5.8:

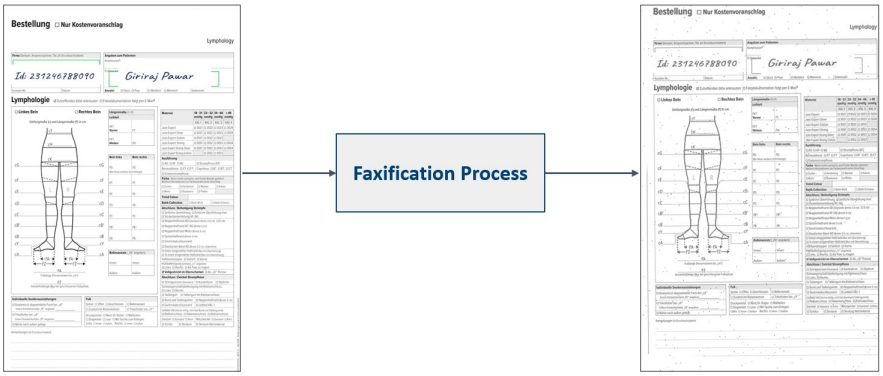


Figure 5.9: Faxification Process of Synthetic Document Images. This process uses several image transformations in order to make a clean gray-scale image look like it was sent via fax.

5.6.1 Training a Classifier on Synthetic Document Images

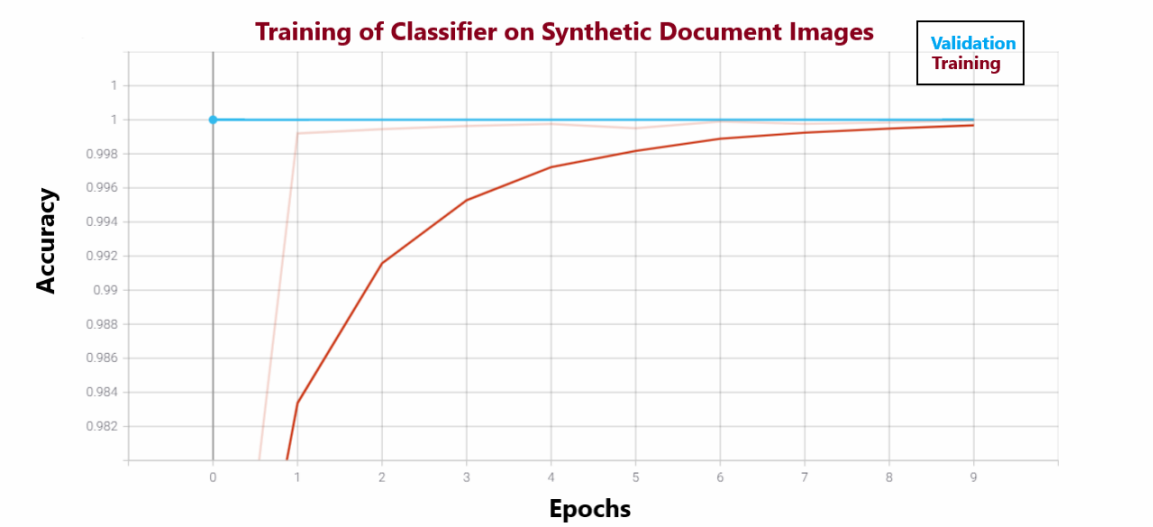


Figure 5.10: Accuracy Vs Epoch Plot, While Training the Classifier on Synthetic Document Images.

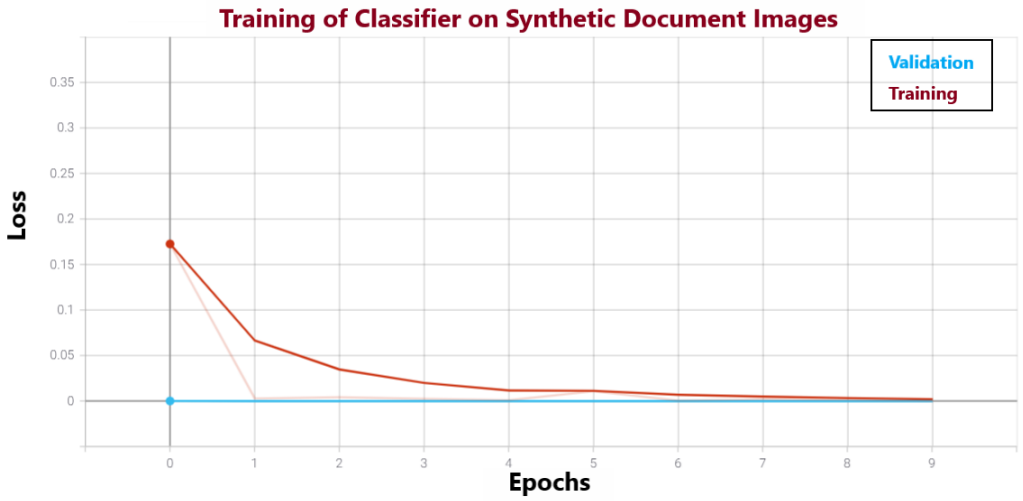


Figure 5.11: Loss Vs Epoch Plot, While Training the Classifier on Synthetic Document Images.

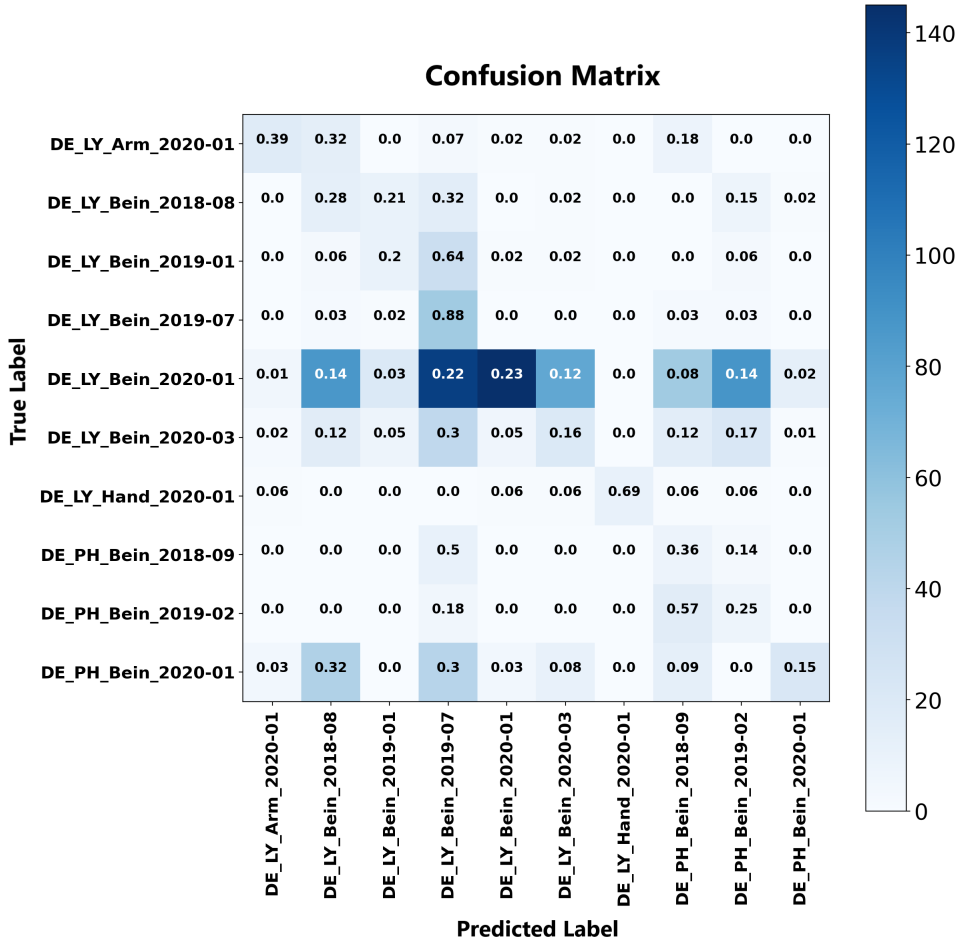


Figure 5.12: Confusion Matrix. The Classifier is trained using Synthetic Document Images Dataset. The Performance of the Classifier evaluated using real test document images.

	Precision	Recall	F1-score	Support
DE_LY_Arm_2020-01	0.59	0.39	0.47	44
DE_LY_Bein_2018-08	0.07	0.28	0.11	47
DE_LY_Bein_2019-01	0.21	0.20	0.20	50
DE_LY_Bein_2019-07	0.16	0.88	0.27	60
DE_LY_Bein_2020-01	0.92	0.23	0.37	624
DE_LY_Bein_2020-03	0.18	0.16	0.17	128
DE_LY_Hand_2020-01	1.00	0.69	0.81	16
DE_PH_Bein_2018-09	0.07	0.36	0.12	22
DE_PH_Bein_2019-02	0.05	0.25	0.09	28
DE_PH_Bein_2020-01	0.59	0.15	0.24	143
Accuracy			0.26	1162
Macro Average	0.38	0.36	0.29	1162
Weighted Average	0.64	0.26	0.31	1162

Table 5.3: Classification Report. Whne

5.6.2 Training a Classifier on faxified Document Images

The faxification process uses several image transformations to make a clean gray-scale image look like it was sent via fax. The faxification process creates another 100,000 faxified document images. Typically, fax machines use a horizontal resolution of 1728 pixels and transmit only black-and-white images, which might be dirty and are generally also not aligned perfectly. This leads to several typical artifacts, which are mimicked by the faxify tool. The faxification process consists of the steps like Conversion to a grayscale image, Gamma transforms, Resize canvas, Rescale to fax width, Rotation, Add vertical lines (with a probability), Add noise, Brightness transform, and Binarization.

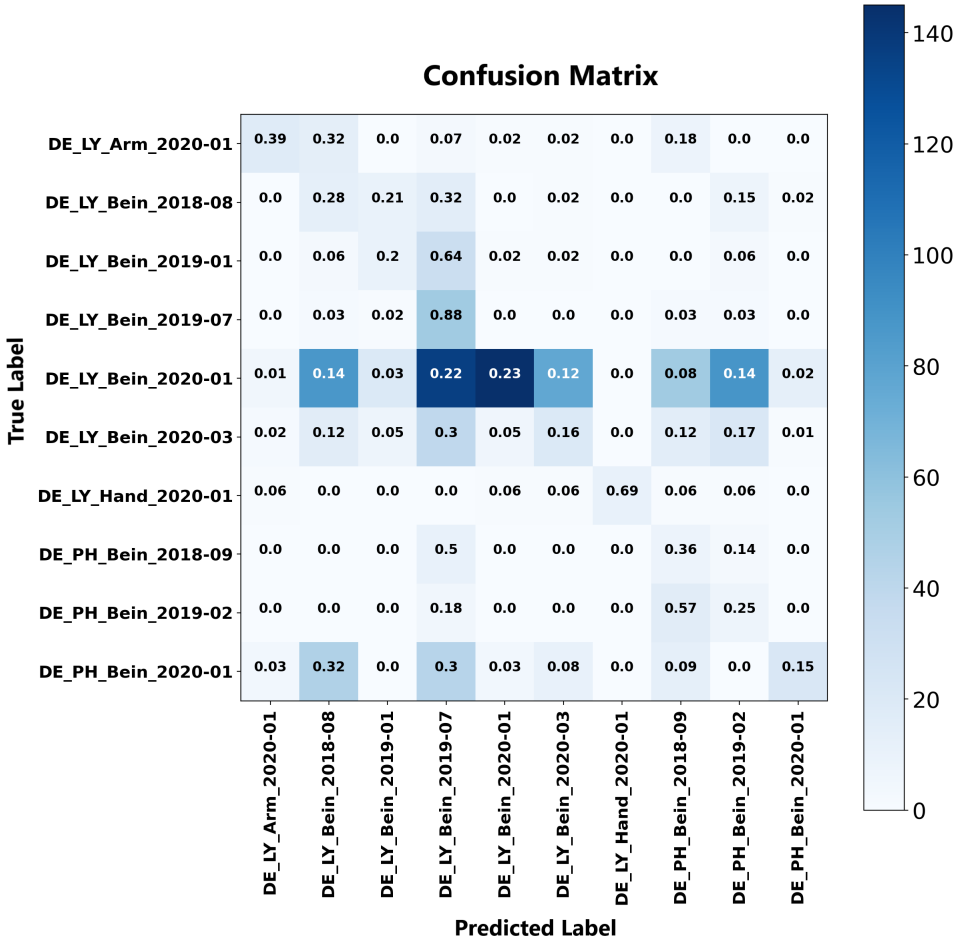


Figure 5.13: Confusion Matrix. The Classifier is trained using Synthetic Document Images Dataset. The Performance of the Classifier evaluated using real test document images.

	Precision	Recall	F1-score	Support
DE_LY_Arm_2020-01	1.00	0.98	0.99	44
DE_LY_Bein_2018-08	1.00	0.28	0.43	47
DE_LY_Bein_2019-01	0.46	0.24	0.32	50
DE_LY_Bein_2019-07	0.37	1.00	0.54	60
DE_LY_Bein_2020-01	0.86	0.04	0.08	624
DE_LY_Bein_2020-03	0.18	0.95	0.30	128
DE_LY_Hand_2020-01	0.76	1.00	0.86	16
DE_PH_Bein_2018-09	0.80	0.36	0.50	22
DE_PH_Bein_2019-02	0.66	0.96	0.78	28
DE_PH_Bein_2020-01	0.99	0.97	0.98	143
Accuracy			0.40	1162
Macro Average	0.71	0.68	0.58	1162
Weighted Average	0.76	0.40	0.33	1162

Table 5.4: Classification Report.

5.6.3 Training a Classifier on CycleGAN Generated Document Images

5.7 Evaluation Metrics

5.8 Evaluation

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

Classification report and metrics

5.8.1 Overview of Performance Gap between Data Distributions

Th

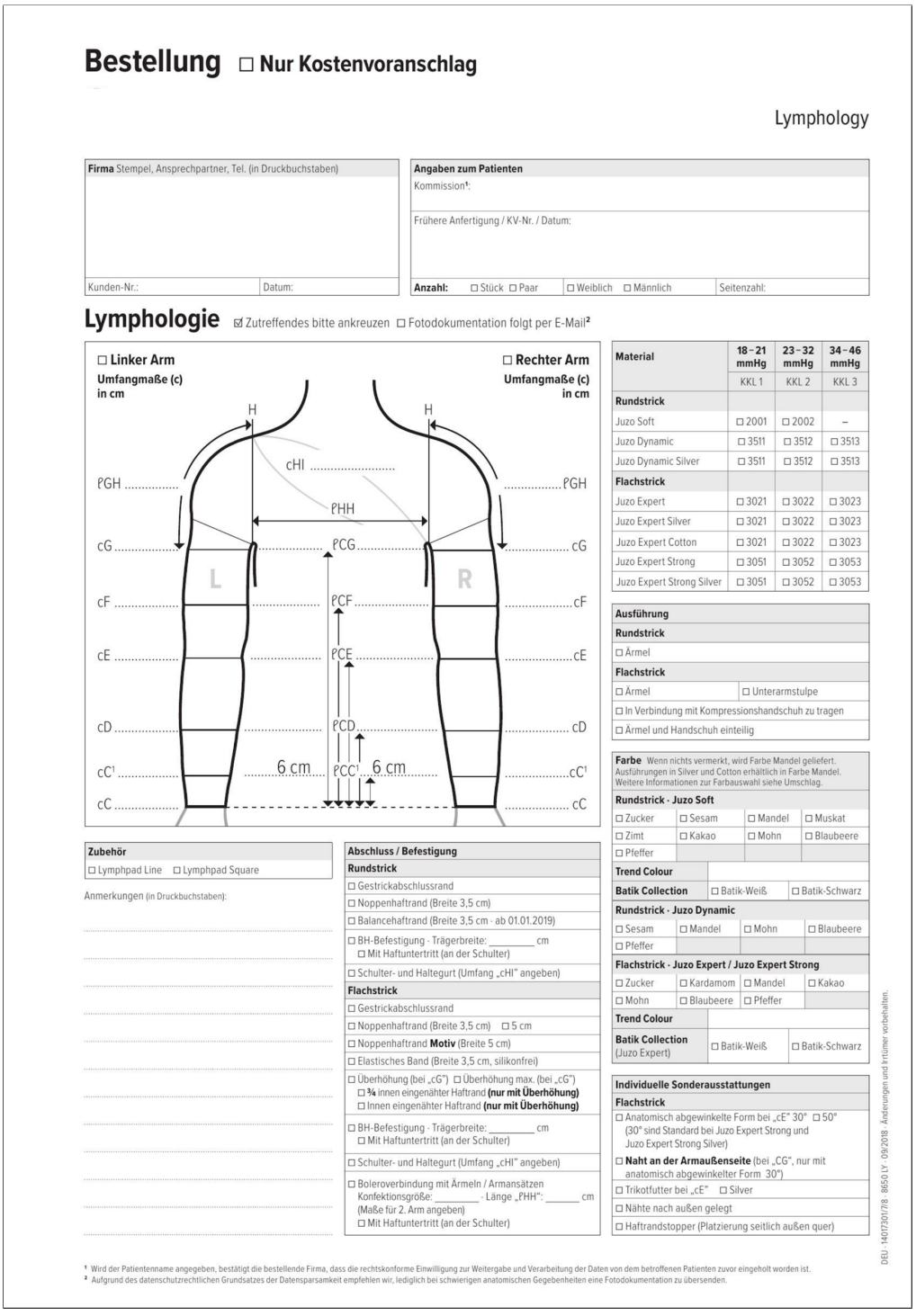
6. Conclusion and Future Work

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

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

A. Appendix

A.1 Document Images



Bestellung ☐ Nur Kostenvoranschlag

Lymphology

Firma Stempel, Ansprechpartner, Tel. (in Druckbuchstaben)	**Angaben zum Patienten**
	Kommission[1]:
	Frühere Anfertigung / KV-Nr. / Datum:
Kunden-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):

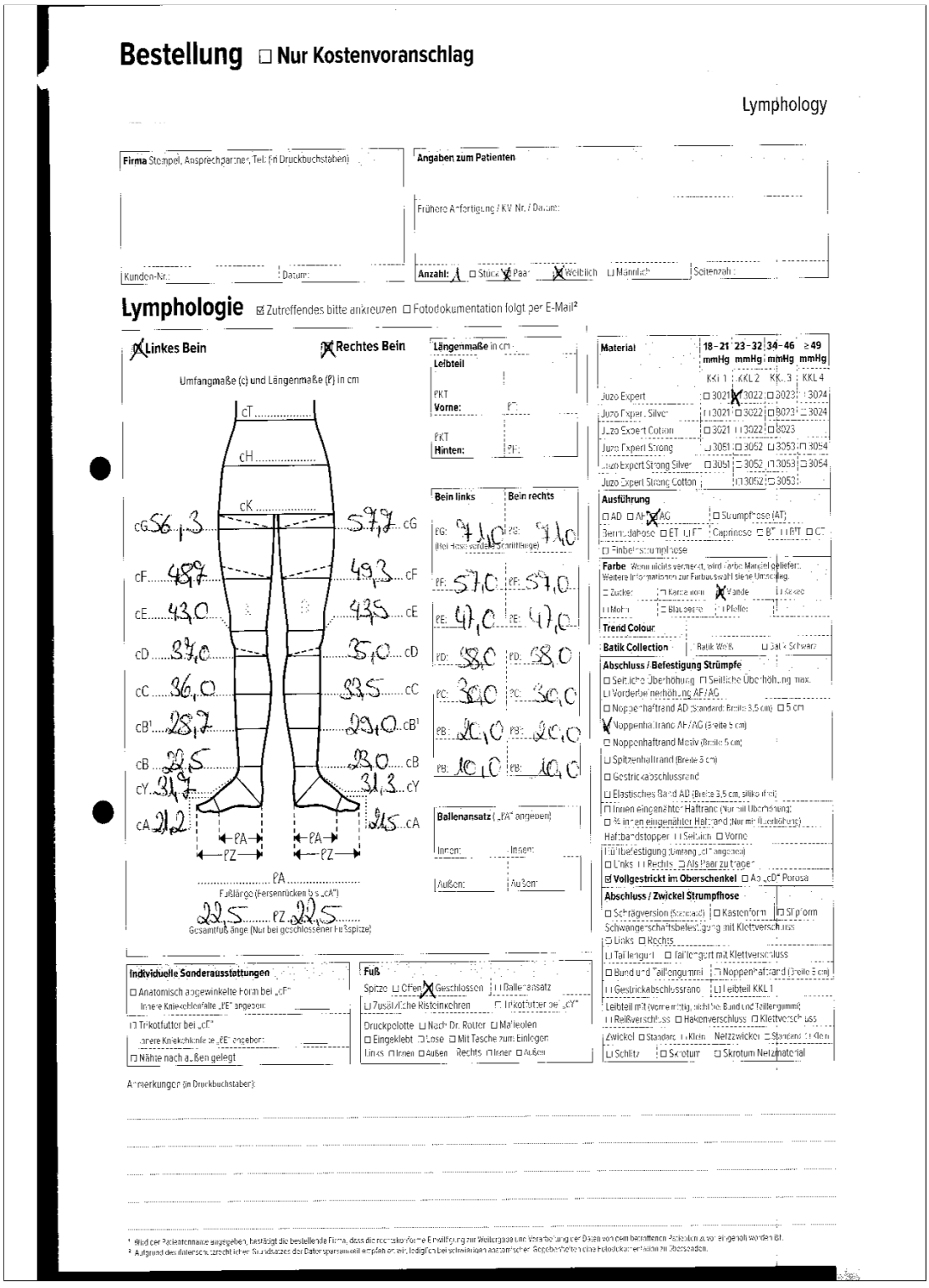
Abschluss / Befestigung
Rundstrick
☐ Gestrickabschlussrand
☐ Noppenhaftrand (Breite 3,5 cm)
☐ Balancehaftrand (Breite 3,5 cm · ab 01.01.2019)
☐ BH-Befestigung · Trägerbreite: _____ cm ☐ Mit Haftuntertritt (an der Schulter)
☐ Schulter- und Haltegurt (Umfang „cHI" angeben)
Flachstrick
☐ Gestrickabschlussrand
☐ 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 „℘HH": _____ cm (Maße für 2. Arm angeben) ☐ Mit Haftuntertritt (an der Schulter)

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

[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 · 1401730178 · 8650 LY · 09/2018 · Änderungen und Irrtümer vorbehalten.

Figure A.1: Empty Document Image.



Bestellung ☐ **Nur Kostenvoranschlag**

Lymphology

Firma Stempel, Ansprechpartner, Tel.: (in Druckbuchstaben)

Angaben zum Patienten

Frühere Anfertigung / KV-Nr. / Datum:

Kunden-Nr.: Datum:

Anzahl: 1 ☐ Stück ☒ Paar ☒ Weiblich ☐ Männlich Seitenzahl:

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

☒ **Linkes Bein** ☒ **Rechtes Bein**

Umfangmaße (c) und Längenmaße (P) in cm

cT, cH, cK, cG 56,3 / 57,7, cF 48,7 / 49,3, cE 43,0 / 43,5, cD 37,0 / 35,0, cC 36,0 / 33,5, cB¹ 28,7 / 29,0, cB 22,5 / 23,0, cY 31,7 / 31,3, cA 21,2 / 21,5, PA, PZ

Fußlänge (Fersenrücken bis „cA")

22,5 PZ 22,5

Gesamtfußlänge (Nur bei geschlossener Fußspitze)

Längenmaße in cm:

Leibteil

PKT **Vorne:** PF:

PKT **Hinten:** PH:

Bein links	**Bein rechts**
PG: 71,0 (Bei Hose: vordere Schrittlänge)	PG: 71,0
PF: 57,0	PF: 57,0
PE: 41,0	PE: 41,0
PD: 38,0	PD: 38,0
PC: 30,0	PC: 30,0
PB: 20,0	PB: 20,0
PB: 10,0	PB: 10,0

Ballenansatz („PA" angeben)

Innen: Innen:

Außen: Außen:

Material	**18–21 mmHg**	**23–32 mmHg**	**34–46 mmHg**	**≥ 49 mmHg**
	KKL 1	KKL 2	KK. 3	KKL 4
Juzo Expert	☐ 3021	☒ 3022	☐ 3023	☐ 3024
Juzo Expert Silver	☐ 3021	☐ 3022	☐ 3023	☐ 3024
Juzo Expert Cotton	☐ 3021	☐ 3022	☐ 3023	
Juzo Expert Strong	☐ 3051	☐ 3052	☐ 3053	☐ 3054
Juzo Expert Strong Silver	☐ 3051	☐ 3052	☐ 3053	☐ 3054
Juzo Expert Strong Cotton		☐ 3052	☐ 3053	

Ausführung

☐ AD ☐ AF ☒ AG ☐ Strumpfhose (AT)

Bermudahose ☐ ET ☐ LT ☐ Caprihose ☐ BT ☐ TBT ☐ CT

☐ Einbeinstrumpfhose

Farbe Wenn nichts vermerkt, wird in Farbe Mandel geliefert. Weitere Informationen zur Farbauswahl siehe Umschlag.

☐ Zucker ☐ Kaffee ☒ Mande ☐ Sesam

☐ Mohn ☐ Blaubeere ☐ Pfeffer

Trend Colour

Batik Collection ☐ Batik Weiß ☐ Batik Schwarz

Abschluss / Befestigung Strümpfe

☐ Seitliche Überhöhung ☐ Seitliche Überhöhung max.

☐ Vorderbein nerhöhung AF/AG

☐ Noppenhaftrand AD (Standard: Breite 3,5 cm) ☐ 5 cm

☒ Noppenhaftrand AF/AG (Breite 5 cm)

☐ Noppenhaftrand Mistiv (Breite 5 cm)

☐ Spitzenhaftrand (Breite 5 cm)

☐ Gestrickabschlussrand

☐ Elastisches Band AD (Breite 3,5 cm, silikonfrei)

☐ Innen eingenähter Haftrand (Nur mit Überhöhung)

☐ Innen eingenähter Haftrand (Nur mit Überhöhung)

Haftbandstopper ☐ Seitlich ☐ Vorne

Fußbefestigung (Umfang „cA" angeben)

☐ Links ☐ Rechts ☐ Als Paar zu tragen

☑ **Vollgestrickt im Oberschenkel** ☐ Ab „cD" Porosa

Abschluss / Zwickel Strumpfhose

☐ Schrägversion (Standard) ☐ Kastenform ☐ Stiefform

Schwangerschaftsbefestigung mit Klettverschluss

☐ Links ☐ Rechts

☐ Taillengurt ☐ Taillengurt mit Klettverschluss

☐ Bund und Taillengummi ☐ Noppenhaftrand (Breite 5 cm)

☐ Gestrickabschlussrand ☐ Leibteil KKL 1

Leibteil mit (vorne mittig, sichtbar) Bund und Taillengummi:

☐ Reißverschluss ☐ Hakenverschluss ☐ Klettverschluss

Zwickel ☐ Standard ☐ Klein Netzzwickel ☐ Spandex ☐ Baum

☐ Schlitz ☐ Skrotum ☐ Skrotum Netzmaterial

Individuelle Sonderausstattungen

☐ Anatomisch angewinkelte Form bei „cF"

Innere Kniekehlenfalte „PE" angeben:

☐ Trikotfutter bei „cF"

Innere Kniekehlenfalte „PE" angeben:

☐ Nähte nach außen gelegt

Fuß

Spitze ☐ Offen ☒ Geschlossen ☐ Ballenansatz

☐ Zusätzliche Ristoinkehren ☐ Trikotfutter bei „cY"

Druckpolotte ☐ Nach Dr. Rotter ☐ Malleolen

☐ Eingeklebt ☐ Lose ☐ Mit Tasche zum Einlegen

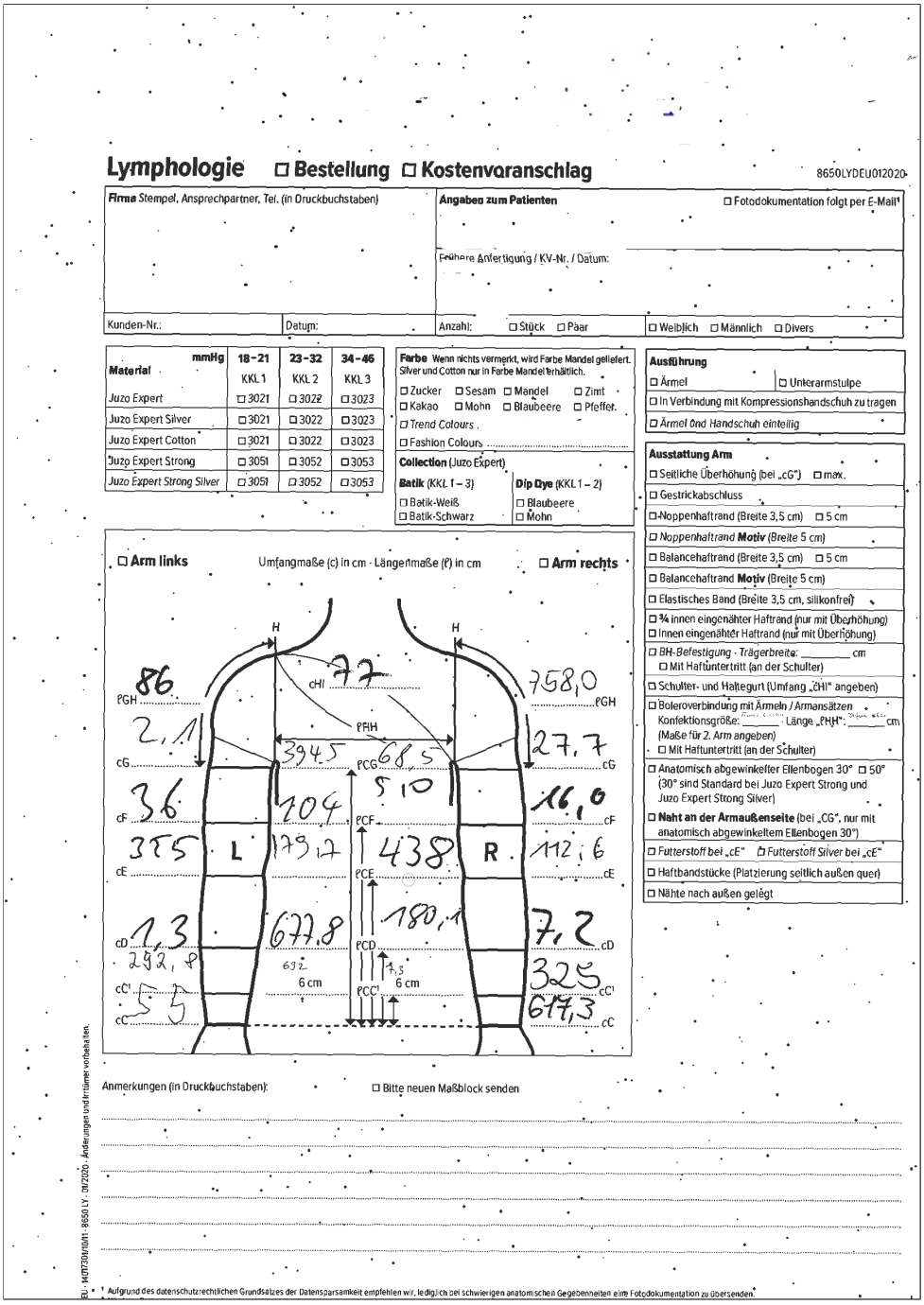
Links ☐ Innen ☐ Außen Rechts ☐ Innen ☐ Außen

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 gegeben eingeholt worden ist.

² Aufgrund des limitierten zeitlichen Rundschutzes der Datensparsamkeit empfehlen wir, lediglich bei schwierigen anatomischen Gegebenheiten eine Fotodokumentation zu übersenden.

Figure A.2: Real Document Image.



Lymphologie ☐ **Bestellung** ☐ **Kostenvoranschlag** 8650LYDEU012020

Firma Stempel, Ansprechpartner, Tel. (in Druckbuchstaben)

Angaben zum Patienten ☐ Fotodokumentation folgt per E-Mail*

Frühere Anfertigung / KV-Nr. / Datum:

Kunden-Nr.: Datum: Anzahl: ☐ Stück ☐ Paar ☐ Weiblich ☐ Männlich ☐ Divers

Material / mmHg	18–21 KKL 1	23–32 KKL 2	34–46 KKL 3
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

Farbe Wenn nichts vermerkt, wird Farbe Mandel geliefert. Silver und Cotton nur in Farbe Mandel erhältlich.

☐ Zucker ☐ Sesam ☐ Mandel ☐ Zimt
☐ Kakao ☐ Mohn ☐ Blaubeere ☐ Pfeffer
☐ *Trend Colours*
☐ Fashion Colours

Collection (Juzo Expert)

Batik (KKL 1 – 3)
☐ Batik-Weiß
☐ Batik-Schwarz

Dip Dye (KKL 1 – 2)
☐ Blaubeere
☐ Mohn

Ausführung
☐ Ärmel ☐ Unterarmstulpe
☐ In Verbindung mit Kompressionshandschuh zu tragen
☐ *Ärmel und Handschuh einteilig*

Ausstattung Arm
☐ Seitliche Überhöhung (bei „cG") ☐ max.
☐ Gestrickabschluss
☐ Noppenhaftrand (Breite 3,5 cm) ☐ 5 cm
☐ *Noppenhaftrand* **Motiv** (Breite 5 cm)
☐ Balancehaftrand (Breite 3,5 cm) ☐ 5 cm
☐ Balancehaftrand **Motiv** (Breite 5 cm)
☐ Elastisches Band (Breite 3,5 cm, silikonfrei)
☐ 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 „PHH": _____ cm
(Maße für 2. Arm angeben)
☐ Mit Haftuntertritt (an der Schulter)
☐ Anatomisch abgewinkelter Ellenbogen 30° ☐ 50°
(30° sind Standard bei Juzo Expert Strong und Juzo Expert Strong Silver)
☐ **Naht an der Armaußenseite** (bei „cG", nur mit anatomisch abgewinkeltem Ellenbogen 30°)
☐ *Futterstoff bei „cE"* ☐ *Futterstoff Silver bei „cE"*
☐ Haftbandstücke (Platzierung seitlich außen quer)
☐ Nähte nach außen gelegt

☐ **Arm links** Umfangmaße (c) in cm · Längenmaße (l) in cm ☐ **Arm rechts**

Anmerkungen (in Druckbuchstaben): ☐ Bitte neuen Maßblock senden

EU · M/07301/0/01 · 8650 LY · 06/2020 · Änderungen und Irrtümer vorbehalten.

* Aufgrund des datenschutzrechtlichen Grundsatzes der Datensparsamkeit empfehlen wir, lediglich bei schwierigen anatomischen Gegebenheiten eine Fotodokumentation zu übersenden.

Figure A.3: Faxified Document Image.

A.2 Classifier Architecture Diagram

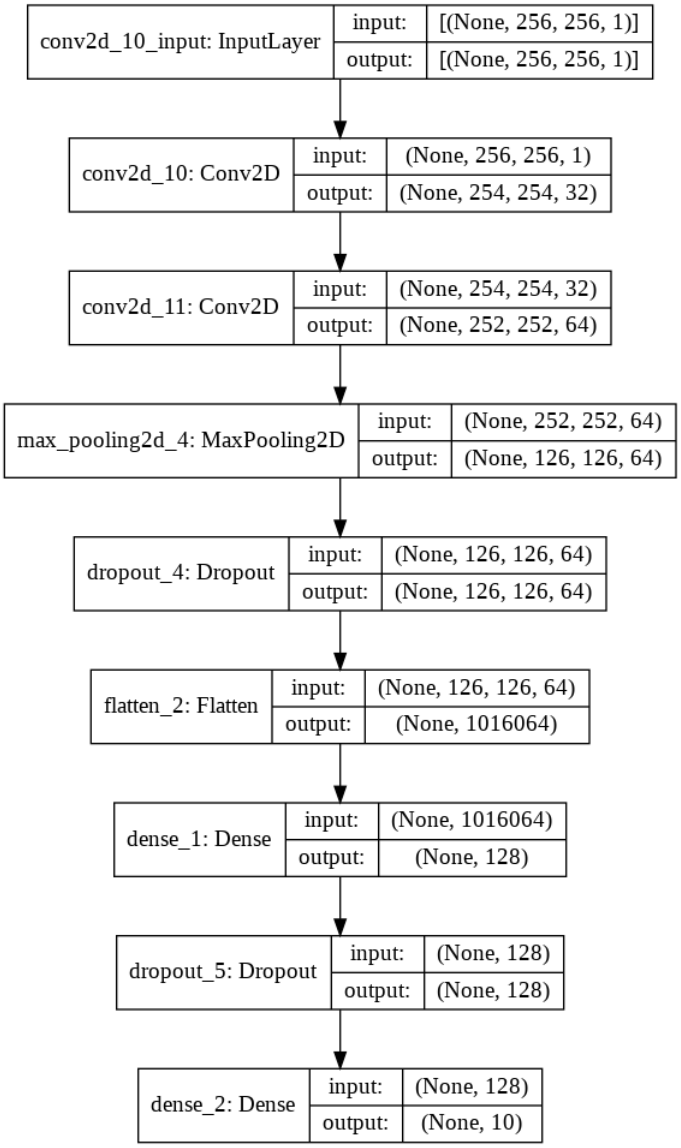


Figure A.4: Classifier Architecture Diagram.

A.3 Generator Architecture Diagram

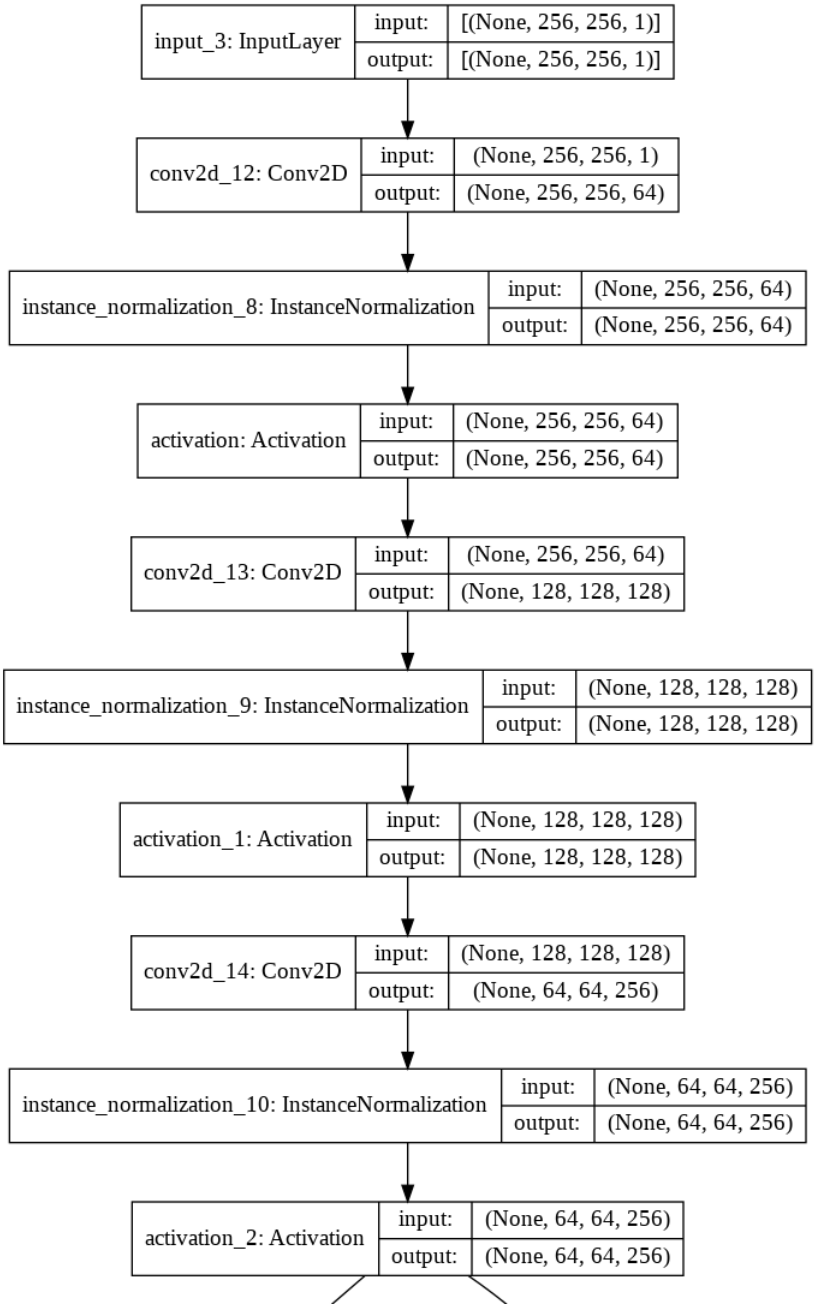


Figure A.5: Generator Architecture Diagram. Continue to Next Page.

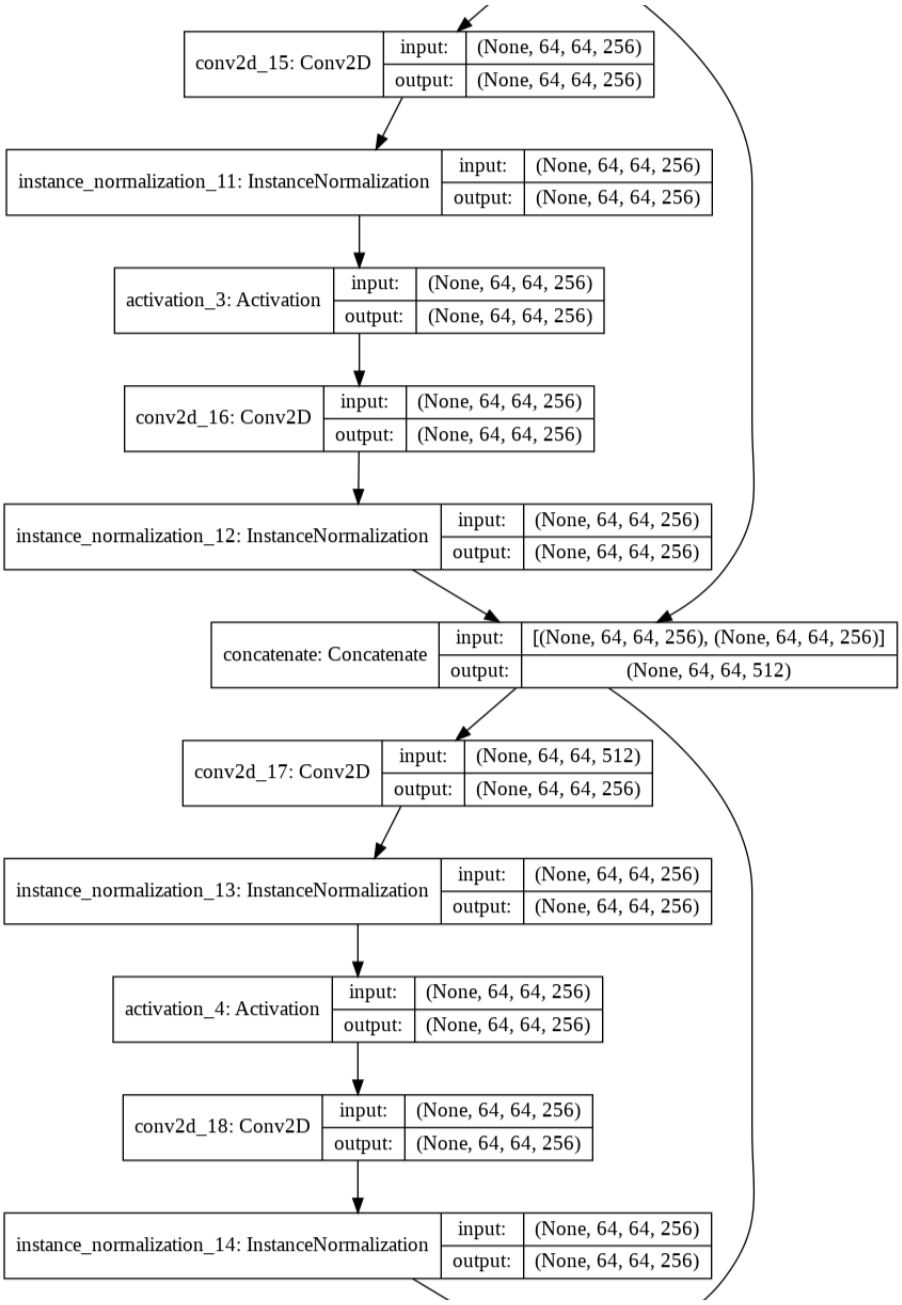


Figure A.6: Generator Architecture Diagram. Continue to Next Page.

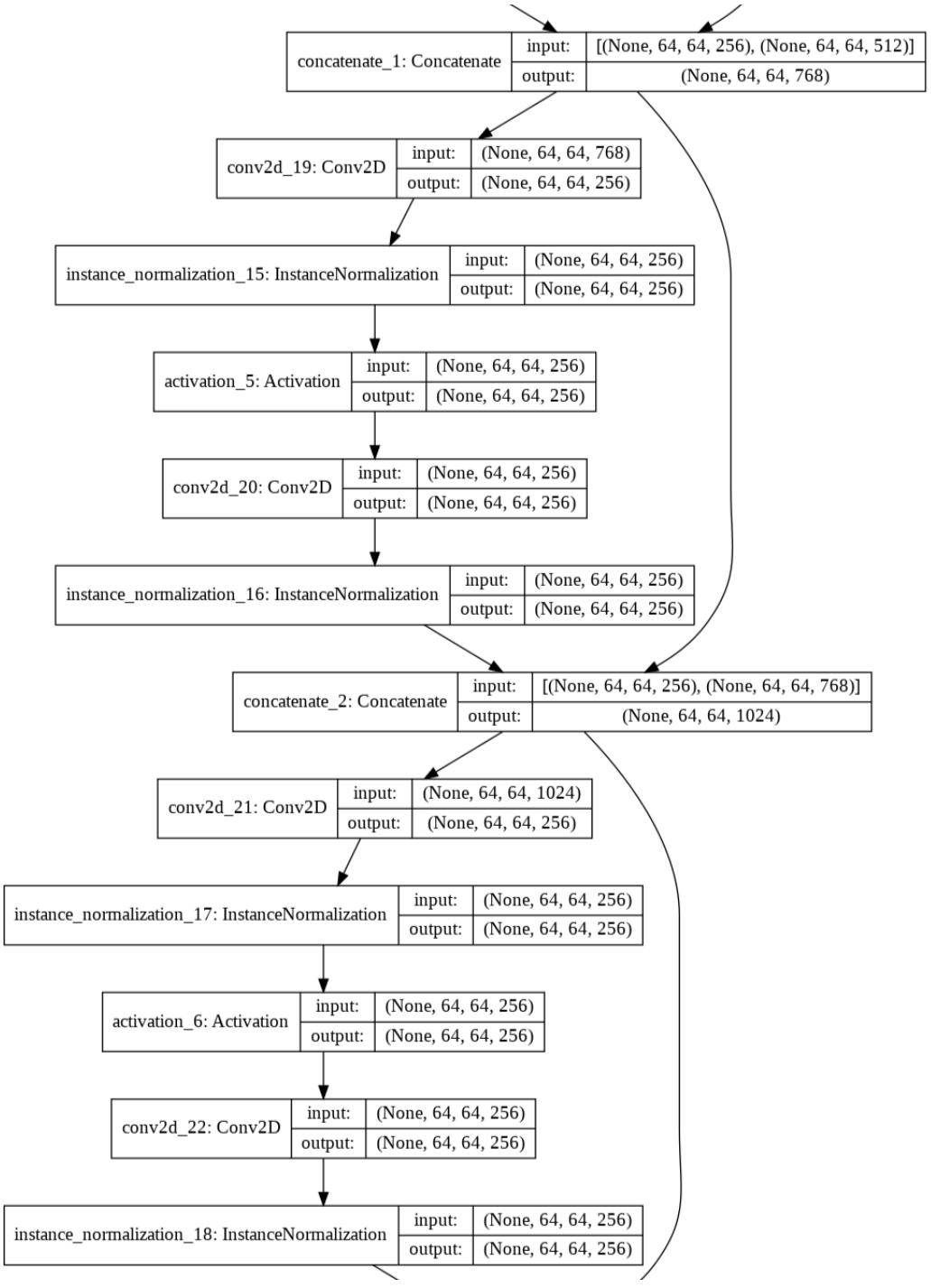


Figure A.7: Generator Architecture Diagram. Continue to Next Page.

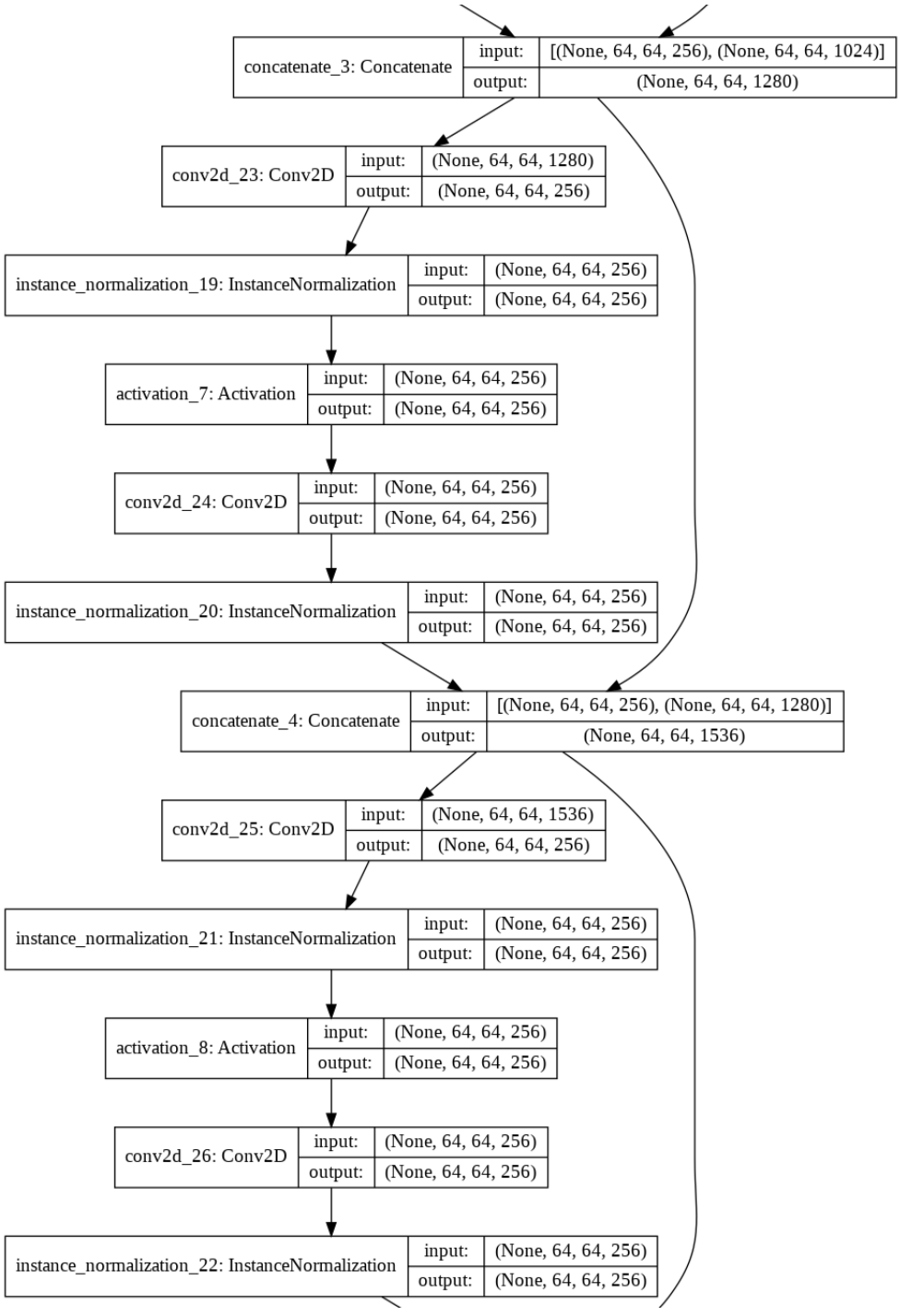


Figure A.8: Generator Architecture Diagram. Continue to Next Page.

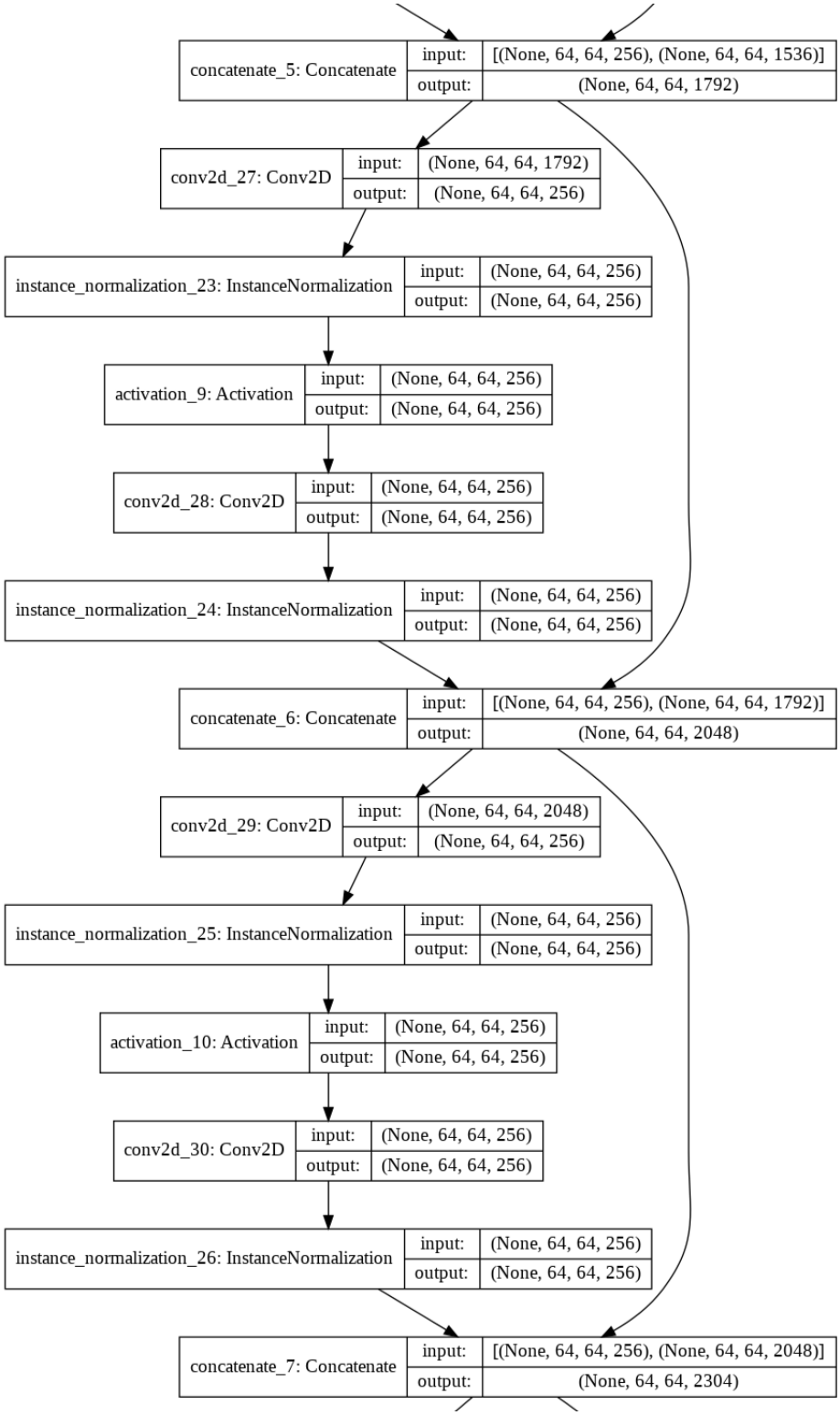


Figure A.9: Generator Architecture Diagram. Ends Here.

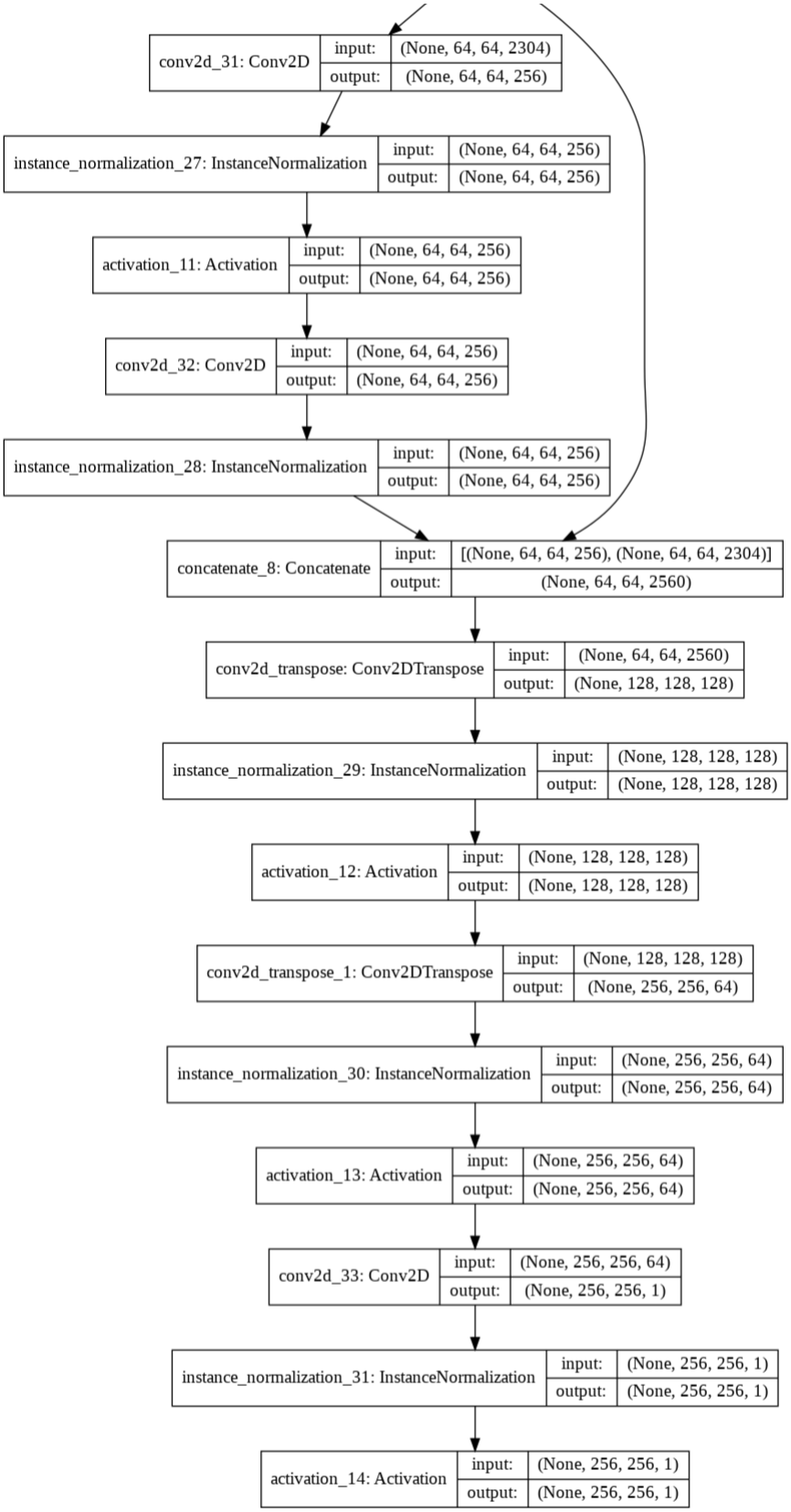


Figure A.10: Generator Architecture Diagram. Ends Here.

A.4 Discriminator Architecture Diagram

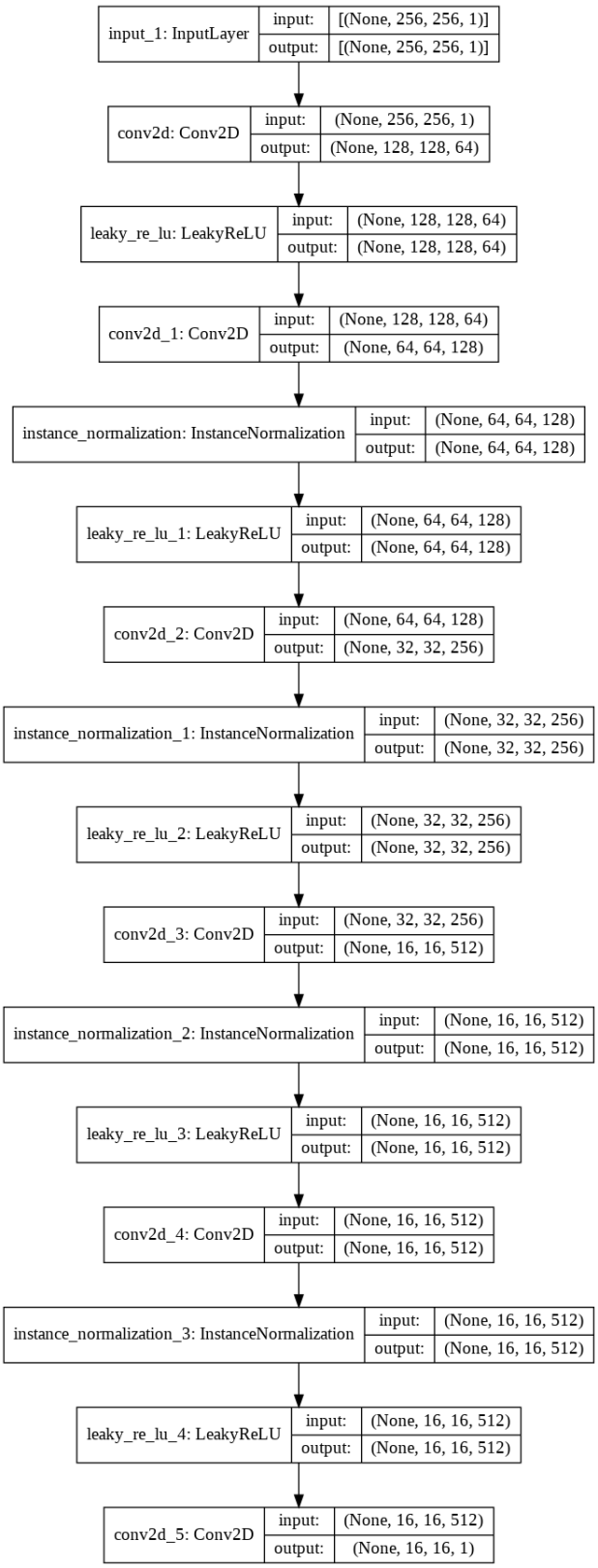


Figure A.11: Discriminator Architecture Diagram.

Bibliography

[AAB+15] 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.

[ACB17] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017.

[BMDR12] Yoshua Bengio, Grégoire Mesnil, Yann Dauphin, and Salah Rifai. Better mixing via deep representations, 2012.

[BTLAY14] Yoshua Bengio, Eric Thibodeau-Laufer, Guillaume Alain, and Jason Yosinski. Deep generative stochastic networks trainable by backprop, 2014.

[COR+16] 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.

[DBP+17] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. Adversarially learned inference, 2017.

[DKD17] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning, 2017.

[FGW+18] 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.

[Fuk80] 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.

[GAA+17] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans, 2017.

[GBB11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. JMLR Workshop and Conference Proceedings.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[GPAM+14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[HKS20] Patrick Hemmer, Niklas Kühl, and Jakob Schöffer. Deal: Deep evidential active learning for image classification, 2020.

[HRU+18] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2018.

[HW68] 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.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[IZZE18] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2018.

[JAFF16] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution, 2016.

[KH+09] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[KSH12a] 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 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[KSH12b] 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.

[LBBH98] 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.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[LBK18] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. Unsupervised image-to-image translation networks, 2018.

[LCY14] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network, 2014.

[LT16] Ming-Yu Liu and Oncel Tuzel. Coupled generative adversarial networks, 2016.

[LTM+19] 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.

[LW16] Chuan Li and Michael Wand. Precomputed real-time texture synthesis with markovian generative adversarial networks, 2016.

[LYWW11] 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.

[MC12] Giorgio Metta and Angelo Cangelosi. *Cognitive Robotics*, pages 613–616. Springer US, Boston, MA, 2012.

[MG19] P Manisha and Sujit Gujar. Generative adversarial networks (gans): What it can generate and what it cannot?, 2019.

[MLX+17] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks, 2017.

[MPPSD17] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks, 2017.

[NH10] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress.

[PEZZ20] Taesung Park, Alexei A. Efros, Richard Zhang, and Jun-Yan Zhu. Contrastive learning for unpaired image-to-image translation, 2020.

[RDS+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge, 2015.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[RMC16] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.

[RMH+20] Ievgen Redko, Emilie Morvant, Amaury Habrard, Marc Sebban, and Younès Bennani. A survey on domain adaptation theory: learning bounds and theoretical guarantees, 2020.

[Rud17] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

[RZL17] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions, 2017.

[SAH10] Joshua Susskind, Adam Anderson, and Geoffrey E Hinton. The toronto face dataset. Technical report, Technical Report UTML TR 2010-001, U. Toronto, 2010.

[SK19] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, 2019.

[SPT+17] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Josh Susskind, Wenda Wang, and Russ Webb. Learning from simulated and unsupervised images through adversarial training, 2017.

[SVV19] Monika Sharma, Abhishek Verma, and Lovekesh Vig. Learning to clean: A gan perspective, 2019.

[TBSM19] 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.

[TPW16] Yaniv Taigman, Adam Polyak, and Lior Wolf. Unsupervised cross-domain image generation, 2016.

[TTT20] Hoang Thanh-Tung and Truyen Tran. On catastrophic forgetting and mode collapse in generative adversarial networks, 2020.

[vdOLV19] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding, 2019.

[YG14] 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.

[YNDT18] Rikiya Yamashita, Mizuho Nishio, Richard Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, 9, 06 2018.

[YSZ+16] 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.

[ZKSE18] Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A. Efros. Generative visual manipulation on the natural image manifold, 2018.

[ZML17] Junbo Zhao, Michael Mathieu, and Yann LeCun. Energy-based generative adversarial network, 2017.

[ZPIE20] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2020.

[ZQD+20] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning, 2020.