Removal of Imperfections in Digital Scans using Generative Adversarial Networks

Bachelor's Thesis of

Mohamad Algoabra

At the Department of Informatics and Electrical Engineering
University of Rostock

Matrikel-Nr.: 219204154
Birthday: 18. October 1998
First reviewer: Dr.-Ing. Hannes Grunert
Second reviewer: Dr.-Ing. Holger Meyer

24. October 2022 – 13. March 2023

Faculty of Computer Science and Electrical Engineering
Institute of Computer Science
University of Rostock
18059 Rostock

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Rostock, 13.03.2023



. . . . . .

(Mohamad Algoabra)

Abstract

Document digitization is becoming increasingly important as organizations and institutions move towards a more digital workflow. However, scanned document images often suffer from damage caused by different real-life scenarios, which makes them difficult to read and use. In this thesis, we explore the use of generative adversarial networks (GANs) to enhance scanned images and remove imperfections, such as coffee stains and other distortion factors. The problem is formulated as an image-to-image translation task between two domains, and we compare the performance of two GAN types: Pix2Pix, a supervised image-to-image translation model that uses paired data, and CycleGAN, an unsupervised image-to-image translation model that uses unpaired data.

To address this problem, we developed a data pipeline to generate appropriate data to train the aforementioned models. Furthermore, we developed a prototype that allows users to easily test out these models. The effectiveness of the proposed methods was evaluated in detail using various criteria, such as Peak Signal-to-Noise Ratio (PSNR), Structural Similarity Index Measure (SSIM), and Fréchet Inception Distance (FID). We also assessed the impact of these approaches on improving Optical Character Recognition (OCR) efficiency. The results showed that Pix2Pix can significantly improve the quality of scanned images and remove defects such as coffee stains, while CycleGAN performed averagely.

Overall, this study provides a perspective on improving the digitization process by using GANs to address image imperfections. The data pipeline and prototype developed in this work can be used to improve the quality of scanned images and facilitate the transition to a more digital workflow. Future research could explore the possibility of further improving the performance of this method by incorporating other types of GANs or alternative approaches.

Keywords: Document enhancement, Degraded document binarization, Scanned images Cleaning, Generative Adversarial Networks, Pix2Pix, CycleGAN

Acknowledgment

I would like to express my appreciation to my parents for their unwavering support, encouragement, and patience during this period. Their love, understanding, and moral support were indispensable to keeping me motivated and focused on my goals. Their belief in me never wavered, and I am grateful for the sacrifices they made for me.

I would also like to extend my sincere gratitude to my supervisor Dr. Ing. Hannes Grunert, for his invaluable guidance, support, and cooperation throughout my thesis writing journey. His expertise and willingness to share his knowledge have been instrumental in shaping my research and helping me achieve my academic goal. I am also deeply grateful to Daniel Pokrandt for providing me with the necessary equipment and resources to conduct my experiments, which greatly contributed to my research.

Once again, thank you to my supervisor and my parents for their invaluable contributions to my thesis writing.

Contents

Abstract i

1. Introduction 1
- 1.1. Motivation . . . 1
- 1.2. Problem Formulation . . . 2
- 1.3. Objectives of the Thesis . . . 2
- 1.4. Structure of the Thesis . . . 3

2. Theoretical Foundation 5
- 2.1. Artificial Neural Network . . . 5
  - 2.1.1. Activation Functions . . . 6
  - 2.1.2. Loss Functions . . . 7
  - 2.1.3. Optimizers . . . 8
  - 2.1.4. Normalization . . . 9
- 2.2. Convolutional Neural Network . . . 10
  - 2.2.1. VGG and VGG Loss . . . 11
  - 2.2.2. ResNet and Residual Blocks . . . 12
- 2.3. Autoencoders . . . 13
  - 2.3.1. Convolutional Autoencoders . . . 14
  - 2.3.2. U-Net . . . 14
  - 2.3.3. Variational Autoencoders . . . 15
- 2.4. Generative Adversarial Network . . . 16
  - 2.4.1. Generative Adversarial Network Training . . . 17
  - 2.4.2. GAN Loss Functions . . . 17
  - 2.4.3. Conditional Generative Adversarial Network . . . 20
- 2.5. Image-to-Image Translation . . . 21

3. Literature Review 23
- 3.1. Search Process . . . 23
- 3.2. The Results . . . 25
- 3.3. Selected Approaches . . . 29

4. Methods 31
- 4.1. General remarks . . . 31
  - 4.1.1. Hardware environment . . . 31
  - 4.1.2. Software environment . . . 33
- 4.2. Pix2Pix . . . 35
  - 4.2.1. Generator architecture . . . 35

4.2.2. Discriminator architecture . . . 36
4.2.3. Loss function . . . 38
4.2.4. Implementation . . . 38
4.3. CycleGAN . . . 44
4.3.1. Generator architecture . . . 45
4.3.2. Discriminator architecture . . . 45
4.3.3. Loss function . . . 46
4.3.4. Implementation . . . 47
4.4. Data Preparation . . . 53
4.4.1. Raw Data Creation: . . . 53
4.4.2. Data Pipeline . . . 54
4.4.3. Data Preprocessing . . . 57
4.5. Prototype . . . 58
4.5.1. The Sidebar . . . 58
4.5.2. The Main Viewing Area . . . 60
4.6. System Architecture . . . 61

5. Evaluation setup **63**
5.1. Testing Dataset . . . 63
5.2. Quantitative Evaluation Metrics . . . 64
5.3. OCR Evaluation Metrics . . . 66
5.4. Models and Training details . . . 67

6. Experiments and Results **69**
6.1. Experiments scenario . . . 69
6.2. Results . . . 69

7. Conclusion and Future work **79**
7.1. Conclusion . . . 79
7.2. Future work . . . 80

A. Test Cases Appendix **89**

B. Results Appendix **129**

C. Code Appendix **149**

List of Figures

1.1. Examples of the documents . . . 1

2.1. The different elements of the neuron . . . 5
2.2. A multi-layer network that has an input layer, two hidden layers, and an output layer. . . . 6
2.3. How the convolution layers work . . . 10
2.4. The architecture of the Residual Block . . . 12
2.5. The architecture of Autoencoders . . . 13
2.6. The architecture of Unet . . . 14
2.7. The architecture of GAN . . . 16
2.8. The architecture of Conditional GAN . . . 20
2.9. Some examples for I2I translation applications . . . 21

4.1. The architecture of Pix2Pix . . . 35
4.2. Pix2Pix U-Net Generator . . . 36
4.3. Pix2Pix Discriminator (PatchGAN) . . . 37
4.4. The architecture of CycleGAN . . . 44
4.5. CycleGAN Generator . . . 45
4.6. Cycle consistency loss . . . 46
4.7. Examples for the raw data . . . 53
4.8. The default data pipeline . . . 54
4.9. The default data pipeline . . . 55
4.10. Examples for Scanning Effects . . . 55
4.11. Examples for degraded paper textures and coffee stains . . . 56
4.12. Examples for the default pipeline outputs . . . 56
4.13. Prototype Interface . . . 58
4.14. Prototype Sidebar File uploader and Model selector . . . 59
4.15. Prototype Sidebar Automatic adjustments . . . 59
4.16. Prototype Sidebar Manual adjustments . . . 60
4.17. Prototype Main Viewing Area . . . 60
4.18. System Architecture . . . 61

6.1. PSNR line chart (higher is better) for test case 1 . . . 71
6.2. SSIM line chart (higher is better) for test case 1 . . . 72
6.3. FID line chart (lower is better) for test case 1 . . . 73
6.4. Test case 1 . . . 73

A.1. PSNR line chart (higher is better) for test case 2 . . . 90
A.2. SSIM line chart (higher is better) for test case 2 . . . 91

A.3. FID line chart (lower is better) for test case 2 91
A.4. Test case 2 92
A.5. PSNR line charts (higher is better) for test case 3 98
A.6. SSIM line charts (higher is better) for test case 3 99
A.7. FID line charts (lower is better) for test case 3 100
A.8. Test case 3 100
A.9. PSNR line charts (higher is better) for test case 4 108
A.10. SSIM line charts (higher is better) for test case 4 108
A.11. FID line charts (higher is better) for test case 4 109
A.12. Test case 4 110
A.13. PSNR line charts (higher is better) for test case 5 116
A.14. SSIM line charts (higher is better) for test case 5 116
A.15. FID line charts (higher is better) for test case 5 117
A.16. Test case 5 118
A.17. PSNR line charts (higher is better) for test case 6 124
A.18. SSIM line charts (higher is better) for test case 6 124
A.19. FID line charts (higher is better) for test case 6 124
A.20. Test case 6 125

1. Introduction

This chapter will reveal the motives and objectives of the project along with the formulation of the main problem that will be addressed in this thesis.

1.1. Motivation

Digitization has become an intertwined part of all aspects of modern life, especially in industry, because the outcome of digitalization is higher process efficiency, lower transaction costs, and better control [1]. Today, all sectors depend on automating most data-related processes, such as financial market or patient data. However, these data are not always present in digital form and can be easily processed because documents and paper transactions still play a major role in various fields. There are also a huge number of ancient documents that must be archived and preserved for future generations. These documents are often damaged for many reasons, such as scratches, wrinkles, dust, coffee stains, moisture, and other real-life scenarios. In addition, damage can appear in digital scans due to poor digitization conditions, such as using smartphone cameras that usually show several types of blur or shadow. Furthermore, some digital scans may contain watermarks, stamps, or other annotations [2] that cause misreading or difficulty in reading their contents, whether on humans or optical character recognition (OCR) systems that play an important role in digitizing the world. Therefore, the automatic processing of digital scans to convert them into a form that can be understood by the OCR system or by humans plays a very important role in the digital archiving of documents in all fields.



Figure 1.1.: Examples of the documents

1.2. Problem Formulation

Once we recognize the significance of properly archiving documents for future generations and expediting the processing of digital scans, which constitutes a significant portion of our daily activities, it becomes necessary to consider methods for enhancing the quality of these documents to avoid potential issues. However, addressing this matter manually can be impractical in many cases given the vast number of documents and scanned images involved. Therefore, we must seek an automated solution that is highly efficient. In this thesis, we endeavored to find such a solution, which we formulated as an Image-to-Image Translation problem, where the objective is to take images from one domain and convert them in such a way that they have the style and characteristics of images from another domain. Specifically, we aim to transform the damaged scanned images, defined as the source domain, into clean images, defined as the target domain. To achieve this goal, we propose the use of Generative Adversarial Networks (GANs) as the core technology for the solution. GANs are a type of deep neural networks that can learn to generate realistic images from a given input distribution.

Research question

The research question we seek to answer in this thesis is:

Research question

What is the potential effectiveness of using Generative Adversarial Networks (GANs) to enhance the quality of scanned documents that have been damaged or contain imperfections while preserving their content? Can this methodology be considered a practical preprocessing technique for preparing documents for digital archiving?

To address this question, we will utilize and implement several advanced GAN-based image-to-image translation models, and evaluate their efficacy by means of several quantitative metrics. Furthermore, we will investigate the potential of these models in enhancing the accuracy of text extraction from OCR models. We also intend to conduct several test cases utilizing different hyperparameters to compare the efficiency of these models and determine the one that yields the best results.

1.3. Objectives of the Thesis

The aim of this study is to investigate the potential of Generative Adversarial Networks (GANs) for restoring damaged digitally scanned documents to their original state or, more precisely, enhance their quality for digital preservation. To accomplish this objective, the following steps were taken:

- Development of a scalable pipeline to generate appropriate datasets for training, validation, and testing of the models. The pipeline will produce both paired and unpaired datasets with varying types of damage and imperfections to ensure that the models are robust enough to handle a wide range of scenarios.

- Development of a framework for restoring damaged documents utilizing state-of-the-art GAN architectures specifically designed for image-to-image translation problems.

- Evaluation and comparison of the performance of multiple GAN architectures to enhance the quality of digital scanned documents in various scenarios. We will use various metrics to measure the performance of these architectures, which will enable us to identify the most effective ones for different types of digital scanned documents and scenarios.

1.4. Structure of the Thesis

This section aims to provide readers with a concise overview of the organization and content of each chapter in this thesis. By presenting a comprehensive summary of the structure and substance of each chapter, we hope to enhance readers' comprehension of the research material and facilitate easier navigation through it. Ultimately, our goal is to enable readers to better comprehend the thesis as a whole.

Chapter 2 which is the Theoretical Foundation, will offer a brief overview of the theoretical background related to this work. Specifically, this chapter will focus on deep learning concepts, generative models, and the image-to-image translation problem. We will provide a detailed explanation of generative models, particularly GANs, and how they can be applied to solve image-to-image translation problems.

Chapter 3 which is the Literature Review, will offer a comprehensive overview of the research methodology we used, the relevant work and state of the art in the image-to-image translation field. We will conduct a thorough survey of recent research in this field, examining the various techniques and algorithms that have been proposed. Furthermore, we will provide a brief explanation of some of the most promising solutions, outlining their respective strengths and weaknesses.

Chapter 4 which is Methods, we will present the models and methods that we used in this work. We will provide a detailed explanation of how these methods work and all the technical matters related to the development of the solutions. This chapter will be particularly useful for readers who are interested in understanding the technical details of the models.

Chapter 5 which is Evaluation Setup, will provide a detailed description of how we evaluated the models. We will explain the testing datasets, evaluation matrices, and model hyperparameters used in the evaluation process. We will also provide an explanation of the rationale behind our evaluation methodology.

Chapter 6 which is Experiments and Results, will provide a comprehensive comparison and evaluation of all the results obtained for each model. We will present the results of our experiments in a clear and concise manner, making it easy for readers to understand the relative performance of each model.

Finally, in **Chapter 7** which is Conclusion, we will provide a summary and conclusion of the thesis. We will discuss what we achieved in this work, highlight the contributions of our research, and provide an analysis of the effective and ineffective solutions to the research problem.

2. Theoretical Foundation

This chapter gives a brief overview of the basics and background concepts used in this work. Other aspects and additional information can be found in the resources.

2.1. Artificial Neural Network

Artificial neural networks are one of the most important concepts in deep learning today, because they are so versatile and have impressive results, which makes them ideal for handling large and complex machine learning tasks such as classifying lots of images, running speech recognition services, and recommending the best videos to watch to hundreds of millions of users or even to produce new data that is very similar to real data [3].

Inspired by biological brains, these models are designed to abstractly simulate the functions of interconnected biological neurons by passing input features through several layers that transform the input using a different set of operations as shown in the following figure:

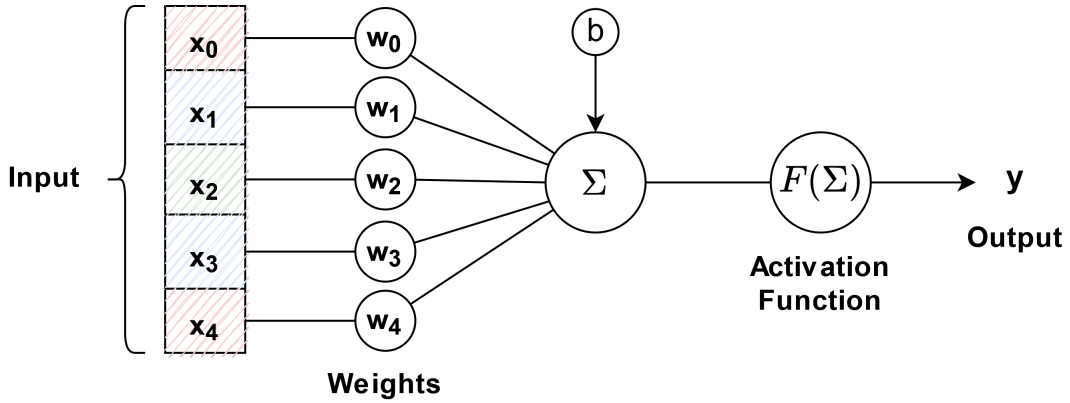


Figure 2.1.: The different elements of the neuron

Figure 2.1 describes the architecture of one simple neuron that consists of the main operations: scalar multiplication, a summation and a special function called activation function.

The neuron is defined mathematically as follows:

$$y = F(\sum_i (x_i \cdot w_i) + b) \tag{2.1}$$

So that:

1. In the beginning each feature x_i is multiplied by a certain weight w_i, where :
   - x_i is either numerical value that represent the input data, or the output of another neuron.
   - w_i is numerical value that represent either the importance of the input or the importance of the connections between the neurons.
2. Then we compute the weighted sum from the previous step and summed with the bias value and this produces a single output value.
3. Then, we apply the result of the weighted sum as an input to the activation function F, which in most cases is a non-linear function.

After we have seen how each neuron works, we can define a simple neural network as a combination of 3 sections: first the input layer that contains the properties of the input, then a multiple layers created by combining many neurons in an organized manner into multiple interconnected layers called the hidden layers and finally the output layer that specifies the output [4, 3, 5]. This can be seen more clearly in Figure 2.2.

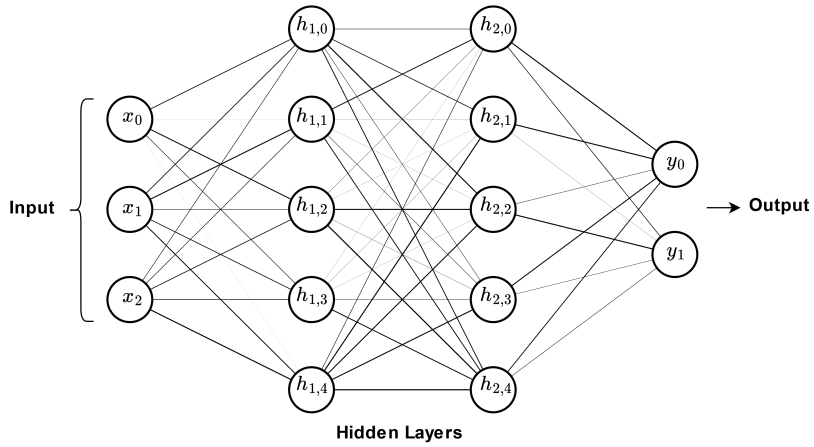


Figure 2.2.: A multi-layer network that has an input layer, two hidden layers, and an output layer.

2.1.1. Activation Functions

The Activation Functions are formulas applied to the output of a neural network's neurons to determine whether they should be activated or not, and these functions can be categorized into two primary types:

1. Linear Activation Function.
2. Non-linear Activation Functions.

But as we mentioned earlier, non-linear functions are used most of the time, because the data is not usually found in a linear form. Therefore, the use of non-linear functions allows neural networks to learn a better generalization of the data [4, 6].

These are the most common activation functions:

- Identity function:
$$F(x) = x \tag{2.2}$$
- Sigmoid:
$$F(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$
- Tanh:
$$F(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.4}$$
- ReLU:
$$F(x) = max(0, x) \tag{2.5}$$
- Leaky ReLU:
$$F(x) = max(\alpha x, x) \tag{2.6}$$
where:
  - α is a small constant, that provides a small slope to allow for the learning of negative input values instead of a flat slope in ReLU.

Each activation function has its own unique properties and can be used for specific tasks and scenarios in neural network models.

2.1.2. Loss Functions

The loss function is crucial in training a neural network, as it evaluates the quality of the model by comparing predicted and target output values. Its primary objective is to minimize the value of the loss function using gradient descent, which ultimately leads to a better model. The lower the loss function value, the more accurate the model is in predicting the target output. Hence, reducing the loss function is essential in achieving the best possible performance of the model.

These are the most common loss functions:

- **Mean Squared Error (MSE)/L2 loss:**
  MSE is a loss function that used to minimize the sum of the all the squared differences between the true value y and the predicted value $\hat{y}$.
$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.7}$$
- **Mean Absolute Error (MAE)/L1 loss:**
  MAE is a loss function that used to minimize the sum of the all the absolute differences between the true value y and the predicted value $\hat{y}$.
$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.8}$$

- **Binary Cross Entropy (BCE):**
  BCE is a loss function used in binary classification tasks that compares each of the predicted probabilities to the actual category output which can be either 0 or 1 i.e. one of two predefined categories.

$$BCE = -\frac{1}{n}\sum_{i=1}^{n} y_i \cdot log(p(\hat{y}_i)) + (1 - y_i) \cdot log(1 - p(\hat{y}_i)) \tag{2.9}$$

- **Categorical Cross Entropy (CCE):**
  CCE is a loss function used in multi-class classification tasks that compares each of the predicted probabilities to the actual category output which can one of several predefined categories.

$$CCE = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij} \cdot log(p(\hat{y}_{ij})) \tag{2.10}$$

2.1.3. Optimizers

Optimizers are algorithms used to adjust the parameters in the neural network such as weights and learning rate during the training process in order to minimize the losses that calculated by the loss functions [7], which are described in the previous subsection 2.1.2.
Some common optimizers are:

- **Gradient Descent:** This is the simplest optimizer that relies on the first-order derivative (gradient) of a loss function to determine the optimal adjustments to the model weights. The purpose of this algorithm is to minimize the loss function by iteratively updating the weights in the direction of the negative gradient [7].

- **Stochastic Gradient Descent (SGD):** It's a variation of Gradient Descent that updates the model parameters with more frequency. Unlike Gradient Descent, which updates the parameters after computing the loss on the entire dataset, SGD updates the parameters after computing the loss on each training sample. So, in a single pass through a dataset of 1000 rows, SGD will update the model parameters 1000 times, compared to just once with Gradient Descent [7].

- **Momentum:** It is a variation of SGD that considers the previous gradients to accelerate the optimization process. By using an exponentially weighted moving average of the gradients, the optimizer is able to smooth out the fluctuations and adjust the step size in the direction of the gradient. The Momentum optimizer has two main hyperparameters: the learning rate and the beta, which controls the decay rate of the momentum [8]. This helps the optimizer to escape from shallow local minima and converge faster to the global minimum.

- **AdaGrad (Adaptive Gradient):** is an adaptive learning optimization algorithm that can automatically adjust the learning rate for each parameter based on its occurrence frequency, making it suitable for sparse data since it can adaptively adjust the learning rate for each parameter based on the available information [9].

- **Adam (Adaptive Moment Estimation):** is an optimization algorithm that uses a combination of momentum and adaptive learning rates to optimize the weights of a neural network. The momentum term helps the optimizer to continue to move in the same direction as previous steps, while the adaptive learning rate adjusts the step size based on the gradient of the loss function. The algorithm uses two moving averages of the gradient and its square, to estimate the first and second moments of the gradient. These are used to calculate the learning rate for each parameter, which is then scaled by the momentum term [10, 11].

2.1.4. Normalization

Normalization techniques in neural networks are used with the purpose of standardizing and transforming inputs to a uniform scale. Typically, the goal is to center and rescale the data between 0 and 1 or -1 and 1, depending on the data type. To achieve this, a prevalent approach is to compute the mean and standard deviation of the data and normalize each sample by subtracting the mean and dividing by the standard deviation. Normalization aids in stabilizing gradient descent, facilitating the use of larger learning rates, and faster convergence of the models [12]. There are several types of normalization techniques:

- **Batch Normalization:** is a normalization technique introduced in [13] that is used to normalize the activations of each batch during training. This involves gathering all data points in a batch and normalizing them with the same mean and standard deviation for each dimension of the input [14]. It helps to reduce the internal covariate shift, which refers to the change in the distribution of activations due to the update of the parameters. This makes the training of deep networks more stable and faster.

- **Instance Normalization:** as presented in [15], is a normalization technique that normalizes the activations of each instance in the batch independently during training, instead of normalizing across the entire batch. This means that the only difference from batch normalization is that the mean and variance are not computed over the entire batch but rather only over the same data point. It is particularly useful for style transfer and image generation tasks, as it helps to reduce the style discrepancy between the input and output images [14].

- **Layer Normalization:** is a normalization technique introduced in [16] that is employs a similar technique to batch normalization, but instead of normalizing input features across the batch dimension, it normalizes the input across the features. In essence, layer normalization performs normalization on a per-data point basis. Additionally, the mean and variance are shared across all hidden units (i.e. neurons) in the layer [14].

- **Group Normalization:** is a normalization technique introduced in [17] that is normalizes across groups of channels for each training data point, making it a balance between LayerNorm and InstanceNorm. While Layer Normalization takes the channel dimension into account during computation and Instance Normalization doesn't, GroupNorm groups channels into clusters and normalizes within each group [14].

2.2. Convolutional Neural Network

While the concept of *regular neural networks* was presented in the previous section, there are different types of neural networks, which are used for different use cases and data types, and one of the most important of these types is the convolutional neural network or CNN for short, which has become famous in the field of computer vision for its superiority over the rest of the types in image processing tasks such as image classification and object recognition.

CNNs have three main types of layers that are different from a regular neural network:

1. Convolutional layers:

   The main difference between a regular neural network and a CNN is the convolution layers which are the most important building block of a CNN. A convolution layer consists of a set of filters defined by a set of learnable weights [4]. Each filter is applied across all receptive areas of the input data and learn multiple features in parallel for a given input by reducing the input to a form that is easy to process, without losing features that are critical to a good prediction.

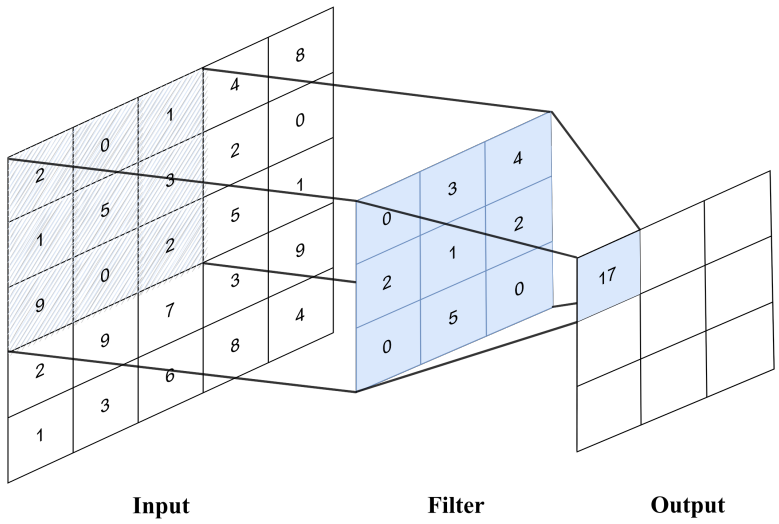


Figure 2.3.: How the convolution layers work

As shown in the diagram above, the convolutional layers are calculated as follows:

$$\begin{aligned} Output[0][0] = &(2*0)+(0*3)+(1*4) \\ &+(1*2)+(5*1)+(3*2) \\ &+(9*0)+(0*5)+(2*0) = 17 \end{aligned}$$

Then the filter will move and the second cell will be calculated until the entire input is covered.

There are three parameters that affect the size of the output and the way these layers work [18]:

a) The number of filters: affects the depth of output.

b) Stride: is the number of pixels or the step size, that the filter sliding over between each convolution.

c) Padding: is usually used when the filters do not fit the input image. This sets all elements that fall outside of the input matrix to zero, producing a larger or equally sized output.

2. Pooling layers:

Pooling layers work similarly to convolutional layers and they also reduce the dimensions of the input. They moving a filter across the entire input, but the difference is that this filter doesn't have any weights. Instead, it applies an pooling operation to the values inside the receptive area.

There are two main types of the pooling operation:

a) Max pooling: While the filter is working on the input data, it selects the pixel with the maximum value and sends it to the output.

b) Average pooling: While the filter is working on the input data, it calculates the average value within the receptive area and sends it to the output.

2.2.1. VGG and VGG Loss

VGG is a Convolutional Neural Network architecture proposed in [19]. It is a multi-layer deep neural network architecture where each block of this network consists of 2 or 3 convolutional layers and a pooling layer, and finally 2 fully-connected Layer and then the output layer and it has several variants, the most important are VGG-16 and VGG-19, which consists of 16 and 19 convolutional layers [19, 3].

VGG Loss is a content loss introduced in [20] that is based on the ReLU activation layers of the pre-trained 19-layer VGG network [19, 20].

$$\mathcal{L}_{VGG/i.j} = \frac{1}{W_{i,j}H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} (\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}(G_{\theta_G}(I^{LR}))_{x,y})^2 \tag{2.11}$$

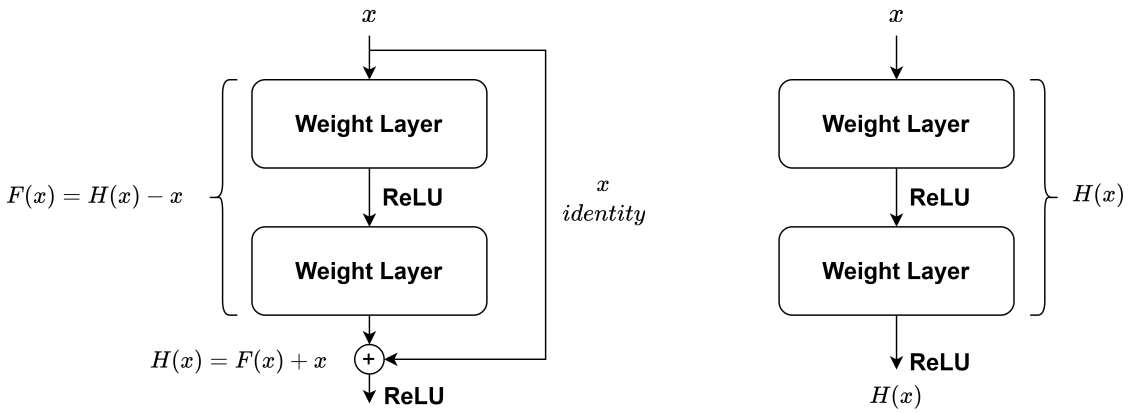
where:

- $\phi_{i,j}$ indicate the feature map obtained by the j-th convolution layer (after activation) before the i-th max pooling layer within the given VGG19 network.
- $G_{\theta_G}(I^{LR})$ is the feature representations of a reconstructed image.
- I^{HR} is the feature representations of the reference image.
- $W_{i,j}$ and $H_{i,j}$ are the dimensions of the respective feature maps within the VGG network.

2.2.2. ResNet and Residual Blocks

Residual Networks, or ResNets for short, are a Convolutional Neural Network architecture proposed in [21]. This architecture tries to solve the gradient vanishing problem and degradation problem that occurs in deep convolutional networks. The gradient vanishing occurs during backpropagation when the neural network training algorithm tries to find the weights that make the loss function reach the minimal value until the gradient becomes too small and then disappears, and the optimization cannot continue [22]. The degradation problem is that the deeper layers in the networks get a higher error rate compared to the first layers.

This approach relies on the Residual Blocks or Identity Blocks shown in Fig. 2.4, which are skip-connection blocks that learn residual functions with reference to the layer inputs, instead of learning unreferenced functions [21]. This makes gradients flow through network layers easier and ensures that important features continue to be learned through to the final layers, which allows us to build deep CNNs without the gradient vanishing problem or the degradation problem to occurs.



(a) Residual Block (b) Normal Block

Figure 2.4.: The architecture of the Residual Block

As shown in Fig. 2.4, we can see that, in addition to the normal connections, there is a skip connection that skips some layers in the model. With the skip connection The original mapping is recast into $H(x) = F(x) + x$, because when we add the input x to the network output via the skip-connection, the neural network will be forced to learn $F(x) = H(x) - x$ rather than $F(x) = H(x)$ and this called residual learning.
where:

- x is the input to the Residual block.
- $F(x)$ is a small neural network with several convolution blocks.
- $H(x)$ is the target function that the neural network will learn.

2.3. Autoencoders

Before we discuss how *Generative adversarial networks* work, we will first start with autoencoders, which have been first introduced in [23] as a unique neural network architecture consisting of two neural networks concatenated together to reconstruct the input:

1. **Encoder:** This is the first network, which compresses the input into a lower-dimensional latent code.
2. **Decoder:** This is the second network, which reconstructs the output from the lower-dimensional representation.

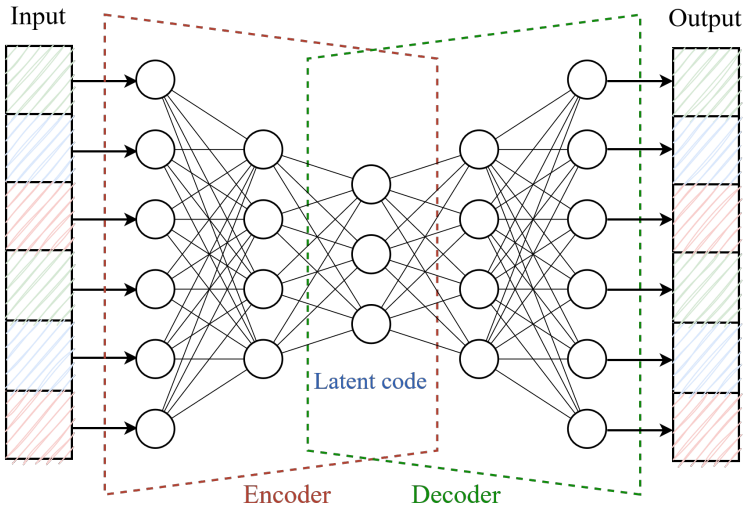


Figure 2.5.: The architecture of Autoencoders

As shown in Fig. 2.5, we can see both the encoder and the decoder are connected to create the autoencoder so that the encoder receives a d-dimensional input and encodes it into a latent code with lower dimensions. This means that the encoder will learn a function that reduces the size of the input and creates a more abstract representation of it. On the other hand, the decoder receives the latent code that has been generated by the encoder and reconstructs the original input from this low-dimensional representation, which means that it learns a function that returns the latent code to its original form before the encoding.

There are four parameters that play a role in the training of the autoencoder [24]:

1. Latent code size.
2. Number of layers, in general, both networks should have the same number of layers.
3. Number of nodes per layer.
4. Loss function, in general, MSE or BCE is used depending on the situation, for more information see this section 2.1.2.

2.3.1. Convolutional Autoencoders

As we have already seen in the *Convolutional Neural Networks section*, CNNs are the best way to process images, so if we want to build an autoencoder for image processing, we have to use the features of CNNs in it. As we've seen so far, encoders have two main networks, one to compress the input into a latent code, and the other to reconstruct that code back to the original input. From here we note that we can use both *convolutional layers* and *pooling layers* in the encoder because they reduce the dimensions of the input as required, but on the contrary, the decoder rebuilds the input that has been reduced and return it to the original size and for this we use what is called Transposed Convolutional Layers.

Transposed Convolutional Layers, in contrast to the convolutional layers, are a upsampling techniques that increases the dimensions of the input to match the output dimensions.

2.3.2. U-Net

U-Net is an architecture for semantic segmentation introduced in [25] and can be considered as a special *convolutional autoencoder* architecture using skip connections so that the structure of the network looks like the English letter "U", hence the name U-Nets. As we said earlier, this network is an improvement over the convolutional autoencoder, and therefore it contains two main networks, the encoder and decoder, which were explained in the previous section. What distinguishes this structure from its previous one is what is called skip connections, which connect each layer of the encoder with its decoder counterpart, look at Fig. 2.6. These connections pass spatial information to the decoder so that no noise occurs in the output after the reconstruction of the output because normally a lot of information will be lost during the encoding process and this improves the network output.

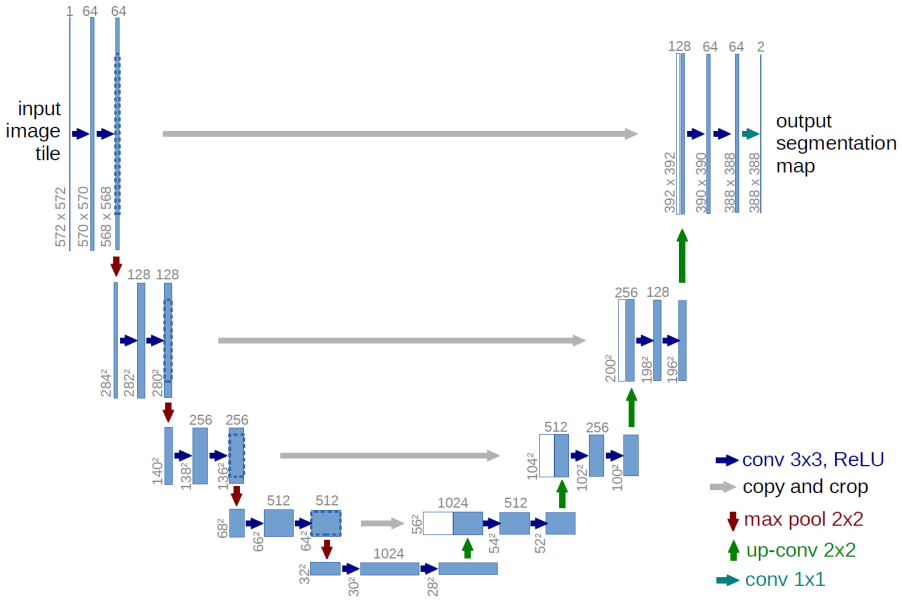


Figure 2.6.: The architecture of U-Net [25]

2.3.3. Variational Autoencoders

So far it seems that the purpose of the autoencoder is to only repeat its inputs. But this is not true, the most important feature in these models is the abstract representation of the data so that the most important properties are encoded in the latent code layer and then decoding them so that they will produce an output that carries the basic characteristics and this is very useful in denoising problems etc. But with that, the result will always the same for specific input, or in other words, there will always be a single reconstruction of one input sample. To change this, the variational autoencoders where proposed in [26]. Variational autoencoders are similar to the *autoencoders*, they consists of an encoder and a decoder, but the main difference is that instead of the encoder learning the latent code of the input, it learns the probability distribution, in other words instead of outputting the discrete values, we will have a probability distribution for each latent feature, which makes the latent space continuous. Then, the decoder can randomly select a sample of a latent variable from its distribution range and produce the output from this sample [4]. Thus, we can get many different results for a single input and for this reason, we can consider VAE as a generative model.

2.4. Generative Adversarial Network

Generative adversarial networks, or GANs for short, are deep generative models consisting of two or more neural networks. This architecture was first introduced in [27] and was built from two different neural networks each with a specific task. These two neural networks are:

1. **Generator:** This is the model that is responsible for generating new data samples by learning the distribution over the data, which is synthesized from the random vector [27]. In other words, it takes a random vector (typically Gaussian distribution) as an input and produces an output similar to the data it was trained on.

2. **Discriminator:** This is the Model that is responsible for determine whether its inputs are coming from the real database, or fake and generated by the generator.

During the training process both models are trained together as one system but the generator and the discriminator have two opposite objectives. On the one hand, the discriminator is trying to get better at distinguishing between real and fake data. On the other hand, the generator tries to generate more accurate data so that the discriminator cannot distinguish it from the real one. Thus, both models are constantly trying to outsmart each other, hence the name "adversarial" [4, 3].

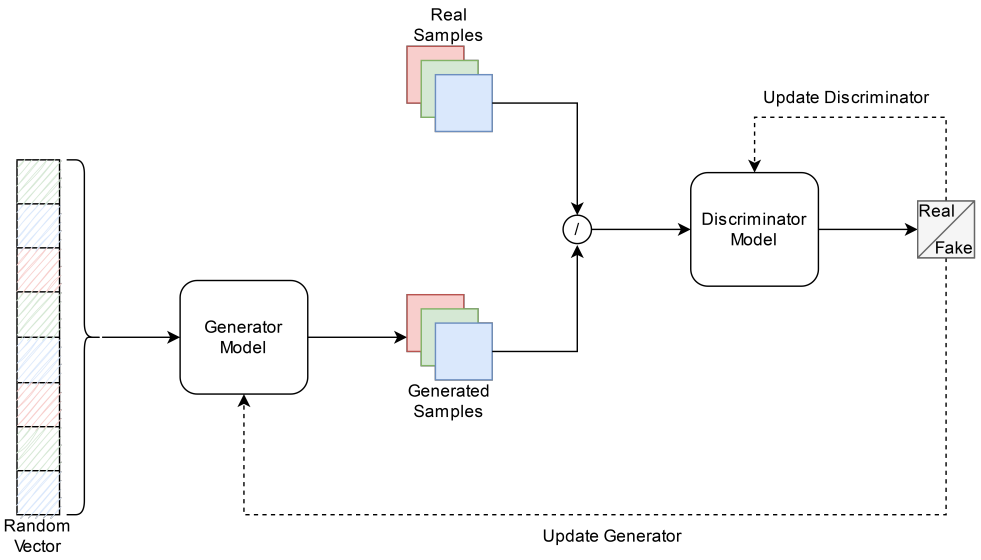


Figure 2.7.: The architecture of GAN

As shown in Fig. 2.7, we can see how the structure of the GANs is interconnected so that the generator receives a random vector and from this input it generates a data sample that flows to the discriminator and there it takes two alternating inputs the real sample and the fake sample and tries to determine these inputs. Ultimately, the goal of this architecture is to make the generator so good that the discernible cannot differentiate between its output and real data, hence the name "generative".

2.4.1. Generative Adversarial Network Training

As mentioned earlier, GANs consists of two different models with opposite goals, and for this reason, these models cannot be trained like other neural networks. The GAN training can be considered as a sequential minimax zero-sum game of two players [4] where the training proceeds in alternating phases in each iteration:

- **Phase 1:** Train the discriminator and freeze the generator, this means that at this point the backpropagation is only improving the discriminator. This phase can be divided into several steps:
  1. A number of samples are taken from the real data and set as real. In other words, the label for these samples is set to 1.
  2. With the same number of samples taken from the real data, fake samples are generated by the generator and set as fake i.e. setting label 0 for these samples.
  3. Then the discriminator starts training on this set of data for one step using the *BCE loss function*, because the discriminator considered as a binary classifier whose goal is to distinguish between real and fake data.
- **Phase 2:** Train the generator and freeze the discriminator, this means that at this point the backpropagation is only improving the generator. This phase can be divided into several steps:
  1. First, we generate a set of fake data via the generator, and they are all set as real i.e. labels 1 (this time, we don't add real data in the set).
  2. Then the discriminator is used to find out whether the data is fake or real.
  3. Then the generator starts training to generate data based on the feedback coming from the discriminator, and because the weights of the discriminator are frozen at this phase, we can improve the performance of the generator, and this is what maximizes the loss of the discriminator by making the generator better without making the discriminator worse.

As we can see, the process of training this adversarial system constitutes a min-max game, so that the goal is to minimize the generator loss and maximize the loss of the discriminator at the same time, and for this, distinctive loss functions were created to train this type of neural network architecture.

2.4.2. GAN Loss Functions

Here we will show some loss functions that are designed to train GANs:

- **Min-Max GAN Loss:**

  The Min-Max loss function is the standard loss function for GANs which was introduced in [27], the original paper that introduced this type of neural network.

This loss function is used so that the generator tries to minimize it while the discriminator tries to maximize it.

$$\mathcal{L}_{GAN}(G, D) = \min_G \max_D \mathbb{E}_x[log(D(x))] + \mathbb{E}_z[log(1 - D(G(z)))] \tag{2.12}$$

where:

- $\mathbb{E}_x$: is the expected value over all real data samples.
- $\mathbb{E}_z$: is the expected value over all random inputs to the generator.
- $D(x)$: is the discriminator's estimate of the probability that a sample of real data x is real.
- $G(z)$: is the generator's output when given noise z.
- $D(G(z))$: is the discriminator's estimate of the probability that a fake instance is real.

- **Non-Saturating GAN Loss:**

  Another alternative to the standard loss function was introduced in the original paper is the Non-Saturating GAN Loss, which modifies the generator section of the previous equation 2.4.2, to overcome the saturation problem that appears in the early stages of training. When the generator is weak, the discriminant is more efficient because the generated data is significantly different from the real data. This leads to saturation in $log(1 - D(G(z)))$. To solve this problem, instead of having the generator to minimize the loss in $log(1 - D(G(z)))$, we can train it to maximize $log(D(G(z)))$. In other words, the generator will aim to maximize the probability of the generated samples being real and not to minimize the probability of images being predicted as fake [27, 28].

- **Least Squares GAN Loss:**

  Least Squares GAN or LSGAN loss is a type of loss function used in generative adversarial networks, which was introduced in [29]. Unlike traditional GANs that use binary cross-entropy (see equation 2.1.2) loss for the discriminator, LSGAN adopts the least squares loss function. The LSGAN loss function calculates the difference between the output of the discriminator network and the target output, which is 1 for real images and 0 for generated images [30]. The goal of the discriminator network is to output high scores for real data and low scores for fake data, while the generator network is trained to produce data that results in high scores from the discriminator. By utilizing this loss function, the model can produce more stable and high-quality results compared to traditional GANs. LSGAN loss functions are defined as follows:

$$\begin{aligned} \min_D \mathcal{V}_{LSGAN}(D) &= \frac{1}{2}\mathbb{E}_x[(D(x) - b)^2] + \frac{1}{2}\mathbb{E}_z[(D(G(z)) - a)^2] \\ \min_G \mathcal{V}_{LSGAN}(G) &= \frac{1}{2}\mathbb{E}_z[(D(G(z)) - c)^2] \end{aligned} \tag{2.13}$$

where:

- a is the label for fake data, which is 0
- b is the label for real data, which is 1
- c is the value that G wants D to believe for fake data, which is 1.

- **Wasserstein GAN Loss:**

  The Wasserstein or WGAN loss was proposed in [31]. This method is considered one of the best alternatives to the standard loss function for GANs because it improves the stability of learning during training and solves common problems that occur with GANs such as Mode Collapse and Vanishing Gradient problems. This method works in a slightly different way to the regular method for GANs in that instead of using a discriminator to classify generated data as real or fake, WGAN changes the functionality of the discriminator so that it outputs a scores that represent the realness or fakeness of a particular data instance [31, 28]. In other words, discriminator does not actually classify the inputs, instead for each sample it outputs a number between 0 and 1.

  Because the model can't really distinguish between the real and the fake, the WGAN discriminator is actually called a "critic" rather than a "discriminator" [32].

  Loss functions are defined as follows:

  - **Critic Loss:** $D(x) - D(G(z))$ so that it tries to maximize this function, which means that it tries to increase the difference between real samples and fake samples [32].
  - **Generator Loss:** $D(G(z))$ so that it tries to maximize this function, which means that it tries to maximize the discriminator's output for its fake samples [32].

2.4.3. Conditional Generative Adversarial Network

A conditional generative adversarial network, or cGAN for short, was introduced in [33]. It is a conditional type of GANs intended to control the generation of data, where both the Generator and discriminator receive some additional conditioning input information Look at Fig. 2.8. This information could be the class of the input or some other property that is correlated with the output we want, for example images from another domain for image-to-image translation. This improvement comes with faster, more stable training and better-quality results.

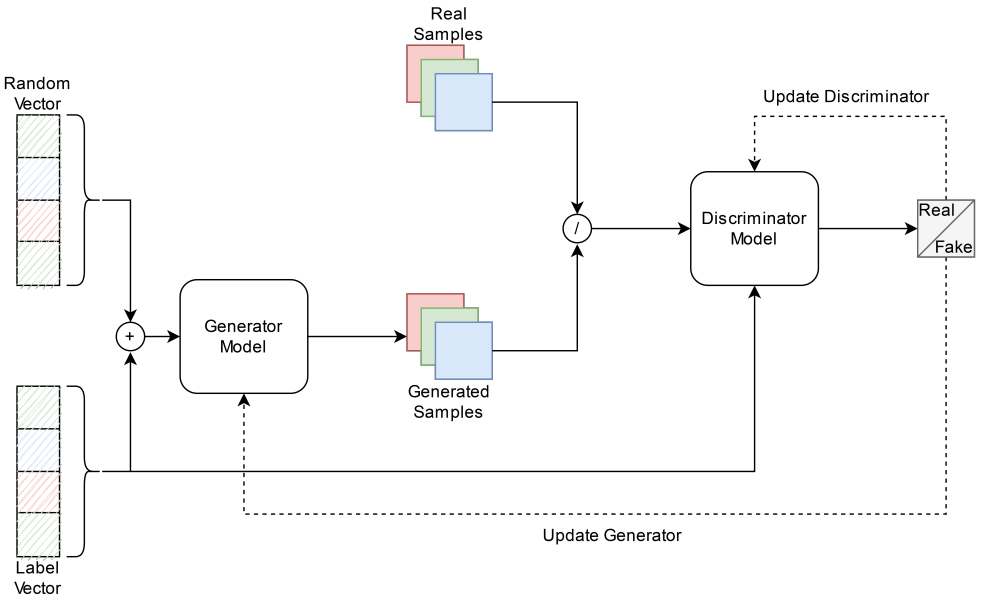


Figure 2.8.: The architecture of Conditional GAN

Compared to regular GANs, conditional GANs make a small modification to the standard GAN loss function to make it possible to include additional information by replacing the real data x with $x|y$ and the generated data z with $z|y$, which means that instead of modeling the joint probability, the conditional GAN is modeling the conditional probability.

$$\mathcal{L}_{cGAN}(G, D) = \min_G \max_D \mathbb{E}_x[log(D(x|y))] + \mathbb{E}_z[log(1 - D(G(z|y)))] \tag{2.14}$$

where:

- y: represents the additional informations.
- $D(x|y)$: is the discriminator's estimate of the probability that a sample of real data x is real under the condition of y.
- $G(z|y)$: is the generator's output when given noise z under the condition of y.
- $D(G(z|y))$: is the discriminator's estimate of the probability that a fake instance is real under the condition of y.

For more details look at the Min-Max loss function in equation 2.4.2

2.5. Image-to-Image Translation

Image-to-Image Translation is a class of computer vision and machine learning problems that focuses to transform images from one domain into another so that they have the style and properties of images from another domain, i.e. to learn the mapping between the input image and the output image. This type of problem exists in many fields and has wide uses, for example, problems of improving medical images taken with a microscope or improving images of the ocean depths which are usually blurred and many other problems can be included in this category of problems, see Fig. 2.9. In other words, the main goal of Image-to-Image Translation is to generate new images that are similar to the target domain while preserving the structure and content of the original image. In this thesis we have formulated the problem of improving scanned images as a problem of this kind, so that the damaged scanned images are the source field and we want to convert them into clean images.

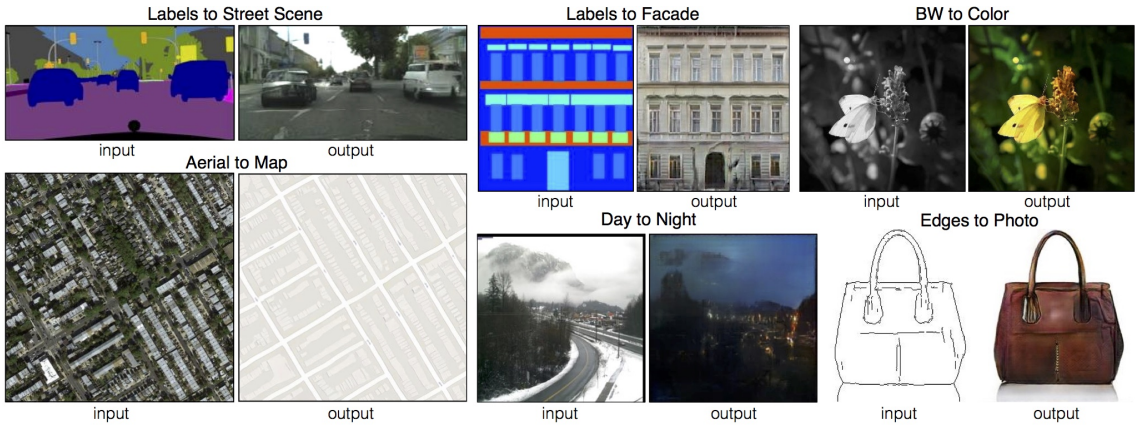


Figure 2.9.: Some examples for I2I translation applications [34]

This type of problem can be divided into two main types:

1. **Supervised Image-to-Image Translation:**

   Supervised Image-to-Image Translation aims to translate the source images to the target domain through paired datasets consisting of compatible pairs, each in a specific domain. Which means that the dataset consists of images from two different domains, each image in one domain has a corresponding image in the other domain.

2. **Unsupervised Image-to-Image Translation:**

   Unsupervised Image-to-Image Translation aims to translate the source images to the target domain through two large datasets of unrelated training images, which means that there is no matching pair for each image in the other domain. In general, most of the problems with translating images are of this type, and the reason is that in real life, pairs of images for both domains cannot always exist.

3. Literature Review

This chapter reviews the related work to the problem we are facing. It will present how the papers were collected, then briefly explain the promising results, compare them with each other, and finally choose some approaches to try in solving the problem.

3.1. Search Process

This section of the thesis provides a detailed review of the literature search process used to identify sources and research relevant to the problem of improving and removing imperfections from digitally scanned images. The literature search method has been structured in a clear and organized manner, making it easy to access the relevant works that have been identified.

1. **Problem Formulation:**

   Clearly defining the problem is a crucial initial step in the research as it establishes the foundation for the study and serves as a guide for the selection of appropriate methods and techniques. In this thesis, the problem of improving and removing imperfections from digitally scanned images was formally formulated as an image-to-image translation problem, for more details see Section 2.5. Formulating the problem of improving and removing imperfections from digitally scanned images as an image-to-image translation problem is a natural choice because it aligns with the task at hand. Image-to-image translation refers to the process of converting an image from one representation to another, such as from a low-quality scanned image to a high-quality version. In this case, the input image is a low-quality scanned image with imperfections, and the goal is to generate a high-quality version of the image that is clean. By framing the problem in this way, it allows for the use of techniques and methods developed for image-to-image translation, such as GANs, which have been shown to be effective in improving the quality of images. Additionally, this formulation also allows for the use of evaluation metrics commonly used in image-to-image translation tasks, such as peak signal-to-noise ratio (PSNR) and structural similarity index (SSIM), to measure the performance of the proposed approaches.

2. **Define Search Queries and Search Engines:**

   After the problem is appropriately defined, it is imperative to begin the process of identifying suitable search queries that can efficiently locate relevant literature and research material pertaining to the given problem. Consequently, it is important to carefully choose the most appropriate search queries that accurately correspond to the problem and its scope to ensure precision and pertinence. Considering these factors, we established the following search queries:

- "Image To Image Translation Using GANs"
- "Improve Scanned Documents with GANs"
- "GANs for Image Enhancement"
- "Removing Imperfections from Scanned Images using GANs"
- "Image Denoising with GANs"
- "Image Deblurring with GANs"
- "Supervised/Unsupervised Image To Image Translation"
- "Image Inpainting with GANs"

These search queries are relevant to the research as they focus on the use of GANs for image-to-image translation and image enhancement, which aligns with the problem of improving and removing imperfections from digitally scanned images. Additionally, these queries also include specific terms such as " improve scanned image" and "removing imperfections" which are relevant to the research problem.

After determining the search queries that will be used, it is important to identify which search engines to use to find relevant literature and research for this problem. Choosing the right search engines is essential as it can greatly impact the quality and quantity of the results obtained, so we have defined the following search engines:

Search Engine	Link
Google Scholar	https://scholar.google.com/
ACM digital library	https://dl.acm.org/search/advanced
arXiv	https://arxiv.org/search/cs
Papers with Code	https://paperswithcode.com/
IEEE Xplore	https://ieeexplore.ieee.org/

These search engines are relevant to the computer science and machine learning research as they provide access to a wide range of scholarly literature that is relevant to our problem.

3. **Quick filtering:**

After the completion of the previous steps, we commence with Quick filtering, a crucial phase in the search process as it allows us to quickly and efficiently sift through the large number of results obtained from the search engines. The goal of this step is to determine the relevance of the literature and research to the problem at hand by reading the title and abstract of the papers and articles. During this step, we have look for papers and articles that are directly related to our problem and present a promising approach to solving the problem.

This step starts by reading the title and abstract of each paper or article. The title should be informative and give a good indication of what the paper or article is about. The abstract should provide a brief summary of the paper or article and should provide enough information to determine the relevance of the work to the research problem. After reading

the title and abstract, you will then look at the applied examples of the approach that the work presents. This will give you a better understanding of how the approach is used and how well it performs. Papers and articles that do not fit the problem criteria will be excluded from further consideration.

4. **Promising Results selecting:**

   After excluding most of the papers that do not fit the criteria of the objective of this thesis, we start selecting the Promising Results by reading the remaining papers more carefully and select the most promising results from them. The goal is to identify the most relevant and promising research that aligns with the objectives of the thesis. This process involves reading the entire paper or article, including the methodology, results, and conclusions. We will then evaluate the strengths and weaknesses of the paper, as well as how well we expect it to solve the research problem. The selected papers or articles will be further analyzed and discussed in the next section.

3.2. The Results

In this section, we will discuss the promising results achieved by applying the research steps outlined in the previous section. The results we obtained are as follows:

- **Pix2Pix:**

  Pix2Pix is a GAN-based model for supervised image-to-image translation introduced in [34]. It uses a conditional GAN to learn mapping from input images to output images with corresponding features, allowing it to produce realistic results. The model was trained with a paired dataset of input and target images, enabling it to perform tasks such as image colorization, style transfer, and edge-to-image conversion. It has been used in various applications, demonstrating its effectiveness in generating accurate and visually pleasing output images.

  Pros:

  - Easy to train and efficient
  - Can be used for various image-to-image translation tasks

  Cons:

  - Requires paired training data, meaning that there needs to be a corresponding output image for each input image used during training.
  - It can handle images with a size of 256×256 pixels, which means that it has difficulty creating high-resolution images while preserving detail and output quality.
  - May produce low-quality results if the training data is not representative of the target domain.

- **Pix2PixHD:**

  Pix2PixHD introduced in [35] is a variant of pix2pix that designed to generates high-resolution images using a multi-scale generator and discriminator architecture, where

the generator generates images at multiple resolutions and the discriminator evaluates the images at each resolution.

Pros:

- Generates high-resolution images
- Can be used for various image-to-image translation tasks

Cons:

- Requires paired training data like Pix2Pix
- Computationally expensive, as it requires more resources and processing power compared to the original pix2pix model.
- May produce low-quality results if the training data is not representative of the target domain.

- **BicycleGAN:**

  BicycleGAN is a GAN-based multi-modal for supervised image-to-image translation introduced in [36]. It is designed to handle the problem of multimodal image-to-image translation, where multiple possible outputs exist for a given input. It extends Pix2Pix by learning a stochastic mapping from the source to the target, and shows interesting diversity in the generated samples.

  Pros:

  - Generates multiple possible variations for a given input.

  Cons:

  - It can be difficult to control the specific output that the model produces.
  - Requires paired training data.
  - The model may not generalize well to unseen data.
  - It may not be suitable for all types of image-to-image translation tasks.

- **CycleGAN:**

  CycleGAN is a GAN-based model for unsupervised image-to-image translation introduced in [37], which uses a cycle consistency loss to ensure that the generated images are consistent with the input images. The model consists of two generators and two discriminators, one for each domain (e.g. source and target).

  Pros:

  - Can be used for unsupervised image-to-image translation, meaning that it does not require paired training data.
  - Generates good images that are consistent with the input images.
  - The cycle consistency loss helps to ensure that the generated images are realistic.

  Cons:

  - The model may not work well when the two domains have a large domain gap.

  - May produce low-quality results if the training data is not representative of the target domain.

- **DiscoGAN:**

  DiscoGAN is a GAN-based model for unsupervised image-to-image translation introduced in [38]. It uses two generators and two discriminators, one for each domain, and two reconstruction loss, one for both the domain to improve the quality of the generated images. DiscoGAN aims to learn the cross-domain relations between the two domains and to generate images that are consistent with the input images.

  Pros:

  - Can be used for unsupervised image-to-image translation, meaning that it does not require paired training data.
  - DiscoGAN is able to discover the underlying structures of the two domains, in order to translate the images from one domain to another.

  Cons:

  - Difficulty handling large domain gaps.
  - It can handle images with a size of 64×64 pixels, which means that it has difficulty creating high-resolution images while preserving detail and output quality.
  - May produce low-quality results if the training data is not representative of the target domain.

- **UNIT:**

  UNIT is a GAN-based model for unsupervised image-to-image translation introduced in [39] that utilizes a shared-latent space to align the distributions of two domains. The main idea behind UNIT is to use two autoencoders, two generators and two discriminators for both domains, and use an adversarial loss and a cycle-consistency loss to align the distributions and preserve the structure of the images during translation.

  Pros:

  - Can be used for unsupervised image-to-image translation, meaning that it does not require paired training data.
  - It can handle multiple domains and multiple image modalities, making it a versatile method for image-to-image translation.
  - The shared-latent space allows to align multiple domains and generate new samples with structure and quality.

  Cons:

  - The results may not be as good as supervised methods when the domains have a high degree of dissimilarity.
  - It can be sensitive to the choice of the architecture and hyperparameters of the generator and discriminators.

  - May produce low-quality results if the training data is not representative of the target domain.

- **CUT:**

  CUT (Contrastive Unpaired Translation) a GAN-based model for unsupervised image-to-image translation introduced in [40] with a loss function called patch-wise contrastive loss to transfer images from a source domain to a target domain while preserving the structure of the input.

  Pros:

  - Can be used for unsupervised image-to-image translation, meaning that it does not require paired training data.
  - Faster and more efficient in terms of memory.

  Cons:

  - Model may not be able to learn the nuances of the images as well as a supervised model trained on paired data, which could lead to lower translation quality.
  - May produce low-quality results if the training data is not representative of the target domain.

Model	**Data Type**	**Networks**	**Image Size (in pixel)**	**Idea**
Pix2Pix	Paired data	1G and 1D	256×256	conditional GAN
pix2pixHD	Paired data	2G and 3D	2048×1024	multi-scale architectures
BicycleGAN	Paired data	2AE, 2G and 2D	128×128	cVAE-GAN and cLR-GAN
CycleGAN	Unpaired data	2G and 2D	256×256	cycle consistency loss
DiscoGAN	Unpaired data	2G and 2D	64×64	two reconstruction losses
UNIT	Unpaired data	2VAE, 2G and 2D	640×480	shared latent space
CUT	Unpaired data	1G and 1D	256×256	contrastive learning

Table 3.1.: This table shows which GAN methods were found, with relevant information to compare them.

3.3. Selected Approaches

In this section, we will introduce the approaches that will be utilized in this thesis. The selection is based on the different approaches presented in the previous section. We will divide the chosen methods into two categories, corresponding to the two formulations of our problem (see Section 2.5): one for Supervised Image-to-Image Translation and another for Unsupervised Image-to-Image Translation.

For supervised image-to-image translation, we have selected Pix2Pix for our problem of improving and denoising/deblurring scanned documents. This model has been shown to be very effective in a variety of image-to-image translation tasks, and so we like to test it on our problem. Pix2PixHD, which is another variant of Pix2Pix for handling high-resolution images, was initially considered, but was ultimately excluded due to its computationally expensive nature compared to Pix2Pix. As for the BicycleGAN, it is the other modification of Pix2Pix to generate several variations for each input, and this matter is not important in our case, so it was excluded.

For unsupervised image-to-image translation, we have selected CycleGAN. It is based on the idea of using a cycle-consistency loss to ensure that the translated images can be translated back to the original images, and this gives us two generators, one of which is to transfer the image from the first domain to the second, and the other does the opposite, which helps us in data augmentation for other training later. The reason for choosing CycleGAN is that it is a well-established model and has been shown to be effective in a variety of image-to-image translation tasks, so we like to test if it will be a good fit for our problem.

These methods will be used in the following chapter to put the insights gained from the literature review into practice. Therefore, the following chapter will provide a detailed description of these methods, and the evaluation chapter will outline the steps taken to ensure the validity and reliability of the results through the evaluation setup.

4. Methods

In this chapter, we present the methods and approaches used in our study to evaluate the effectiveness of GAN-based models for improving scanned images. The primary objective of this study is to test and compare the performance of different GAN architectures for this task. We have selected a combination of supervised and unsupervised approaches including Pix2Pix and CycleGAN to achieve this objective. The chapter starts by providing a detailed description of the methods, including their architectures, training procedures and implementation. We also will delve into the technical aspects of the proposed solutions, including the tools, methodologies, and techniques that will be used to bring the solution to fruition. The main goal of this chapter is to provide a clear and thorough understanding of the methods and the underlying rationale behind them.

4.1. General remarks

Initially, we will provide general remarks about the software and hardware environment used in this work. This will include the choice of programming language, libraries, and tools used for developing and training the system, as well as the specifications of the computer system used for running the model. By providing this information, we aim to provide a clear understanding of the environment used in this work.

Pix2Pix[1] and CycleGAN[2] were implemented in accordance with the references [34, 37] and their open source implementation.

All the visual aids presented in this work were created using Diagrams.net[3]

4.1.1. Hardware environment

Training hardware environment:

The hardware environment used for the training in this work consisted of a computer from the university with an AMD EPYC 7413 24-core processor, 16 GB of RAM, an RTX A6000 graphics card with 48GB of GDDR6 memory, and Google Colab Pro+. The AMD EPYC 7413 processor is a powerful server-grade processor with 24 cores and 48 threads. The high core count makes it ideal for running complex computations and multi-tasking. This processor has a base clock speed of 2.8 GHz and a boost clock speed of 3.9 GHz, which makes it efficient

[1] `https://github.com/phillipi/pix2pix`, accessed 24. Feb. 2023
[2] `https://github.com/junyanz/CycleGAN`, accessed 24. Feb. 2023
[3] `https://www.diagrams.net/`

for high-performance computing tasks like deep learning. The RTX A6000 graphics card is a high-end GPU designed for accelerating the training of deep neural networks. It features 10752 CUDA cores and 48 GB of GDDR6 memory, making it ideal for processing large batches of data during training. The large amount of memory available on this graphics card is particularly useful for processing large datasets and models. 16 GB of RAM is a relatively small amount of memory for training large machine learning models, but it may have been sufficient for the specific dataset and model used in this work. However, depending on the size of the dataset and the complexity of the model, more RAM may have been required for optimal performance. Using Google Colab Pro+ along with the university's computer hardware environment allows for a powerful and efficient hardware environment for training machine learning models. The cloud-based platform provides access to high-end hardware that can speed up the training process and reduce the time required for large-scale machine learning tasks.

The use of university hardware and Google Colab Pro+ highlights the importance of access to powerful computing resources for machine learning research. While not all researchers may have access to similar hardware environments, the use of cloud-based platforms like Google Colab Pro+ can provide a cost-effective way to access high-end hardware resources for machine learning tasks.

Testing hardware environment:

The hardware environment used for testing in this work consisted of a computer with an AMD Ryzen 9 5900HS processor, 16 GB of RAM, an RTX 3080 graphics card with 8GB of GDDR6 memory. The AMD Ryzen 9 5900HS processor is a powerful processor designed for high-end computing tasks. It features 8 cores and 16 threads. This makes it good for tasks such as testing machine learning models, as it can efficiently process large amounts of data and perform complex computations. The RTX 3080 graphics card is a high-end GPU that is good for testing of machine learning models. It features 8704 CUDA cores and 8 GB of GDDR6 memory, making it good for processing large amounts of data and performing complex calculations during testing. 16 GB of RAM is a moderate amount of memory for testing machine learning models, and it have been sufficient for the specific models and datasets used in this work. However, larger datasets or more complex models may have required additional RAM for optimal performance.

Overall, the hardware environment used for testing in this work is well-suited for machine learning tasks. The combination of the AMD Ryzen 9 5900HS processor and the RTX 3080 graphics card provides a good computing resources for efficiently processing large amounts of data and performing complex calculations. While the 16 GB of RAM is a moderate amount of memory, it may be sufficient for many machine learning tasks and datasets.

It is worth noting that the specific hardware environment used for training or testing may vary depending on the size and complexity of the models and datasets used in different machine learning projects. However, the combination of a powerful processor, high-end graphics card, and sufficient RAM is generally a good starting point for testing machine learning models.

4.1.2. Software environment

The software environment used in this work consisted of several libraries and tools that were essential for developing and testing the deep learning model. The main programming language used was Python 3.10, and several libraries including PyTorch (1.13), CUDA (11.7), NumPy (1.24.1), OpenCV (4.7.0.68), PIL (9.3.0), Seaborn (0.11.2), Plotly (5.8.0), Streamlit (1.17.0), augraphy (8.0.1), Albumentations (1.3.0), PyMuPDF (1.20.1), and pdf2image (1.16.0).

Python is a popular programming language in the field of deep learning and machine learning due to its flexibility, readability, and a wide range of available libraries and frameworks. Python 3.10 was used as the main programming language in this work because it was compatible with the required libraries, some of which did not support the latest version 3.11 at the time of this writing, so Python 3.10 was the best option to ensure compatibility with all of the libraries used.

PyTorch + CUDA were used in this work to take advantage of the powerful parallel computing capabilities of NVIDIA GPUs, which can significantly speed up deep learning training and inference. PyTorch is an open-source machine learning library based on the Torch library, which provides fast, flexible, and easy-to-use tools for building deep neural networks. PyTorch is also widely used in the deep learning community and has a large and active developer community that provides frequent updates, bug fixes, and new features. Additionally, CUDA is a parallel computing platform and programming model that enables developers to use the power of NVIDIA GPUs for high-performance computing tasks. Together, PyTorch and CUDA made it possible to train and test the deep learning model in this work efficiently and effectively.

NumPy is used in this work to perform numerical computations and operations that involve arrays and matrices, which is a common requirement in many machine learning applications, including image processing. OpenCV uses Numpy arrays to store and manipulate image data. Similarly, PyTorch relies heavily on Numpy arrays for data and matrices manipulation, as it provides efficient operations for linear algebra, Fourier analysis, and random number generation, among others. Therefore, it is an essential library for numerical computing, and its integration with other libraries enhances the efficiency and usability of the overall software environment.

OpenCV and PIL were used in this work to manipulate and process the image data for deep learning training and inference. OpenCV is an open-source computer vision library that provides tools for image and video analysis, manipulation, and processing. It offers a wide range of functions for various image processing tasks, such as resizing, cropping, and filtering images. Additionally, PIL (Python Imaging Library) is a library that adds support for opening, manipulating, and saving many different image file formats. PIL is useful for image preprocessing, which is a critical step in any image-based deep learning task, as it can improve the quality and consistency of the data. PIL provides tools for resizing, cropping, and transforming images, as well as for applying various image filters and enhancements. Together, OpenCV and PIL provided powerful tools for the image processing and preparation required for training and testing the deep learning models in this work.

Pandas was used in this work to collect and manipulate data during the evaluation process. Specifically, it was used to read in the results of the models, which were output as a CSV file, and manipulate the data into an appropriate form for analysis and visualization. Pandas is a Python library that provides easy-to-use data structures and data analysis tools, making it ideal for working with tabular data.

Seaborn and Plotly were used to visualize the results because they offer powerful and flexible tools for creating high-quality and interactive visualizations in Python. Seaborn is a Python data visualization library that is built on top of Matplotlib and provides a higher-level interface for creating informative and attractive statistical graphics. Plotly, on the other hand, is an interactive visualization library that allows users to create and share interactive, web-based visualizations in Python.

Streamlit was used to make a prototype for using pre-trained models because it provides an easy-to-use framework for creating web applications with Python. Streamlit allows users to build interactive and responsive web applications without requiring extensive knowledge of web development technologies such as HTML, CSS, or JavaScript. This makes it an ideal choice for creating quick prototypes of machine learning models, where the focus is on showcasing the functionality of the models rather than the design of the application.

Augraphy and Albumentations were used to make image augmentations and other transformations because they are powerful and efficient libraries for creating custom data augmentations for image processing tasks. Augraphy provides scanned image effects, which can generate numerous variations of an original document by simulating a variety of paper-oriented process distortions, including effects from scanning, printing, and faxing. Albumentations, on the other hand, is a fast and flexible image augmentation library that is designed for use with deep learning frameworks such as PyTorch. It offers a wide range of transformations that can be applied to images, such as rotations, flips, and color adjustments.

PyMuPDF and pdf2image were used to handle PDF files because they provide efficient tools for working with PDF documents like converting PDF files to various image formats and resolutions, which is useful in machine learning projects that involve processing document images.

4.2. Pix2Pix

Pix2Pix is a generative adversarial network (see section 2.4) introduced in [34] with the aim of being an approach to solve the supervised image-to-image translation problem (see section 2.5). It is based on the conditional generative adversarial network that has been introduced in section 2.4.3, where we condition an input image in the source domain and generate a corresponding output image in the target domain. This approach not only learns the mapping from the input image to the output image, but also it is learning the loss function to train this mapping.

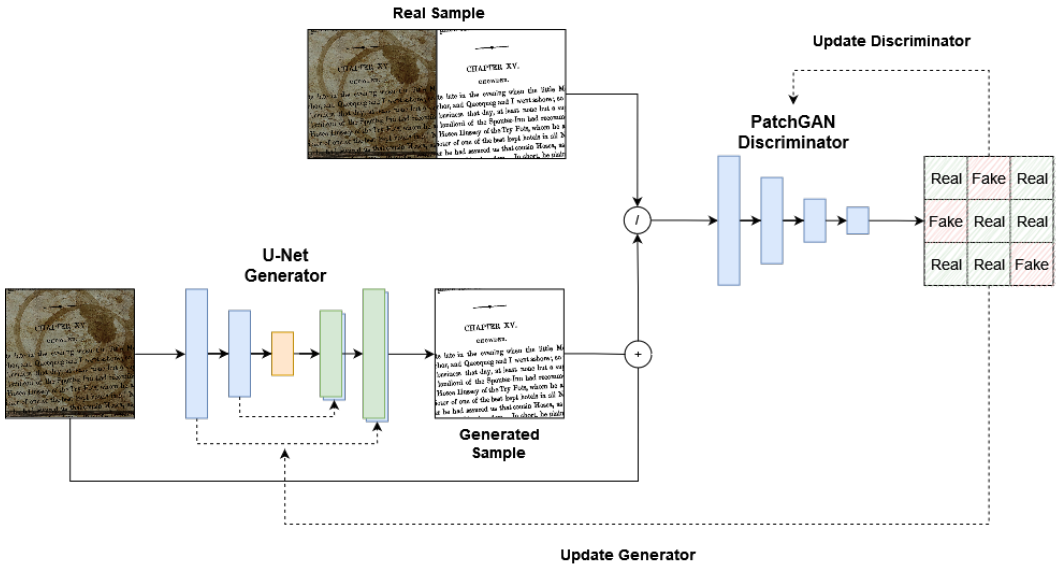


Figure 4.1.: The architecture of Pix2Pix

This allows for flexibility in applying the same approach to different image-to-image translation problems that require different loss functions [34]. As we mentioned earlier, this approach is supervised approach, which means that the training data for this model should be consist of pairs of images from different domains, with a correspondence between each pixel in the pair. The generator and discriminator are designed differently, with the generator using a U-Net architecture (see section 2.3.2) and the discriminator using a patch-based approach to give more detailed feedback on different parts of the image. Pix2Pix uses an input image, such as a damaged scanned image, as the conditioning factor for the generator to produce a realistic clean output image. The discriminator receives the input image along with either the real target output or the generated output and must determine if the image is real or fake, see Fig 4.1.

4.2.1. Generator architecture

The generator architecture in Pix2Pix is based on the U-Net architecture [34] that has been introduced in section 2.3.2. The U-Net architecture utilizes a specific type of structure known as an encoder-decoder with skip connections, in which the network's architecture is divided into two distinct portions. The first portion, referred to as the encoder, contains convolutional layers and max pool layers to progressively down-sample the input image. The second portion,

referred to as the decoder, contains layers such as convolutional transpose to upscale the image, thus increasing the resolution of the output image. The skip connections connect the corresponding blocks that have the same sizes between the two portions before going into each block in the decoder [34, 25]. This structure enables the preservation of low-level features while generating outputs. However, there are some significant distinctions between the Pix2Pix generator and the original U-Net. The encoder in Pix2Pix contains downsample-blocks that include a convolutional layer, a BatchNorm layer, and a LeakyReLU activation function. Additionally, the decoder in Pix2Pix uses upsample-blocks that are contains a transposed convolution, a BatchNorm layer, and a ReLU activation function. Moreover, dropout layers were added to the first three blocks of the decoder [34].

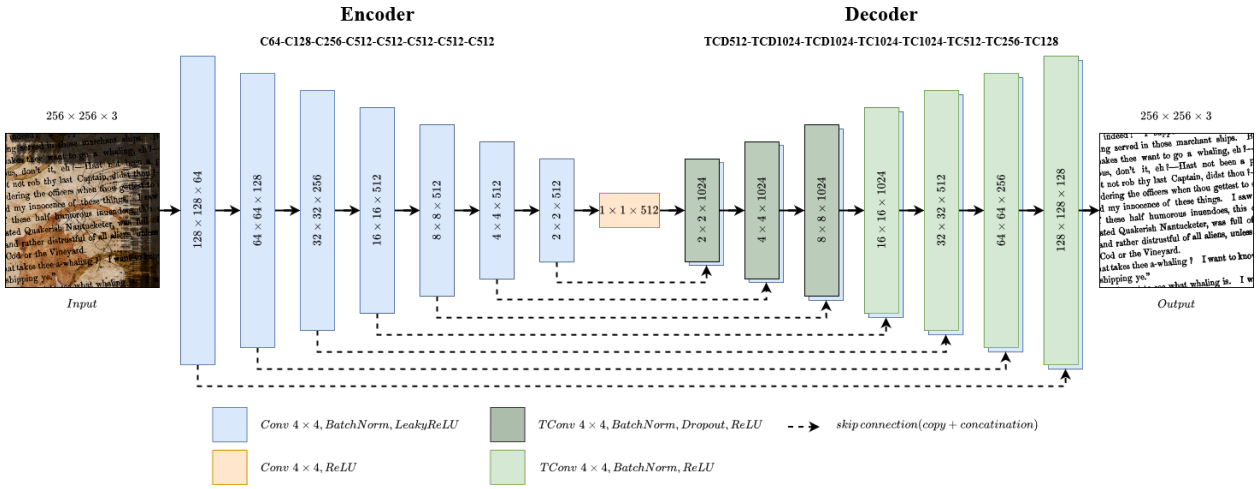


Figure 4.2.: Pix2Pix U-Net Generator

As we can see in the Fig 4.2 the encoder takes in an input image of size 256×256 pixels with 3 channels, and through eight encoder blocks, it compresses the input while decreasing the spatial size by a factor of two in each block. The final output of the encoder has a spatial size of 1×1 pixel and 512 channels. Meanwhile, on the decoder side, it starts with an input of 1×1 pixel and uses eight decoder blocks to increase the spatial size back to the original size of 256×256 pixels with 3 channels, resulting in the generated image as output.

4.2.2. Discriminator architecture

The discriminator architecture in Pix2Pix is patch-based architecture called Markovian discriminator or PatchGAN [34]. Instead of evaluating the full image, it evaluates details on a $N \times N$ patch level, determining if each $N \times N$ patch is real or fake. This results in a feature map representing the loss, with each patch in the original image corresponding to a single pixel in the loss map with value between 0 and 1 where 0 is fake and 1 is real. The primary advantages of the PatchGAN discriminator come from the facts that they have fewer training parameters, run faster, it can be applied to arbitrarily large images and It also evaluates the quality of the input image according to the quality of the local patches, rather than their global property [34, 41].

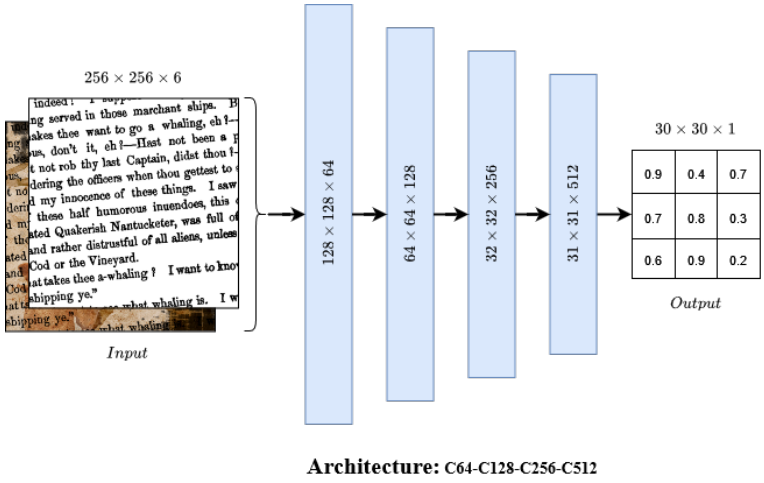


Figure 4.3.: Pix2Pix Discriminator (PatchGAN)

As we can see in the Fig 4.3 A pair of samples (one from each domain) are concatenated along the depth channel, and this 6 channels 256×256 pixels image is treated as the actual input of the PatchGAN discriminator. The discriminator network maps the input to a 1 channel 30×30 pixels image, which is the loss map that used to calculate the loss. It consists of five blocks, each containing a convolution layer with a kernel size of 4 and a stride of 2 (except for the 4th and 5th blocks, which have a stride of 1), a BatchNorm layer, and a LeakyReLU activation function (excluding the 5th block).

The patch size that has been used in PatchGAN is 70×70 and it has calculated by considering the different convolution layers in the discriminator network. For each layer, the size of the input patch corresponding to a single pixel in the output feature map is calculated using the formula:

$$input_patch_size = k + s * (output_patch_size - 1)$$

Thus, the input patch size per pixel in the output feature map for each layer in the discriminator network can be computed as follows:

- 5th layer: k=4, s=1. Input patch size is 4 (equal to kernel size).
- 4th layer: k=4, s=1. Input patch size is 4 + 1 * (4-1) = 7.
- 3rd layer: k=4, s=2. Input patch size is 4 + 2 * (7-1) = 16.
- 2nd layer: k=4, s=2. Input patch size is 4 + 2 * (16-1) = 34.
- 1st layer: k=4, s=2. Input patch size is 4 + 2 * (34-1) = 70.

Consequently, the end result is a patch size of 70×70, which are transformed into separate pixels in the 30×30 loss map. Any pixel outside the 70×70 patch will not have an impact on the corresponding pixel in the loss map [41].

4.2.3. Loss function

Pix2Pix model trained similar to conditional GAN, which we introduced in section 2.4.3, but with a slight modification. It uses a combination of adversarial loss (see section 2.4.2) and reconstruction loss. The adversarial loss is computed using a discriminator network that tries to distinguish the generated image from the target image, while the generator network tries to produce an image that can fool the discriminator under the condition of the input. The reconstruction loss is the L1 loss (see Equation 2.8) between the generated image and the target image, which helps to ensure that the generated image is visually similar to the target image. So, the loss function is as follows:

$$\mathcal{L}_{Pix2Pix}(G, D) = \min_{G} \max_{D} \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G) \tag{4.1}$$

where:

- $\mathcal{L}_{cGAN}(G, D)$ is the conditional GAN loss function, see Equation 2.14
- $\mathcal{L}_{L1}(G)$ is the L1/MAE loss function between the generated data and the ground truth data, see Equation 2.8
- λ controls the relative importance for $\mathcal{L}_{L1}(G)$,

For more details on the GANs training process see Section 2.4.1.

4.2.4. Implementation

After having reviewed the key elements of Pix2Pix, it is now appropriate to delve into its implementation with greater detail. This section will guide you through all the necessary steps for creating a Pix2Pix model, which includes constructing the generator and discriminator networks, configuring hyperparameters, and ultimately, training the model. Before delving into Pix2Pix's implementation, it is important to first introduce the libraries that will be used:

```
1 import torch
2 import torch.nn as nn
```

Listing 4.1: Libraries used in CycleGAN

where:

- **torch** is the main PyTorch package, which provides tensor computation with strong support for GPU acceleration, automatic differentiation, and various other functionalities that make it a popular choice for deep learning research and applications.
- **torch.nn** is a subpackage within PyTorch that provides various building blocks for constructing neural networks. It includes various pre-built layers (such as convolutional layers, fully-connected layers, activation functions, etc.) and tools for defining and training neural networks.

4.2.4.1. Generator network

As mentioned earlier in section 4.2.1, the construction of the generator's network involves utilizing a U-Net architecture that comprises an encoder-decoder with skip connections. The encoder part of the network consists of 8 downsample blocks that serve to reduce the dimensionality of the input image. The encoder is constructed in the following manner:

```
encoder_layers = [
  # The first downsample-block with the specified parameters
  nn.Conv2d(in_channels=3,out_channels=64,kernel_size=4,stride=2,padding=1),
  nn.LeakyReLU(0.2),
  # The second downsample-block with the specified parameters
  nn.Conv2d(in_channels=64,out_channels=128,kernel_size=4,stride=2,padding=1),
  nn.BatchNorm2d(128), nn.LeakyReLU(0.2),
  # The third downsample-block with the specified parameters
  nn.Conv2d(in_channels=128,out_channels=256,kernel_size=4,stride=2,padding=1),
  nn.BatchNorm2d(256), nn.LeakyReLU(0.2),
  # The fourth downsample-block with the specified parameters
  nn.Conv2d(in_channels=256,out_channels=512,kernel_size=4,stride=2,padding=1),
  nn.BatchNorm2d(512), nn.LeakyReLU(0.2),
  # The fifth downsample-block with the specified parameters
  nn.Conv2d(in_channels=512,out_channels=512,kernel_size=4,stride=2,padding=1),
  nn.BatchNorm2d(512), nn.LeakyReLU(0.2),
  # The sixth downsample-block with the specified parameters
  nn.Conv2d(in_channels=512,out_channels=512,kernel_size=4,stride=2,padding=1),
  nn.BatchNorm2d(512), nn.LeakyReLU(0.2),
  # The seventh downsample-block with the specified parameters
  nn.Conv2d(in_channels=512,out_channels=512,kernel_size=4,stride=2,padding=1),
  nn.BatchNorm2d(512), nn.LeakyReLU(0.2),
  # The eighth downsample-block with the specified parameters
  nn.Conv2d(in_channels=512,out_channels=512,kernel_size=4,stride=2,padding=1),
  nn.LeakyReLU(0.2)]
```

Listing 4.2: Pix2Pix encoder

As we see, this code snippet defines the encoder part of the U-Net architecture used in the Pix2Pix model. Each downsample block is defined as a series of operations on the input tensor. The operations include a 2D convolutional layer with specified parameters such as input and output channels, kernel size, stride, and padding, followed by a batch normalization layer and a leaky rectified linear unit (LeakyReLU) activation function with a negative slope of 0.2.
The first downsample block takes a 3-channel input image and outputs a 64-channel feature map with a kernel size of 4, a stride of 2, and padding of 1. Subsequent downsample blocks double the number of channels and halve the spatial dimensions of the feature maps until the eighth downsample block outputs a 512-channel feature map with a spatial dimension of 1×1 pixel. Note that the eighth downsample block does not apply batch normalization, which is a deviation from the standard U-Net architecture.

After the encoder, the generator's network includes a decoder that applies 8 upsample blocks with skip connections to reconstruct the output image from the encoded feature map. The decoder is constructed in the following manner:

```
decoder_layers = [
  # The first upsample-block with the specified parameters
  nn.ConvTranspose2d(in_channels=512,out_channels=512,kernel_size=4,stride=2,
  padding=1), nn.BatchNorm2d(512), nn.ReLU(True), nn.Dropout2d(0.5),
  # The second upsample-block with the specified parameters
  nn.ConvTranspose2d(in_channels=1024,out_channels=512,kernel_size=4,stride=2,
  padding=1), nn.BatchNorm2d(512), nn.ReLU(True), nn.Dropout2d(0.5),
  # The third upsample-block with the specified parameters
  nn.ConvTranspose2d(in_channels=1024,out_channels=512,kernel_size=4,stride=2,
  padding=1), nn.BatchNorm2d(512), nn.ReLU(True), nn.Dropout2d(0.5),
  # The fourth upsample-block with the specified parameters
  nn.ConvTranspose2d(in_channels=1024,out_channels=512,kernel_size=4,stride=2,
  padding=1), nn.BatchNorm2d(512), nn.ReLU(True),
  # The fifth upsample-block with the specified parameters
  nn.ConvTranspose2d(in_channels=1024, out_channels=256,kernel_size=4,stride=2,
  padding=1), nn.BatchNorm2d(256), nn.ReLU(True),
  # The sixth upsample-block with the specified parameters
  nn.ConvTranspose2d(in_channels=512,out_channels=128,kernel_size=4,stride=2,
  padding=1), nn.BatchNorm2d(128), nn.ReLU(True),
  # The seventh upsample-block with the specified parameters
  nn.ConvTranspose2d(in_channels=256,out_channels=64,kernel_size=4,stride=2,
  padding=1), nn.BatchNorm2d(64), nn.ReLU(True),
  # The eighth upsample-block with the specified parameters
  nn.ConvTranspose2d(in_channels=64, out_channels=3,kernel_size=4,stride=2,
  padding=1), nn.Tanh()]
```

Listing 4.3: Pix2Pix decoder

As we see, this code defines the decoder part of the U-Net architecture used in the Pix2Pix model. Each upsample block is composed of a transpose convolutional layer with specified parameters, followed by batch normalization, a ReLU activation function, and dropout regularization with a rate of 0.5 in the first three upsample blocks. The kernel size for all the upsample blocks is 4, stride is 2, and padding is 1.
The first upsample block takes a 512-channel encoded feature map and outputs a 512-channel feature map. Subsequent upsample blocks halve the number of channels and double the spatial dimensions of the feature maps until the eighth upsample block outputs a 3-channel image with the same spatial dimensions as the input image. The decoder applies skip connections to combine features from the corresponding downsample block in the encoder and the upsample block in the decoder. The final activation function used in the last upsample block is a hyperbolic tangent (Tanh) function, which maps the pixel values of the output image to the range [-1, 1]. You can find the complete generator in the appendix C.3

4.2.4.2. Discriminator network

As previously noted in section 4.2.2, the implementation of the discriminator's network consists of a series of blocks including Convolutional layer, Batch Normalization layer, and LeakyReLU activation layer as follows:

```
discriminator_layers = [
  # The first block with the specified parameters
  nn.Conv2d(in_channels=6,out_channels=64,kernel_size=4,stride=2,
  padding=1), nn.LeakyReLU(0.2, True),
  # The second block with the specified parameters
  nn.Conv2d(in_channels=64,out_channels=128,kernel_size=4,stride=2,
  padding=1),  nn.BatchNorm2d(128), nn.LeakyReLU(0.2, True),
  # The thierd block with the specified parameters
  nn.Conv2d(in_channels=128,out_channels=256,kernel_size=4,stride=2,
  padding=1),  nn.BatchNorm2d(256), nn.LeakyReLU(0.2, True),
  # The fourth block with the specified parameters
  nn.Conv2d(in_channels=256,out_channels=512,kernel_size=4,stride=1,
  padding=1),  nn.BatchNorm2d(512), nn.LeakyReLU(0.2, True),
  # The final block with the specified parameters
  nn.Conv2d(in_channels=512,out_channels=1,kernel_size=4,stride=1,
  padding=1)]
```

Listing 4.4: Pix2Pix discriminator layers

This code snippet 4.4 is composed of five blocks that define the discriminator network architecture. The first block takes a 6-channel input tensor (combining the source and target channels) and performs a 2D convolutional filter with 64 output channels, using a kernel size of 4, a stride of 2, and a padding of 1. The output then undergoes a LeakyReLU activation function with a slope of 0.2. In the second block, the output from the first block (which has 64 channels) is subjected to a 2D convolutional filter with 128 output channels, using a kernel size of 4, a stride of 2, and a padding of 1. The output is normalized via batch normalization and then passed through a LeakyReLU activation function. The third block takes the output from the second block (which has 128 channels) and applies a 2D convolutional filter with 256 output channels, using a kernel size of 4, a stride of 2, and a padding of 1. The output is normalized via batch normalization and then passed through a LeakyReLU activation function. In the fourth block, the output from the third block (which has 256 channels) is subjected to a 2D convolutional filter with 512 output channels, using a kernel size of 4, a stride of 1, and a padding of 1. The output is normalized via batch normalization and then passed through a LeakyReLU activation function. Finally, the fifth block takes the output from the fourth block (which has 512 channels) and performs a 2D convolutional filter with 1 output channel, using a kernel size of 4, a stride of 1, and a padding of 1. The resulting output is a 30×30 tensor with 1 channel, which is used by the adversarial loss to improve the GAN model.

You can find the complete discriminator in the appendix C.4

4.2.4.3. Hyperparameters

Hyperparameters are essential in determining the performance of any model and can be adjusted to optimize its training process. In the case of Pix2Pix, the following hyperparameters are utilized by default:

- **Learning rate:** The learning rate determines how quickly the model updates its internal parameters during training. A lower learning rate can lead to more precise updates, but can also increase the amount of time it takes for the model to converge on a solution. In the case of Pix2Pix, the default learning rate is set to 0.0002.
- **Batch size:** Batch size refers to the number of samples that are processed by the model during each iteration of training. A larger batch size can lead to faster training times, but may also require more memory and can make it difficult for the model to converge on a solution. In the case of Pix2Pix, the default batch size is set to 1, meaning that the model processes one image at a time during training.
- **L1 lambda:** L1 lambda is a hyperparameter that balances the weight of the L1 loss term in the generator's loss function. The L1 loss measures the difference between the generated image and the target image in terms of absolute pixel differences. A higher L1 lambda value puts more emphasis on the L1 loss term and results in sharper, more accurate images, while a lower value gives less weight to the L1 loss and can produce smoother but less accurate images. The default value for L1 lambda in Pix2Pix is set to 100.
- **Optimizer:** The optimizer is a function that determines how the model updates its internal parameters during training. Adam is a popular optimizer that is commonly used in machine learning models. In the case of Pix2Pix, Adam is utilized as the default optimizer.
- **Adam beta 1 and beta 2:** Adam beta 1 and beta 2 are hyperparameters that determine the decay rates for the first and second moments of the gradients during training, respectively. These hyperparameters impact the speed and stability of the model's training process. In the case of Pix2Pix, the default values for Adam beta 1 and beta 2 are set to 0.5 and 0.999, respectively.

4.2.4.4. Training algorithm

The training algorithm for Pix2Pix follows a similar approach to regular GANs (refer to section 2.4.1) but with some differences in the loss function (see section 4.2.3).

During training, the discriminator is updated using only adversarial loss to ensure that the generated images are realistic and indistinguishable from the real images. On the other hand, the generator is updated using both adversarial loss and L1 loss to ensure that the generated images have a similar pixel-level appearance to the ground truth images. By combining these two losses, the generator can produce high-quality translations between the two domains.

Algorithm 1 Pix2Pix training algorithm

```
1:  procedure Pix2Pix
        // Initialize discriminators networks
2:      D ← discriminator_network()
        // Initialize generators networks
3:      G ← generator_network()
4:      for each epoch do
5:          for each batch of images from domain A and domain B do
                // Train the discriminator
6:              generated_B ← G(real_A)
7:              loss_D ← adversarial_loss(real_B, target_value = 1.0)+
8:                       adversarial_loss(generated_B, target_value = 0.0)
9:              update_weights(loss_D, D.weights)
                // Train the generator
10:             generated_B ← G(real_A)
11:             loss_G ← adversarial_loss(generated_B, target_value = 1.0)+
12:                      lambda_weight * L1_loss(real_B, generated_B)
13:             update_weights(loss_G, G.weights)
14:         end for
15:     end for
16: end procedure
```

4.3. CycleGAN

CycleGAN is an approach based on generative adversarial networks introduced in [37] for solving the problem of unsupervised image-to-image translation (see section 2.5), meaning that it can be trained without the need for paired datasets. It is a bidirectional generative model based on the concept of cycle consistency. It consists of four neural networks, two generators and two discriminators. The reason for that because CycleGAN addresses the problem of image-to-image translation by treating it as an image reconstruction problem, where the input image is transformed into another domain using the first generator, and then transformed back into the original domain using the second generator. The accuracy of the reconstruction is measured by calculating the mean squared error (see section 2.1.2) between the original image and the reconstructed image. This helps to ensure that the content and structure of the original image is preserved during the translation process.

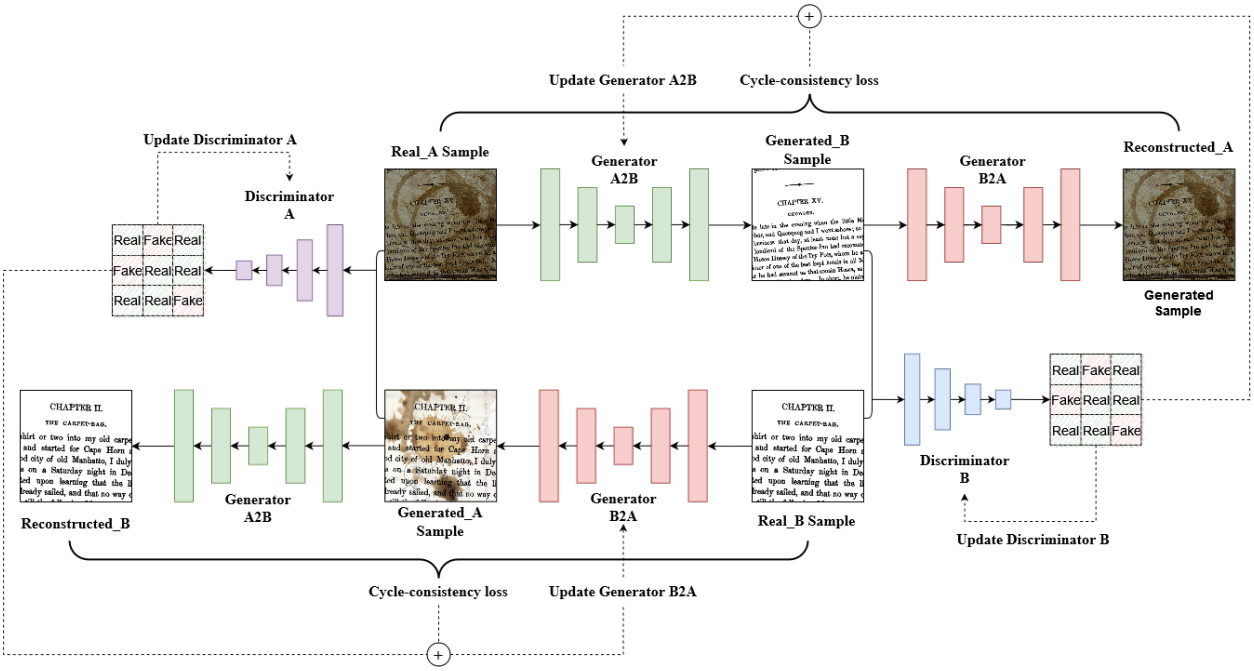


Figure 4.4.: The architecture of CycleGAN

As we can see in the Fig. 4.4, the process starts by feeding a sample from domain A into generator A2B, producing a translated image in domain B. This translated image then passed through the discriminator B and with real B samples to calculate the adversarial loss. At the same time, the translated image is also processed through generator B2A to generate a reconstructed image to calculate the cycle consistency loss between the reconstructed and original image. The adversarial loss is used to update the discriminator weights, and both the adversarial and cycle consistency losses are used to update the generator weights. This process repeats with generator B2A and discriminator A until generated domain A and B images resembling the real domains.

4.3.1. Generator architecture

The generator in CycleGAN is based on an encoder-decoder architecture, which is augmented with residual blocks from the ResNet architecture [37] that has been introduced in section 2.3 and 2.2.2. The encoder part of the generator compresses the input image into a lower-dimensional representation using downsample-blocks that include a convolutional layer, a instance normalization layer, and a ReLU activation function, while the decoder part expands the compressed representation back to the original image size using upsample-blocks that are contains a transposed convolution layer, a instance normalization layer, and a ReLU activation function. The residual blocks in the generator allow for the learning of residual functions that help improve the quality of the generated images.

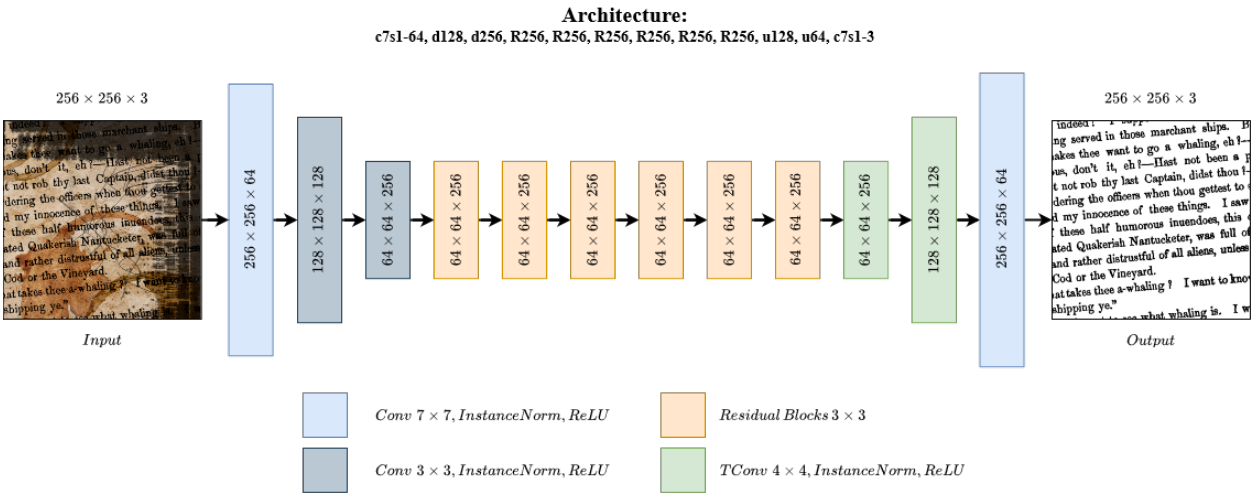


Figure 4.5.: CycleGAN Generator

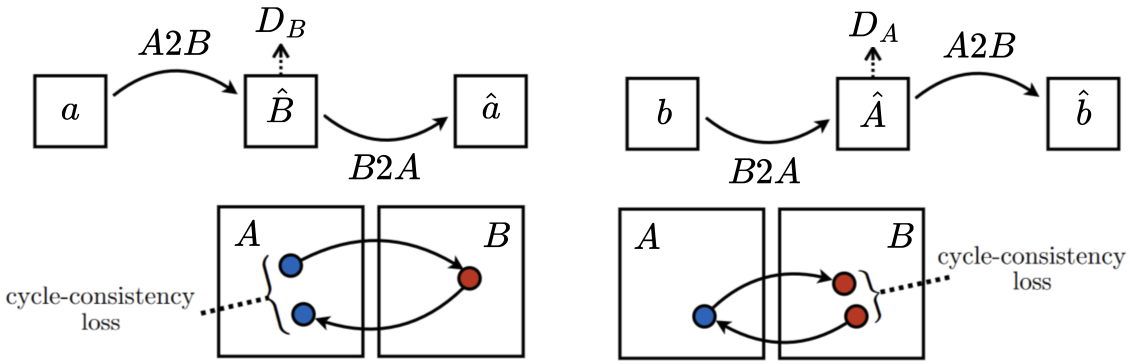
As we can see in the Fig. 4.5, the generator first compresses the input image, which has dimensions of 256×256 pixels with 3 channels, to a lower-dimensional representation of size 64×64 pixels with 256 channels through a series of three encoder blocks. This representation is then processed by a series of residual blocks, which interpret the encoding through the use of skip connections to stabilize the model and preserve the features of the original image. Finally, the generator utilizes a series of upsampling layers through the decoder part to increase the spatial resolution of the representation and produce the final output image, which has the same dimensions as the input image.

4.3.2. Discriminator architecture

The discriminator in CycleGAN is patch-based architecture called Markovian discriminator or PatchGAN [34, 37]. It's the same discriminator architecture used in the Pix2Pix method, except that the input image has a depth channel of 3, instead of 6, and it uses instance normalization layer instead of batch normalization layer. Refer to section 4.2.2 for further details.

4.3.3. Loss function

CycleGAN is trained using combination of adversarial losses for each generator and discriminator pair (see section 2.4.2), and the concept of cycle consistency loss [37], which aims to keep the output of the GAN generator consistent with the original input. The first generator converts an image from one domain to another, and the cycle consistency loss requires that when the generated image is reconstructed back to the original domain using the second generator, it should be similar to the original image.



(a) Forward Cycle Consistency

(b) Backward Cycle Consistency

Figure 4.6.: Cycle consistency loss [37]

As we can see in the Fig. 4.6, cycle consistency loss is comprised of two components for each direction, forward cycle consistency and backward cycle consistency [37].

1. Forward cycle consistency ensures that for each image a from domain A, it can be transformed back to its original form through the image translation cycle. This means that starting with an image from domain A, after transforming it through the first generator $A2B$ and then transforming the result through the second generator $B2A$, the final image should closely resemble the original image, i.e., $a \rightarrow A2B(a) \rightarrow B2A(A2B(a)) \approx a$

2. Backward cycle consistency ensures that for each image b from domain B, it can be transformed back to its original form through the image translation cycle. This means that starting with an image from domain B, after transforming it through the second generator $B2A$ and then transforming the result through the first generator $A2B$, the final image should closely resemble the original image, i.e., $b \rightarrow B2A(b) \rightarrow A2B(B2A(b)) \approx b$.

The combination of these two constraints forms the cycle consistency loss, which serves as a crucial component in maintaining the authenticity and integrity of the generated images. By ensuring that the generated images retain the semantic and structural information of the original images, the cycle consistency loss reduces the possibility of generating distorted or unrealistic images and leads to more coherent and recognizable results from the model.

The cycle consistency loss is calculated as follows:

$$\mathcal{L}_{\text{cyc}}(A2B, B2A) = \mathbb{E}_a[||B2A(A2B(a)) - a||_1] + \mathbb{E}_b[||A2B(B2A(b)) - b||_1] \tag{4.2}$$

where:

- a and b are samples from domain A and B
- $A2B$ is the first generator to convert images from domain A to domain B
- $B2A$ is the second generator to convert images from domain B to domain A
- $||...||_1$ denotes the mean absolute error, or MAE see equation 2.8
- $\mathbb{E}_a[||B2A(A2B(a)) - a||_1]$ is the forward cycle consistency
- $\mathbb{E}_b[||A2B(B2A(b)) - b||_1]$ is the backward cycle consistency

As mentioned earlier, the loss function in CycleGAN is a combination of adversarial losses for each generator-discriminator pair, and the cycle consistency loss. Therefore, the following form is the final loss function:

$$\begin{aligned}\mathcal{L}_{\text{CycleGAN}}(A2B, B2A, D_A, D_B) = &\min_{A2B,B2A}\max_{D_A,D_B} \mathcal{L}_{\text{GAN}}(A2B, D_B, A, B) \\ &+ \mathcal{L}_{\text{GAN}}(B2A, D_A, B, A) \\ &+ \lambda\mathcal{L}_{\text{cyc}}(A2B, B2A)\end{aligned} \tag{4.3}$$

where:

- $\mathcal{L}_{GAN}(A2B, D_B)$ and $\mathcal{L}_{GAN}(B2A, D_A)$ are the adversarial losses (GAN losses), see section 2.4.2
- $\mathcal{L}_{cyc}(A2B, A2B)$ is the cycle consistency loss
- λ controls the relative importance of the cycle consistency loss

This loss helps to maintain stability during the training process and ensures that the generated images have a coherent structure that resembles the original images, not just random noise.

4.3.4. Implementation

Now that we've seen the main components of CycleGAN, it's time to explore its implementation in more detail. This section covers all the steps necessary to implement a CycleGAN model, including the construction of the generator and discriminator networks, setting up the hyper-parameters, and finally, the training process. Before we delve deeper into the implementation process of CycleGAN, let's start with the used libraries, which are the same as those used to implement pix2pix:

```
1 import torch
2 import torch.nn as nn
```

Listing 4.5: Libraries used in CycleGAN

4.3.4.1. Generator network

As previously noted in section 4.3.1, the implementation of the generator's network involves the utilization of an encoder-decoder structure surrounding a several of residual blocks. The encoder network implements one initial block and two downsample blocks, which are used to reduce the input image to a lower-dimensional representation. The encoder is implemented as follows:

```
encoder_layers = [
  # The initial block with the specified parameters
  nn.ReflectionPad2d(3),
  nn.Conv2d(in_channels=3,out_channels=64,kernel_size=7,stride=1,padding=0),
  nn.InstanceNorm2d(64), nn.ReLU(True),
  # The first downsample-block with the specified parameters
  nn.Conv2d(in_channels=64,out_channels=128,kernel_size=3,stride=2,padding=1),
  nn.InstanceNorm2d(128), nn.ReLU(True),
  # The second downsample-block with the specified parameters
  nn.Conv2d(in_channels=128,out_channels=256,kernel_size=3,stride=2,padding=1),
  nn.InstanceNorm2d(256), nn.ReLU(True)]
```

Listing 4.6: CycleGAN encoder

As we see this code snippet 4.6 defines three encoder layers, including one initial block and two downsample blocks. The first block is an initial block that takes a 3-channel input tensor (representing the RGB channels of an image) and applies a reflection padding of size three to the input tensor. This is followed by a 2D convolutional layer with a kernel size of 7×7, 64 output channels, and strides of 1 and 0 padding. Then, an instance normalization layer is applied to the output tensor, followed by a rectified linear unit (ReLU) activation function to produce a 64-channel tensor. The second block is a downsample block that takes the 64-channel output tensor from the first block and applies a 2D convolutional layer with a kernel size of 3×3, 128 output channels, a stride of 2, and padding of 1. An instance normalization layer is then applied to the output tensor, followed by a ReLU activation function, producing a 128-channel tensor. The third block is also a downsample block that takes the 128-channel output tensor from the second block and applies a 2D convolutional layer with a kernel size of 3×3, 256 output channels, a stride of 2, and padding of 1. An instance normalization layer is then applied to the output tensor, followed by a ReLU activation function resulting in a 256-channel tensor.

After the encoder, the output is then processed through several residual blocks. Each residual block consists of two 3×3 convolutional layers with the same number of filters as the input tensor, two instance normalization layers, and ReLU activation function, for further details refer to section 2.2.2. Each residual block is implemented as follows:

```
class ResidualBlock(nn.Module):
  def __init__(self, channels):
    super().__init__()
    # Define the block componants
    block = [nn.Conv2d(channels, channels, 3),  # first convolutional layer
```

```
            nn.InstanceNorm2d(channels),       # instance normalization
            nn.ReLU(True),                     # ReLU activation
            nn.Conv2d(channels, channels, 3),  # second convolutional layer
            nn.InstanceNorm2d(channels)]       # instance normalization
    self.block = nn.Sequential(*block)

  def forward(self, x):
    """Forward function (with skip connections)"""
    out = x + self.block(x)  # add skip connections to the output
    return out
```

Listing 4.7: Residual Block

After the residual blocks, the output is then processed through the decoder, which consists of two upsample blocks and one final block. Each upsample block takes the output from the previous layer and increases its spatial resolution through transposed convolutional layers, instance normalization and ReLU activation function. While the final block consists of convolutional layer followed by Tanh activation function. The decoder is implemented as follows:

```
decoder_layers = [
  # The first upsample-block with the specified parameters
  nn.ConvTranspose2d(in_channels=256,out_channels=128,kernel_size=3,stride=2,
  padding=1), nn.InstanceNorm2d(128), nn.ReLU(True),
  # The second upsample-block with the specified parameters
  nn.ConvTranspose2d(in_channels=128,out_channels=64,kernel_size=3,stride=2,
  padding=1), nn.InstanceNorm2d(64), nn.ReLU(True),
  # The final block with the specified parameters
  nn.ReflectionPad2d(3),
  nn.Conv2d(in_channels=64,out_channels=3,kernel_size=7,stride=1,padding=0),
  nn.Tanh()]
```

Listing 4.8: CycleGAN decoder

This code snippet 4.8 defines three decoder layers, including two upsample blocks and one final block. The first block is an upsample block that takes a 256-channel input tensor and applies a 2D transpose convolutional layer with a kernel size of 3×3, 128 output channels, stride of 2, and padding of 1. An instance normalization layer was then applied to the output tensor, followed by a ReLU activation function that produced a 128-channel tensor. The second block is also an upsample block that takes the 128-channel output tensor from the first block and applies a 2D transpose convolutional layer with a kernel size of 3×3, 64 output channels, stride of 2, and padding of 1. An instance normalization layer was then applied to the output tensor, followed by a ReLU activation function, resulting in a 64-channel tensor. The third block is the final block that takes the 64-channel output tensor from the second block and applies a reflection padding of size 3 to the input tensor. This is followed by a 2D convolutional layer with a kernel size of 7×7, three output channels (representing the RGB channels of an image), a stride of one, and no padding. Finally, the Tanh activation function was applied to the output tensor.

Now that we have all the components, we can put everything together to build the CycleGAN generator model. You can find the complete generator in the appendix C

4.3.4.2. Discriminator network

As previously noted in section 4.3.2 and section 4.2.2, the implementation of the discriminator's network consists of a series of blocks including Convolutional layer, Instance Normalization layer, and LeakyReLU activation layer as follows:

```
discriminator_layers = [
  # The first block with the specified parameters
  nn.Conv2d(in_channels=3,out_channels=64,kernel_size=4,stride=2,
  padding=1), nn.LeakyReLU(0.2, True),
  # The second block with the specified parameters
  nn.Conv2d(in_channels=64,out_channels=128,kernel_size=4,stride=2,
  padding=1),  nn.InstanceNorm2d(128), nn.LeakyReLU(0.2, True),
  # The thierd block with the specified parameters
  nn.Conv2d(in_channels=128,out_channels=256,kernel_size=4,stride=2,
  padding=1),  nn.InstanceNorm2d(256), nn.LeakyReLU(0.2, True),
  # The fourth block with the specified parameters
  nn.Conv2d(in_channels=256,out_channels=512,kernel_size=4,stride=1,
  padding=1),  nn.InstanceNorm2d(512), nn.LeakyReLU(0.2, True),
  # The final block with the specified parameters
  nn.Conv2d(in_channels=512,out_channels=1,kernel_size=4,stride=1,
  padding=1)]
```

Listing 4.9: CycleGAN discriminator layers

As we see this code snippet 4.9 defines five blocks. The first block of the network takes a 3-channel input tensor and applies a 2D convolutional filter with 64 output channels, a kernel size of 4, stride of 2, and padding of 1. The resulting output is then passed through a LeakyReLU activation function with slope of 0.2. The second block takes the output of the first block, which has 64 channels, and applies a 2D convolutional filter with 128 output channels, a kernel size of 4, stride of 2, and padding of 1. The output is then normalized using instance normalization and passed through a LeakyReLU activation function. The third block takes the output of the second block, which has 128 channels, and applies a 2D convolutional filter with 256 output channels, a kernel size of 4, stride of 2, and padding of 1. The output is then normalized using instance normalization and passed through a LeakyReLU activation function. The fourth block takes the output of the third block, which has 256 channels, and applies a 2D convolutional filter with 512 output channels, a kernel size of 4, stride of 1, and padding of 1. The output is then normalized using instance normalization and passed through a LeakyReLU activation function. Finally, the fifth block takes the output of the fourth block, which has 512 channels, and applies a 2D convolutional filter with 1 output channel, a kernel size of 4, stride of 1, and padding of 1. The resulting output is a 30×30 with 1-channel tensor, which will used by the adversarial loss to improve the GAN model. You can find the complete discriminator in the appendix C

4.3.4.3. Hyperparameters

Hyperparameters are essential in determining the performance of any model and can be adjusted to optimize its training process. In the case of CycleGAN, the following hyperparameters are utilized by default:

- **Learning rate:** The learning rate determines how quickly the model updates its internal parameters during training. A lower learning rate can lead to more precise updates, but can also increase the amount of time it takes for the model to converge on a solution. In the case of CycleGAN, the default learning rate is set to 0.0002.
- **Batch size:** Batch size refers to the number of samples that are processed by the model during each iteration of training. A larger batch size can lead to faster training times, but may also require more memory and can make it difficult for the model to converge on a solution. In the case of CycleGAN, the default batch size is set to 1, meaning that the model processes one image at a time during training.
- **Cycle lambda:** Cycle lambda is a hyperparameter that is specific to CycleGAN. It determines the weight given to the cycle consistency loss during training, which is a measure of how well the model is able to map images between the two sets in both directions. A higher cycle lambda value can lead to a more accurate mapping between image sets, but can also make it more difficult for the model to converge on a solution. In the case of CycleGAN, the default cycle lambda value is set to 10.
- **Optimizer:** The optimizer is a function that determines how the model updates its internal parameters during training. Adam is a popular optimizer that is commonly used in machine learning models. In the case of CycleGAN, Adam is utilized as the default optimizer.
- **Adam beta 1 and beta 2:** Adam beta 1 and beta 2 are hyperparameters that determine the decay rates for the first and second moments of the gradients during training, respectively. These hyperparameters impact the speed and stability of the model's training process. In the case of CycleGAN, the default values for Adam beta 1 and beta 2 are set to 0.5 and 0.999, respectively.

4.3.4.4. Training algorithm

The training algorithm for CycleGAN follows the same training approach as for regular GANs (see section 2.4.1) but with the difference in the loss function and the number of neural networks. In CycleGAN, two generators and two discriminators are used. The generators are trained to translate images from one domain to another, while the discriminators are trained to distinguish between real and fake images.

During training, the generators and discriminators are updated iteratively using adversarial loss and cycle-consistency loss. Adversarial loss ensures that the generated images are indistinguishable from the real images, while cycle-consistency loss ensures that the generated images can be translated back to their original domain without losing information.

Algorithm 2 CycleGAN training algorithm

```
1: procedure CycleGAN
       // Initialize discriminators networks
2:     D_A ← discriminator_network()
3:     D_B ← discriminator_network()
       // Initialize generators networks
4:     G_A2B ← generator_network()
5:     G_B2A ← generator_network()
6:     for each epoch do
7:         for each batch of images from domain A and domain B do
               // Train discriminator A
8:             D_A_real ← D_A(real_A)
9:             D_A_fake ← D_A(G_B2A(real_B))
10:            loss_D_A ← adversarial_loss(D_A_real, target_value = 1.0)+
11:                       adversarial_loss(D_A_fake, target_value = 0.0)
12:            update_weights(loss_D_A, D_A.weights)
               // Train discriminator B
13:            D_B_real ← D_B(real_B)
14:            D_B_fake ← D_B(G_A2B(real_A))
15:            loss_D_B ← adversarial_loss(D_B_real, target_value = 1.0)+
16:                       adversarial_loss(D_B_fake, target_value = 0.0)
17:            update_weights(loss_D_B, D_B.weights)
               // Train generator A2B
18:            generated_B ← G_A2B(real_A)
19:            reconstructed_A ← G_B2A(generated_B)
20:            loss_cycle_consistency_A ← cycle_consistency_loss(real_A, reconstructed_A)
21:            D_B_fake ← D_B(generated_B)
22:            loss_G_A2B ← adversarial_loss(D_B_fake, target_value = 1.0)+
23:                         lambda_weight * loss_cycle_consistency_A
24:            update_weights(loss_G_A2B, G_A2B.weights)
               // Train generator B2A
25:            generated_A ← G_B2A(real_B)
26:            reconstructed_B ← G_A2B(generated_A)
27:            loss_cycle_consistency_B ← cycle_consistency_loss(real_B, reconstructed_B)
28:            D_A_fake ← D_A(generated_A)
29:            loss_G_B2A ← adversarial_loss(D_A_fake, target_value = 1.0)+
30:                       lambda_weight * loss_cycle_consistency_B
31:            update_weights(loss_G_B2A, G_B2A.weights)
32:        end for
33:    end for
34: end procedure
```

4.4. Data Preparation

Dataset preparation is a key aspect of our research on improving of scanned images using GANs. In this section the focus will be on discussing the different techniques used for preparing the dataset for training advanced generative models such as Pix2Pix and CycleGAN. These models rely on large amounts of suitable data for training, and the quality of the final output is directly influenced by the quality of the input data. The process of dataset preparation involves tasks such as data collection, cleaning, augmentation, and normalization, that aim to make the data suitable for training the models.

4.4.1. Raw Data Creation:

In order to train our Pix2Pix and CycleGAN models, we created a varied raw dataset of text images in a clear digital PDF format. This dataset contained a mix of automatically generated text and text from uncopyrighted books, presented in different font styles, Languages and sizes. To expand the dataset and diversify it even further, we incorporated old newspaper images sourced from Wikimedia Commons[4], which are licensed under Creative Commons CC0 and come with OCR digital copies. We also incorporated data from the publicly available Wossidlo Digital Archive[5] as a source for images of old documents. This website provides a collection of historical document images, although this posed some challenges since the data does not have clean pairs. As a result, we only utilized this data for CycleGAN training, as it does not require paired data.

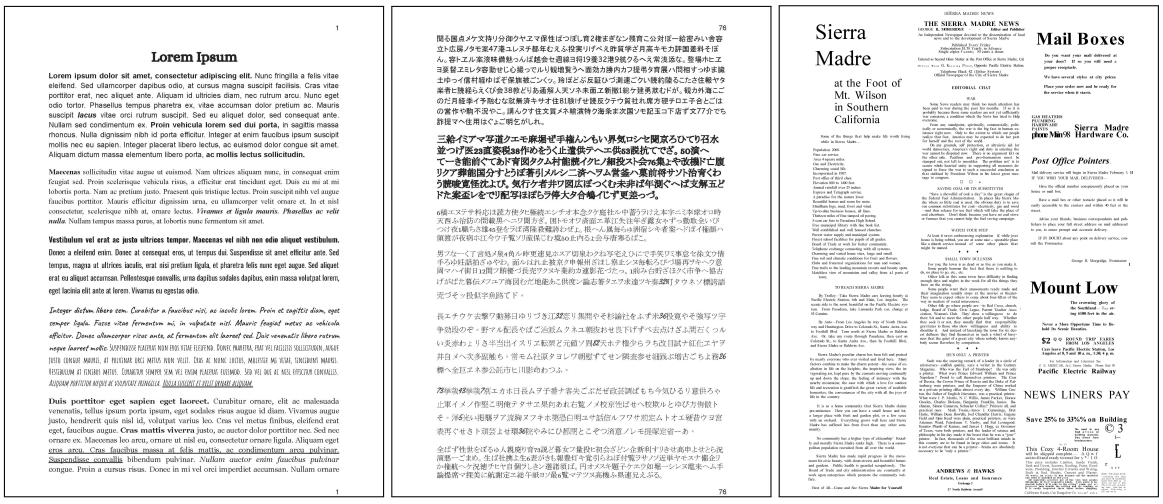


Figure 4.7.: Examples for the raw data

To preprocess the raw data and create the training dataset for the GAN models, we devised a data pipeline with several stages, each serving a distinct purpose in processing the data. Further details regarding this pipeline will be provided in the following section.

[4] https://commons.wikimedia.org/wiki/Main_Page, accessed 18. Feb. 2023

[5] https://apps.wossidia.de/webapp/run, accessed 18. Feb. 2023

4.4.2. Data Pipeline

An image processing data pipeline is essential for creating suitable training data for models that aim to enhance and clean scanned images. With this in mind, we have developed a flexible and scalable data pipeline that can be easily modified as necessary. Our default pipeline consists of several steps, each serving a distinct purpose in processing the data. To begin, we start with a clean PDF file as the raw data source, from which we extract the images. We use a configuration file to specify various parameters of the pipeline, including the desired image dimensions, as well as the types of augmentations and transformations to be applied. Once the PDF file and configuration file are in place, the data pipeline begins its processing.

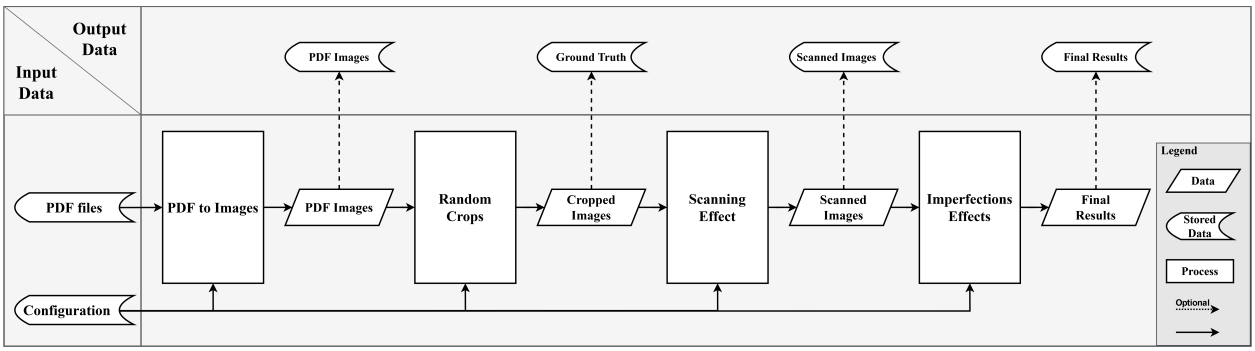


Figure 4.8.: The default data pipeline

As shown in Fig. 4.8, the default pipeline composed of four processing steps, which are outlined as follows:

The First Stage: PDF to Images

The main purpose of this stage is to convert a PDF file into high-quality images that can be utilized as the groundwork for various tasks in the data pipeline. To initiate the conversion process, the desired PDF document is selected, and the configuration file specifies the necessary parameters, such as the number of pages, image resolution, and format. *PyMuPDF* library is used to execute the conversion task, providing various functionalities that include the extraction of pages and converting them to high-resolution image formats, among other functionalities. Converting the PDF into images improves the handling and processing of data, making it more accessible for subsequent stages in the data pipeline.

The Second Stage: Random Crops

The second stage of the pipeline takes advantage of the high-quality images generated in the first stage and crops them randomly to generate a unified dataset that can be used to train the models later. The number of crops is specified in the configuration file, and these crops serve as ground truth for later stages in the data pipeline. To perform this cropping process, we use the *Albumentations* library, which provides efficient and flexible image augmentation capabilities. Along with cropping, we apply scaling (zoom) to the images, while maintaining the resolution and dimensions of the cropped images. By doing so, we obtain a variety of font sizes in each cropped image, which adds to the diversity of the dataset. To add more realism to the dataset, we apply some Elastic random distortion to the images that have been cropped. This helps

to simulate real-world conditions and ensures that the models can handle various distortions and still produce high-quality images. These cropped images act as the input for subsequent stages in the data pipeline, where they will be further processed to simulate degraded image conditions.

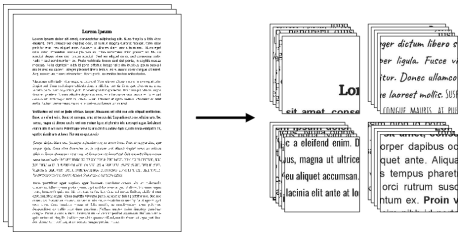


Figure 4.9.: The default data pipeline

The Third Stage Scanning Effects

The third stage of the pipeline continues the process of simulating degradation and distortion that scanned document images typically undergo, building on the previous stages. To accomplish this, the *Augraphy* library is utilized, which offers a variety of augmentations and transformations that simulate the appearance of scanned images. These effects are specified in the configuration file. The library provides a series of distortion and transformation effects to convert clean source documents into scanned-like images. Initially, the source document is separated into two layers - an ink-layer representing text and graphics, and a paper-layer that can be either white or have a randomly chosen texture. Both layers are then further processed to imitate the effects of paper printing, faxing, scanning, and copying, generating multiple variations from a single source document, resulting in numerous realistic and degraded document images. Once both the ink and paper layers have been completed, the ink is applied to the paper with the desired effects. The resulting merged document image is then subject to additional distortions, such as the addition of folds or other physical deformations that are determined by the interplay between the paper and ink layers.

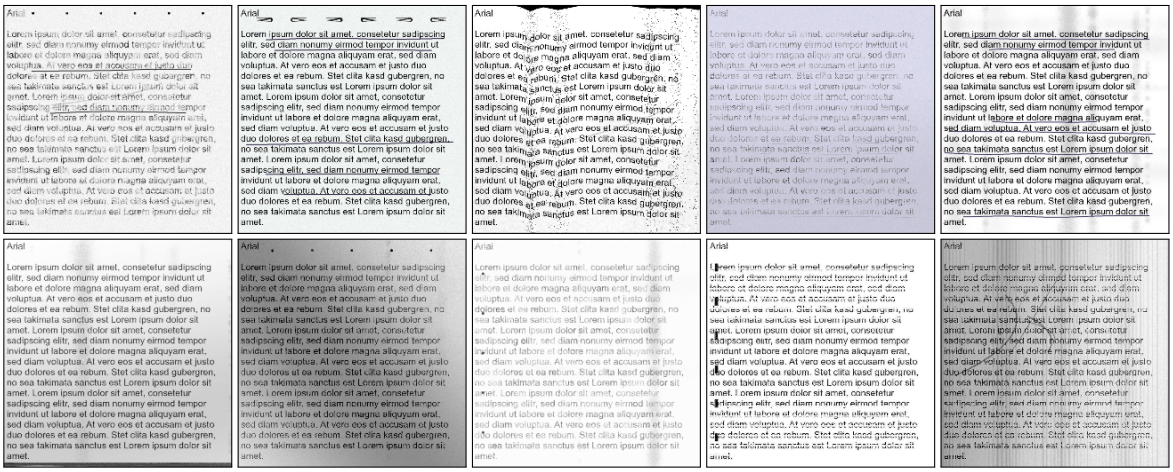


Figure 4.10.: Examples for Scanning Effects

The Fourth Stage: Imperfections Effects

In the fourth stage of the document image degradation pipeline, the aim is to introduce additional degradation and imperfections to the scan-like images generated in the previous stages. To achieve this, various degraded paper textures and coffee and ink stains are added, which are specified in the configuration file and randomly applied to each image. These textures and stains are from Creative Commons sources, generated using modern text to image artificial intelligence techniques like DALL·E 2 and stable diffusion, or created manually.

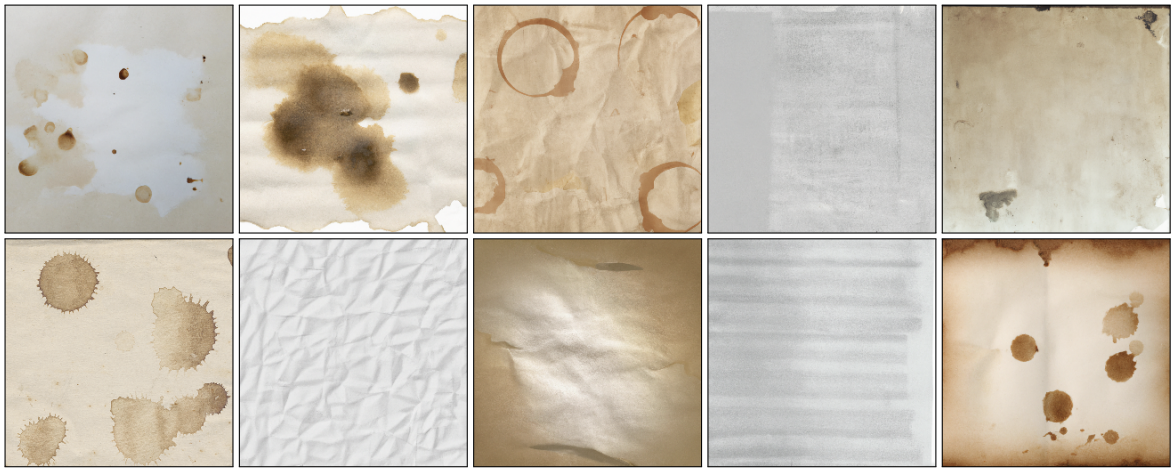


Figure 4.11.: Examples for degraded paper textures and coffee stains

To ensure that the models can better learn and generalize in real-world situations, blending techniques such as multiply blend mode and darken mode with various opacities are utilized to smoothly merge defects with the third stage layers. This stage adds a variety of textures and stains that mimic the natural aging and deterioration of documents over time, resulting in a more diverse and realistic set of degraded document images. As a result, the models can better handle a wider range of real-world scenarios, including images that have been impacted by environmental factors or natural aging. Furthermore, the blending techniques used in this stage produce a more cohesive image, where the degradation and distortion blend seamlessly with the overall picture. This enhances the models' ability to perceive and distinguish between the genuine text and the degradation present in the image.



Figure 4.12.: Examples for the default pipeline outputs

Once the data is prepared, it is split into two distinct datasets, training and testing datasets. The training dataset is used for training the models, while the test dataset is used to evaluate the performance of the models and fine-tune them as necessary, for further details about testing datasets and evaluation refer to chapter 5. Additionally, the data pipeline can produce both paired and unpaired datasets for a variety of computer vision tasks, such as supervised and unsupervised image-to-image translation (refer to section 2.5). This data can be used to train various machine learning models, such as Generative Adversarial Networks (GANs) to learn the correlation between degraded and clean images and generate high-quality images from degraded inputs.

4.4.3. Data Preprocessing

Data preprocessing is a critical step in any machine learning project, especially when dealing with image data. In this section, we will discuss the preprocessing steps employed to prepare the data for training and testing/using the Pix2Pix and CycleGAN models. The pre-processing steps for both models consist of image resizing, cropping, and normalization. These steps are necessary to ensure that the images have a uniform size, are centered around the area of interest, and are normalized to a common scale. The goal of this process is to standardize the data and eliminate any unnecessary variations that may hinder the performance of the models.

In order to prepare the data for training in Pix2Pix and CycleGAN, various preprocessing steps are employed for both image domains. The initial step involves resizing and cropping both images to a fixed size of 256×256 pixels. Afterward, the pixel values are normalized to ensures that the output is mapped to a value range of [-1, 1]. This step is critical in improving neural network convergence by giving all features an equal range of values and preventing some from dominating others due to wide value ranges. In addition to normalization, another preprocessing technique used involves applying random horizontal or vertical flipping to the images. This technique creates a new set of images that improve the model's ability to generalize to unseen data. Overall, these preprocessing steps are critical in standardizing the data and eliminating any unnecessary variations that may hinder the performance of the models.

It should be noted that pre-processing consists of two types: one for training data and one for test/user data. The standardized pre-processing process is applied to the training data, treating each sample as a patch and using it to train the models. However, it is not possible to pre-process the test/user data as one patch for the entire sample, as real samples have varying dimensions, not just 256×256. To maintain data integrity and adhere to the same principles of training data pre-processing, the test/user data is divided into patches that are proportional in size to the training data. These patches are processed separately and then fed into the model one by one. Once all patches are processed, they are combined again. This ensures that the test/user data undergoes the same pre-processing steps as the training data and that its dimensions remain unchanged.

4.5. Prototype

In this section, we will present a prototype that demonstrates the usage of the GAN models we have trained to remove imperfections and improve scanned images. The prototype offers a simple and intuitive user interface that enables users to effortlessly upload their scanned images and apply pre-trained GAN models on them. It also provides several other useful features, including the option to compare the original and enhanced images side-by-side and the ability to save the improved images. Furthermore, we will delve into other features of the prototype in more detail later in this section. The prototype was developed using *Streamlit*, a Python-based framework for building interactive web applications for machine learning and data science.

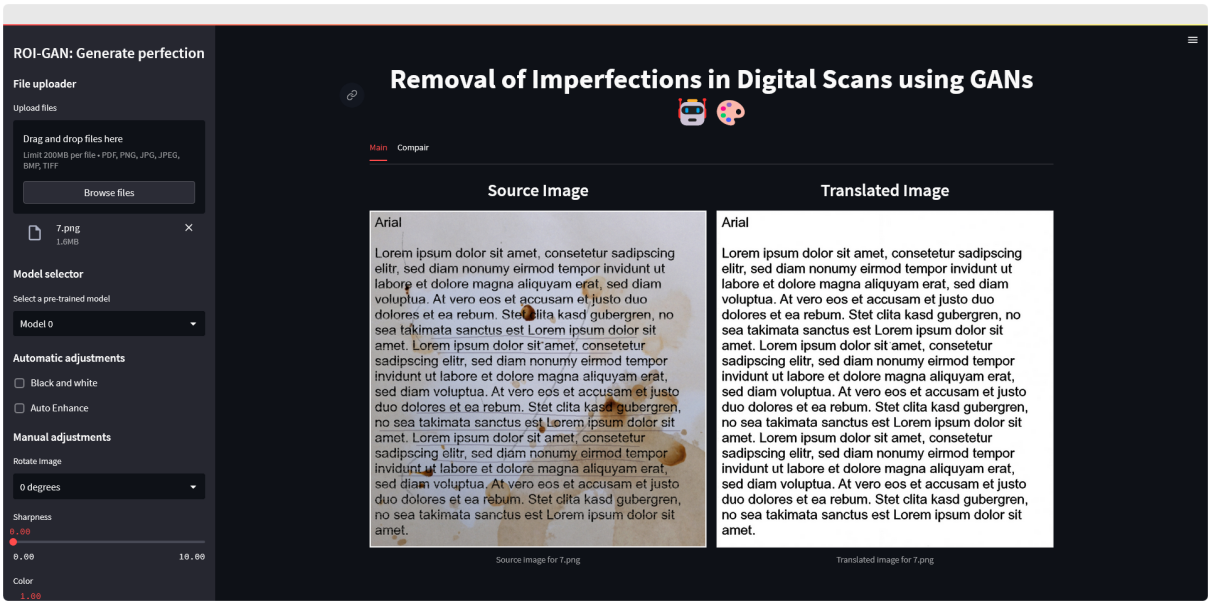


Figure 4.13.: Prototype Interface

As we can see in Fig. 4.13, The prototype interface is partitioned into two primary components to enhance the user's experience. The first component is the Sidebar that contains a range of functionalities that the user can access. On the other hand, the second component is the main viewing area, which is designed to show the input and processed images. This area is the primary focus of the prototype interface as it allows the user to view the images they are working on and monitor the progress of their work. By partitioning the prototype interface into these two primary components, the user can easily navigate and use the various features of the interface. The sidebar provides quick access to all the functionalities, while the main display area serves as the central workspace.

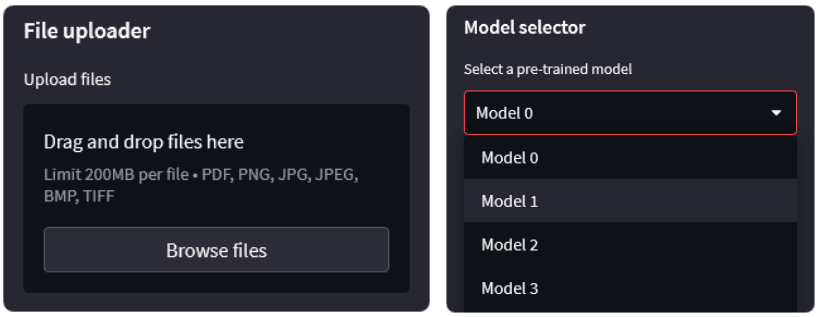
4.5.1. The Sidebar

As mentioned, the sidebar houses all available functionalities to interact with the software. These functionalities could be in the form of buttons, drop-down menus, or any other graphical representation that allows users to interact with the software. Typically, the sidebar is located on the left side of the interface, and it's designed to be easily accessible to the user.

The available functionalities are as follows:

1. **File uploader:** With this feature, users can upload their own files to the program for processing, with each file limited to a size of 200MB. The files can be selected by clicking on the "browse files" button or by dragging and dropping them to the designated location. Multiple files can be uploaded simultaneously, and the feature supports a variety of file extensions such as PDF, PNG, JPG, JPEG, BMP, and TIFF.

2. **Model selector:** This feature allows users to select a pre-trained GAN model that they wish to apply to the uploaded files from a drop-down list. To make the pre-trained models available in the list, they must be added to the "pre-trained models" folder within the project.



(a) File uploader (b) Model selector

Figure 4.14.: Prototype Sidebar File uploader and Model selector

3. **Automatic adjustments:** This feature enables the user to automatically adjust the uploaded images by applying certain modifications. These adjustments are converting the image to grayscale or applying the CLAHE algorithm, which can help to avoid over-amplification of contrast. The purpose of these automatic adjustments is to assess whether they enhance the quality of the output generated by the model. However, the effectiveness of these adjustments may vary depending on the condition of the image that was entered. In some cases, the automatic adjustments may result in improved results, while in other cases it may lead to worse results.

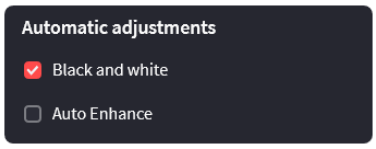


Figure 4.15.: Prototype Sidebar Automatic adjustments

4. **Manual adjustments:** By means of this feature, users can manually fine-tune the uploaded images by manipulating different variables, such as Sharpness, Color, Brightness, and Contrast. The user can adjust these variables using a slider that can be dragged left

or right until they achieve the desired result. In addition, the user can rotate the image in multiple angels (0, 90, 180, and 270 degrees) using a drop-down list. The purpose of making these modifications is to identify the optimal combination of values that can improve the quality of the outputs produced by the model.

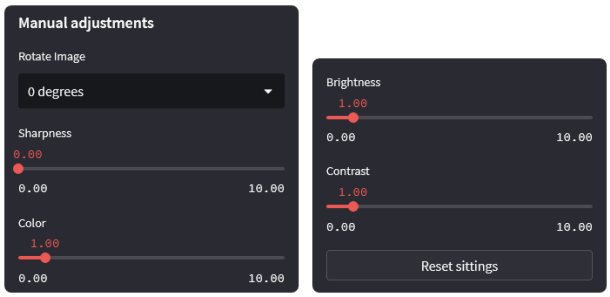
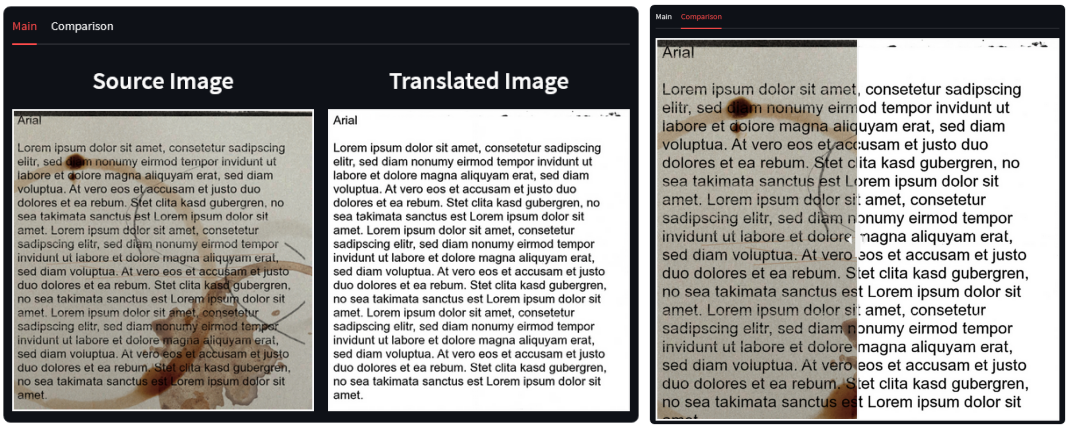


Figure 4.16.: Prototype Sidebar Manual adjustments

5. **Download:** This feature allows the user to download all the processed and enhanced images in the form of a zip file.

4.5.2. The Main Viewing Area

The primary section of the prototype's interface is the main viewing area, where the user can observe the images they have uploaded and the outcomes of any processing that has been carried out on these images. This area is the most significant element of the interface and is composed of two tabs. The first tab, titled "Main", exhibits the uploaded images on the left side and the transformed images on the right side. The second tab, called "Comparison," enables the user to scrutinize the images more closely by using a slider that can be shifted to the left to view the output, or to the right to view the input, enabling the user to compare the results effectively.



Main tab Comparison tab

Figure 4.17.: Prototype Main Viewing Area

4.6. System Architecture

In this section, we will present the architecture of the system and the workflow between its main components. The system comprises four distinct parts: Data Preparation, Training, Testing, and the Interface. Each section serves a particular purpose in accomplishing the system's ultimate objective. We will discuss how data flows within the system and the processes involved in each section.

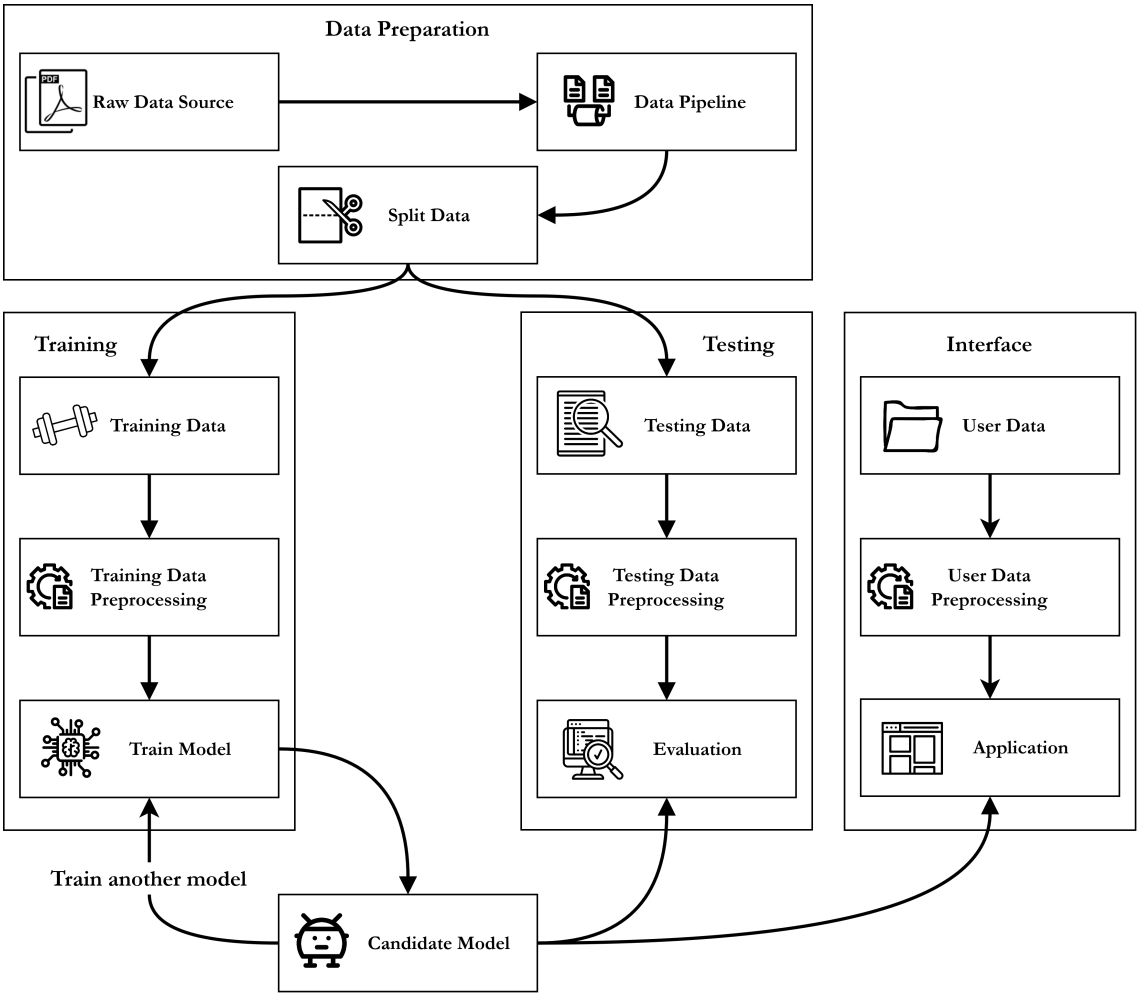


Figure 4.18.: System Architecture

As we see in Fig. 4.18 the first component of the system is the data preparation section, which begins with processing the raw data using the data pipeline to generate data in the required format for the task (see section 4.4). Once the data is processed, it is divided into two parts: the first part is used for training, while the second part is reserved for testing. The training data is then directed to the training section, where it undergoes preliminary processing to prepare it for training (see section 4.4.3). Subsequently, the model trains on this processed data. After completing the training process, the trained model flows into the testing section for evaluation, and the Interface section where users can leverage the pre-trained model to immediately use the results.

5. Evaluation setup

In this chapter, we will outline the evaluation setup for the models. This procedure aims to determine the effectiveness of these methods in improving scanned images. When evaluating the improvement of scanned images, it is important to take into consideration factors such as image resolution and the preservation of fine details in the translated images. Therefore, the evaluation will consider three key elements: testing datasets, evaluation metrics, models and training details. The testing datasets will be carefully selected to reflect a diverse range of scanning scenarios and ensure that the evaluation results are representative of real-world situations. The evaluation metrics used will quantify the effectiveness of the model in terms of visual quality, preservation of important details, and overall improvement of the scanned images. Finally, we will describe the models and training details, including the dataset used for training, the optimization algorithm, and the hyperparameters used in the experiments.

5.1. Testing Dataset

Testing datasets are essential component of evaluating the performance of our study. A paired dataset is required for the evaluation as it allows us to assess the performance of the model objectively and draw meaningful conclusions about the quality of the generated images by comparing them to real images, thus providing a basis for determining the quality of the generated images and measures the model's ability to maintain important features like color, texture, and structure.

In our evaluation process, we will generate two types of test datasets, each containing 200 paired images of various languages, fonts, and sizes, for each scenario in our evaluation process (see section 6.1). These datasets will be produced using a data pipeline and supplemented with images from the kaggle Denoising Dirty Documents dataset[1]. This approach will enable us to evaluate our model's performance in handling various types of input data, such as documents with different languages, styles, and layouts. Additionally, the use of such a diverse dataset will allow us to assess the model's generalizability and efficiency in processing different types of documents.

By evaluating the model's performance under such diverse conditions, we will gain a more comprehensive understanding of its capabilities and limitations, which will help us make informed decisions about the quality of the generated images. Overall, the use of a diverse test dataset is essential for obtaining a more accurate assessment of our model's performance and generalizability.

[1] `https://www.kaggle.com/c/denoising-dirty-documents`, accessed in 26. Feb. 2023

5.2. Quantitative Evaluation Metrics

Evaluating the performance of Generative Adversarial Networks (GANs) for image-to-image translation tasks can be challenging, as the success of these models is often measured by subjective quality of the generated images. To address this issue, several quantitative evaluation metrics have been proposed. Some of the most commonly used metrics in this context include:

- **Peak Signal-to-Noise Ratio (PSNR):**

  PSNR is a commonly used image quality metric that measures the difference between a reference image (ground truth) and the generated image. By expressing this difference in decibels (dB), PSNR aims to quantify the amount of noise and degradation in image quality as a result of lossy compression or other processes that result in a reduction of the image's structural and visual information [42]. PSNR is calculated as follows:

$$PSNR(x, y) = 20 \times \log_{10} \left(\frac{MAX_x}{\sqrt{MSE(x, y)}}\right) \tag{5.1}$$

  where:

  - MAX_x is the maximum possible pixel value that exists in our original image.
  - MSE is the Mean Squared Error between the reference image and the reconstructed image see equation 2.1.2

  It's important to note that while PSNR can be a useful metric for assessing image quality in some cases, it has several limitations, including being sensitive to only the magnitude of the error and not to its distribution. Additionally, high PSNR values do not necessarily guarantee good image quality in a perceptual sense [43]. Therefore, PSNR should be used in combination with other metrics.

- **Structural Similarity Method (SSIM):**

  SSIM is another widely used image quality metric that measures the structural similarity between two images. It provides a perception-based measure of the similarity between two images by considering the structural information, luminance information, and contrast information [44]. SSIM is calculated as follows:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{5.2}$$

  where:

  - μ_x and μ_y are the means of images x and y
  - σ_x and σ_y are the standard deviations of images x and y
  - σ_{xy} is the cross-covariance of images x and y
  - c_1 and c_2 are small constants used to stabilize the division

SSIM index ranges from -1 to 1, with a value of 1 indicating perfect structural similarity between the two images and a value of -1 indicating no structural similarity [44]. However, it's important to keep in mind that like PSNR metric, the use of SSIM as a sole metric for evaluating image quality is limited, and it's recommended to combine it with other metrics.

- **Fréchet Inception Distance (FID):**

  FID is a evaluation metric for Generative Adversarial Networks (GANs) introduced in [45]. It measures the distance between the feature representations of real and generated images in an intermediate layer of a pre-trained neural network called Inception-v3. A lower FID score indicates a higher level of similarity between the real and generated images and is therefore desirable [45, 46]. The calculation of FID involves the following steps:

  1. Extraction of features of real and generated images from an intermediate layer of the pre-trained model Inception-v3. These features are used to obtain a multivariate Gaussian distribution for each set of images.
  2. Calculation of means and covariance matrices for both the real and generated images.
  3. Calculation of Fréchet distance between the two multivariate Gaussian distributions using the means and covariance matrices. The Fréchet distance is a measure of the similarity between the two distributions and can be thought of as the maximum difference between the two distributions along all possible alignments. It can be calculated as follows:

     $$FID = ||\mu_r - \mu_g||^2 + Tr(C_r + C_g - 2(C_r * C_g)^{\frac{1}{2}}) \tag{5.3}$$

     where:

     - μ_r and μ_g are the feature-wise mean of the real images and generated images
     - C_r and C_g are the covariance matrix of the real images and generated images
     - $||\mu_r - \mu_g||^2$ sum squared difference between the two means
     - Tr is the trace of a matrix.

  FID has been shown to be a more robust evaluation metric compared to other metrics, as it considers both the distribution and quality of the generated images [47]. However, it's important to keep in mind that like any evaluation metric, FID has its limitations, and it's best to use it in combination with other metrics and visual inspection to make an overall evaluation of the quality of generated images.

As previously stated, relying on any single metric alone is inadequate for a complete evaluation of the GAN model's performance in image-to-image translation task. Therefore, a comprehensive evaluation should consider all previous metrics that can uncover issues like mode collapse or instability during training. Furthermore, in addition to the quantitative metrics, it's also important to consider the visual inspection of each of the images generated during the evaluation process. This inspection can provide valuable information about the performance of the models under various input conditions.

5.3. OCR Evaluation Metrics

Ensuring the reliability and accuracy of the resulting text is crucial for OCR systems. In our study, we aimed to assess the effectiveness of the Pix2Pix and CycleGAN models in enhancing the performance of OCR systems. To evaluate the accuracy of these systems, two commonly used metrics are character error rate (CER) and word error rate (WER).

- **Character Error Rate (CER):**

  CER measures the percentage of character-level errors in the extracted text compared to the ground truth. It is calculated as the total number of substitutions, deletions, and insertions divided by the total number of characters in the ground truth [48]. The formula for CER is:

  $$CER = \frac{(S + D + I)}{N} \tag{5.4}$$

  where:

  - S is the number of substitutions (incorrect characters substituted for correct ones)
  - D is the number of deletions (correct characters missing from extracted text)
  - I is the number of insertions (incorrect characters added to the extracted text)
  - N is the total number of characters in the original text

- **Word Error Rate (WER):**

  WER measures the percentage of word-level errors in the extracted text compared to the ground truth. It is calculated as the total number of substitutions, deletions, and insertions divided by the total number of words in the ground truth [48]. The formula for WER is:

  $$WER = \frac{(S_w + D_w + I_w)}{N_w} \tag{5.5}$$

  where:

  - S_w is the number of substitutions (incorrect words substituted for correct ones)
  - D_w is the number of deletions (correct words missing from extracted text)
  - I_w is the number of insertions (incorrect words added to the extracted text)
  - N_w is the total number of words in the original text

Both CER and WER range from 0 to 1, with a lower score indicating higher accuracy. However, it is worth noting that CER and WER may exceed 1 in cases where the extracted text contains more characters or words than the original text.

5.4. Models and Training details

As we delve deeper into our experimental setup, it's essential to consider the various factors that influenced the performance of our models. This includes the training datasets, architecture of the models, hyperparameters, and other relevant details that contributed to their performance. Understanding these specifics will give us a better understanding of how the models were trained and how they were able to achieve their results.

- **Training data:** All models were trained on two distinct datasets, each containing 10,000 images in each respective domain. The pix2pix model's dataset was paired, meaning each image had a corresponding target image. Conversely, the CycleGAN dataset was unpaired, with no such corresponding images. Prior to training, all images underwent pre-processing, which involved resizing them to 256 x 256 and normalizing their pixel values. The first dataset consisted of text-only images, with the aim of train the model to generate clean and binary outcomes, i.e. black and white images. The second dataset, on the other hand, contained color images, which were utilized to train the model to generate clean results while maintaining the color of the images.
- **Generator architectures:** Both of the models were trained using two different generator architectures in order to evaluate the impact of architecture on performance. The first architecture was the original U-Net generator, which consisted of 8 encoder blocks and 8 decoder blocks that were connected through skip connections (see section 4.2.1). The second architecture was inspired by the CycleGAN generator (see section 4.3.1) with 9 residual blocks.
- **Discriminator architecture:** Both of the models were trained using 70×70 PatchGAN (see section 4.2.2 and 4.3.2)
- **Normalization layers:** Both of the models were training with the use of two distinct normalization layers, namely batch normalization and instance normalization (see section 2.1.4). The purpose of this was to assess the effect of the normalization layers on the overall performance of the models.
- **Loss functions:** Both models were trained using three different adversarial loss functions in order to investigate the impact these functions have on the results.
- **Optimization method:** All the models were trained using the Adam optimizer (see section 2.1.3) with a learning rate of 0.0002 and beta values of (0.5, 0.999).
- **Batch size:** All the models were trained with the batch size set to 1.
- **Epochs:** The training procedure was terminated after completing 100 epochs, with periodic saving occurring every 5 epochs.

6. Experiments and Results

In this chapter, we will examine the experiments performed in this study and showcase the results achieved based on the evaluation metrics, testing datasets, and training details outlined in the previous chapter.

6.1. Experiments scenario

Our upcoming experiments will evaluate the model's performance in an essential scenario, which centers on binarization. Binarization entails the conversion of scanned, degraded images into clear black and white images that preserve all crucial details. The objective of this scenario is to assess the model's ability to clean text images effectively, which is particularly relevant for applications such as Optical Character Recognition (OCR) systems that require high-quality images to accurately extract text. The effectiveness of a binarization technique can be determined by its ability to improve text visibility in such images and eliminate any unwanted artifacts or noise.

6.2. Results

The results of this study demonstrate the differences between Pix2Pix and CycleGAN in improving scanned images under the aforementioned scenario. We evaluated the performance of both models using three metrics: PSNR, SSIM, and FID, which were introduced in section 5.2. The comparison of the two models was based on quantitative measures as well as visual inspection of the output images and OCR recognition. The aim of this study was to investigate which model performs better and to provide insights into their respective strengths and limitations. This section presents the results of our experiments, which include a detailed analysis of the performance of Pix2Pix and CycleGAN in improving scanned images under different hyperparameters.

In this report, we aim to present the outcomes of our experimentation conducted under the Binarization scenario. Our study was designed to examine the impact of various hyperparameters on the performance of each model. To ensure a fair comparison between the models, we used the same hyperparameters in each test case. The primary objective of our study was to determine the optimal combination of model and hyperparameters that would produce the best results for improving the quality of scanned images in the Binarization scenario. To achieve this, we performed six separate test cases, each of which aimed to investigate the effect of specific parameters on the models' performance. By analyzing the outcomes of these tests, we were able to provide valuable insights into the most effective approach to enhancing image quality in this specific scenario.

Test Case 1:

In this test case, we aim to compare the performance of the two models, Pix2Pix and CycleGAN, in enhancing the quality of scanned images, with a specific focus on their generator architectures, which are listed as follows:

1. A ResNet featuring 9 residual blocks, with a total of 11.378 million parameters.
2. A U-Net with 8 blocks in the encoder and 8 blocks in the decoder, comprising a total of 54.410 million parameters.

Our primary objective is to assess the models' capacity to eliminate noise, artifacts, and other imperfections, such as coffee stains, from the images. To ensure a fair comparison, we trained both models using the same set of hyperparameters, which are provided below:

General Hyperparameter	
Adversarial loss function	Min-Max GAN loss
Normalization type	Batch normalization
Discriminator architecture	70×70 PatchGAN
Optimization method	Adam optimizer with beta values of (0.5, 0.999)
Batch size	1
Learning rate	0.0002
Special Hyperparameter	
Pix2Pix L1 Lambda	100
CycleGAN cycle Lambda	10

To assess the effectiveness of the models, we tested them on the testing dataset as described in Section 5.1. This dataset includes images with varying levels of noise, degradation, and imperfections as well as images of different sizes and resolutions. To evaluate the performance of the models, we used the metrics introduced in Section 5.2.

Quantitative Comparison:

Initially, we present the results of the quantitative measures applied to assess the models' performance. Our evaluation was based on various established metrics, as introduced in Section 5.2, which are widely employed in image-to-image translation. These metrics provide an objective assessment of the capacity of the model to convert images from one domain to another and enable the evaluation of its performance in comparison to a ground truth or a set of recognized labels. By analyzing these metrics, we can gain a more comprehensive understanding of how well the model is operating and identify any areas that require improvement.

To visualize the performance of the model over time, we created line charts that displayed its performance every five epochs during the 100-epoch training period for each metric. This approach enables us to efficiently compare models and identify the ones that perform better.

Figure 6.1 provides a clear representation of the progress of the model throughout the training process, as measured by the PSNR metric (higher is better).

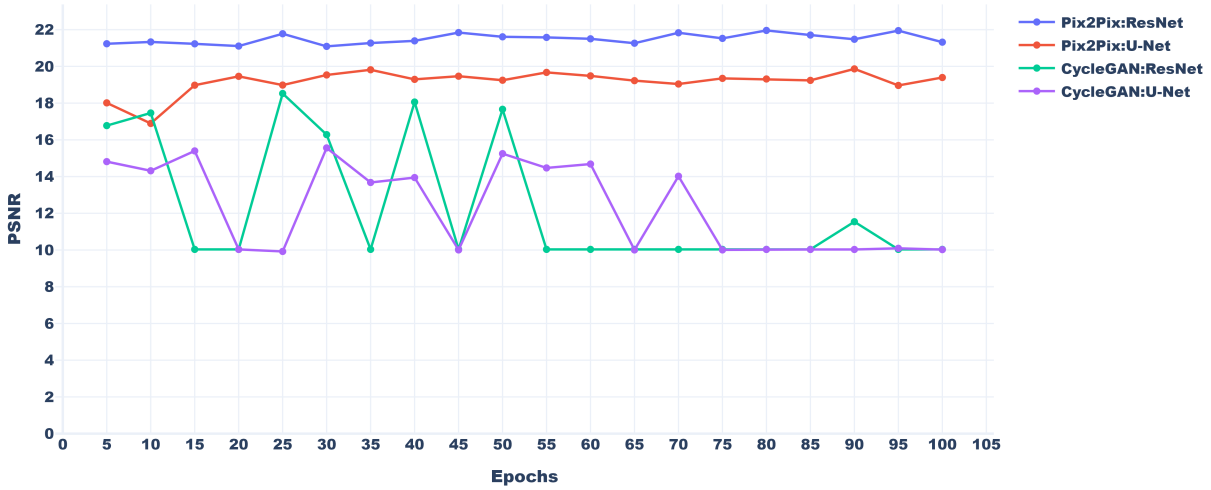


Figure 6.1.: PSNR line chart (higher is better) for test case 1

We observed that the performance of the models varied significantly across epochs. Upon initial inspection, it is evident that the Pix2Pix model outperforms the CycleGAN model in both generator architectures.

The Pix2Pix with ResNet model stands out with the highest PSNR score across all epochs, reaching a maximum score of 21.96 dB at epoch 80. On the other hand, the CycleGAN with U-Net and ResNet models consistently show the lowest PSNR scores, with some exceptions.

The results indicate that increasing the number of epochs does not always lead to a better model performance, as demonstrated by the decreasing PSNR scores in later epochs. The choice of generator architecture also appears to play a crucial role in determining the performance, with the ResNet architecture often delivering superior performance compared to the U-Net architecture. It is worth noting that the Pix2Pix models maintained a relatively stable performance rate during training, indicating that they converged quickly. In contrast, the performance of the CycleGAN model was unstable. For instance, while CycleGAN with ResNet achieved acceptable results at epochs 5, 10, 25, 30, 40, and 50, it eventually collapsed after epoch 50. Similarly, the CycleGAN with U-Net volatile results in the earlier epochs and then experiences a decline in performance after epoch 70.

In conclusion, the PSNR results suggest that the Pix2Pix with ResNet model is better suited for the task at hand than the other models. However, to determine the models that can deliver maximum performance with the given hyperparameters, it is necessary to assess the remaining metrics. While the PSNR metric only measures the amount of image noise and may not always accurately represent image quality, the SSIM metric considers luminance information, contrast information, and structural information to assess the similarity in structure between images.

Figure 6.2 provides a clear representation of the progress of the model throughout the training process, as measured by the SSIM metric (higher is better).

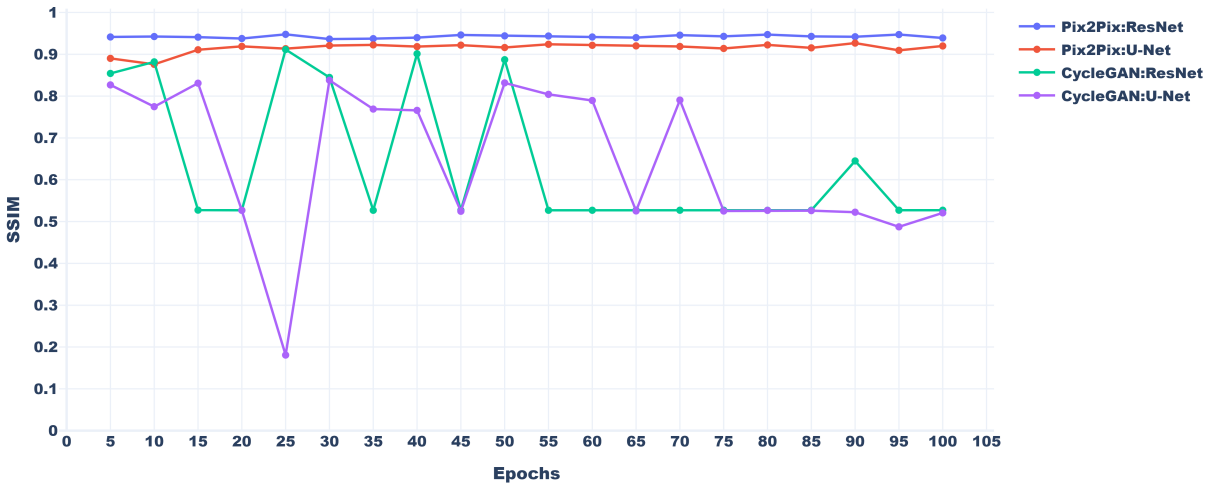


Figure 6.2.: SSIM line chart (higher is better) for test case 1

It is evident that the SSIM results closely resemble PSNR results. Model performance varies based on the type of model and the number of epochs used, and increasing the number of epochs does not always improve model performance. The Pix2Pix model consistently performs the best, achieving high SSIM values across all eras, while the CycleGAN model is generally unstable and performs poorly.

In terms of the generator architecture, the Pix2Pix model with ResNet consistently outperformed the other models, achieving SSIM values close to 0.95. The Pix2Pix model with U-Net also performs well, achieving SSIM values ranging from 0.90 to 0.92 for most ages. For CycleGAN, both U-Net and ResNet architectures show similar behavior as in PSNR, with SSIM values ranging from 0.5 to 0.9 across different epochs. It is worth noting that CycleGAN with ResNet collapsed after epoch 50 and CycleGAN with U-Net after epoch 70.

In conclusion, the results obtained from the SSIM indicate that the performance of the model is comparable to that of the PSNR results, but with higher values. This suggests that although the generated image may have a greater degree of structural similarity to the ground truth image, there may still be some noise or distortion that negatively affects its PSNR score. While PSNR and SSIM are reliable metrics for evaluating image quality on an individual basis, we chose to use a third metric, namely, the Frechet Inception Distance (FID), to assess the overall performance of our models. This is because PSNR and SSIM measure image quality individually and then calculate the mean value across the entire dataset, whereas FID considers the distribution of the generated images as a whole, as compared to the distribution of the real images, providing a more comprehensive evaluation of the models' overall performance.

Figure 6.3 provides a clear representation of the progress of the model throughout the training process, as measured by the FID metric (lower is better).

As we can see, Figure 6.3 clearly indicates that the performance of the models follows a similar pattern to the previous two metrics, except for FID, where lower values indicate better results.

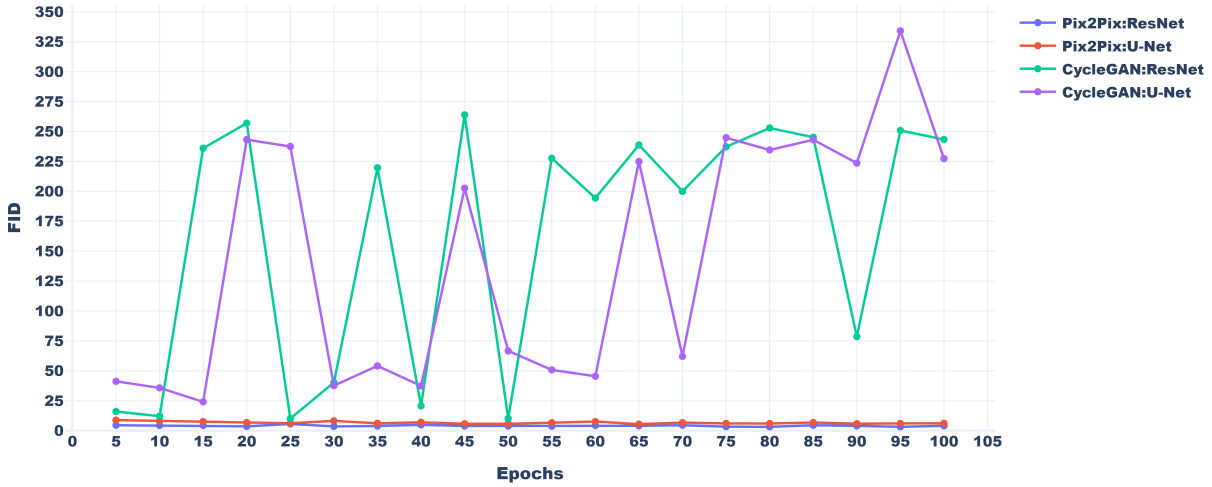


Figure 6.3.: FID line chart (lower is better) for test case 1

This particular metric is highly reliable in identifying the best models, as it presents a more distinct view of the differences in results, where the good models produce very low values, while the bad models produce very high values.

After analyzing the results of all three metrics, we can confidently conclude that the best model under these parameters is Pix2Pix with ResNet, closely followed by Pix2Pix with U-Net. However, it is noteworthy that the CycleGAN models did not exhibit stability in this test case.

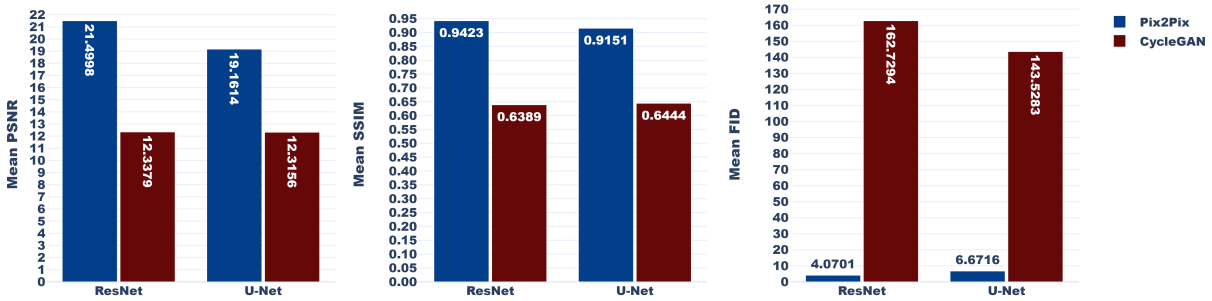


Figure 6.4.: Test case 1

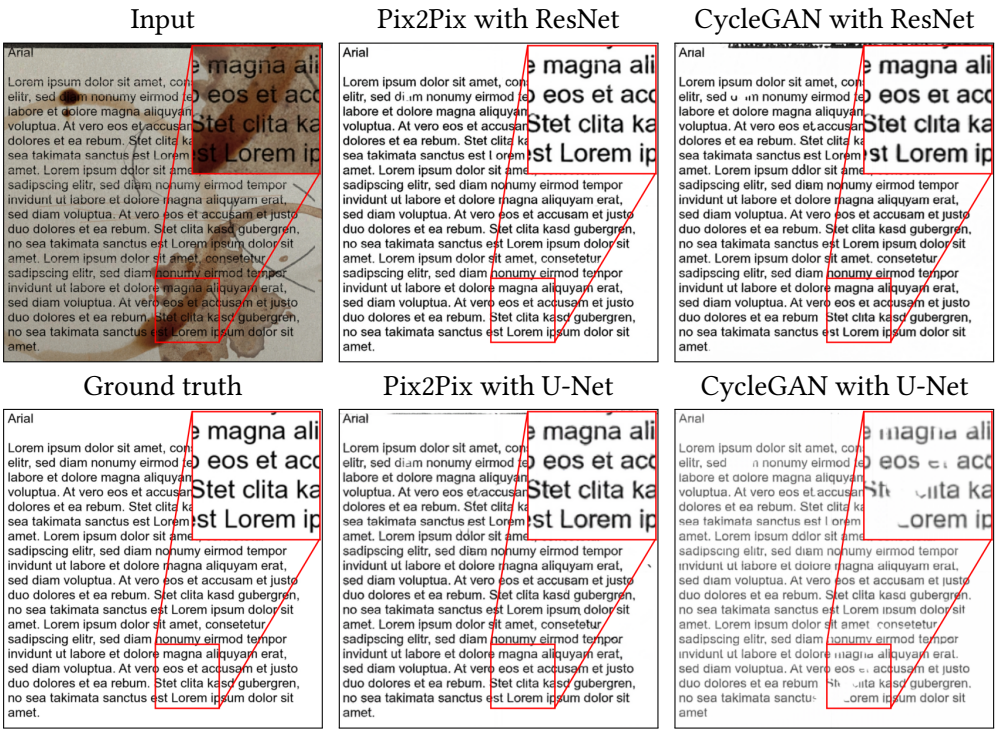
Figure 6.4 shows the mean of all epochs for each model, giving us a snapshot of the quality of the models which yielded the following results:

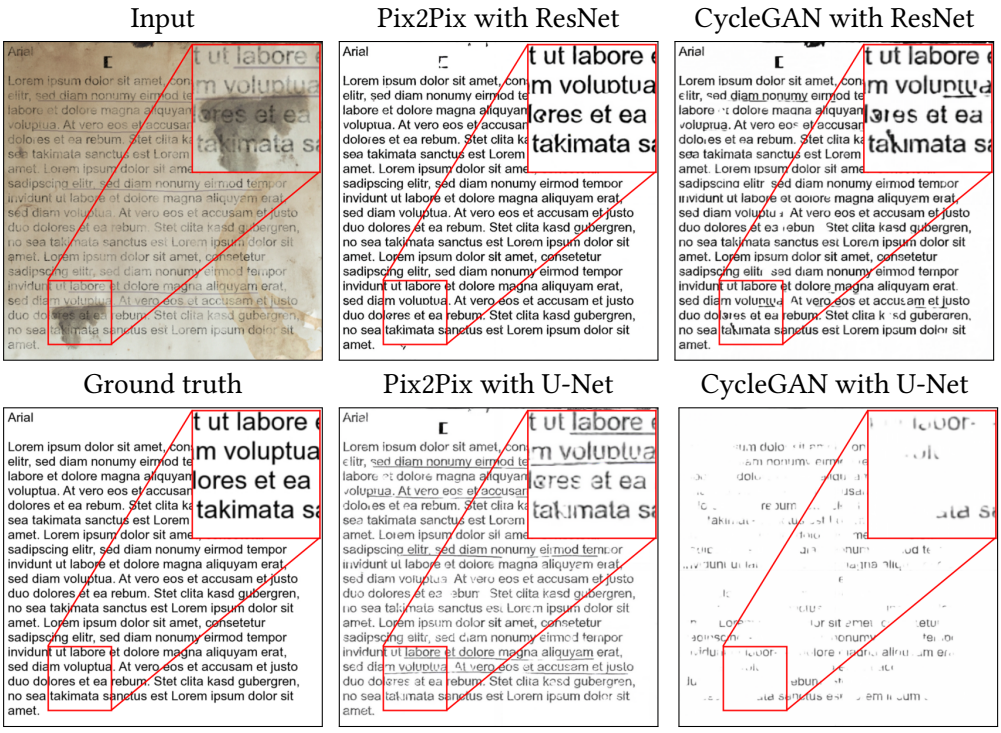
Model	Mean PSNR ↑	Mean SSIM ↑	Mean FID ↓
Pix2Pix with ResNet	**21.4998**	**0.9423**	**4.0701**
Pix2Pix with U-Net	19.1614	0.9151	6.6716
CycleGAN with ResNet	12.3379	0.6389	162.7294
CycleGAN with U-Net	12.3156	0.6444	143.5283

Table 6.1.: Test case 1 results table

Visual Inspection:

While quantitative metrics provide a useful means of evaluating the performance of models, they often fail to capture details that can only be detected through visual inspection. Therefore, after we have carried out a quantitative evaluation, we will visually inspect different samples to confirm the performance of the models. In these tests, we review only the best epochs, so the models used are as follows: Pix2Pix with ResNet at epoch 80, Pix2Pix with U-Net at epoch 90, CycleGAN with ResNet at epoch 25, and CycleGAN with U-Net at epoch 15. The following tests show the results of each model for the same inputs:







Note that the red lines are not originally present in the images and have been added to highlight specific details.

The aforementioned examples serve as a representation of the various samples that were tested. For further samples, we kindly direct you to the appendix B where additional samples are available for review.

OCR recognition:

Optical Character Recognition (OCR) is an important technology that enables computers to interpret and convert scanned images into machine-readable formats. Tesseract OCR, which is an open-source OCR engine[1] frequently used for text-recognition tasks, may encounter difficulties with low-quality scanned images. Thus, the purpose of this section is to demonstrate the performance of a models on diverse test samples, evaluating its capacity to improve Tesseract OCR's accuracy in text extraction, using two metrics: character error rate (CER) and word error rate (WER), which were introduced in Section 5.3.

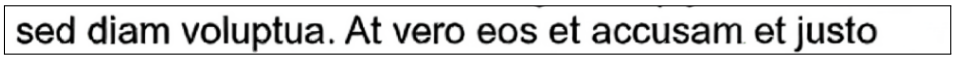
Sample: sed diam voluptua. At vero eosret accusiam et justo

- **Source image:**



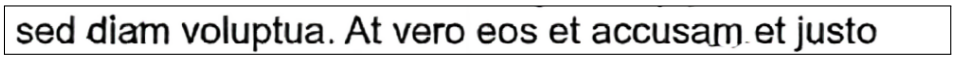
  Extracted text: . voluptua. t –
  Character Error Rate: 0.74
  Word Error Rate: 0.9

- **Pix2Pix with ResNet:**



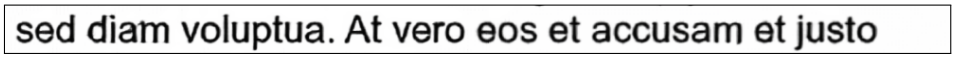
  Extracted text: sed diam voluptua. At vero eosfet accusiam et justo
  Character Error Rate: 0.04
  Word Error Rate: 0.3

- **Pix2Pix with U-Net:**



  Extracted text: sed diam voluptua. At vero eosret accusamet justo
  Character Error Rate: 0.04
  Word Error Rate: 0.4

- **CycleGAN with ResNet:**



  Extracted text: sed diam voluptua. At vero eos‘et accusam et justo
  Character Error Rate: 0.02
  Word Error Rate: 0.2

- **CycleGAN with U-Net:**



  Extracted text: sed diam voluptua. At vero ea.-.7 =. accusam ct iusto
  Character Error Rate: 0.18
  Word Error Rate: 0.4

[1] https://github.com/tesseract-ocr/tesseract

Similar to the example provided earlier, we computed the mean of Character Error Rate and Word Error Rate for 100 different sentences, and obtained the subsequent outcomes for each model:

Model	**Mean Character Error Rate ↓**	**Mean Word Error Rate ↓**
Source	0.6435	0.8125
Pix2Pix with ResNet	**0.0584**	**0.2807**
Pix2Pix with U-Net	0.0947	0.3930
CycleGAN with ResNet	0.1097	0.4045
CycleGAN with U-Net	0.2926	0.5619

Table 6.2.: Test case 1 OCR recognition results table

As shown in Table 6.2, we evaluated the performance of five different cases on 100 different sentences using mean CER and mean WER. The first row represents the performance of the source image, which is the original image without any improvement. The remaining four rows represent the performance of four different models: Pix2Pix with ResNet, Pix2Pix with U-Net, CycleGAN with ResNet, and CycleGAN with U-Net.

We can observe that the model with the highest performance is Pix2Pix with ResNet, which achieves a mean CER of 0.0584 and a WER of 0.2807. This outcome suggests that the model successfully replicated the original text with few errors. Pix2Pix with U-Net also demonstrated good performance, achieving a CER of 0.0947 and a WER of 0.3930, although it was not as effective as Pix2Pix with ResNet.

However, the CycleGAN models had lower accuracy than the Pix2Pix models. CycleGAN with ResNet had a CER of 0.1097 and WER of 0.4045, and CycleGAN with U-Net produced the highest error rates with a CER of 0.2926 and WER of 0.5619. This suggests that the CycleGAN models were less effective in enhancing the scanned images for optical character recognition (OCR), resulting in less accurate text extraction.

In conclusion, the table indicates that the Pix2Pix models with ResNet and U-Net were more effective in improving OCR accuracy than the CycleGAN models. These results suggest that the Pix2Pix models are better suited for this particular task and more appropriate for enhancing the OCR accuracy in similar situations.

Best Model:

Overall, the Pix2Pix models demonstrated remarkable proficiency in enhancing the quality of scanned images and rectifying imperfections, whereas the CycleGAN models exhibited erratic performance in numerous instances. Among all the tested models, the Pix2Pix model incorporating the ResNet architecture in the generator produced the most exceptional outcomes in all test cases (see Appendix A), with minor variances in each case. Hence, it can be deduced that the most effective model for improving scanned images and eliminating imperfections is the Pix2Pix model with the following configurations:

General Hyperparameter	
Adversarial loss function	LSGAN or Min-Max GAN loss
Normalization type	Instance normalization
Generator architecture	ResNet with 9 residual blocks
Discriminator architecture	70×70 PatchGAN
Optimization method	Adam optimizer with beta values of (0.5, 0.999)
Batch size	1
Learning rate	0.0002
Special Hyperparameter	
Pix2Pix L1 Lambda	100

For more details, see Appendix A which contains a comprehensive set of test cases that examine the impact of different loss functions and hyperparameters on the performance of the models. These test cases were conducted to provide a thorough analysis of the models and to determine which combinations of loss functions and hyperparameters would yield the best results. By examining a range of different cases, we were able to gain a better understanding of how the models would perform under various perparameters. The results of these tests are presented in detail in the appendix and provide valuable insights into the behavior of the models. Overall, the inclusion of these additional test cases enhances the robustness and comprehensiveness of our analysis.

7. Conclusion and Future work

This chapter summarizes the main findings of this study and provides suggestions for future research. The conclusion section revisits the research questions and objectives and provides a brief overview of our findings. The future work section outlines the potential research directions that could build upon the current study's findings and contribute to a deeper understanding of the topic.

7.1. Conclusion

In conclusion, this research aimed to investigate the effectiveness of using generative adversarial networks (GANs) to enhance the quality of scanned documents that have been damaged or contain imperfections, while preserving their content.

The research addressed two central questions:

1. What is the potential effectiveness of using Generative Adversarial Networks (GANs) to enhance the quality of scanned documents that have been damaged or contain imperfections while preserving their content?

2. Can this methodology be considered a practical preprocessing technique for preparing documents for digital archiving?

To address these questions, this study sought to explore two types of GANs, Pix2Pix and CycleGAN, which are employed to transform images across two domains. The decision to employ both models was motivated by the recognition that the problem may manifest in two different ways. Pix2Pix is a supervised image-to-image translation approach that requires paired data, whereby an input image from one domain is mapped to an output image in another domain, with labeled and aligned image pairs already available. On the other hand, CycleGAN is an unsupervised image-to-image translation method that does not require paired data. In this case, the model learns to map an image from one domain to another by identifying the fundamental relationship between the two domains. Given the crucial role that data plays in the training of GANs, a data pipeline has been developed to generate data suitable for both supervised and unsupervised image-to-image translation models. A data pipeline has been designed to preprocess clean PDFs and produce data of appropriate sizes for different domains. To ensure that the generated data accurately reflects real-world scenarios, various techniques were employed to simulate different types of image damage and imperfections. These techniques emulate the damage that typically occurs in real life scanned images, including scratches, stains, creases, and other deformities. The resulting data can then be utilized to train the GAN models to effectively repair the damaged images.

The methods were evaluated for their effectiveness using multiple quantitative measures, including PSNR, SSIM, and FID. Moreover, the influence of these methods on enhancing OCR performance was assessed using Character Error Rate (CER) and Word Error Rate (WER) metrics.

The research findings indicate that GANs can significantly enhance scanned images and restore them to their original state without losing details. However, there is a noticeable difference in performance between the two models, with the Pix2Pix model surpassing the CycleGAN model in all test cases. This is due to the Pix2Pix model's use of paired data, which allows for more precise mapping between input and output images, while CycleGAN uses unpaired data that complicates the task and increases instability during the training period.

Moreover, the research demonstrates that using different generator networks can yield different results. In this study, the use of ResNet outperformed U-Net. The study also highlights the impact of hyperparameters, such as loss functions and learning rates, on the model's efficiency. The study also highlights the impact of hyperparameters, such as loss functions and learning rates, on the model's efficiency. Our findings indicate that LSGAN provides the highest level of stability overall, while WGAN has the poorest performance. The results also show that each model behaves differently concerning learning rates. For instance, decreasing the learning rate resulted in a minor decrease in Pix2Pix's performance but contributed to enhancing CycleGAN's stability to some degree. However, further research is required to explore other models and ways to enhance the ability of unsupervised image translation models.

In summary, this research confirms that GANs are a powerful method for improving scanned images while retaining their content. The Pix2Pix model is more effective than the CycleGAN model, and using ResNet as the generator network is preferable to using Unet. Increasing the number of training images can also enhance the models' capabilities. This research provides valuable insights into GANs' potential for digital archiving and restoring scanned documents.

7.2. Future work

The results obtained in this thesis demonstrate the potential of Generative Adversarial Networks (GANs) in improving damaged scanned images and removing imperfections. However, there are several areas for future research that could further enhance the performance of the proposed method.

One potential avenue for future work is to investigate the use of attention mechanisms in the GANs. Attention mechanisms have been shown to improve the performance of deep learning models in a variety of tasks by allowing them to focus on specific areas of the input. By incorporating attention mechanisms into the GANs used in this thesis, it may be possible to further improve the quality of the generated images and reduce artifacts.

Another potential direction for future work is to explore the use of diffusion models for image restoration. Diffusion models are a class of generative models that have recently shown promise in image generation tasks. By learning a diffusion process that gradually removes

noise from the input image, diffusion models can produce high-quality images with fewer artifacts compared to traditional GAN-based methods.

Finally, it could be beneficial to investigate the performance of the methods on different types of damaged scanned images. In this thesis, the focus was on removing noise and stains. However, there are other types of noise and imperfections that can occur in scanned images, such as blur, skew, or color distortion. By testing the methods on a wider range of damaged images, it would be possible to evaluate its effectiveness in different scenarios and identify potential limitations.

In conclusion, there are several avenues for future research that could further enhance the performance of the proposed methods. By incorporating attention mechanisms, exploring diffusion models, and testing the methods on different types of damaged images, it may be possible to develop a more robust and effective image restoration technique for damaged scanned images.

References

[1] Why you should digitalize your business and how to succeed. Pagero, https://www.pagero.com/blog/digitalisation-why-digitalise-your-business/, Dec. 2019. [Online; accessed 1.11.2022].

[2] Mohamed Ali Souibgui and Yousri Kessentini. DE-GAN: A conditional generative adversarial network for document enhancement. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(3):1180–1191, Mar. 2022.

[3] Aurelien Geron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* O'Reilly Media, Inc., 2nd edition, 2019.

[4] I. Vasilev, D. Slater, G. Spacagna, and P. Roelants. *Python Deep Learning - Second Edition: Exploring Deep Learning Techniques and Neural Network Architectures with PyTorch, Keras, and TensorFlow, 2nd Edition.* Packt Publishing, 2019.

[5] Rich Stureborg. Artificial Neural Networks for Total Beginners. Medium, https://towardsdatascience.com/artificial-neural-networks-for-total-beginners-d8cd07abaae4, May 2020. [Online; accessed 12.11.2022].

[6] SAGAR SHARMA. Activation Functions in Neural Networks. Medium, https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6, July 2021. [Online; accessed 12.11.2022].

[7] Sanket Doshi. Various Optimization Algorithms For Training Neural Network. Medium, https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6, August 2020. [Online; accessed 12.11.2022].

[8] Vitaly Bushaev. Stochastic Gradient Descent with momentum - Towards Data Science. Medium, https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d, June 2018. [Online; accessed 20. Feb. 2023].

[9] Roan Gylberth. An Introduction to AdaGrad - Konvergen.AI - Medium. Medium, https://medium.com/konvergen/an-introduction-to-adagrad-f130ae871827, June 2018. [Online; accessed 20. Feb. 2023].

[10] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations*, 2014.

[11] Vitaly Bushaev. Adam — latest trends in deep learning optimization. Medium, https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c, October 2018. [Online; accessed 20. Feb. 2023].

[12] Zhe Ming Chng. Using Normalization Layers to Improve Deep Learning Models - MachineLearningMastery.com. MachineLearningMastery, https://machinelearningmastery.com/using-normalization-layers-to-improve-deep-learning-models, June 2022. [Online; accessed 16. Feb. 2023].

[13] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015.

[14] Deep Learning normalization methods. Tung M Phung's Blog, https://tungmphung.com/deep-learning-normalization-methods, January 2021. [Online; accessed 16. Feb. 2023].

[15] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Instance Normalization: The Missing Ingredient for Fast Stylization. *CoRR*, abs/1607.08022, 2016.

[16] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. *CoRR*, abs/1607.06450, 2016.

[17] Yuxin Wu and Kaiming He. Group Normalization. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XIII*, volume 11217 of *Lecture Notes in Computer Science*, pages 3–19. Springer, 2018.

[18] What are Convolutional Neural Networks? IBM, https://www.ibm.com/cloud/learn/convolutional-neural-networks, January 2021. [Online; accessed 12.11.2022].

[19] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *ICLR*, 2015.

[20] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network, 2016.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *CVPR*, pages 770–778. IEEE Computer Society, 2016.

[22] PyTorch ResNet: The Basics and a Quick Tutorial. https://www.run.ai/guides/deep-learning-for-computer-vision/pytorch-resnet. [Online; accessed 12.11.2022].

[23] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning Internal Representations by Error Propagation*, page 318–362. MIT Press, Cambridge, MA, USA, 1986.

[24] Arden Dertat. Applied Deep Learning - Part 3: Autoencoders. Medium, https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798, October 2017. [Online; accessed 13.11.2022].

[25] O. Ronneberger, P.Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351 of *LNCS*, pages 234–241. Springer, 2015. (available on arXiv:1505.04597 [cs.CV]).

[26] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[27] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative Adversarial Networks. *CoRR*, abs/1406.2661, 2014.

[28] Jason Brownlee. A Gentle Introduction to Generative Adversarial Network Loss Functions. MachineLearningMastery, `https://machinelearningmastery.com/generative-adversarial-network-loss-functions`, September 2019. [Online; accessed 20. Nov. 2022].

[29] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, Zhen Wang, and Stephen Paul Smolley. Least Squares Generative Adversarial Networks. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2813–2821, 2016.

[30] Jason Brownlee. How to Develop a Least Squares Generative Adversarial Network (LSGAN) in Keras - MachineLearningMastery.com. MachineLearningMastery, `https://machinelearningmastery.com/least-squares-generative-adversarial-network`, January 2021. [Online; accessed 18. Feb. 2023].

[31] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein Generative Adversarial Networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 214–223. JMLR.org, 2017.

[32] Loss Functions | Machine Learning. Google Developers, `https://developers.google.com/machine-learning/gan/loss`. [Online; accessed 20. Nov. 2022].

[33] Mehdi Mirza and Simon Osindero. Conditional Generative Adversarial Nets. *CoRR*, abs/1411.1784, 2014.

[34] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-Image Translation with Conditional Adversarial Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5967–5976, 2017.

[35] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[36] Jun-Yan Zhu, Richard Zhang, Deepak Pathak, Trevor Darrell, Alexei A. Efros, Oliver Wang, and Eli Shechtman. Toward Multimodal Image-to-Image Translation. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, page 465–476. Curran Associates Inc., 2017.

[37] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2242–2251. IEEE Computer Society, 2017.

[38] Taeksoo Kim, Moonsu Cha, Hyunsoo Kim, Jung Kwon Lee, and Jiwon Kim. Learning to Discover Cross-Domain Relations with Generative Adversarial Networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1857–1865. PMLR, 06–11 Aug 2017.

[39] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. Unsupervised Image-to-Image Translation Networks. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[40] Taesung Park, Alexei A. Efros, Richard Zhang, and Jun-Yan Zhu. Contrastive Learning for Conditional Image Synthesis. In *ECCV*, 2020.

[41] Greg Walters and John Hany. *Hands-On Generative Adversarial Networks with PyTorch 1.x*. Packt, 2019.

[42] Peak Signal-to-Noise Ratio as an Image Quality Metric. `https://www.ni.com/de-de/innovations/white-papers/11/peak-signal-to-noise-ratio-as-an-image-quality-metric.html`, August 2011. [Online; accessed 4. Feb. 2022].

[43] Fernando A. Fardo, Victor H. Conforto, Francisco C. de Oliveira, and Paulo S. Rodrigues. A Formal Evaluation of PSNR as Quality Measurement Parameter for Image Segmentation Algorithms, 2016.

[44] Pranjal Datta. All about Structural Similarity Index (SSIM): Theory + Code in PyTorch. Medium, `https://medium.com/srm-mic/all-about-structural-similarity-index-ssim-theory-code-in-pytorch-6551b455541e`, December 2021. [Online; accessed 4. Feb. 2023].

[45] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, page 6629–6640. Curran Associates Inc., 2017.

[46] Ayush Thakur. How to Evaluate GANs using Frechet Inception Distance (FID). `https://wandb.ai/ayush-thakur/gan-evaluation/reports/How-to-Evaluate-GANs-using-Frechet-Inception-Distance-FID---Vmlldzo0MTAxOTI`, February 2023. [Online; accessed 9. Feb. 2023].

[47] Yaniv Benny, Tomer Galanti, Sagie Benaim, and Lior Wolf. Evaluation Metrics for Conditional Image Generation. *Int. J. Comput. Vision*, 129(5):1712–1731, May 2021.

[48] Kenneth Leung. Evaluate OCR Output Quality with Character Error Rate (CER) and Word Error Rate (WER). Medium, https://towardsdatascience.com/evaluating-ocr-output-quality-with-character-error-rate-cer-and-word-error-rate-wer-853175297510, January 2022. [Online; accessed 5. Mar. 2023].

A. Test Cases Appendix

This appendix contains additional test cases that were not included in the main body of the document. These test cases were conducted to further validate the effectiveness of the methodologies used in this study. The additional test cases cover a variety of hyperparameters, providing a more comprehensive view of the performance of the methods under different hyperparameters.

Test Case 2:

In this test case, we aimed to revisit the performance comparison between the Pix2Pix and CycleGAN models to enhance the quality of the scanned images. However, in contrast to the first test case, we used an instance normalization layer instead of batch normalization in the generator architecture of both models. The focus of this test case is to assess the models' ability to eliminate noise, artifacts, and imperfections, such as coffee stains from the images. To ensure a fair comparison, we kept the same set of hyperparameters as in the previous test case, with the exception of the normalization layer, which yielded the following set of hyperparameters:

General Hyperparameter	
Adversarial loss function	Min-Max GAN loss
Normalization type	Instance normalization
Discriminator architecture	70×70 PatchGAN
Optimization method	Adam optimizer with beta values of (0.5, 0.999)
Batch size	1
Learning rate	0.0002
Special Hyperparameter	
Pix2Pix L1 Lambda	100
CycleGAN cycle Lambda	10

Quantitative Comparison:

Building on our previous approach, we will continue to employ quantitative comparison as a means to evaluate the performance of the models in our second test case. Similar to the first test case, we used various established metrics (see Section 5.2). By using these metrics, we can objectively assess the model's ability to convert images from one domain to another and compare its performance with that of the ground truth. Additionally, we created line charts that displayed the performance of the model every five epochs during the 100-epoch training period for each metric. This visualization technique allows us to easily compare the models and identify those that perform better over time. By utilizing these methods, we can gain a comprehensive understanding of how well the model is operating and determine areas that may require improvement.

Figure 6.1 presents a straightforward representation of how the model progressed during the training phase, as measured using the PSNR metric. A higher value on the chart indicates that the model performed better.

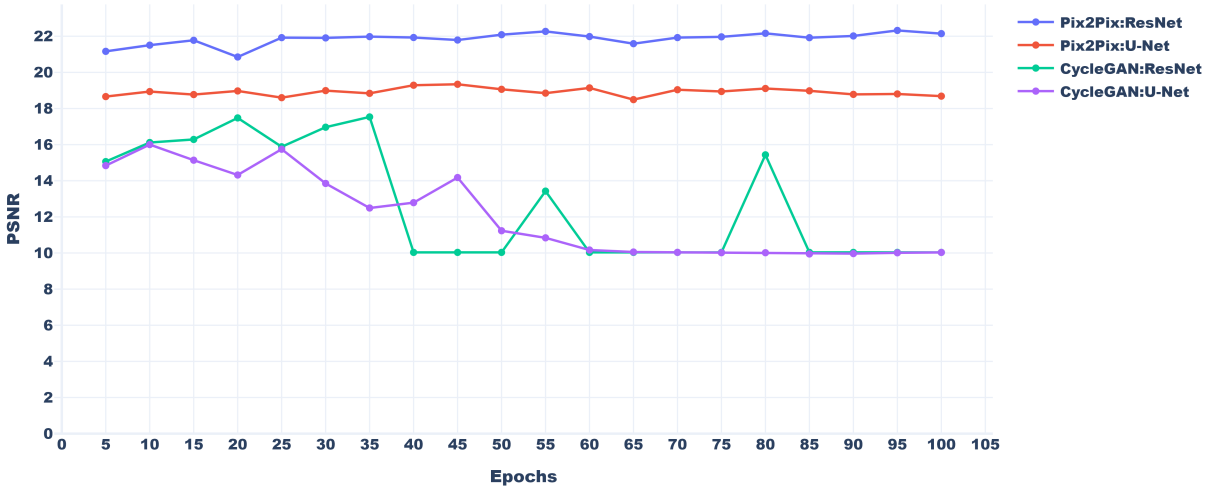


Figure A.1.: PSNR line chart (higher is better) for test case 2

We can observe that the PSNR performance of the models in the second test case is highly comparable to that of the first test case, particularly in terms of Pix2Pix's performance stability. The Pix2Pix model with ResNet architecture continues to outshine the other models, achieving the highest PSNR score of 22.32 dB in epoch 95 and 22.2687 dB in epoch 55. However, the CycleGAN models with U-Net and ResNet architectures exhibited only slight performance stabilization in the earlier epochs before collapsing again. In particular, CycleGAN with ResNet architecture maintains acceptable results between 15 dB to 17.5 dB until epoch 35, then experiences a sharp decline, while CycleGAN with U-Net architecture gradually loses performance after epoch 25, with its highest value being around 16 dB at epoch 10.

As in the first test case, it is worth noting that increasing the number of epochs does not always translate into better performance in most cases. The choice of generator architecture also remains an important factor in determining the model performance. The Pix2Pix models continue to converge rapidly and maintain a relatively stable performance rate during training, while the CycleGAN models still exhibit unstable performance.

Overall, based on the PSNR results in this test case, it can be concluded that the Pix2Pix model with the ResNet architecture remains the most suitable option for the given task. Moving forward, we will examine the results of these models using the SSIM scale, which considers factors such as luminance, contrast, and structural information.

The subsequent Figure A.2 displays the progress of the models throughout the training process, as assessed using the SSIM metric. A greater value depicted on the chart indicates that the model achieves better performance.

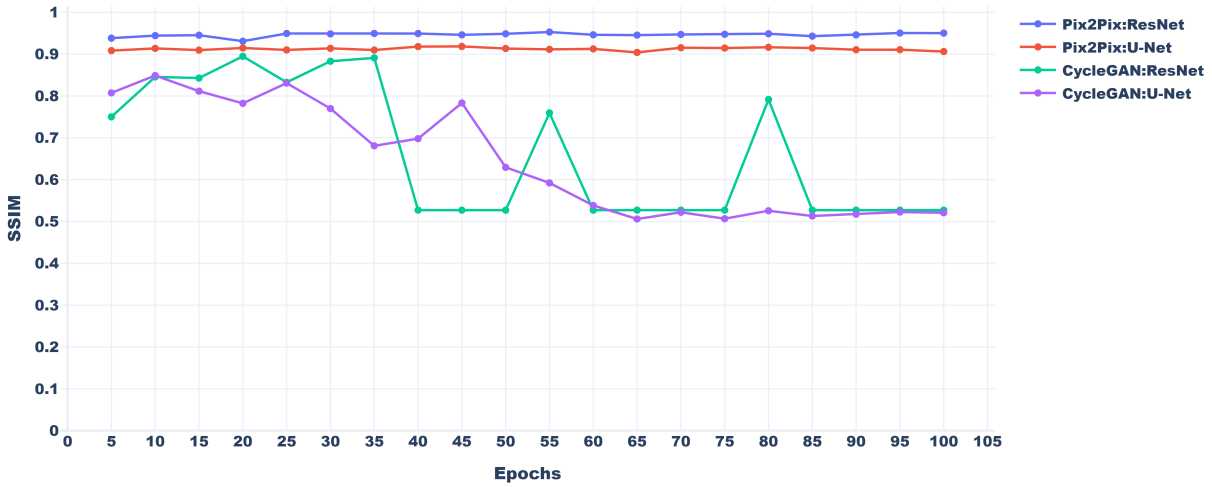


Figure A.2.: SSIM line chart (higher is better) for test case 2

As we can see, Figure A.2 shows that the behavior of the models is generally similar to the previous PSNR metric A.1, which confirms the quality of the models in terms of luminance, contrast, and structural information. The highest-performing model was Pix2Pix with ResNet after 55 epochs, which achieved an SSIM value of 0.9528. The second-best performing model is Pix2Pix with ResNet after 95 epochs, which achieved an SSIM value of 0.9504. Notably, the best model in terms of SSIM was the second-best performer in the PSNR test, whereas the second-best performer in SSIM was the top-performing model in the PSNR test.

The CycleGAN models tend to be generally unstable, as in the PSNR test. However, there is a momentary period of stability with the resnet structure, which persists until approximately the 35th epoch. Nonetheless, this stability is not sustained and the model eventually collapses, as shown in the chart. On the other hand, the Unet model displays a certain level of stability up until roughly the 25th epoch, after which it gradually begins to decline.

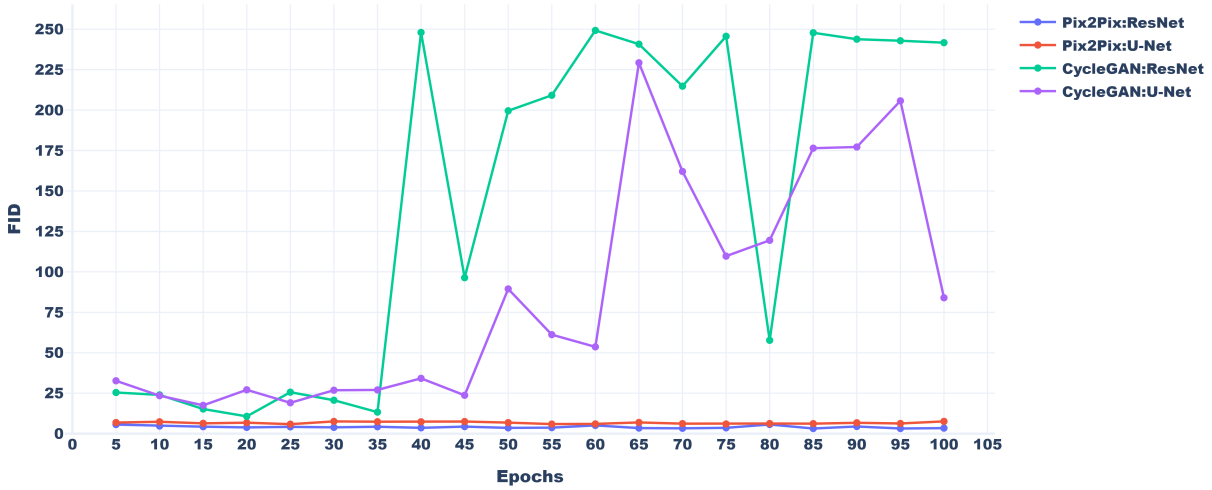


Figure A.3.: FID line chart (lower is better) for test case 2

Overall, the Pix2Pix models exhibited a more stable performance than the CycleGAN models, even in the second test case. The collapse of the CycleGAN models is clearly evident in the Figure A.3, which further supports this conclusion.

After examining the three metrics, we can state that the Pix2Pix model with ResNet performed the best under the given hyperparameters, followed by Pix2Pix with U-Net. It's noteworthy that the CycleGAN models did not demonstrate stability even in this test scenario. To compare both test cases, we prepared a bar chart that displays the mean value for each metric and facilitates comparison with the first test case.

This analysis aims to determine whether the adoption of instance normalization in place of batch normalization led to an enhancement in the models' quality.

As we can see in the Figure A.4, it can be observed that using Instance normalization instead

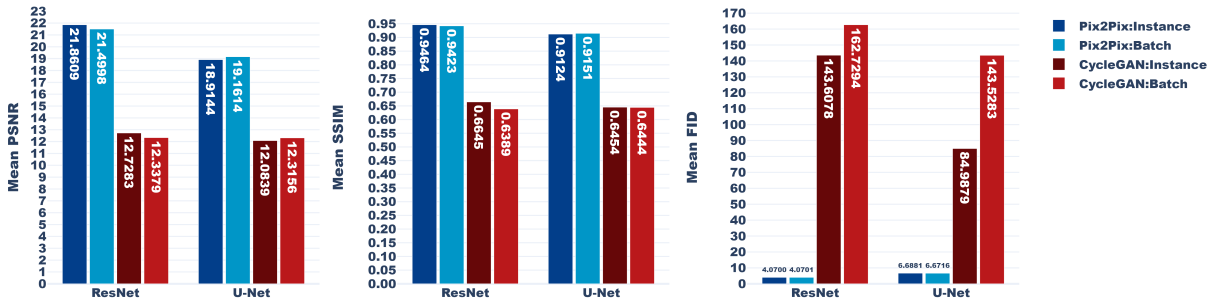


Figure A.4.: Test case 2

of batch normalization results in a slight performance improvement for the ResNet architecture in both models. However, when it comes to the U-Net architecture, the situation is different, and there is a decline in performance, except for the FID metric, which shows an improvement in the results.

The following table shows the results of the comparison more precisely, considering the architecture of the generator and the type of normalization:

Model	**Mean PSNR ↑**	**Mean SSIM ↑**	**Mean FID ↓**
Pix2Pix with ResNet and Instance norm	**21.8609**	**0.9464**	**4.0700**
Pix2Pix with ResNet and Batch norm	21.4998	0.9423	4.0701
Pix2Pix with U-Net and Instance norm	18.9144	0.9124	6.6881
Pix2Pix with U-Net and Batch norm	19.1614	0.9151	6.6716
CycleGAN with ResNet and Instance norm	12.7282	0.6645	143.6078
CycleGAN with ResNet and Batch norm	12.3379	0.6389	162.7294
CycleGAN with U-Net and Instance norm	12.0839	0.6454	84.9879
CycleGAN with U-Net and Batch norm	12.3156	0.6444	143.5283

Table A.1.: Test case 2 results table

Visual Inspection:

Following the analysis of the quantitative measurements, we will proceed to visually inspected various samples to validate the effectiveness of the models. Specifically, we examine only the good performing epochs, which include Pix2Pix with ResNet at epoch 95, Pix2Pix with U-Net at epoch 95, CycleGAN with ResNet at epoch 20, and CycleGAN with U-Net at epoch 10. The following tests show the outcomes of each example for a given input:





Note that the red lines are not originally present in the images and have been added to highlight specific details.

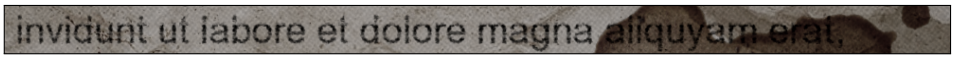
The aforementioned examples serve as a representation of the various samples that were tested. For further samples, we kindly direct you to the appendix B where additional samples are available for review.

OCR recognition:

This section aims to evaluate the effectiveness of a model in improving the accuracy of Tesseract OCR. The model's performance will be assessed on different test samples using two metrics, namely character error rate (CER) and word error rate (WER). Tesseract OCR often encounters challenges when processing low-quality scanned images, thus highlighting the need to determine the models potential to improve Tesseract OCR's accuracy in text extraction. The goal is to demonstrate the models performance on various test samples to determine its ability to enhance the accuracy of Tesseract OCR using character error rate (CER) and word error rate (WER), which were introduced in Section 5.3.

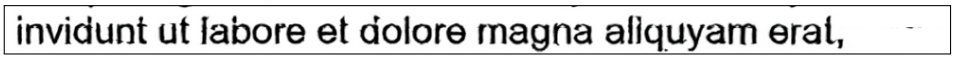
Sample: invidunt ui Iabore et dolore magha aliquyam erat,

- **Source image:**



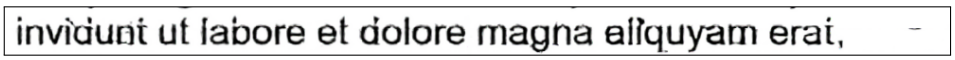
Extracted text: invf(:;!_;;4r»3i_'ui Sabore ei doiore magma
Character Error Rate: 0.6939
Word Error Rate: 1.0

- **Pix2Pix with ResNet:**



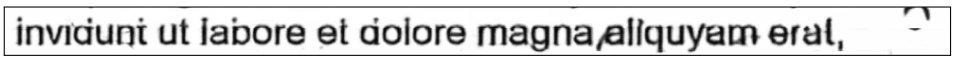
Extracted text: invidunt ui Iabore et dolore magné aliquyam eiat,
Character Error Rate: 0.0816
Word Error Rate: 0.5

- **Pix2Pix with U-Net:**



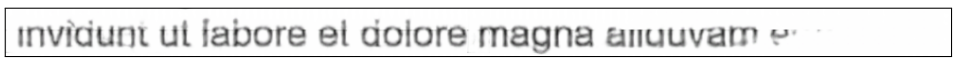
Extracted text: invrdunt ui labore el dolore magni-a aliquyam ekai,
Character Error Rate: 0.1429
Word Error Rate: 0.6250

- **CycleGAN with ResNet:**



Extracted text: invidunt ui labore el dolore magnépllquyam eial,
Character Error Rate: 0.1633
Word Error Rate: 0.6250

- **CycleGAN with U-Net:**



Extracted text: Invuuunt ul labore el dolore magna -muuvaw -»
Character Error Rate: 0.3265
Word Error Rate: 0.6250

Similar to the example provided earlier, we computed the mean of Character Error Rate and Word Error Rate for 100 different sentences, and obtained the subsequent outcomes for each model:

Model	Mean Character Error Rate ↓	Mean Word Error Rate ↓
Source	0.6435	0.8125
Pix2Pix with ResNet	**0.0517**	**0.2572**
Pix2Pix with U-Net	0.0900	0.3883
CycleGAN with ResNet	0.2312	0.5473
CycleGAN with U-Net	0.2501	0.5906

Table A.2.: Test case 2 OCR recognition results table

Table A.2 presents the evaluation of five different cases on 100 sentences, using CER and WER. The first row of the table represents the performance of the source, which is the original image with no enhancements. The remaining four rows correspond to the performance after using the four models: Pix2Pix with ResNet, Pix2Pix with U-Net, CycleGAN with ResNet, and CycleGAN with U-Net.

Upon examination of the table, it is evident that the Pix2Pix model with ResNet achieved the highest performance in improving OCR accuracy, with a mean CER of 0.0517 and a mean WER of 0.2572. These results suggest that the model was successful in replicating the original text with very few errors. The Pix2Pix model with U-Net also demonstrated good performance, with a CER of 0.0900 and a WER of 0.3883, although it was not as effective as the Pix2Pix model with ResNet.

In contrast, the CycleGAN models had lower accuracy compared to the Pix2Pix models, with CycleGAN with ResNet having a CER of 0.2312 and a WER of 0.5473, and CycleGAN with U-Net producing the highest error rates with a CER of 0.2501 and a WER of 0.5906. These results suggest that the CycleGAN models were less effective in enhancing the scanned images for OCR, resulting in less accurate text extraction.

In conclusion, based on the findings presented in the table, it can be concluded that the Pix2Pix models with ResNet and U-Net were more effective in improving OCR accuracy than the CycleGAN models. These results suggest that the Pix2Pix models are better suited for this particular task and may be more appropriate for enhancing OCR accuracy in similar situations.

Test Case 3:

Our objective in this test case was to compare the performance of the Pix2Pix and CycleGAN models in enhancing the quality of the scanned images. Unlike previous test cases, we used three different adversarial loss functions for each model, instead of using only the original loss function, to gain a deeper understanding of their effects. The three loss functions included in this test case are as follows:

- Original GAN Loss (Min-Max GAN Loss)
- LSGAN Loss (Least Squares GAN Loss)
- WGAN Loss (Wasserstein GAN Loss)

For more information on loss functions, please refer to Section 2.4.2.

The main focus of this test case was to evaluate the ability of the models to remove noise, artifacts, and imperfections, such as coffee stains from the images, and how the different loss functions affected their performance. We ensured a fair comparison by maintaining the same set of hyperparameters used in the previous test cases, except for the loss functions:

General Hyperparameter	
Normalization type	Instance normalization
Discriminator architecture	70×70 PatchGAN
Optimization method	Adam optimizer with beta values of (0.5, 0.999)
Batch size	1
Learning rate	0.0002
Special Hyperparameter	
Pix2Pix L1 Lambda	100
CycleGAN cycle Lambda	10

Quantitative Comparison:

We will begin the evaluation process for the third test case using the same quantitative comparison approach. Similar to the previous test cases, we utilized various established metrics (as outlined in Section 5.2) to objectively evaluate the models' performance in transforming images from one domain to another and compared it against the ground truth. In addition, we created line charts that showcase the model's performance at intervals of five epochs over a 100-epoch training period for each metric. This visualization technique enables us to easily compare the models and identify those that perform better over time. By utilizing these techniques, we can obtain a good understanding of the model's effectiveness.

The line charts in this test case show provide a representation of how the models progressed during the training phase for each of the three loss functions as measured by the metrics. Each chart is divided into four subcharts that describe a particular model and provide information for each of the three loss functions.

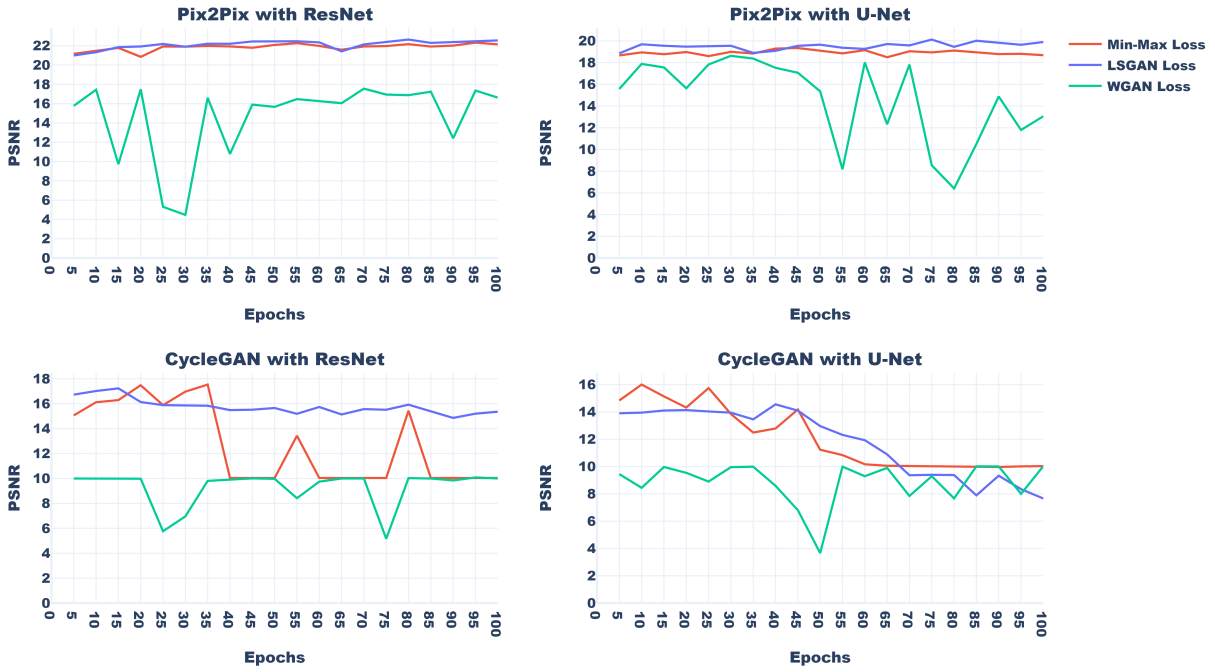


Figure A.5.: PSNR line charts (higher is better) for test case 3

We begin by presenting the results of the PSNR metric in Figure A.5. Upon initial observation, it is evident that both the Min-Max and LSGAN loss functions exhibit significantly superior performance in all models compared to the WGAN loss function. The latter demonstrated poor results and instability across boards. In contrast, the LSGAN loss function yields better overall outcomes and stability than other functions. Additionally, the Min-Max function appears to be superior to LSGAN during the early epochs of CycleGAN models; however, it fails to maintain stability, unlike LSGAN.

The Pix2Pix model with ResNet architecture maintains its superiority over the other models, even when using different loss functions. Specifically, it achieves the highest PSNR of 22.64 dB at epoch 80 using the LSGAN loss function, and 22.32 dB at epoch 95 using the Min-Max loss function. Similarly, the Pix2Pix version with U-Net architecture demonstrates the best results when using the LSGAN loss function, achieving a top PSNR of 20.12 dB at epoch 75. For the first time, the CycleGAN model with ResNet architecture exhibited performance stability when using the LSGAN loss function. In contrast, the CycleGAN model with the U-Net architecture shows less volatility when using the LSGAN loss function, but its performance gradually declines after epoch 40, with its highest PSNR value peaking at approximately 14.55 dB. Notably, CycleGAN with ResNet architecture maintains acceptable results ranging from 15 dB to 17.22 dB across all epochs.

Based on the aforementioned results, it can be deduced that the LSGAN loss function provides the most favorable and stable outcomes overall, whereas the WGAN loss function is entirely unsuitable for these models, resulting in poor outcomes for all tested models. Nevertheless, relying solely on one metric is inadequate, necessitating an assessment of the remaining two metrics. To further support this conclusion, we examine the outcomes of the SSIM metric, which

provides more precise information concerning the models' performance regarding luminance, contrast, and structural information.

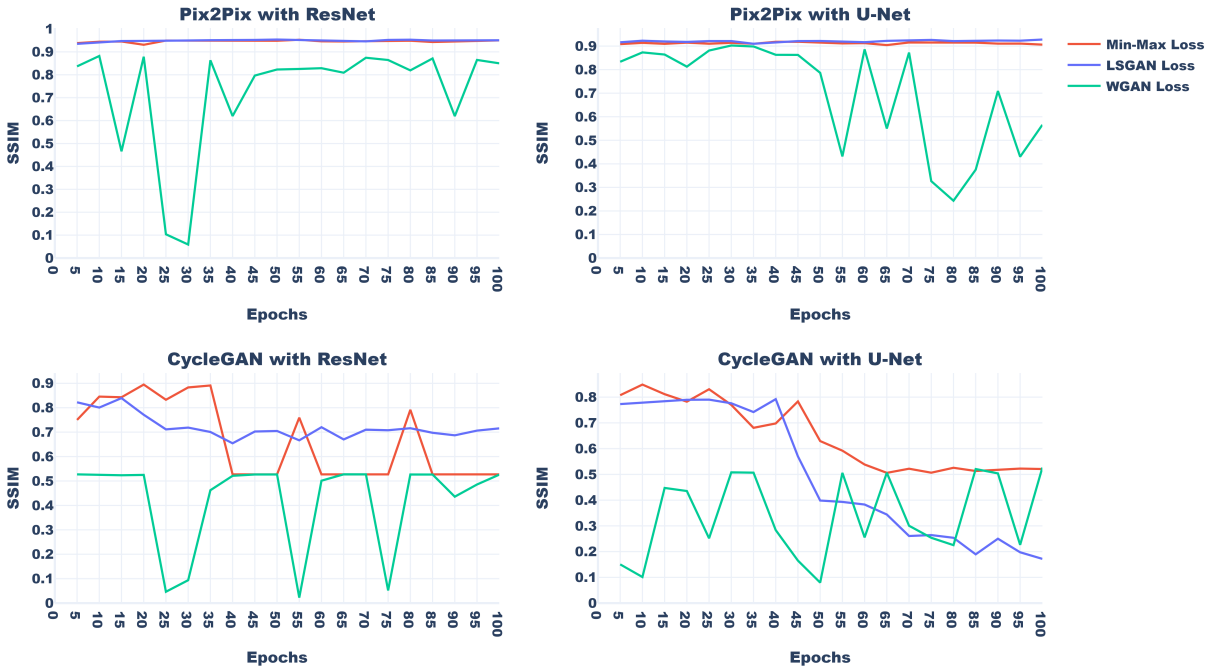


Figure A.6.: SSIM line charts (higher is better) for test case 3

As shown in Figure A.6, the results obtained from the SSIM metric are largely similar to those obtained from the PSNR, which supports our conclusion that the LSGAN loss function generally yields the best performance. However, that the CycleGAN model with U-Net architecture shows a more pronounced decline in SSIM values compared to PSNR when LSGAN is used.

In terms of specific results, the Pix2Pix model with ResNet architecture consistently demonstrates the best performance, with SSIM values ranging from 0.94 to 0.95 for both LSGAN and Min-Max loss functions. The Pix2Pix model with the U-Net architecture follows, with results ranging from 0.9 to 0.92. By contrast, the CycleGAN model exhibited weaker overall results. Nevertheless, it is noteworthy that the use of the LSGAN loss function and ResNet architecture provides some degree of stability to CycleGAN, with results ranging from 0.65 to 0.83.

Following the examination of the outcomes for the two previous metrics, we now turn our attention to the third measure, which yields more robust results because it considers the distribution of the generated images as a whole, as compared to the distribution of the real images, providing a more comprehensive evaluation of the models' overall performance.

As we see in Figure A.7, the FID results reveal a similar perfomance patterns across all models, supporting our previous conclusion that the LSGAN loss function exhibits the highest level of stability. Conversely, the WGAN loss function yielded poor results in all instances and was the least stable. Similar to the previous tests, the Pix2Pix model with ResNet architecture demonstrated superior performance compared to all other models, followed by Pix2Pix with U-

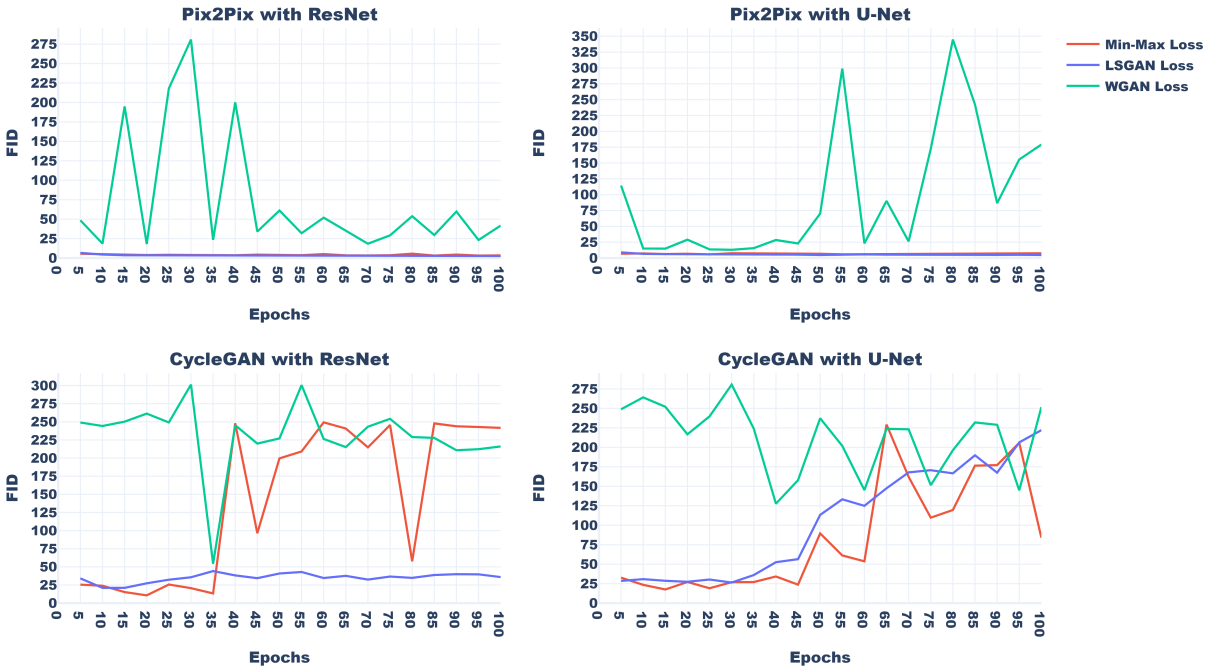


Figure A.7.: FID line charts (lower is better) for test case 3

Net architecture, CycleGAN with ResNet architecture, and CycleGAN with U-Net architecture. These findings are further elucidated in Figure A.8, which displays the mean of all epochs for each model in this test case.

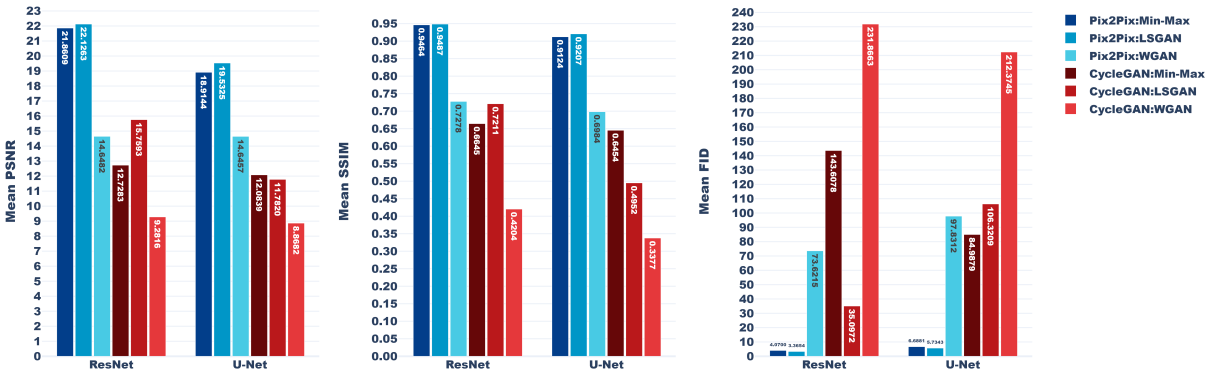


Figure A.8.: Test case 3

As depicted in the Figure A.8, it is evident that the utilization of various loss functions has a notable impact on the performance of the models. The results of this test case indicate that the use of the LSGAN loss function leads to the most favorable outcomes for all models, with the exception of CycleGAN with U-Net architecture, which consistently performs poorly across all configurations. The Min-Max loss function, on the other hand, produces results that are generally comparable to those of LSGAN, while the WGAN loss function consistently yields suboptimal results across all models.

The following table shows the results of the comparison more precisely, taking into account the architecture of the generator and the type of loss function:

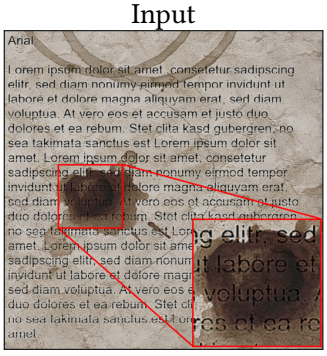
Model	Loss Function	Mean PSNR ↑	Mean SSIM ↑	Mean FID ↓
Pix2Pix with ResNet	Min-Max Loss	21.8609	0.9464	4.07
Pix2Pix with ResNet	LSGAN Loss	**22.1263**	**0.9487**	**3.3654**
Pix2Pix with ResNet	WGAN Loss	14.6482	0.7278	73.6215
Pix2Pix with U-Net	Min-Max Loss	18.9144	0.9124	6.6881
Pix2Pix with U-Net	LSGAN Loss	19.5325	0.9207	5.7343
Pix2Pix with U-Net	WGAN Loss	14.6457	0.6984	97.8312
CycleGAN with ResNet	Min-Max Loss	12.7283	0.6645	143.6078
CycleGAN with ResNet	LSGAN Loss	15.7593	0.7211	35.0972
CycleGAN with ResNet	WGAN Loss	9.2816	0.4204	231.8663
CycleGAN with U-Net	Min-Max Loss	12.0839	0.6454	84.9879
CycleGAN with U-Net	LSGAN Loss	11.7820	0.4952	106.3209
CycleGAN with U-Net	WGAN Loss	8.8682	0.3377	212.3745

Table A.3.: Test case 3 results table

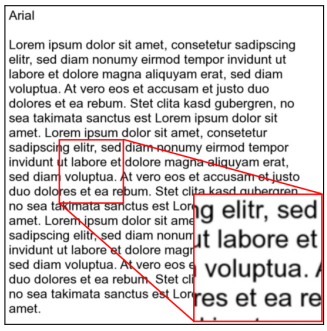
Visual Inspection:

Following the analysis of the quantitative measurements, we will proceed to visually inspect various samples to validate the effectiveness of the models. Specifically, we will examine just the good performing epochs, which include Pix2Pix with ResNet at epoch 95, Pix2Pix with U-Net at epoch 95, CycleGAN with ResNet at epoch 20, and CycleGAN with U-Net at epoch 10. The following tests showcase the outcomes of each example for a given input:

Input

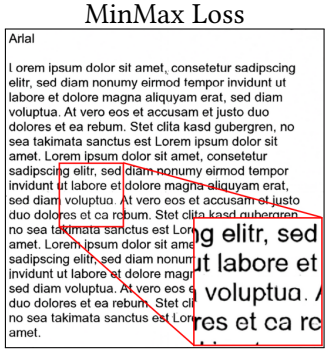


Ground truth

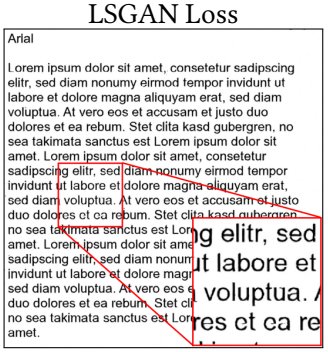


- Pix2Pix with ResNet:

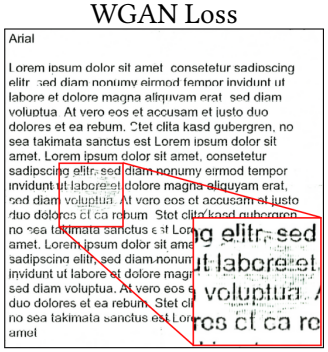
MinMax Loss



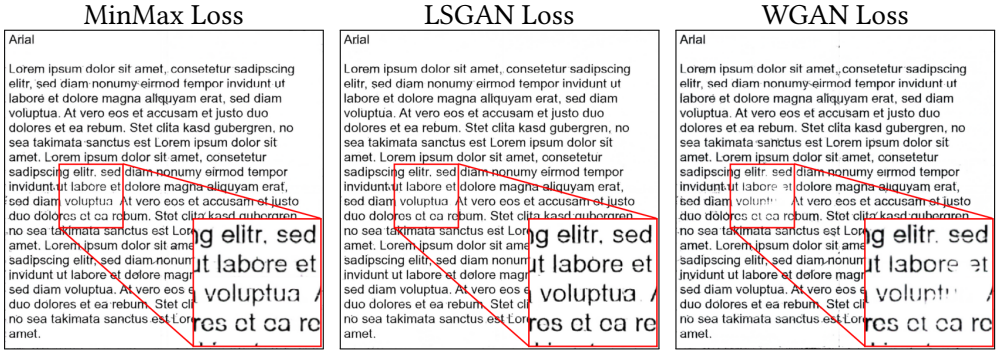
LSGAN Loss



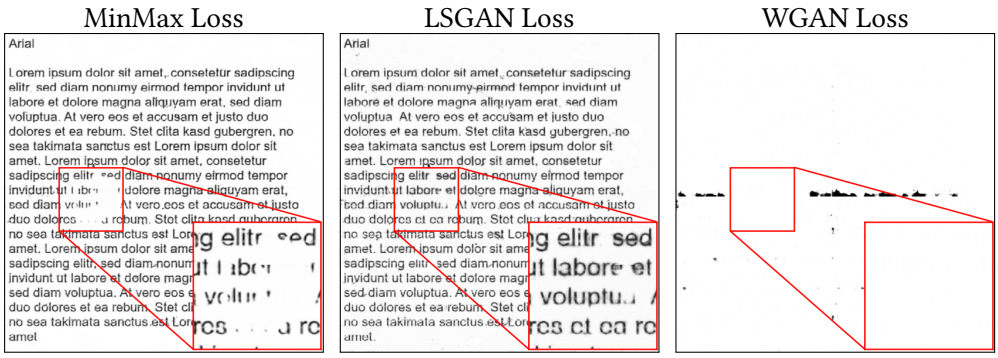
WGAN Loss



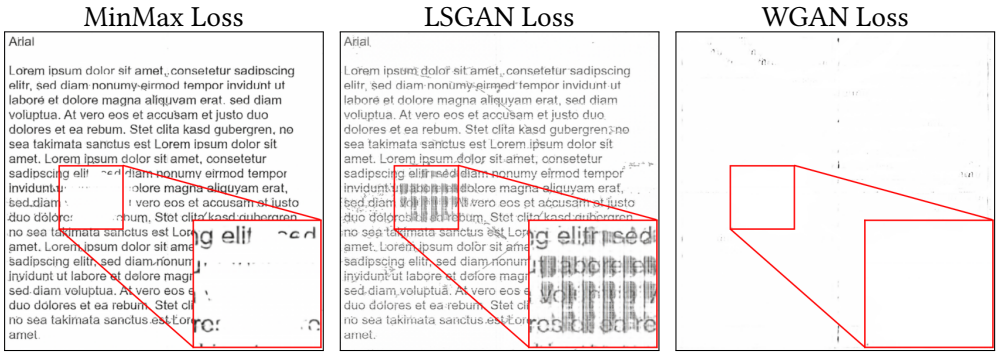
- Pix2Pix with U-Net:



- CycleGAN with ResNet:



- CycleGAN with U-Net:



Note that the red lines are not originally present in the images and have been added to highlight specific details.

The aforementioned examples serve as a representation of the various samples that were tested. For further samples, we kindly direct you to the appendix B where additional samples are available for review.

OCR recognition:

This section aims to evaluate the effectiveness of a model in improving the accuracy of Tesseract OCR, an open-source OCR engine, which is widely used for text recognition. The model's performance will be assessed on different test samples, as Tesseract OCR can encounter difficulties with low-quality scanned images. Therefore, the goal is to showcase the performance of the models on various test samples to determine its ability to enhance the accuracy of Tesseract OCR in text extraction using character error rate (CER) and word error rate (WER), which were introduced in Section 5.3.

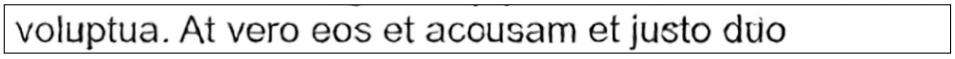
Sample: voluptua. At vero eos et accusam et justo duo

- **Source image:**



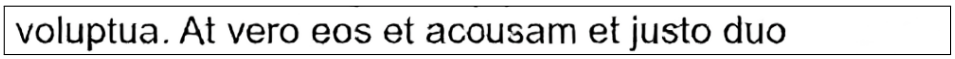
  Extracted text: Strange and incorrect symbols were extracted
  Character Error Rate: 1.0667
  Word Error Rate: 1.0

- **Pix2Pix with ResNet and Min-Max Loss:**



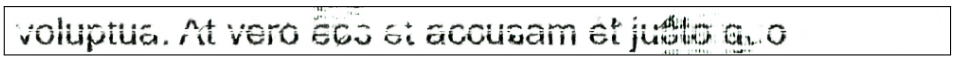
  Extracted text: voluptua. At vero eds et acduisam et justo duo
  Character Error Rate: 0.0667
  Word Error Rate: 0.2222

- **Pix2Pix with ResNet and LSGAN Loss:**



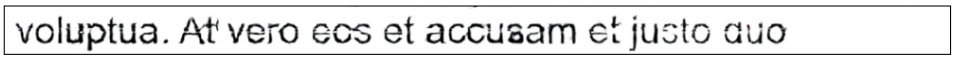
  Extracted text: voluptua. At vero eds et acduisam et justo duo
  Character Error Rate: 0.0667
  Word Error Rate: 0.2222

- **Pix2Pix with ResNet and WGAN Loss:**



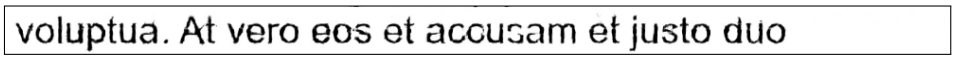
  Extracted text: Strange and incorrect symbols were extracted
  Character Error Rate: 1.8889
  Word Error Rate: 1.8889

- **Pix2Pix with U-Net and Min-Max Loss:**



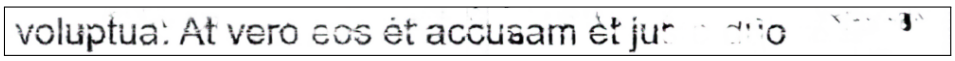
  Extracted text: voluptua. At‘ vero ecs et accuaam et justo auo
  Character Error Rate: 0.0889
  Word Error Rate: 0.4444

- **Pix2Pix with U-Net and LSGAN Loss:**



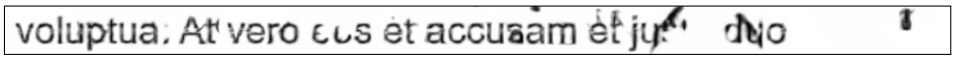
Extracted text: voluptua. At vero eds et accusam et justo duo
Character Error Rate: 0.0222
Word Error Rate: 0.1111

- **Pix2Pix with U-Net and WGAN Loss:**



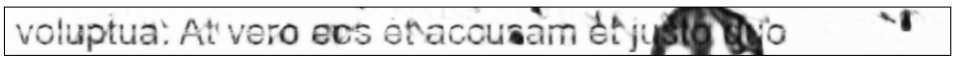
Extracted text: voluptua‘. At vero ass ét acbuisam étjuf ‘"0
Character Error Rate: 0.3111
Word Error Rate: 0.7778

- **CycleGAN with ResNet and Min-Max Loss:**



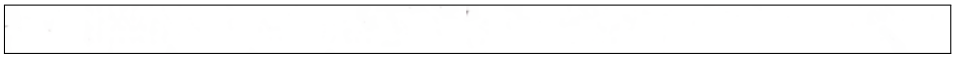
Extracted text: voluptua./>41‘ vero eos et accusém éf jué‘ duo
Character Error Rate: 0.2444
Word Error Rate: 0.5556

- **CycleGAN with ResNet and LSGAN Loss:**



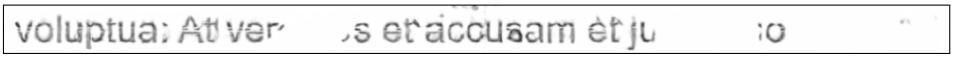
Extracted text: voiuptua‘. At‘ var ecé .:t‘accui5
Character Error Rate: 0.5778
Word Error Rate: 1.0

- **CycleGAN with ResNet and WGAN Loss:**



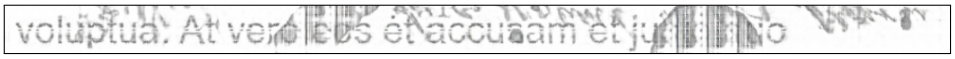
Extracted text:
Character Error Rate: 1.0
Word Error Rate: 1.0

- **CycleGAN with U-Net and Min-Max Loss:**



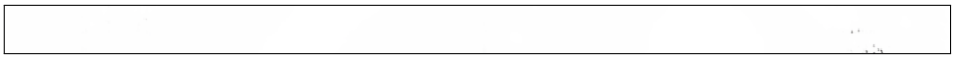
Extracted text: voluptua: Ar var’ ,5 erabéuaam em :0
Character Error Rate: 0.4889
Word Error Rate: 1.0

- **CycleGAN with U-Net and LSGAN Loss:**



Extracted text: Strange and incorrect symbols were extracted
Character Error Rate: 0.9333
Word Error Rate: 1.0

- **CycleGAN with U-Net and WGAN Loss:**



Extracted text:
Character Error Rate: 1.0
Word Error Rate: 1.0

Similar to the example provided earlier, we computed the mean of Character Error Rate and Word Error Rate for 100 different sentences, and obtained the subsequent outcomes for each model:

Model	**Loss Function**	**Mean CER ↓**	**Mean WER ↓**
Source	-	0.6435	0.8125
Pix2Pix with ResNet	Min-Max Loss	**0.0476**	**0.2374**
Pix2Pix with ResNet	LSGAN Loss	0.0526	0.2389
Pix2Pix with ResNet	WGAN Loss	0.1969	0.5118
Pix2Pix with U-Net	Min-Max Loss	0.0900	0.3883
Pix2Pix with U-Net	LSGAN Loss	0.0622	0.3023
Pix2Pix with U-Net	WGAN Loss	0.1297	0.4508
CycleGAN with ResNet	Min-Max Loss	0.1198	0.4296
CycleGAN with ResNet	LSGAN Loss	0.2379	0.5773
CycleGAN with ResNet	WGAN Loss	0.9988	1.005
CycleGAN with U-Net	Min-Max Loss	0.2312	0.5473
CycleGAN with U-Net	LSGAN Loss	0.4173	0.6587
CycleGAN with U-Net	WGAN Loss	0.9943	1.0

Table A.4.: Test case 3 OCR recognition results table

Table A.4 presents the evaluation of 13 different cases on 100 sentences, using two metrics, namely mean CER and mean WER. The models compared include the source image with no enhancements, as well as several variations of the Pix2Pix and CycleGAN models with different loss functions and network architectures.

The results show that the source image has the highest mean CER and WER, indicating that the models are improving the accuracy of the generated text. Among the Pix2Pix models, the best performance was achieved by the model using ResNet and Min-Max Loss, with a mean CER of 0.0476 and a mean WER of 0.2374. The worst performance among the Pix2Pix models

was observed for the model using ResNet and WGAN Loss, with a mean CER of 0.1969 and a mean WER of 0.5118.

For the CycleGAN models, the best performance was achieved by the model using ResNet and Min-Max Loss, with a mean CER of 0.1198 and a mean WER of 0.4296. The worst performance among the CycleGAN models was observed for the model using ResNet and WGAN Loss, with a mean CER of 0.9988 and a mean WER of 1.005.

Overall, the results suggest that the choice of loss function and network architecture can have a significant impact on the performance of image-to-image translation models. The Pix2Pix model with ResNet and Min-Max Loss appears to be the most effective, followed closely by the Pix2Pix model with ResNet and LSGAN Loss, and the Pix2Pix model with U-Net and LSGAN Loss. However, further experiments are needed to confirm these results and to explore the performance of these models on different datasets.

Test Case 4:

The primary objective of this test case was to conduct a comparative analysis of the efficacy of the Pix2Pix and CycleGAN models in enhancing the quality of scanned images by integrating a VGG loss function into each model's loss function. Specifically, the aim was to investigate whether the inclusion of a VGG loss function had a significant impact on model performance. For more information about the VGG loss function, please refer to section 2.2.1.

The main focus of this test case was to evaluate the models' ability to eliminate noise and imperfections such as coffee stains from images, and the impact of incorporating perceptual loss on their performance. To ensure a fair and unbiased comparison, we kept a consistent set of hyperparameters throughout the experiment which are as follows:

General Hyperparameter	
Adversarial loss function	LSGAN loss
Normalization type	Instance normalization
Discriminator architecture	70×70 PatchGAN
Optimization method	Adam optimizer with beta values of (0.5, 0.999)
Batch size	1
Learning rate	0.0002
Special Hyperparameter	
Pix2Pix L1 Lambda	100
CycleGAN cycle Lambda	10

In this particular test case, the LSGAN adversarial loss function was employed due to its superior performance observed in the previous test case. Additionally, it also confers some degree of robustness to the CycleGAN model.

Quantitative Comparison:

Similar to previous test cases, the evaluation process for the fourth test case commenced with the adoption of a quantitative comparison approach. To objectively evaluate the models' performance, we utilized the three metrics described in the 5.2 section. Furthermore, we will generate line charts illustrating the model's performance at 5-epochs intervals over a 100-epoch training period for each metric, similar to the previous test cases.

The line charts presented in this test case represent the models' progression throughout the training phase, measuring their performance using the metrics, regardless of whether the VGG loss function was used. Each chart is split into four subcharts, each describing a specific model.

Figure A.9 shows that the incorporation of the VGG loss function resulted in a slight improvement in the performance of the Pix2Pix models with both generator architectures, as determined by the PSNR metric, while maintaining a relatively consistent performance throughout all epochs. Conversely, the utilization of the VGG loss function in CycleGAN models with both generator structures resulted in significantly inferior outcomes.

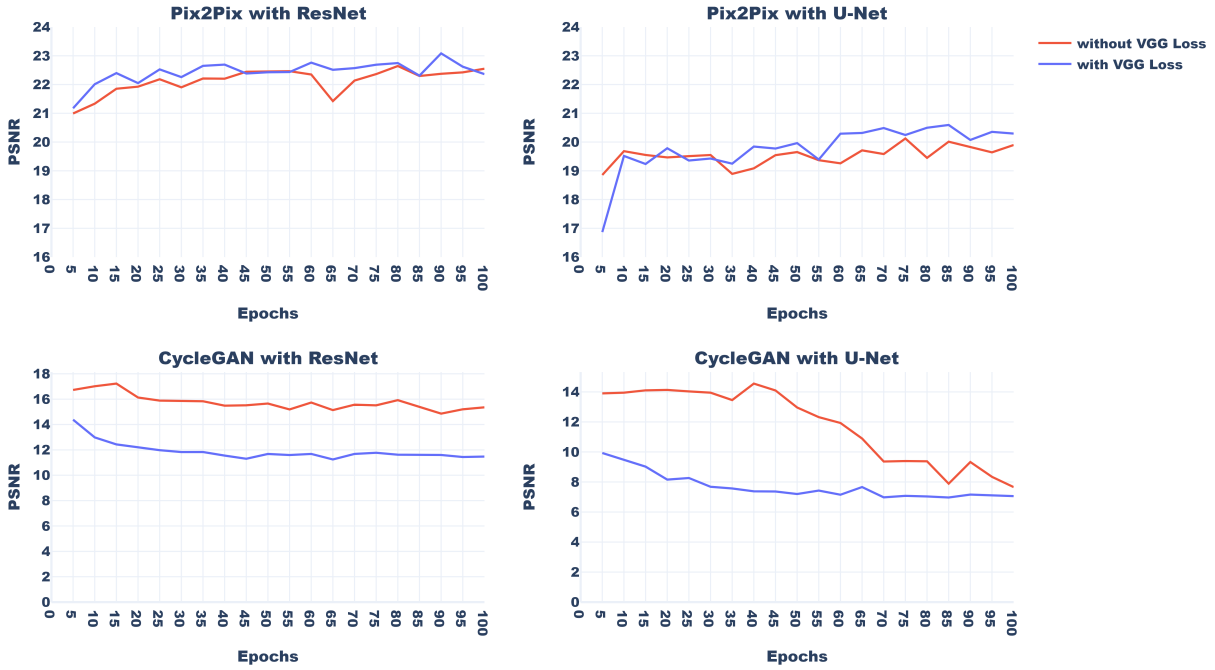


Figure A.9.: PSNR line charts (higher is better) for test case 4

The Pix2Pix model with ResNet architecture maintains its superiority over other models. As we have seen in all previous tests, even when using the VGG loss function, the model's performance continued to improve, while remaining superior to other models. Notably, the model achieved the highest PSNR of 23.08 dB in epoch 90 when using the VGG loss function, which is the best result obtained so far. Similarly, the Pix2Pix with the U-Net architecture exhibits better results with the VGG loss function, achieving a peak PSNR of 20.59 dB in epoch 85. However, the CycleGAN models with both architectures experienced a decline in performance when using the VGG loss function, likely because of the construction of the CycleGAN Loss function.

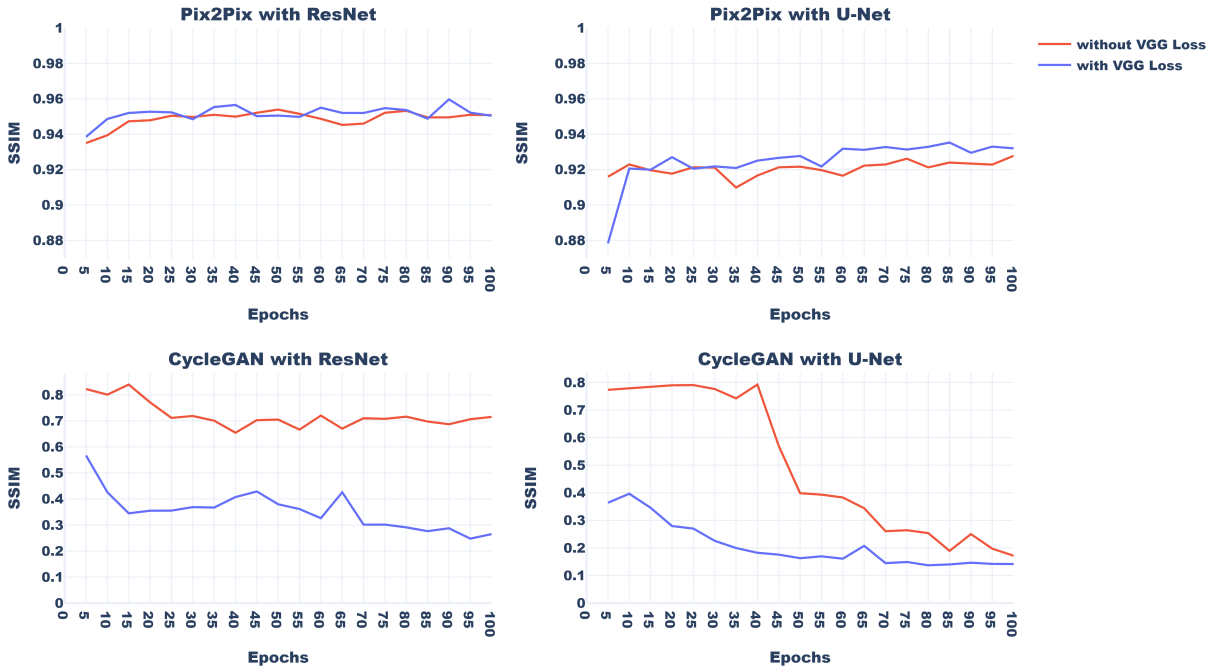


Figure A.10.: SSIM line charts (higher is better) for test case 4

A comparable pattern to the PSNR can also be observed in the SSIM metric for all the models in Figure A.10. This indicates that the Pix2Pix models for both generator architectures exhibit an enhanced ability to preserve structural information when the VGG loss function is incorporated, albeit only slightly. This is exemplified by the superior performance of the Pix2Pix model with the ResNet architecture, which achieves the highest score so far of 0.959 in epoch 90. Similarly, the Pix2Pix model with the U-Net architecture achieved the best result of 0.935 at epoch 85. However, the addition of the VGG loss function resulted in a significant decline in the performance of the CycleGAN models.

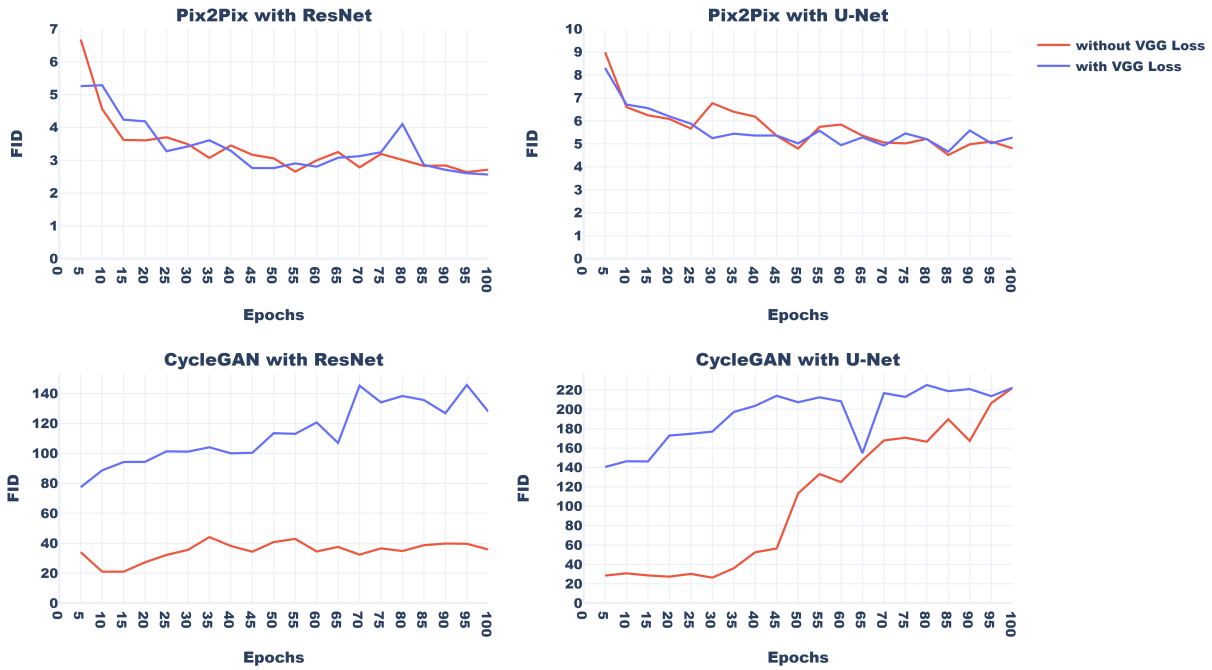


Figure A.11.: FID line charts (higher is better) for test case 4

As we can see in Figure A.11, the FID metric also shows similar behavior to PSNR and SSIM, which reinforces the observations made by these metrics. The Pix2Pix models with ResNet and U-Net architectures consistently outperform the CycleGAN models in terms of FID scores as the other metrics, which indicates that they are better at generating images that are closer to the ground truth.

Moreover, the addition of the VGG loss function has a positive impact on the performance of Pix2Pix models, as they achieve even lower FID scores in some epochs with VGG loss than without it. However, the same cannot be said for CycleGAN models, as their FID scores increase significantly with the inclusion of the VGG loss function.

To gain a better understanding of the overall performance of each model, we examined the means of their scores across all epochs for each metric. This is presented in Figure A.12.

As observed in Figure A.12, incorporating the VGG loss function results in a marginal enhancement in the performance of the Pix2Pix models across all metrics. By contrast, the addition of

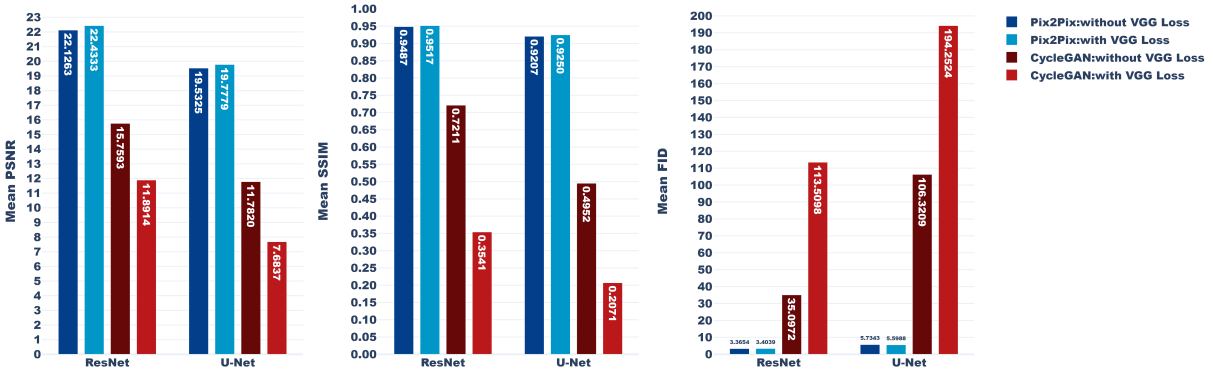


Figure A.12.: Test case 4

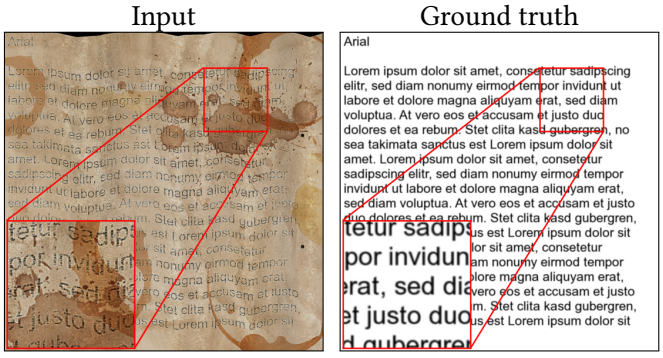
the VGG loss function diminished the quality of the CycleGAN models. We can see the results more accurately in table A.5

Model	With VGG	Mean PSNR ↑	Mean SSIM ↑	Mean FID ↓
Pix2Pix with ResNet	True	**22.4333**	**0.9517**	3.4039
Pix2Pix with ResNet	False	22.1263	0.9487	**3.3654**
Pix2Pix with U-Net	True	19.7779	0.9250	5.5988
Pix2Pix with U-Net	False	19.5325	0.9207	5.7343
CycleGAN with ResNet	True	11.8914	0.3541	113.5098
CycleGAN with ResNet	False	15.7593	0.7211	35.0972
CycleGAN with U-Net	True	7.6837	0.2071	194.2524
CycleGAN with U-Net	False	11.7820	0.4952	106.3209

Table A.5.: Test case 4 results table

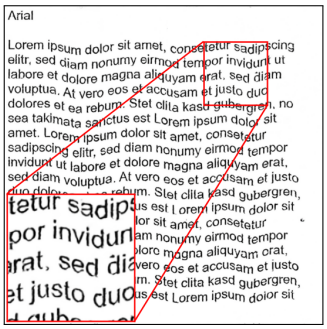
Visual Inspection:

After analyzing the quantitative measurements, we will visually inspect sample to verify the effectiveness of the models. We will focus on the good-performing epochs, namely Pix2Pix with ResNet at epoch 90, Pix2Pix with U-Net at epoch 85, CycleGAN with ResNet at epoch 15, and CycleGAN with U-Net at epoch 30. These tests will demonstrate the outcomes of each example for a given input:

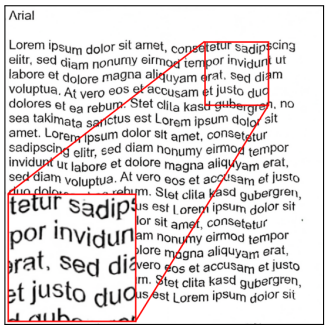


- Pix2Pix with ResNet:

Without VGG Loss

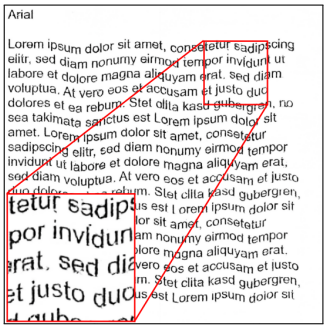


With VGG Loss

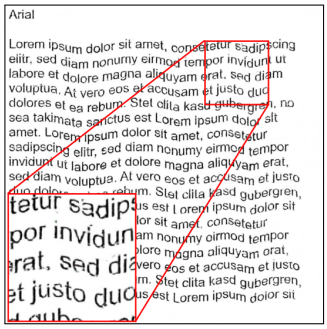


- Pix2Pix with U-Net:

Without VGG Loss

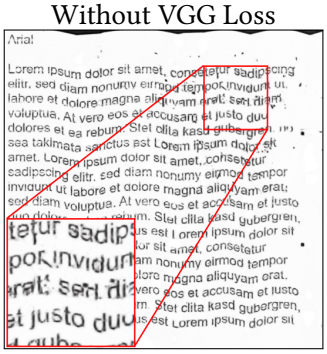


With VGG Loss

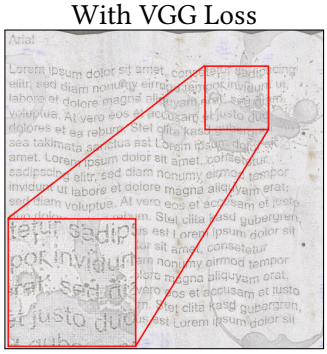


- CycleGAN with ResNet:

Without VGG Loss

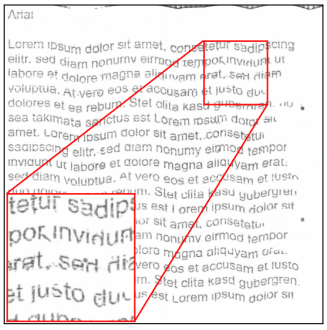


With VGG Loss

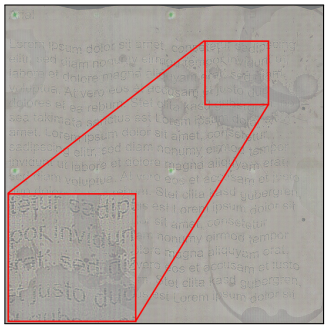


- CycleGAN with U-Net:

Without VGG Loss



With VGG Loss



Note that the red lines are not originally present in the images and have been added to highlight specific details.

The aforementioned example serve as a representation of one sample that was tested. For further samples, we kindly direct you to the appendix B where additional samples are available for review.

OCR recognition:

This section aims to evaluate the effectiveness of a model in improving the accuracy of Tesseract OCR, an open-source OCR engine, which is widely used for text recognition. The model's performance will be assessed on different test samples, as Tesseract OCR can encounter difficulties with low-quality scanned images. Therefore, the goal is to showcase the performance of the models on various test samples to determine its ability to enhance the accuracy of Tesseract OCR in text extraction using character error rate (CER) and word error rate (WER), which were introduced in Section 5.3.

Sample: amet. Lorem ipsum dolor sit amet, consetetur

- **Source image:**



Extracted text: orern ipsum dblfiametfl consetetur
Character Error Rate: 0.4091
Word Error Rate: 0.7143

- **Pix2Pix with ResNet (without VGG Loss):**



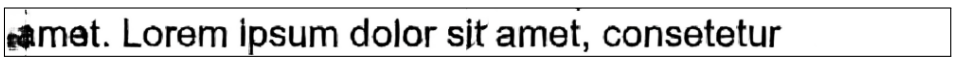
Extracted text: hmet. Lorem ipsum dolor sit amef, consetetur
Character Error Rate: 0.0455
Word Error Rate: 0.2857

- **Pix2Pix with ResNet (with VGG Loss):**



Extracted text: amet. Lorem ipsum dolor sit amef, consetetur
Character Error Rate: 0.0227
Word Error Rate: 0.1429

- **Pix2Pix with U-Net (without VGG Loss):**



Extracted text: gimet. Lorem ipsum dolor sit amef, consetetur
Character Error Rate: 0.0682
Word Error Rate: 0.2857

- **Pix2Pix with U-Net (with VGG Loss):**



Extracted text: ;amet. Lorem ipsum dolor sit amef, consetetur
Character Error Rate: 0.0455
Word Error Rate: 0.2857

- **CycleGAN with ResNet (without VGG Loss):**



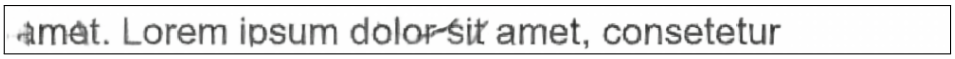
Extracted text: Lorem ipsum dolorsit amef, consetetur
Character Error Rate: 0.1818
Word Error Rate: 0.5714

- **CycleGAN with ResNet (with VGG Loss):**



Extracted text: m§tV§i'VLo rem ipsurn amef, consetetur
Character Error Rate: 0.4318
Word Error Rate: 0.8571

- **CycleGAN with U-Net (without VGG Loss):**



Extracted text: amet. Lorem ipsum dolorrsit amet. consetetur
Character Error Rate: 0.0455
Word Error Rate: 0.4286

- **CycleGAN with U-Net (with VGG Loss):**



Extracted text:
Character Error Rate: 1.0
Word Error Rate: 1.0

Similar to the example provided earlier, we computed the mean of Character Error Rate and Word Error Rate for 100 different sentences, and obtained the subsequent outcomes for each model:

Model	With VGG	Mean CER ↓	Mean WER ↓
Source	-	0.6435	0.8125
Pix2Pix with ResNet	False	**0.0494**	**0.2376**
Pix2Pix with ResNet	True	0.0522	0.2521
Pix2Pix with U-Net	False	0.0625	0.3041
Pix2Pix with U-Net	True	0.0640	0.2933
CycleGAN with ResNet	False	0.2379	0.5773
CycleGAN with ResNet	True	0.6497	0.7817
CycleGAN with U-Net	False	0.3101	0.5791
CycleGAN with U-Net	True	0.8340	0.9243

Table A.6.: Test case 4 OCR recognition results table

Table A.6 presents the evaluation of 9 different cases on 100 sentences, using two evaluation metrics, mean CER and mean WER, for OCR recognition. The models being compared include the source image with no enhancements, as well as several variations of the Pix2Pix and CycleGAN models with different network architectures, and with or without VGG Loss.

Firstly, we can see that the source image without any enhancements has the higher mean CER and mean WER compared to all the models except for CycleGAN with U-Net and VGG, which indicates that the models are improving the accuracy of the OCR recognition.

Among the Pix2Pix models, the best performance was observed for the model using ResNet without VGG Loss, with a mean CER of 0.0494 and a mean WER of 0.2376. This model outperformed the Pix2Pix models using U-Net architecture, which had higher mean CER and mean WER values. Adding VGG Loss did not significantly improve the performance of the Pix2Pix models.

For the CycleGAN models, the best performance was observed for the model using ResNet without VGG features, with a mean CER of 0.2379 and a mean WER of 0.5773. However, this model's performance was still not as good as the best-performing Pix2Pix model. Adding VGG Loss to the CycleGAN models had a negative effect on their performance, as all the models had higher mean CER and mean WER values when VGG Loss were added.

Overall, the results suggest that the Pix2Pix model using ResNet architecture without VGG features is the most effective model for OCR recognition, as it outperforms all the other models in both mean CER and mean WER values. It is interesting to note that adding VGG Loss did not improve the performance of the OCR, and in some cases, had a negative effect. However, further experiments may be needed to confirm these results and to explore the performance of these models on different OCR recognition datasets.

Test Case 5:

In this test case, the main goal was to conduct a comparative analysis of the effectiveness of the Pix2Pix and CycleGAN models in enhancing the quality of scanned images by varying the learning rate for each model. Specifically, we aimed to determine whether decreasing the learning rate has a significant impact on the stability and performance of the models. The focus of the analysis is to evaluate the models' capability to eliminate noise and imperfections, such as coffee stains, from the images and to examine the impact of the learning rate on their performance. To ensure a fair comparison, we maintained a constant set of hyperparameters throughout the experiment, as listed below:

General Hyperparameter	
Adversarial loss function	LSGAN loss
Normalization type	Instance normalization
Discriminator architecture	70×70 PatchGAN
Optimization method	Adam optimizer with beta values of (0.5, 0.999)
Batch size	1
Special Hyperparameter	
Pix2Pix L1 Lambda	100
CycleGAN cycle Lambda	10

For this test case, the learning rates utilized were 0.0001 and 0.0002.

Quantitative Comparison:

Similar to the preceding evaluations, the fifth test case begins with a quantitative comparison approach to assess model performance. An objective evaluation of the models will be performed using three metrics: PSNR, SSIM, and FID (see Section 5.2). The performance is presented through line charts that display the progress at 5-epochs over the course of the 100-epoch training period, analogous to the prior tests. Performance metrics for each learning rate will be represented through subchart, demonstrating each model's progress throughout the training process.

The results depicted in Figure A.13 reveal that a learning rate of 0.0001 marginally degraded the PSNR performance of the Pix2Pix models when compared to a learning rate of 0.0002, while maintaining consistent performance across all epochs. Conversely, learning rate 0.0001 exhibited superior PSNR performance for CycleGAN models, particularly in the early epochs with the ResNet architecture and generally with the U-Net architecture.

Consistent with previous test cases, the Pix2Pix model with ResNet architecture retains its superiority over the other models, even when the learning rate is altered, as evidenced by the findings. It is worth highlighting that the model yielded the highest PSNR of 22.64 dB in the 80th epoch with a learning rate of 0.0002 and the highest PSNR of 22.37 dB in the 80th epoch with a 0.0001 learning rate, which are close results with a slight drop when using a smaller learning rate. Similarly, the Pix2Pix model with the U-Net architecture exhibited a similar pattern.

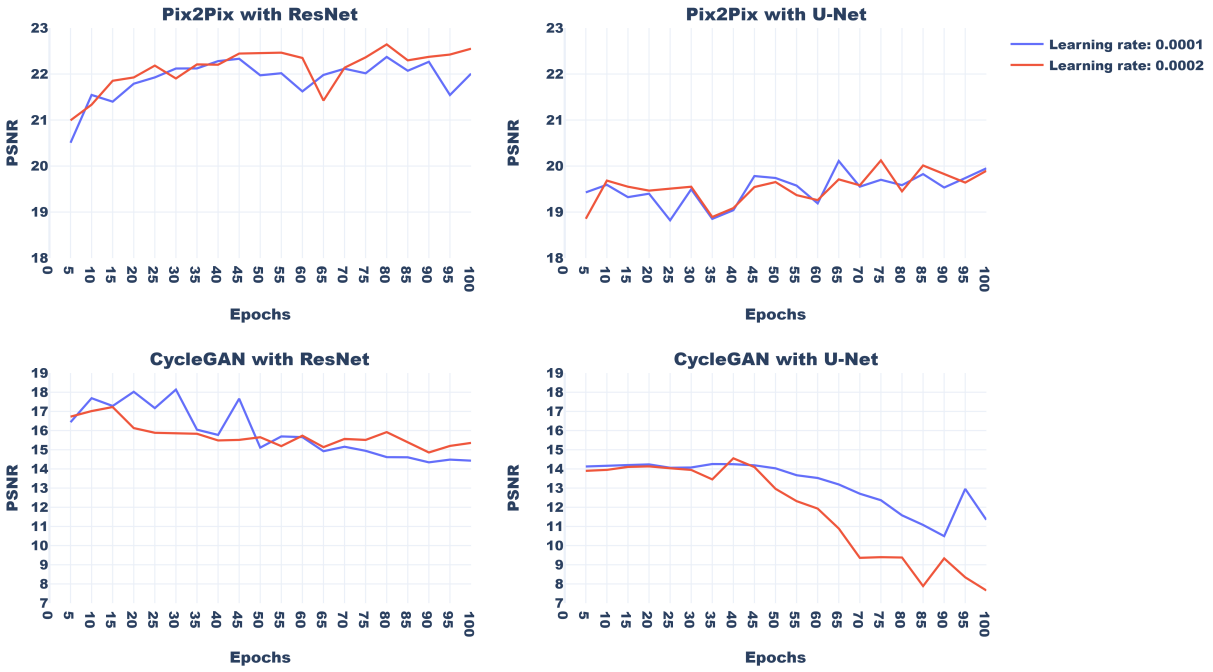


Figure A.13.: PSNR line charts (higher is better) for test case 5

However, both architectures of the CycleGAN models demonstrate improved performance at a learning rate of 0.0001, with the ResNet architecture of CycleGAN still outperforming U-Net. Remarkably, the CycleGAN model with ResNet architecture achieved the highest PSNR of 18.13 dB at epoch 30 with a learning rate of 0.0001 and the highest PSNR of 17.22 dB at epoch 15 with a learning rate of 0.0002.

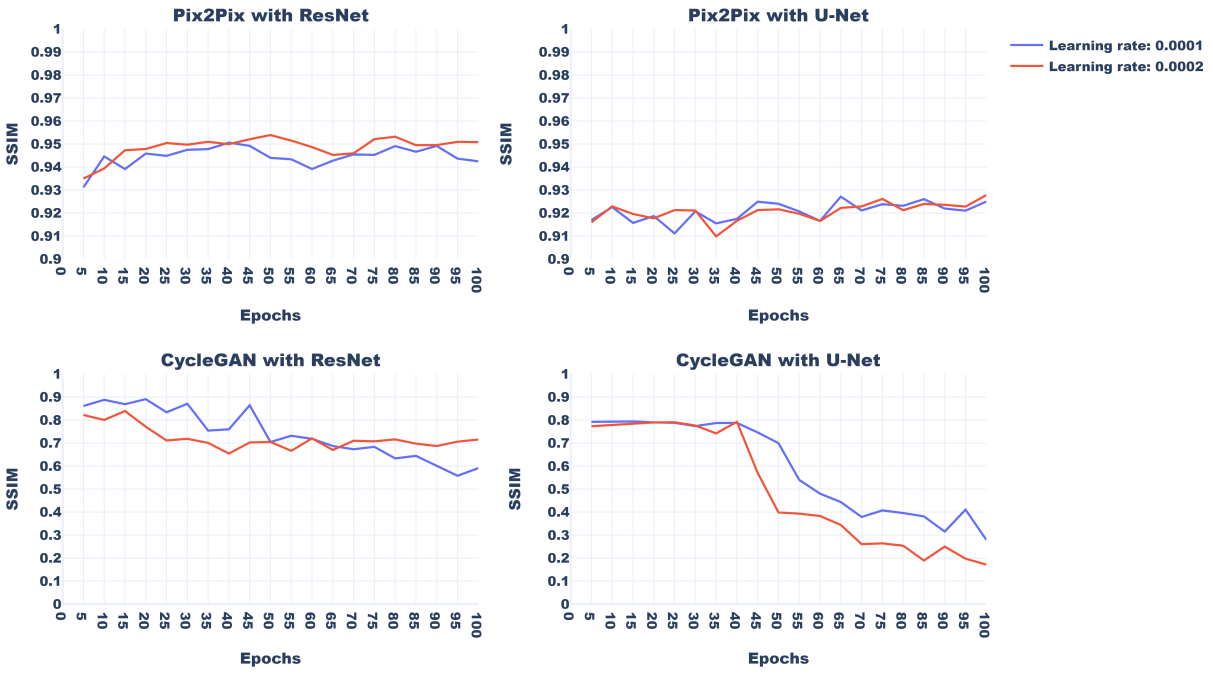


Figure A.14.: SSIM line charts (higher is better) for test case 5

The SSIM metric displayed in Figure A.14 exhibits a trend parallel to that of PSNR across all models. This indicates that the Pix2Pix models for both generator architectures show a good ability to preserve structural information for both learning rates with a very small superiority in the learning rate of 0.0002. On the other hand, reducing the learning rate shows a good improvement in the performance of the CycleGAN models in preserving structural information, but Pix2Pix is still highly concordant. Noting that the performance of CycleGAN, with a learning rate of 0.0001, declined after epoch 65 under a learning rate of 0.0002.



Figure A.15.: FID line charts (higher is better) for test case 5

As we can see in Figure A.15, the FID metric also shows similar behavior to PSNR and SSIM, which reinforces the observations made by these metrics. The Pix2Pix models with ResNet and U-Net architectures consistently outperform the CycleGAN models in terms of FID scores as the other metrics, which indicates that they are better at generating images that are closer to the ground truth.

Moreover, using a learning rate of 0.0001 had a slightly negative effect on the performance of the Pix2Pix models, as they achieved higher FID scores in some epochs compared to a learning rate of 0.0002. However, the same cannot be said for the CycleGAN models, as FID scores with a learning rate of 0.0001 are shown to be lower than a learning rate of 0.0002, interestingly, the quality of the CycleGAN models worsens as the training progresses for both learning rates.

To gain a more understanding of the overall performance of each model, we can examine the mean of their scores across all epochs for each metric. This is presented in Figure A.16.

As shown in the bar chart A.16, reducing the learning rate leads to a slight decrease in the performance of the Pix2Pix models for all metrics. However, decreasing the learning rate results

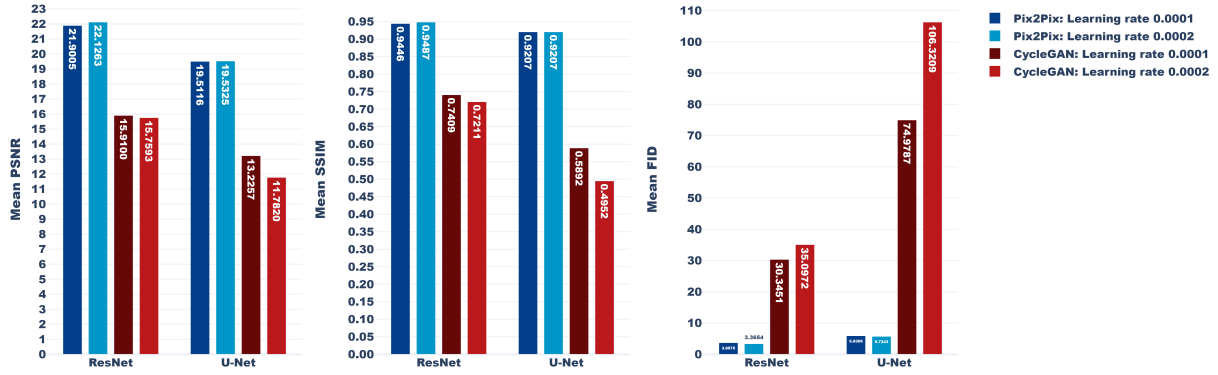


Figure A.16.: Test case 5

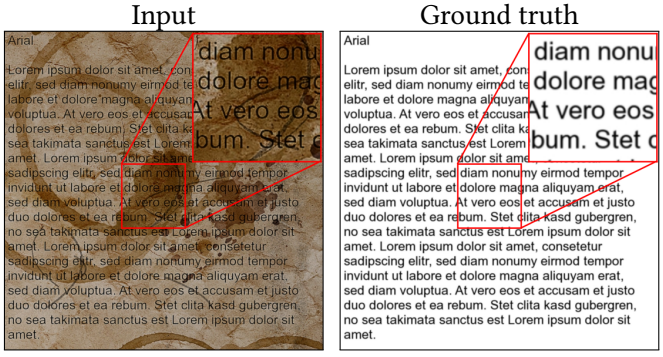
in an overall improvement in the quality of the CycleGAN models. We can see the results more accurately in table A.7

Model	Learning rate	Mean PSNR ↑	Mean SSIM ↑	Mean FID ↓
Pix2Pix with ResNet	0.0001	21.9005	0.9446	3.6975
Pix2Pix with ResNet	0.0002	**22.1263**	**0.9487**	**3.3654**
Pix2Pix with U-Net	0.0001	19.5116	0.9207	5.9305
Pix2Pix with U-Net	0.0002	19.5325	0.9207	5.7343
CycleGAN with ResNet	0.0001	15.9100	0.7409	30.3451
CycleGAN with ResNet	0.0002	15.7593	0.7211	35.0972
CycleGAN with U-Net	0.0001	13.2257	0.5892	74.9787
CycleGAN with U-Net	0.0002	11.7820	0.4952	106.3209

Table A.7.: Test case 5 results table

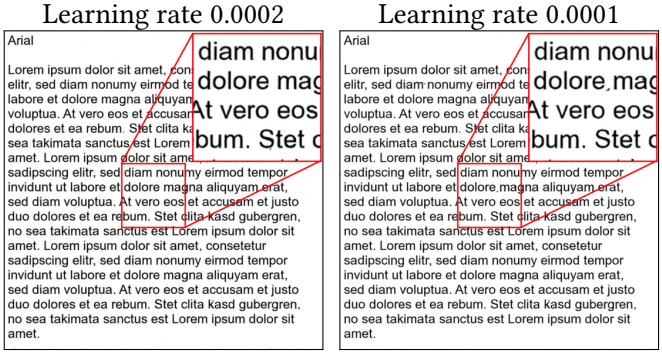
Visual Inspection:

Following the analysis of the quantitative measurements, we will proceed to visually inspect various samples to validate the effectiveness of the models. Specifically, we will examine just the good performing epochs, which include Pix2Pix with ResNet at epoch 90, Pix2Pix with U-Net at epoch 85, CycleGAN with ResNet at epoch 15, and CycleGAN with U-Net at epoch 30. The following tests showcase the outcomes of each example for a given input:



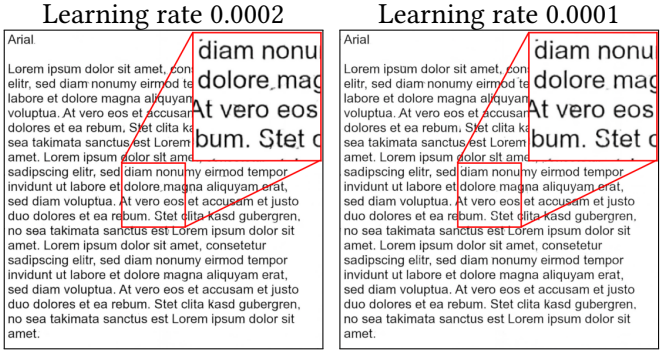
- Pix2Pix with ResNet:

Learning rate 0.0002 | Learning rate 0.0001



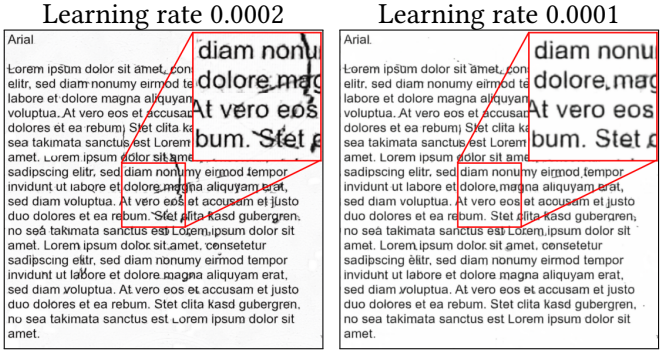
- Pix2Pix with U-Net:

Learning rate 0.0002 | Learning rate 0.0001



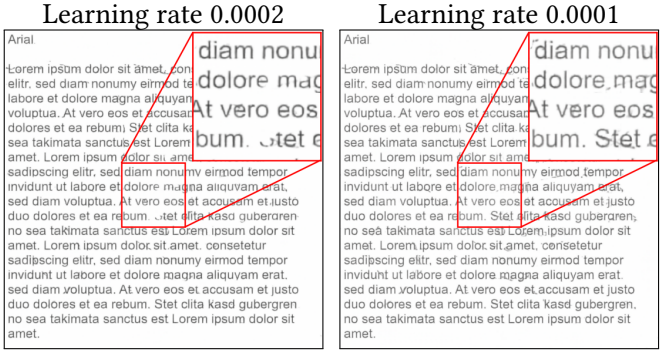
- CycleGAN with ResNet:

Learning rate 0.0002 | Learning rate 0.0001



- CycleGAN with U-Net:

Learning rate 0.0002 | Learning rate 0.0001



Note that the red lines are not originally present in the images and have been added to highlight specific details.

The aforementioned example serve as a representation of one sample that was tested. For further samples, we kindly direct you to the appendix B where additional samples are available for review.

OCR recognition:

This section aims to evaluate the effectiveness of the models in improving the accuracy of Tesseract OCR. The model's performance will be assessed on different test samples, as Tesseract OCR can encounter difficulties with low-quality scanned images. Therefore, the goal is to showcase the performance of the models on various test samples to determine its ability to enhance the accuracy of Tesseract OCR in text extraction using character error rate (CER) and word error rate (WER), which were introduced in Section 5.3.

Sample: sea takimata sanctus est Lorem ipsum dolor sit

- **Source image:**



  Extracted text: Strange and incorrect symbols were extracted
  Character Error Rate: 1.0217
  Word Error Rate: 0.875

- **Pix2Pix with ResNet (With Learning rate 0.0001):**



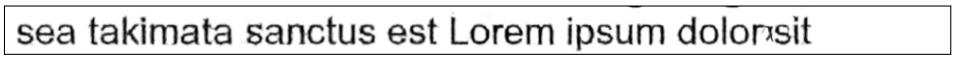
  Extracted text: sea takimata sanctus est Lorem ipsuim dolbrsit
  Character Error Rate: 0.0652
  Word Error Rate: 0.375

- **Pix2Pix with ResNet (With Learning rate 0.0002):**



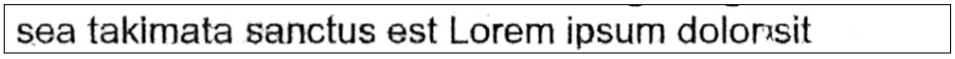
  Extracted text: sea takimata sanctus est Lorem ipsuim dolhr sit
  Character Error Rate: 0.0435
  Word Error Rate: 0.25

- **Pix2Pix with U-Net (With Learning rate 0.0001):**



  Extracted text: sea takimata sanctus est Lorem ipsuim dolbnsit
  Character Error Rate: 0.087
  Word Error Rate: 0.375

- **Pix2Pix with U-Net (With Learning rate 0.0002):**



Extracted text: sea taklmata sanctus est Lorem ipsuvm dolbrxsit
Character Error Rate: 0.087
Word Error Rate: 0.5

- **CycleGAN with ResNet (With Learning rate 0.0001):**



Extracted text: sea lakimata sanctus est Lorem ipsuim dolbrxslt
Character Error Rate: 0.1087
Word Error Rate: 0.5

- **CycleGAN with ResNet (With Learning rate 0.0002):**



Extracted text: sea takimala sanctus est Lorem Ip§u‘m dolE>nsi1
Character Error Rate: 0.1739
Word Error Rate: 0.5

- **CycleGAN with U-Net (With Learning rate 0.0001):**



Extracted text: sea takimata sanctus est Lorem iosu_rn dolbrxsit
Character Error Rate: 0.1304
Word Error Rate: 0.375

- **CycleGAN with U-Net (With Learning rate 0.0002):**



Extracted text: sea takimata sanctus est Lorem ipsum dolomslt
Character Error Rate: 0.0652
Word Error Rate: 0.25

Similar to the example provided earlier, we computed the mean of Character Error Rate and Word Error Rate for 100 different sentences, and obtained the subsequent outcomes for each model:

Model	Learning rate	Mean CER ↓	Mean WER ↓
Source	-	0.6435	0.8125
Pix2Pix with ResNet	0.0001	0.0523	0.2520
Pix2Pix with ResNet	0.0002	**0.0494**	**0.2376**
Pix2Pix with U-Net	0.0001	0.0598	0.2888
Pix2Pix with U-Net	0.0002	0.0625	0.3041
CycleGAN with ResNet	0.0001	0.1765	0.4957
CycleGAN with ResNet	0.0002	0.2401	0.5750
CycleGAN with U-Net	0.0001	0.3198	0.6123
CycleGAN with U-Net	0.0002	0.2935	0.5868

Table A.8.: Test case 4 OCR recognition results table

Table A.8 shows the performance of the different models and learning rates for OCR recognition, as measured by the mean CER and WER. The first row in the table labeled "Source" represents the baseline performance without any image-to-image translation. The other rows represent the performance of different models using different learning rates.

From the table, we can see that the best-performing model is Pix2Pix with ResNet using a learning rate of 0.0002, CER of 0.0494, and WER of 0.2376. This model outperforms the baseline by a large margin, indicating that image-to-image translation can significantly improve OCR recognition performance.

We can also observe that using a higher learning rate generally leads to worse performance in some models, as seen in the higher CER and WER values for Pix2Pix with U-Net and CycleGAN with ResNet when trained with a learning rate of 0.0002, compared to 0.0001. This is likely due to overfitting caused by the higher learning rate, which can cause the model to converge too quickly and not generalize well to new data.

Overall, this table provides insight into the impact of different image-to-image translation models and learning rates on OCR recognition performance. However, to ensure the reliability of these findings and to explore the efficacy of these models on diverse OCR datasets, it may be necessary to conduct further experiments, as the results are somewhat divergent from the quantitative performance evaluation. The underlying reason for this discrepancy lies in the fact that our study only employed a sample of 100 sentences, which is comparatively small, and the law of small numbers posits that such small samples can potentially lead to inconclusive or equivocal outcomes.

Test Case 6:

The main goal of this test case was to perform a comparative analysis of Pix2Pix models to evaluate their effectiveness in improving the quality of scanned images. The analysis compared the performance of the models with and without the L1 loss function, which is a component of the Pix2Pix loss function (see Section 4.2.3). The primary focus was to determine whether the addition of the L1 loss function to the cGAN loss function had a significant impact on the model results. The evaluation primarily concentrates on assessing the capacity of the models to eliminate noise and defects, such as coffee stains, from the images and the influence of the L1 loss function on the performance of the Pix2Pix models. To ensure a consistent and fair comparison, we kept the hyperparameters constant throughout the experiment, and they are listed below:

General Hyperparameter	
Adversarial loss function	Min-Max loss
Normalization type	Instance normalization
Discriminator architecture	70×70 PatchGAN
Optimization method	Adam optimizer with beta values of (0.5, 0.999)
Batch size	1
Learning rate	0.0002

Quantitative Comparison:

The sixth test case employed a quantitative comparison approach, similar to the previous cases, to evaluate the performance of the models. The evaluation is conducted objectively using three metrics: PSNR, SSIM, and FID, as explained in Section 5.2. Performance will be presented through line charts, which will display the progress of the models in 5 epochs over a 100 epoch training period with and without L1 Loss.

The results of the test case are presented in Figure A.17, which demonstrates that using the L1 loss function leads to a significant improvement in the PSNR performance of the Pix2Pix models at all epochs when compared to not using it. This improvement is due to the ability of the L1 loss function to preserve small details during image conversion from one domain to another.
The Pix2Pix model with ResNet architecture was the best performer, achieving the highest score of 22.32 dB in epoch 95 when the L1 loss function was used. On the other hand, the highest score without L1 is 21.48 dB in epoch 95. Similarly, the U-Net architecture Pix2Pix model, achieving a score of 19.34 dB in epoch 45 when the L1 loss function is utilized, whereas the best score without L1 is 18.76 dB in epoch 70.

Figure A.18 shows a similar pattern in the SSIM metric to that of the PSNR for all models. This observation implies that the Pix2Pix models with both generator architectures exhibit an enhanced ability to preserve structural information when the L1 loss function is used in almost all epochs. Among the two models, the Pix2Pix model with ResNet architecture remained the best performer, achieving the highest score of 0.9528 in the 55th epoch when the L1 loss

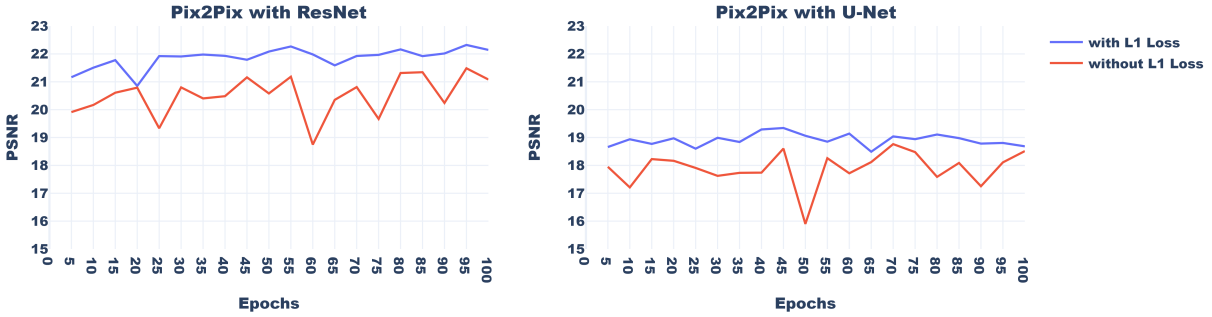


Figure A.17.: PSNR line charts (higher is better) for test case 6

function was used, while the highest score without L1 was 0.9412 in the 95th epoch.
In addition, the Pix2Pix model with the U-Net architecture also behaved similarly, achieving a score of 0.9184 in epoch 45 when using the L1 loss function, whereas the best score without L1 was 0.9130 in epoch 70. Interestingly, the model without L1 outperformed at epoch 100 for this metric.

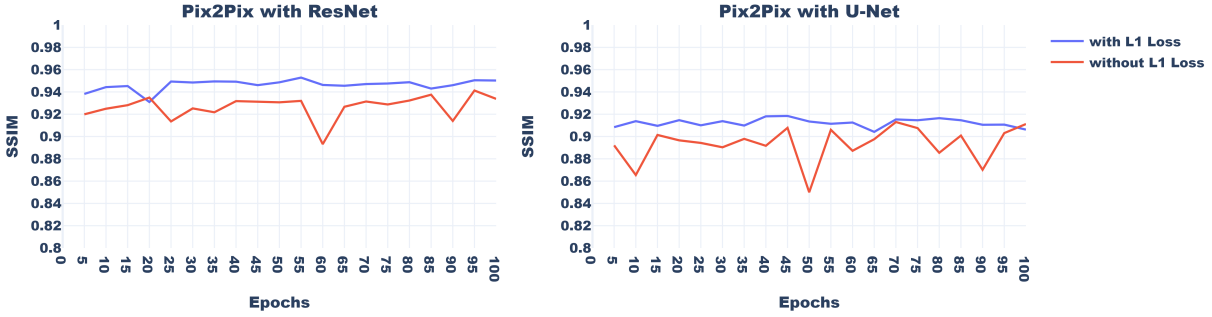


Figure A.18.: SSIM line charts (higher is better) for test case 6

The results of the FID scores reinforce our observations from the previous metrics, as shown in Figure A.19. The models that incorporate the L1 loss function consistently outperform the others across all the cases. In addition, the use of L1 provides greater stability during training, resulting in less fluctuation in the results. Conversely, the absence of L1 often leads to fluctuations in results.

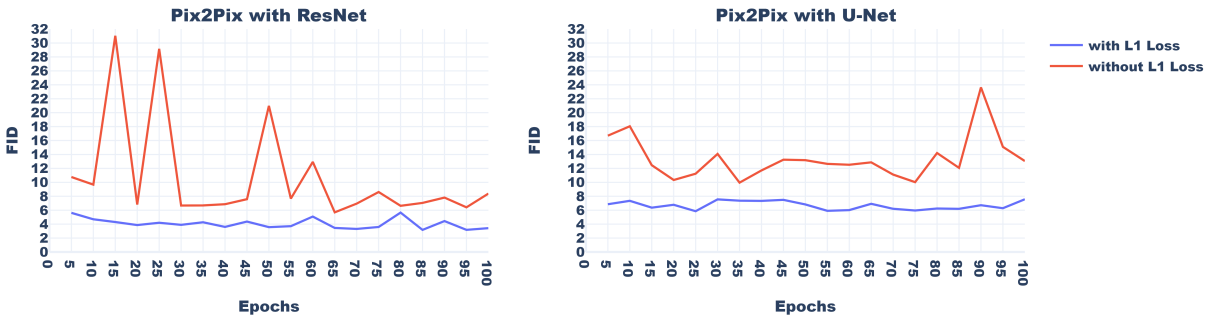


Figure A.19.: FID line charts (higher is better) for test case 6

To gain better insights into the performance of each model, you can look to the Figure A.20, which presents the mean score for each metric across all epochs.

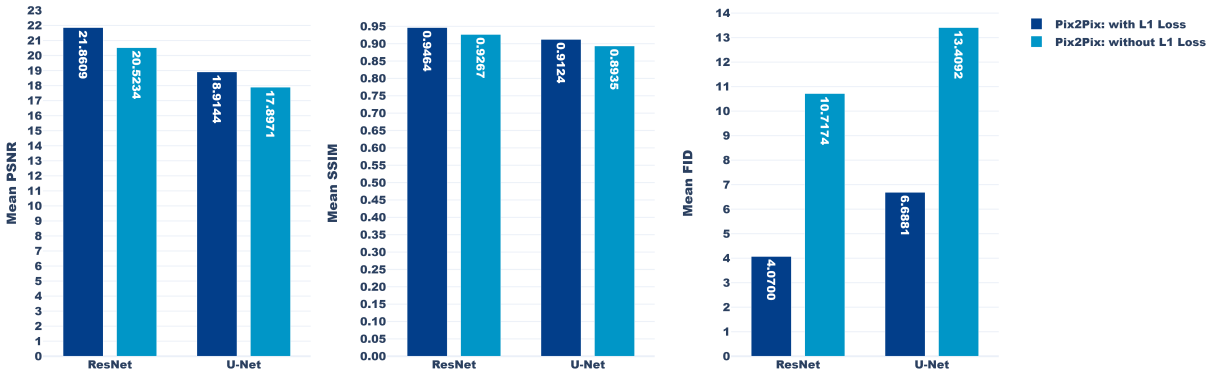


Figure A.20.: Test case 6

As we can see in the bar Figure A.20, reducing the learning rate leads to a slight decrease in the performance of the Pix2Pix models for all metrics. On the other hand, decreasing the learning rate results in an overall improvement in the quality of the CycleGAN models. We can see the results more accurately in table A.9

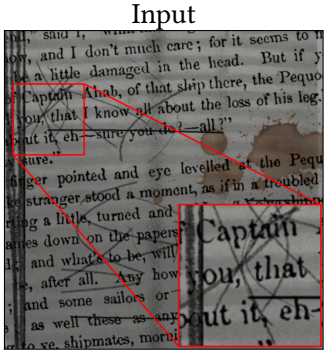
Model	**With L1 Loss**	**Mean PSNR ↑**	**Mean SSIM ↑**	**Mean FID ↓**
Pix2Pix with ResNet	False	20.5234	0.9267	10.7174
Pix2Pix with ResNet	True	**21.8609**	**0.9464**	**4.0700**
Pix2Pix with U-Net	False	17.8971	0.8935	13.4092
Pix2Pix with U-Net	True	18.9144	0.9124	6.6881

Table A.9.: Test case 6 results table

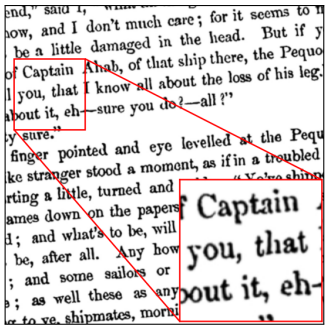
Visual Inspection:

Once the quantitative measurements have been analyzed, we will proceed to visually inspect various samples to validate the models' effectiveness. Specifically, we will examine only the epochs that performed well, such as Pix2Pix with ResNet at epoch 90, Pix2Pix with U-Net at epoch 85, CycleGAN with ResNet at epoch 15, and CycleGAN with U-Net at epoch 30. The subsequent tests will demonstrate the outcomes of each example for a given input:

Input

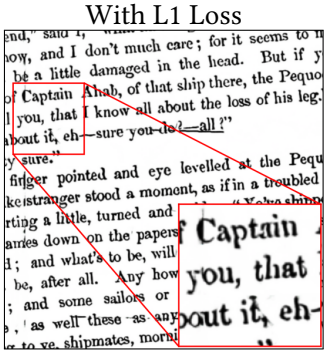


Ground truth

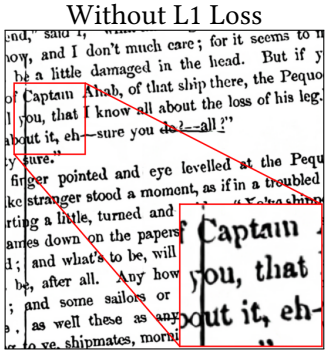


- Pix2Pix with ResNet:

With L1 Loss

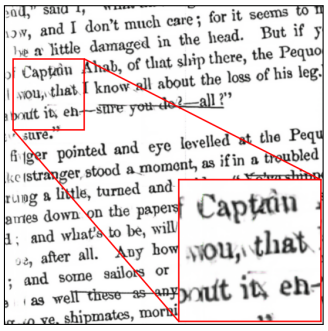


Without L1 Loss

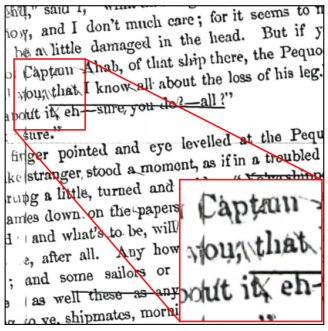


- Pix2Pix with U-Net:

With L1 Loss



Without L1 Loss



Note that the red lines are not originally present in the images and have been added to highlight specific details.

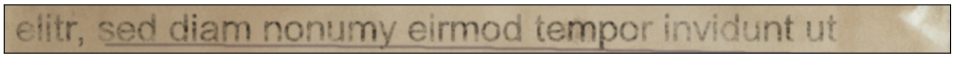
The aforementioned example serve as a representation of one sample that was tested. For further samples, we kindly direct you to the appendix B where additional samples are available for review.

OCR recognition:

The purpose of this section is to assess how well the models improve the accuracy of Tesseract OCR. The objective is to demonstrate how the models perform on different test samples and determine their ability to enhance Tesseract OCR's accuracy in extracting text using character error rate (CER) and word error rate (WER), which were introduced in Section 5.3.

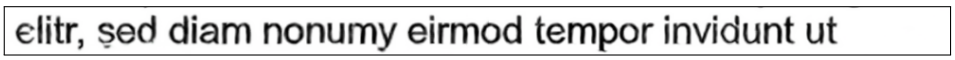
Sample: elitr, sed diam nonumy eirmod tempor invidunt ut

- **Source image:**



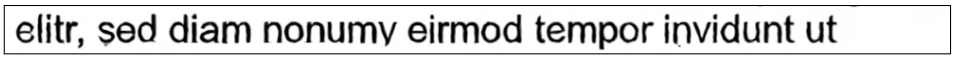
  Extracted text: nonum eirmod tem or invidunt ut
  Character Error Rate: 0.375
  Word Error Rate: 0.75

- **Pix2Pix with ResNet (With L1 Loss):**



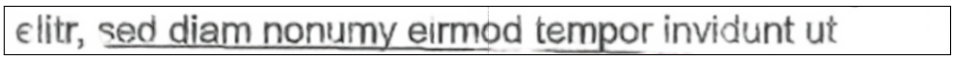
  Extracted text: elitr, sed diam nonumy eirmod tempor invidunt ut
  Character Error Rate: 0.0
  Word Error Rate: 0.0

- **Pix2Pix with ResNet (Without L1 Loss):**



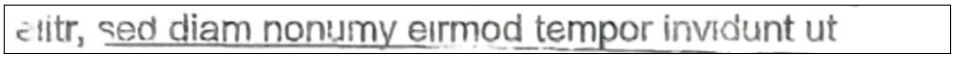
  Extracted text: elitr, sed diam nonumv eirmod tempor invidunt ut
  Character Error Rate: 0.0208
  Word Error Rate: 0.125

- **Pix2Pix with U-Net (With L1 Loss):**



  Extracted text: elitr, wed gigm nonumy emnod tempor invidunt ut 7
  Character Error Rate: 0.1667
  Word Error Rate: 0.5

- **Pix2Pix with U-Net (Without L1 Loss):**



  Extracted text: e titr, =ed giam nonumy e|rmqcl_tgmgor inv:du_r1t ut 1
  Character Error Rate: 0.3542
  Word Error Rate: 1.0

Similar to the example provided earlier, we computed the mean of Character Error Rate and Word Error Rate for 100 different sentences, and obtained the subsequent outcomes for each model:

Model	With L1 Loss	Mean CER ↓	Mean WER ↓
Source	-	0.6435	0.8125
Pix2Pix with ResNet	True	**0.0476**	**0.2374**
Pix2Pix with ResNet	False	0.0652	0.3014
Pix2Pix with U-Net	True	0.1124	0.4402
Pix2Pix with U-Net	False	0.1442	0.5070

Table A.10.: Test case 4 OCR recognition results table

Table A.10 presents the results of different Pix2Pix models with/without L1 Loss for improving OCR recognition. The performance of the models is measured by mean CER and mean WER, where lower values are better.

The first row of the table labeled "Source" represents the baseline performance without any improving. The subsequent rows represent the performance of different Pix2Pix models with ResNet and U-Net architectures, both with and without L1 Loss.

The results indicate that all the Pix2Pix models outperform the baseline performance, as evidenced by the lower CER and WER values. Among the models, the Pix2Pix with ResNet architecture and L1 Loss achieved the best performance with a CER of 0.0476 and a WER of 0.2374. On the other hand, the Pix2Pix with U-Net architecture and no L1 Loss achieved the worst performance with a CER of 0.1442 and a WER of 0.5070.

In general, adding L1 Loss to the Pix2Pix models appears to have a positive impact on OCR recognition, as the models with L1 Loss achieved better performance than those without it. The ResNet architecture also appears to be better suited for OCR recognition, as it outperforms the U-Net architecture in all cases.

B. Results Appendix

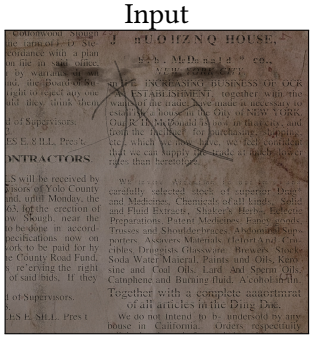
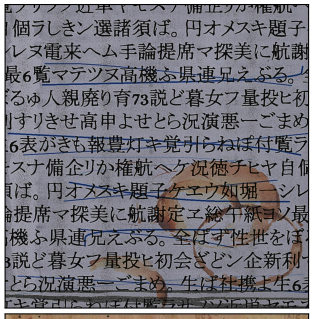
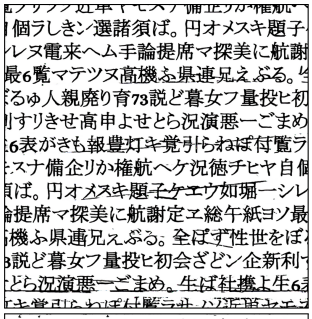
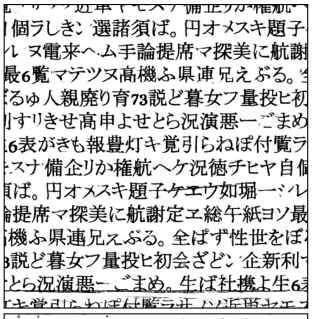
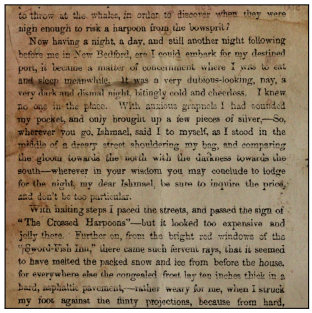
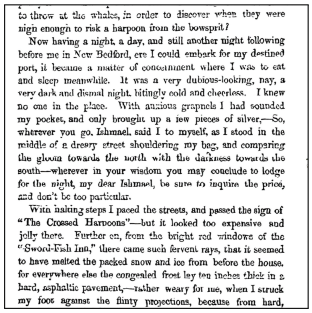
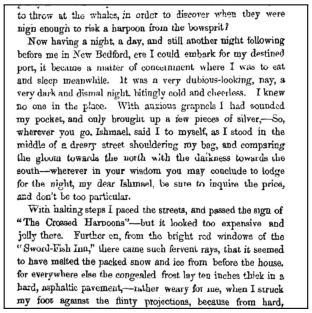
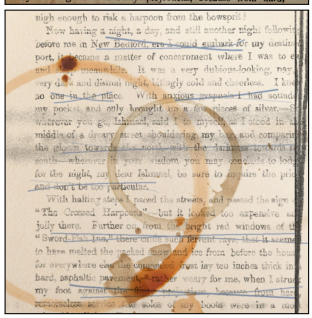
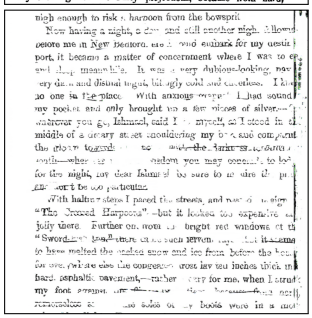
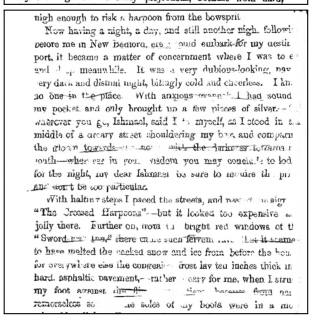
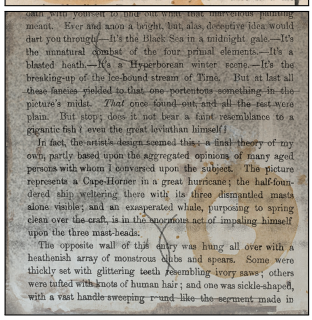
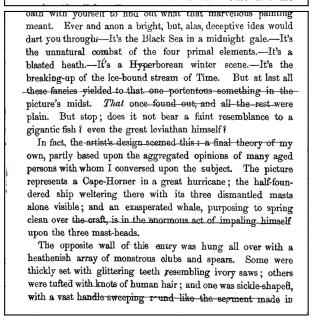
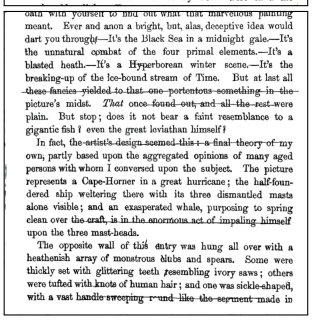
This appendix contains additional image inputs that were used for visual inspection as part of our research analysis. These images were collected to provide a more comprehensive understanding of the data and to ensure the accuracy of our results. Visual inspection is an important step in many research projects as it allows for the identification of potential errors or anomalies that may not be evident through statistical analysis alone. By including these additional images in our results appendix, we aim to provide a more transparent and thorough presentation of our research findings.

Test Case 1 and 2 Results

- Pix2Pix with ResNet:



- Pix2Pix with U-Net:

Input	With Batch Norm	With Instance Norm
		
		
		
		
		

- CycleGAN with ResNet:



- CycleGAN with U-Net:



Test Case 3 Results

- Pix2Pix with ResNet:



- Pix2Pix with U-Net:



- CycleGAN with ResNet:



- CycleGAN with U-Net:



Test Case 4 Results

- Pix2Pix with ResNet:



- Pix2Pix with U-Net:



- CycleGAN with ResNet:



- CycleGAN with U-Net:

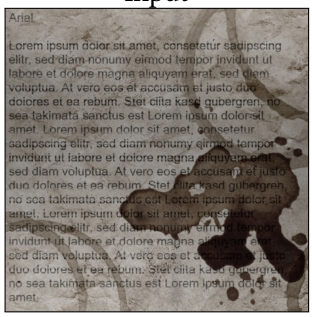
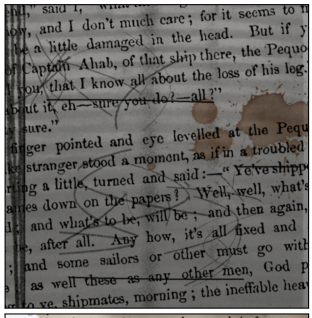
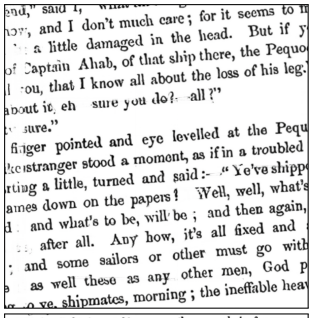
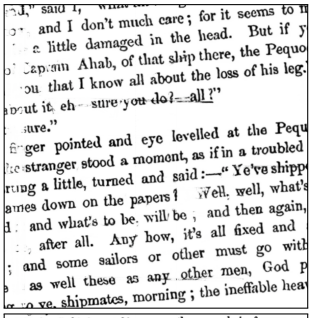
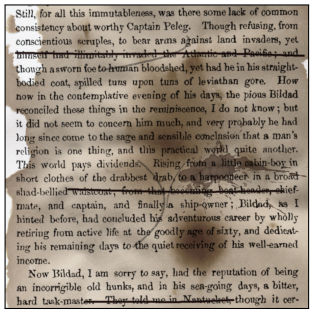
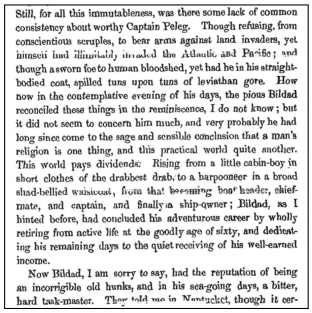
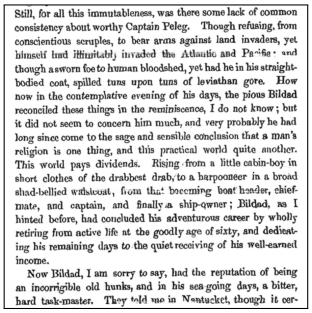
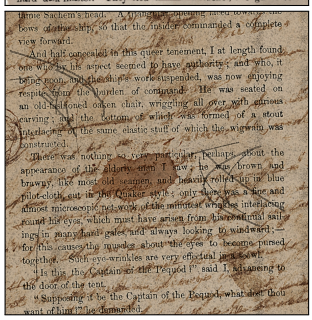
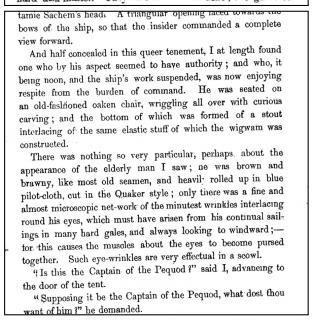
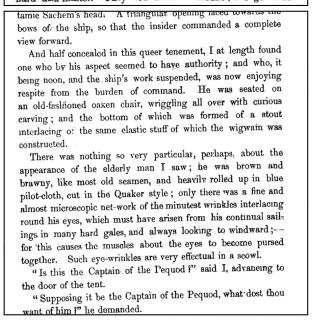
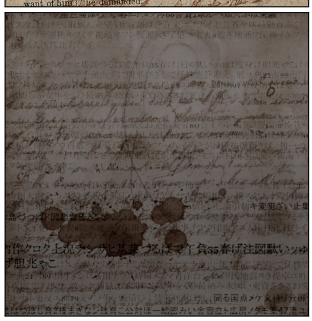
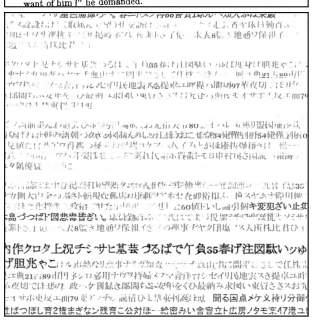
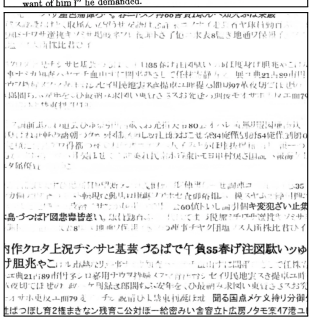


Test Case 5 Results

- Pix2Pix with ResNet:



- Pix2Pix with U-Net:

Input	Learning rate 0.0001	Learning rate 0.0002
		
		
		
		
		

- CycleGAN with ResNet:



- CycleGAN with U-Net:



Test Case 6 Results

- Pix2Pix with ResNet:



- Pix2Pix with U-Net:



C. Code Appendix

All the codes presented in the thesis appendix have undergone a refactoring process to enhance their readability. The aim of this was to ensure that any reader, regardless of their programming background, can easily follow and understand the code.

Pix2Pix Implementation

Pix2Pix Generator (U-Net)

```
class Generator(nn.Module):
  def __init__(self, in_channels, out_channels):
    super().__init__()
    # encoder
    encoder_layers = [
      # The first downsample-block with the specified parameters (128x128x64)
      Downsample_Block_1(in_channels=3,out_channels=64,kernel_size=4,stride=2,
      padding=1),
      # The second downsample-block with the specified parameters (64x64x128)
      Downsample_Block_2(in_channels=64,out_channels=128,kernel_size=4,
      stride=2,padding=1),
      # The third downsample-block with the specified parameters (32x32x256)
      Downsample_Block_3(in_channels=128,out_channels=256,kernel_size=4,
      stride=2,padding=1),
      # The fourth downsample-block with the specified parameters (16x16x512)
      Downsample_Block_4(in_channels=256,out_channels=512,kernel_size=4,
      stride=2,padding=1),
      # The fifth downsample-block with the specified parameters (8x8x512)
      Downsample_Block_5(in_channels=512,out_channels=512,kernel_size=4,
      stride=2,padding=1),
      # The sixth downsample-block with the specified parameters (4x4x512)
      Downsample_Block_6(in_channels=512,out_channels=512,kernel_size=4,
      stride=2,padding=1),
      # The seventh downsample-block with the specified parameters (2x2x512)
      Downsample_Block_7(in_channels=512,out_channels=512,kernel_size=4,
      stride=2,padding=1),
      # The eighth downsample-block with the specified parameters (1x1x512)
      Downsample_Block_7(in_channels=512,out_channels=512,kernel_size=4,
      stride=2,padding=1)]
```

```
    # decoder
    decoder_layers = [
      # The first upsample-block with the specified parameters (2x2x512)
      Upsample_Block_1(in_channels=512,out_channels=512,kernel_size=4,
      stride=2, padding=1),
      # The second upsample-block with the specified parameters (4x4x512)
      Upsample_Block_2(in_channels=1024,out_channels=512,kernel_size=4,
      stride=2, padding=1),
      # The third upsample-block with the specified parameters (8x8x512)
      Upsample_Block_3(in_channels=1024,out_channels=512,kernel_size=4,
      stride=2, padding=1),
      # The fourth upsample-block with the specified parameters (16x16x512)
      Upsample_Block_4(in_channels=1024,out_channels=512,kernel_size=4,
      stride=2, padding=1),
      # The fifth upsample-block with the specified parameters (32x32x256)
      Upsample_Block_5(in_channels=1024, out_channels=256,kernel_size=4,
      stride=2, padding=1),
      # The sixth upsample-block with the specified parameters (64x64x128)
      Upsample_Block_6(in_channels=512,out_channels=128,kernel_size=4,
      stride=2, padding=1),
      # The seventh upsample-block with the specified parameters (128x128x64)
      Upsample_Block_7(in_channels=256,out_channels=64,kernel_size=4,
      stride=2, padding=1),
      # The eighth upsample-block with the specified parameters (256x256x3)
      Upsample_Block_8(in_channels=64, out_channels=3,kernel_size=4,
      stride=2, padding=1)]

    self.encoder = nn.ModuleList(*encoder_layers)
    self.decoder = nn.ModuleList(*decoder_layers)

  def forward(self, x):
    skips_cons = []
    for encoder_block in self.encoder:
      x = encoder_block(x)
      skips_cons.append(x)
    skips_cons = list(reversed(skips_cons[:-1]))
    decoders = self.decoder[:-1]
    for decoder_block, skip in zip(decoders, skips_cons):
      x = decoder_block(x)
      x = torch.cat((x, skip), axis=1)
    return x
```

Listing C.1: Pix2Pix generator

Note that both downsample-block and upsample-block are detailed in section 4.2.4.1.

Pix2Pix Discriminator

```
class Discriminator(nn.Module):

  def __init__(self, in_channels, out_channels, n_blocks):
    super().__init__()
    discriminator_layers = []

    discriminator_layers.extend([
      # The first block with the specified parameters
      nn.Conv2d(in_channels=6,out_channels=64,kernel_size=4,stride=2,
      padding=1), nn.LeakyReLU(0.2, True),
      # The second block with the specified parameters
      nn.Conv2d(in_channels=64,out_channels=128,kernel_size=4,stride=2,
      padding=1),  nn.BatchNorm2d(128), nn.LeakyReLU(0.2, True),
      # The thierd block with the specified parameters
      nn.Conv2d(in_channels=128,out_channels=256,kernel_size=4,stride=2,
      padding=1),  nn.BatchNorm2d(256), nn.LeakyReLU(0.2, True),
      # The fourth block with the specified parameters
      nn.Conv2d(in_channels=256,out_channels=512,kernel_size=4,stride=1,
      padding=1),  nn.BatchNorm2d(512), nn.LeakyReLU(0.2, True),
      # The final block with the specified parameters
      nn.Conv2d(in_channels=512,out_channels=1,kernel_size=4,stride=1,
      padding=1)
    ])

    self.model = nn.Sequential(*discriminator_layers)

  def forward(self, input, target):
    x = torch.cat([input, target], axis=1)
    return self.model(x)

```

Listing C.2: Pix2Pix discriminator layers

CycleGAN Implementation

CycleGAN Generator (ResNet)

```
class Generator(nn.Module):

  def __init__(self, in_channels, out_channels, n_blocks):
    super().__init__()
    generator_layers = []
    # Add the encoder layers
    generator_layers.extend([
      # The first downsample-block with the specified parameters
      nn.ReflectionPad2d(3),
      nn.Conv2d(in_channels=3,out_channels=64,kernel_size=7,stride=1,
      padding=0), nn.InstanceNorm2d(64), nn.ReLU(True),
      # The second downsample-block with the specified parameters
      nn.Conv2d(in_channels=64,out_channels=128,kernel_size=3,stride=2,
      padding=1), nn.InstanceNorm2d(128), nn.ReLU(True),
      # The thierd downsample-block with the specified parameters
      nn.Conv2d(in_channels=128,out_channels=256,kernel_size=3,stride=2,
      padding=1), nn.InstanceNorm2d(256), nn.ReLU(True)])

    # The residual blocks
    for i in range(n_blocks):
      generator_layers.append(ResidualBlock(256,))

    # The decoder layers
    generator_layers.extend([
      # The first upsample-block with the specified parameters
      nn.ConvTranspose2d(in_channels=256,out_channels=128,kernel_size=3,
      stride=2,padding=1), nn.InstanceNorm2d(128), nn.ReLU(True),
      # The second upsample-block with the specified parameters
      nn.ConvTranspose2d(in_channels=128,out_channels=64,kernel_size=3,
      stride=2,padding=1), nn.InstanceNorm2d(64), nn.ReLU(True),
      # The final block with the specified parameters
      nn.ReflectionPad2d(3),
      nn.Conv2d(in_channels=64,out_channels=3,kernel_size=7,stride=1,
      padding=0), nn.Tanh()])

    self.model = nn.Sequential(*generator_layers)

  def forward(self, input):
    return self.model(input)
```

Listing C.3: CycleGAN generator

CycleGAN Discriminator

```
class Discriminator(nn.Module):

  def __init__(self, in_channels, out_channels, n_blocks):
    super().__init__()
    discriminator_layers = []

    discriminator_layers.extend([
      # The first block with the specified parameters
      nn.Conv2d(in_channels=3,out_channels=64,kernel_size=4,stride=2,
      padding=1), nn.LeakyReLU(0.2, True),
      # The second block with the specified parameters
      nn.Conv2d(in_channels=64,out_channels=128,kernel_size=4,stride=2,
      padding=1),  nn.InstanceNorm2d(128), nn.LeakyReLU(0.2, True),
      # The thierd block with the specified parameters
      nn.Conv2d(in_channels=128,out_channels=256,kernel_size=4,stride=2,
      padding=1),  nn.InstanceNorm2d(256), nn.LeakyReLU(0.2, True),
      # The fourth block with the specified parameters
      nn.Conv2d(in_channels=256,out_channels=512,kernel_size=4,stride=1,
      padding=1),  nn.InstanceNorm2d(512), nn.LeakyReLU(0.2, True),
      # The final block with the specified parameters
      nn.Conv2d(in_channels=512,out_channels=1,kernel_size=4,stride=1,
      padding=1)
    ])

    self.model = nn.Sequential(*discriminator_layers)

  def forward(self, input):
    return self.model(input)

```

Listing C.4: CycleGAN discriminator layers