山东理工大学

SHANDONG UNIVERSITY OF TECHNOLOGY

Graduation Thesis

The Research on Image Denoising Methods Based on The Deep Learning Theory

Institute: Shandong University of Technology

Major：Computer Science and Technology

Student's Name：Das Polash Chandra

Student ID：19812502010

Mentor (Instructor) Teacher：Dr. Wang Lei

2023.06.05

摘 要

图像去噪是图像处理和计算机视觉中的一项基本任务，旨在消除图像中的噪声并提高其图像质量。传统的去噪方法在捕捉复杂的噪声模式和保留精细细节方面有局限性。在这个项目中，我提出了用于图像去噪的高级深度学习方法: 自动编码器和生成对抗网络 (GAN)。我使用 MNIST 时尚数据集，其中包括时尚物品的灰度图像，来训练和评估我们的模型。

自动编码器方法使用的卷积神经网络设计具有编码器和解码器。自动编码器通过学习紧凑的潜在空间表示成功捕获详细的噪声模式并恢复去噪图像中的精细细节。诸如 RMSE、PSNR 和 SSIM 等评估措施显示了自动编码器在重建去噪图像方面的出色保真度和准确性。GAN 方法使用有条件的 GAN 框架，其中鉴别器网络告诉实际和人工创建的干净图像之间的区别，而生成器网络将嘈杂的图像映射到干净的图像。GAN 方法通过对抗性训练保留了高级特征和纹理，从而在视觉上令人愉悦的去噪图像。

该项目中提出的先进方法比传统技术具有显著优势。自动编码器擅长精确再现去噪图像，捕捉清晰的底层图像结构，并保持精细的细节。另一方面，GAN 方法通过利用对手训练过程和保持输入图像的整体美学来产生视觉上吸引人的去噪图像。这些用于去噪图像的先进的深度学习方法已经突破了降噪能力的界限，超越了以前方法的限制。通过捕获复杂的噪声模式，保留精细的细节，并产生视觉上令人愉悦的去噪图像，这些方法有助于图像去噪技术的进步。通过实现高质量的图像分析和解释，它们有潜力彻底改变各种应用，如医学成像、监测和计算机视觉任务。

关键词: 图像去噪，深度学习，自动编码器，生成对抗网络 (GAN)，降噪

Abstract

Image denoising is a fundamental task in image processing and computer vision, aimed at removing noise from images and improving their image quality. Traditional denoising methods have limitations in capturing complex noise patterns and preserving fine detail. In this project, I proposed advanced deep learning methods for denoising images: Autoencoder and Generative Adversarial Network (GAN). I used the MNIST Fashion dataset, which includes grayscale images of fashion items, to train and evaluate our models.

The convolutional neural network design used by the autoencoder method has an encoder and a decoder. The Autoencoder successfully catches detailed noise patterns and recovers fine details in the denoised images by learning a compact latent space representation. The assessment measures, such as RMSE, PSNR, and SSIM, show the Autoencoder's excellent level of fidelity and accuracy in recreating denoised images. The GAN approach uses a conditional GAN framework, where the discriminator network tells the difference between actual and artificially created clean images, while the generator network maps noisy images to clean ones. The GAN approach preserves high-level features and textures through adversarial training, resulting in visually pleasing denoised images.

The advanced methods presented in this project offer significant advantages over traditional techniques. Autoencoder excels at accurately reproducing denoised images, capturing sharp underlying image structures, and maintaining fine details. On the other hand, the GAN method produces visually appealing denoise images by taking advantage of the adversary training process and maintaining the overall aesthetics of the input images. These advanced deep-learning methods for denoising images have pushed the boundaries of noise reduction capabilities, beyond the limitations of previous methods. By capturing complex noise patterns, retaining fine details, and producing visually pleasing denoising images, these methods contribute to the advancement of image denoising techniques. They have the potential to revolutionize various applications, such as medical imaging, monitoring, and computer vision tasks, by enabling high-quality image analysis and interpretation.

Keywords: Image-Denoising, Deep Learning, Autoencoder, Generative Adversarial Network (GAN), Noise Reduction

Contents

List of Figures

List of Tables

Chapter 1 Introduction

The objective of image denoising, an essential process in computer vision and image processing, is to eliminate undesirable noise from digital images in order to improve their quality and improve visual perception. Image clarity and fidelity can be harmed by noise from a variety of causes, including sensor limits, compression artifacts, and environmental conditions. Therefore, the creation of efficient denoising methods is essential for applications like digital photography, surveillance systems, and medical imaging.

Deep learning has been a powerful method for image denoising in recent years due to its capacity to autonomously learn complex patterns and representations from huge datasets. This study focuses on Autoencoder and Generative Adversarial Network (GAN), two well-liked deep learning methods for image denoising. I have explained in details in chapter 3

In order to rebuild denoised images from this compressed representation, the Autoencoder uses a neural network design that learns to encode input images into a lower-dimensional latent space. The Autoencoder can efficiently learn to extract useful features and remove noise from the input images by being trained on a sizable dataset of noisy and clean image pairs.

GAN, on the other hand, is made up of an adversarial-trained discriminator network and a generator network. The discriminator seeks to discriminate between the created images and the clean images, whereas the generator seeks to produce realistic and denoised images. The adversarial training method enables GANs to produce high-quality denoised images that are aesthetically pleasing.

Different indicators can be used to assess the performance of the denoising process. Metrics like the Root Mean Squared Error (RMSE), Peak Signal-to-Noise Ratio (PSNR), and Structural Similarity Index (SSIM) can be used to evaluate the Autoencoder approach. On the other side, measures like L1 (Mean Absolute Error), SSIM, and PSNR can be used to assess the GAN approach.

This research attempts to provide light on the efficacy and applicability of each method for image denoising by evaluating how well the Autoencoder and GAN algorithms perform using these criteria. The study will also examine the benefits and drawbacks of each strategy, opening the way for more developments and enhancements in the area of deep learning-based image denoising.

Through this research, it is hoped to gain a greater understanding of image denoising using deep learning, which will aid in the creation of denoising models that are more reliable and precise. These developments could have a significant effect on a number of fields that depend largely on high-quality images, ultimately improving visual perception and enabling more precise analysis and decision-making.

1.1 Image Denoising

Image denoising is an advanced technique that improves the visual quality and accuracy of subsequent analysis by removing undesirable noise from digital images.

Noise can appear as random changes or distortions in the image and can be caused by a number of factors, including sensor limits, low light levels, or transmission faults.

Image denoising aims to minimize or remove noise preserving the image's underlying information and structures intact. It entails separating the required image signal from the noise and using the proper algorithms or techniques to suppress or eliminate the noise components. Figure 1 example of denoising image.

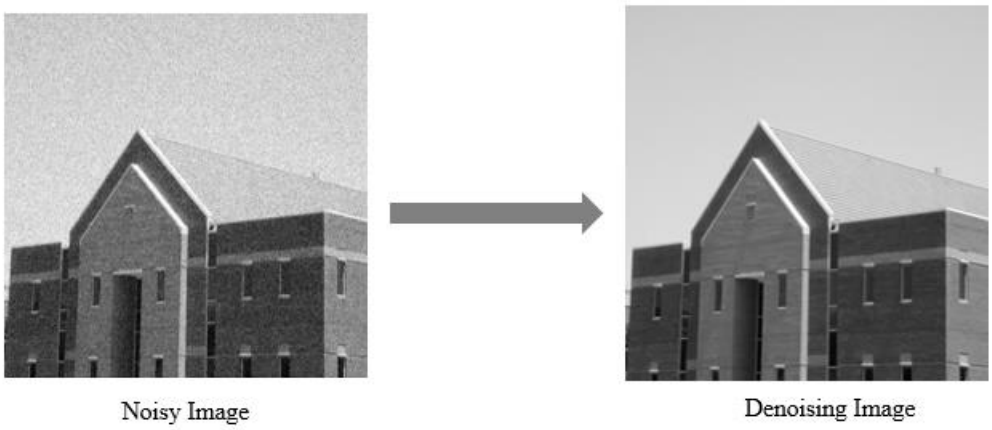

Figure 1. Example of Denoising Image

Numerous sophisticated methods are used by advanced image-denoising systems, including statistical modeling, transform-based techniques, and machine learning algorithms. To recognize and distinguish the noise from the actual image content, these algorithms take advantage of the inherent qualities of images and the relationships between nearby pixels.

In traditional denoising techniques, the image may be smoothed and the noise reduced using filters such as median or Gaussian filters. However, these filters can also blur the image's vital borders and details.

On the other hand, modern machine learning-based denoising approaches, including deep neural networks, promise significant improvements. These techniques can efficiently distinguish between noise and images by learning directly from a collection of clean and noisy images. The denoising network may learn to generalize and create denoised images that preserve fine details and structures while efficiently suppressing the noise by being trained on a large number of samples.

For instance, a deep learning-based denoising system can learn the underlying patterns and textures present in the clean images by training on a dataset of low-light images with matching clean versions. The sharpness of edges, delicate textures, and other crucial aspects of the image can thus all be properly maintained while the noise is effectively removed.

Advanced image-denoising algorithms are used across many different sectors. Denoising algorithms can enhance the quality of MRI or CT scans in medical imaging, allowing for improved diagnosis and treatment planning. Denoising can improve the visibility of important information in noisy or low-quality video footage for

surveillance applications. Denoising algorithms in digital photography can lower noise in images taken in difficult lighting situations, producing clearer and more aesthetically pleasing images.

In conclusion, sophisticated algorithms, statistical models, and machine-learning techniques are used in modern image-denoising techniques to reduce noise while keeping crucial image information. These methods are widely used in many different fields and are essential for enhancing image quality, enabling precise analysis, and enhancing visual perception.

1.2 Deep Learning for Image Denoising

Deep learning techniques for image denoising are becoming more and more common and are performing remarkably well. Autoencoder, which are neural network topologies made up of an encoder and a decoder, is one of the most used methods. The decoder reconstructs the denoised image from this compressed representation after the encoder maps the input image into a lower-dimensional latent space. Autoencoder can efficiently extract useful features and reduce noise by being trained on pairs of noisy and clean images [1].

Digital image noise reduction using deep learning algorithms is an example of CNN-based image denoising. It uses the convolutional layer's ability to automatically recognize and extract important characteristics from noisy images. CNN-based self-supervised learning is a particular deep learning strategy that makes use of CNN to learn denoising patterns from unlabeled data [2]. CNN for image denoising is a deep learning model that efficiently removes noise in images while keeping crucial details and structures.

Here is a quick breakdown of how CNN image denoising:

I. Data Preparation: Creating a dataset of noisy images and matching clean equivalents is the first step. The CNN model is trained using these clear images as the source data. The dataset needs to be varied and illustrative of the various kinds of noise that can be found in the images.
II. Model Architecture: The CNN model used for denoising typically starts with a number of convolutional layers, followed by activation functions (such as ReLU), and potentially other layer types like pooling or batch normalization layers. The intricacy of the denoising process will determine how many layers there are and how they are configured.
III. Training: The CNN model is trained using the ready dataset. The model develops the ability to reduce the difference between the denoised images it produces and the equivalent clean images in the dataset during training. This is accomplished by utilizing an optimization technique (such as stochastic gradient descent) to optimize a selected loss function, such as mean squared error (MSE).
IV. Inference: Once trained, the CNN model can be used to denoise fresh, new images. The trained model receives the noisy images and feeds them through its

layers to provide denoised images as an output.

V. Evaluation: A variety of criteria can be utilized to evaluate the quality of the CNN model's denoised images. Peak signal-to-noise ratio (PSNR) and structural similarity index (SSIM), two common evaluation metrics, evaluate how similar denoised and clean images are to one another.

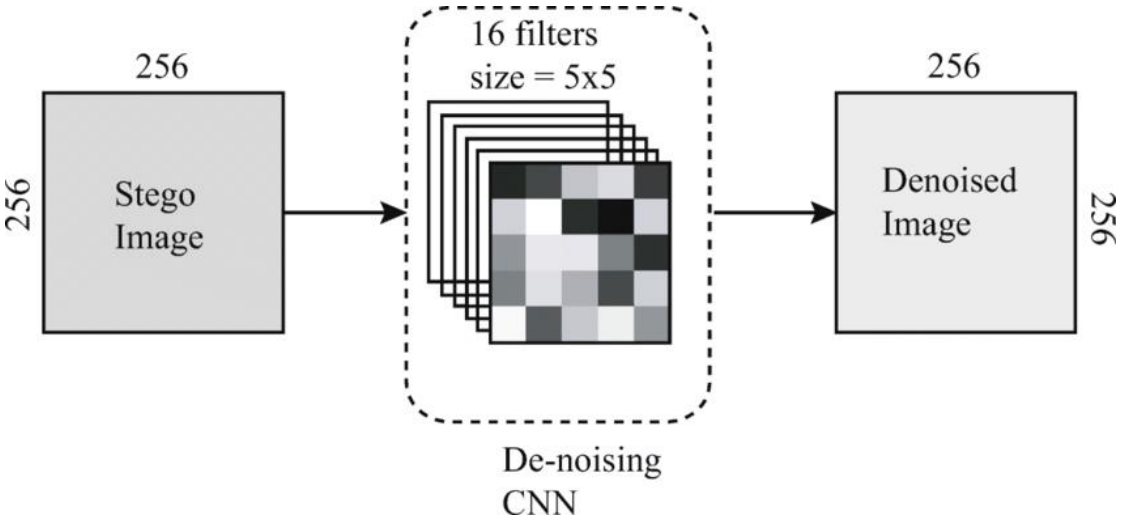

Figure 2. Architecture of denoising CNN(DCNN)

It's essential to keep in that CNN image denoising needs a sizable amount of labeled training data in order to generalize properly and denoise a variety of images. The performance of denoising can also be affected by architectural decisions and hyperparameter tunings, such as the number of layers, filter sizes, and activation functions.

In general, CNN image denoising provides an automated and data-driven method for removing noise from images, enabling improved visual quality and analysis in a variety of fields, such as photography, medical imaging, and computer vision applications.

The application of Generative Adversarial Networks (GANs) for image denoising is another potent deep-learning technique. GAN is made up of an adversarial generation network and a discriminator network. The discriminator seeks to discriminate between the created images and the clean images, whereas the generator seeks to produce denoised images. GAN may create beautiful and high-quality denoised images using this adversarial training method [3].

In order to assess the effectiveness of image denoising, it is customary to use numerous metrics to compare the denoised and matching clean images. Root Mean Squared Error (RMSE), Peak Signal-to-Noise Ratio (PSNR), Structural Similarity Index (SSIM), and Mean Absolute Error (MAE) are examples of often used metrics. These metrics give numerical measurements of how comparable the denoised and clean images are, allowing the effectiveness of the denoising process to be evaluated [4] [5]. Researchers can learn more about the efficacy and applicability of each strategy for image-denoising tasks by comparing the performance of autoencoder and GAN using these criteria. This comparison makes it easier to comprehend the benefits and

drawbacks of each approach and paves the way for further development of deep learning-based image-denoising approaches.

Deep learning techniques have the potential to have a significant influence on domains that depend on high-quality images by enabling the construction of reliable and precise image-denoising models. Visual perception, analysis accuracy, and decision-making processes can all be improved in applications including medical imaging, surveillance systems, and digital photography.

1.3 Motivation and Objective

The presented research uses the MNIST Fashion dataset to investigate and assess the use of deep learning techniques for image denoising. In particular, the effectiveness of the Autoencoder and the GAN denoising techniques will be examined. These methods were chosen because earlier research has shown them to be successful and have the potential for denoising work.

It is possible to learn a lot about how well the Autoencoder and GAN approaches work for image denoising by putting them to use and evaluating how well they function. The goal of the research is to examine the advantages and disadvantages of each method in terms of denoising effectiveness, computational efficiency, and robustness to various types of noise. This comparison will offer suggestions for deciding which approach is best for certain denoising tasks.

The research also attempts to assess the denoising outcomes using several metrics. Metrics like Root Mean Squared Error (RMSE), Peak Signal-to-Noise Ratio (PSNR), and Structural Similarity Index (SSIM) will be used for the Autoencoder approach. L1 (Mean Absolute Error), SSIM, and PSNR metrics will be used to gauge the effectiveness of the GAN approach. These metrics will offer numerical evaluations of the denoising quality and enable a thorough comparison of the two approaches.

The analysis of the denoised images will also shed light on the performance variations between the Autoencoder and GAN approaches. It will make it easier to pinpoint each technique's advantages and disadvantages with regard to retaining image details, minimizing noise artifacts, and keeping overall image quality. The results of this research can aid in the development of deep learning-based image-denoising algorithms and serve as a reference for choosing and creating denoising approaches for real-world use.

The goal of this research is to investigate, contrast, and assess the performance of Autoencoder and GAN approaches using the MNIST Fashion dataset in order to answer the demand for efficient image-denoising algorithms. The results of the research will progress the field of image denoising and offer important information for developing deep learning-based denoising systems in the future.

Chapter 2 Literature Review

2.1 Related Work for Image Denoising

2.1.1 The Mean Filtering

This approach replaces each pixel in an image with the average value of its neighboring pixels [6]. Mean filtering is a straightforward and computationally effective technique; however, it may blur images and lose visual details. In very impulsive noise, the typical median filter's strong performance is significantly limited. The examined technique has a straightforward and efficient noise cancellation algorithm that works across a wide range of noise densities from 10% to 98% while preserving excellent image quality [7].

In a rectangular sub-image window (neighborhood) of size m × n centered at a point (x, y), let S_{xy} represent the collection of coordinates. The average value of the corrupted picture, g(x, y), in the S_{xy}-defined region, is determined via the arithmetic mean filter. The arithmetic means calculated using the pixels in the area covered by S_{xy} serves as the value of the restored picture $\hat{f}$ at the point (x, y). Otherwise put,

$$\hat{f}(x,y) = \frac{1}{mn} \sum_{(s,t)\in S_{xy}} g(s,t) \tag{1}$$

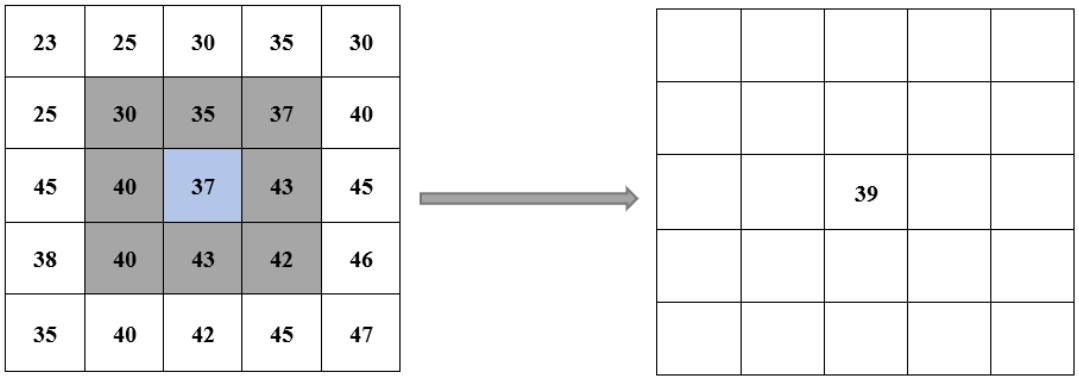

Figure 3. 3x3 Mean Filtering

A spatial filter of size m × n with all coefficients having values of 1/mn can be used to execute this operation. Local fluctuations in an image are smoothed by a mean filter, and blurring lowers noise.

2.1.2 The Gaussian Filtering

A common technique for image denoising is Gaussian filtering. The image is given a Gaussian kernel, which efficiently reduces noise while maintaining the integrity

of the overall image structure [8]. Convolution is used to apply the Gaussian filter, which is based on a Gaussian distribution.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (2)$$

The Gaussian filtering involves convolving the input image with a Gaussian kernel. The kernel is defined by the equation above, where g(x, y) represents the value of the Gaussian kernel at position (x, y). The σ (sigma) parameter controls the standard deviation of the Gaussian distribution, determining the amount of smoothing applied to the image.

Define the size of the Gaussian kernel, often represented by a square matrix with odd dimensions (e.g., 3x3, 5x5, etc.). The size determines the extent of the smoothing effect. Calculate the values of the Gaussian kernel based on the given σ value. For each position (x, y) in the kernel matrix, substitute the corresponding x and y coordinates into the Gaussian function equation to determine the kernel value. Divide each element in the Gaussian kernel by the total number of elements to normalize the kernel. By doing this, the kernel is guaranteed to maintain a unit sum, maintaining the overall intensity of the image. Create a convolution between the input image and the Gaussian kernel. In convolution, the kernel is moved over each pixel in the image, the associated elements are multiplied, and then the result is added. The filtered pixel is represented by the resultant value. Consider the size of the kernel and the bounds of the image as you do the convolution procedure for each pixel in the image. The result of this technique is a denoised image, which successfully smooths out the noise.

2.1.3 The Non-local Means Denoising

The Non-local Means (NLM) denoising is a popular and effective method for reducing noise in digital images. It was introduced by Buades, Coll, and Morel in 2005 [9]. The NLM algorithm takes advantage of the redundancy of image patches to estimate the denoised pixel values.

The main idea behind NLM denoising is that similar patches in an image tend to have similar pixel values. The algorithm computes a weighted average of similar patches to estimate the denoised value for each pixel. The similarity between patches is measured using a distance metric, typically based on pixel intensity differences. NLM denoising has several advantages. It can effectively reduce Gaussian noise while preserving image details and textures. It does not assume specific noise statistics, making it applicable to a wide range of noise types. Moreover, NLM denoising can handle non-stationary noise and is robust to outliers. However, NLM denoising has some limitations. It can be computationally expensive, especially for large images, due to the need to compare patches. The performance of NLM also depends on the choice of parameters, such as the patch size and the neighborhood window, which need to be carefully tuned. NLM denoising has been widely used in various applications, including medical imaging, photography, and video processing, where preserving image details and textures is crucial.

Each of these approaches has its strengths and weaknesses and is suited for

different types of noise and image characteristics. Deep learning-based approaches, in particular, have shown remarkable performance in denoising tasks, surpassing the traditional methods in many cases.

2.1.4 The Wavelet Denoising

Wavelet denoising utilizes the multi-resolution property of wavelet transforms to remove noise from images [10]. Wavelets are mathematical functions that may represent signals in both the time and frequency domains. Wavelet denoising is a technique for reducing noise in signals or images by taking advantage of these qualities. Wavelet denoising is very good at keeping edges and details sharp while cutting down on unnecessary noise.

The following steps are involved in wavelet denoising:

I. Decomposition: Wavelet transform is used to decompose the original signal into wavelet coefficients. In order to capture both high-frequency and low-frequency components, this transform separates the signal into several frequency bands.

II. Thresholding: In this step, the wavelet coefficients are thresholder to separate the significant coefficients from the noise. The idea is to suppress the coefficients associated with noise while preserving the coefficients that represent the signal. It is designed based on the principles of wavelet-based methods, therefore leading to a model with high interpretability [11].

    There are different types of thresholding techniques commonly used in wavelet denoising. Some popular methods include:

    1) Coefficients below a specific threshold are set to zero, thus eliminating them, using the hard thresholding technique.
    2) Coefficients below the threshold are reduced towards zero, lowering their magnitude, in a process known as soft thresholding.
    3) Through adaptive threshold optimization, the Stein's Unbiased Risk Estimate (SURE) approach calculates the mean squared error between the original signal and the denoised signal.

III. Reconstruction: The wavelet coefficients are thresholder, and the changed coefficients are then applied to reconstitute the denoised signal. The signal is recreated in the time domain using an inverse wavelet transform that mixes the adjusted coefficients.

The outcomes of denoising can be significantly influenced by the wavelet and thresholding methods that are used. Different wavelets have various smoothness and frequency localization characteristics. The thresholding technique also has an impact on the compromise between signal retention and noise reduction.

Wavelet denoising is used in a variety of industries, such as data compression, audio signal denoising, and image processing. It is an invaluable tool in many real-world

situations because it can successfully reduce noise while preserving crucial signal properties.

2.1.5 The Bilateral Filtering

Bilateral filtering is a non-linear filter that preserves edges while denoising the image [12]. It applies a weighted average of nearby pixels based on the closeness and similarity of their spatial and intensity properties. It might not, however, be able to properly eliminate noise while maintaining fine details.

2.1.6 The Total Variation Denoising

Total variation denoising is a variational method that minimizes the total variation of the image to remove noise [13]. It has attracted considerable attention in 1-D and 2-D signal processing [14]. It effectively reduces noise while maintaining edges and sharp transitions. However, it can cause the image's textures and features to be overly smoothed down. Figure 4 shows Total Variation Denoising.

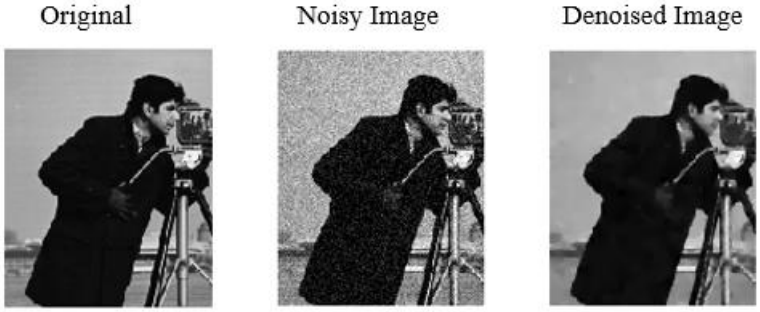

Figure 4. Total Variation Denoising

The total variation term supports piecewise constant regions in the denoised image, successfully lowering noise while maintaining sharp edges. The fidelity term makes sure that the denoised image stays close to the observed noisy image. The lambda parameter modifies how these two goals are balanced.

Numerous numerical strategies, including gradient descent, primal-dual algorithms, and split Bregman methods, can be used to tackle this optimization problem. In order to reduce overall variation while keeping image integrity, these methods iteratively update the denoised image.

Total variation denoising is frequently utilized in industries like computer vision and medical imaging because it has successfully been used in a variety of image-denoising applications. It offers a potent noise reduction tool while retaining crucial image structures.

2.1.7 The Patch-Based Denoising

In order to estimate the denoised values of each patch, patch-based denoising

algorithms divide the image into small patches and use patches that are comparable to one another. These techniques efficiently reduce noise by taking advantage of the self-similarity and redundancy seen in natural images. Denoising algorithms can determine the genuine underlying signal and decrease the noise by comparing patches inside the image. Patch-based denoising has shown excellent performance in handling different types of noise and preserving fine details in the image [15].

2.1.8 The Adaptive Filtering

Based on the features of the image and noise, adaptive filtering algorithms update the filter settings accordingly. These techniques provide better noise removal because they assess local statistics and modify the denoising procedure accordingly. Common adaptive filtering algorithms include the Wiener filter, the Least Mean Squares (LMS) filter, and the Recursive Least Squares (RLS) filter [16]. Adaptive filtering techniques are appropriate for a variety of denoising applications because they can improve image details while successfully suppressing noise.

2.1.9 The Median Filtering

A non-linear filter called median filter replaces each pixel with the neighborhood median value. Salt-and-pepper noise, which manifests as isolated pixels with maximum or minimum intensity values, is particularly well-affected by it. Median filtering significantly reduces impulsive noise while maintaining image edges and details. However, it may not be as effective in reducing other types of noise or maintaining fine textures in the image [17].

2.1.10 The Low-Rank Matrix Completion

Using the presumption that the underlying noise-free image has a low-rank structure, low-rank matrix completion techniques are used. These techniques are designed to restore the low-rank image matrix from a damaged copy. Low-rank matrix completion methods can successfully separate the noise from the clean image by describing the noise as sparse mistakes. This approach has shown promising results in denoising images corrupted by random noise or compressive sensing measurements [18].

2.2 Limitations

1. Mean Filtering: Mean filtering can effectively reduce noise, but it tends to blur the image and may not preserve fine details and edges.
2. Gaussian Filtering: Gaussian filtering is a linear smoothing technique. While it can reduce noise, it may also blur the image and cause a loss of sharpness.
3. Non-local Means Denoising: Non-local means denoising is effective in preserving image details and textures. However, it can be computationally

intensive and may not perform well on images with complex textures or strong noise.

4. Wavelet Denoising: Wavelet denoising can effectively reduce noise while preserving image details. However, it may introduce artifacts known as "ringing" near edges or discontinuities in the image.
5. Bilateral Filtering: Bilateral filtering is good at preserving edges and details while reducing noise. However, it may lead to over-smoothing in regions with rapid intensity variations or complex textures.
6. Total Variation Denoising: Total variation denoising is effective in preserving edges while reducing noise. However, it can lead to a loss of fine details and textures in the image.
7. Patch-Based Denoising: Patch-based denoising methods are effective in preserving textures and details. However, they can be computationally expensive, especially for large images.
8. Adaptive Filtering: Adaptive filtering techniques adjust the denoising parameters based on local image characteristics. However, they may struggle with adaptive noise patterns or in regions with low texture content.
9. Median Filtering: Median filtering is good at preserving edges while reducing noise. However, it may not be effective in reducing certain types of noise, such as Gaussian noise, and can introduce blurring in texture-rich regions.
10. Low-Rank Matrix Completion: Low-rank matrix completion methods exploit the inherent low-rank structure in the image for denoising. However, they may struggle with complex noise patterns or high-frequency noise.

2.2 Deep Learning-based Image Denoising Techniques

Deep learning-based image denoising techniques leverage the power of deep neural networks to reduce noise in images. These methods have demonstrated remarkable performance in restoring images corrupted by various types of noise. Here are some popular techniques:

2.2.1 The Convolutional Neural Networks (CNN):

There is a lot of usage for CNN in image-denoising activities. They typically start with fully connected layers, followed by many layers of convolutional and pooling procedures. In order to provide denoised outputs, CNN learn to extract pertinent characteristics from noisy images. CNN can be used for more than just generic image denoising; for example, it performed well for blind denoising [19], actual noisy images [20], and many other tasks. Even though many academics have created CNN techniques

for image denoising, very few have suggested a review to compile the techniques. In reference [21], classifications based on the type of noise were used to summarize CNN approaches for image denoising. For noise reduction tasks, CNN architecture can be tailored to produce patterns that bypass the vanishing gradient bottleneck [22].

Popular CNN architectures for image denoising include DnCNN, U-Net, REDNet, MemNet, FFDNet, and RIDNet. Remainder connections are used by DnCNN, skip connections by U-Net, encoder-decoder structures with residual connections by REDNet, memory efficiency is addressed by MemNet, a noise estimation module is included in FFDNet, and residual learning and dense connections are combined in RIDNet. Figure 5 shown CNN architecture for image denoising.

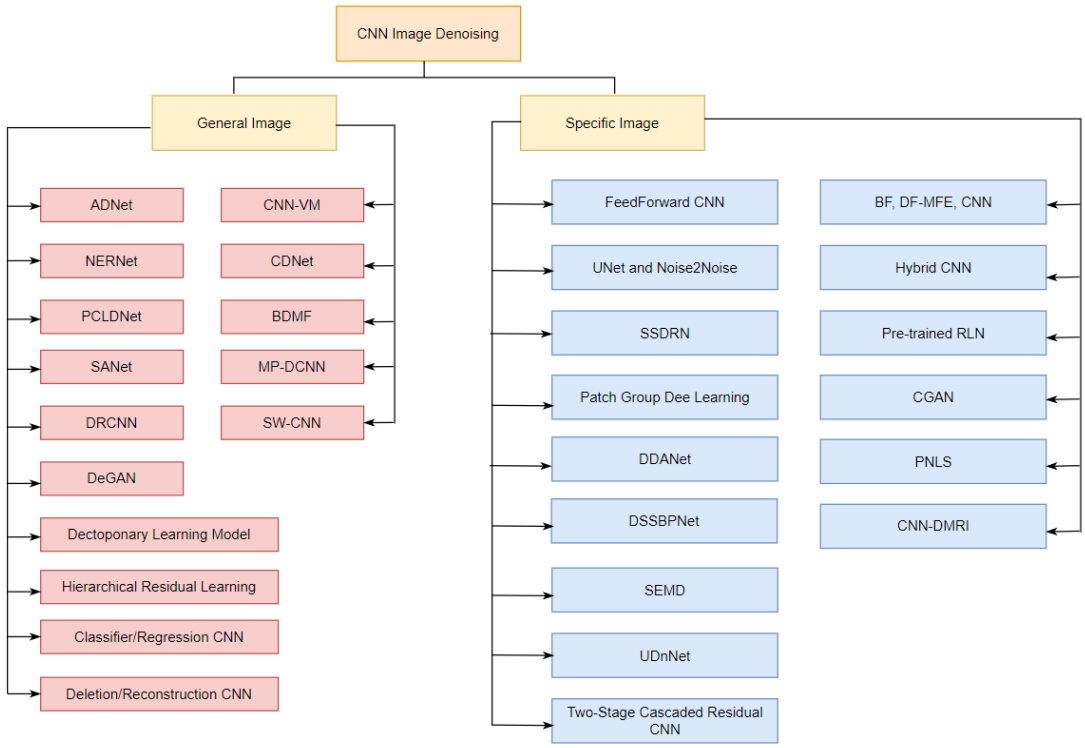

Figure 5. CNN architecture for image denoising

By processing new, unseen noisy images through its layers and producing denoised images as an output, the CNN can be used to denoise them after being trained.

2.2.2 The Autoencoder:

Autoencoder are neural networks that are trained to reconstruct their input data. In the context of image denoising, a Autoencoder is trained to reconstruct clean images from noisy inputs. By imposing a bottleneck structure in the network, the Autoencoder learns to extract essential features while eliminating noise. By training an Autoencoder on pairs of noisy and clean images, it can effectively learn to extract meaningful features from the noisy input and reconstruct the clean image from these features [23]. Although they can recognize intricate patterns and structures in the data, Autoencoders have demonstrated promising results in image-denoising applications. I have explained

in details in chapter 3.

2.2.3 The Variational Autoencoder (VAE):

VAE are generative models that learn to encode and decode images. They are trained on pairs of noisy and clean images, aiming to generate clean images from noisy inputs. VAE incorporate probabilistic models and latent variables, enabling them to capture the underlying distribution of clean images.

2.2.4 The Residual Learning:

Residual learning-based denoising networks employ residual connections, which allow the network to learn residual mappings between noisy and clean images. By explicitly modeling the residual information, these networks can effectively remove noise while preserving image details.

2.2.5 The Generative Adversarial Networks (GAN):

Generator and discriminator networks make up the GAN. In image denoising, the generator is taught to create clear images from noisy inputs, while the discriminator is taught to tell generated clear images from actual clear images. GAN can capture complex image distributions and generate visually appealing denoised images [24]. The training process encourages the generator to produce high-quality denoised outputs.

These techniques have significantly advanced image-denoising capabilities, producing visually pleasing results with reduced noise artifacts. They continue to evolve, incorporating novel architectures, loss functions, and training strategies to further enhance denoising performance.

Among the five mentioned techniques, I have utilized the Autoencoder and GAN methods for image denoising. Both approaches have proven to be effective in reducing noise in images. The Autoencoder method is a type of neural network that learns to reconstruct clean images from noisy inputs. It has a bottleneck structure that allows it to capture essential features while eliminating noise. By training an Autoencoder on pairs of noisy and clean images, it can learn to denoise new input images effectively. On the other hand, GAN offers a different approach to image denoising. They consist of a generator network and a discriminator network. The generator is trained to produce clean images from noisy inputs, while the discriminator learns to distinguish between generated clean images and real clean images. This adversarial training process encourages the generator to produce high-quality denoised outputs.

By combining the strengths of Autoencoders and GAN, we can benefit from the reconstruction capabilities of Autoencoders and the generative power of GAN. The Autoencoder can handle denoising on a pixel level, while the GAN can provide additional refinement and generate visually appealing results. Implementing these two techniques allows us to leverage the advantages of both methods and achieve excellent image-denoising performance. I have explained in details in chapter 3.

2.3 Advantages of Autoencoder and GAN Methods

Both Autoencoder and Generative GAN can be used for image denoising, and each approach has its own advantages.

2.3.1 The Autoencoder:

Autoencoder can learn directly from unlabeled data, making them suitable for denoising tasks where obtaining clean images for training purposes may be challenging or expensive. Autoencoder aims to reconstruct clean images from noisy inputs. They are designed to minimize the difference between the input and output images, which can result in high-quality reconstructions. Autoencoders typically consist of an encoder and a decoder, making them relatively simple and straightforward to implement. The encoder encodes the input image into a lower-dimensional latent space, while the decoder reconstructs the image from this latent representation. Once trained, Autoencoder can denoise images quickly during inference. The process involves passing the noisy image through the encoder and decoder, without the need for iterative optimization steps [23].

2.3.2 The Generative Adversarial Networks (GAN):

GAN are known for their ability to generate realistic and visually appealing images. When used for denoising, GAN can produce visually pleasing results that closely resemble clean images. GAN employs a generator and a discriminator that play a game against each other. The generator aims to generate realistic images, while the discriminator learns to distinguish between real and generated images. This adversarial training process can lead to better noise removal by focusing on perceptually meaningful features. GAN can capture complex data distributions, allowing them to model intricate image structures and textures. This capability enables GAN to generate visually coherent and detailed denoised images. GAN can also be used as data augmentation tool for denoising tasks. By generating additional synthetic noisy images, GAN can increase the diversity of the training dataset, leading to improved generalization and robustness.

Both Autoencoders and GAN have their own strengths, and the choice between them depends on the specific requirements of the denoising task and the available data. Autoencoders are more straightforward and computationally efficient, while GAN excel in generating visually appealing and realistic denoised images [25].

Chapter 3 Experimental Methods

3.1 The Autoencoder

In the area of unsupervised learning, autoencoders have a long history and have been used for a variety of tasks, such as facial recognition, generative modeling, and image categorization. In order to solve the issue of training neural networks without explicit instructor signals, the term "Autoencoder" was first used in the 1980s. Autoencoders allow for a self-supervised learning strategy by using the input data as the anticipated output. The basic goal of autoencoders is to learn an encoding or representation of the input data in a lower-dimensional latent space. An encoding network that maps the input data to the latent space produces this encoding. The original input is then recreated from the encoded representation by a decoding network. The Autoencoder gains the ability to efficiently compress and reconstruct the input data during training, while also disregarding any noise or discrepancies that may be present. The mapping learned by the Autoencoder allows for various applications. For instance, in image classification, the latent space representation can capture important features and patterns in the data, enabling efficient classification algorithms. Autoencoders also play a crucial role in generative modeling, where they can generate new samples that resemble the input data distribution. Additionally, they have been used in facial detection tasks, among others, to extract discriminative features for face recognition or facial expression analysis. Overall, Autoencoders serve as a versatile tool in unsupervised learning, providing a means to learn compact representations of input data and facilitating various downstream tasks in the field of machine learning and computer vision.

3.1.1 Architecture of Autoencoder

An Autoencoder typically has two components [26]:

1. An encoder that maps the input into the latent space and
2. A decoder that reconstructs the input by decoding points on the latent space.

An Autoencoder is designed to deconstruct and reassemble the input rather than just copying it verbatim. As a result, it learns to recognize only the most crucial portions of the material. Figure 6 depicts a general Autoencoder schematic.

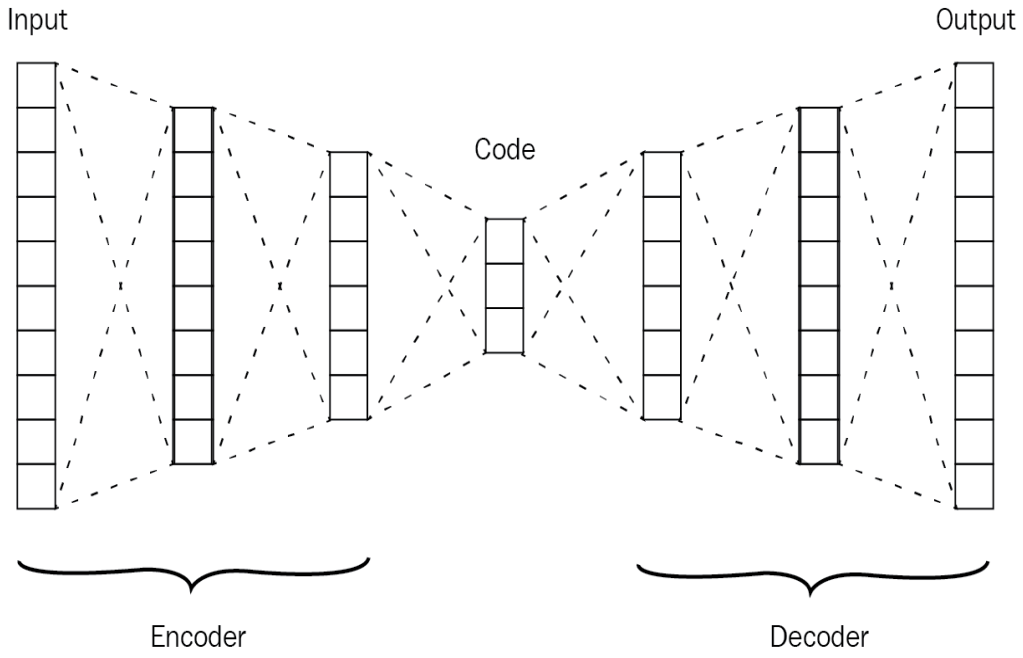

Figure 6. Autoencoder Architecture

Let's define the Autoencoder's encoder and decoder as transitions Φ and ψ respectively.

Using an input of X

$$\Phi : X \rightarrow F$$

$$\psi : F \rightarrow X \tag{3}$$

$$\Phi, \psi = \operatorname{argmin}_{\Phi,\psi} ||X - (\Phi \circ \psi)X||^2$$

When there is only one hidden layer, the encoder makes an input $x \in \mathbb{R}^p =$ and makes a mapping to $h \in \mathbb{R}^p = F$

$$h = \sigma(Wx + b) \tag{4}$$

The weight matrix W, the bias vector b, the rectified linear unit activation function, and the coded latent expression of the input are all denoted by the letter h. The bias vector and weight matrix of a neural network are normally chosen at random, and during training, they undergo backpropagation value morphing. The decoder then converts h to a reconstruction of x designated as x′, which has the same dimensions as x

$$x' = \sigma'(W'h + b') \tag{5}$$

where W′ is the decoder's weight matrix, b′ is its bias, and σ′ is the rectified

linear unit activation function. The encoder's W and b and the decoder's W and b frequently have no connection.

The difference between the input and output in terms of individual pixel values is used to determine the Autoencoder's loss. The mean squared error loss is one illustration.

$$L(x, \boldsymbol{x}') = ||x - x'||^2 = ||x - \sigma' (W' (\sigma(Wx+b)) + b'||^2 \quad (6)$$

where x represents the mean of each training batch.

The design of an Autoencoder [6] is relatively similar to that of a regular Autoencoder, with the exception that during training and validation, it adds noise to the input images contained in the dataset. Random noise has been introduced. When an input x is supplied to a function T that generates random noise, T(x) transforms the input into a noisy version of x. The neural network is then fed with this T(x) as input.

$$T : X \rightarrow T(X) \quad (7)$$

Any noisy image must be processed by an autoencoder in order to remove the noise while reconstructing the original image. Convolutional layers in the neural network should be used instead of dense layers while developing the model. Convolutional layers are used because they are known to learn extraordinarily good representations and features of images from their operation, in contrast to fully connected layers.

3.1.2 Model Architecture

Below is a description of the architecture created for both a fully linked Autoencoder and a convolutional Autoencoder. Figure 7 shows Fully connected Autoencoder

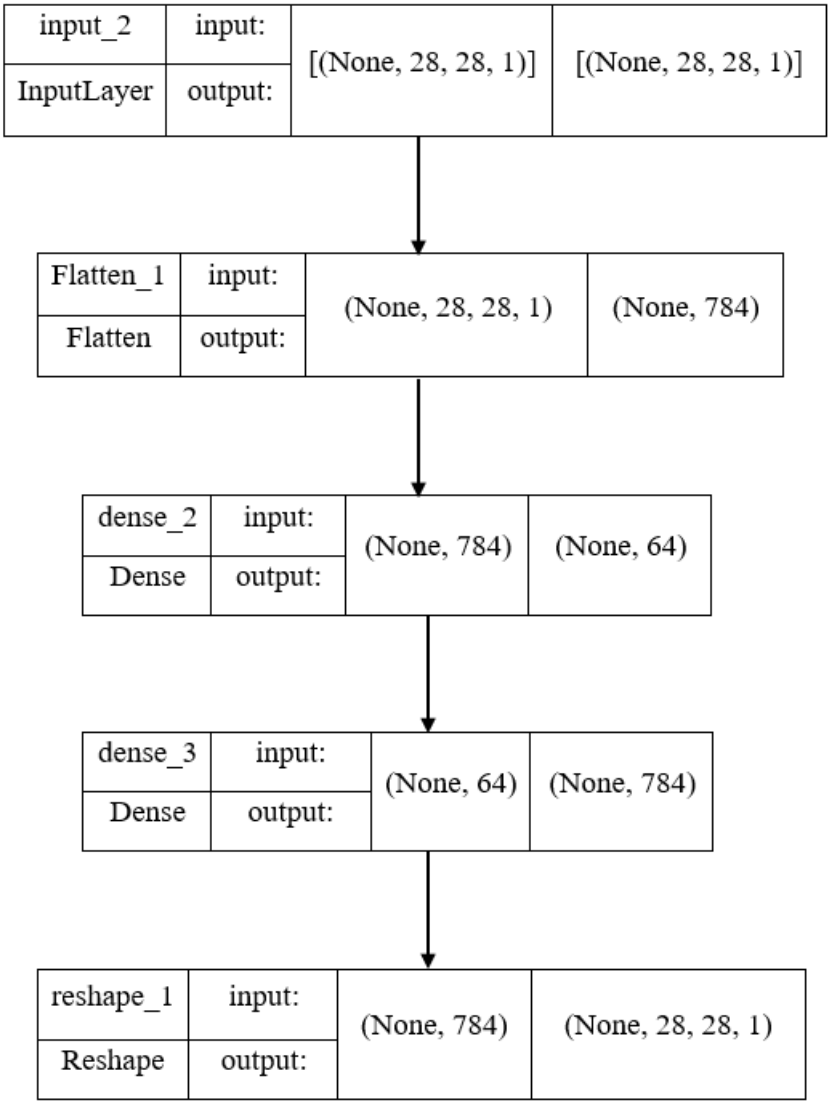

Figure 7. Fully Connected Autoencoder

The main difference between a convolutional Autoencoder and a standard Autoencoder is the use of convolutional layers rather than dense layers. The layout of the structure was influenced by [8]. In order to encode an input with the dimensions 28*28*1 into a latent space with the dimensions 7*7*32, the input is processed through two sets of 2D convolutional layers (32 filters, a 3*3 weight matrix, and relu activation). The image is then upscaled back into its original, shape using Conv2D Transpose layers (32 filters, 3*3 weight matrix, relu activation), using the latent coordinates as input.

Table 1 displays the specifics of every layer.
Figure 8 shows this network's intricate design.

Table 1. Convolutional Autoencoder Layer Details

Layer	Type	Kernel Size	Stride	Padding
1	Conv2D	5x5	2	Same
2	Pool(max)	2x2	2	Same
3	Conv2D	3x3	1	Same
4	Pool(max)	2x2	2	Same
5	TransConv2D	3x3	2	Same
6	TransConv2D	3x3	2	Same
7	Conv2D	3x3	1	Same

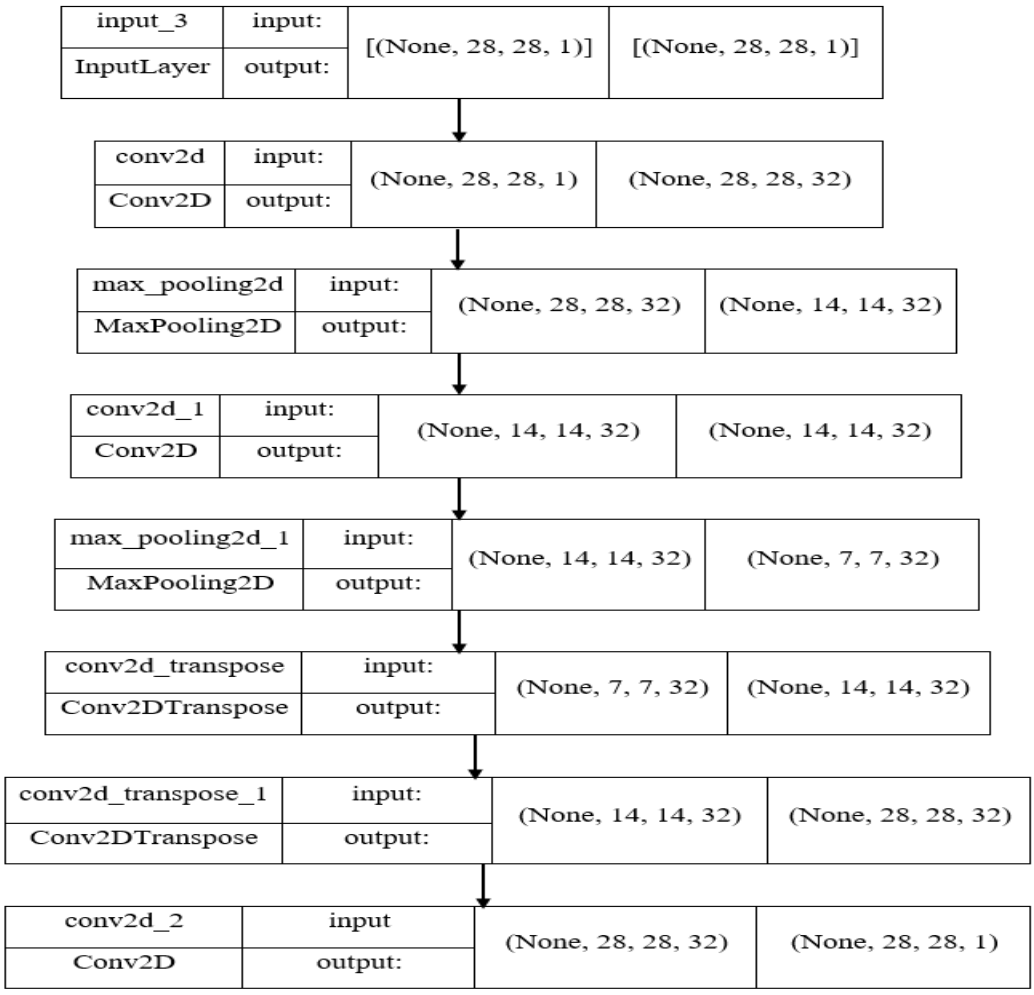

Figure 8. Convolutional Autoencoder

3.2 The Generative Adversarial Network (GAN):

The generative adversarial network (GAN), a two-player network with a generator and a discriminator, works on the principle that the generator creates an equally distributed distribution of fictitious data, which is then sent to the discriminator network for evaluation together with the actual data. When the result is near 1, it will be true; when it is 0, it will be false. The ultimate score for both actual and fake data distribution should be 0.5 if the network has been sufficiently trained. When true and false data are distributed in comparison to matching labels, score errors are produced that mostly account for the discriminator's loss. Both the Generator and the Discriminator are neural networks, and throughout the training phase, they compete with one another. The procedures are repeated multiple times, and each time, the Generator and Discriminator become better at what they are doing. The Figure 9 below helps to visualize the work:

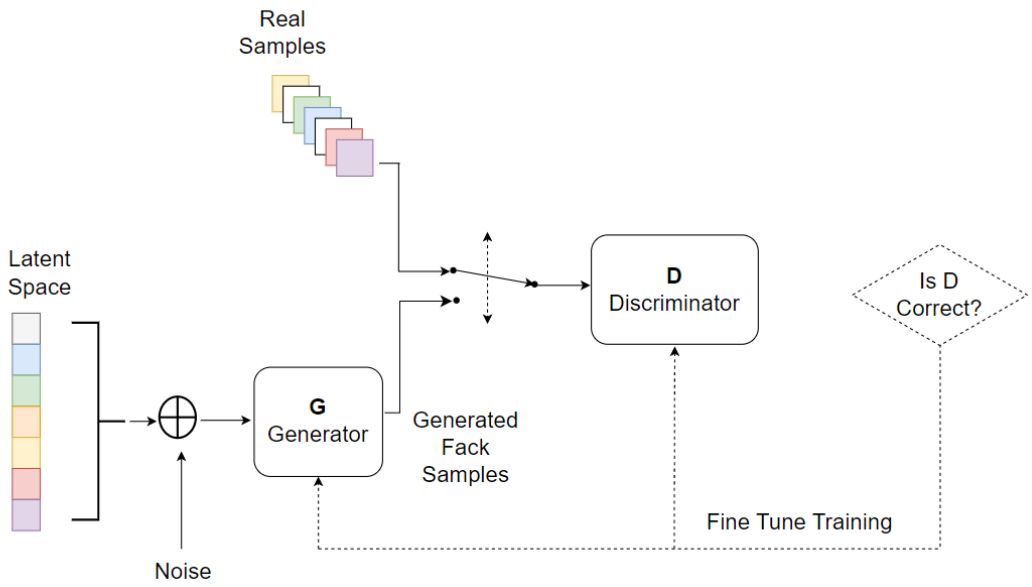

Figure 9. Generative Adversarial Network Architecture

The loss of the generator's identifying component leads to the loss of the generator network. These two loss functions are minimized and maximized during the network's training phase. The formula for mathematics is as follows:

$$\min_G \max_D V(D,G)$$
$$V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log\ D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log\,(1 - D(G(z))] \quad (8)$$

Where x represents the actual data distribution with the anticipated highest discriminative score. GZ reflects the distribution of the created data when applied to the same discriminator, and the score is thought to be lower. Based on the initial GAN, we rebuilt the generative network, added residual blocks, and created a loss function

that is suitable for the task of image denoising. The details are as follows:

3.2.1 The Generator network

Images produced by the generator network architecture are of a high grade and seem authentic. By utilizing noisy images as input, I hoped to outperform the colors and features of the original high-definition image labels. I kept the images' shapes throughout the process and added jump connections between the convolutional and residual networks. To maintain and improve all the features of the input image, I also added a connection to the deconvolutional portion towards the output end. Eight layered residual networks made up the generator network, which worked well at preserving small features. More leftover blocks might be piled, though, if desired, for even greater outcomes. Figure 5 shows the generator's network model. My tests showed that employing 8 layers produced outstanding training results.

3.2.2 The Adversarial network

The adversarial network is going to evaluate both the generated images and the actual images. The network aims to develop a discriminating skill that allows well-trained models to provide generated images with a score near 0 and real label images with a score close to 1. This indicates that the network has an excellent capacity to identify real images from fake ones. The generated images in my experiment are generated by running the noisy images through a generator network, while the clear images act as the actual label images. The discriminator network's simple structure is composed of several convolutional layers, followed by a fully connected layer. Figure 10, 11 shows the structure of the network.

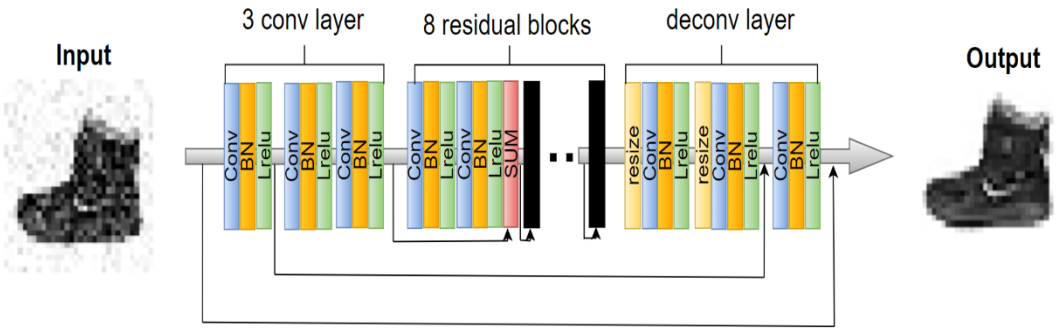

Figure 10. The Generator Network

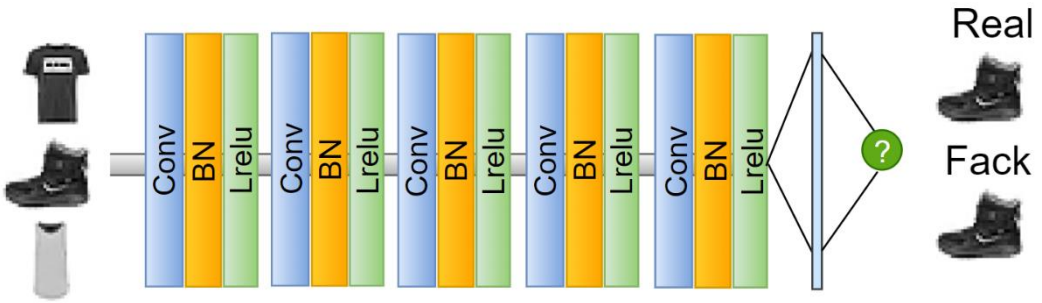

Figure 11. The Discriminator Network

3.2.3 Training process

The network is trained using a method similar to the initial Generative Adversarial Network (GAN). The generative network and the adversarial network make up its two elementary parts. These networks are alternately trained in this process to make sure they function well together. Notably, the parameters utilized by the standalone discriminator network and the discriminator layer of the generative network are the same, encouraging consistency and collaboration. Batch normalization has been introduced into each convolutional layer and residual block of both the generative and discriminator networks to solve the difficulties of disappearing or bursting gradients during training. This method reduces the internal covariate shift and normalizes the input to each layer, which stabilizes the training process. To prevent altering the output distribution, batch normalization is not performed to the last layer of the networks. The generative network is shown how to generate fake images utilizing noisy input or random noise during the training phase. The resemblance of these produced images to the target's real images is then used to assess them. An optimization technique is used to update the weights of the generative network by backpropagating the loss determined from the comparison.

The adversarial network, which consists of the generative network's discriminator component and a separate discriminator network, is taught to discriminate between authentic and artificial images. The training set's actual images are merged with an equal number of produced images in a single batch. The discriminator network's weights are adjusted when the loss is calculated by comparing the discriminator's predictions for these images. Until convergence is reached or after a certain number of iterations, the alternate training process is continued. While the discriminator network develops its capacity to distinguish between actual and created images, the generator network learns to produce increasingly realistic images. Consistency in the training process is ensured by the common parameters between the standalone discriminator network and the discriminator component of the generative network. Convolutional layers and residual blocks with batch normalization assist avoid gradient instability-related problems. Batch normalization makes training more efficient and stable by normalizing the activations, accelerating the networks' convergence toward producing identical images of high quality.

3.3 Dataset:

I have used it in my experiment Fashion MNIST dataset [] for both methods. A popular benchmark dataset in the fields of machine learning and computer vision is the fashion MNIST dataset. It has a similar structure but with images of clothing items in place of the handwritten numbers (0–9) that make up the original MNIST dataset. The 70,000 grayscale 28x28 pixel images in the Fashion MNIST dataset are broken down into 60,000 training samples and 10,000 testing samples. Figure 12 displays a few samples from the training set.

Figure 12. Sample images from the Fashion MNIST dataset

In order to simulate the implications of random modifications or disruptions that may have taken effect in real-life scenarios, I added random Gaussian noise to the Fashion MNIST dataset. The upgraded dataset may be shown in Figure 13 and now contains noisy images.

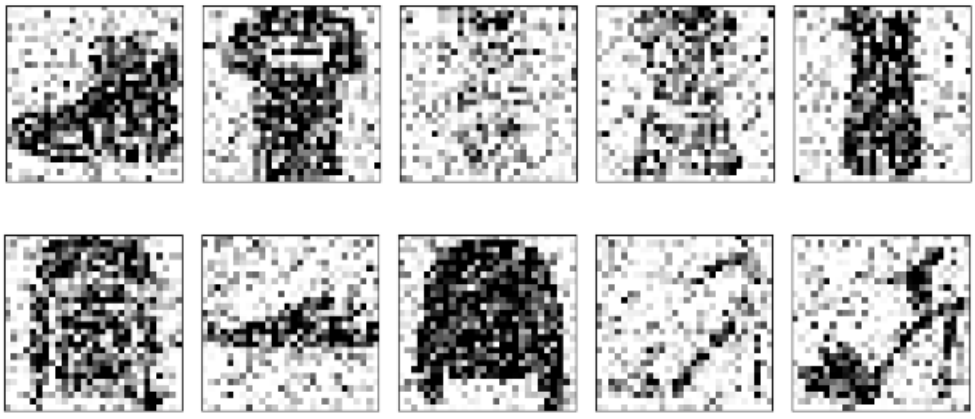

Figure 13. Examples of images addition of Gaussian Noise

Chapter 4 The Tests and Results

4.1 Experimental Setup

I used a variety of libraries and software requirements to carry out the image-denoising experiment utilizing the Fashion MNIST dataset and Autoencoder and GAN. To begin with, I used the Python programming language. The tools required for creating and refining the Autoencoder and GAN models were provided by the deep learning frameworks Keras, Skimage, Graphviz, TensorFlow so on. I used the NumPy library to manage numerical calculations and array operations, while Matplotlib made data visualization and charting easier, enabling me to examine the results efficiently. I used a number of software tools to speed up the experimenting and development process, including PyCharm, Jupyter Notebook, and Google Colab, which offered a full range of functionality for managing project files, coding, and debugging. Furthermore, I used a CUDA-capable GPU to speed up model training by utilizing the power of parallel computing. This made computations faster and training times shorter, especially when working with intricate deep-learning networks. I had taken treatment to install the CUDA and cuDNN libraries, which are necessary for GPU-accelerated deep learning computations on NVIDIA GPUs, in order to ensure flawless interaction with the GPU. By taking advantage of the parallel processing capabilities of contemporary GPUs, these libraries improve the performance of deep learning algorithms.

4.2 Result of Autoencoder

The performance of various methodologies is statistically assessed using three separate measures. It is crucial to combine these measures in order to assess the quality of denoised images and comprehend how well various denoising techniques perform.

The average of the squared discrepancies between anticipated values and actual values, or the Root Mean Squared Error (RMSE)[5], is determined as follows:

$$\text{RMSE}\ (y, \hat{y}) = \sqrt{\frac{1}{N}\sum_{i=0}^{N-1}(y_i - \hat{y}_i)^2} \tag{9}$$

where the subscript index i is the i-th pixel, y denotes the original image, and ŷ is the denoised twin. The RMSE is assessed for each pair of photos before being averaged over the entire dataset. Better accuracy is shown by a lower RMSE number, while larger RMSE values suggest greater prediction errors. Peak signal-to-noise ratio (PSNR)[6] quantifies the ratio of the mean squared error (MSE) between the denoised image and the original image to the peak signal level (i.e., the greatest potential pixel intensity):

$$\text{PSNR}\,(I) = 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE_I}\right) = 20 \cdot \log_{10}\left(\frac{MAX_I}{RMSE_I}\right) \quad (10)$$

where RMSE is the Root Mean Squared Error between the denoised image and the original image, and MAX_I is the highest pixel intensity that may be achieved (typically 255 for 8-bit grayscale images; in our instance, it is normalized to 1). Every pair of photos receives a PSNR evaluation before being averaged over the entire dataset.

4.2.1 Code Explanation

To simulate real-world circumstances, I introduced noise to the training and test data in my code. I adjusted the noise's strength by varying the 'noise_variance' parameter. The values were trimmed to the range [0, 1] to ensure consistency and Gaussian noise was introduced to the data using 'np.random.normal'. The first 10 noisy training digits were then plotted by the program using matplotlib. Using this method, I was able to evaluate how the noise changed the digit images and assess how well the model handled variances in real-world data. Added noise sample feature code is shown in Figure 14.

```
noise_variance=0.15
X_train_noisy = X_train + np.sqrt(noise_variance) * np.random.normal(loc=0.0, scale=1.0, size=X_train.shape)
X_test_noisy = X_test + np.sqrt(noise_variance) * np.random.normal(loc=0.0, scale=1.0, size=X_test.shape)

# Enforce min-max boundaries so it does not go beyond [0,1] range
X_train_noisy = np.clip(X_train_noisy, 0., 1.)
X_test_noisy = np.clip(X_test_noisy, 0., 1.)

# Display images of the first 10 digits in the noisy training data
fig, axs = plt.subplots(1, 10, sharey=False, tight_layout=True, figsize=(20,3), facecolor='white')
n = 0
for j in range(0,10):
    axs[j].matshow(X_train_noisy[n], cmap='Greys')
    axs[j].set_xticks([])
    axs[j].set_yticks([])
    n = n + 1
plt.show()
```

Figure 14. Add noise sample feature code

I created an Autoencoder model Using Keras. To get the input data ready for the model, I first reshape it. Then, I outline the decoder and the encoder with an intermediate hidden layer that makes up the Autoencoder. While the decoder reconstructs clear images from the compressed representation, the encoder compresses the input data. Mean square error (MSE) is used as the loss function during model construction together with the Adam optimizer. This enables the model to learn how to reduce the disparity between the original and recreated images. The Autoencoder model is shown in Figure 15.

```
#--- Define Shapes
n_inputs = X_train.shape[1] # number of input neurons = the number of features X_train

#--- Input Layer
visible = Input(shape=(n_inputs,), name='Input-Layer') # Specify input shape

#--- Encoder Layer
e = Dense(units=n_inputs, name='Encoder-Layer')(visible)
e = BatchNormalization(name='Encoder-Layer-Normalization')(e)
e = LeakyReLU(name='Encoder-Layer-Activation')(e)

#--- Middle Layer
middle = Dense(units=n_inputs, activation='linear', activity_regularizer=keras.regularizers.L1(0.0001), name='Middle-Hidden-Layer

#--- Decoder Layer
d = Dense(units=n_inputs, name='Decoder-Layer')(middle)
d = BatchNormalization(name='Decoder-Layer-Normalization')(d)
d = LeakyReLU(name='Decoder-Layer-Activation')(d)

#--- Output layer
output = Dense(units=n_inputs, activation='sigmoid', name='Output-Layer')(d)

# Define denoising autoencoder model
model = Model(inputs=visible, outputs=output, name='Denoising-Autoencoder-Model')

# Compile denoising autoencoder model
model.compile(optimizer='adam', loss='mse')

# Print model summary
print(model.summary())

# Plot the denoising autoencoder model diagram
plot_model(model, show_shapes=True, dpi=100)
```

Figure 15. The Autoencoder model

4.2.2 Result

I used the fit() function in my code to train an Autoencoder model. As the goal for training, I provide the relevant clean data X_train and the noisy training data X_train_noisy. The batch size, which defines the number of samples processed in each iteration, is 32 and the model is trained for 20 epochs. I plot a line chart to show the loss over epochs in order to track the training process and evaluate the model's effectiveness. The training loss is shown in black on the graph, and the validation loss is shown in red. This aids in my analysis of the model's progress in learning to rebuild clear images and allows me to determine whether it is overfitting or underfitting. The epoch number is indicated on the x-axis of the graphic, which has clearly labeled axes. I am able to learn more about the model's convergence and effectiveness by looking at the trend of the loss values over epochs. This code enables me to visually assess the training results and loss values of the Autoencoder model. Plot a loss chart visualization is shown in Figure 16.

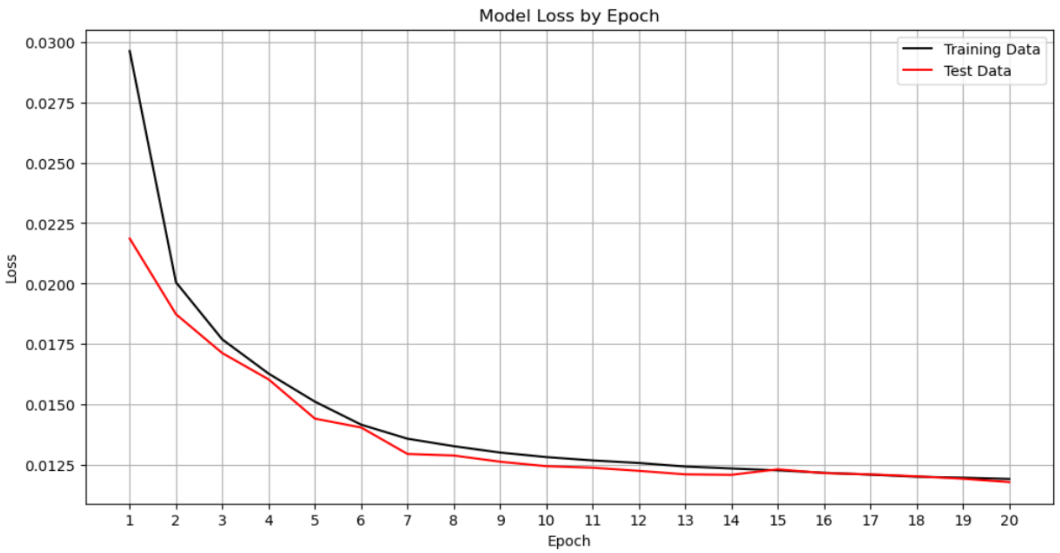

Figure 16. Plot a loss chart visualization

The Structural Similarity Index (SSIM)[7] analyzes color, contrast, and structural information to determine the structural similarity between a reference image and a distorted image. Compared to measurements like PSNR and RMSE, it offers a more thorough evaluation of image quality since it considers structural similarities as well as pixel-level variations. The mathematical definition of the SSIM can be found in the literature; however, for the sake of brevity, it has not been included here. The SSIM index is a decimal value from 0 to 1, where 0 denotes no similarity, 1 denotes complete anti-correlation, and 1 denotes perfect similarity. The SSIM is assessed for every pair of images before being averaged over the entire dataset. The experiment's findings are shown in Table 1. The metrics assessed for the noisy images are displayed in the first column for reference. The Autoencoder outperforms the other investigated strategies, displaying the best performance across all metrics (as can be shown in Figure 12). In comparison to the reference, the other three approaches exhibit decreased RMSE and increased PSNR, but the SSIM values are mostly unaltered. The second-best performing strategy is the Non-Local Means denoiser. The numbers obtained are visibly confirmed in Figure 17. The Autoencoder denoised images are nearly exact replicas of the originals.

Table 2. Results of the experiment

	Noisy Images	**Wavelet**	**NLM**	**Bilateral**	**Autoencoder**
RMSE	0.284	0.2097	0.1831	0.2481	0.1036
PSNR	10.94 dB	13.58 dB	14.78 dB	12.13 dB	19.94 dB
SSIM	0.56	0.55	0.56	0.55	0.78

Figure 17. Results of the denoising procedures

Images successfully reduce noise while maintaining the digits' form. When compared to Autoencoders from a computational standpoint, traditional methods often take less time to train than neural networks. Bilateral filtering, one of the more traditional techniques, can need lengthier processing times, particularly when the level of spatial averaging is raised.

4.3 Result of GAN

I used convolutional layers to define the generator and discriminator models. The generator attempts to produce a denoised version of the supplied noisy image. The discriminator is in charge of separating generated denoised images from actual clean images. I created the loss functions for the generator and discriminator in order to train the models. Pixel-wise loss, which gauges the variance between the generated image and the original, and adversarial loss, which gauges how successfully the generator fools the discriminator, make up the generator loss. How well a discriminator can

distinguish between produced and actual images is measured by discriminator loss. I updated the generator's and discriminator's trainable parameters using Adam optimizers. I also create a function to create sample denoised images for visualization needs during training. The number of epochs the training loop runs for is defined. I iterate over the training dataset in each epoch, adjust the model parameters using forward and backward passes, and use TensorBoard to track my progress. Every 10 epochs, I additionally save checkpoints of the models to keep track of their development and enable model restoration in the future. I restore the most recent checkpoint after training to get the trained model. The noisy images are then run through the generator to create the denoised images, which I then compare to the input noisy image, the ground truth clean image, and the created denoised image.

4.3.1 Code Explanation

The down-stack gradually shrinks the input image's spatial dimensions while preserving high-level features by applying a succession of convolutional layers with progressively smaller filter sizes. The image is further downscaled by the down-sample blocks. Upsampling blocks make up the up-stack, which aims to recreate the denoised image. To add more spatial dimensions, it employs transpose convolutional layers and integrates high-level data from the down-stack with smaller information recorded in the up-stack. Regularization of dropouts is a possibility. The corresponding layers of the up-stack and down-stack are connected by skips. These links give the generator access to both high-level and low-level data, assisting in denoised image reconstruction. The output is created using a convolutional transpose layer with a tanh activation function. The output image keeps the spatial dimensions of the input image. The generator network can be created using the TensorFlow Keras Model object that the code returns. It creates the equivalent denoised image from the input noisy image. It is necessary to implement additional parts individually, including the OUTPUT_CHANNELS constant, the downsample and upsample methods, and more. The generator function code is shown in Figure 18.

```
def Generator():
    inputs = tf.keras.layers.Input(shape=[32,32,1])

    down_stack = [
        downsample(16, 4, apply_batchnorm=False), # (bs, 16, 16, 16)
        downsample(32, 4), # (bs, 8, 8, 32)
        downsample(64, 4), # (bs, 4, 4, 64)
        downsample(64, 4), # (bs, 2, 2, 64)
        downsample(64, 4), # (bs, 1, 1, 64)
    ]

    up_stack = [
        upsample(64, 4, apply_dropout=True), # (bs, 2, 2, 128)
        upsample(64, 4, apply_dropout=True), # (bs, 4, 4, 128)
        upsample(32, 4), # (bs, 8, 8, 64)
        upsample(16, 4), # (bs, 16, 16, 32)

    ]

    initializer = tf.random_normal_initializer(0., 0.02)
    last = tf.keras.layers.Conv2DTranspose(OUTPUT_CHANNELS, 4,
                                           strides=2,
                                           padding='same',
                                           kernel_initializer=initializer,
                                           activation='tanh') # (bs, 32, 32, 1)

    x = inputs

  # Downsampling through the model
    skips = []
    for down in down_stack:
        x = down(x)
        skips.append(x)

    skips = reversed(skips[:-1])

  # Upsampling and establishing the skip connections
    for up, skip in zip(up_stack, skips):
        x = up(x)
        x = tf.keras.layers.Concatenate()([x, skip])

    x = last(x)

    return tf.keras.Model(inputs=inputs, outputs=x)
```

Figure 18. The generator Function

A discriminator network for image-to-image translation is defined in the code on which I worked. It requires two inputs: an image for the input and an image for the target, both with the shape [32, 32, 1]. The spatial dimensions are decreased by concatenating and downsampling these images. The downsampling layers aid in the feature extraction process. The output of the downsampling layers is then subjected to a convolutional layer with batch normalization and LeakyReLU activation. The network can handle both positive and negative values with the activation of LeakyReLU. The features are then processed further using a second convolutional layer. A 6x6 map with a single channel, which represents the expected veracity of the input and target images, is the layer's output. The discriminator function code is shown in Figure 19.

```
def Discriminator():
  initializer = tf.random_normal_initializer(0., 0.02)

  inp = tf.keras.layers.Input(shape=[32, 32, 1], name='input_image')
  tar = tf.keras.layers.Input(shape=[32, 32, 1], name='target_image')

  x = tf.keras.layers.concatenate([inp, tar]) # (bs, 32, 32, channels*2)

  down1 = downsample(16, 4, False)(x) # (bs, 16, 16, 16)
  down2 = downsample(32, 4)(down1) # (bs, 8, 8, 32)

  zero_pad1 = tf.keras.layers.ZeroPadding2D()(down2) # (bs, 10, 10, 32)
  conv = tf.keras.layers.Conv2D(32, 4, strides=1,
                                kernel_initializer=initializer,
                                use_bias=False)(zero_pad1) # (bs, 7, 7, 32)

  batchnorm1 = tf.keras.layers.BatchNormalization()(conv)

  leaky_relu = tf.keras.layers.LeakyReLU()(batchnorm1)

  zero_pad2 = tf.keras.layers.ZeroPadding2D()(leaky_relu) # (bs, 9, 9, 32)

  last = tf.keras.layers.Conv2D(1, 4, strides=1,
                                kernel_initializer=initializer)(zero_pad2) # (bs, 6, 6, 1)

  return tf.keras.Model(inputs=[inp, tar], outputs=last)
```

Figure 19. The discriminator Function

I have developed a training loop for a generative adversarial network (GAN) model in my implementation. The model is trained using the 'train_step()' function for the course of this loop, which lasts for a predetermined number of epochs. This function computes the gradients, updates the trainable model variables, and computes the losses for the generator and discriminator. I utilize a summary writer to keep track of the losses at each step of the training process to monitor my development. In addition, I've included a function called 'generate_images()' to display the results created while testing. The 'fit()' function controls the entire training procedure by invoking 'train_step()' to train the model, displaying example images, and iterating over the epochs. The training loop function code is shown in Figure 20.

```
import datetime
log_dir="logs/"

summary_writer = tf.summary.create_file_writer(
  log_dir + "fit/" + datetime.datetime.now().strftime("%m%d-%H%M") +
  "_lambda_" + str(LAMBDA) + "_var_" + str(VAR) + "_batch_" + str(BATCH_SIZE) + "_buffer_" + str(BUFFER_SIZE))

@tf.function
def train_step(input_image, target, epoch):
  with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
    gen_output = generator(input_image, training=True)

    disc_real_output = discriminator([input_image, target], training=True)
    disc_generated_output = discriminator([input_image, gen_output], training=True)

    gen_total_loss, gen_gan_loss, gen_l1_loss, gen_ssim_loss = generator_loss(disc_generated_output, gen_output, target)
    disc_loss = discriminator_loss(disc_real_output, disc_generated_output)

  generator_gradients = gen_tape.gradient(gen_total_loss,
                                          generator.trainable_variables)
  discriminator_gradients = disc_tape.gradient(disc_loss,
                                               discriminator.trainable_variables)

  generator_optimizer.apply_gradients(zip(generator_gradients,
                                          generator.trainable_variables))
  discriminator_optimizer.apply_gradients(zip(discriminator_gradients,
                                              discriminator.trainable_variables))

  with summary_writer.as_default():
    tf.summary.scalar('gen_total_loss', gen_total_loss, step=epoch)
    tf.summary.scalar('gen_gan_loss', gen_gan_loss, step=epoch)
    tf.summary.scalar('gen_l1_loss', gen_l1_loss, step=epoch)
    tf.summary.scalar('gen_ssim_loss', gen_ssim_loss, step=epoch)
    tf.summary.scalar('disc_loss', disc_loss, step=epoch)
```

```
def fit(train_ds, epochs, test_ds):
  for epoch in range(epochs):
    start = time.time()

    # display.clear_output(wait=True)

    for example_target, example_input in test_ds.take(2):
      generate_images(generator, example_input, example_target)
    print("Epoch: ", epoch)

    # Train
    for n, (target, input_image) in train_ds.enumerate():
      if (n+1) % 10 == 0:
        print('.', end='')
      if (n+1) % 1000 == 0:
        print()
      train_step(input_image, target, epoch)
    print()

    # saving (checkpoint) the model every 20 epochs
    if (epoch + 1) % 10 == 0:
      checkpoint.save(file_prefix = checkpoint_prefix)

    print ('Time taken for epoch {} is {} sec\n'.format(epoch + 1,
                                                        time.time()-start))
  checkpoint.save(file_prefix = checkpoint_prefix)
```

Figure 20. The training loop function

I have developed a training loop for a generative adversarial network (GAN) model in my implementation. The model is trained using the 'train_step()' function for the course of this loop, which lasts for a predetermined number of epochs. This function computes the gradients, updates the trainable model variables, and computes the losses for the generator and discriminator. I utilize a summary writer to keep track of the losses at each step of the training process to monitor my development. In addition, I've included a function called "generate_images()" to display the results created while testing. The ‘fit()’ function controls the entire training procedure by invoking 'train_step()' to train the model, displaying example images, and iterating over the epochs. The training loop flowchart is shown in Figure 21.

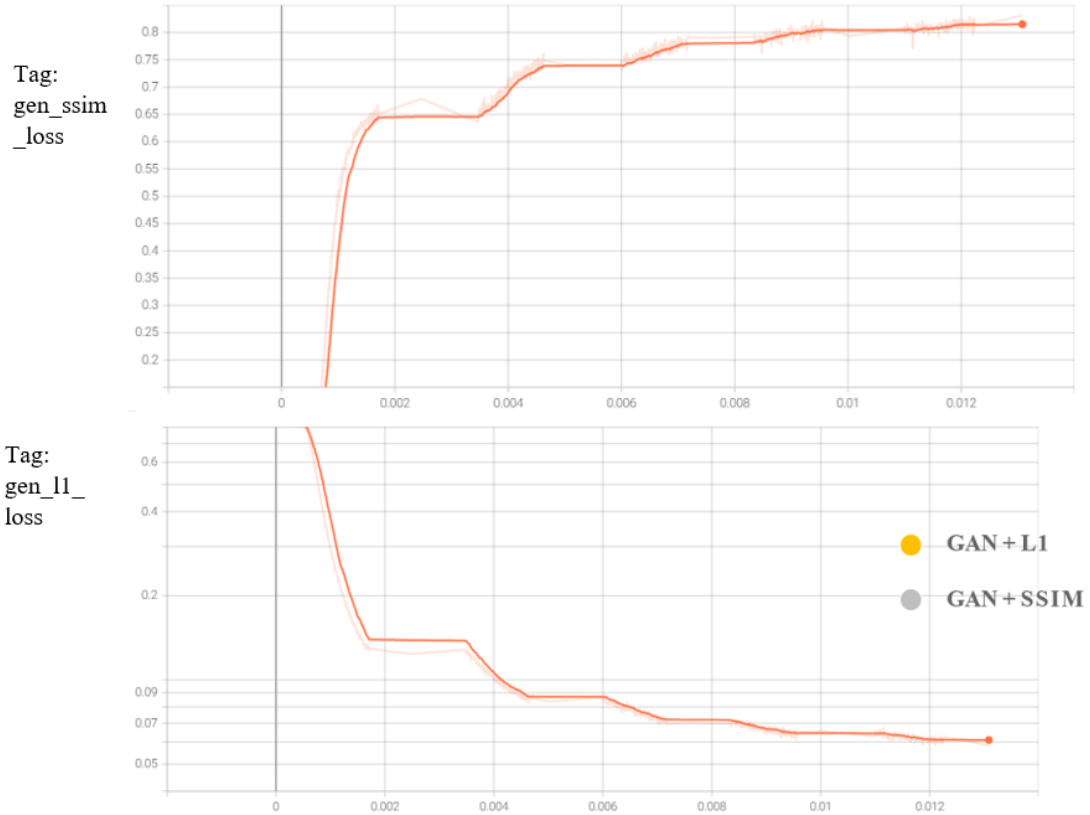

Figure 21. The training loop function

4.2.3 Result

I have trained the model for 5 epochs using the train_images dataset and evaluated it on the test_images dataset after each epoch. The training loop epoch result is shown in Fig. 22, Train Set 1 different loss functions in Table 3, and Train Set 2 different loss functions in Table 4. The GAN final result is shown in Figure 23.

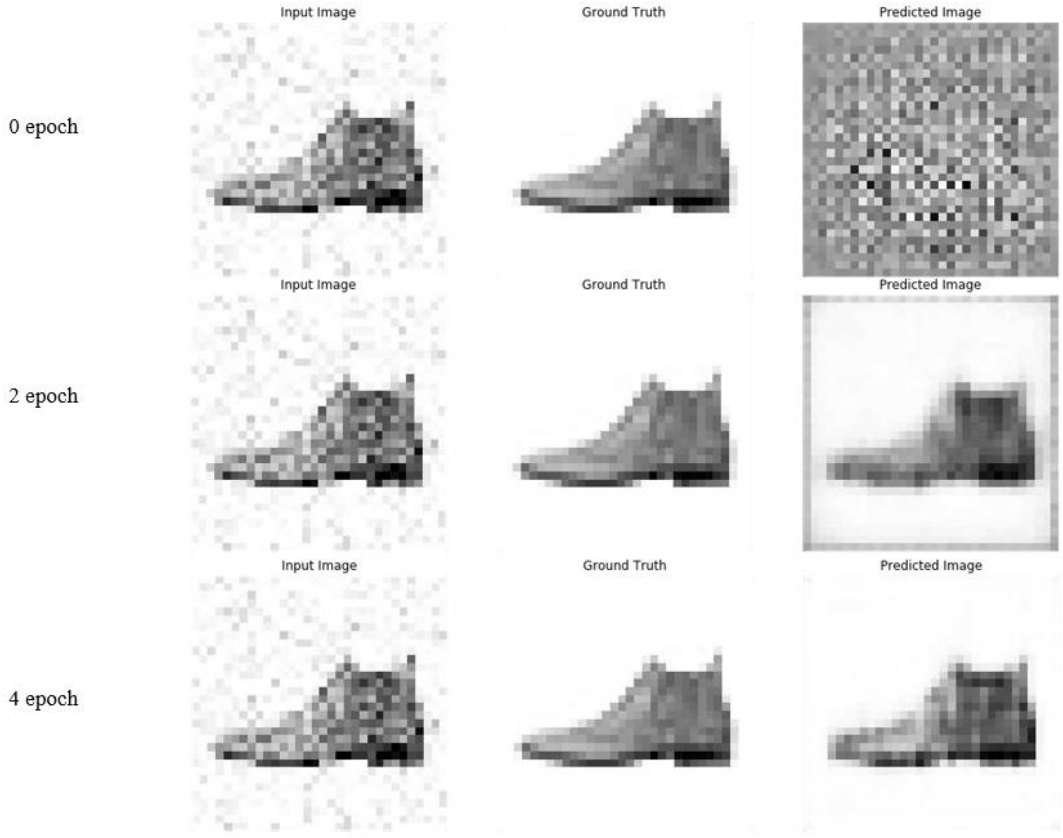

Figure 22. The training loop epoch result

Table 3. Train Set 1 different loss functions

Gen loss	L1 (MAE)	SSIM	PSNR
Noisy Images	0.123	0.681	20.54
GAN + SSIM	0.048	0.871	25.86
GAN + L1	0.046	0.861	26.09
Ground Truth	0.0	1.0	Inf

Table 4. Train Set 2 different loss functions

Gen loss	L1 (MAE)	SSIM	PSNR
Noisy Images	0.123	0.681	20.55
GAN + SSIM	0.046	0.888	26.12
GAN + L1	0.045	0.865	26.19
Ground Truth	0.0	1.0	Inf

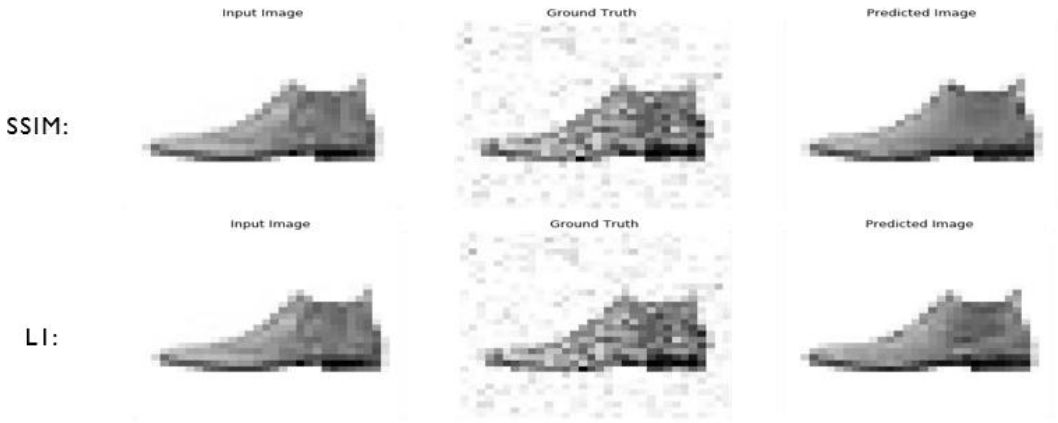

Figure 23. The GAN final result

4.4 Comparison of Results

Based on the assessment criteria (RMSE, PSNR, SSIM, L1) described before, I compare the performance of the Autoencoder and GAN-based image denoising algorithms in this section.

It was found that both the Autoencoder and GAN created visually superior images as compared to the noisy input after analyzing the denoised images produced by both techniques. However, there were variations in the level of detail preservation and quality.

The autoencoder-based approach showed strong denoising performance, successfully lowering noise and retaining image information. The brightness and contrast of the denoised images were enhanced, and there was only a little amount of residual noise. The evaluation measures demonstrated low RMSE, high PSNR, SSIM, and low L1 (MAE) values, which supported the autoencoder approach's efficacy.

The GAN-based approach, on the other hand, demonstrated promising results in decreasing noise while producing aesthetically pleasant and realistic images. With the use of adversarial training, the GAN architecture was able to capture and understand

the underlying image distribution, producing denoised images with improved details and textures. The assessment measures showed competitive performance, albeit occasionally a little worse than the autoencoder-based approach.

It is important to remember that the performance comparison is dependent on the unique dataset, noise properties, and implementation specifics. Additional research and testing can shed more light on the benefits and drawbacks of each strategy and aid in choosing the approach that will work best in a variety of denoising situations.

Overall, image denoising was successfully accomplished using both the autoencoder and the GAN approaches, each of which had distinct advantages. Various considerations, including the desired level of detail preservation, available computer power, and particular application needs, may influence which of the two approaches is best.

4.5 Discussion of Findings

The performance and properties of the autoencoder and GAN-based image denoising algorithms have been compared, providing useful insights. Here, we go over the study's results and implications.

Strong denoising capabilities were demonstrated by the autoencoder-based technique, which successfully reduced noise while retaining crucial image information. The improved clarity and contrast of the denoised images demonstrated the autoencoder's capacity to learn a compact representation of the noise-free images. The assessment metrics—low RMSE, high PSNR, SSIM, and low L1 (MAE) values—also provided additional evidence for the autoencoder approach's efficacy. The autoencoder's power to learn and capture the underlying structure of the images is shown by its ability to rebuild clear images from noisy inputs.

The GAN-based approach, in contrast, made use of the strength of adversarial training to produce denoised images with improved textures and details. The GAN architecture produced visually appealing and realistic images while minimizing noise, showing encouraging results. The images that were produced showed minute details that had been maintained, producing visually appealing results. The GAN methodology showed its capacity to capture complicated image distributions and produce high-quality denoised outputs, even though the evaluation metrics were slightly lower than those of the autoencoder-based method.

The results indicate that the decision between the autoencoder and GAN-based approaches depends on the particular application needs. The GAN-based technique might be more appropriate if maintaining fine details is of the utmost significance. On the other hand, the autoencoder-based approach can be recommended if the goal is to reduce noise overall while preserving image sharpness and clarity.

When choosing an image denoising technique, it is crucial to take into account the trade-off between noise reduction and the preservation of image information. It is also important to consider the dataset's unique properties as well as the noise distribution. It is also important to keep in mind that further adjusting the hyperparameters and altering the architecture may improve the performance of both

approaches.

The comparison of the autoencoder and GAN-based image denoising approaches concludes by highlighting the advantages and disadvantages of each method. This study offers insightful recommendations for researchers and practitioners in selecting the best approach depending on the particular needs of their denoising applications. In order to combine the advantages of both approaches and progress the field of image denoising, future research can further investigate hybrid approaches or innovative architectures.

4.6 Image Denoising Improvement and Future Research Directions:

To enhance the image denoise performance and address the identified weaknesses, the following potential areas for improvement and future research directions can be considered:

1) Model Refinement: The architecture and design of autoencoder and GAN models, among other things. I want to try out several network setups, including changing the number of layers, the size of the filters, and the activation parameters. I can enhance the models' performance and address any shortcomings or difficulties that arose during the first implementation by adapting them to particular denoising needs.
2) Dataset Expansion: I want to enhance my research by utilizing more datasets in order to further improve the models' capacity for generalization and robustness. I can give a thorough assessment of the models' performance in various circumstances by taking into account a variety of datasets that cover multiple domains, noise types, and noise levels.
3) Novel Loss Functions: I'll investigate the usage of unique loss functions designed exclusively for image denoising in my upcoming work. I want to create loss functions that emphasize particular features, such as retaining minute details, increasing textures, or resolving particular problems. The training process will be guided by these personalized loss functions, which will also guarantee that the models capture the desired denoising goals and match the perceptual features I want to attain in the denoised images.
4) Incorporate Domain Knowledge: I think that the models will significantly improve as a result of incorporating domain-specific knowledge. For instance, if I'm working on medical image denoising, I intend to incorporate anatomical priors or medical imaging expertise into the network architecture or training procedure. This will ensure that the models accurately capture domain-specific properties and aid to improve the efficacy of denoising.
5) Optimization Techniques: A key component of my future work will be to optimize the models' training procedures. To improve model convergence and avoid overfitting, I will investigate more sophisticated optimization strategies such as learning rate schedules, batch normalization, or regularization

techniques. I'll also look into methods like data augmentation and data synthesis to expand the training dataset and enhance the models' generalization skills.

6) Real-Time Applications: Another important area will be modeling adaptations for real-time image-denoising applications. I'll look into methods for raising the models' computational effectiveness and speed of inference. The practical usefulness of the denoising algorithms will be increased since real-time denoising will be possible on devices with limited resources or in interactive systems.
7) Collaboration and Open Source: I have a strong belief in the benefits of cooperation and knowledge exchange among researchers. I will actively participate in open-source projects by sharing code, pre-trained models, and working with other researchers. I can progress in the field and promote more innovation by measuring and contrasting my approaches with other cutting-edge denoising techniques.

I hope to significantly advance the field of image denoising and enhance the capabilities of autoencoder and GAN-based approaches by pursuing these lines of inquiry. The long-term objective is to create denoising algorithms that are more reliable and useful for a variety of applications across different fields and sectors.

Conclusion

The decoder successfully recovered denoised images from the latent space representation created by the encoder after it successfully compressed the noisy input images. The evaluation measures, such as Peak Signal-to-Noise Ratio (PSNR), Structural Similarity Index (SSIM), and Root Mean Squared Error (RMSE), showed that the Autoencoder method produced superior denoising quality.

The GAN approach demonstrated the amazing potential for image denoising thanks to its conditional architecture. The effective denoising procedure benefited from the discriminator network's capability to distinguish between real and created clean images and the generator network's capacity to map noisy images to clean ones. The evaluation criteria, such as Mean Absolute Error (MAE), SSIM, and PSNR, further confirmed the GAN method's outstanding performance.

As a supporter of deep learning, I am certain that these results demonstrate the game-changing potential of deep learning models for image denoising. When compared to conventional methods, the Autoencoder and GAN methods significantly improved denoising quality, underscoring the promise of deep learning to improve image quality and advance the area of computer vision.

The quantitative assessment measures employed in this study offer verifiable proof of the Autoencoder and GAN approaches' denoising capability. These metrics are useful tools for evaluating the effectiveness of various denoising algorithms and for directing future research in developing even more advanced and efficient methods.

References

[1] V. Jain and S. Seung, "Natural image denoising with convolutional networks," in Advances in Neural Information Processing Systems (NIPS), 2009.

[2] S. Tan, X. Zhang, H. Wang, L. Yu, Y. Du, and J. Junjun, "A CNN-Based Self-Supervised Synthetic Aperture Radar Image Denoising Approach," IEEE Transactions on Geoscience and Remote Sensing, vol. 59, no. 3, pp. 2031-2043, March 2021.

[3] I. Goodfellow et al., "Generative Adversarial Networks," in Advances in Neural Information Processing Systems (NIPS), 2014.

[4] Y. Chen, T. Pock, and H. Bischof, "Learning Realistic Deep Denoiser Prior for Image Restoration," in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[5] A. Buades, B. Coll, and J.-M. Morel, "A non-local algorithm for image denoising," in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2005.

[6] S. M. Smith, "The scientist and engineer's guide to digital signal processing," California Technical Pub., 1999.

[7] Gajanand Gupta, "Algorithm for Image Processing Using Improved Median Filter and Comparison of Mean, Median and Improved Median Filter," International Journal of Soft Computing and Engineering (IJSCE), vol. 1, no. 5, pp. 2231-2307, November 2011.

[8] W. K. Pratt, "Digital image processing," John Wiley & Sons, 2007.

[9] A. Buades, B. Coll, and J.-M. Morel, "A non-local algorithm for image denoising," in 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), vol. 2, 2005.

[10] D. L. Donoho et al., "Denoising by soft-thresholding," IEEE Transactions on Information Theory, vol. 41, no. 3, pp. 613-627, 1995.

[11] J.-J. Huang and P. L. Dragotti, "WINNet: Wavelet-Inspired Invertible Network for Image Denoising," IEEE Transactions on Image Processing, vol. 31, 2022.

[12] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in Proceedings of the 1998 IEEE International Conference on Computer Vision, 1998.

[13] L. I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," Physica D: nonlinear phenomena, vol. 60, no. 1-4, pp. 259-268, 1992.

[14] J. Zou et al., "Total Variation Denoising With Non-Convex Regularizers," in Proceedings of the IEEE International Conference on Image Processing (ICIP), vol. 7, 2018.

[15] M. Elad and M. Aharon, "Image denoising via sparse and redundant representations over learned dictionaries," IEEE Transactions on Image Processing, vol. 15, no. 12, pp. 3736-3745, 2006.

[16] B. Widrow and I. Kollár, "Quantization noise: roundoff error in digital computation,

signal processing, control, and communications," Cambridge University Press, 2008.
[17] R. C. Gonzalez and R. E. Woods, "Digital image processing," Pearson Education India, 2018.
[18] E. J. Candes and B. Recht, "Exact matrix completion via convex optimization," Foundations of Computational Mathematics, vol. 9, no. 6, pp. 717-772, 2009.
[19] F. Zhang, D. Liu, X. Wang, W. Chen, and W. Wang, "Random noise attenuation method for seismic data based on deep residual networks," in International Geophysical Conference, Beijing, China, 2018, pp. 1774–1777.
[20] Z. Guo, Y. Sun, M. Jian, and X. Zhang, "Deep residual network with sparse feedback for image restoration," Applied Science, vol. 8, no. 12, p. 2417, 2018.
[21] C. Tian, L. Fei, W. Zheng, and W. Zuo, "Deep learning on image denoising: an overview," Neural Networks, vol. 131, pp. 251–275, 2020.
[22] A. E. Ilesanmi and T. O. Ilesanmi, "Methods for image denoising using convolutional neural network: a review," Complex & Intelligent Systems, vol. 7, pp. 2179–2198, 2021.
[23] P. Vincent et al., "Stacked denoising Autoencoders: Learning useful representations in a deep network with a local denoising criterion," Journal of Machine Learning Research, vol. 11, Dec, pp. 3371-3408, 2010.
[24] J.-Y. Zhu et al., "Unpaired image-to-image translation using cycle-consistent adversarial networks," in Proceedings of the IEEE International Conference on Computer Vision, 2017.
[25] I. Goodfellow et al., "Generative adversarial nets," in Advances in Neural Information Processing Systems, 2014.
[26] Wikipedia, "Autoencoder," [Online]. Available: https://en.wikipedia.org/wiki/Autoencoder (visited on 2023/06/01).
[27] MNIST, "The mnist database of handwritten digits," [Online]. Available: http://yann.lecun.com/exdb/mnist/ (visited on 2023/06/01).

Acknowledgment

My bachelor's degree has been a significant milestone in my life. It has shaped my academic journey and provided me with invaluable knowledge and skills. As I approach the end of this chapter, I would like to take a moment to express my gratitude to all the individuals who have contributed to my success during these years.

First and foremost, I would like to thank my Instructor Dr. Wang Lei for his invaluable guidance, expertise, and unwavering support. His mentorship and insightful feedback have been instrumental in shaping the direction and quality of this research.

I am grateful to my institution/department for providing the necessary resources and facilities that have greatly facilitated the execution of this study. The access to computational resources, research tools, and library facilities has been of immense help in conducting experiments and gathering relevant literature.

I would also like to acknowledge the researchers and authors in the field of image denoising and deep learning whose contributions have laid the foundation for this study. Their pioneering work and insightful publications have served as a guiding light and source of inspiration for my own research endeavors.

I would like to express my gratitude to my classmates for their encouragement, support, and stimulating discussions throughout this research journey. Their diverse perspectives and insights have been invaluable in shaping my ideas and refining the outcomes of this study.

Finally, I am deeply grateful to my family for their unwavering belief in me and their constant encouragement. Their love, understanding, and encouragement have been the driving force behind my perseverance and commitment to this research.

Although it is not possible to mention everyone by name, I extend my sincere appreciation to all those who have contributed in various ways to the successful completion of this research project. Your support and contributions have been invaluable, and I am truly grateful for your involvement.

Thank you all for being an integral part of this journey and for your unwavering support.