Politecnico di Torino

Politecnico di Torino

Master's Degree in Electroni Engineering
A.a. 2024/2025
Sessione di Laurea Luglio 2025

Deep Joint Denoising and Compression for Satellite Images

Relatori:
prof. Enrico Magli
prof. Diego Valsesia
prof. Dan Schonfeld
prof. Ahmet Enis Cetin

Candidati:
Matteo Carnevale Schianca

Copyright by

Matteo Carnevale Schianca

2025

Dedicated to my family, my girlfriend, and my friends. Without them, I wouldn't be who I am today.

ACKNOWLEDGMENTS

I would like to give my warmest thanks to my family: **mamma**, **papà**, and **Chiara**, who have always been by my side and supported me every step of the way. A special thanks goes to my **nonna**, who, before every exam and during every tough moment, was always ready to help—just like my parents—driving me where I needed to go and giving me encouragement. I am also deeply grateful to my girlfriend **Marzia**, who has stayed close to me even when I was stressed or in a bad mood.

I want to thank my friend **Leonardo**, who is always around and ready to offer help, as well as **Alessandro**, **Luca**, **Edoardo**, **Federico**, and **Vittorio**, who have been there for me in difficult times, showing genuine concern and talking things through. I am equally thankful to **Eugenio** and **Paolo**, who, despite the distance, have always shown their support and willingness to help. My sincerest gratitude also goes to professor **Michele Goano** and supervisor **Jenna Stephens**, who, in a challenging moment, offered their generous assistance without expecting anything in return. Lastly, I thank all those who, day by day, remain by my side, supporting me with a kind word, a gentle touch, or a warm hug.

This research greatly benefited from the constant guidance and valuable feedback provided by my professors and advisors at every stage of development. **Dan Schonfeld**, **Enrico Magli (Politecnico di Torino)** and **Diego Valsesia (Politecnico di Torino)** helped me in shaping the direction of my work, offering continuous feedback on methodology and helping me refine the experimental design. They provided key suggestions regarding the theoretical and technical concept behind this work. Their combined expertise was indispensable for shaping the experimental approach, resolving problems

as they arose, and continuously improving the outcome. They have always been ready to support me in times of difficulty, offering their help and presence. Their support greatly contributed to the success of this work. I am deeply grateful for their support.

MCS

CONTRIBUTIONS OF AUTHORS

This thesis is structured into several chapters, each focusing on different aspects of my research: from the theoretical foundations of neural networks, through learned image compression (specifically the mean-scale hyperprior model), and toward image denoising approaches that integrate both baseline architectures and customized enhancements. Throughout these chapters, I draw upon a variety of references to provide background, support my arguments, and acknowledge the works that inspired or informed my methods.

Neural Network Fundamentals

A brief historical and theoretical overview of neural networks sets the stage for the thesis. In particular, I rely heavily on the textbook *Deep Learning* by Goodfellow, Bengio, and Courville (1) to discuss the core principles, common architectures, and training techniques. Some classic works on perceptrons, multilayer networks, and early training algorithms are also cited whenever relevant to illustrate key transitions in the evolution of deep learning methods (for example, Minsky and Papert's *Perceptrons* (2), the concept of backpropagation from Rumelhart et al. (3), and the principle of Hebbian learning from Hebb (4)).

Learned Image Compression and the Mean-Scale Hyperprior

When discussing learned image compression, I introduce key concepts such as quantization, entropy coding, and variational autoencoders. These concepts are drawn in part from general overviews of data compression (5; 6) and from works specifically addressing end-to-end compression approaches using deep neural networks. The mean-scale hyperprior model, which is central to my thesis, is mainly based on Ballé et

al. (7), along with supporting ideas from related works on hyperpriors and transform coding (8; 9; 10; 11; 12). In the corresponding chapter, I cite these references to acknowledge their contributions to the design of my own compression framework.

Denoising and Baseline Architectures

Later in the thesis, I shift focus to denoising. I outline the general state of image denoising research and the transition from classical methods to deep-learning-based approaches, referencing survey papers like Ren et al. (13). I also mention self-supervised or weakly supervised denoising strategies proposed by Noise2Noise (14), Noise2Void (15), and Noise2Self (16) to highlight the landscape of techniques that do not require clean ground truth.

As a simple baseline architecture for denoising, I introduce NAFNet (17). I reference it when discussing straightforward yet powerful design principles for image restoration. In my experiments, I integrate certain features from NAFNet (and other relevant network designs) into my new model, combining them with the hyperprior-based compression framework to tackle the denoising task more effectively.

Dataset: SEN12MS

All experiments presented in the thesis utilize the SEN12MS dataset (18), which provides multi-spectral and SAR (Sentinel-1/2) imagery for deep-learning applications. Its companion papers (18; 19; 20; 21; 22; 23) provide additional details on Sentinel missions, geolocation accuracy, and satellite data processing (e.g., via Google Earth Engine). I draw on these works to justify my data preparation steps and experiment protocols, citing them in the text where appropriate.

Remarks on Citations and Contributions

Overall, the thesis references these works whenever I introduce a concept or method closely tied to the original research. For example, when discussing neural network backgrounds, I cite classical and modern foundational papers. When detailing my experiments with learned compression and the mean-scale hyperprior, I cite Ballé's series of works and others in this area. When focusing on denoising, I refer to the recent survey and the specific self-supervised approaches that informed my understanding of modern denoising pipelines. The SEN12MS dataset references are cited when I describe the data and experiment setup. This approach ensures proper acknowledgment of each author's role and contribution and shows how these published ideas collectively form the theoretical and practical basis of my thesis.

Acknowledgments for Advisors and Professors

Final Remarks

In summary:

- **Neural networks:** Primarily based on deep-learning textbook materials (1), with classic references for historical depth.
- **Compression and hyperprior:** Built on top of the mean-scale hyperprior approach in (7) and related works (8; 9; 10; 12; 5; 6; 11).
- **Denoising:** Surveilled through (13), and the basic denoising architecture used: NAFNet (17).
- **SEN12MS dataset:** Data-related sections draw upon (18; 19; 20; 21; 22; 23) for multi-spectral and SAR imagery details.

These references have either been cited directly in the text where each concept or technique is explained or will appear in close proximity to the sections to which they apply. This ensures transparency regarding how these works underpin my own methods and extensions. To my knowledge, no references are missing. If any additional references are needed, they can be added easily.

TABLE OF CONTENTS

CHAPTER PAGE

CHAPTER PAGE

CHAPTER PAGE

LIST OF TABLES

LIST OF FIGURES

FIGURE **PAGE**

SUMMARY

Modern imaging systems across various domains—from embedded electronics to satellite platforms—must handle growing amounts of data while facing strict constraints on energy, memory, and bandwidth. In many cases, such as real-time Earth observation or surveillance, images are also compromised by sensor noise and environmental interference. Conventional approaches usually adopt a two-step pipeline: first denoising each image to remove artifacts, then compressing it for storage or transmission. This separation can lead to higher computational cost, more complex hardware requirements, and inefficient use of bits.

To address these issues, this thesis proposes a **joint denoising-compression framework** that simultaneously learns to eliminate noise and reduce image size in a more effective and less complex way than using the two techniques sequentially, i.e. first denoising and then compressing the image. The foundation of this work is the **Mean Scale Hyperprior** architecture, a learned-compression model that adaptively allocates bits based on the complexity of image features. By introducing denoising operations directly into the compression pipeline, the network can filter out noise before encoding, thereby allocating fewer bits to irrelevant artifacts.

The thesis compares **two distinct training approaches**. In both approaches the models are trained with a noisy data input, however in the first approach the model is trained to reconstruct the exact same noisy input, meaning that the output of the models are compared with the same noisy data given as input. Therefore the models learn to reproduce exactly the image, preserving noise as part of the learned representation. Instead in the second approach, the network receives as input always noisy data but

it is explicitly trained to produce in output clean version of the input data, effectively removing noise. This distinction has significant implications for resource-constrained systems: when the model focuses on clean outputs, it learns to drop unnecessary noise information, which can save bits and lead to higher-quality reconstructions at the same bitrate.

To validate these methods under **realistic conditions**, experiments are conducted using the **SEN12MS** dataset, a well-annotated set of multispectral satellite images. A custom noise model simulates the kinds of interference commonly found in satellite sensors, reflecting the practical challenges of this environment. Performance is measured via common compression metrics (bitrate) and reconstruction quality metrics such as **Peak Signal-to-Noise Ratio (PSNR)**. Results consistently show that networks trained to produce clean outputs achieve better rate-distortion trade-offs and maintain higher PSNR for a given bitrate compared to those trained on noisy targets.

Overall, the thesis concludes that **integrating denoising and compression** yields significant advantages for satellite operations and other bandwidth-limited scenarios. Fewer bits are wasted on encoding noise, total processing steps are reduced, and final reconstructions maintain strong visual quality. These findings demonstrate that a joint denoising-compression approach and the associated training method can enable more resource-efficient models, reduce overall system complexity, and outperform conventional two-step solutions that treat denoising and compression separately. By directly integrating noise removal into a single compression architecture, this approach lowers computational overhead and leads to stronger rate-distortion performance in a unified framework.

CHAPTER 1

INTRODUCTION

In modern life, technology pervades a wide range of contexts - from embedded systems and portable devices to advanced control networks - where efficiency in both storage and data transmission is paramount. In particular, embedded systems and low-power devices face stringent energy and memory constraints, making every bit of data a precious resource. At the same time, these devices still need to capture, process, and transmit high-quality images for tasks like surveillance, medical diagnostics, or visual analytics.

Such situations pose a fundamental question: **how can I guarantee high-quality images while minimizing both the resources required and distortion in real-world application?** Often, images are degraded by various forms of noise - due to sensor limitations, environmental factors, or other artifacts. To overcome this, it is common to either apply a separate denoising algorithm before compressing the image or use different post-processing techniques to improve the final quality once the image has been reconstructed. This approaches increase the output quality but greatly increases the processing cost and complexity.

In this thesis I show the results achieved with a different approach. Instead of the two-step approach I evaluate the performance of a joint denoising-compression framework that combine both operations into a single, integrated model.

Our methodology aims to modify an already existing deep-compression architecture, the Mean Scale Hyperprior, to incorporate denoising in the compression pipeline. This direct integration aims to suppress noise at the source, eliminating redundant in-

formation enhancing the compressed representation, the transmission rate and the reconstruction quality. The idea is that fewer bits are allocated to encoding irrelevant artifacts such as noise, resulting in a more efficient and robust system and potentially also resulting in an increased reconstruction quality.

In practical terms, I train end-to-end this joint denoising-compression architecture using noisy images as inputs and clean, noise-free images as targets. By means of this supervised strategy, the network learns to map noisy data to its reconstructed clean form, concurrently reducing the bit needed to encode the feature of the image. Early stage noise removal helps the system to lower the complexity of the representation and to reduce the storage and bandwidth requirements.

By simultaneously removing noise and encoding the data, this approach lowers the bit rate while preserving or even enhancing the quality of the reconstructed data. For resource-constrained systems, this translates into reduced memory usage, faster inference times, and lower power consumption - very important advantages in today's embedded and mobile applications.

Two Distinct Training Approaches. While many existing techniques learn to encode images as they are - whether clean or noisy - this thesis examines both the traditional approach and an alternative: training the model to reconstruct clean, noise-free outputs from noisy inputs. In the first case, noise is treated as part of the signal to be faithfully reproduced; in the second, noise is explicitly discarded by the model, leading to more efficient compression and often improved visual quality. By systematically comparing these two training strategies, I clarify how different objectives affect resource usage, bitrate allocation, and the final reconstructed image - especially under conditions typical of satellite operations. Demonstrating the strengths, weaknesses, and performance

trade-offs of these approaches is a central focus of this work and form the basis on how best to integrate denoising into compression pipelines.

Why focus on satellite image acquisition? Satellites operate under resource and power constraints similar to embedded systems: they rely on finite on-board energy, must handle large amounts of data, and have limited downlink bandwidth to transmit imagery to ground stations. In these circumstances, any method that reduces data size while preserving quality can significantly cut operational costs and enable high-fidelity remote sensing. Furthermore, satellites operate in harsh space where sensors are subjected to strong interferences that induces several kinds of deterioration among that noise. Consequently, a robust denoising and compression pipeline can be essential for ensuring clear and reliable data products used in applications such as disaster management, agricultural monitoring, and environmental studies.

Furthermore, the similarity in constraints - energy efficiency, limited computational budgets, and tight storage requirements - makes satellite image acquisition an excellent case study for validating joint denoising-compression methods initially designed for embedded platforms. To work with a real-world scenario I use an already-available well-annotated and curated dataset called SEN12MS, which comprises multispectral satellite images. SEN12MS offers a complete dataset for training and evaluating deep models. To replicate reasonable satellite settings, w e implement a noise model reflecting the real interference and environmental noise that affects the sensor. By blending noisy models into SEN12MS data, I replicate the conditions under which satellites typically capture and transmit imagery. This allows us to both train and evaluate the architecture under realistic conditions.

CHAPTER 2

NEURAL NETWORKS

2.1 Foundations of Neural Networks

2.1.1 Historical Context and Basic Concepts

Historically, *neural networks* refers to the biological structure of neurons and synapses. The neuron is the core computational unit of the system. After receiving and processing the input stimuli, the neurons generate sequences of electrical impulses that they transmit to other neurons via different synaptic connections. During the learning process, these links either strengthen or inhibit, thereby determining the general dynamics of the system.

This dynamics inspire neural network algorithms. The primary concept is to let a computer, like humans, learn from data observations. Inspired by the biophysical dynamics of neurons, artificial neurons applied in machine learning models alter their output depending on weighted inputs. The network learns by repeatedly changing link weights, which helps it to precisely identify input data. Once trained, the network can accurately identify fresh data using acquired features. While neural networks easily accomplish tasks like image recognition, audio processing, and natural language understanding, classical computers find these difficult.

McCulloch and Pitts (24) proposed a simplified model of neuronal function in 1943 and explained how networks of such units could perform logical operations. These so-called McCulloch-Pitts neurons are the conceptual basis of most modern neural network algorithms. Later, in 1949, Hebb proposed an innovative principle of learning in his work *The Organization of Behavior: A Neuropsychological Theory* (4). He suggested that the

connection between neurons strengthens when they are active simultaneously, increasing the circuit's sensitivity to the same stimulus in the future.

Researchers revived interest in artificial neural networks in the late 1950s and 1960s with Rosenblatt's perceptron model (25). He suggested that multi-layer architectures (perceptrons) can, in principle, solve more complicated classification tasks than a single McCulloch-Pitts neuron; however, there was no general training algorithm for such multi-layer networks at the time. Later, Minsky and Papert (2) highlighted the geometric nature of these learning problems and provided a rigorous function detailing which tasks single-layer perceptrons can and cannot perform.

The problem was how to train this network. The work of Rumelhart and his colleagues proposed an interesting answer (3). They showed that gradient descent can effectively train multi-layer neural networks through backpropagation.

Limitations in computer capacity and the absence of efficient training methods caused a fall in popularity of neural networks in the last part of the twentieth century. Thanks to many big, high-quality datasets and improved computer technology today, neural networks are once more becoming trendy. These days, problems including autonomous driving, medical diagnostics, and financial forecasting rely on them.

Let's now delve into a presentation of the most significant steps and discoveries throughout the Neural Network and Deep Learning journey. All the material and work presented here is my own, though it draws on and is inspired by the reference book *Machine Learning with Neural Networks* by Bernhard Mehlig (26).

2.1.2 McCulloch-Pitts Neurons

McCulloch and Pitts (**?**) presented one of the first simplified neuron models. Usually referred to as a *threshold unit*, the neuron generates a binary output from a limited number of weighted inputs:

$$s_i(t+1) = \mathrm{sgn}\left(\sum_{j=1}^{N} w_{ij}\, s_j(t) \;-\; \theta_i\right), \tag{2.1}$$

(FIG?)the index of the working neuron is i; it receives in input the outputs of the previous neurons to which it is connected (from $j = 1$ to N) multiplied by the value w_{ij} that represents the strengths of the connection from the neuron j to neuron i. sgn(b) is the *signum* function,

$$\mathrm{sgn}(b) \;=\; \begin{cases} -1, & b < 0, \\ +1, & b \geq 0, \end{cases} \tag{2.2}$$

and θ_i is the threshold for neuron i. In this model, $s_i(t) \in \{-1, +1\}$ represents whether neuron i is inactive (-1) or active ($+1$) at discrete time t. The summation

$$b_i(t) \;=\; \sum_{j=1}^{N} w_{ij}\, s_j(t) \;-\; \theta_i \tag{2.3}$$

is known as the *local field* of neuron i. Every neuron updates its state by comparing the local field $b_i(t)$ to zero.

Referred to as weight, the term w_{ij} captures the strength of the synaptic connection between two neurons, therefore reflecting their relationship. The weight has either positive, negative, or even zero value - meaning no connection. These perfect models capture the fundamental concept of neurons as computing units aggregating weighted inputs and

generating (probably non-linear) outputs. Although the McCulloch-Pitts formalism is simple, many recent neural network research methods are based on it.

These perfect models capture the fundamental concept of neurons as computing units aggregating weighted inputs and generating (probably non-linear) outputs. Although the McCulloch-Pitts formalism is simple, many recent neural network research methods are based on it.

2.2 Perceptrons and Supervised Learning

A significant milestone in neural network research came with Rosenblatt's *perceptrons* (25). Rosenblatt's *perceptrons* (25) marked a turning point in neural network study. The perceptrons are a feedforward architecture - the connections proceed from the input to the output without deviation - and do not include lateral connections or feedback loops. They involve multiple instantiations of *layers* of McCulloch-Pitts neuron. The standard structure comprises one or more *hidden* layers and an *output* layer. The training phase consists of iterative adjustments, that is, small parameter updates, repeated until the network classifies the input data correctly.

Recent discoveries reveal that perceptrons with several hidden layers - so-called *deep networks* - can remarkably successfully identify complicated data, including images, with consistency.

2.2.1 Perceptrons in Classification

Rosenblatt (25) first proposed the concept of linking several layers of McCulloch-Pitts neurons in a feedforward fashion. He named such networks *perceptron*. Figure 1 shows a schematic arrangement with an input layer on the left (shown by N input terminals), a hidden layer of J neurons, and an output layer of M neurons.

The input patterns and the corresponding target (or label) vectors are denoted by

$$\mathbf{x}^{(\mu)} = \left[x_1^{(\mu)} x_2^{(\mu)} : x_N^{(\mu)}\right], \qquad \mathbf{t}^{(\mu)} = \left[t_1^{(\mu)} t_2^{(\mu)} : t_M^{(\mu)}\right]. \tag{2.4}$$

where $\mu = 1, \ldots, p$ is the index representing the different pattern in the dataset.

The input pattern progresses through sequential computations from the input layer to the output layer, where the output vector is provided.

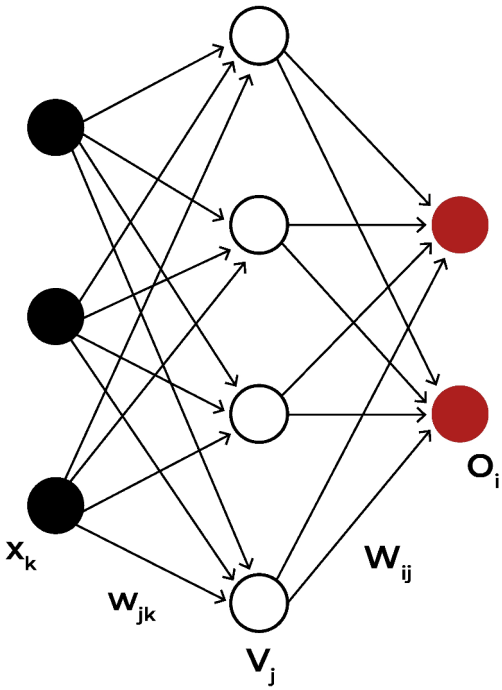

Figure 1: Single-layer Perceptron Schematic.

In the hidden neurons, the output or local field is calculated using the weights w_{jk}, which connect the previous neuron k with the current neuron j, along with the thresholds θ_j according to the formula:

$$b_j = \sum_k w_{jk} x_k - \theta_j, \tag{2.5}$$

and pass the result through an activation function $g(\cdot)$. For instance,

$$V_j = g(b_j). \tag{2.6}$$

The outputs from the hidden layers are then fed into the output neurons. Each output neuron combines its inputs using weights W_{ij}, which represent the connection between hidden neuron j and output neuron i, along with a threshold Θ_i.

$$O_i = g\Big(\sum_j W_{ij} V_j - \Theta_i\Big). \tag{2.7}$$

the index $i = 1, \ldots, M$ labels the output neurons.

For a classification task, all the different output vectors $O_i^{(\mu)}$ should match the corresponding target vector $t_i^{(\mu)}$ for each pattern μ and each output i. Perceptrons are iteratively *trained* until the condition $O_i^{(\mu)} = t_i^{(\mu)}$ is satisfied for each pattern. At a high level, this is done by repeatedly applying a small adjustment to the weights if the current network output differs from the targetc (Section 2.3 for more detail).

2.2.2 Single-Layer Perceptron: Linear Separability

In a single-layer perceptron, there is only one neuron receiving an N-dimensional inputs $\mathbf{x}$ and producing a binary output $O \in \{-1, +1\}$. The neuron's output can be written as

$$O = \operatorname{sgn}\big(\mathbf{w} \cdot \mathbf{x} - \theta\big), \tag{2.8}$$

where θ is a threshold and $\operatorname{sgn}(\cdot)$ is the signum function. Geometrically, this neuron can *separate* inputs with $t = +1$ from those with $t = -1$ by a hyperplane in $\mathbb{R}^N$ specified by $\mathbf{w} \cdot \mathbf{x} = \theta$ (see Figure 2).

If the classification problem is such that a single hyperplane can separate all $t^{(\mu)} = \pm 1$, the problem is *linearly separable*. In two dimensions $\mathbb{R}^2$, a line that separates the points labeled $+1$ from those labeled -1 can be found.

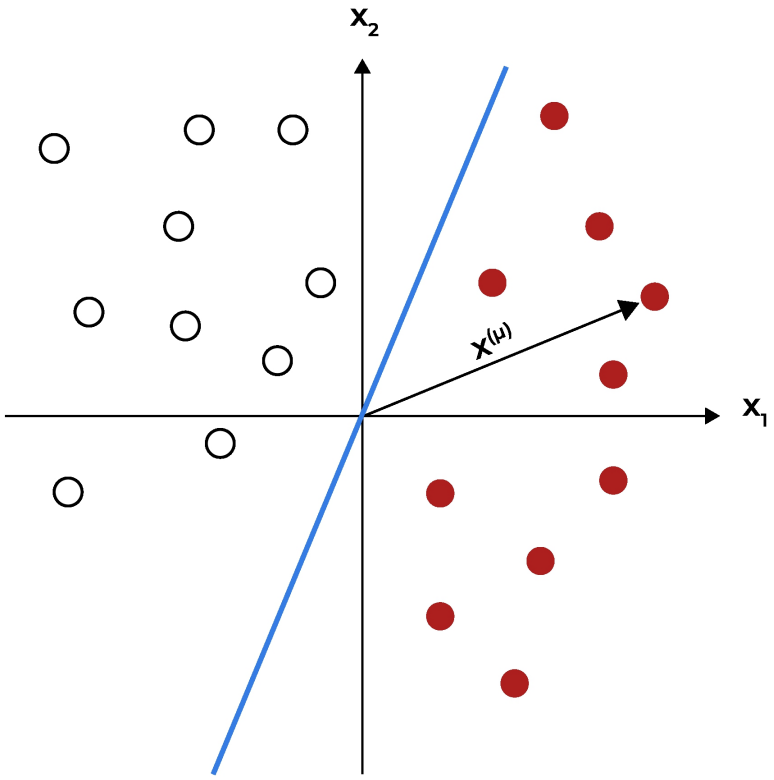

Figure 2: Liner Separability in 2D.

2.2.2.1 Examples: Boolean Functions

A classic illustration is the *AND* boolean function of two binary inputs. One can plot the four possible input values (x_1, x_2) along with their desired outputs t, and then find a line that separates $t = -1$ from $t = +1$. In contrast, the *XOR* function is not linearly separable in two dimensions; no single line can correctly separate all data points. Still,

it is possible to embed XOR into higher dimensions or use more complex neural network architectures (Section 2.4) to achieve correct classification.

2.3 Gradient Descent

2.3.1 Iterative Learning Rules

Consider a single-layer perceptron aiming to satisfy $O^{(\mu)} = t^{(\mu)}$ for a set of p patterns $\{\mathbf{x}^{(\mu)}, t^{(\mu)}\}$. A straightforward approach to adjusting the weights is as follows. After presenting a training pattern that the network misclassifies, the weights $\mathbf{w}$ are updated by a small step along (or against) $\mathbf{x}^{(\mu)}$ to tilt the decision boundary appropriately:

$$\mathbf{w} \leftarrow \mathbf{w} + \eta\, t^{(\mu)}\, \mathbf{x}^{(\mu)}. \tag{2.9}$$

This is known as Rosenblatt's perceptron learning rule (25). A slightly refined version is

$$\mathbf{w} \leftarrow \mathbf{w} + \eta\big(t^{(\mu)} - O^{(\mu)}\big)\, \mathbf{x}^{(\mu)}, \tag{2.10}$$

which makes no change if the network already classifies $\mathbf{x}^{(\mu)}$ correctly (i.e., $O^{(\mu)} = t^{(\mu)}$).

2.3.2 Gradient Descent for Linear Units

Inspired by the iterative learning rule, the gradient descent further improves the training process. It computes, for every parameter, the error function's gradient. It then modifies these values such that the total error is less. Most contemporary neural networks are built on this approach, which guarantees consistent performance improvement over several repetitions.

Consider a *linear unit*, that is a neuron with a *linear activation function* and with the thresholds $\theta = 0$.

$$O_i^{(\mu)} = \sum_k w_{ik}\, x_k^{(\mu)} \tag{2.11}$$

In this case the classification problem $O_i^{(\mu)} = t_i^{(\mu)}$ is satisfied for each pattern when the weights take the form:

$$O_i^{(\mu)} = \sum_k w_{ik}\, x_k^{(\mu)} = t_i^{(\mu)} \tag{2.12}$$

The solution makes use of the *overlap matrix* Q, whose members reflect the pairwise inner products of the input patterns:

$$Q_{\mu\nu} = \frac{1}{N} \mathbf{x}^{(\mu)} \cdot \mathbf{x}^{(\nu)}. \tag{2.13}$$

Here $\mathbf{x}^{(\mu)}$ and $\mathbf{x}^{(\nu)}$ denotes the μ-th and ν-th input pattern; N is the dimensionality of each input vector. The solution is obtained by inverting Q.

$$w_{ik} = \frac{1}{N} \sum_{\mu,\nu} t_i\mu) \left(Q^{-1}\right)_{\mu\nu} x_k\nu. \tag{2.14}$$

μ, ν index the training patterns while $t_i^{(\mu)}$ are the goal labels for the i-th output unit. This expression reveals how the weight update depends on the correlations between input patterns recorded by Q.

Assume that the input patterns are linearly independent, allowing Q to be invertible and so the solution of equation (Equation 2.14) exists. The solution can be found by defining an *energy function*[1], usually called loss or cost function, and solve it iteratively:

$$H = \frac{1}{2} \sum_{\mu,i} \left[t_i^{(\mu)} - O_i^{(\mu)} \right]^2, \tag{2.16}$$

this equation is minimal when $O_i^{(\mu)} = t_i^{(\mu)}$ for all patterns μ and outputs i.

Since the output is given by

$$O_i^{(\mu)} = \sum_k w_{ik} \, x_k^{(\mu)}, \tag{2.17}$$

the energy H depends on the weights $\mathbf{w}$. The derivative are computed with regard to an arbitrary weight w_{mn} to minimise H by means of gradient descent:

$$\begin{aligned} \frac{\partial H}{\partial w_{mn}} &= \frac{\partial}{\partial w_{mn}} \left[\frac{1}{2} \sum_{\mu,i} \left(t_i^{(\mu)} - O_i^{(\mu)} \right)^2 \right] = \sum_{\mu,i} \left(t_i^{(\mu)} - O_i^{(\mu)} \right) \left(-\frac{\partial O_i^{(\mu)}}{\partial w_{mn}} \right), \\ \frac{\partial O_i^{(\mu)}}{\partial w_{mn}} &= \delta_{im} \, x_n^{(\mu)}. \end{aligned} \tag{2.18}$$

Combining this leads to:

---

[1]An alternative notation for the quadratic cost function has the form:

$$H(\mathbf{w}, b) = \frac{1}{2n} \sum_x \| d_i - f(x_i, \mathbf{w}, b) \|^2, \tag{2.15}$$

where $f(x_i, \mathbf{w}, b)$ represents the network's output for input x, and d_i is the true label.

$$\frac{\partial H}{\partial w_{mn}} = -\sum_{\mu,i} \left(t_i^{(\mu)} - O_i^{(\mu)} \right) \delta_{im} \, x_n^{(\mu)} = -\sum_{\mu} \left(t_m^{(\mu)} - O_m^{(\mu)} \right) x_n^{(\mu)}. \tag{2.19}$$

To sum up applying *gradient descent* in the weight space on $H(\mathbf{w})$ with learning rate η, the weight update rule becomes

$$\Delta w_{mn} = -\eta \, \frac{\partial H}{\partial w_{mn}} = \eta \sum_{\mu} \left(t_m^{(\mu)} - O_m^{(\mu)} \right) x_n^{(\mu)}. \tag{2.20}$$

This resembles the perceptron rule (Equation 2.10). Observe that linear perceptrons can only solve classification problems with $p \leq N$ (or *linearly independent* input patterns). More complicated problems often require either more dimensions or nonlinear activation functions in hidden layers.

2.4 Multi-Layer Perceptrons for Nonlinear Boundaries

Some classification tasks are *not* linearly separable in the original space. However, by adding hidden layers with nonlinear activation functions, the network can implement *piecewise linear* or *more sophisticated* decision boundaries. Figure 3 illustrates a two-dimensional example: a single straight boundary cannot separate the two labeled classes, but multiple hidden neurons can partition the plane into different regions, each associated with $t = +1$ or $t = -1$.

2.4.0.1 Boolean XOR:

A classic example is the XOR problem in two dimensions, which is not solvable by any single threshold neuron. Nevertheless, a small network with two hidden neurons can split the input plane suitably so that the final output neuron combines hidden states to yield the correct classification.

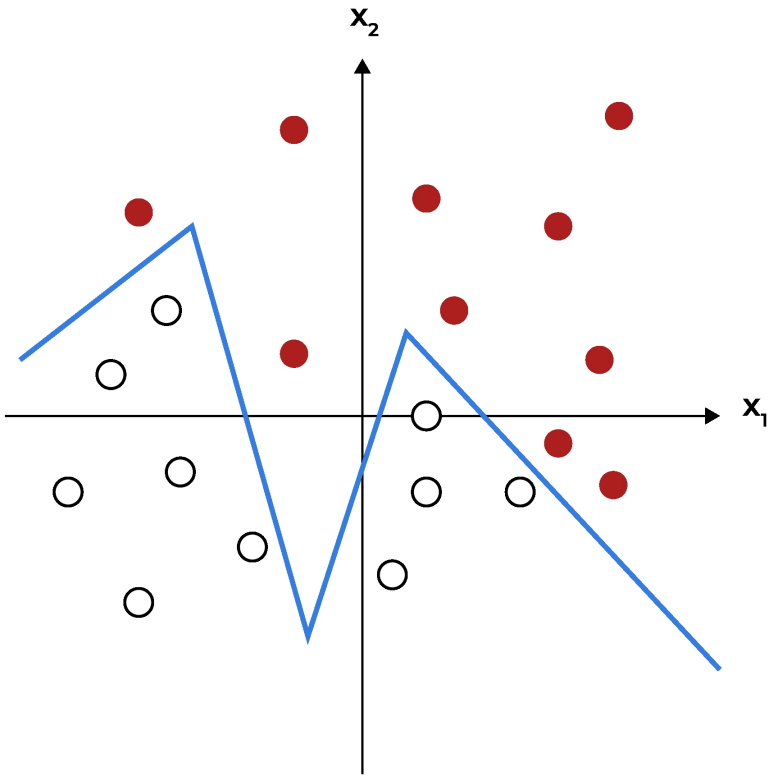

Figure 3: Non-Linear Classification Boundaries.

2.5 Chain Rule and Error Backpropagation

This section expands the discussion about gradient descent, examining the training process more deeply.

Consider a feed-forward network composed of a hidden layer and on output layer. The input vector $\mathbf{x}^{(\mu)}$ has N components and the network outputs form an M-dimensional vector

$$\mathbf{O}^{(\mu)} \;=\; \left[O_1^{(\mu)} O_2^{(\mu)} \vdots O_M^{(\mu)}\right], \tag{2.21}$$

which should ideally match the M-component target $\mathbf{t}^{(\mu)}$ for each input $\mathbf{x}^{(\mu)}$.

The following equation represents how the output of the intermediate layer and the last layer are computed:

$$V_j^{(\mu)} = g\left(b_j^{(\mu)}\right) \qquad \text{with} \qquad b_j^{(\mu)} = \sum_{k=1}^{N} w_{jk}\, x_k^{(\mu)} - \theta_j, \tag{2.22a}$$

$$O_i^{(\mu)} = g\left(B_i^{(\mu)}\right) \qquad \text{with} \qquad B_i^{(\mu)} = \sum_{j} W_{ij}\, V_j^{(\mu)} - \Theta_i. \tag{2.22b}$$

Here $g(\cdot)$ is a differentiable (or piecewise differentiable) *activation function*, and $V_j^{(\mu)}$ are the hidden-layer outputs.

The gradient-descent updates on the *energy function* are used to systematically adjust the weights $\{W_{ij}, w_{jk}\}$ and thresholds $\{\Theta_i, \theta_j\}$ so that $\mathbf{O}^{(\mu)}$ closely matches $\mathbf{t}^{(\mu)}$. The energy function used has the form:

$$H \;=\; \frac{1}{2} \sum_{\mu,i} \left[t_i^{(\mu)} - O_i^{(\mu)}\right]^2. \tag{2.23}$$

This function vanishes if and only if $O_i^{(\mu)} = t_i^{(\mu)}$ for all patterns μ and all output components i. The gradient-descent weight updates take the form

$$\delta W_{mn} = -\eta\, \frac{\partial H}{\partial W_{mn}}, \qquad \delta w_{mn} = -\eta\, \frac{\partial H}{\partial w_{mn}}, \tag{2.24}$$

where $\eta > 0$ is the *learning rate*. Substituting Eq. (Equation 2.23) into Eq. (Equation 2.24) and applying the chain rule reveals how to calculate these partial derivatives from local quantities in the network.

2.5.0.1 Weights Feeding into the Output Layer

First, for the weights W_{mn} connecting the hidden layer to the output layer:

$$\frac{\partial H}{\partial W_{mn}} = -\sum_{\mu,i} \left[t_i^{(\mu)} - O_i^{(\mu)}\right] \frac{\partial O_i^{(\mu)}}{\partial W_{mn}}. \tag{2.25}$$

Since $O_i^{(\mu)} = g(B_i^{(\mu)})$ and $B_i^{(\mu)}$ itself is a linear combination of $V_j^{(\mu)}$,

$$\frac{\partial O_i^{(\mu)}}{\partial W_{mn}} \;=\; g'(B_i^{(\mu)}) \delta_{im} V_n^{(\mu)}, \tag{2.26}$$

where $g'(\cdot)$ is the derivative of g, and δ_{im} is the Kronecker delta ensuring that W_{mn} only affects $O_m^{(\mu)}$. As a result:

$$\begin{aligned} \delta W_{mn} = -\eta \, \frac{\partial H}{\partial W_{mn}} &= \eta \sum_{\mu} \left[t_m^{(\mu)} - O_m^{(\mu)}\right] g'(B_m^{(\mu)}) \; V_n^{(\mu)} \\ &\equiv \eta \sum_{\mu} \Delta_m^{(\mu)} \; V_n^{(\mu)}, \end{aligned} \tag{2.27a}$$

$$\Delta_m^{(\mu)} = \left[t_m^{(\mu)} - O_m^{(\mu)}\right] g'(B_m^{(\mu)}). \tag{2.27b}$$

Hence the *weighted output error* for pattern μ at output neuron m is $\Delta_m^{(\mu)}$, which is minimal whenever $O_m^{(\mu)} = t_m^{(\mu)}$.

2.5.0.2 Weights Feeding into the Hidden Layer.

For the weights w_{mn} that connect the *input* layer to the hidden layer:

$$\frac{\partial H}{\partial w_{mn}} = -\sum_{\mu,i} \left[t_i^{(\mu)} - O_i^{(\mu)}\right] \frac{\partial O_i^{(\mu)}}{\partial w_{mn}}. \tag{2.28}$$

Similar as before, the output of the network $O_i^{(\mu)}$ depends on the output of the intermediate hidden layer $V_n^{(\mu)}$ through the following relation:

$$O_i^{(\mu)} = g(B_i^{(\mu)}), \tag{2.29}$$

where $B_i^{(\mu)}$ is a linear combination of $V_j^{(\mu)}$:

$$B_i^{(\mu)} = \sum_j W_{ij} V_j^{(\mu)} - \Theta_i. \tag{2.30}$$

Again $V_j^{(\mu)}$ is a linear combination of the input multiplied by the corresponding weight:

$$V_j^{(\mu)} = g\Big(\sum_{k=1}^{N} w_{jk} x_k^{(\mu)} - \theta_j\Big) \tag{2.31}$$

There fore the term $\frac{\partial O_i^{(\mu)}}{\partial w_{mn}^{(\mu)}} = \sum_{\mu,i} \frac{\partial O_i^{(\mu)}}{\partial V_\ell^{(\mu)}} \frac{\partial V_\ell^{(\mu)}}{\partial w_{mn}}$. Giving the following equation:

$$\frac{\partial H}{\partial w_{mn}} = -\sum_{\mu,i} \left[t_i^{(\mu)} - O_i^{(\mu)}\right] \frac{\partial O_i^{(\mu)}}{\partial V_\ell^{(\mu)}} \frac{\partial V_\ell^{(\mu)}}{\partial w_{mn}}. \tag{2.32}$$

Observe that

$$\frac{\partial O_i^{(\mu)}}{\partial V_\ell^{(\mu)}} \;=\; g'(B_i^{(\mu)})W_{i\ell}, \qquad \frac{\partial V_\ell^{(\mu)}}{\partial w_{mn}} \;=\; g'(b_\ell^{(\mu)})\delta_{\ell m}x_n^{(\mu)}. \tag{2.33}$$

Putting these together leads to

$$\begin{aligned} \delta w_{mn} &= \eta \sum_\mu \sum_i \Delta_i^{(\mu)} W_{im} g'(b_m^{(\mu)}) x_n^{(\mu)}, \\ \Delta_i^{(\mu)} &= \Big[t_i^{(\mu)} - O_i^{(\mu)}\Big] g'(B_i^{(\mu)}). \end{aligned} \tag{2.34}$$

which can be rewritten as

$$\delta w_{mn} = \eta \sum_\mu \delta_m^{(\mu)} x_n^{(\mu)}, \tag{2.35a}$$

$$\delta_m^{(\mu)} = \sum_i \Delta_i^{(\mu)} W_{im} g'(b_m^{(\mu)}). \tag{2.35b}$$

2.5.0.3 Threshold Updates

Thresholds Θ_m at the output layer and θ_m at the hidden layer are updated analogously:

$$\begin{aligned} \delta\Theta_m &= -\eta \frac{\partial H}{\partial \Theta_m} = \eta \sum_\mu^p \Big[t_m^{(\mu)} - O_m^{(\mu)}\Big]\Big[-g'(B_m^{(\mu)})\Big] = -\eta \sum_\mu \Delta_m^{(\mu)}, \\ \delta\theta_m &= -\eta \frac{\partial H}{\partial \theta_m} = \eta \sum_\mu^p \sum_i \Delta_i^{(\mu)} W_{im}\Big[-g'(b_m^{(\mu)})\Big] = -\eta \sum_\mu \delta_m^{(\mu)}. \end{aligned} \tag{2.36}$$

2.6 Stochastic Gradient Descent Algorithm

In *stochastic gradient descent*, (SGD), the weights and thresholds are updated after each pattern (or a small *mini-batch* of patterns). The index ℓ is used to keep track of the different layer. The input layer is labeled as $\ell = 0$, while the output layer is represented by $\ell = L$. The state of the neurons in layer ℓ are given by $V_j^{(\ell)}$, while $w_{jk}^{(\ell)}$ represents the connection between the neuron j and k in the ℓ layer. The errors at stage ℓ is $\delta_k^{(\ell)}$ as per Equation 2.35b. Using this notation, the output of an intermediate layer $V_j^{(\ell)}$ is given by:

$$V_j^{(\ell)} = g\left(\sum_k w_{jk}^{(\ell)} V_k^{(\ell-1)} - \theta_j^{(\ell)}\right). \tag{2.37}$$

Consequently, the different updates take the form:

$$\delta W_{mn} = \eta\, \Delta_m^{(\mu)}\, V_n^{(\mu)}, \delta w_{mn} = \eta\, \delta_m^{(\mu)}\, x_n^{(\mu)}, \delta\Theta_m = -\eta\, \Delta_m^{(\mu)}, \delta\theta_m = -\eta\, \delta_m^{(\mu)}. \tag{2.38}$$

Here, the network sees a single sample $(\mathbf{x}^{(\mu)}, \mathbf{t}^{(\mu)})$, computes the corresponding errors, and then immediately updates $\mathbf{w}$ and $\mathbf{W}$. The term *stochastic* arises because the gradient direction for individual samples can fluctuate away from the *true* gradient (averaged over the entire dataset).

Algorithm 1 sketches the *sequential* version of this procedure for a network with L layers of neurons (labeling the input layer as $\ell = 0$). One often groups patterns into *mini-batches* containing a few samples, which strikes a balance between purely stochastic weight changes and full batch updates. In that case, sums over the mini-batch replace

the single-pattern updates, and the learning rate η may absorb an extra factor of $1/m_B$ depending on the implementation convention.

Algorithm 1 Stochastic Gradient Descent with Backpropagation

Initialize weights $w_{jk}^{(\ell)}$ (random) and thresholds $\theta_j^{(\ell)} = 0$.
for $\nu = 1, \ldots, \nu_{\max}$ **do**
  Choose an index μ (possibly at random).
  Forward Pass:
  for $\ell = 1, \ldots, L$ **do**
    $V_j^{(\ell)} \leftarrow g\Big(\sum_k w_{jk}^{(\ell)} V_k^{(\ell-1)} - \theta_j^{(\ell)}\Big)$ ▷ $V_k^{(0)} \equiv x_k^{(\mu)}$ (the input).
  end for
  Output Error: $\delta_i^{(L)} \leftarrow g'\big(b_i^{(L)}\big)\Big[t_i^{(\mu)} - V_i^{(L)}\Big]$.
  Backward Pass:
  for $\ell = L, \ldots, 2$ **do**
    for each neuron j in layer $(\ell - 1)$ **do**
      $\delta_j^{(\ell-1)} \leftarrow \Big(\sum_i \delta_i^{(\ell)} w_{ij}^{(\ell)}\Big)\, g'\big(b_j^{(\ell-1)}\big)$
    end for
  end for
  Parameter Updates:
  for $\ell = 1, \ldots, L$ **do**
    for each neuron m in layer ℓ **do**
      for each index n in layer $(\ell - 1)$ **do**
        $w_{mn}^{(\ell)} \leftarrow w_{mn}^{(\ell)} + \eta\, \delta_m^{(\ell)} V_n^{(\ell-1)}$
      end for
      $\theta_m^{(\ell)} \leftarrow \theta_m^{(\ell)} - \eta\, \delta_m^{(\ell)}$
    end for
  end for
end for

2.7 Data Preprocessing

Prior to training, it is usually advantageous to *center* and *scale* the inputs so that each component of $\mathbf{x}$ has approximately zero mean and unit variance across the training set:

$$\frac{1}{p}\sum_{\mu=1}^{p} x_k^{(\mu)} = 0, \quad \frac{1}{p}\sum_{\mu=1}^{p}\left[x_k^{(\mu)}\right]^2 = 1. \tag{2.39}$$

This common preprocessing step ensures that excessively large local fields are less likely to appear at the start of training. It also helps avoid saturating non-linear activations such as $\sigma(b)$ or $\tanh(b)$, whose derivatives become tiny for large $|b|$. Further, it reduces correlations across components that might impede convergence if the input coordinates vary on disparate scales.

2.8 Challenges in the training process

2.8.1 Overfitting

A critical aspect of supervised learning is *generalization*. When a neural network learns the training data too well, including noise and irrelevant details a phenomenon named overfitting can raise. As a result, the model performs exceptionally well on the training set but poorly on unseen data. A flag that indicates the model is starting to overfit can be the training loss continuing to decrease, but the validation loss starting increasing. That can happen because:

- A model with too many parameters (e.g., large neural networks) can quickly memorize the training data instead of learning general patterns.
- The training dataset is too small; the network may overfit the limited examples it has seen.

- The network might learn these as patterns if the dataset contains noise or mislabeled examples, reducing its generalization ability.

Cross validation mitigates overfitting by splitting the data into a *training* set and a *validation* set. While training proceeds on the training set, the performance (e.g., the energy H or the fraction of misclassified patterns) should be monitored also on the validation set. As soon as the validation metric ceases to improve - and potentially starts to worsen - the training process should be stopped. This *early stopping* strategy helps prevent learning spurious structure.

Other solutions include:

- Regularization (e.g., L^2 penalty).
- Dropout: Randomly deactivating neurons during training.

2.8.2 Adaptive Learning Rate

Choosing a fixed learning rate η can be tricky: large η fosters fast changes but risks instability, while small η yields slow but stable improvements. One often augments the basic gradient update by adding a momentum term:

$$\delta w_{mn}^{(t)} = -\eta \left.\frac{\partial H}{\partial w_{mn}}\right|_{\mathbf{w}^{(t)}} + \alpha\, \delta w_{mn}^{(t-1)}, \tag{2.40}$$

where $\alpha \in [0, 1)$ is the *momentum constant*. This update rule encodes a form of *inertia*: if the gradient $\partial H/\partial w_{mn}$ stays relatively constant through several iterations, then δw_{mn} accumulates, yielding a faster escape from shallow minima. If the gradient oscillates in sign, the updates partially cancel each other, promoting smoother training dynamics.

2.8.3 Summary of Key Points

- **Backpropagation:** The chain rule applied in layered networks allows efficient computation of the gradient of the energy function (Equation 2.23).
- **Stochastic Gradient Descent (SGD):** Rather than summing over all training patterns, weights can be updated after processing a single pattern or a mini-batch. This yields a noisy but often more effective descent in weight space.
- **Preprocessing:** Centering and scaling inputs (and possibly applying PCA) can significantly aid training convergence.
- **Overfitting:** Larger networks can memorize training sets too well, impairing generalization. Cross validation and early stopping mitigate this.
- **Adaptive Learning Rates:** Momentum methods, such as Eq. (Equation 2.40) and Nesterov's approach, can achieve faster and more robust convergence than a fixed learning rate alone.

Putting these elements together, is obtained a powerful strategy for fitting large feed-forward neural networks to labeled data while maintaining reasonable generalization properties.

CHAPTER 3

DEEP LEARNING

3.1 How Many Hidden Layers?

Chapter 2 presented how a single hidden layer can solve classification problems that are not *linearly separable*. Here, the aim is to investigate when one hidden layer might suffice to solve the task and under what circumstances multiple hidden layers, i.e. deeper networks, become useful or even necessary.

A natural way to address these questions is to consider the network's task as *function approximation*. In the training process, the network use $\mathtt{p}$ labeled examples $\left[\mathbf{x}^{(\mu)}, \mathbf{t}^{(\mu)}\right]$, where each input $\mathbf{x}^{(\mu)} \in \mathbb{R}^{\mathsf{N}}$ must be mapped to a target $\mathbf{t}^{(\mu)}$, to learn how to approximate the true map

$$\mathtt{t}(\mathbf{x}) : \mathbb{R}^{\mathsf{N}} \longrightarrow \mathbb{R} \quad \text{(or possibly to a discrete set)}. \tag{3.1}$$

In effect, $\mathtt{t}(\mathbf{x})$ is replaced by $\mathsf{O}(\mathbf{x})$, the output of the network, and the goal is to make $\mathsf{O}(\mathbf{x}) \approx \mathtt{t}(\mathbf{x})$. How many *hidden layers* are required to reach the intended accuracy?

3.1.1 Universal Approximation Results

3.1.1.1 Real-Valued Targets.

Consider first real-valued inputs $\mathbf{x} \in \mathbb{R}^{\mathsf{N}}$ and a continuous target $\mathtt{t}(\mathbf{x}) \in \mathbb{R}$ and a network with one or two hidden layers using *sigmoid* or *hyperbolic tangent* as activation function.

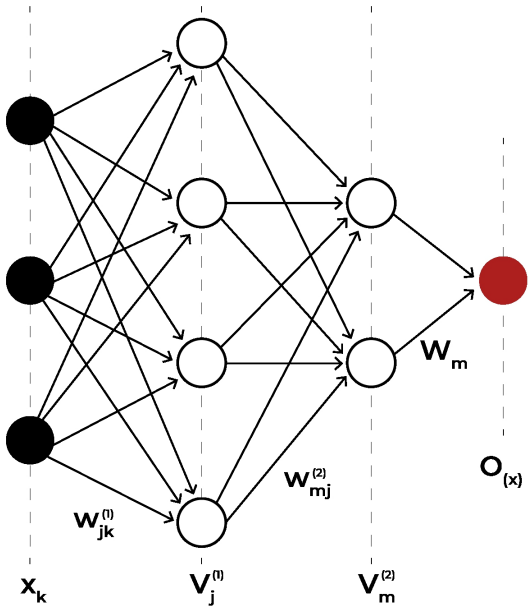

Figure 4: Single-layer Perceptron Schematic.

In Figure 4, for example, a two-layer network might approximate $\mathbf{t}(\mathbf{x})$ by

$$O(\mathbf{x}) \;=\; \sum_{m} W_m \; g\Bigg(\sum_{j} w^{(2)}_{mj} \; g\Big(\sum_{k} w^{(1)}_{jk} \; x_k - \theta^{(1)}_j\Big) \;-\; \theta^{(2)}_m\Bigg) \;-\; \Theta. \tag{3.2}$$

For *one-dimensional* inputs ($N = 1$), it is known that a *single hidden layer* of sigmoid neurons can approximate any continuous function arbitrarily well, provided there are sufficiently many hidden units. Under reasonable assumptions, this result - known as the *universal approximation theorem* (27) - extends to higher dimensions. Thus, on compact domains, one hidden layer is sufficient to approximate continuous functions.

3.1.1.2 Discrete Targets.

In the previous chapter, it was shown how *Boolean* functions of N binary inputs can be represented by a *single hidden layer* of 2^{N} McCulloch-Pitts neurons. For small N, this is possible; but, for high N, 2^{N} hidden units is excessive and becomes intractable. Still, one can build more effective solutions by using several hidden layers.

3.1.2 Depth Versus Width

While a single hidden layer can, in principle, approximate a wide variety of functions, additional hidden layers sometimes do so *more efficiently* or *more compactly.*For example, some problems can be solved using a smaller number of total neurons arranged in multiple layers:

- A naive single hidden layer approach might require 2^{N} neurons (Section 2.4).
- Carefully chaining smaller building blocks (e.g., XOR networks) yields a layered architecture of only $\mathcal{O}(\mathsf{N})$ neurons.

Deeper networks could therefore cut the required quantity of units or parameters.

3.2 Vanishing and Exploding Gradients

Vanishing and Exploding Gradients can occur during the backpropagation process when the gradients (partial derivatives of the loss function concerning the parameters) become very small (vanishing) or very large (exploding). Consider a network with L layers of hidden units (and possibly very large L). During training when the earliest layers parameters are adjusted via gradient descent, the error signals (i.e., the gradient information) may shrink exponentially as they propagate backward through multiple layers. Equivalently, if weights are large, small local errors can *explode* exponentially when viewed in earlier layers.

3.2.0.1 A Simple Example.

A conceptual toy model is a chain of scalar neurons,

$$v^{(\ell)} = g\Big(w^{(\ell)} v^{(\ell-1)} - \theta^{(\ell)}\Big), \quad \ell = 1, \dots, L, \tag{3.3}$$

with $v^{(0)} = x$ as input and $v^{(L)}$ as the final output. Taking the derivative $\partial v^{(L)}/\partial v^{(\ell)}$ involves a product of $L-\ell$ factors of $w^{(k)}\, g'\big(b^{(k)}\big)$. If these factors are each < 1 in absolute value, the gradient becomes tiny by the time it reaches layer ℓ, causing extremely slow weight updates (vanishing gradients). Similarly, if $|w^{(k)}\, g'(b^{(k)})| > 1$ on average, this product can explode. Either way, training becomes irregular and unreliable.

3.2.0.2 Partial Solutions.

Actually, numerous design or initialization techniques help to reduce gradient instability:

- **ReLU activation** (Section 3.2.1) can help avoid saturating gradients typical of sigmoid or tanh units.
- **Residual connections** (Section 3.2.2) skip layers entirely, offering shortcuts for error signals and thus reducing the number of multiplicative factors in the chain rule.
- **Batch normalization** (Section 3.2.3) maintains normalized activations at each layer, preventing large drift in internal states that exacerbates exploding or vanishing gradients.

Still, the vanishing/exploding gradient issue remains a fundamental obstacle in very deep networks and must be handled carefully.

3.2.1 Rectified Linear Units (ReLU)

Instead of sigmoid or tanh activations, many modern networks use *Rectified Linear Units*:

$$\mathrm{ReLU}(b) = \max\{0, b\}. \tag{3.4}$$

Such units have piecewise constant derivatives, which can reduce saturation effects. Specifically,

$$g'(b) = \begin{cases} 1, & b > 0, \\ 0, & b < 0, \end{cases} \tag{3.5}$$

so the gradient of a ReLU neuron is either 1 or 0 rather than an exponentially small value. This can lessen the vanishing-gradient problem, although exploding gradients can still arise if weight magnitudes become very large.

3.2.1.1 Sparsity.

Another benefit is that many ReLU outputs are exactly 0, producing sparse activation patterns across hidden layers. Sometimes sparsity increases linear separability in hidden-layer representations, hence accelerating learning.

3.2.1.2 Regularization.

Because ReLUs are unbounded for large positive inputs, weights may grow over time. A small *weight decay* (Section 3.2.4.1) or other regularization helps to constrain them. Empirical evidence has shown that ReLUs often train faster than their smooth, saturating counterparts.

3.2.2 Shortcuts: Residual and Skip Connections

Skip connections improve deep network architecture by directly connecting the output of some layer to the input of next ones, so bypassing intermediary connections between layers.

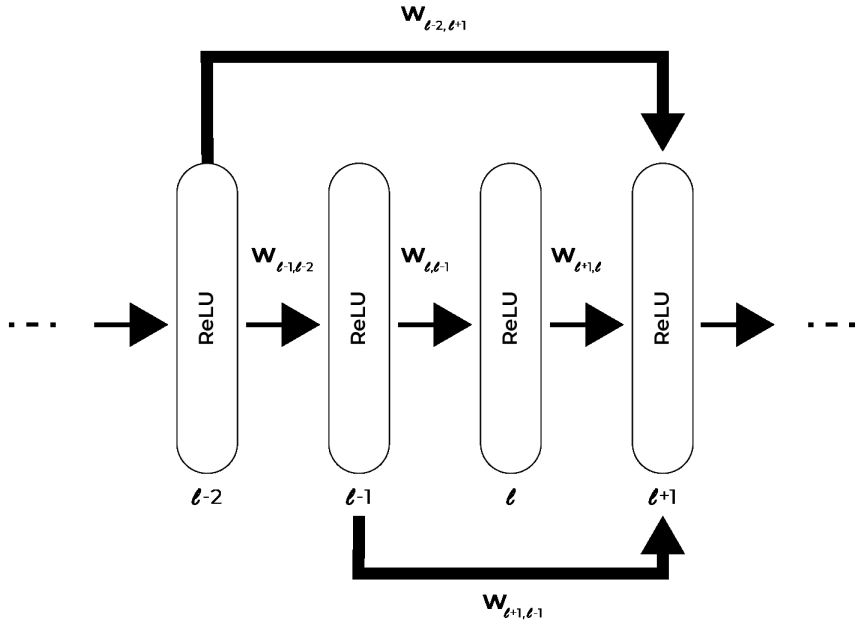

Figure 5: Residual and Skip Connection Schematic.

Figure 5 shows a simplified schematic: besides $w^{(\ell+1,\ell)}$ from layer ℓ to $\ell+1$, there is also the connection $w^{(\ell+1,\ell-2)}$ from layer $\ell-2$ and the connection $w^{(\ell+1,\ell-1)}$ from layer $\ell-1$. This connections adds more terms to the chain rule, therefore changing the backpropagation error computation. Effectively, there are now multiple paths through which gradients can flow backward, for example:

$$\delta^{(\ell-2)} \;=\; \delta^{(\ell-1)}\, w^{(\ell-1,\ell-2)}\, g'\big(b^{(\ell-2)}\big) \;+\; \delta^{(\ell+1)}\, w^{(\ell+1,\ell-2)}\, g'\big(b^{(\ell-2)}\big). \tag{3.6}$$

$$\delta^{(\ell-1)} \;=\; \delta^{(\ell)}\, w^{(\ell,\ell-1)}\, g'\big(b^{(\ell-1)}\big) \;+\; \delta^{(\ell+1)}\, w^{(\ell+1,\ell-1)}\, g'\big(b^{(\ell-1)}\big). \tag{3.7}$$

Because there is a *shorter* path to earlier layers, vanishing gradients become less severe. In *residual networks* (28), the sum

$$b_j^{(\ell+1)} \;=\; \sum_k w_{jk}^{(\ell+1,\ell)} V_k^{(\ell)} - \theta_j^{(\ell+1)} \;+\; V_j^{(\ell-2)} + V_j^{(\ell-1)}, \tag{3.8}$$

means layer $\ell+1$ receives both the standard feed-forward contribution and the outputs from layers $\ell-1$ and $\ell-2$. Many deep models in computer vision applications includes these residual approaches.

3.2.3 Batch Normalization

Batch normalization (29) extends the idea of input centering to hidden layers. Each mini-batch of layer inputs $\{V_j^{(\mu)}\}$ is centered and normalized:

$$\bar{V}_j^{(\mu)} \;=\; \frac{V_j^{(\mu)} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}, \tag{3.9}$$

where μ_j and σ_j^2 are the mean and variance computed over the mini-batch, $\epsilon > 0$ is a small constant, and j indexes neuron coordinates. Then, during the training process, the parameter γ_j, β_j are learned to allow the network to correctly restore scale and shift via the affine transformation $\gamma_j \bar{V}_j^{(\mu)} + \beta_j$. Batch normalization stabilizes layer distributions, often speeding training by mitigating internal covariate shift (a change in the input distribution that negatively affects the performance). Empirically, batch normalization reduces sensitivity to initial weights, addresses vanishing gradients, and can act like a regularizer.

3.2.4 Regularization

Deep networks are extremely *overparameterized*. Without constraints, they can overfit training data (Section 2.8), therefore compromising generalization. Additional techniques are employed to inhibit overfitting:

3.2.4.1 Weight Decay (L2 or L1 Regularization)

Weight decay modifies the energy function by adding a penalty proportional to the norm of the weights:

$$H' = H + \frac{\gamma}{2} \sum_{i,j} w_{ij}^2, \tag{3.10}$$

leading to an update rule

$$\delta w_{mn} = -\eta \frac{\partial H}{\partial w_{mn}} - \eta \gamma w_{mn}. \tag{3.11}$$

This is also called *L2-regularization*. In *L1-regularization*, one penalizes $\sum_{i,j} |w_{ij}|$ instead, shrinking smaller weights toward zero and often inducing sparser connections. Both

methods (and variants like max-norm constraints) can curb weight growth and limit overfitting.

3.2.4.2 Pruning

Pruning permanently removes weights or entire neurons deemed unnecessary after or during training. One approach is to set any small weight w_{ij} to zero, effectively deleting that connection, then skip updates to it (making the change irreversible). Pruning tends to reduce network size substantially, sometimes improving generalization by eliminating redundant parameters.

3.2.4.3 Dropout

During training, `dropout` randomly disables (sets outputs to zero) a fraction q of hidden neurons for each mini-batch iteration. Because the network is effectively forced to learn a robust representation in multiple smaller configurations, overfitting is reduced. During inference every neuron is active hence the output is scaled by $1 - q$ to match the expected activation levels. Dropout is used often with other regularizing techniques.

3.2.4.4 Data Augmentation

Usually, bigger training sets help to reduce overfitting. Although gathering additional data can be unfeasible, one can create artificial augmentation of a dataset by using minor changes (e.g., rotations, shifts, or noise addition). In image classification tasks, where small image differences are still valid samples that preserve labels but increase the dataset, this is very common.

3.3 Outputs and Energy Functions

Often, *different* activation functions are used in the output layer. For instance, *softmax* outputs are common when mapping inputs to one of M exclusive classes:

$$O_i = \frac{e^{b_i^{(L)}}}{\sum_{k=1}^{M} e^{b_k^{(L)}}}, \quad b_i^{(L)} = \sum_j w_{ij}^{(L)} V_j^{(L-1)} - \theta_i^{(L)}. \tag{3.12}$$

By construction, $O_i \geq 0$ and $\sum_{i=1}^{M} O_i = 1$, making O_i interpretable as a probability that input $\mathbf{x}$ belongs to class i. A suitable objective function is then the *negative log-likelihood*,

$$H = -\sum_{\mu} \sum_{i} t_i^{(\mu)} \ln O_i^{(\mu)}, \tag{3.13}$$

which vanishes if $O_i^{(\mu)} = t_i^{(\mu)}$ for all μ and i. Gradient-based learning yields the update rule

$$\delta w_{mn} = \eta \sum_{\mu} \left(t_m^{(\mu)} - O_m^{(\mu)}\right) V_n^{(\mu)}, \quad \delta\theta_m = -\eta \sum_{\mu} \left(t_m^{(\mu)} - O_m^{(\mu)}\right), \tag{3.14}$$

notably lacking an extra derivative $g'\left(b_m^{(L)}\right)$ factor, thereby avoiding some saturation-induced slowdowns.

This means that different networks with different purposes require a careful choice of network composition in terms of neurons, parameters, layers, dimensions, and activation functions. Once this is set, an appropriate Energy function to optimize must be carefully picked and minimized (or maximized) according to the training goal.

3.4 Summary

Multiple hidden layers define a *deep network* and allows significantly more precise function approximations than a shallow architecture of comparable scale. The primary challenges of the backpropagation-based training of such networks consist in:

1. **Vanishing or Exploding Gradients:** the repeated products of derivatives across layers the gradient may become quite small or quite large resulting in vanishing or exploding gradient. Unstable or slow convergence may follow from this.
2. **Overfitting:**Deep networks can easily fit the training set but fail to generalize, so regularizing methods are needed considering the number of employed parameters (weight decay, dropout, pruning, etc.).
3. **Architecture and Initialization:**Together with a proper weights initialization, well selecting the architecture structure, the activation function, and other network's parameters can help to reduce the gradient issue. Another commonly useful tool is batch normalizing.
4. **Dataset:** a big and high-quality dataset for a good training quality.
5. **Loss Function:** find a suitable and effective loss function determines the quality of the training process.

On some tasks - including vision, speech, and natural language processing - deep learning has outperformed conventional machine-learning techniques despite these challenges.

CHAPTER 4

CONVOLUTIONAL NETWORKS

Originally proposed in the 1980s, convolutional neural networks have become well-known for their performance on large-scale image classification problems. One turning point was made by Krizhevsky's work (30) in the ImageNet competition. Their method showed that while having much less trainable parameters than equivalent fully connected networks, convolutional architectures excel in object recognition. This involves two main benefits:

1. **Lower computational cost:** reduced learnable parameters via a sparse connection design and shared weights lowers computing cost, hence accelerating training.
2. **Regularization effect:** fewer connections limit the capacity of the network in a more efficient fashion, therefore reducing the inclination for overfitting.

Convolutional networks conceptually reflect several features of early visual processing in the brain. The network treat an input image as a 2D (or 3D, if color channels are included) spatial array, and detects *local features* (edges, corners, etc.). Convolutional layers apply the same local filters over all picture areas generating different *feature maps*.

4.1 Convolution Layers

Figure 6 depicts the core of a convolutional feature map. Assuming an input image of 10×10 pixels, a 3×3 *receptive field* (filter kernel) is slid over the image with unit stride in both horizontal and vertical directions. Every kernel position multiply a 3×3 square of local pixels by the kernel weights, then sum the result up (plus a threshold)

and pass it via an activation function $g(\cdot)$. Since the kernel slides one pixel at time, the produced *feature map* is 8×8 (in this example):

$$V_{ij} = g\Big(\sum_{p=1}^{3}\sum_{q=1}^{3} w_{pq}\, x_{p+i-1,\, q+j-1} - \theta\Big). \tag{4.1}$$

It is important to observe that, for *every* i, j, the *identical* kernel weights w_{pq} and threshold θ are reused. For this feature map, only $3 \times 3 + 1 = 10$ parameters are learned.

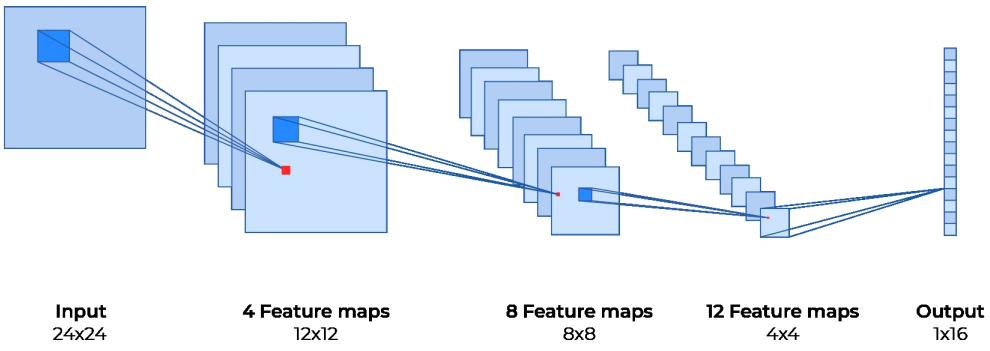

Figure 6: Convolutional Layer and Feature Maps Extraction Schematic.

4.1.1 General Form

Normally, convolution layers employ varying kernel sizes (3×3, 5×5, etc.) with a selected stride (s_1, s_2) and zero padding around the image. Padding the input, for example, helps to maintain its original spatial dimension, therefore a 10×10 input produces a 10×10 feature map instead of an 8×8 one. If the input image have R channels, each filter w^r_{pq} consists in R layers to address the depth dimension. This results in the general formula:

$$V_{i,j,k} = g\Big(\sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} w_{pqrk} \; x_{p+s(i-1),\, q+s(j-1),\, r} - \theta_k \Big), \tag{4.2}$$

where k indexes several feature maps that try to learn the various aspects of the image (edges, corners, textures, etc.). Modern deep architectures stack many convolution layers and additional operation (Section 4.2) in sequence.

4.1.1.1 Backpropagation with Shared Weights.

Training convolutional layers via gradient descent 1) involves computing $\partial H/\partial w_{mn}$ and $\partial H/\partial \theta$ as usual. Since the weight w_{mn} is shared across all kernel positions, the partial derivative accumulates contributions from *every* point (i, j) using w_{mn}. One kernel weight is applied to several receptive fields, so this weight sharing considerably lowers parameter counts and *increases* the effective amount of samples each weight sees, reducing overfitting.

4.2 Pooling Layers

Pooling layers (Figure 7) further reduce spatial dimensionality by *aggregating* value that are spatially close in a feature map. *Max-pooling*, for example, consider only the maximal value of the analyzed locations in the feature maps. *RMS pooling* averages local

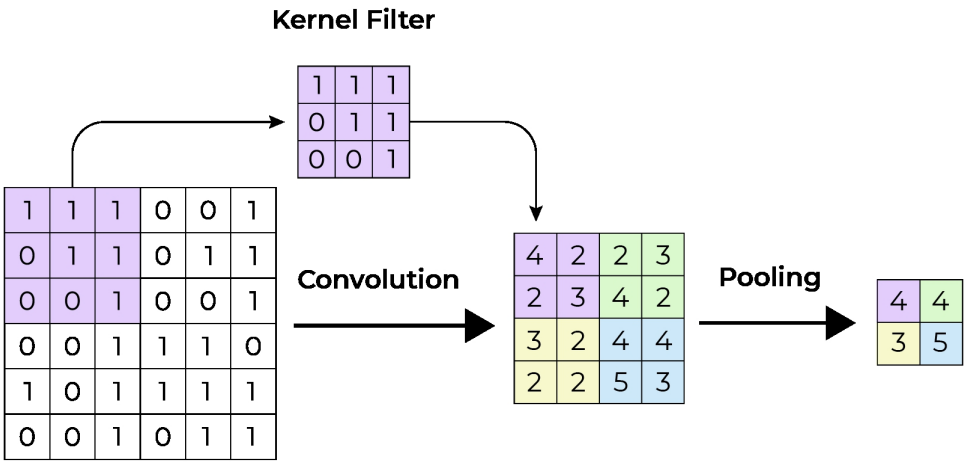

Figure 7: Convolutional and Pooling Operation Scheme.

squares instead; various variations are feasible. The pooling procedure just subsampling the convolutional outputs to produce a smaller, more compact representation; it has *no trainable parameters*.

4.2.0.1 Combination with Convolutional Layers.

Usually, a *convolution-pooling* cycle goes as:

1. With a given kernel dimension, stride and padding, the convolutional layer extracts feature maps.
2. A chosen pooling layer downsamples these feature maps.

3. This *convolution-pooling* cycles are repeated to generate higher-level features while lowering spatial resolution.
4. At the end, output neurons and completely connected layers can be added for classification.

Figure 8 shows an example: several convolutional layers with pooling layers and a fully connected layer using a softmax function outputs the probability of the input to belong to one of the different target categories.

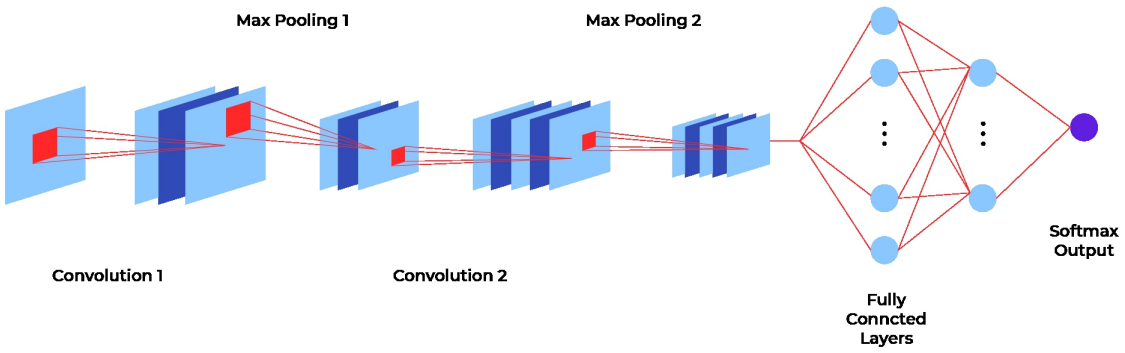

Figure 8: General Convolution Neural Network with a final fully connected layer.

4.3 ImageNet and Performance

Following Krizhevsky et al. (30) who achieved great results on the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC), convolutional networks became common. ImageNet (31) is a dataset of $\sim 10^7$ images labeled across thousands of categories. Top-5

accuracy measures performance by showing the frequency with which the correct label show in the top five projections of the model.

4.4 Practical Considerations and Limitations

Although in many large-scale picture applications convolutional networks show great accuracy, they present several drawbacks:

- **Lack of global understanding:** Local *patches* are identified separately by construction. While deep stacking of layers create somewhat global features, convolutional networks can still misclassify an entire object by confusing local textures or patterns.
- **Sensitivity to distribution shifts:** The learned filters may become useless if new images deviate significantly from the training distribution. Misclassifications can result from simple changes, including localized noise in various patterns.
- **Adversarial examples:** Little, undetectable changes to an image can push the model's representation past a decision boundary and generate high-confidence mistakes. These images show how convolutional networks could split feature space in unexpected ways.

Convolutional networks can outperform human-level performance on certain benchmark tests, however they lack the depth of knowledge and generalizing capacity shown by human vision. This differences become evident when images or noise deviate from training circumstances or when global context is essential.

4.5 Convolutional Neural Networks Summary

Image recognition has benne revolutionized by the advent of conventions neural networks. The main adcantages they bring are:

1. **Sparse Connectivity and Weight Sharing:** each kernel - that is, feature map - applies the same small set of weights over the whole input image, hence greatly lowering parameter counts and regularizing the model.
2. **Local Feature Extraction:** convolutional layers captures important visual structure by concentrating on local patterns like edges or corners.
3. **Pooling and Downsampling:** this techniques compress spatial data, speeding up training, and preserving significant characteristics.

Though the success in computer vision, medical imaging, and other disciplines is enormous, convolutional methods struggle with global context, out-of-distribution inputs, and adversarial perturbations. Even if the foundations set by convolutional architectures will probably continue to support next-generation vision models, many major challenges still surround how to better include more resilient invariances and more general context.

CHAPTER 5

COMPRESSION

5.1 Data Compression Overview

Data compression aims to more compactly represent information in a latent space. In computation, communication, and storage, it is a fundamental domain. Many survey publications have detailed compression techniques and applied them to different data types including text, image, audio, and video. Early work concentrated on broad mathematical foundations, though specific reviews were dedicated exclusively to lossless compression methods. Others provided performance evaluations of diverse DC algorithms, including Huffman coding, Lempel–Ziv (LZ) methods, and transform-based standards such as JPEG or MPEG. A general overview can be found in (5) While data storage and transmission technologies have advanced thanks to technical developments, the rising need for data transfer and storage keeps outpacing these developments.

For example, when streaming a video, data compression is extremely important. The bandwidth requirements vary significantly based on resolution, but even standard definition (SD) requires a minimum of 3Mbps, and for high definition (HD), it goes up to 5-8Mbps. 4K UHD streaming requires at least 25Mbps.

The same applies to smart home devices, IoT sensors, and portable medical equipment. These systems run with resources, memory, and power constraints. Effective data compression methods let these devices consistently transmit data without running out of their resources.

5.2 Information Theory

Information theory is the branch of study behind the mathematics of data compression. The information source is represented as a discrete memoryless source, which generates symbols from an alphabet $X = \{x_i\}_{i=1}^{M}$ with respective probabilities p_i. The objective is to facilitate the efficient encoding and transfer of information.

The information content associated with a symbol is inversely correlated with its probability, which means rarer symbols convey greater information density.

$$I(x_i) = \log_2 \frac{1}{p_i} = -\log_2 p_i \tag{5.1}$$

The average information per symbol is defined by the source's entropy, that represents a fundamental limit for lossless compression.

$$H(X) = \sum_{i=1}^{M} p_i I(x_i) = \sum_{i=1}^{M} p_i \log_2 \frac{1}{p_i} \tag{5.2}$$

5.2.1 Entropy Limits

Entropy provides a theoretical upper bound on the efficiency of lossless compression: coding schemes cannot reduce the average bit length below the source entropy, $H(X)$.

$$H(X) \leq R \tag{5.3}$$

5.2.2 Rate-Distortion Theory

Unlike lossless techniques, which have as top priority the integrity of the original signal, lossy algorithms have to consider the information loss. The quality of the recon-

structed signal, thereby the amount of information loss, is evaluated through distortion. Minimizing it while achieving a low bit rate is the goal.

A central theme in rate-distortion theory is to find an optimal tradeoff between rate and distortion that meets the requirements. To quantitatively examine this tradeoff, the bit rate can be expressed as a function of distortion levels for a given reconstruction quality. This can be difficult and often non-linear, therefore it is easier to work with analytical frameworks.

Inside the framework of discrete, independent, and identically distributed (i.i.d.) random processes, a distortion metric equation for the sequence $\hat{X}$ reconstructed from the original sequence X can be described as follows:

$$d(X, \hat{X}) = \frac{1}{m} \sum_{j=1}^{m} d(x_j, \hat{x}_j) \tag{5.4}$$

m is the sequence's length; $d(x_j, \hat{x}_j)$ is the per-component distortion, therefore indicating the degree of difference between the reconstructed signal $\hat{x}_j$ and the original one x_j. For example, the squared error $(x_j - \hat{x}_j)^2$ is commonly used as distortion.

An explicit formula for the rate-distortion curve can be obtained if the information sources follow a Gaussian distribution with zero mean and variance σ^2:

$$R(D) = \frac{1}{2} \log_2 \left(\frac{\sigma^2}{D} \right) \tag{5.5}$$

where $R(D)$ indicates the minimum rate needed to guarantee the existence of an information source able to describe X with a distortion level D.

5.3 Compression Pipeline Breakdown

The data compression pipeline consists of numerous sequential stages working together to lower data size while trying to preserve important information. Though particular implementations differ, the following describes the overall process:

1. **Input Signal (Continuous or Discrete):** if the source is continuous - for example for analog signals like audio or video - it must first be sampled to provide a discrete-time representation.
2. **Quantization (for Continuous-Valued Data):** after sampling, each discrete-time sample is associated to one of a finite set of amplitude levels. This quantizing step causes truncation or rounding, approximating the original sample.
3. **Modeling or Transform (Lossless/Lossy):** depending on the data type and application requirements, the compression system detects and exploits data redundancy:
    - **Sample Encoding:** variable-length coding use a smaller number of bits for data that appear more often.
    - **Spatial Correlation:** predictive models or transforms leverage correlations between neighboring pixels to avoid transmitting repetitive information.
    - **Temporal Correlation:** consecutive frames in a video or repeated sensor readings often share a great deal of overlap. By comparing these frames or readings, compression can discard duplicated data and improve efficiency.
4. **Residual Coding or Direct Coding:** once the model (or transform) has extracted the relevant features or predicted values, the algorithm encodes the residuals

(or transformed coefficients). In lossless scenarios, the exact data or residual must be preserved. In lossy methods, quantization may be more aggressive in discarding finer details that are less perceptually important.

5. **Entropy Coding:** finally, the data (or residuals) is passed to an entropy coder (e.g., Huffman or Arithmetic coding) to assign shorter codewords to the most frequent symbols. This step compacts the bitstream further based on the probabilities of the symbols.

6. **Compressed Output:** the result of entropy coding forms the compressed bitstream. This output can be stored or transmitted efficiently, requiring fewer bits than the original data.

7. **Decoding (Reconstruction):** The receiving side applies the inverse procedures: entropy decoding, inverse modeling/transform or prediction, and potential dequantization (for lossy methods). The final output is an approximation of the original data ($\hat{X}$).

5.3.1 Data Compression categorizations

In addition to the amount of permissible data loss, compression techniques can also be grouped based on the particular type of data they're designed to manage.

5.3.1.1 Lossless or Lossy compression:

The accuracy of the reconstructed data classifies the compression techniques:

- **Lossless Compression:** guarantees that X can be completely reconstructed from the compressed data Y. In applications like text processing, financial transactions, and medical imaging, this is important.

- **Lossy Compression:** allows an approximation of the original data ($\hat{X} \approx X$) by allowing some error levels as long as it stays imperceptible by consumers. Since they often result in greater compression ratios, lossy approaches are chosen in applications where losing a minor amount of the information is acceptable.

5.3.1.2 Text Compression:

Text compression is usually lossless since even a single-character difference may affect the semantic meaning. Among popular methods are Lempel-Ziv (LZ) variations, Arithmetic coding, and Huffman.

5.3.1.3 Audio and Video Compression:

Audio compression often exploits psychoacoustic models to remove inaudible or less-perceptible frequencies. Common examples are AAC or MP3, where enhanced compression ratios are allowed by partial data removal. Video compression seeks the redundancy existing temporally across frames and spatially within a single frame.

5.3.1.4 Image Compression:

Given the enormous amounts of visual data produced by current technology, image compression is very important. In this context, the techniques used are often clustered under transform- or predictive-based approaches:

- In *transform-based*, after applying quantization and entropy coding, pictures from the spatial domain are converted to another domain - such as frequency. While more recently, wavelet-based techniques (e.g., JPEG 2000) provide progressive transmission and higher rate-distortion performance for some image classes, standards like JPEG utilize a blockwise Discrete Cosine Transform (DCT) approach.

- *Predictive* or *residual-based* uses the relationship among adjacent pixels to predict information from already processed data. The difference-that is, the prediction error, is encoded.

5.3.2 Sampling and Quantization:

Data compression requires two preliminary stages:

I the sampling frequency is lower, the resulting discrete-time representation may suffer aliasing from insufficient sampling rate, resulting in distortion when reconstructed. The problem is that any frequency content above $f_s/2$ reflects back into the base-band generating overlapping spectral components and notable distortion during reconstruction.

- **Sampling:** creates several discrete-time sample sequences from a continuous-time source. Let a continuous-time signal be $x(t)$, than the sampling procedure generates $x[n] = x(nT)$, where T is the sample interval and n is an integer index.
- **Quantization:** convert the continuous amplitude of the sampled signal into a limited range of discrete values. This phase lowers amplitude-related redundancy and turns continuously valued signals into codewords.

The *Nyquist-Shannon* sampling theorem is extremely important to reconstruct $x(t)$ without aliasing after being sampled. The theorem states that the sampling frequency $f_s = \frac{1}{T}$ must be at least twice the highest frequency component present in the signal.

5.3.2.1 Quantization: Characteristics of Errors

Once a signal is sampled, each sample $x[n]$ is encoded with a finite set of discrete amplitude values, a process called quantization.

The output of a mid-riser quantizer using Δ as the step size for a sample $x[n]$ can be written as

$$Q(x[n]) = \Delta \left\lfloor \frac{x[n]}{\Delta} + \frac{1}{2} \right\rfloor. \tag{5.6}$$

The real axis is discretized into uniform width intervals Δ. The quantity

$$e[n] = x[n] - Q(x[n]) \tag{5.7}$$

is the quantization error. $e[n]$ can be approximated as a white noise-like process with dynamics $\pm\frac{\Delta}{2}$ for sufficiently precise quantization steps and certain assumptions about the signal distribution.

5.3.2.2 Uncertainty in the Quantization Process

Since implies rounding or truncation by nature, the quantization process generates uncertainty about the original value. A uniform quantizer has a mean-squared quantization error of about:

$$\sigma_e^2 \approx \frac{\Delta^2}{12}, \tag{5.8}$$

This model assumes high number of quantization levels and an evenly distributed input signal throughout one quantization interval.

Normally signals depart from these basic assumptions, however this models can be used when the quantization steps are small compared to the signal dynamics.

5.3.2.3 Practical Implications

Sampling and quantization are also closely related to hardware and software limitations:

- **Hardware Constraints:** the precision of the quantizer step Δ is affected by resolution of the used analog-to-digital converter (ADC). Furthermore, system cost, analog bandwidth, and power consumption affect the maximum possible sampling frequency.
- **Data Rates and Storage:** higher sampling frequency and more precise quantization produce more accurate results but require more bits per sample, therefore leads to higher data storage or transmission requirements. Although compression methods lower the effective bit rate, it must be carefully balanced with the distortion and uncertainty this techniques introduce.
- **Applications:** in applications where low distortion is paramount, lossless compression is chosen. On the other hand, in settings where modest distortion is tolerated in order to use less bandwidth and reduce costs, lossy techniques are used.

5.3.3 Entropy Coding

Entropy coding is the final stage of the compression pipeline, where the processed signal is translated into a sequence of bits.The main idea is to assign each symbol a_i in the alphabet $X = \{a_i\}_{i=0}^{N-1}$- which is simply the complete set of distinct symbols forming the data stream- a concise binary codeword $c(a_i)$, according to the symbol probability $p(a_i) = p_i$. The idea is to assign shorter codewords to symbols that appear commonly, decreasing the average code length. Two techniques are:

- **Huffman Coding:** it assigns shorter codewords to encode symbols that appears with higher probabilities, producing a prefix-free binary tree[1].

---

[1]A prefix-free binary tree is a structure guaranteeing that no code is the prefix of another, so symbols can be unambiguously decoded without lookahead.

- **Arithmetic Coding:** it encodes symbol sequences with intervals within $[0, 1]$. The algorithm continually refines the interval based on symbol probabilities.

5.4 Quality Metrics

Compression algorithm are evaluated in terms of speed and quality of reconstructed data through the following metrics:

- **Compression Level:** this is quantified by the compression ratio, which indicates the average number of bits needed to encode a sample (bit rate).
- **Distortion:** distortion quantifies the deviation between the original data and its reconstructed form.

5.4.1 Metrics for Compression Quality and Performance

The compression ratio indicates the average number of bits needed to encode each sample.

$$\mathrm{CR} = \frac{S_{\mathrm{orig}}}{S_{\mathrm{comp}}} \tag{5.9}$$

S_{orig} is the original data size (in bits) and S_{comp} the compressed size.

Bits-per-pixel (bpp) and *bits-per-character* (bpc) are respectively the standards used to show how many bits are needed to encode every pixel or character.

Space Savings show the reduction brought by the compression in terms of file size:

$$\text{Space Savings} = 1 - \left(\frac{S_{\mathrm{comp}}}{S_{\mathrm{orig}}}\right). \tag{5.10}$$

For example, if the original file is large 10MB and it is compressed to 2MB, it means 0.80 (or 80%) of storage space is preserved.

Compression speed can be measured by cycles per byte (CPB), which shows how many CPU cycles are needed for each byte of data processed.

Mean Squared Error (MSE) or *Root Mean Squared Error* (RMSE) measures the distorion of the reconstructed signal.

$$\mathsf{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2 \tag{5.11}$$

$$\mathsf{RMSE} = \sqrt{\mathsf{MSE}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2} \tag{5.12}$$

Peak Signal-to-Noise Ratio (PSNR) is another criteria to evaluate the distortion of the reconstructed signal. It calculates the ratio of a signal's maximum power to its power noise:

$$\mathsf{PSNR} \;=\; 20\,\log_{10}\!\left(\frac{\mathsf{MAX}\{X\}}{\mathsf{RMSE}}\right), \tag{5.13}$$

where MAX_X is the maximum intensity value in the original data.

Structural Similarity Index (SSIM) evaluate perceptual factors such as luminance, contrast, and structural details, providing a more human-centric perspective on image fidelity.

CHAPTER 6

TRANSFORM CODING FOR IMAGE COMPRESSION WITH SCALE HYPERPRIOR

6.1 Introduction to Transform Coding

An effective data compression system compresses data while preserving essential information, enabling accurate reconstruction. Transform coding it is a common used technique, it use a parametric transformations to transform high-dimensional data into a latent representation, facilitating efficient storage and transmission.

Firstly, the image vector $\mathbf{x}$ is encoded into a latent representation $\mathbf{y}$. This representation is then quantized to produce $\hat{\mathbf{y}}$. Before transmission, entropy coding methods are employed.

During the decoding process, these steps are reversed: first, $\hat{\mathbf{y}}$ is recovered from the compressed bitstream, and then it is transformed back into an approximate reconstruction of the original image, denoted as $\hat{\mathbf{x}}$, using a synthesis transform. The transforms function used ((g_a and g_s) are parameterized as generic nonlinear functions. The parameters ϕ_g and θ_g represent the weights and biases of these networks.

Since this procedure introduces inaccuracies, a balancing solution to the rising rate-distortion problem needs to be found:

- **Distortion (D):** characterizes the difference between the original image $\mathbf{x}$ and its reconstruction $\hat{\mathbf{x}}$.
- **Rate (R):** accounts for the bit rate of the compressed representation, the higher the value the higher are the number of bit required and therefore the lower is the

distortion. On the other hand, a lower value implies a lower number of bit and an higher distortion. The rate can be expressed as:

$$R = \mathbb{E}_{\mathbf{x}\sim p_{\mathbf{x}}} \left[-\log_2 p_{\hat{\mathbf{y}}}(Q(g_a(\mathbf{x}; \phi_g))) \right], \tag{6.1}$$

, Q is the quantization function and $p_{\hat{\mathbf{y}}}$ predicts how likely each quantized latent value is. It helps estimate the number of bits needed to encode those values - more accurate $p_{\hat{\mathbf{y}}}$ means better compression.

The transforms g_a and g_s influence both Rate and distortion by modulating the quantization granularity, how finely the latent representation is (quantized - finer steps require more bits but incur lower distortion), and the representation warping, describes how g_a reshapes or "warps" the input into a latent space before quantization, and how g_s reconstructs the signal afterward. An increase in the Rate generally reduces distortion, while the reverse also holds. This trade-off can be parameterized by a coefficient λ, resulting in:

$$\min_{\phi_g, \theta_g} \mathbb{E}_{\mathbf{x}\sim p_{\mathbf{x}}} \left[\lambda D(\mathbf{x}, \hat{\mathbf{x}}) + R(\hat{\mathbf{y}}) \right]. \tag{6.2}$$

6.1.1 Optimization Challenges

Optimizing the transform parameters (ϕ_g and θ_g) presents difficulties mainly related to the non-differentiability brought by quantization. The gradient of the quantization function is zero almost everywhere, making optimization via gradient descent useless.

The following approximations are used:

- **Gradient Substitution:** a suitable approximation of the gradient of the quantizer is implemented (11).

- **Uniform Noise Approximation:** Additive uniform noise replaces the quantization step during the training phase (10). While preserving quantization integrity during inference, this method helps gradient approximation. The noisy and quantized representation are respectively $\tilde{\mathbf{y}}$ and $\hat{\mathbf{y}}$.

The optimization problem can be reinterpreted inside a variational autoencoder (VAE) framework:

- The **analysis transform** functions estimates the latent representation $\tilde{\mathbf{y}}$ from the input $\mathbf{x}$.
- The **synthesis transform** tries to reconstruct $\hat{\mathbf{x}}$ from $\tilde{\mathbf{y}}$, operating similarly to a generative model.

In probabilistic modeling for compression, the goal is to minimize the Kullback-Leibler (KL) divergence between the approximate posterior $q(\tilde{y}|x)$ and the true posterior $p(\tilde{y}|x)$. Bayes' theorem's true posterior $p(\tilde{y}|x) = \frac{p(x|\tilde{y})p(\tilde{y})}{p(x)}$, represents the exact probability distribution of $\tilde{y}$ given x. The true posterior is usually difficult to use given the marginal likelihood $p(x)$. instead, a computationally effective alternative is a parameterized approximation posterior $q(\tilde{y}|x)$. The KL divergence:

$$\mathrm{KL}(q(\tilde{y}|x)\|p(\tilde{y}|x)) = \int q(\tilde{y}|x) \log \frac{q(\tilde{y}|x)}{p(\tilde{y}|x)} d\tilde{y}, \tag{6.3}$$

is minimized in order to find an approximate posterior $q(\tilde{y}|x)$ that is optimized to be as similar as possible to $p(\tilde{y}|x)$. This ensures efficient encoding while preserving fidelity, aligning the learned model with the compression objective.

To measure how well the approximate posterior $\mathsf{q}(\tilde{\mathbf{y}}|\mathbf{x})$ aligns with the true posterior $\mathsf{p}(\tilde{\mathbf{y}}|\mathbf{x})$ on average, the expectation of the KL divergenze is minimized, ensuring that the overall "distance" (in KL sense) across the entire data distribution is reduced, rather than just for a single data value:

$$\mathbb{E}_{\mathbf{x}\sim\mathsf{p}_\mathbf{x}}\mathrm{KL}\left[\mathsf{q}(\tilde{\mathbf{y}}|\mathbf{x})\|\mathsf{p}(\tilde{\mathbf{y}}|\mathbf{x})\right] \tag{6.4}$$

Observe the definition of the KL divergence involves an expectation with respect to $\tilde{\mathbf{y}}$ drawn from the approximate posterior $\mathsf{q}(\tilde{\mathbf{y}}|\mathbf{x})$. Thus, for each data point $\mathfrak{x}$, the mismatch between $\mathsf{q}(\tilde{\mathbf{y}}|\mathbf{x})$and $\mathsf{p}(\tilde{\mathbf{y}}|\mathbf{x})$ must be integrated over all latent-variable values $\tilde{\mathbf{y}}$. As so, the goal to be minimized turns into:

$$\mathbb{E}_{\mathbf{x}\sim\mathsf{p}_\mathbf{x}}\mathrm{KL}\left[\mathsf{q}(\tilde{\mathbf{y}}|\mathbf{x})\|\mathsf{p}(\tilde{\mathbf{y}}|\mathbf{x})\right] = \mathbb{E}_{\mathbf{x}\sim\mathsf{p}_\mathbf{x}}\mathbb{E}_{\tilde{\mathbf{y}}\sim\mathsf{q}}\left[-\log\mathsf{q}(\tilde{\mathbf{y}}|\mathbf{x}) + \log\mathsf{p}_{\mathbf{x}|\tilde{\mathbf{y}}}(\mathbf{x}|\tilde{\mathbf{y}}) + \log\mathsf{p}_{\tilde{\mathbf{y}}}(\tilde{\mathbf{y}})\right] + \mathsf{C}. \tag{6.5}$$

- The term $-\log\mathsf{q}(\tilde{\mathbf{y}}|\mathbf{x})$ is zero since the latent variable $\tilde{\mathfrak{y}}$ is modeled by taking the true latent $\mathfrak{y}$ and adding *constant-width uniform noise*[1] and therefore the resulting density $\mathsf{q}(\tilde{\mathbf{y}}|\mathbf{x})$ is uniform over a fixed interval.
- The term $-\log\mathsf{p}_{\mathbf{x}|\tilde{\mathbf{y}}}$ is the **weighted distortion**, representing the reconstruction error.
- The term $-\log\mathsf{p}_{\tilde{\mathbf{y}}}$ is the **rate**, representing the bit cost of encoding $\tilde{\mathbf{y}}$.

---

[1]This uniform-noise approximation is introduced because true quantization step is *not* differentiable, making it incompatible with gradient-based optimization. By replacing it with uniform noise during training, the model's parameters can be updated via gradients. At test time, one can still perform actual quantization.

To optimize compression, an accurate model for the entropy distribution $p_{\tilde{y}}$ is essential. One common approach is to use a factorized prior model:

$$p_{\tilde{y}}(\tilde{y}) = \prod_i \left[p_{y_i}(\psi_i) * U\left(-\frac{1}{2}, \frac{1}{2}\right) \right] (\tilde{y}_i), \tag{6.6}$$

where ψ_i are parameters learned to represent each univariate distribution p_{y_i}. This approach simplifies the model but may fail to capture dependencies between latent variables.

6.2 Learned Analysis and Synthesis Transforms

This section details the transforms used for compressing and then reconstruction the image. Emphasizing the flow of data through convolution, down/upsampling, and nonlinear normalizing, the diagrams in Figure 10 depict the forward and inverse processes. The block g_a and g_s represent the autoencoder architecture, instead the autoencoder related to the hyperprior correspond to h_a and h_s. The block Q represents quantization, while AE, AD and CB represent respectively arithmetic encoder, arithmetic decoder and the compressed bit-stream.

There has traditionally been a preference for orthogonal linear transforms (e.g., block transforms or wavelets) to reduce correlations in image signals before entropy coding. Linear transforms alone cannot completely handle higher-order relationships; hence, using nonlinear operations helps to improve decorrelation even further (8). Inspired by local normalization mechanisms in biological vision, the approach in (9) proposes a generalized divisive normalization (GDN) to reduce statistical dependencies in natural images. The synthesis stage then uses an invertible form of that normalization (IGDN), combined with the complementary operations in reverse order.

6.2.1 Forward Analysis Transform

The forward (analysis) transform, denoted by g_a, aims to map an input image $\mathbf{x}$ to a latent representation $\mathbf{y}$. As shown on the left side of Figure 10, the process is divided into three stages of convolution, subsampling, and divisive normalization. In the kth stage, the input is a set of feature maps $\{u_j^{(k)}\}$, and the output is $\{u_i^{(k+1)}\}$. Each stage begins with an *affine convolution*:

$$v_i^{(k)}(m,n) \;=\; \sum_j \big(h_{k,i,j} * u_j^{(k)}\big)(m,n) \;+\; c_{k,i}, \tag{6.7}$$

where $h_{k,i,j}$ denotes the convolution filters for the kth stage, $c_{k,i}$ is a learned bias, and (m,n) indexes spatial positions. Next, *downsampling* is performed:

$$w_i^{(k)}(m,n) \;=\; v_i^{(k)}\big(s_k m,\; s_k n\big), \tag{6.8}$$

where s_k is the integer downsampling factor for stage k. Finally, the *generalized divisive normalization* (GDN) acts elementwise on these downsampled outputs:

$$u_i^{(k+1)}(m,n) \;=\; \frac{w_i^{(k)}(m,n)}{\Big(\beta_{k,i} \;+\; \sum_j \gamma_{k,i,j}\,\big(w_j^{(k)}(m,n)\big)^2\Big)^{\frac{1}{2}}}, \tag{6.9}$$

where $\beta_{k,i} > 0$ and $\gamma_{k,i,j} \geq 0$ are trainable parameters. Repeated over three successive stages, the transform progressively reduces spatial resolution and normalizes feature maps to remove correlations.

The complete set of parameters$\{h,\, c,\, \beta,\, \gamma\}$ across all stages is typically denoted as the analysis parameter vector $\boldsymbol{\phi}$. These parameters are jointly learned during training.

6.2.2 Inverse Synthesis Transform

The inverse (synthesis) transform, denoted by g_s, execute the same operation of the analysis pipeline in the reverse order. It receives a latent code $\hat{\mathbf{y}}$ and attempts to reconstruct an approximation $\hat{\mathbf{x}}$ of the input image. As shown on the right side of Figure 10, the operations within each stage are reversed in order: an invertible normalization (IGDN) is followed by upsampling and then an affine convolution. For the kth stage, the intermediate variables are indexed similarly by (m, n).

The *invertible GDN* (IGDN) operation is specified by

$$\widehat{w}_i^{(k)}(m,n) \;=\; \frac{\widehat{u}_i^{(k)}(m,n)}{\left(\widehat{\beta}_{k,i} + \sum_j \widehat{\gamma}_{k,i,j}\left(\widehat{u}_j^{(k)}(m,n)\right)^2\right)^{\frac{1}{2}}}, \tag{6.10}$$

where $\widehat{u}_i^{(k)}$ denotes the input tensor to this IGDN stage, and $\{\widehat{\beta}, \widehat{\gamma}\}$ are the learned normalization parameters. Next, *upsampling* is performed:

$$\widehat{v}_i^{(k)}(m,n) \;=\; \begin{cases} \widehat{w}_i^{(k)}\left(\frac{m}{\widehat{s}_k}, \frac{n}{\widehat{s}_k}\right), & \text{if } \frac{m}{\widehat{s}_k} \text{ and } \frac{n}{\widehat{s}_k} \text{ are integers,} \\ 0, & \text{otherwise,} \end{cases} \tag{6.11}$$

where $\widehat{s}_k$ is the corresponding upsampling factor (matching s_k from the forward pass). Finally, the *affine convolution* for reconstruction is

$$\widehat{u}_i^{(k+1)}(m,n) \;=\; \sum_j \left(\widehat{h}_{k,i,j} * \widehat{v}_j^{(k)}\right)(m,n) \;+\; \widehat{c}_{k,i}. \tag{6.12}$$

These steps repeat for three stages until a final reconstructed image $\hat{\mathbf{x}}$ is obtained. The inverse parameters $\{\widehat{h}, \widehat{c}, \widehat{\beta}, \widehat{\gamma}\}$ are collected into the synthesis vector θ, optimized jointly with the forward parameters.

6.3 Overview of the Scale Hyperprior

In conventional transform coding frameworks, significant spatial dependencies are present among the components of the latent representation $\hat{\mathbf{y}}$. The integration of an hyperprior (as shown in Figure 9), as proposed in (7) that is an extension of the architecture presented in (9), helps the architecture to better model the dependencies through the implementation of extra layers to enhance the parameter ψ_i estimation. This addition carries an extra set of random variables, $\tilde{\mathbf{z}}$, used to accurately model these dependencies.

Once $\tilde{\mathbf{z}}$ is known, it essentially controls the distribution of $\tilde{\mathbf{y}}$, hence making the elements of $\tilde{\mathbf{y}}$ conditional independent. Each element of $\tilde{\mathbf{y}}$ thus has a simpler, independent distribution depending on a given $\tilde{\mathbf{z}}$. $\tilde{\mathbf{z}}$ helps the model to precisely account for spatially changing scales in $\tilde{\mathbf{y}}$, hence improving expressiveness of the representation.

The latent representation $\tilde{\mathbf{y}}$ is made of different $\tilde{y}_i$, each of them is modeled as a Gaussian distribution with zero-mean and standard deviation σ_i. These standard deviations are predicted using the transform h_s applied to $\tilde{\mathbf{z}}$:

$$p_{\tilde{\mathbf{y}}|\tilde{\mathbf{z}}}(\tilde{\mathbf{y}}|\tilde{\mathbf{z}}, \theta_h) = \prod_i \left[\mathcal{N}\left(0, \tilde{\sigma}_i^2\right) * \mathcal{U}\left(-\frac{1}{2}, \frac{1}{2}\right)\right](\tilde{y}_i), \tag{6.13}$$

where $\tilde{\boldsymbol{\sigma}} = h_s(\tilde{\mathbf{z}}; \theta_h)$ represents the spatially varying standard deviations, and $\mathcal{U}$ denotes a standard uniform distribution.

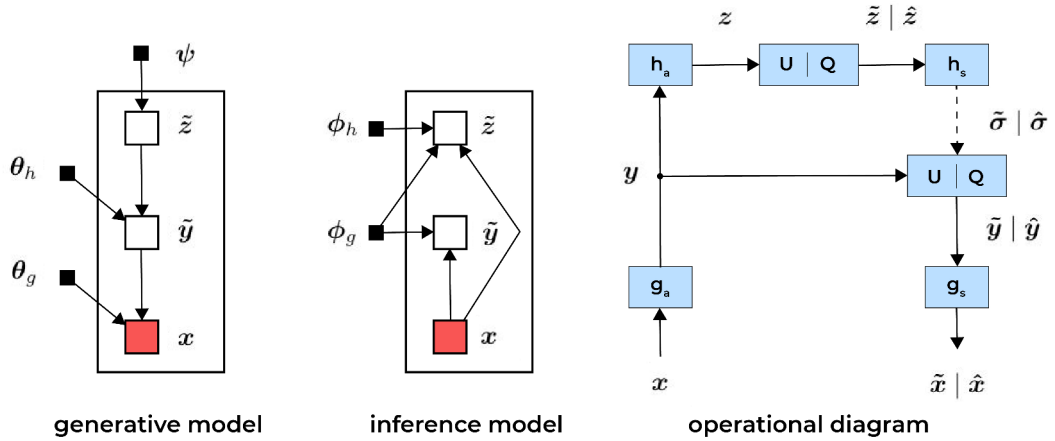

Figure 9: Transform coding model with the integration of the hyperprior and the operational workflow of the model.

An additional parametric transform h_a is applied atop $\mathbf{y}$, resulting in a single joint variational posterior:

$$q(\tilde{\mathbf{y}}, \tilde{\mathbf{z}}|\mathbf{x}, \phi_g, \phi_h) = \prod_i U\left(\tilde{y}_i|y_i - \frac{1}{2}, y_i + \frac{1}{2}\right) \cdot \prod_j U\left(\tilde{z}_j|z_j - \frac{1}{2}, z_j + \frac{1}{2}\right), \tag{6.14}$$

where $\mathbf{y} = g_a(\mathbf{x}; \phi_g)$ and $\mathbf{z} = h_a(\mathbf{y}; \phi_h)$.

The latent representation $\mathbf{y}$ efficiently captures the spatial distribution of standard deviations, and enables a stronger modeling of the dependencies.

Similarly to $\tilde{\mathbf{y}}$, the hyperprior $\tilde{\mathbf{z}}$ is modeled as:

$$p_{\tilde{\mathbf{z}}|\psi}(\tilde{\mathbf{z}}|\psi) = \prod_i \left[p_{z_i|\psi(i)} \left(\psi(i)\right) * \mathcal{U}\left(-\frac{1}{2}, \frac{1}{2}\right) \right] (\tilde{z}_i), \tag{6.15}$$

where $\psi(i)$ are parameters containing each univariate distribution $p_{z_i|\psi(i)}$.

The modified expectation of KL divergence to be minimized is:

$$\mathbb{E}_{\mathbf{x}\sim p_{\mathbf{x}}} \mathbb{E}_{\tilde{\mathbf{y}},\tilde{\mathbf{z}}\sim q} \left[\log q(\tilde{\mathbf{y}}, \tilde{\mathbf{z}}|\mathbf{x}) - \log p_{\mathbf{x}|\tilde{\mathbf{y}}}(\mathbf{x}|\tilde{\mathbf{y}}) - \log p_{\tilde{\mathbf{y}}|\tilde{\mathbf{z}}}(\tilde{\mathbf{y}}|\tilde{\mathbf{z}}) - \log p_{\tilde{\mathbf{z}}}(\tilde{\mathbf{z}}) \right] + C \tag{6.16}$$

- The first term ($\log q$) is approximated to zero due to the uniform densities of unit width of q as explained in the previous section 6.1.1.
- The second term ($-\log p_{\mathbf{x}|\tilde{\mathbf{y}}}$) represents distortion.
- The third and fourth terms ($-\log p_{\tilde{\mathbf{y}}|\tilde{\mathbf{z}}}$ and $-\log p_{\tilde{\mathbf{z}}}$) encode the cross-entropy costs for $\tilde{\mathbf{y}}$ and $\tilde{\mathbf{z}}$, respectively. The fourth term can be interpreted as side information.

6.3.1 Compression Process with a Scale Hyperprior

The compression process is as follows:

1. The encoder generates spatially variable standard deviation $\mathbf{y}$ applying the analysis transform g_a to the input $\mathbf{x}$.
2. The information about these standard deviations is elaborated by the transform h_a, resulting in $\mathbf{z}$.
3. $\mathbf{z}$ is quantized to produce $\hat{\mathbf{z}}$, which is compressed and transmitted as side information.

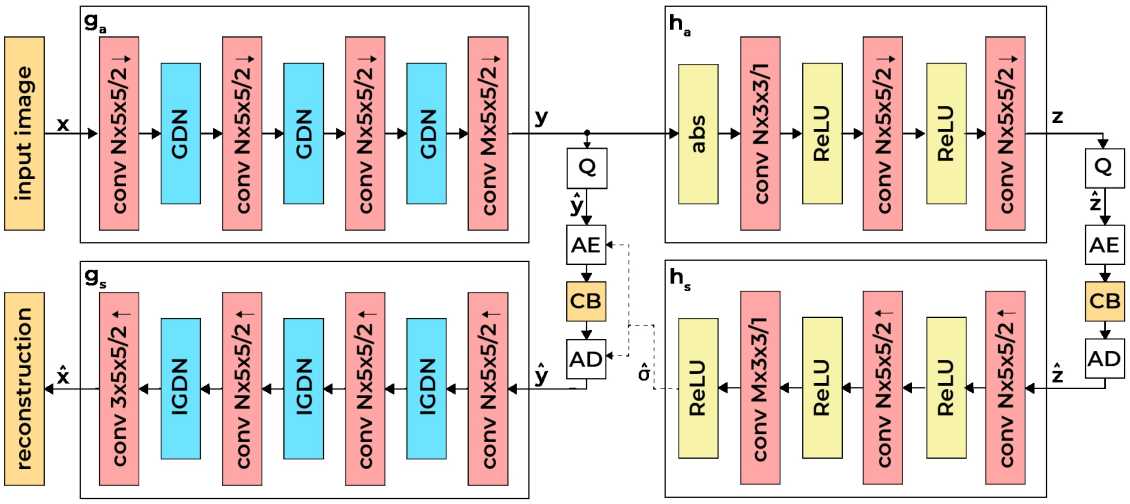

Figure 10: Architecture of the Scale Hyperprior model.

4. The decoder reconstructs $\hat{\mathbf{z}}$ from the transmitted signal and applies h_s to estimate $\hat{\sigma}$, the spatial distribution of standard deviations.

5. Using $\hat{\sigma}$, the decoder recovers $\hat{\mathbf{y}}$, which is then fed into the synthesis transform g_s to reconstruct the image $\hat{\mathbf{x}}$.

6.4 Conclusion

Integrating variational autoencoding with transform coding achieves state-of-the-art performance in lossy compression applications.

The scale hyperprior enhances traditional transform coding: it models spatial dependencies through latent variables, which allows the model to capture these dependencies more effectively in the latent representation. The architecture, leveraging on side infor-

mation $\hat{\mathbf{z}}$ modeling, enhances not only rate-distortion performance but also provides a robust mechanism for adaptive image compression.

CHAPTER 7

IMAGE DENOISING

Image denosing has progressed a lot in the years, starting from the classical approach where the theoretical, mathematical and analytical frameworks were the basis for every technique to the new upcoming methods guided by the breakthrough of deep learning. In the following a quick overview of the most importat step in the history of image denoisy is presented, drawing inspiration from (13).

7.1 Problem Definition

The image denoising task tries to recover a clean image$\mathbf{x} \in \mathbb{R}^N$ from a noisy observation $\mathbf{y} \in \mathbb{R}^N$, by designing a denoiser $\hat{\mathbf{x}} = D(\mathbf{y}, \sigma)$ that approximates the true image $\mathbf{x}$ given $\mathbf{y}$ and the noise level σ. The noisy data is:

$$\mathbf{y} = \mathbf{x} + \mathbf{v}, \tag{7.1}$$

where the noise $\mathbf{v} \in \mathbb{R}^N$ is modeled as a zero-mean independent and identically distributed (i.i.d.) Gaussian noise with variance σ^2, denoted as $\mathbf{v} \sim \mathcal{N}(0, \sigma^2\mathbf{I})$.

The Mean Squared Error (MSE), compute the average square difference between the clean image and the reconstructed one, evaluating the quality of the denoising method used:

$$\text{MSE} = \mathbb{E}\left[\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2\right] = \mathbb{E}\left[\|\mathbf{x} - D(\mathbf{y}, \sigma)\|_2^2\right], \tag{7.2}$$

If the average square difference between the clean image and the reconstructed one is minimal, meaning if the MSE is minimized, the denoisiner that can recover x is optimal:

$$\hat{\mathbf{x}}_{\text{MMSE}} = \mathbb{E}[\mathbf{x}|\mathbf{y}], \tag{7.3}$$

where MMSE stands for Minimum Mean Squared Error. It is difficult to implement a denoiser that efficiently achieves minimum mean square error in practice.

7.1.1 Challenges in Image Denoising

The simplicity of the problem definition is deceptive. Different filtering methods are able to lower noise. However, they degrade image content such as edges, textures, and fine details. Effective denoising requires a careful design and algorithms to balance noise reduction with preservation of important visual structures.

7.1.2 The Gaussianity Assumption

The above formulation assumes that the noise follows a zero-mean i.i.d. Gaussian distribution. This assumption, that simplifies the analysis, is supported by:

- **Central Limit Theorem:** the theorem states that given $X_1, X_2, \ldots, X_n$ i.i.d. random variables with finite mean μ and finite, nonzero variance σ^2. Consider the standardized sum:

$$S_n = \frac{1}{\sqrt{n}} \sum_{i=1}^{n} \Big(\frac{X_i - \mu}{\sigma} \Big). \tag{7.4}$$

  Then, as $n \to \infty, S_n \xrightarrow{d} N(0,1)$. In other words, for large n, S_n converges in distribution to the standard normal. Due to this theorem, many physical processes contributing to noise accumulation result in Gaussian distributions.

- **Poisson Noise and Transformation:** although photon noise in imaging systems is inherently Poissonian, at high photon counts, the Poisson distribution approximates a Gaussian distribution. Variance-stabilizing transforms can convert the noise to Gaussian-like behavior for low photon counts.
- **Mathematical Simplicity:** Gaussian noise leads to elegant mathematical formulations, making derivations such as the log-likelihood $\mathtt{p}(\mathbf{y}|\mathbf{x})$ and related models tractable.
- **MMSE Denoisers and Theoretical Importance:** MMSE denoisers designed for Gaussian noise possess critical theoretical properties that enable their use in inverse problems.

7.2 Image Denoising – The Classical Era

Driven by developments in signal and image processing, the evolution of efficient image denoising methods has been quite important. Two key periods define this evolution: the classical era, which runs from the 1970s to the early 2010s, and the AI revolution, which started in 2012 and is still under progress.

7.2.1 The Bayesian Perspective in Denoiser Design

Two common approaches to recover $\mathbf{x}$ given noisy image model $\mathbf{y} = \mathbf{x} + \mathbf{v}$ ($\mathbf{v} \sim \mathcal{N}(0, \sigma^2\mathbf{I})$ is zero-mean Gaussian noise) are the Maximum Likelihood Estimation (MLE) and Maximum A Posteriori (MAP) Estimation.

7.2.1.1 Maximum Likelihood Estimation (MLE)

MLE estimates the value of $\mathbf{x}$ that maximizes the likelihood of the observed noisy image $\mathbf{y}$:

$$\hat{\mathbf{x}}_{\text{MLE}} = \arg\max_{\mathbf{x}} p(\mathbf{y}|\mathbf{x}). \tag{7.5}$$

For Gaussian noise, the likelihood is:

$$p(\mathbf{y}|\mathbf{x}) = \text{const} \cdot \exp\left(-\frac{\|\mathbf{y}-\mathbf{x}\|_2^2}{2\sigma^2}\right). \tag{7.6}$$

Maximizing this likelihood leads to:

$$\hat{\mathbf{x}}_{\text{MLE}} = \mathbf{y}. \tag{7.7}$$

This result shows the ill-posed problem since the MLE solution fails to incorporate any additional knowledge or constraints that would provide a better reconstruction, it simply produces the noisy image without efficiently denoising it.

7.2.1.2 Bayesian Posterior and MAP Estimation

The MAP estimate tries to overcome the limitations of MLE, balancing the fidelity to the noisy observation y and regularization:

$$p(\mathbf{x}|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{x}) \cdot p(\mathbf{x})}{p(\mathbf{y})}. \tag{7.8}$$

By incorporating the prior distribution $p(\mathbf{x})$, the MAP estimate is defined as:

$$\hat{\mathbf{x}}_{\text{MAP}} = \arg\max_{\mathbf{x}} p(\mathbf{x}|\mathbf{y}) = \arg\min_{\mathbf{x}} \left\{ \frac{\|\mathbf{y} - \mathbf{x}\|_2^2}{2\sigma^2} - \log(p(\mathbf{x})) \right\}. \tag{7.9}$$

The regularizer used in the MAP estimation is $-\log(p(\mathbf{x}))$, where the term $p(x)$ is the prior distribution. This factor drives the solution toward more realistic outcomes, thereby providing high-quality reconstructions, penalizing estimate of $\mathbf{x}$ that have low prior probability, suggesting that they are unlikely candidates for the original $\mathbf{x}$.

7.2.1.3 Minimum Mean Squared Error (MMSE) Estimation

The MMSE estimator minimizes the expected squared error:

$$\hat{\mathbf{x}}_{\text{MMSE}} = \mathbb{E}[\mathbf{x}|\mathbf{y}] = \int \mathbf{x} \cdot p(\mathbf{x}|\mathbf{y})\, d\mathbf{x}. \tag{7.10}$$

Computing the MMSE estimate is difficult due to the complexity of the posterior distribution. This has led to the adoption of MAP-based approaches in classical image denoising.

7.2.1.4 Key Developments in Priors

Several major trends have characterized the evolution of priors:

- **From Gaussian to Heavy-tailed Distributions:** instead of easy to use Gaussian priors, heavy-tailed distributions (distribution where a relatively higher probability is placed on extreme (very large or very small) values than light-tailed distributions) are used to better capture natural image statistics.
- **From Linear to Non-linear Representations:** linear techniques like Principal Component Analysis (PCA) gave way to non-linear methods, including wavelets and sparse modeling, enabling richer representations.
- **From Classical to the Deep Learning Priors:** instead of depending just on theoretical assumptions, the priors can learn from large datasets and can change their internal representations.

7.3 The Deep Learning Revolution in Image Denoising

Deep learning has revolutionized image processing including denoising. Designing powerful denoisers now has new directions thanks to the ability to train overparameterized networks with non-convex goals.

Deep learning techniques achieve better results on Peak Signal-to-Noise Ratio (PSNR) and visual quality than conventional approaches most of the times.

7.3.1 Supervised Denoising

Usually, supervised deep learning-based denoising consists in the following steps:

1. **Dataset Preparation:** collect a large dataset of clean images $\mathcal{X} = \{\mathbf{x}_k\}_{k=1}^M$. Generate noisy counterparts $\mathcal{Y} = \{\mathbf{y}_k\}_{k=1}^M$, where $\mathbf{y}_k = \mathbf{x}_k + \mathbf{v}_k$ and $\mathbf{v}_k \sim \mathcal{N}(0, \sigma^2\mathbf{I})$.
2. **Model Architecture:** define a parametric denoising function $\hat{\mathbf{x}} = D_\Theta(\mathbf{y}, \sigma)$, parameterized by Θ, using architectures such as CNNs, UNet, or Transformers.
3. **Loss Function:** define a loss function to compare the denoised output $\hat{\mathbf{x}}$ with the ground truth $\mathbf{x}$, e.g.,
$$L(\Theta) = \sum_{k=1}^{M} \|\mathbf{x}_k - D_\Theta(\mathbf{y}_k, \sigma)\|_2^2. \tag{7.11}$$
4. **Optimization:** minimize the loss $L(\Theta)$ using stochastic gradient descent (SGD) with backpropagation.

7.3.2 Unsupervised and Self-supervised Techniques

When clean images are not available, unsupervised and self-supervised techniques are used such Noise2Noise (14), Noise2Void (15), and Noise2Self (16). These methods train denoisers by directly exploiting noisy image features including statistical consistency and self-similarity.

7.3.3 Denoising Architectures and Advancements

The evolution of denoising architectures highlights the field's rapid progress:

- **Classical to Modern Networks:** Initial methods using feed-forward CNNs evolved into UNet-based architectures and, more recently, Transformer-based models.
- **Training Objectives:** While traditional methods prioritize MSE and PSNR, newer approaches explore perceptual losses like Structural Similarity Index Measure (SSIM) and Learned Perceptual Image Patch Similarity (LPIPS) for visually pleasing results.

- **Generative Models:** GAN-based denoising challenges the denoiser to produce outputs indistinguishable from authentic images, emphasizing perceptual quality.

7.3.4 Color Image Denoising

Most denoisers are adapted to work with color images through techniques like:

- **Joint Channel Processing:** Flatten RGB channels into longer vectors for joint processing.
- **Channel Separation:** Operate on channels separately.
- **Deep Learning Approaches:** Process RGB images as 3D tensors with 3D convolutions or other specialized methods that account for color layer interactions.

7.4 Conclusion

The classical era of image denoising set fundamental groundwork for current methods. The context of imagine denoising has changed with the move from conventional techniques to deep learning. Although clean data availability makes supervised approaches most effective, unsupervised and self-supervised procedures present strong alternatives.

CHAPTER 8

IMAGE RESTORATION AND DENOISING TECHNIQUES

As explained in the previous chapter, image acquisition, transmission, or processing can introduce degradation, including noise, blur, or other distortion. Image restoration tries to recover a clean image from its degraded version.

Advancements in deep learning have led to state-of-the-art (SOTA) results in image restoration tasks. Many of these methods are based on the UNet architecture, a classical design that employs a U-shaped structure with skip connections. Variants of UNet have demonstrated significant performance gains with increased system complexity. This complexity can be categorized broadly into inter-block complexity and intra-block complexity.

8.1 Architectural Complexities in Image Restoration

8.1.1 Inter-block Complexity

Every stage in multi-stage networks improves the output of the one before it. The idea is to break difficult tasks into more manageable smaller tasks. Inter-block complexity arises from multi-stage architectures where performances are improved along with the computational overhead. On the other hand, single-stage approaches with a reduced complexity can provide good performance. Combining these methods allows hybrid approaches to get SOTA performances.

8.1.2 Intra-block Complexity

Intra-block complexity focuses on the architecture of individual network blocks where different techniques are applied to improve block-level performances. For instance:

- **Channelwise Attention** reduces the computational and memory overhead of self-attention by replacing spatial attention maps with channelwise attention maps.
- **Gated Linear Units** are used to enhance non-linear modeling capabilities.
- **Window-based Attention** is introduced to allow the architecture to focus on localized regions of an image.
- **Depthwise Convolutions** are added in feed-forward layers to improve the network capability of capturing local features.

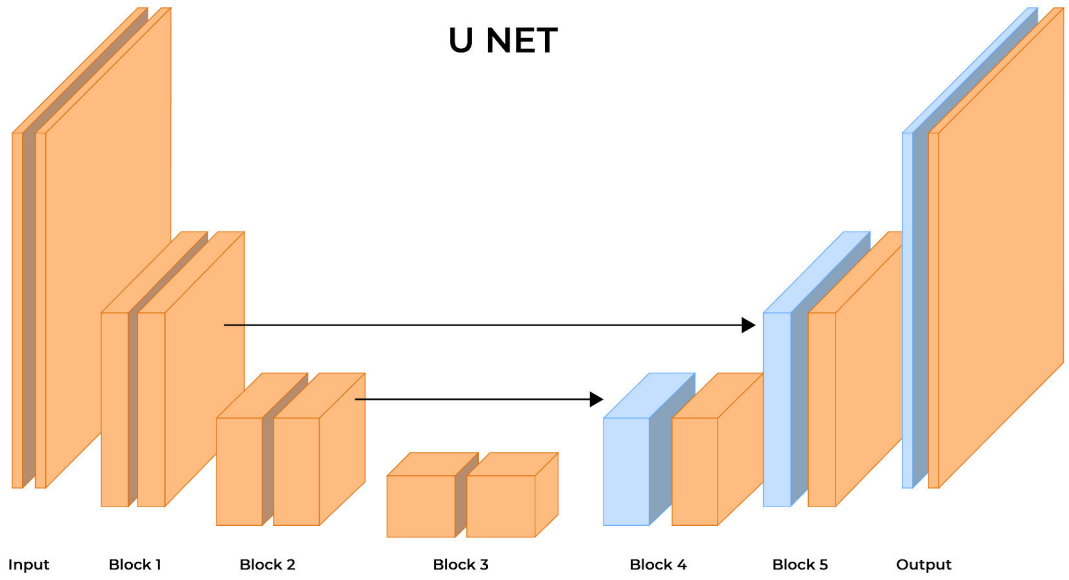

Figure 11: U-Net architecture used.

8.2 Building a Simple Baseline

The objective presented in (17) is to find efficient and effective architectures for image restoration tasks: components are added if their necessity is verified. For this purpose, the performance of each component is evaluated on the SIDD and GoPro datasets. SIDD contains real smartphone images shot under different lighting conditions, paired with corresponding clean reference images. Deep learning models are trained on it to reduce noise patterns in the images. Video sequences from a GoPro camera constitute the GoPro dataset; motion-blurred frames and their corresponding sharp ground truth. Deep learning techniques for deblurring and other restoration tasks are trained and evaluated using this dataset.

The presented baseline achieves state-of-the-art results with minimal computational costs, showing how increasing complexity is not the only path to better performance. The architecture is called *Nonlinear Activation Free Network (NAFNet)*

8.2.1 Architecture

The architecture features a single-stage U-shaped (Figure 11) with skip connections to reduce inter-block complexity. Inspired by UNet(32), this architecture is achieve competitive results, as per empirical data presented in the original paper (17) (see Tables Table I and Table II).

8.2.2 Designing the Internal Block

The internal structure of each block needs to be carefully designed.

8.2.2.1 Plain Block

The idea is to start with a simple architecture of basic components - convolution, Rectified Linear Unit (ReLU), and shortcut connections - and gradually add more com-

plex block or feature that are needed. These base elements are arranged as Figure 12 a shows.

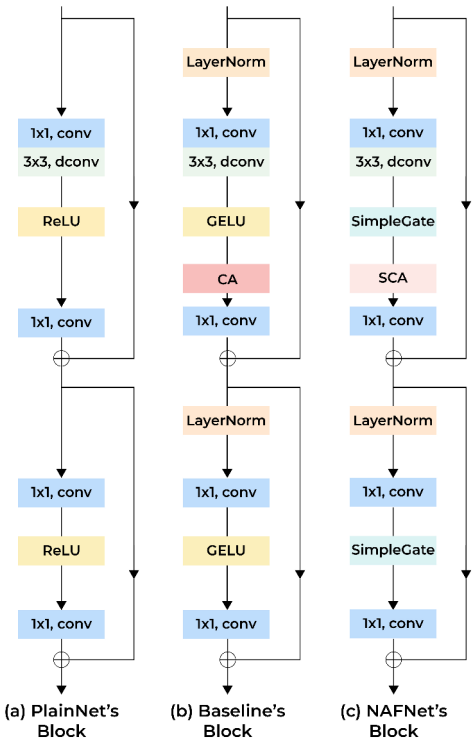

Figure 12: Block Scheme.

8.2.2.2 Normalization

Training deep neural networks depends critically on normalization process that improve the statistics of the data processed, facilitating the training phase.

Adding Layer Normalization to the plain block stabilizes training and improve performance as empirically shown in (17):

- +0.44 dB on the SIDD dataset (from 39.29 dB to 39.73 dB).
- +3.39 dB on the GoPro dataset (from 28.51 dB to 31.90 dB).

8.2.2.3 Activation Functions

ReLU is a widely used activation function in computer vision tasks. However, recent SOTA methods increasingly favor the Gaussian Error Linear Unit (GELU) (33) that is a smoother version of ReLU. In the presented architecture, ReLU is replaced with GELU in the plain block, resulting in the following:

- Comparable performance on the SIDD dataset (39.73 dB with ReLU versus 39.71 dB with GELU).
- A non-trivial improvement on the GoPro dataset (31.90 dB with ReLU versus 32.11 dB with GELU).

8.2.2.4 Attention Mechanisms

Attention mechanisms are integral to modern computer vision models. While vanilla self-attention (34) captures global information, its quadratic computational complexity makes it impractical for high-resolution image restoration. Alternative approaches include:

- **Window-based Attention:** applied in fixed-size local windows, it reduces computation but sacrifices global information.
- **Channel-wise Attention:** Modifies spatial attention to channel attention, guaranteeing computing efficiency while preserving global information.

Adding channel attention to the plain block achieve bettere results as shown by (17):

- +0.14 dB on the SIDD dataset (39.71 dB to 39.85 dB).
- +0.24 dB on the GoPro dataset (32.11 dB to 32.35 dB).

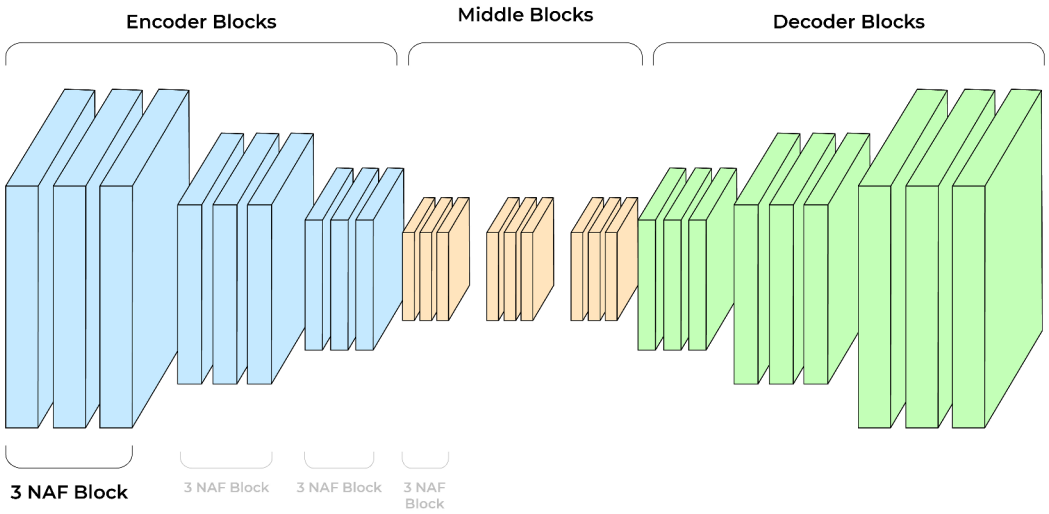

Figure 13: Baseline Architecture.

8.2.3 Summary of the Baseline

The resulting block structures are reported in Figure 12. The Figure 12.b show a simpler structure, but ensure that each component is effective, producing a robust model. The Figure 12.c present additional simplified block that further enhances the performance without increasing the complexity too much. This additional blocks will

be shown in the section 8.3. The baseline architecture (Figure 13) present a encoder-decoder structure with a U-Shaped scheme. Each layer of the encoder or decoder is made by different NAF block (Figure 12.b) that sequentially lower or increase the data dimensionality and simultenously lowering the noise in the recovered data. This is the architecture presented by (17) which surpasses previous State-Of-The-Art results on the SIDD and GoPro datasets with significantly lower computational costs (as illustrated in Tables Table I and Table II).

8.3 Gated Linear Units and Simplified Channel Attention for Image Restoration

Gated Linear Units (GLUs) (35) work by multiplying element-wise the outputs of two linear transformation layers - one of which passes through a nonlinear activation function. This approach indirectly increases nonlinearity and can enable the network to learn more complex representations.

8.3.1 Simplification of GLUs

The presented architecture (17) simplifies GLUs by removing the explicit nonlinear activation function without degrading performance. High performance of the GLU mechanism is maintained by its built-in nonlinearity resulting from the product of two linear transformations.

Gated Linear Units (GLUs) are a powerful mechanism that introduces nonlinearity into neural networks. GLUs can be formulated as:

$$\text{Gate}(\mathbf{X}, f, g, \sigma) = f(\mathbf{X}) \odot \sigma(g(\mathbf{X})), \tag{8.1}$$

$\mathbf{X}$ is the feature map, f and g are linear transformations, σ is a nonlinear activation function, and $\odot$ represents element-wise multiplication.

Nonlinearity is incorporated in the GLU structure by means of the product of two transformations, hence improving model performance. Including Gated Linear Units complicates the block, going against the simplicity goal. One alternative is to use the GELU (Gaussian Error Linear Unit) function(33) that can be approximated as:

$$\mathrm{GELU}(x) = x\Phi(x) \approx 0.5x\left(1 + \tanh\left[\sqrt{\frac{2}{\pi}}\left(x + 0.044715x^3\right)\right]\right). \tag{8.2}$$

Observe that GELU is identical to GLU when f and g are identity functions and σ acts as $\Phi(x)$. This insight suggests that GLU inherently contains nonlinearity and can function without a dedicated nonlinear activation function. The simplified GLU used in the presented architecture splits the feature map into two parts and multiplies them element-wise as shown in Figure 14.c. In this way it implicitly introduces the nonlinearity:

$$\mathrm{SimpleGate}(\mathbf{X}, \mathbf{Y}) = \mathbf{X} \odot \mathbf{Y}, \tag{8.3}$$

$\mathbf{X}$ and $\mathbf{Y}$ are equal-sized halves of the same split feature map.

By substituting the GLU with the proposed SimpleGate, the performance on image restoration tasks improves as shown in (17):

- $+0.08$ dB on the SIDD dataset (39.85 dB $\rightarrow$ 39.93 dB).
- $+0.41$ dB on the GoPro dataset (32.35 dB $\rightarrow$ 32.76 dB).

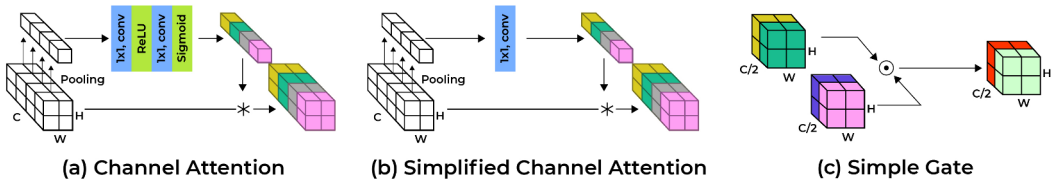

Figure 14: Simplified Channel Attention and Simple Gate.

8.4 Simplified Channel Attention

Channel attention is a technique to capture global information efficiently. Shown in Figure 14, it is defined as:

$$\mathrm{CA}(\mathbf{X}) = \mathbf{X} * \sigma\left(W_2 \cdot \max\{0, W_1 \cdot \mathbf{pool}(\mathbf{X})\}\right) \tag{8.4}$$

- $\mathrm{pool}(\mathbf{X})$ is a global average pooling operation that aggregates spatial information into channels.
- W_1 and W_2 are fully-connected layers.
- σ is a nonlinear activation function (e.g., Sigmoid).
- $*$ represents channel-wise multiplication.

This operation generates weights for each channel based on global information, which are used to adjust the feature map. In the NAFNet (17) a variant called Simplified Channel Attention (SCA) is implied:

$$\mathrm{SCA}(\mathbf{X}) = \mathbf{X} * (W \cdot \mathbf{pool}(\mathbf{X})), \tag{8.5}$$

where W is a single transformation matrix.

Simplified Channel Attention maintains the two most critical aspects of channel attention while reducing its complexity:

- Aggregation of global information.
- Channel-wise interaction.

Despite its simplicity, SCA achieves similar performance to the original channel attention mechanism as shown in (17):

- $+0.03\,\mathrm{dB}$ on the SIDD dataset ($39.93\,\mathrm{dB} \rightarrow 39.96\,\mathrm{dB}$).
- $+0.09\,\mathrm{dB}$ on the GoPro dataset ($32.76\,\mathrm{dB} \rightarrow 32.85\,\mathrm{dB}$).

8.5 Summary

Image restoration and denoising advances are driven by architectural breakthroughs and clever computational resource use. Multi-stage and single-stage designs handle inter-block complexity, while methods including channel-wise attention, GLUs, and locally improved networks address intra-block difficulty.

The presented simplifications and modifications are implemented in Nonlinear Activation Free Network (NAFNet) which achieves state-of-the-art results while reducing computational complexity. NAFNet provides an efficient framework for image restoration tasks and competitive performance, the work's results (17) are reported in the Tables Table I and Table II.

TABLE I: PERFORMANCE OF BASELINE AND NAFNET ON THE SIDD DATASET.

Method	**PSNR (dB)**	**Complexity (GMACs)**
Baseline	40.30	84
NAFNet	40.30	65

TABLE II: PERFORMANCE OF BASELINE AND NAFNET ON THE GOPRO DATASET.

Method	**PSNR (dB)**	**Complexity (GMACs)**
Baseline	33.40	84
NAFNet	33.69	65

CHAPTER 9

EXPERIMENTS

9.1 Introduction

In this chapter, the experimental setup and methodology used to validate the proposed approach to jointly denoise and compress satellite imagery are exposed. The discussing starts the motivation for integrated denoising and compression (9.2). Next, the dataset used is presented(9.3), followed by the noise model that simulates real-world satellite conditions (9.4). The baseline MeanScaleHyperprior architecture is showed in (9.5). Finally, the several modifications that embed denoising capabilities directly within the compression framework are presented (9.6). The quantitative and qualitative results of these experiments will be presented in the next chapter 10.

9.2 Integrated Denoising and Compression for Satellite Imagery

This chapter presents my central contribution: a framework for jointly performing image denoising and compression within a single architecture. The motivating scenario involves satellite images acquired in the R, G, B, and NIR bands, which are often corrupted by noise due to sensor limitations and challenging environmental conditions. Traditional solutions typically chain two separate models - one for denoising, another for compression - but such an approach can be complex and suboptimal in terms of rate-distortion performance. Instead, I try to modify an existing compression network, the *MeanScaleHyperprior*, to remove noise while encoding images simultaneously.

I hypothesize that incorporating a denoising component directly into the compression pipeline reduces both the distortion of the transmitted images and the bit rate required.

In other words, if satellite images are denoised early, less extraneous information needs to be encoded and transmitted. This approach thereby reduce bandwidth constraints and improves reconstruction quality.

In the experiments, I use the Sentinel 12MS dataset (Section 9.3), on which I simulate a noise model for training. The theoretical background of the noise model applied to simulate satellite noise can be found in Section 9.4. [1]

9.3 Dataset Description

The availability of curated annotated datasets is important for developing machine learning models since modern deep learning techniques typically require large-scale data to achieve high generalization.

Unlike conventional computer vision, which deals with everyday objects in standard photographs, remote sensing data covers various imaging conditions and modalities. This intrinsic complexity makes interpretation more challenging.

9.3.1 The SEN1-2 and SEN12MS Datasets

To advance deep learning research in remote sensing, the SEN1-2 dataset (19) was published in 2018. It contains approximately 280,000 pairs of corresponding Sentinel-1 Synthetic Aperture Radar (SAR) and Sentinel-2 optical images. SEN1-2 aimed to simplify the connection between classical computer vision problem and remote sensing tasks. The data were enormously simplified, as only vertically polarized (VV) Sentinel-1 imagery in dB scale was provided, and Sentinel-2 images were reduced to RGB images with adjusted histograms without any geolocation information.

---

[1]The implemented code is reported 9.4.1.

Based on feedback from the community, a successive version suitable for the broader remote sensing community has been published. This is the dataset used in the experiments and it is called SEN12MS (18). This dataset contains complete multi-spectral information in geocoded imagery and is detailed in the sections below.

9.3.2 SEN12MS Dataset Composition

The SEN12MS dataset reuse images and informations acquired from different spatial mission as follow:

9.3.2.1 Sentinel-1 Data

The Sentinel-1 expedition used two satellites equipped with C-band SAR sensors, that are Synthetic Aperture Radar (SAR) instruments that operate at microwave frequencies in the C-band portion of the electromagnetic spectrum - commonly defined as roughly 4 to 8 GHz, and that are capable of acquiring imagery regardless of weather conditions. By using a predefined schedule for data collection, the system avoids overlapping requests, resulting suitable for applications that aim to track changes over extended periods(22).

In SEN12MS (18), the Sentinel-1 data come from ground-range-detected (GRD) images taken using the commonly used Interferometric Wide Swath (IW) mode. These images show how much radar energy bounces back from the Earth's surface (the "backscatter coefficient"), stored in decibels (dB). The data are recorded at about 5 m resolution along the satellite's flight path (azimuth) and 20 m resolution in the perpendicular direction (range).

The acquired images were ortho-rectification[1] using precise orbit information and elevation data. Aside from this correction, no additional preprocessing was applied, preserving the data in a near-original state.

9.3.2.2 Sentinel-2 Data

Two identical satellites placed 180° apart in the same orbit constitute the Sentinel-2 mission (21). This program offers multi-spectral images with information on land surfaces of Earth. Sentinel-2 is especially suitable for vegetation monitoring throughout the growing season with a swath width[2] of up to 290km and a revisit time of over five days.

In SEN12MS (18), multi-spectral Sentinel-2 data cubes, that are a stack of images captured at multiple wavelengths (spectral bands) arranged in three dimensions - two for location on Earth (latitude and longitude) and one for the different spectral bands. These cubes come from the original georeferenced granules, meaning each pixel in the images has precise latitude-longitude information so it can be accurately mapped to the Earth's surface.

Beyond the standard ortho-rectification step - which corrects for distortions caused by the satellite's viewing angle, the curvature of the Earth, and variations in terrain - a sophisticated mosaicking process of the different images was applied.

---

[1]ortho-rectification is the process of geometrically correcting an image to reduce distortions, sensor artifacts, earth curvature and so on, and align the image with coordinates on the ground, restoring geometric integrity

[2]A swath is the width of ground (or ocean) a satellite can image in a single pass, typically measured in kilometers, which determines how wide an area is captured in each observation.

9.3.2.3 MODIS Data and Land Cover Information

The Moderate Resolution Imaging Spectroradiometer (MODIS) captures images of the entire Earth almost every day, with spatial resolutions ranging from 250 m to 1,000 m across various spectral bands. From these calibrated reflectance data, a hierarchical classification based on the Land Cover Classification System (LCCS) - supported by additional post-processing steps - generates annually updated global land-cover maps (18). The typical resolution of these maps is approximately 500 m.

In SEN12MS (18), land-cover information sourced from the MCD12Q1 V6 product [1] was resampled to a 10 m pixel size to match the resolution of the Sentinel-1 and Sentinel-2 image patches. Expressly, four distinct layers of classification were provided:

- *IGBP Classification Layer*: this layer categorizes the Earth's land surface based on the International Geosphere–Biosphere Programme (IGBP) scheme. Typical classes include evergreen needleleaf forests, savannas, croplands, and urban areas.
- *LCCS-Derived Land-Cover Layer*: this layer refines broad land-cover categories by incorporating attributes such as vegetation type or density.
- *LCCS-Derived Land-Use Layer*: Also based on LCCS definitions, the land-use layer focuses on how land is utilized (e.g., agricultural production, forestry, or urban development), rather than its biophysical characteristics.
- *LCCS Surface Hydrology Layer*: This component highlights the presence and extent of water such as rivers, lakes, and so on.

---

[1] The MCD12Q1 series is a standard MODIS land-cover product distributed by the National Aeronautics and Space Administration (NASA). Version 6 (V6) provides annually updated global land-cover classifications derived from calibrated reflectance data collected by the Terra and Aqua satellites.

9.3.3 Data Preparation Using Google Earth Engine

Google Earth Engine (GEE) (23) was used to generate large-scale, multi-sensor remote sensing image patches for SEN12MS (18). A random sampling of regions of interest (ROIs) across the globe and within different meteorological seasons was performed.

9.3.3.1 Export and Post-Processing

All Sentinel-1, Sentinel-2, and MODIS data were exported as GeoTiffs at a nominal 10 m scale.

9.3.4 Dataset Curation

Upon local retrieval, an inspection protocol was applied:

- Each full-scene triplet (Sentinel-1, Sentinel-2, and MODIS Land Cover) was visualized. Scenes with large no-data areas, undetected clouds, or artifacts were discarded.
- The retained 252 scenes were split into 256×256 pixel patches, each overlapping by 50%. This overlap was chosen to maximize sample count without excessively compromising patch independence.
- A second visual inspection identified and removed patches with localized issues (e.g., residual clouds, jet streams, or distortions). Ultimately, 180,662 valid patch triplets remained, forming the SEN12MS dataset, requiring 421.3 GiB of storage.

9.3.5 Dataset Structure

The final collection includes 180,662 triplets of Sentinel-1, Sentinel-2, and MODIS data. The dataset is structured into four main branches:

- `ROIs1158_spring`

- `ROIs1868_summer`
- `ROIs1970_fall`
- `ROIs2017_winter`

These seed values (1158, 1868, 1970, 2017) correspond to random samplings during different northern hemisphere seasons. Each branch contains multiple ROIs or "scenes" subdivided into patches. Filenames follow the pattern:

`ROIsSSSS_SEASON_DD_pXXX.tif`

where `SSSS` is the seed, `SEASON` is the meteorological season, `DD` is a data identifier (`s1`, `s2`, or `lc`), and `pXXX` is a unique patch ID.

The absence of a fixed training/testing split allows end-users to partition data best suited to their particular tasks (e.g., grouping by seasons or by specific ROIs).

For the thesis purpose the branch ROIs1868_summer of SEN12MS is used. It is more than enough for the experiments requirements.

9.4 Noise Model for Satellite Imagery

Satellite-based optical sensors typically generate images affected by multiple noise sources. These images exhibit pixel-dependent variance because noise may include an **additive component** (often modeled as Gaussian or "dark/electronic" noise) and a **multiplicative component** (often modeled as Poisson or "photon/shot" noise).

$$\sigma_n(x, y) = \sqrt{a^2 + b\, I_{nf}(x, y)}, \tag{9.1}$$

where $I_{nf}(x, y)$ denotes the noise-free signal at pixel coordinates (x, y), a represents the dark (Gaussian) noise level, and b corresponds to the scale of the photon (Poisson)

noise. These parameters closely relate to how the sensor electronics and photon detection processes operate.

9.4.0.1 Additive (Dark) Noise

The term a^2 in Equation (Equation 9.1) model the noise component known as dark noise, that is generated from the thermal activity and readout electronics of the sensors. Dark noise is essentially constant over the image and affects lower-intensity areas more noticeably where the noise to signal ratio can be higher.

9.4.0.2 Multiplicative (Photon) Noise

The second term, $b\, I_{nf}(x, y)$ in Equation (Equation 9.1), model the photon (or shot) noise that describes the intrinsic randomness in the arrival of photons at the detector, reflecting the quantum nature of light. As pixel intensity increases, the random fluctuations also grow proportionally, reflecting the physical behavior of photonic detection.

9.4.1 Code

In the following is reported the implemented noise model in python:

```
[language=python]

    def add_noise(image, a_range=(10.0, 40.0), b=3.0):
    """
    Simulation of 'shot noise' (Poisson) and 'read/dark
       noise' (Gaussian)
    on a 16-bit image (range [0, 65535]), mimicking a
       satellite sensor.
```

```
    Parameters:
    - image    : np.ndarray, clean image (range [0, 65535])
       on which to simulate the noise
    - a_range : tuple (min_std, max_std) specifying the
       uniform random interval for the standard deviation of
       the Gaussian noise (in DN).
    - b        : scaling factor to translate DN into average
       photon/electron counts.

    Returns:
    - noisy_image: np.ndarray (float or same dtype as `image
       `) in [0, 65535], containing the noisy image.
    """

    # Randomly sample the std of the read noise (Gaussian)
    a = np.random.uniform(a_range[0], a_range[1])

    # Gaussian noise (read/dark noise), mean 0 and std = a
    read_noise = np.random.normal(loc=0.0, scale=a, size=
       image.shape)

    # Compute mean counts (lambda) for the Poisson
    # b * image => average number of photons/electrons
```

```
    lambdas = np.maximum(b * image, 0)

    # Poisson sampling (shot_counts)
    shot_counts = np.random.poisson(lam=lambdas)

    # Zero-mean shot noise = (Poisson count) - (mean value)
    shot_noise = shot_counts - lambdas

    # Sum original image, read noise, and shot noise
    noisy_image = image + read_noise + shot_noise

    # Final clip to stay within 16-bit limits [0, 65535]
    noisy_image = np.clip(noisy_image, 0, 65535)

    return noisy_image / 65535.0
```

9.4.2 Relevance for Image Compression and Denoising

In typical satellite acquisition settings, raw images have non-negligible noise levels; sending or storing such noisy data in compressed form without denoising can degrade performance. An example of the noise simulated on the training images is reported in Figure 15 (other results are reported at the end).

Our work aims to develop a compression model that simultaneously removes noise and compresses the input image. Denoising the image before or during compression can preserve details, reduce bitrates more effectively, and improve the decompressed result.

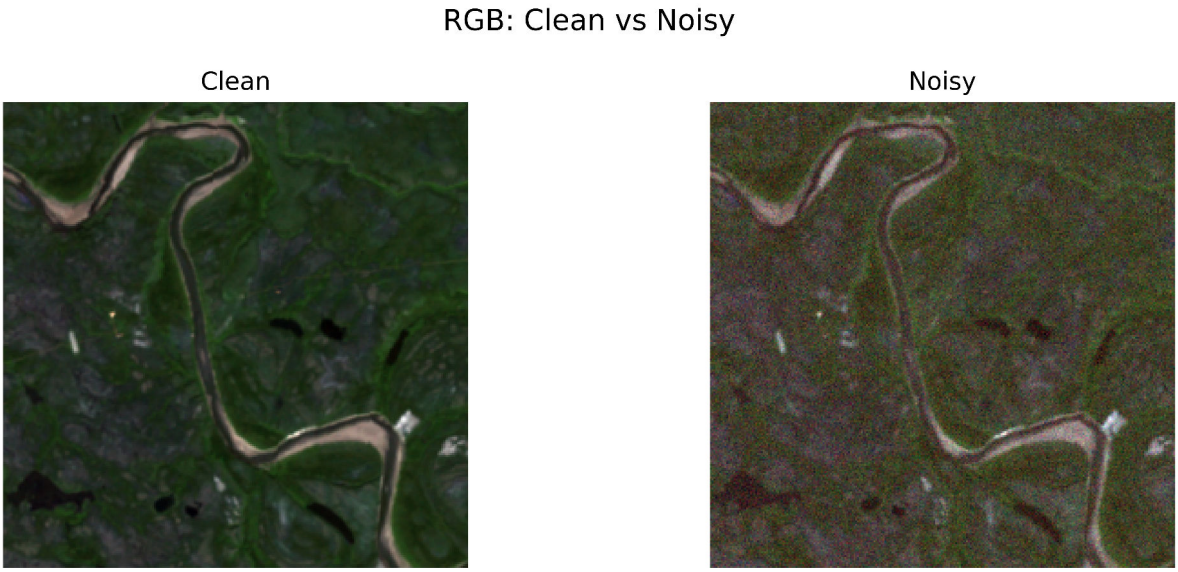

Figure 15: the RGB clean and noisy version of the training image.

9.5 Baseline: MeanScaleHyperprior Overview

MeanScaleHyperprior is a widely used architecture for learned image compression, a schematic diagrab is proposed in Figure 16. It consists of:

- **Analysis and synthesis transform** (g_a and g_s), which map images to a latent representation and reconstruct them back, respectively.
- **Hyperprior analysis and synthesis** (h_a and h_s), which encode and decode side information (e.g. scale or variance parameters) to further refine latent entropy coding.
- The block Q represents quantization, while AE, AD and CB represent respectively arithmetic encoder, arithmetic decoder and the compressed bit-stream.

- The difference between the Scale hyperprior is that together with the scale also the mean is estimated in the hyperprior branch.

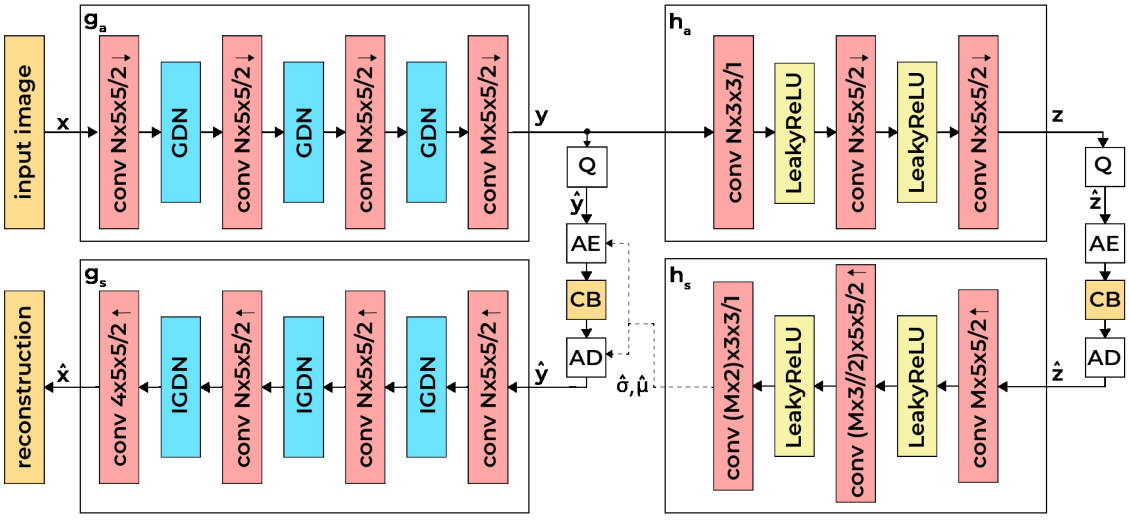

Figure 16: Architecture of the **Mean Scale Hyperprior** model.

In the MeanScaleHyperprior, no explicit denoising modules are included. The network is trained to minimize a rate-distortion objective, balancing the bit rate R with a distortion metric D (e.g., MSE) on reconstructed images. However, when the input data are corrupted by noise - as commonly encountered in satellite systems - the latent representation may become less efficient to encode, resulting in higher bit rates or lower fidelity.

9.6 Proposed Architectures for Joint Compression and Denoising

I explored four architectures that integrate denoising components within the Mean-ScaleHyperprior framework. Each variant attempts to remove noise while simultaneously learning an efficient latent representation.

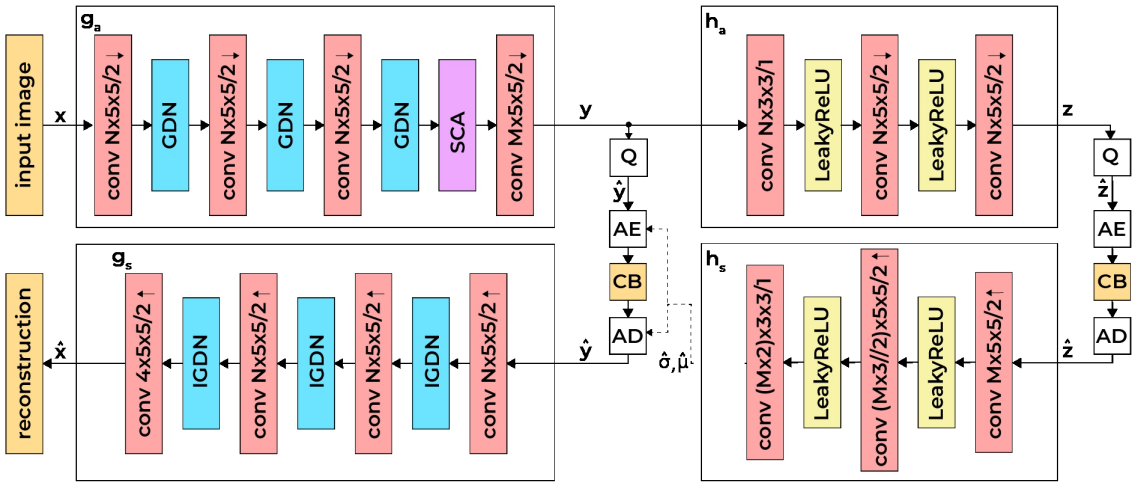

Figure 17: Architecture of the **Denoisy Mean Scale Hyperprior** model.

9.6.1 DenoisyMeanScaleHyperprior

9.6.1.1 Motivation

In *DenoisyMeanScaleHyperprior*, I incorporate a simplified channel attention (SCA) module into the primary compression pipeline as shown in Figure 17. The idea is to attenuate noise early on so that subsequent transforms encode mostly clean features.

9.6.1.2 Architecture

1. **SCA Block:** A lightweight attention mechanism that assigns channel-wise weights to feature maps. It emphasizes cleaner features and suppresses noise-corrupted channels.
2. **Integration:** The SCA block is added within g_a, right after the convolution and nonlinearity layers, allowing the network to learn how to attenuate noise before the main compression process.
3. **Hyperprior Processing:** After passing through SCA, the latent is processed by h_a and h_s as usual to generate side information for entropy coding.

9.6.2 LatentDenoiser

9.6.2.1 Motivation

Instead of focusing on noise suppression in the pixel domain, *LatentDenoiser* defers denoising to the latent space. In many cases, the latent representation can isolate noise patterns more effectively, making denoising tasks simpler than in the raw image domain. A scheme of the architecture is shown in Figure 18

9.6.2.2 Architecture

1. **Primary Analysis:** The noisy image is transformed by g_a into a latent.

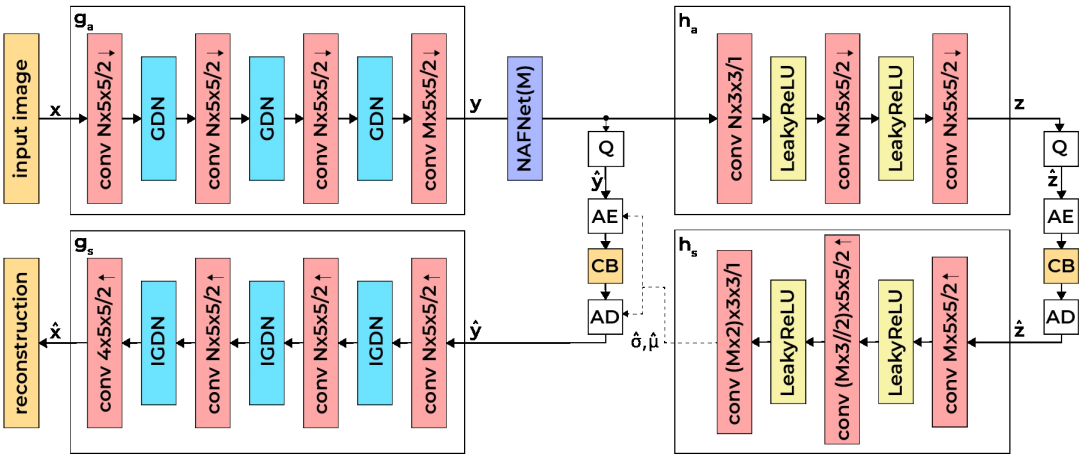

Figure 18: Architecture of the **Latent Denoiser** model.

2. **NAFNet in Latent Space:** A dedicated NAFNet-based denoiser is inserted between g_a and g_s to remove noise in the latent domain.
3. **Hyperprior Processing:** The denoised latent is passed through h_a and h_s to obtain side information and finalize the compression.
4. **Final Reconstruction:** g_s reconstructs the compressed image.

9.6.3 NAFCompression

9.6.3.1 Motivation

While *LatentDenoiser* insert a single denoising stage in the latent, *NAFCompression* repeatedly interleaves NAFBlocks throughout the entire pipeline. I incorporate *single NAFBlocks* at different points, ensuring noise is suppressed progressively. The aim is to

remove noise sooner, resulting in a more compact and less entropic latent representation. The model workflow is depicted in Figure 19.

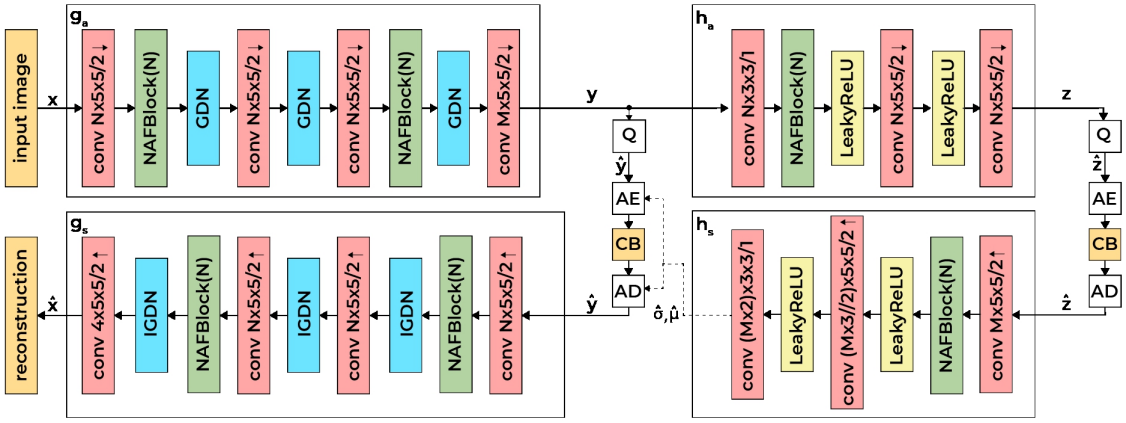

Figure 19: Architecture of the **NAFCompression** model.

By repeating NAFBlocks throughout the analysis and synthesis transforms (both main and hyperprior), the model refines and denoises features at multiple scales. This comprehensive integration can lead to more robust noise suppression and better rate-distortion performance, albeit with an increase in computational complexity.

9.7 Conclusion and Next Steps

In this chapter, I introduced four approaches for merging denoising and compression into a single learned framework:

- *DenoisyMeanScaleHyperprior*, which uses a simplified channel attention block to filter out noise early on.
- *LatentDenoiser*, which places a NAFNet-based denoiser in the latent space.
- *NAFCompression*, which repeatedly inserts NAFBlocks throughout both the main and hyperprior transforms.

All these variants aim to reduce redundancy in the latent representation by removing noise, thereby improving rate-distortion performance. In the subsequent chapter, I will present quantitative and qualitative results demonstrating how each architecture performs in terms of bit rate, distortion, and computational overhead.

CHAPTER 10

RESULTS

10.1 Overview

The performance of the five distinct compression and denoising architecture are compared:

1. **DenoisyMeanScaleHyperprior** (blue)
2. **NAFCompression** (green)
3. **LatentDenoiser** (red)
4. **MeanScaleHyperprior** (orange)

Figure 20 displays the corresponding rate-distortion curves, where PSNR is plotted against the average bitrate. As expected, all methods improve in PSNR with increasing bitrate, but each method follows a distinct trajectory depending on how it balances compression fidelity and the treatment of noise.

From a high-level perspective, the starting model **MeanScaleHyperprior** (orange curve) and **NAFCompression** (magenta curve) show the best results across a broad range of bitrates.

In contrast, methods such as **DenoisyMeanScaleHyperprior** (red) and **Latent-Denoiser** (green) demand higher bitrates or result in slightly lower PSNR when the noise is extreme.

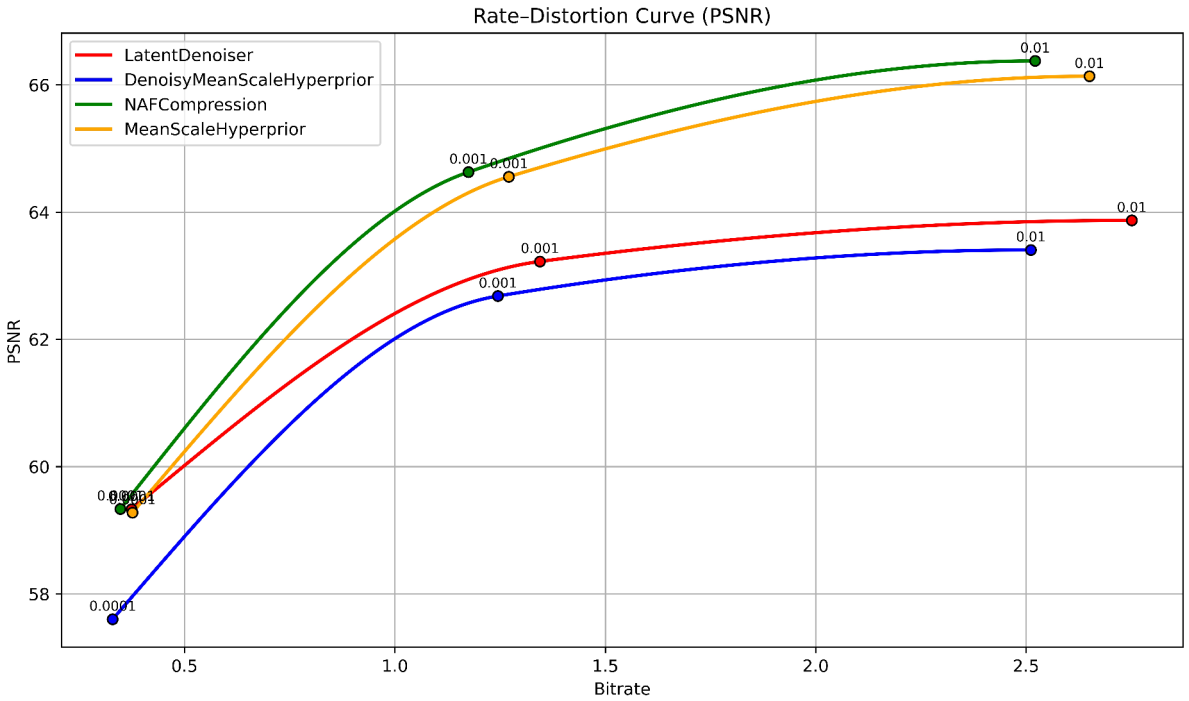

Figure 20: Rate-Distortion Curves (PSNR) for five tested approaches.

10.2 Strengths and Weaknesses of Training Objectives

In deep learning-based image compression, a model is trained by taking an input image - potentially noisy - compressing and subsequently decompressing it, and then comparing the reconstructed output to a target image. The goal is to minimize the difference between the target and the reconstructed images - thus maximizing visual fidelity - while also achieving the lowest possible bitrate, measured in bits per pixel. When dealing with noisy inputs, two main training strategies can be considered: in one, the model receives a noisy image but is trained to reconstruct the clean version by comparing its output with a noise-free target; in the other, the model instead aims to reconstruct the same

noisy image, preserving the noise rather than removing it. These different objectives lead to distinct approaches in how the model learns to balance compression performance and image quality.

Noisy-Target Training: **Strengths:**

- Captures actual pixel variations present in noisy conditions.
- Can be beneficial for scenarios where preserving certain noise characteristics is necessary.

Weaknesses:

- The model invests bits in encoding noise fluctuations, leading to higher bitrates.
- Reconstruction quality will be penalized by the presence of noise.

Clean-Target Training (Proposed Method): **Strengths:**

- Filters out noise, thereby saving bits and improving compression efficiency.
- Yields sharper and denoised outputs.

Weaknesses:

- May not faithfully reproduce noise if the application requires its reconstruction.

In this thesis, I wan to show that the *clean-target* strategy can achieves superior rate-distortion performance when the final goal is a noise-free reconstruction. The model used to simulate the noise on the training set of image is reported in 9.4. The approaches

are tested on the MeanScaleHyperprior and NAFCompression models to achieve both effective denoising and robust compression.

10.3 Analysis of Rate-Distortion Performance in the Presence of Noise

This chapter examines the rate-distortion behavior of different compression models when trained and tested on noisy images. Figure 21 presents a set of Peak Signal-to-Noise Ratio (PSNR) curves plotted against the average bitrate (in bits per pixel). Four variations are compared:

1. **NAFCompression_NoisyTraining** (red curve): NAFCompression trained with the noisy-target approach.
2. **MeanScaleHyperprior_NoisyTraining** (green curve): MeanScaleHyperprior trained with the noisy-target approach.
3. **NAFCompression** (blue curve): NAFCompression trained with the clean-target approach.
4. **MeanScaleHyperprior** (magenta curve): MeanScaleHyperprior trained with the clean-target approach.

Each curve reflects how effectively a given model reconstructs the target image for different value of the parameter λ, that balances rate and distortion during the loss function evaluation. I will delve into why these curves assume their specific shapes, how noise affects the training process, and what these results suggest about encoding efficiency for clean versus noisy images.

10.4 Background on Rate-Distortion and PSNR

Rate-distortion theory explores the fundamental trade-off between the bitrate of a compressed signal and the distortion (or error) resulting from decompression. In image

compression, the bitrate measures the average number of bits required to encode one pixel, while distortion is often quantified via PSNR. Higher PSNR indicates less distortion and, therefore, a more faithful reconstruction. However, achieving higher PSNR typically requires more bits.

When noise is introduced into the training set - especially if the model is trained to reconstruct noisy images - its presence influences the content being compressed. Models attempting to encode noisy images inherently dedicate bits to represent noise fluctuations. Conversely, models that are trained to reconstruct clean images learn to encode the underlying structure of the image and discard the noise component.

10.5 Interpretation of the PSNR Curves

Figure 21 plots PSNR (vertical axis) versus the bitrate (horizontal axis). On the left of the plot all models show an higher distortion (lower PSNR) but use a lower number of bit to encode the images(lower bitrate). As the bitrate increases (moving right), the PSNR improves. The main observations are:

i) **Models Trained on Noisy Targets Show Higher Bitrate for Comparable Distortion.** The *NAFCompression_NoisyTraining* and *MeanScaleHyperprior_NoisyTraining* curves (red and green) represent the model trained with the noisy-target approach. They generally show at higher bitrates for the same PSNR level as their clean-target counterparts. This is because, during training, these models learn to encode not only the core image content but also the noise fluctuations. The extra bits required to capture these random variations cause a systematic shift in the curve to the right.

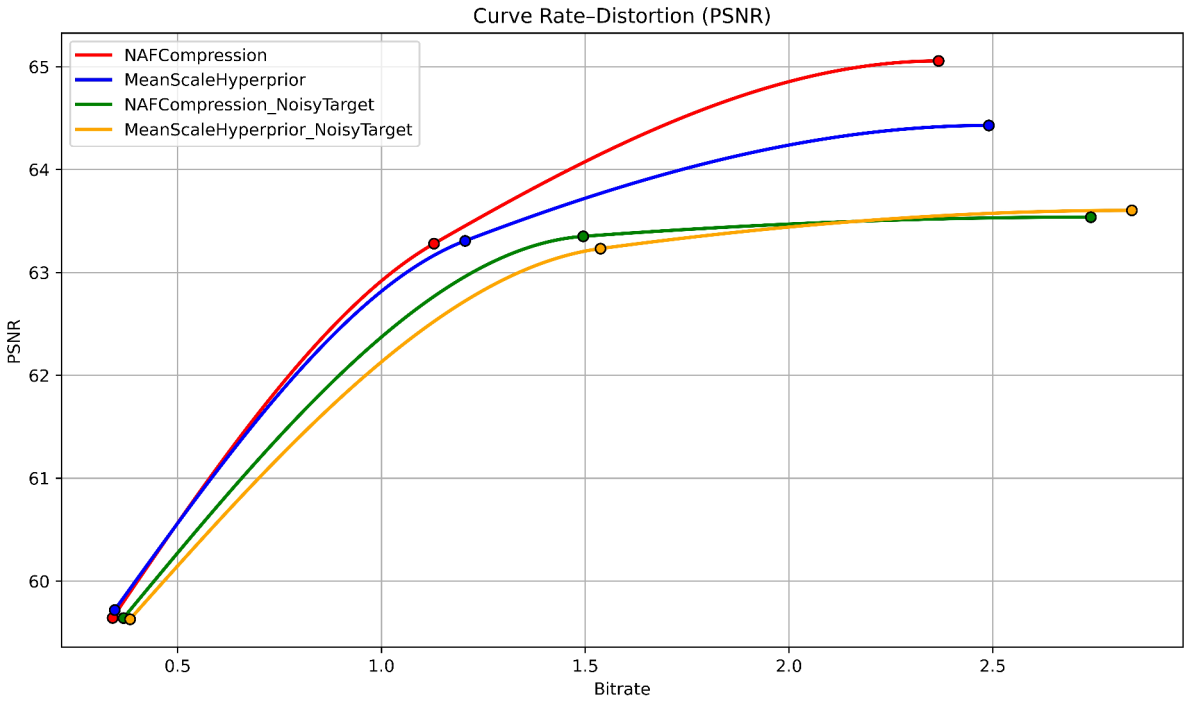

Figure 21: Rate-Distortion Curves (PSNR) for various training strategies.

ii) **PSNR are comparable.** When one model is trained to predict a clean target, it focuses on discarding the noise. During training, its PSNR is measured against the clean target, which guides the model to ignore irrelevant noisy details from the input. In contrast, a model trained on a noisy target has its training metrics measured against the noisy image, so it aims to replicate noise details as well. To compare the two approaches, once the training phase is complete, all the models must be evaluated on their ability to reconstruct the same clean images starting from their noisy versions.

iii) **Testing on Clean Images.** When I evaluate both the training approaches on the same clean reference images, the comparison becomes fairer. For the noisy-target training approach might appear paradoxical that the trained model can produce good results even though they are not trained to reconstruct clean images. The explanation might be that the discrepancies introduced during the compression and reconstruction phase by the model (often manifested as slight blur or local artifacts) might make undetectable the local error caused by the attempt to reconstruct the noisy image, resulting in PSNRs not far from the ones obtained with the clean-target training approach. However, while still achieving good PSNR value, the models trained with a noisy-target approach present higher bitrates, reflecting the fact that the network's latent codes has learned to capture noise features.

10.6 Effects of Noise on Bitrate Allocation

A critical insight is that noise itself demands extra encoding resources. When the training objective explicitly includes noisy pixels as the reconstruction target, additional requirements are needed:

- **Extra Bits for Noise:** The model has no incentive to discard noise, so it dedicates encoding capacity to replicate noise patterns, increasing the total bit cost.
- **Higher λ Values and Balancing Fidelity:** At certain higher λ settings, the compression system prioritizes fidelity over bit savings. In noisy-training scenarios, "fidelity" includes matching the noise pattern. Therefore, the model's latent representation grows in size to capture the random fluctuations.
- **Clean-Target Models Save Bits:** In contrast, a model trained with a clean target learns that noise is irrelevant. Hence, it focuses on reconstructing the essential

structures and textures of the image, yielding a lower overall bitrate for a similar level of perceived quality.

As shown in Figure 21, the green curve (*NAFCompression*) often lies above or to the right of the blue curve (*NAFCompression_NoisyTraining*) at comparable PSNR values, indicating that more bits are spent when noise must be encoded. Likewise, the blue curve (*MeanScaleHyperprior_NoisyTraining*) tends to demand more bits than the magenta curve (*MeanScaleHyperprior*). Though these differences can narrow at lower bitrates and low PSNR, the trend remains consistent as the PSNR and the bitrate increas: encoding noise adds overhead.

CHAPTER 11

CONCLUSION

In this work, I explored the benefits of merging denoising and compression into a single, end-to-end learned system. The objective was to evaluate whether this joint approach could deliver robust image quality while using limited resources and lower processing overhead. Rather than first removing noise and then applying a separate compression method, I show that simultaneously carrying out both steps can have advantages: fewer bits are wasted encoding noise, less hardware is needed for two separate processing pipelines, and inference becomes more efficient.

Building on the MeanScaleHyperprior architecture, the proposed modifications incorporate denoising mechanisms directly into the compression pipeline. The model is trained with noisy inputs but clean targets, so the network learns to reconstruct the clean data and is able to discard noisy data before encoding them. Experimental evaluations, conducted with realistic satellite images and noise simulation, confirm that this approach can achieve better rate-distortion performances.

The rate-distortion curves in Figure 21 show how training the network to directly reconstruct clean data from their noisy counterparts allows the model to ignore superfluous noisy information and focus on the salient features. This results in lower bitrates and higher PSNR compared to training with noisy targets.

The trade-off is straightforward: including noise in the training objective demands a higher bitrate allocation, because the model preserves that noise detail in its latent representation.

The results demonstrate that aiming for clean reconstructions using a joint denoising-compression framework can ensure high performance in resource-constrained settings. This creates opportunities for future work to implement this model in scenarios with limited resources, such as remote sensing, embedded devices, or real-time systems, where finding a good balance between performance and available resources is crucial. It also opens up possibilities to extend this approach to other domains, such as video and/or audio analysis and compression.

CITED LITERATURE

1. Goodfellow, I., Bengio, Y., and Courville, A.: Deep Learning. MIT Press, 2016.

2. Minsky, M. and Papert, S.: Perceptrons: An Introduction to Computational Geometry. MIT Press, 1969.

3. Rumelhart, D. E., Hinton, G. E., and Williams, R. J.: Learning internal representations by error propagation. In Parallel Distributed Processing: Explorations in the Microstructure of Cognition, eds. D. E. Rumelhart and J. L. McClelland. MIT Press, 1986.

4. Hebb, D. O.: The Organization of Behavior: A Neuropsychological Theory. New York, Wiley, 1949.

5. Xu, M., Xu, B., Zhang, Y., and Feng, Y.: A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications. Journal of Parallel and Distributed Computing, 156:1–17, 2021.

6. Gray, R. M. and Neuhoff, D. L.: Quantization. IEEE Transactions on Information Theory, 44(6):2325–2383, 1998.

7. Ballé, J., Minnen, D., Singh, S., Hwang, S. J., and Johnston, N.: Variational image compression with a hyperprior. In International Conference on Learning Representations (ICLR), 2018.

8. Ballé, J., Laparra, V., and Simoncelli, E. P.: Density modeling of images using a generalized normalization transformation. arXiv e-prints, 2015. Presented at the 4th International Conference on Learning Representations (ICLR), 2016.

9. Ballé, J., Laparra, V., and Simoncelli, E. P.: End-to-end optimized image compression. In 5th International Conference on Learning Representations (ICLR), 2017.

10. Ballé, J., Laparra, V., and Simoncelli, E. P.: End-to-end optimization of nonlinear transform codes for perceptual quality. arXiv e-prints, 2016. arXiv:1607.05006, Accepted for presentation at the 32nd Picture Coding Symposium.

11. Theis, L., Shi, W., Cunningham, A., and Huszár, F.: Lossy image compression with compressive autoencoders. arXiv e-prints, 2017. Presented at the 5th International Conference on Learning Representations (ICLR).

12. Freedman, D. and Johnson, R. F.: Density modeling of images using a generalized normalization transformation. arXiv e-prints, 2020. arXiv:2010.05290.

13. Ren, Z., Liu, S., Yang, X., et al.: Image denoising: The deep learning revolution and beyond – a survey paper –. arXiv e-prints, 2021. arXiv:2106.12370.

14. Lehtinen, J., Munkberg, J., Hasselgren, J., Laine, S., Karras, T., Aittala, M., and Aila, T.: Noise2noise: Learning image restoration without clean data. In International Conference on Machine Learning (ICML), pages 2965–2974. PMLR, 2018.

15. Krull, A., Buchholz, T.-O., and Jug, F.: Noise2void: Learning denoising from single noisy images. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 2129–2137, 2019.

16. Batson, J. and Royer, L.: Noise2self: Blind denoising by self-supervision. In International Conference on Machine Learning (ICML), pages 524–533. PMLR, 2019.

17. Zhang, K., Gool, L. V., and Timofte, R.: Simple baselines for image restoration. arXiv e-prints, 2021. arXiv:2104.00419.

18. Schmitt, M., Hughes, L. H., and Zhu, X. X.: SEN12MS — a curated dataset of georeferenced multi-spectral sentinel-1/2 imagery for deep learning and data fusion. In ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences, volume IV-2/W7, pages 153–160, 2019.

19. Schmitt, M., Hughes, L. H., and Zhu, X. X.: The sen1-2 dataset for deep learning in sar-optical data fusion. In ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences, volume IV-1, pages 141–146, 2018.

20. Schmitt, M., Hughes, L. H., Qiu, C., and Zhu, X. X.: Aggregating cloud-free sentinel-2 images with google earth engine. In ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences, volume IV-2/W7, pages 145–152, 2019.

21. Drusch, M., Bello, U. D., Carlier, S., Colin, O., Fernandez, V., Gascon, F., Hoersch, B., Isola, C., Laberinti, P., Martimort, P., Meygret, A., Spoto, F., Sy, O.,

Marchese, F., and Bargellini, P.: Sentinel-2: Esa's optical high-resolution mission for gmes operational services. Remote Sensing of Environment, 120:25–36, 2012.

22. Schubert, A., Small, D., Miranda, N., Geudtner, D., and Meier, E.: Sentinel-1a product geolocation accuracy: Commissioning phase results. Remote Sensing, 7(7):9431–9449, 2015.

23. Gorelick, N., Hancher, M., Dixon, M., Ilyushchenko, S., Thau, D., and Moore, R.: Google earth engine: Planetary-scale geospatial analysis for everyone. Remote Sensing of Environment, 202:18–27, 2017.

24. McCulloch, W. and Pitts, W.: A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics, 5:115–133, 1943.

25. Rosenblatt, F.: The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, 65(6):386–408, 1958.

26. Mehlig, B.: Machine learning with neural networks. arXiv e-prints, 2021. arXiv:1901.05639v4 [cs.LG], 27 Oct 2021.

27. Haykin, S.: Neural Networks: A Comprehensive Foundation. New Jersey, Prentice Hall, 2nd edition, 1999.

28. He, K., Zhang, X., Ren, S., and Sun, J.: Deep residual learning for image recognition. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 770–778, 2016.

29. Ioffe, S. and Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv e-prints, 2015. arXiv:1502.03167.

30. Krizhevsky, A., Sutskever, I., and Hinton, G. E.: Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems (NIPS), eds. F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, volume 25, pages 1097–1105. Curran Associates, Inc., 2012.

31. Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., and Li, F. F.: Imagenet: A large-scale hierarchical image database. In 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 248–255, 2009.

32. Wang, Z., Cun, X., Bao, J., and Liu, J.: Uformer: A general u-shaped transformer for image restoration. arXiv preprint, 2021. arXiv:2106.03106.

33. Hendrycks, D. and Gimpel, K.: Gaussian error linear units (gelus). arXiv preprint, 2016. arXiv:1606.08415.

34. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I.: Attention is all you need. In Advances in Neural Information Processing Systems (NeurIPS), volume 30, 2017.

35. Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D.: Language modeling with gated convolutional networks. In International Conference on Machine Learning (ICML), pages 933–941. PMLR, 2017.

APPENDIX

ADDITIONAL RESULTS

Below, additional simulations as the ssim-bpp and mse-bpp curves are presented to better illustrate the results and provide a perceptual indication of the quality. Furthermore, examples of the trained models ability to reconstruct the image is shown using this sets of images: the clean version, the noisy version, and the version reconstructed by the model.

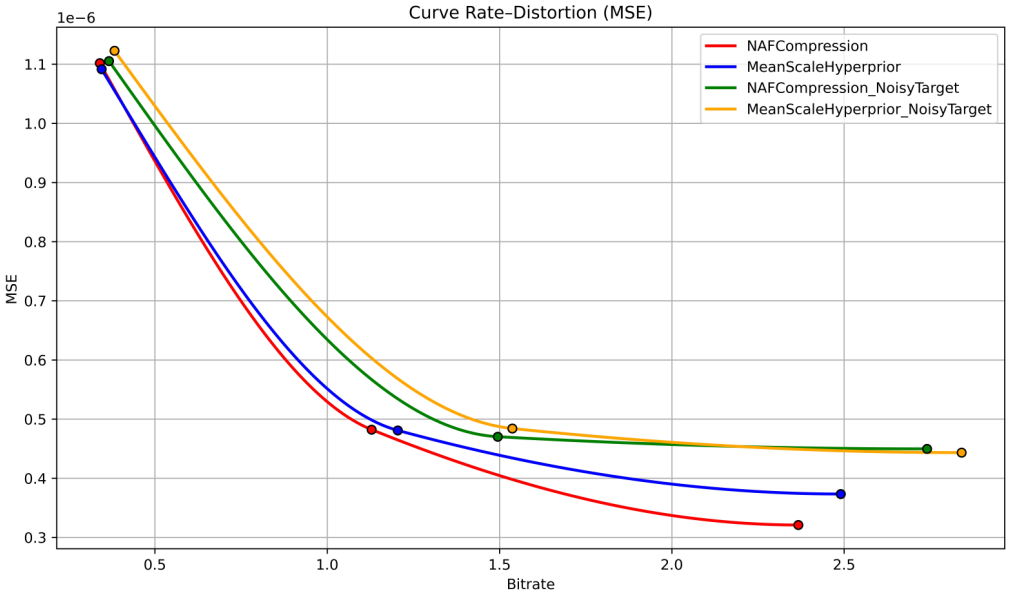

Figure 22: MSE vs bitrates.

APPENDIX (Continued)

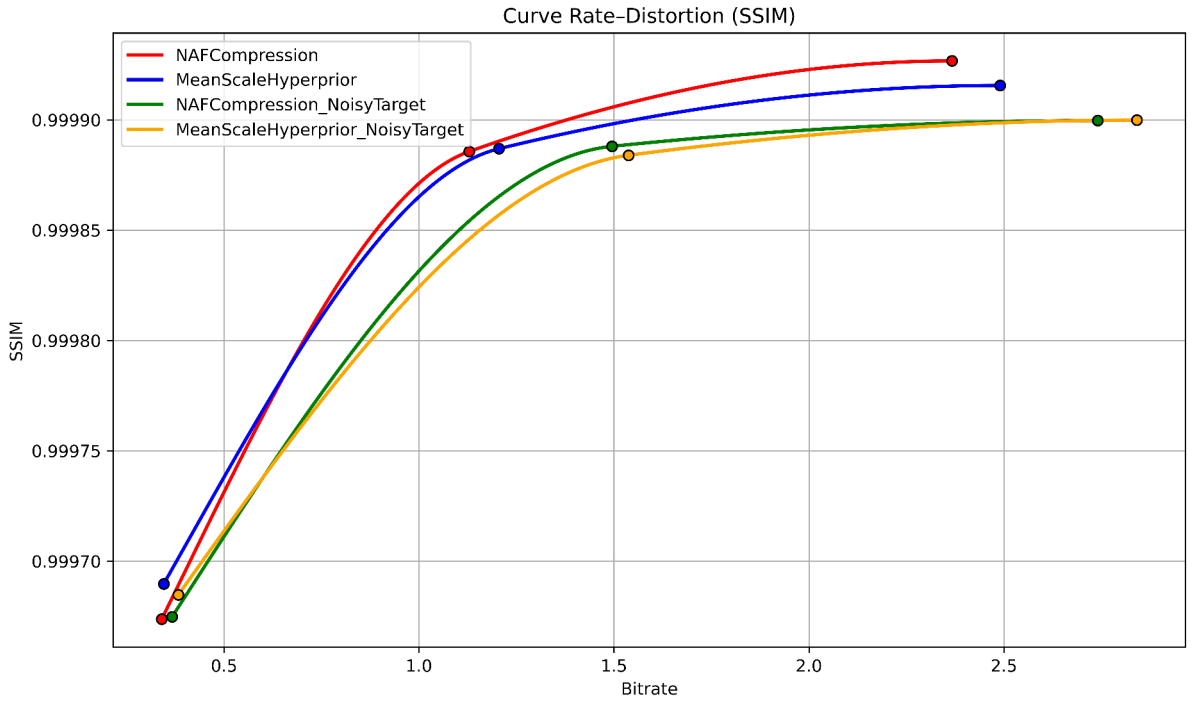

Figure 23: Curve SSIM vs bitrates for the MeanScaleHyperprior and NAFCompression trained with the 2 modalities (noisy or clean target).

The plot Figure 22 and represent the curve of respectively MSE and SSIM against BBP for the MeanScaleHyperprior and NAFCompression model, both trained with the different approach (noisy or clean target). You can observe how the model trained with the clean-target approach show a smaller dispersion cloud around the ground-truth line, meaning that the model is effectively removing noise.

Figure 28, Figure 29, Figure 30, Figure 31 show the dispersion plot for the a sample image after processing it with the MeanScaleHyperprior or NAFCompression model trained with the two different approaches and an hyperparameter of $\lambda = 0.01$. T

APPENDIX (Continued)

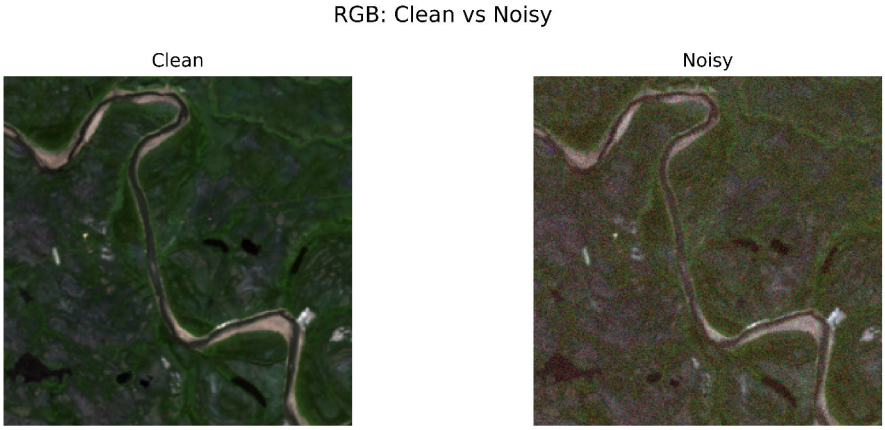

Figure 24: The rgb clean and noisy version of a training image.

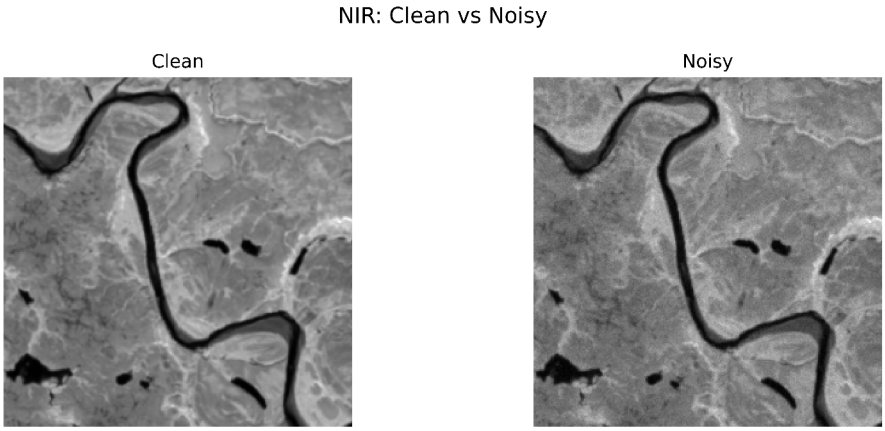

Figure 25: The nir clean and noisy version of a training image.

APPENDIX (Continued)

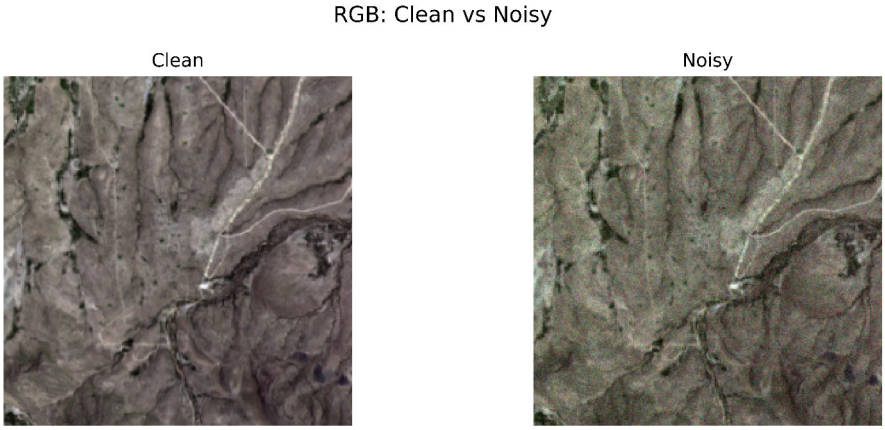

Figure 26: The rgb clean and noisy version of a training image.

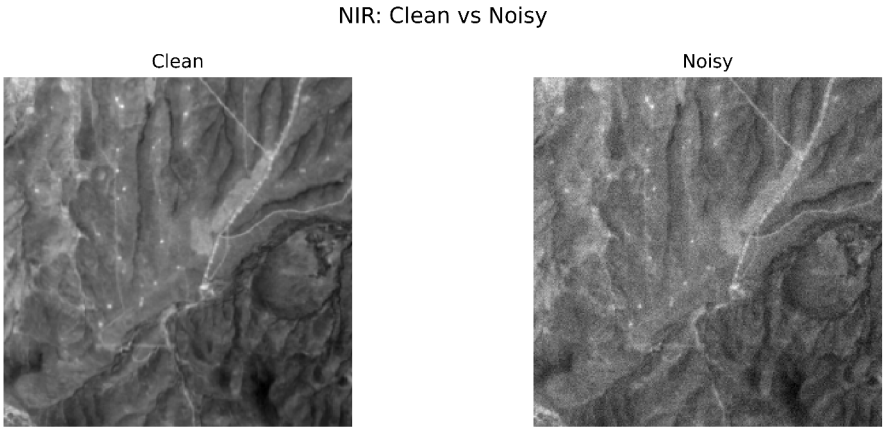

Figure 27: The nir clean and noisy version of a training image.

APPENDIX (Continued)

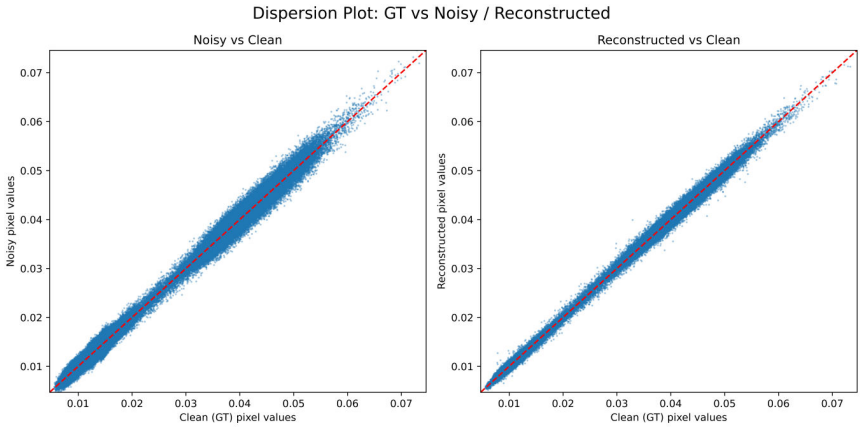

Figure 28: Dispersion plot MeanScaleHyperprior trained with clean target.

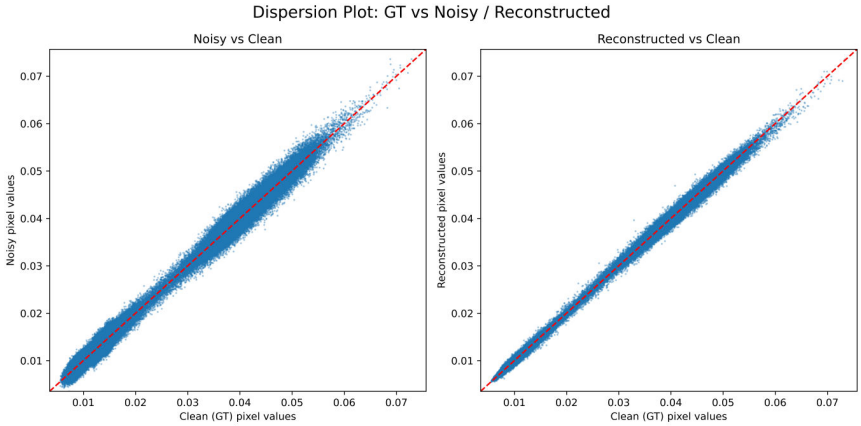

Figure 29: Dispersion plot NAFCompression.

APPENDIX (Continued)

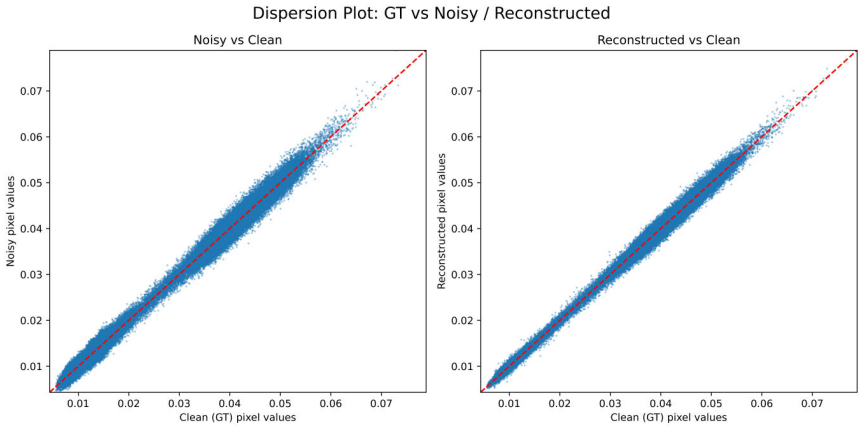

Figure 30: Dispersion plot MeanScaleHyperprior trained with noisy target.

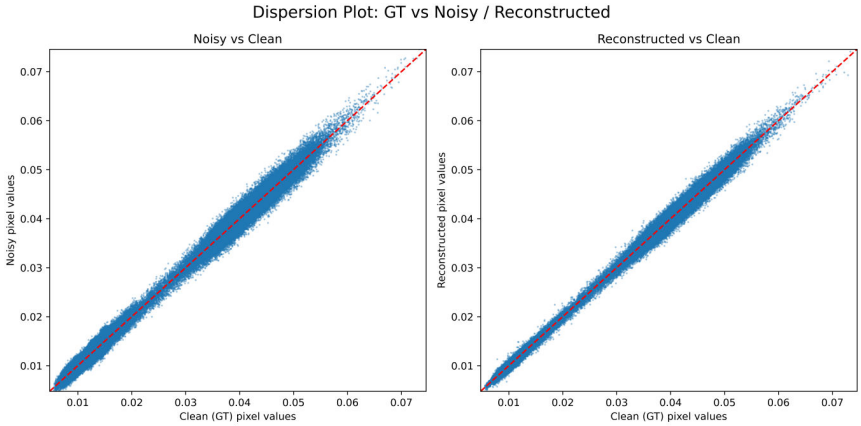

Figure 31: Dispersion plot NAFCompression trained with noisy target.

APPENDIX (Continued)

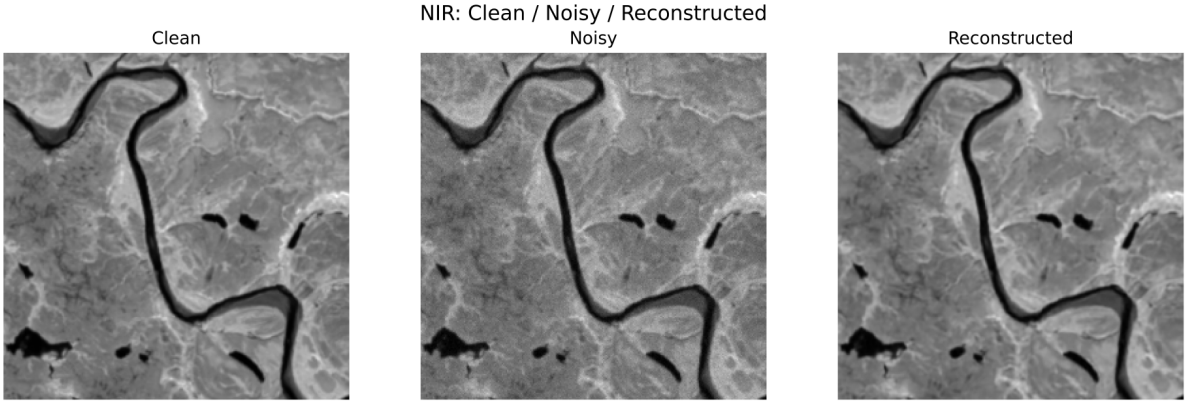

Figure 32: Near-infrared (NIR) clean, noisy, and decoded image with NAFCompression (clean target and lambda=0.01).

Figure 33: RGB clean, noisy, and decoded image with NAFCompression (clean target and lambda=0.01).

APPENDIX (Continued)

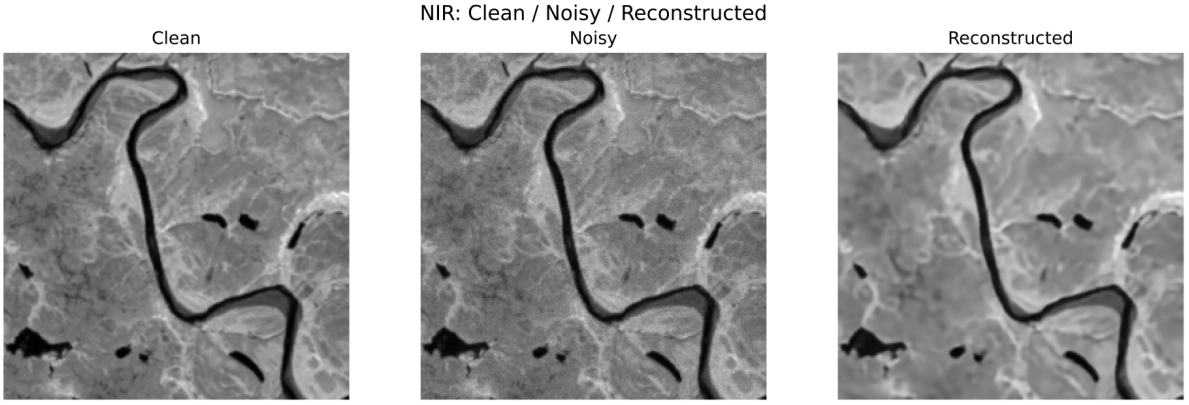

Figure 34: Near-infrared (NIR) clean, noisy, and decoded image with NAFCompression (clean target and lambda=0.0001).

Figure 35: RGB clean, noisy, and decoded image with NAFCompression (clean target and lambda=0.0001).

APPENDIX (Continued)

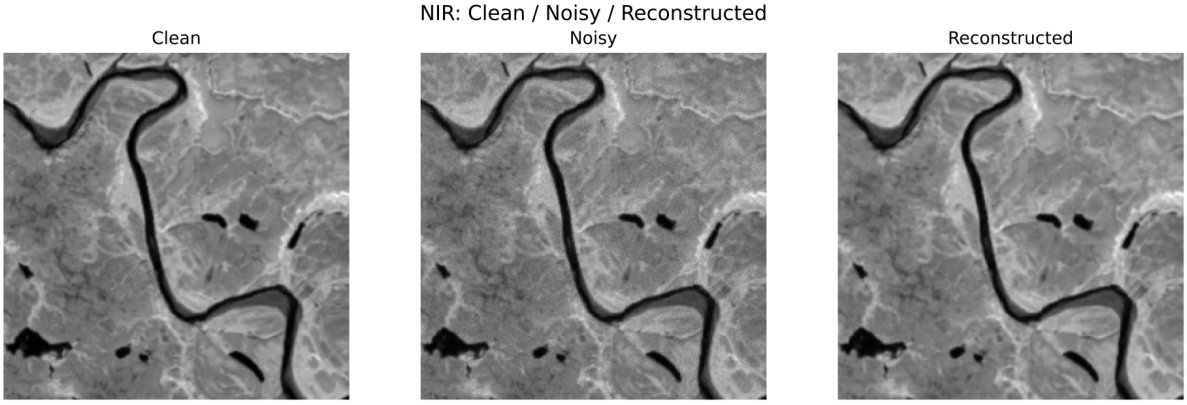

Figure 36: Near-infrared (NIR) clean, noisy, and decoded image with MeanScaleHyperprior (clean target and lambda=0.01).

Figure 37: RGB clean, noisy, and decoded image with MeanScaleHyperprior (clean target and lambda=0.01).

APPENDIX (Continued)

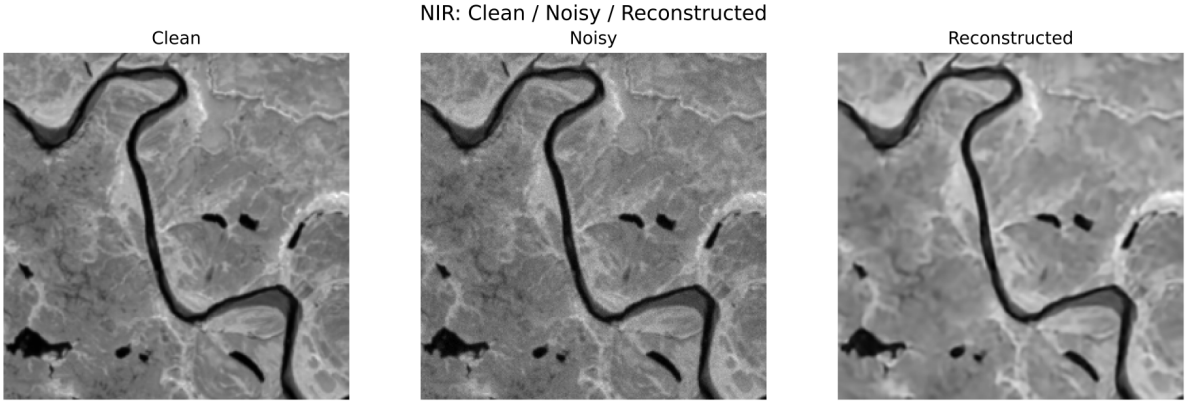

Figure 38: Near-infrared (NIR) clean, noisy, and decoded image with MeanScaleHyperprior (clean target and lambda=0.0001).

Figure 39: RGB clean, noisy, and decoded image with MeanScaleHyperprior (clean target and lambda=0.0001).

APPENDIX (Continued)

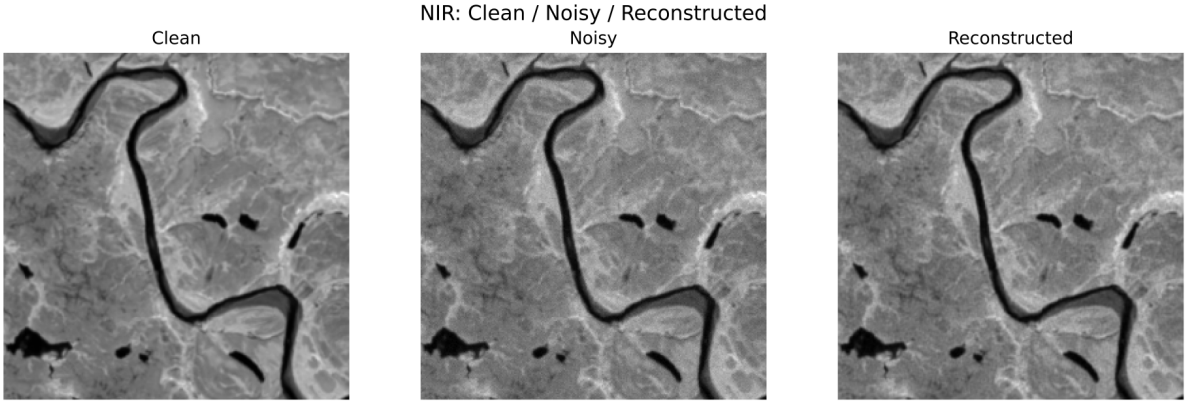

Figure 40: Near-infrared (NIR) clean, noisy, and decoded image with NAFCompression (noisy target and lambda=0.01).

Figure 41: RGB clean, noisy, and decoded image with NAFCompression (noisy target and lambda=0.01).

APPENDIX (Continued)

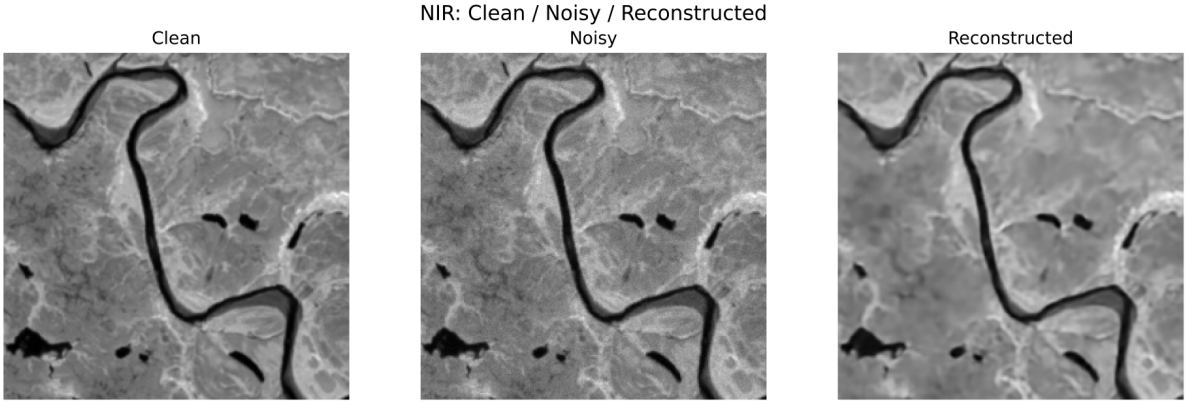

Figure 42: Near-infrared (NIR) clean, noisy, and decoded image with NAFCompression (noisy target and lambda=0.0001).

Figure 43: Near-infrared (NIR) clean, noisy, and decoded image with NAFCompression (noisy target and lambda=0.0001).

APPENDIX (Continued)

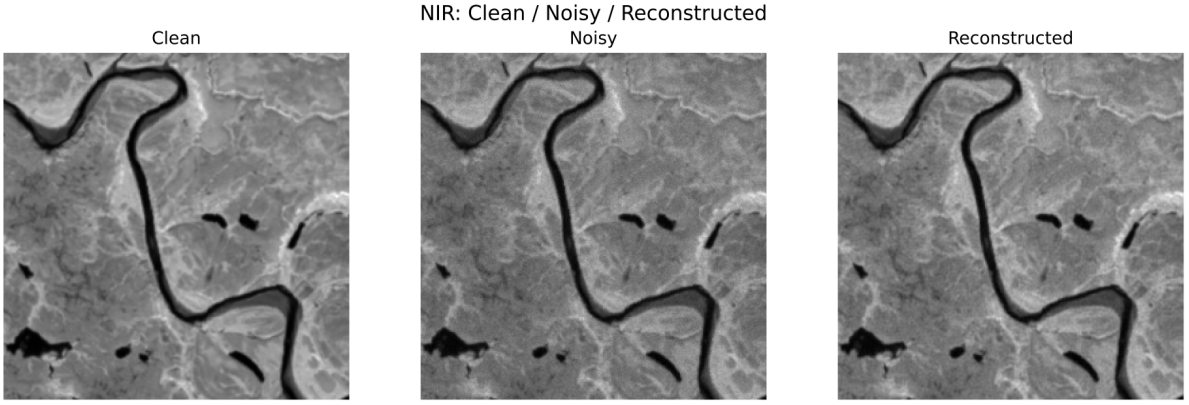

Figure 44: Near-infrared (NIR) clean, noisy, and decoded image with MeanScaleHyperprior (noisy target and lambda=0.01).

Figure 45: RGB clean, noisy, and decoded image with MeanScaleHyperprior (noisy target and lambda=0.01).

APPENDIX (Continued)

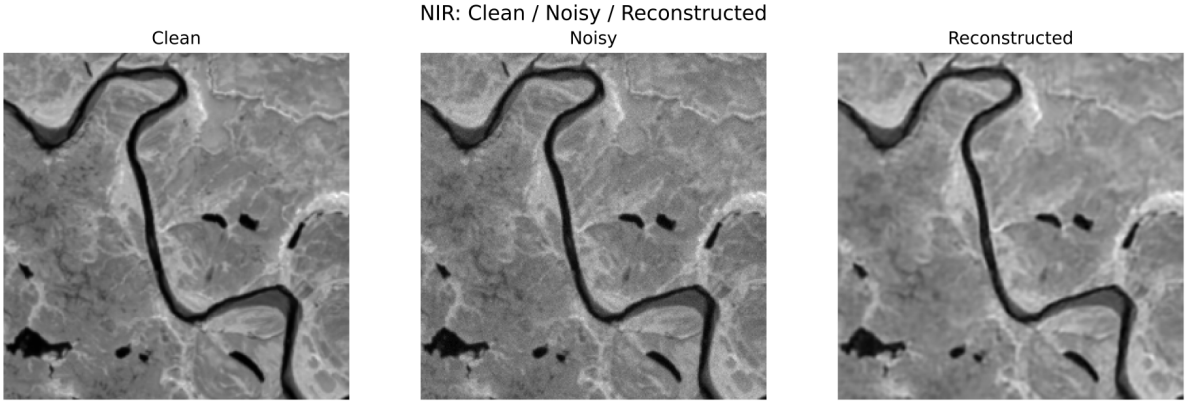

Figure 46: Near-infrared (NIR) clean, noisy, and decoded image with MeanScaleHyperprior (noisy target and lambda=0.0001).

Figure 47: RGB clean, noisy, and decoded image with MeanScaleHyperprior (noisy target and lambda=0.0001).

APPENDIX (Continued)

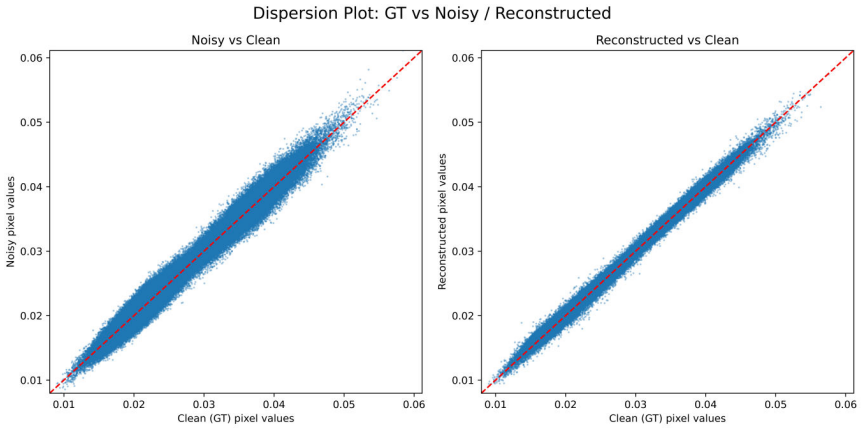

Figure 48: Second dispersion plot MeanScaleHyperprior trained with clean target.

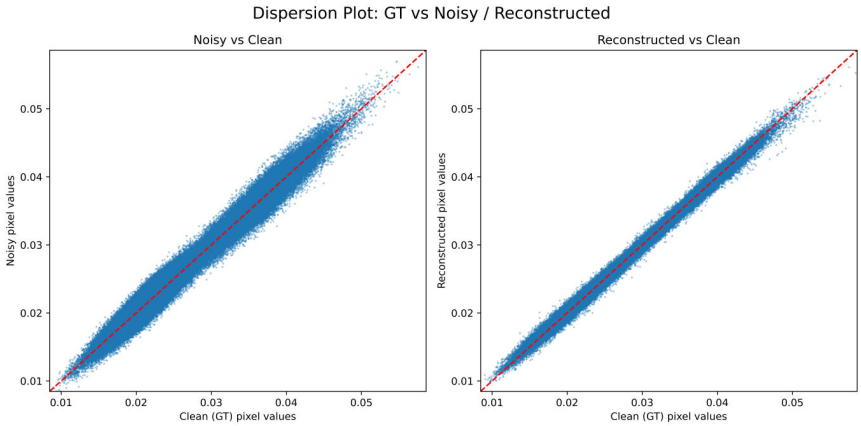

Figure 49: Second dispersion plot NAFCompression trained with clean target.

APPENDIX (Continued)

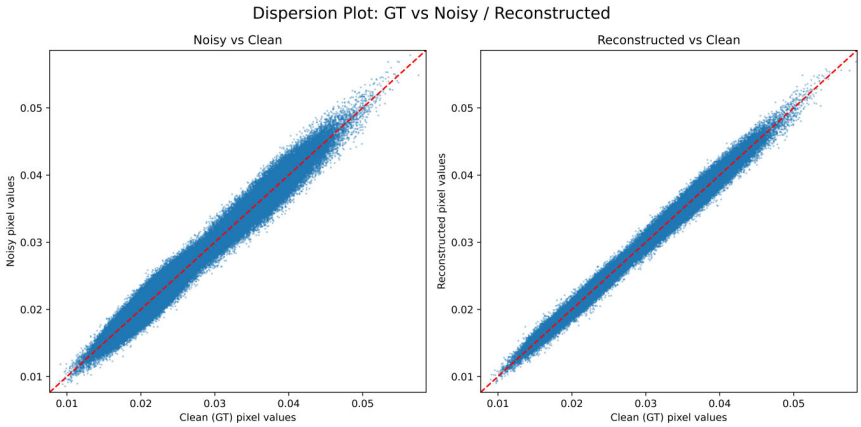

Figure 50: Second dispersion plot MeanScaleHyperprior trained with noisy target.

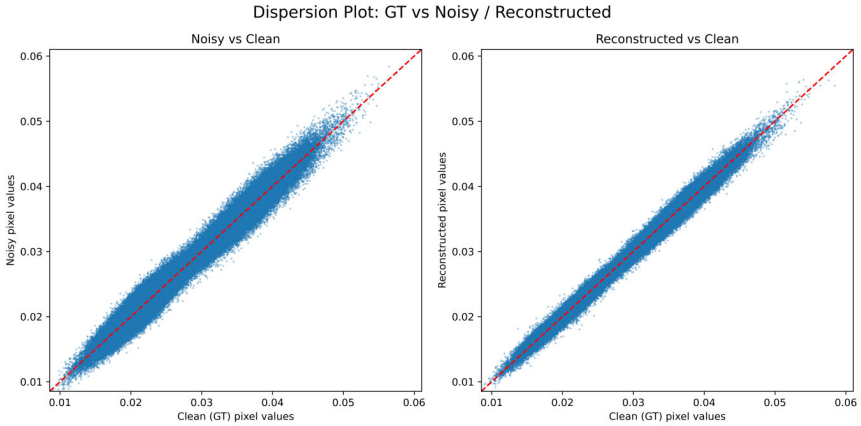

Figure 51: Second dispersion plot NAFCompression trained with noisy target.

APPENDIX (Continued)

Figure 52: RGB clean, noisy, and decoded versions of the second image, obtained with the NAFCompression (clean target and lambda=0.01).

Figure 53: RGB clean, noisy, and decoded versions of the second image, obtained with the NAFCompression (clean target and lambda=0.0001).

APPENDIX (Continued)

Figure 54: RGB clean, noisy, and decoded versions of the second image, obtained with the MeanScaleHyperprior (clean target and lambda=0.01).

Figure 55: RGB clean, noisy, and decoded versions of the second image, obtained with the MeanScaleHyperprior (target at lambda=0.0001).

VITA

NAME Matteo Carnevale Schianca

EDUCATION High School Diploma, Liceo Classico Vittorio Alfieri, Turin, Italy, June 2019
Bachelor's Degree, Electronic Engineering, Politecnico di Torino, Turin, Italy, 2022
Master of Science's Degree Candidate, Electrical and Computer Engineering, University of Illinois at Chicago and Politecnico di Torino, Expected 2025