

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut National Polytechnique de Toulouse (Toulouse INP)

Discipline ou spécialité :

Informatique et Télécommunication

Présentée et soutenue par :

M. VINICIUS ALVES DE OLIVEIRA

le vendredi 21 octobre 2022

Titre :

Aprentissage profond pour la compression embarquée d'images d'observation de la Terre

---

Ecole doctorale :

Mathématiques, Informatique, Télécommunications de Toulouse (MITT)

Unité de recherche :

Institut de Recherche en Informatique de Toulouse (IRIT)

Directeur(s) de Thèse :

MME MARIE CHABERT

M. CHARLY POULLIAT

Rapporteurs :

MME ALINE ROUMY, INRIA RENNES

M. PHILIPPE CARRE, UNIVERSITE DE POITIERS

Membre(s) du jury :

M. MARC ANTONINI, UNIVERSITE DE NICE SOPHIA ANTIPOLIS, Président

M. BRUNO MICKAEL, CENTRE NATIONAL D'ETUDES SPATIALES CNES, Invité(e)

M. CHARLY POULLIAT, TOULOUSE INP, Membre

MME MARIE CHABERT, TOULOUSE INP, Membre

M. SIMON HENROT, THALES ALENIA SPACE, Invité(e)

M. THOMAS OBERLIN, ISAE-SUPAERO, Membre

Acknowledgements

Words cannot express my gratitude to my academic supervisors Marie Chabert, Thomas Oberlin and Charly Poulliat for their invaluable patience and feedback during my thesis. I could not have undertaken this journey without Marie Chabert for her invaluable supervision, support and tutelage. I would also like to thank my industrial supervisors Christophe Latry, Frédéric Falzon, Mickaël Bruno, Roberto Camarero and Simon Henrot for their valuable contributions on my research. This endeavor would not have been possible without the support and immeasurable advises from Mickaël Bruno and Roberto Camarero who never gave up on me even when everything seemed impossible. Additionally, this endeavor would not have been possible without the financial support from the French Space Agency (CNES) and Thales Alenia Space (TAS).

I am also grateful to my colleagues and friends, especially my office mates in IRIT and TéSA, for their editing help, and moral support. Lastly, I would be remiss in not mentioning my family, especially my parents Aluizio and Cezamar and my brother Douglas. Their belief in me has kept my spirits and motivation high during this process.

Abstract

The new generation of satellite instruments enables the acquisition of images with ever-growing spectral and spatial resolutions. The counterpart is that an increasing amount of data must be processed and transmitted to the ground. Onboard image compression becomes thus crucial to preserve transmission channel bandwidth and reduce data transmission time. Recently, convolutional neural networks have shown outstanding results for lossy image compression compared to traditional compression schemes, however, at the cost of a high computational complexity. Autoencoder architectures are trained end-to-end, taking benefit from extensive datasets and computing power available on mighty clusters. Consequently, the potential contributions and feasibility of deep learning techniques for onboard compression are arousing great interest. In this context, nevertheless, computational resources are subject to severe limitations: a trade-off between compression performance and complexity must be established. In this thesis, the main objective is to adapt learned compression frameworks to onboard compression, simplifying them and training them with specific images. In a first step, we propose simplifying these architectures as much as possible while preserving high performance, particularly maintaining the adaptability to handle diverse input images. In a second step, we investigate how such architectures can further be improved by aggregating other functionalities such as denoising. Thus, we intend to incorporate denoising, either considering the above mentioned compression architectures for joint compression and denoising concurrently or as a sequential approach. The sequential approach consists in using, on the ground, a different architecture to denoise the images issued from the preceding learned compression framework. By running experiments on simulated but realistic satellite images, we show that the proposed simplifications to the learned compression framework result in considerably lower complexity while maintaining high performance. Concerning learned compression and denoising, the joint and sequential approaches are beneficial and complementary, allowing to surpass the CNES imaging system performance, and thus opening the path towards operational compression and denoising pipelines for satellite images.

Résumé

La nouvelle génération de satellites permet l'acquisition d'images avec des résolutions spectrales et spatiales toujours plus grandes. La contrepartie est qu'une quantité croissante de données doit être traitée et transmise au sol. La compression embarquée d'images devient donc cruciale pour préserver la bande passante du canal de transmission et réduire le temps de transmission des données. Récemment, les réseaux de neurones convolutifs ont montré des résultats exceptionnels pour la compression d'images avec perte par rapport aux schémas de compression traditionnels, au prix d'une complexité de calcul élevée. Les architectures basés sur l'autoencodeur sont entraînés de bout en bout, tirant parti de grandes bases de données et de la puissance de calcul disponible sur de puissants clusters. Par conséquent, les contributions potentielles et la faisabilité des techniques d'apprentissage profond pour la compression embarqué d'images satellitaires suscitent un grand intérêt. Dans ce contexte, néanmoins, les ressources de calcul sont soumises à de sévères limitations : un compromis entre performances de compression et complexité doit être établi. Dans cette thèse, l'objectif principal est d'adapter les architectures de compression appris à la compression embarquée, de les simplifier et de les entraîner avec des images spécifiques. Dans un premier temps, nous proposons de simplifier au maximum ces architectures tout en préservant des performances élevées, en conservant notamment l'adaptabilité pour traiter des images diverses. Dans un deuxième temps, nous étudions comment de telles architectures peuvent encore être améliorées en agrégeant d'autres fonctionnalités telles que le débruitage. Ainsi, nous avons incorporé le débruitage, soit en considérant les architectures de compression mentionnées ci-dessus pour la compression et le débruitage simultanément, soit en utilisant une approche séquentielle. L'approche séquentielle consiste à utiliser au sol une architecture différente pour débruiter les images compressées issues de l'architecture de compression appris précédent. En réalisant des expériences sur des images satellites simulées mais réalistes, nous montrons que les simplifications proposées pour l'architecture de compression appris entraînent une complexité considérablement moindre tout en maintenant des performances élevées. Tant en compression apprise qu'en débruitage appris aussi, les approches conjointes et séquentielles sont intéressantes et complémentaires, permettant de surpasser les perfor-

mances du système d'imagerie du CNES, et ouvrant ainsi la voie à des chaines opérationnels de compression et de débruitage des images satellites.

Contents

1 Introduction **1**
- 1.1 Context . . . 1
- 1.2 Purpose of the thesis . . . 1
- 1.3 Contributions . . . 2
  - 1.3.1 Outline of the thesis . . . 3

2 Overview of the satellite imaging system **5**
- 2.1 Overview of the satellite imaging system . . . 6
- 2.2 Modulation transfer function (MTF) . . . 6
- 2.3 Satellite image compression . . . 7
  - 2.3.1 Generalities on image compression through wavelet transform coding . . . 7
  - 2.3.2 JPEG2000 . . . 12
  - 2.3.3 Consultative committee for space data systems standard (CCSDS) 122.0-B-2 . . . 14
- 2.4 Satellite image denoising . . . 16
  - 2.4.1 Generalities on image denoising . . . 16
  - 2.4.2 Satellite imaging denoising system . . . 17
  - 2.4.3 Instrumental noise model . . . 17
  - 2.4.4 Instrument noise restitution . . . 18
  - 2.4.5 Variance stabilizing transform . . . 18
  - 2.4.6 NL-Bayes algorithm . . . 19
- 2.5 Deconvolution . . . 22
- 2.6 Dataset of simulated images . . . 23

3 Image compression and denoising with neural networks **25**
- 3.1 Generalities on neural networks . . . 26
- 3.2 Main neural architectures . . . 29
  - 3.2.1 Feedforward neural networks (FNNs) x recurrent neural networks (RNNs) . . . 29
  - 3.2.2 Convolutional neural networks (CNNs) . . . 30
  - 3.2.3 Autoencoders (AE) . . . 33
  - 3.2.4 Generative models . . . 34
- 3.3 Training neural networks . . . 35
  - 3.3.1 Loss function and gradient descent . . . 35
  - 3.3.2 Hyperparameters and model selection . . . 38

3.4 Compression with neural networks . . . 38
3.4.1 Generalities . . . 39
3.4.2 Analysis and synthesis transforms . . . 41
3.4.3 Loss function: rate distortion trade-off . . . 43
3.5 Denoising with neural networks . . . 49
3.5.1 General concepts . . . 49
3.5.2 Residual learning of deep CNN for image denoising (DnCNN) . . . 52
3.5.3 Toward a fast and flexible solution for CNN based Image denoising (FFDNet) . . . 54
3.5.4 Image denoising using deep CNN with batch renormalization (BRDNet) 55
3.5.5 Neural networks for compression artifacts suppression . . . 56
3.5.6 Denoising with GANs . . . 58

4 Learned reduced-complexity onboard satellite image compression 59
4.1 Focus on the two reference learning-based architectures . . . 60
4.1.1 Recalling the main characteristics . . . 60
4.1.2 Statistical analysis of the learned transform . . . 67
4.2 Reduced-complexity variational autoencoder . . . 71
4.2.1 Simplified analysis and synthesis transforms . . . 71
4.2.2 Simplified entropy model . . . 72
4.3 Performance analysis . . . 74
4.3.1 Implementation setup . . . 75
4.3.2 Subjective image quality assessment . . . 76
4.3.3 Impact of the number of filter reduction . . . 78
4.3.4 Impact of the filter kernel support in the main autoencoder . . . 84
4.3.5 Impact of the GDN/IGDN replacement in the main autoencoder . . . 84
4.3.6 Impact of the number of layer reduction in the main autoencoder . . . 86
4.3.7 Impact of the entropy model simplification . . . 88
4.3.8 Discussion about complexity . . . 90
4.4 Conclusions . . . 91

5 Satellite image compression and denoising with neural networks 93
5.1 Selected methods from state-of-the-art . . . 94
5.1.1 End-to-end trainable autoencoder for image compression . . . 94
5.1.2 Denoising with BRDNet . . . 95
5.2 Combining compression and denoising . . . 97
5.2.1 Joint compression and denoising . . . 97
5.2.2 Sequential compression and denoising . . . 98
5.2.3 Denoising as a post-processing after joint compression and denoising . 98
5.3 Performance analysis . . . 99
5.3.1 Implementation setup . . . 99
5.3.2 Compression-only performance of the joint compression and denoising method . . . 100

5.3.3 Compression and denoising performance 101
5.4 Conclusions 108

Conclusion **109**

A Complexity assessment **112**
A.1 Complexity assessment for the end-to-end compression method 112
A.2 Complexity assessment for the sequential denoising method 113

References **113**

List of Figures

2.1 Basic stages of the Pléiades image processing pipeline [Delvit et al. (2019)]. . 6
2.2 DWT; a) Sub-band structure, b) Pléiades image of Cannes, c) Pléiades image of Cannes at level 3 DWT decomposition. 9
2.3 Simulated 12-bit Pléiades images. 24

3.1 Three element-wise activation functions: sigmoid, tanh, and ReLU. 27
3.2 Example of an MLP with three hidden layers and one output layer. 28
3.3 Example of an unfolded computation graph. 30
3.4 Illustration of the convolutional operation. 32
3.5 Illustration of a fully-connected autoencoder. 33
3.6 Illustration of the stochastic mappings learned by the VAE [Kingma and Welling (2019)]. 34
3.7 Example of one unit $y_j^{(i)}$ corresponding to the layer i. 36
3.8 Architecture of the autoencoder [Ballé et al. (2017)]. 41
3.9 Architecture of the DnCNN network [K. Zhang et al. (2017a)]. 53
3.10 Architecture of the FFDNet network [K. Zhang et al. (2018)]. 54
3.11 Architecture of the BRDNet network [Tian et al. (2020)]. 56

4.1 Architecture of the autoencoder [Ballé et al. (2017)] (**left**) and of the variational autoencoder [Ballé et al. (2018)] (**right**). 61
4.2 Four-layer analysis transform G_a [Ballé et al. (2018)]. 63
4.3 Four-layer synthesis transform G_s [Ballé et al. (2018)]. 63
4.4 Three-layer hyperprior analysis transform H_a [Ballé et al. (2018)]. 66
4.5 Three-layer hyperprior synthesis transform H_s [Ballé et al. (2018)]. 66
4.6 Simulated 12-bit Pléiades image of Cannes with size 512×512 and resolution 70 cm 68
4.7 First feature of Cannes image representation, its normalized histogram with Laplacian fitting. 68
4.8 Normalized histogram of the ith feature map and Laplacian fitting $f(., \mu, b)$. . 69
4.9 Normalized histogram of the ith feature map and Laplacian fitting $f(., \mu, b)$. . 70
4.10 Proposed architecture after entropy model simplification: main autoencoder [Ballé et al. (2018)] (**left column**) and simplified auxiliary autoencoder (**right column**). 74
4.11 Subjective image quality analysis (R = 1.15 bits/pixel). 76
4.12 Subjective image quality analysis (R = 1.66 bits/pixel). 77
4.13 Subjective image quality analysis (R = 2.02 bits/pixel). 78

4.14 Rate-distortion curves for the considered learned frameworks and for the CCSDS 122.0-B [B. Book (2017)] and JPEG2000 [Marcellin and Taubman (2002)] standards in terms of MSE and MS-SSIM (dB) (derived as $-10\log_{10}(1-\text{MS-SSIM})$) - Case of rates up to 2 bits/pixel. 80
4.15 Rate-distortion curves at higher rates for learned frameworks and for the CCSDS 122.0-B [B. Book (2017)] and JPEG2000 [Marcellin and Taubman (2002)] standards for MSE in log-log scale - Case of high rates (above 2 bits/pixel). 82
4.16 Impact of the bottleneck size in terms of MSE and MS-SSIM (dB) (derived as $-10\log_{10}(1-\text{MS-SSIM})$) 83
4.17 Impact of the GDN/IGDN replacement and of the filter kernel support on performance in terms of MSE and MS-SSIM (dB) (derived as $-10\log_{10}(1-\text{MS-SSIM})$). 85
4.18 Impact of the number of layer reduction on performance in terms of MSE and MS-SSIM (dB) (derived as $-10\log_{10}(1-\text{MS-SSIM})$). 88

5.1 Architecture of the variational autoencoder [Ballé et al. (2018)]. 95
5.2 Architecture of the BRDNet network [Tian et al. (2020)]. 96
5.3 Sequential compression and denoising scheme. 98
5.4 Joint compression and denoising followed by denoising as a post-processing scheme. 98
5.5 Curves for MS-SSIM (dB) between the output image and the noisy image ($\mathbf{I_n}$). 101
5.6 Curves for MS-SSIM (dB) between the output image and the noise-free image ($\mathbf{I_{nf}}$). 102
5.7 Curves for PSNR (dB) between the output image and the noise-free image ($\mathbf{I_{nf}}$). 102
5.8 Example result on a test image. (a) `CNES-C+NL-Bayes` Delvit et al., 2019. (b) `AE-2-H-C+BRDNet`. (c) `AE-2-H-CD`. (d) Noise-free image ($\mathbf{I_{nf}}$). 104
5.9 Curves for MS-SSIM (dB) between the output image and the noise-free image ($\mathbf{I_{nf}}$). 105
5.10 Curves for MS-SSIM (dB) between the output image and the noise-free image ($\mathbf{I_{nf}}$). 106
5.11 Curves for MS-SSIM (dB) between the output image and the noise-free image ($\mathbf{I_{nf}}$). 106
5.12 Curves for PSNR (dB) between the output image and the noise-free image ($\mathbf{I_{nf}}$). 107

List of Tables

2.1 List of parameters for NL-Bayes and optimal values selected by experts in the original implementation [Masse et al. (2018)]. 20
2.2 Noise model parameters (A, B). 24

4.1 Detailed complexity of `AE-2-H-C (N=64, M=192)` 79
4.2 Comparative complexity of the global architectures - Case of target rates up to 2 bits/pixel. 79
4.3 Comparative complexity of the global architectures - Case of target rates above 2 bits/pixel. 81
4.4 Comparative complexity of the considered architectures - Case of reducing the number of layers. 87
4.5 Comparative complexity of the considered architectures on the hyperprior part- Case of reducing the number of layers. 87
4.6 Reduction of the encoder complexity induced by the Laplacian entropy model on the coding part - Case of rates up to 2 bits/pixel. 89
4.7 Reduction of the encoder complexity induced by the Laplacian entropy model on the coding part - Case of rates above 2 bits/pixel. 90

5.1 Comparative complexity of the considered architectures 103

Notations and acronyms

Generic notations

$\mathbf{x}$	original image	4.1.1.1
$\mathbf{y}$	learned representation	4.1.1.1
$\hat{\mathbf{y}}$	quantized learned representation	4.1.1.1
$\hat{\mathbf{x}}$	reconstructed image	4.1.1.1
N	number of filters composing the convolutional layers	4.1.1.1
$n \times n$	kernel support size	4.1.1.1
M	number of filters composing the last layer of G_a	4.1.1.1
$v_i(k,l)$	value indexed by (k,l) of the output of the ith filter	4.1.1.1
$G_a(.)$	analysis transform	4.1.1.1
$G_s(.)$	synthesis transform	4.1.1.1
$m(\hat{\mathbf{y}})$	actual discrete probability distribution of the learned representation	4.1.1.2
$p_{\hat{\mathbf{y}}}(\hat{\mathbf{y}})$	probability model assigned to the quantized representation	4.1.1.2
λ	parameter that tunes the rate-distortion trade-off	4.1.1.2
$R(\hat{\mathbf{y}})$	rate	4.1.1.2
$D(\mathbf{x},\hat{\mathbf{x}})$	distortion between the original image $\mathbf{x}$ and the reconstructed image $\hat{\mathbf{x}}$	4.1.1.2
J	rate-distortion loss function	4.1.1.2
$H(\hat{\mathbf{y}})$	bit-rate given by the Shannon cross entropy	4.1.1.2
$H_a(\mathbf{y})$	hyperprior analysis transform	4.1.1.3
$H_s(\hat{\mathbf{z}})$	hyperprior synthesis transform	4.1.1.3
$\boldsymbol{\psi}^{(i)}$	distribution model parameter vector associated to each element	4.1.1.3
$\tilde{\mathbf{y}}$	continuous approximation of the quantized learned representation	4.1.1.3
$\mathbf{z}$	set of auxiliary random variables	4.1.1.3
I_j	set of indexes covering the jth feature map	4.2.2
$\mathbf{I_{nf}}$	noise-free image	5.1.2
$\mathbf{I_n}$	noisy image	5.1.2
$\hat{\mathbf{I}}_{\mathbf{nf}}$	predicted denoised image	5.1.2
$\hat{\mathbf{I}}_{\mathbf{n}}$	compressed noisy image	5.1.2
$f(\mathbf{I_n})$	residual image	5.1.2
N_{in}	number of features at the considered layer input	A.1
N_{out}	number of features at the considered layer output	A.1
Param^f	number of parameters associated to the filtering part of the considered layer	A.1
δ	term accounting for the bias of a convolutional filter	A.1
D	downsampling factor	A.1
s_{in}	size of a feature map input	A.1
s_{out}	size of a feature map output	A.1

Generic notations

Operation^f	number of floating points operations	A.1
Param^g	number of parameters associated to each IGDN/GDN	A.1
Operation^g	number of floating points operations of each GDN/IGDN	A.1
Param^b	number of parameters associated to each BRN	A.2
Operation^b	number of floating points operations of each BRN	A.2

Acronyms

MTF	modulation transfer function	2.2
CCSDS	consultative committee for space data systems	2.3.1
JPEG	Joint Photographic Experts Group	2.3.1.1
VST	variance stabilizing transform	2.4.5
MSE	mean squared error	4.1.1.2
PSNR	peak signal to noise ratio	3.4.3.2
MS-SSIM	multi-scale structural similarity	3.4.3.2
RL	residual learning	3.5.1
BN	batch normalization	3.5.1
BRN	batch renormalization	3.5.1
DnCNN	denoising convolutional neural network	3.5.2
FFDNet	flexible denoising network	3.5.3
BRDNet	batch-renormalization denoising network	3.5.4
GDN	generalized divisive normalizations	4.1.1.1
IGDN	inverse generalized divisive normalizations	4.1.1.1
CDF	cumulative distribution function	4.1.1.2
FLOPp	floating point operation per pixel	4.3.3.1

Chapter 1

Introduction

1.1 Context

The new generation of satellite instruments enables the acquisition of images with ever-growing spectral and spatial resolutions. The counterpart is that an increasing amount of data must be processed and transmitted to the ground. Satellite image compression becomes thus essential to preserve transmission channel bandwidth and reduce data transmission effort [Huang (2011)]. Traditional lossy image compression frameworks use transform coding [Goyal (2001)] in association with optimization of a rate-distortion criterion. Concerning computational limitations onboard satellites, the consultative committee for space data systems standard (CCSDS) 122.0-B-2 compression algorithm was designed, taking into account a compromise between complexity and performance [B. Book (2017)].

1.2 Purpose of the thesis

The primary aim of the thesis is to measure the potential contributions and the feasibility of deep learning techniques for onboard satellite compression. Recently, artificial neural networks emerged as powerful data-driven tools capable of outperforming model-based methods in solving many complex problems. In data-driven methods, a learning phase on a sufficiently representative database replaces the extraction of relevant features, which results from a prior choice in classical methods and usually requires substantial human efforts. Deep learning-based methods can attain very competitive results when large datasets and computing power are available for the model training phase. Currently, these two conditions are satisfied in many applications, which explains the intense interest aroused by these methods in diverse domains. Within the context of the thesis, we can have access to exten-

sive databases representative of different landscapes, and the CNES provided the data used in the thesis. Notably, convolutional neural networks (CNNs) have been successful in many computer vision applications [Bengio (2009)] such as classification [J. Hu et al. (2018)], object detection [Redmon et al. (2016)], segmentation [Kaiser et al. (2017)], compression artifacts removal [Dong et al. (2015)] and denoising [K. Zhang et al. (2017a)]. Recently, convolutional neural networks (CNNs) have shown outstanding results for lossy image compression [Ballé et al. (2017), Theis et al. (2017), Rippel and Bourdev (2017), and Ballé et al. (2018)] when compared to traditional compression schemes in terms of rate-distortion trade-off, however, at the cost of a high computational complexity.

In the context of satellite image compression, a trade-off between compression performance and complexity must be considered due to significant computational constraints onboard satellites. For this reason, the main objective of this thesis is to evaluate, both in terms of performance and complexity, learning-based approaches for embedded compression of Earth observation images. Thus, the impact of the different design options on the compression performance should be exhaustively studied for various compression rates, and a comparative complexity study should also be performed.

Moreover, a complementary goal is to integrate other features into the embedded compression solution. There are other challenges related to satellite images, e.g., the semi-multiplicative noise that affects images in their acquisition and requires proper denoising. Thus it would be advantageous to take advantage of neural networks to address both compression and denoising, taking for reference a typical satellite imaging system [Delvit et al. (2019)].

This thesis is co-financed by the French Space Agency (CNES) and Thales Alenia Space (TAS) and co-supervised by the European Space Agency (ESA).

1.3 Contributions

- Experimental analysis of different design options for learned satellite image compression.
- Simplified Laplacian entropy model.
- Joint versus sequential compression and denoising with neural networks.

1.3.1 Outline of the thesis

This document will be divided into two parts. The first one will give a background on the satellite imaging system (see chapter 2) and will deal with the state-of-the-art in image compression and denoising with neural networks (see chapter 3). The second part gathers the contributions of the PhD (see chapters 4 and 5).

List of Publications

Published journal papers

[Alves de Oliveira et al. (2021)]
Alves de Oliveira, V., M. Chabert, T. Oberlin, C. Poulliat, M. Bruno, C. Latry, M. Carlavan, S. Henrot, F. Falzon, and R. Camarero (2021). "Reduced-Complexity End-to-End Variational Autoencoder for on Board Satellite Image Compression," *In Remote Sensing* 27.jan, 2021, 13(3), 447.
Available on: https://doi.org/10.3390/rs13030447

[Alves de Oliveira et al. (2022)]
Alves de Oliveira, V., M. Chabert, T. Oberlin, C. Poulliat, M. Bruno, C. Latry, M. Carlavan, S. Henrot, F. Falzon, and R. Camarero (2022). "Satellite Image Compression and Denoising With Neural Networks," in IEEE Geoscience and Remote Sensing Letters, vol. 19, pp. 1-5, 2022, Art no. 4504105.
Available on: https://doi.org/10.1109/LGRS.2022.3145992

International conferences

[Alves de Oliveira et al. (2020)]
Alves de Oliveira, V., T. Oberlin, M. Chabert, C. Poulliat, B. Mickael, C. Latry, M. Carlavan, S. Henrot, F. Falzon, and R. Camarero (2020). "Simplified entropy model for reduced-complexity end-to-end variational autoencoder with application to on-board satellite image compression". Paper presented at the *7th International Workshop on On-Board Payload Data Compression (OBPDC 2020)*.
Available on: https://hal.archives-ouvertes.fr/hal-03079863/document

Chapter 2

Overview of the satellite imaging system

Contents

2.1 Overview of the satellite imaging system 6
2.2 Modulation transfer function (MTF) 6
2.3 Satellite image compression 7
2.3.1 Generalities on image compression through wavelet transform coding . 7
2.3.2 JPEG2000 12
2.3.3 Consultative committee for space data systems standard (CCSDS) 122.0-B-2 14
2.4 Satellite image denoising 16
2.4.1 Generalities on image denoising 16
2.4.2 Satellite imaging denoising system 17
2.4.3 Instrumental noise model 17
2.4.4 Instrument noise restitution 18
2.4.5 Variance stabilizing transform 18
2.4.6 NL-Bayes algorithm 19
2.5 Deconvolution 22
2.6 Dataset of simulated images 23

2.1 Overview of the satellite imaging system

A classical satellite imaging system is composed of three parts: acquisition, onboard compression, and restoration [Latry et al. (2012)]. The acquisition captures the image scene, onboard compression reduces the image data size to reduce data transmission resources, and restoration aims to reduce degradations originating from the previous steps. We assume that the transmission does not introduce any degradation on the previously compressed image [Carlavan et al. (2012)]. For Earth observation, the acquisition causes a blurring effect due to the effects of instrumental optics [S. M. Wong et al. (2020)]. This undesirable effect can be modeled by the Point Spread Function (PSF) or the Modulation Transfer Function (MTF) in the frequency domain, which is used to characterize the quality of electro-optical imaging systems and, in particular, those onboard satellites. In a typical satellite imaging system, noise can be classified into two main categories depending on the source: instrumental noise and noise arising from the subsequent processing [Latry et al. (2012)]. Instrumental noise arises from acquisition. Its statistics are well-known and present a pixel-dependent variance [Masse et al. (2018)]. Unlike instrumental noise, noise issued from the subsequent processing presents a non-uniform spectral density [Latry et al. (2012)]. It may be introduced by radiometric corrections and onboard compression [Latry et al. (2012)]. Image denoising becomes thus necessary to recover a noise-free image from the received noisy image. It is also needed because otherwise, inverting the MTF by amplifying the high-frequency values would increase the noise. Once the noise has been adequately removed, deconvolution takes place to reverse the MTF effect.

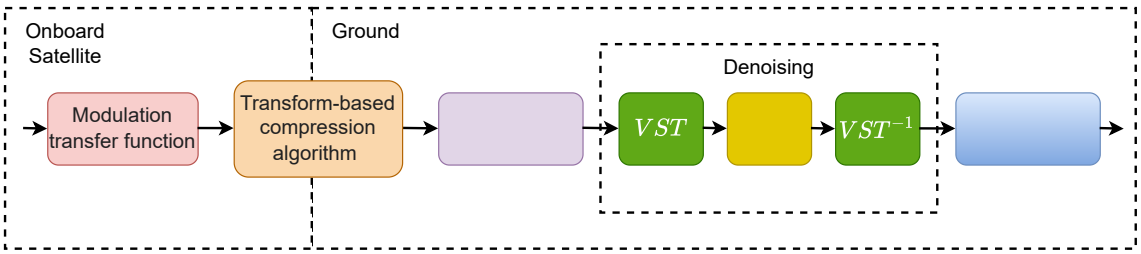


Figure 2.1: Basic stages of the Pléiades image processing pipeline [Delvit et al. (2019)].

2.2 Modulation transfer function (MTF)

Considering the acquisition sensor as a shift invariant linear system, the relation between the radiances of the landscape and the image in the spatial domain is formulated as [S. M.

Wong et al. (2020)]:

$$\mathbf{I}_n(x,y) = h(x,y) * \mathbf{I}_a(x,y) + \Psi(x,y), \tag{2.1}$$

where $\mathbf{I}_a(x,y)$, $h(x,y)$ and $\Psi(x,y)$ represent the landscape image, the point spread function (PSF) and the instrumental noise respectively; $*$ denotes two-dimensional (2D) convolutional operator and (x,y) represents the coordinates. The degradation model in the frequency domain can be expressed as [S. M. Wong et al. (2020)]:

$$\bar{\mathbf{I}}_n(v,w) = H(v,w) \cdot \bar{\mathbf{I}}_a(v,w) + \overline{\Psi}(v,w), \tag{2.2}$$

where v and w represent the frequency coordinates, $\overline{\Psi}(v,w)$ denotes the noise spectrum and H denotes the optical transfer function, which amplitude spectrum is the modulation transfer function (MTF). The symbol $\cdot$ denotes the element-wise multiplication operator. In the acquisition system, when considering an optimized satellite imaging system, the modulation transfer function (MTF) assumes a relatively low value at Nyquist frequency to reduce both telescope diameter and aliasing effects, which is called a diffraction-limited design [Latry et al. (2012)].

2.3 Satellite image compression

2.3.1 Generalities on image compression through wavelet transform coding

Images are composed of pixels, which are made of combinations of primary colors if the images are RGB or spectral bands if the images are multispectral. For instance, an RGB image from a standard digital camera is composed of red, green, and blue channels. In the context of RGB images, a channel is the intensity image made of just one of these primary colors. A grayscale image has just one channel. A multispectral image is composed of several channels or bands, each one containing the amount of radiation measured in a particular wavelength. A panchromatic image is formed from a wide spectrum band that includes almost all visible wavelengths, allowing for a greater spatial resolution in a single band. In this case, the result does not contain any wavelength-specific information [Ose et al. (2016)].

Images present two types of redundancy: spectral and spatial [Subramanya (2001)]. Image compression aims to reduce the number of bits required to represent an image. In multispectral and color images, spectral redundancy generally arises from the correlation between the

different bands. Spatial redundancy is due to the correlation between neighboring pixel values. Before compression, color images are usually transformed to a luminance-chrominance representation. The purpose of this transformation is to decorrelate the spectral or color bands so they can be compressed separately and efficiently [Tretter et al. (2005)]. The wavelet transform, in its turn, represents data with reduced spatial entropy, making data more prone to be compressible than its original form [Marcellin and Taubman (2002)]. An inverse process is required to obtain the reconstructed data from the transformed data. The wavelet transform coding thus fits well in the context of image compression. Consequently, many image compression algorithms rely on wavelet transform coding such as JPEG2000 [Marcellin and Taubman (2002)] and the consultative committee for space data systems standard (CCSDS) 122.0-B-2 [G. Book (2015)]. Wavelet transform allows compressing images while keeping most of their visual quality. Its main concepts and principles will be presented in the following.

2.3.1.1 Wavelet transform

Wavelets are a family of basis functions. The wavelet transform separates a function or a signal into distinct frequency packets localized in the time domain [Panda et al. (2000)]. Thus, wavelets are well suited for analyzing non-stationary data. In other words, a function or a discrete data set, when transformed into a time-scale space using wavelets, shows how it behaves at different measurement scales. Thus, wavelet transforms provide a multiresolution framework for image representation. When wavelets have compact support, it is easier to perform transforms to large data sets with minimal computations [Panda et al. (2000)].

The wavelet transform consists of low-pass and high-pass image data in the vertical or horizontal direction and then in the other direction, followed by sub-sampling [Marcellin and Taubman (2002)]. Wavelet transform usually decomposes the original image into a sequence of new images (usually called wavelet subbands) of decreasing size. This process creates a set of levels of wavelet decomposition that represent the image viewed at different scales.

The discrete wavelet transform (DWT) is an implementation of the wavelet transform using a discrete set of the wavelet scales and translations [Marcellin and Taubman (2002)]. One of the main advantages of the DWT over the continuous counterpart is that the DWT does not add redundancy if decomposed in a wavelet basis, i.e., it represents the transformed data using only as many coefficients as present in the original data, which makes DWT particularly suitable to image compression [Braun et al. (2002)]. Both low-pass and

high-pass filtering result in redundant coefficients, as each filtering operation returns transformed coefficients that maintain the original input size. Consecutively, the resulting data is further downsampled by a factor of two [Marcellin and Taubman (2002)]. Downsampling is performed to ensure that the subspace resolution of the transformed image is not oversampled with respect to the original image. The downsampled data can be reconstructed with the corresponding inverse wavelet transform. The DWT decomposes each image component into multiple frequency bands called subbands. For each level, DWT is applied row-wise and column-wise and results in four sub-bands [Hayat et al. (2009)]:

1 horizontally and vertically low-pass (LL),

2 horizontally low-pass and vertically high-pass (LH),

3 horizontally high-pass anv vertically low-pass (HL),

4 horizontally and vertically high-pass (HH).

R different resolution levels are associated to $(R-1)$-level wavelet decomposition. Each sub-band of the decomposition is identified by its orientation and its corresponding decomposition level (0, 1, ... R-1). The LL band is further decomposed at each resolution level (except the lowest LL_{R-1}). The LL_0 band is decomposed to originate the LL_1, LH_1, HL_1 and HH_1 bands. Consecutively, at the next level, as illustrated in figure 2.2a, the LL_1 band is decomposed. This same process is repeated until LL_{R-1} band is obtained. A level-2 wavelet decomposition of a Pléiades image of Cannes, provided in figure 2.2b, is illustrated in figure 2.2c.



Figure 2.2: DWT; a) Sub-band structure, b) Pléiades image of Cannes, c) Pléiades image of Cannes at level 3 DWT decomposition.

The 2D transformation can be carried out by separately applying the 1D version horizontally and vertically one after the other. For example, when considering the 1D information

(pixel row or pixel column) input signal $S_1, ...S_n$, the low-pass sub-band signal $L_1, ...L_{n/2}$ and high-pass sub-band signal $H_1, ...H_{n/2}$ for the reversible integer-to-integer Daubechies (5/3) wavelet transform are expressed by:

$$H_i = S_{2i+1} - \left\lfloor \frac{1}{2}(S_{2i+2} + S_{2i}) \right\rfloor, \tag{2.3}$$

$$L_i = S_{2i} - \left\lfloor \frac{1}{4}(H_i + H_{i-1}) + \frac{1}{2} \right\rfloor, \tag{2.4}$$

where $\lfloor \cdot \rfloor$ is the rounding operator. Now, when considering the irreversible real-to-real Daubechies (9/7), we define a new set of variables $S_1^{'}, ...S_n^{'}$ where the odd numbered variables ($S_{2n+1}^{'}$) will stand for the first stage lifting outcomes and the even numbered ($S_{2n}^{'}$) represents those of second stage lifting. For real-to-real Daubechies (9/7) wavelet transform, we have [Hayat et al. (2009)]:

$$S_{2i+1}^{'} = S_{2i+2} + a(S_{2i} + S_{2i+2}), \tag{2.5}$$

$$S_{2i}^{'} = S_{2i} + b(S_{2i-1}^{'} + S_{2i+1}^{'}), \tag{2.6}$$

$$H_i = \beta^{'}(S_{2i+1}^{'} + c(S_{2i}^{'} + S_{2i+2}^{'})), \tag{2.7}$$

$$L_i = \beta(S_{2i}^{'} + d(H_{i-1} + H_i)), \tag{2.8}$$

where a, b, c and d are the first, second third and fourth stage parameters, and $\{\beta, \beta^{'}\}$ represent the scaling parameters. This wavelet is used in the JPEG2000 image compression standard [Hayat et al. (2009)].

DWT results in better energy compaction that the discrete cosine transform (DCT) without blocking artifacts after coding. Moreover, the DWT decomposes the image into an L-level dyadic wavelet pyramid [Hayat et al. (2009)]. The resulting wavelet coefficient can be easily scaled in resolution and the wavelet coefficients smaller than a given threshold can be discarded to reconstruct an image with a lower level of detail [Hayat et al. (2009)]. The multi-resolution nature of DWT makes it proper for scalable image coding.

2.3.1.2 Quantization

Quantization is the process of mapping input values from a large set (often a continuous set) to output values in a (countable) smaller set, often with a finite number of symbols. Rounding and truncation are typical examples of quantization processes. Quantization forms the core of essentially all lossy compression algorithms. Its main drawback is the introduced

degradation resulting from the discarded information. The challenge is to quantize efficiently and minimize its impact on quality. In an elementary lossy compression method, each transformed sample (y_i) is independently mapped to a corresponding quantized value ($\hat{y}_i$). This process is known as scalar quantization process which represents the simplest quantization form [Marcellin and Taubman (2002)]. Each $\hat{y}_i$ is associated with an interval on the real range of values, as follows:

$$\hat{y}_i = q_k \quad \text{if } y_i \in \mathrm{I}_k, \tag{2.9}$$

where the intervals I_k are disjoint and cover the real line. In a simple approach, q_k might be selected as the mid-point of the interval I_k. Quantization is also commonly associated with variable quality when varying the quantization step size. Thus, the uniform scalar quantization can be expressed as follows [Xing et al. (2015)]:

$$\hat{y}_i = \lfloor y_i / Q_{step} \rfloor, \tag{2.10}$$

where Q_{step} is the quantization step size and $\lfloor \cdot \rfloor$ is the rounding operator. Respectively, the approximate inverse quantization operator (Q^{-1}) maps each $\hat{y}_i$ back to the original corresponding interval, resulting in $\tilde{y}_i$ [Marcellin and Taubman (2002)]. The inverse quantization mapping can thus be expressed as:

$$\tilde{y}_i = \hat{y}_i \times Q_{step}. \tag{2.11}$$

There are more sophisticated quantization schemes such as adaptive scalar quantization and vector quantization. Adaptive scalar quantization has been proposed to minimize quantization errors arising from input values which are not uniformly distributed. Vector quantization also aims to reduce quantization errors, but on multidimensional inputs.

2.3.1.3 Entropy coder

Compression essentially aims to remove redundancies in the original data. In this context, wavelet transform targets to transform data to obtain a less redundant representation [Marcellin and Taubman (2002)], as previously seen in 2.3.1.1. From information theory [Marcellin and Taubman (2002)], further compression can be achieved by using entropy coding. Entropy coding allows representing information in the most compact form losslessly. Note that entropy coding requires data in an integer form, which can be obtained through the quantization process. The reverse of this conversion is to be performed during decoding.

One of the main types of entropy coding creates and assigns a unique prefix code to each unique symbol that occurs in the input data [Z.-M. Lu and Guo (2017)]. These entropy encoders compress data by representing each fixed-length input symbol with the corresponding variable-length prefix codeword. The length of each codeword is approximately proportional to the negative logarithm of the probability, which comes from Shannon's information theory [Shannon (1948)]. For this, entropy coding exploits the probability distribution patterns of its input data. Therefore, the most common symbols use the shortest codes. Three commonly used entropy coding techniques are Huffman coding, arithmetic coding, and Lempel-Ziv coding. Lempel-Ziv coding is a dictionary based coding, which is suitable for compression of text data [Khan et al. (2013)].

Currently, arithmetic coding is the most popular entropy coding technique used in many image compression standards, including JPEG2000. Note that computational complexity has been an issue of concern in this context. The range coder [Martin (1979)], an entropy coder similar to the arithmetic coder or Huffman coder, partially addressed this issue. The compressed files are minimally larger (less than 0.01% in most cases) than an arithmetic coder, but the operation speed is nearly twice as fast. Arithmetic coding takes advantage of interval subdivision, where successive input symbols are encoded as intervals on the range [0,1) based on their probability of occurrence. Arithmetic coding is superior to Huffman coding since it can assign a fractional number of bits for the codewords of the symbols. In contrast, Huffman coding only supports an integral number of bits [Huang et al. (2004)]. In the CCSDS standard [G. Book (2015)], Rice coding is a recommendation for lossless compression of satellite data. The Rice coding is an adaptive entropy coder which uses a subset of the family of Golomb codes to produce a simpler suboptimal prefix code, resulting in lower computational complexity. Experiments in [Huang et al. (2004)] show that arithmetic coding is superior to Rice coding in terms of compression ratio performance, exhibiting a slighter higher complexity. Later in this thesis, we consider arithmetic coding in our learned-based approaches due to its higher performance.

2.3.2 JPEG2000

In the mid-80s, both the International Telecommunication Union (ITU) and the International Organization for Standardization (ISO) worked to establish a joint international standard for grayscale and color image compression. This effort has been known as Joint Photographic Experts Group (JPEG) [Christopoulos et al. (2000)]. After evaluating several different coding schemes, the JPEG members selected a DCT-based method. The image

is partitioned into rectangular blocks of size 8x8, and each block is transformed separately with the DCT. The main disadvantage of the image compression schemes based on the DCT is that the transform does not reveal any information about the space localization of the frequency components. Because of that, images must be partitioned into blocks that are transformed separately. At high compression ratios, the approximations performed in the quantization step can create significant differences between the neighboring pixels at the edge of the blocks. Consequently, the DWT emerged as one of the alternatives to the DCT block-based transform, which later got the JPEG group's approval. Thus, an evolution of the image compression technology has taken place and an improved standard called JPEG2000 has been established [Christopoulos et al. (2000)]. The primary motivation was to find a more efficient transform and add new features. The JPEG2000 adopted wavelet transforms. JPEG2000 disposes of many attractive features, which were not present in the previous image compression standards. They include [Chai and Bouzerdoum (2001)]:

- Excellent coding performance: it exhibits superior rate-distortion and subjective image quality performance, especially at low bit-rates. This is essential in applications whereby the file size or transmission time is critical.
- Lossless and lossy compression: it supports lossless compression, which is important to applications such as medical imagery and image archival.
- Region of Interest (ROI) coding: it allows to encode some areas of an image with higher quality.
- Spatial and signal-to-noise ratio (SNR) scalability: it supports progressive recovery of images by resolution or quality.
- Good error resilience: bitstream robustness to the presence of bit errors has been improved.

The JPEG2000 encoder follows a common sequence of operations present in a transform coding scheme, which consists of transformation, quantization and entropy coding [Chai and Bouzerdoum (2001)]. The JPEG2000 encoder works as follows. First, the original image with unsigned data is average (DC) level shifted. Then the channel transformation can be performed if the original image has multiple channels (e.g., RGB image). JPEG2000 performs DWT transform that is superior to DCT since it localizes the frequency components in space and does not require the partitioning of images.

Since its beginning, JPEG2000 has been supporting two kinds of biorthogonal transforms: the reversible integer-to-integer Cohen–Daubechies–Feauveau (5/3) and the irreversible real-to-real Cohen–Daubechies–Feauveau (9/7) [Hayat et al. (2009)]. The 2D transformation can

be carried out separately by applying the 1D version horizontally and then vertically sequentially. The wavelet transform decomposes the tile components into different decomposition levels, each containing a certain number of subbands filled with transform coefficients. Before the entropy coding phase, the quantization takes place to reduce the precision of the components. Then, a bit-plane coding technique with 3 passes is applied to each code-block, and the produced symbols are coded using an adaptive binary arithmetic coder [Chai and Bouzerdoum (2001)]. The bit-plane encoding technique used by JPEG2000 is based on the algorithm proposed in [D. Taubman (2000)] called Embedded Block-based Coding with Optimized truncation (EBCOT). The EBCOT algorithm allows elevate compression performance along with attractive features such as SNR scalability, resolution scalability and random access capability.

2.3.3 Consultative committee for space data systems standard (CCSDS) 122.0-B-2

The consultative committee for space data systems standard (CCSDS) defines a compression algorithm intended for use onboard satellites. In particular, a trade-off between performance and complexity is established to allow high-speed hardware implementation [G. Book (2015)]. Moreover, the CCSDS standard supports a memory-efficient implementation which does not require large intermediate frames for buffering. Thus, the algorithm is appropriate for both frame-based image formats (two dimensions acquired simultaneously) and push-broom input formats (i.e., images acquired one line at a time) [G. Book (2015)]. The compressor consists of two functional parts: a DWT module which performs the DWT decomposition and a Bit-Plane Encoder (BPE) which encodes the transformed data.

The CCSDS standard supports grayscale images with integer-valued pixels with a maximum dynamic range of up to 28 bits [G. Book (2015)]. The recommended standard can provide both lossy and lossless compression. The compressor relies on DWT, which supports integer and floating point DWT. In order to mitigate the effects of data loss that may arise from the communication channel, the wavelet-transformed data are partitioned into different segments [G. Book (2015)]. Each segment is compressed independently thus any eventual data loss is limited to the affected segment. The partitioning in different segments also benefits from limiting the memory requirements for some applications. The CCSDS standard differs from the JPEG2000 in many aspects [G. Book (2015)]:

- It specifically targets use onboard satellites;
- It establishes a careful trade-off between compression performance and complexity;

- It facilitates implementation in both hardware and software which benefits from lower complexity;
- It supports a limited set of options, simplifying its application.

The CCSDS standard floating-point DWT is the same as the one included in the JPEG2000. However the recommended integer DWT is a 9/7 DWT for CCSDS and 5/3 DWT for JPEG2000. Empirically, the 9/7 integer DWT resulted in better compression performance when other parts of the algorithm are fixed [G. Book (2015)]. The number of levels of the wavelet decomposition is fixed at three in the CCSDS standard, while it can be up to five in JPEG2000. In general, the JPEG2000 disposes of functionalities that are not available under the CCSDS standard. However, beyond the additional computational complexity, these features also come at the expense of higher overhead that may be significant, especially for small images.

The JPEG2000 standard features error-resilient bit-stream and extra tools to improve performance when transmitting compressed images over noisy channels. The CCSDS standard provides error containment at the segment level. Considering the JPEG2000, the entropy coder calculates the resulting reduction in mean squared error (MSE) distortion for each coding pass and code-block. An additional post-processing operation passes over the compressed blocks to determine the extent to which each code-block's bit stream should be truncated to reach a target bit rate or quality metric. The JPEG2000's rate control approach presents two significant drawbacks [G. Book (2015)]. First, the compressed data value generated by the JPEG2000 rate control technique may not correspond precisely to the desired target bit rate, unlike the CCSDS standard. The users may experience an inconsistent output rate. Note that this rate control procedure relies on an iterative algorithm based on Lagrangian multipliers, which results in high implementation complexity [G. Book (2015)].

The CNES also developed a proprietary compressor controlled by a target quality set-point per block based on the CCSDS standard 122.0-B-2 [Thiebaut et al. (2016)]. The CNES has been working on the characterization of compression errors to find local mathematical criteria to describe the strict and subjective image quality requirements imposed by final users. Indeed, classical global measures such as peak signal-to-noise ratio (PSNR), MSE, or Maximum Error, cannot represent a reliable indicator of the actual image quality on a local perspective because they are computed for the entire image [Thiebaut et al. (2016)]. The CNES has studied mathematical relationships between image complexity and the corresponding quality requirement (or targeted error) by considering other measurable

characteristics in the image (such as its local variance and noise level). The encoding procedure of the CCSDS standard is modified to leverage the local instrument noise level, an ROI map, and the local variance in the wavelet domain to achieve the targeted quality. The reached performances depend on the algorithm's parameters. The first parameter to be set is the instrumental noise threshold which allows the algorithm to perform a sort of low-complexity onboard hard-denoising. It prevents the algorithm from coding noise areas and most of the noise in each block. The second parameter is the local quality target expressed as the compression MSE. A variance-dependent look-up table can define this one in the so-called quality-controlled compression mode or as a factor of the local instrumental noise in the so-called coarse-lossless mode.

2.4 Satellite image denoising

2.4.1 Generalities on image denoising

Denoising is one of the oldest problems in image processing, for which numerous highly efficient algorithms have been proposed. The denoising algorithms can be categorized as: local if they rely on a limited context around the pixel to be denoised and non-local if they parse the whole image to denoise every single pixel [Masse et al. (2018)]. Among them, non-local filters are particularly efficient because they exploit the similarities in textures or structures that are often present in images but can be located in distant areas. The simplest approach is Non-Local Means [Buades et al. (2005a)], which simply performs non-local averages, that can be pixel- or patch-wise. A more elaborate extension named Non-Local Bayes [Lebrun et al. (2013)] improves the technique by performing Bayesian estimation through the estimation of covariance matrices of the patches. Another state-of-the-art technique is termed block-matching and three-dimensional filtering (BM3D) [Dabov et al. (2007)], which includes a step of collaborative filtering in a transformed domain.

Although the above image denoising methods have demonstrated excellent performance, they typically present two main disadvantages. First, these methods require multiple manually chosen parameters, which allows for possible improvements. Second, it is difficult to adapt these algorithms to non-standard noise models, such as the instrumental noise that affects satellite images during acquisition. Therefore, the satellite image denoising needs to suit particular noise characteristics.

2.4.2 Satellite imaging denoising system

In a typical satellite imaging system [Delvit et al. (2019)], the acquired noisy image is compressed onboard the satellite using a lossy compression algorithm such as the CCSDS 122.0-B-2 standard [B. Book (2017)]. The noisy compressed image is then transmitted to an Earth station, decompressed on the ground, and denoised. Note that the Pleiades satellite image denoising process is carried out on the ground segment due to its prohibitive complexity [Delvit et al. (2019)]. The next-generation satellites may address this complexity issue since they may dispose of sufficient onboard processing capacity to allow onboard denoising before compression.

Note that onboard compression introduces compression artifacts, which modify the original instrumental noise statistics, especially by the quantization. Denoising-after compression then requires different approaches such as dequantization strategies, i.e. [Foi et al. (2007) and Oberlin et al. (2019)]. After decompression, in the CNES satellite imaging system [Delvit et al. (2019)], the modified instrumental noise is restored by the instrument noise restitution method. The instrument noise restitution method aims to increase the denoising performance of the subsequent denoising algorithm that assumes Gaussian noise [Delvit et al. (2019)]. The variance stabilizing transform VST [Anscombe (1948)] is then applied to the image to transform the instrumental noise into an additive one. These operations increase the performance of the subsequent customized NL-Bayes denoising algorithm [Masse et al. (2018)], which strongly depends on the noise model. Finally, the inverse VST is applied [Anscombe (1948)].

2.4.3 Instrumental noise model

The acquired images are affected by an instrumental noise with a pixel-dependent variance defined, in the spatial domain, as [Masse et al. (2018)]:

$$\sigma^2{}_n(x, y) = A^2 + B \cdot I_{nf}(x, y) \tag{2.12}$$

where $I_{nf}(x, y)$ is the noise-free image pixel at coordinates (x, y) and (A, B) are known model parameters. A is the standard deviation of the additive part of the noise and B is the Poisson noise variance component. It is worth mentioning that A and B parameters are well known in satellite imagery as well as the MTF. First, the noisy acquired satellite image is compressed onboard the satellite using the compression algorithm CCSDS 122.0-B-2 [B. Book (2017)] or other satellite image compression algorithm such as [Thiebaut et al. (2016)].

2.4.4 Instrument noise restitution

The general idea of the instrument noise restitution technique is to compare each quantized transformed wavelet coefficient (can be at each level of wavelet decomposition or not) to the local expected instrumental noise level computed in the transformed domain [Delvit et al. (2019)]. If the coefficient is equivalent or greater than the local instrumental noise, it is kept unchanged. If it is lower, it is replaced by the local instrumental noise level. The pseudo-code of the instrumental noise restitution of transformed coefficients is the following:

Algorithm 1: Instrumental Noise Restitution

Input: w_{ij} wavelet coefficients at the decompression step of image with $N \times M$ pixels;
(A, B) noise model parameters;
k_1 and k_2: multiplication factors to adjust restitution;
rand: Gaussian white noise process with 0 mean value and unitary variance;
DC_{ij}: the DC coefficient corresponding to the coefficient w_{ij} (may be at each wavelet decomposition level);
Result: w_{ij} after instrumental noise restitution
for $(i, j) \in \{1, N\} \times \{1, M\}$ **do**
    $\sigma_{noise} = \sqrt{A + B \cdot DC_{ij}}$;
    if $|w_{ij}| < k_1 \cdot \sigma_{noise}$ **then**
        $w_{ij} = k_2 \cdot \sigma_{noise} \cdot rand$;
    else
        w_{ij} is kept unchanged;
    end
end

This instrumental noise restitution technique is widely applicable for transform-based compression algorithms like DCT and DWT [Delvit et al. (2019)]. It can be performed both during the decoding process or after the decoding process, the first being more efficient because it avoids additional direct and inverse transform stages.

2.4.5 Variance stabilizing transform

Next, to allow the use of a model-based denoising method that assumes an additive noise, a variance stabilizing transform (VST) [Makitalo and Foi (2012) and Anscombe (1948)] may also be applied to the noisy image. Anscombe transform [Makitalo and Foi (2012) and

Anscombe (1948)] is notably appropriate to the linear noise variance model expressed in Equation 2.12. It is defined as:

$$f(I_r(x,y)) = 2\sqrt{\frac{A^2}{B^2} + \frac{I_n(x,y)}{B} + \frac{3}{8}}, \tag{2.13}$$

as I_n. If we take the values of parameters A and B of typical Earth observation missions, the term 3/8 is largely negligible [Delvit et al. (2019)].

After applying the model-based denoising method, the VST transform must be reversed. The easiest way is applying $f^{-1}(y)$, but the corresponding estimator is biased. Thus, the asymptotically unbiased estimator is proposed in [Makitalo and Foi (2012)] and it is defined as:

$$f^{-1}(y) = \frac{1}{4}y^2 + \frac{1}{4}\sqrt{\frac{3}{2}}y^{-1} - \frac{11}{8}y^{-2} + \frac{5}{8}\sqrt{\frac{3}{2}}y^{-3} - \frac{1}{8}. \tag{2.14}$$

In the particular case of satellite images, the exact unbiased inverse of the generalized Anscombe transform becomes [Mäkitalo and Foi, 2012]:

$$f^{-1}(y) = B\left(\frac{y}{2}\right)^2 - \frac{A^2}{B}. \tag{2.15}$$

Once VST is applied on the image, a traditional model-based denoising method such as BM3D [Dabov et al. (2007)], NL-Bayes [Lebrun et al. (2013)], and NL-Means [Buades et al. (2005a)] can be properly used.

2.4.6 NL-Bayes algorithm

NL-Bayes algorithm is a denoising algorithm derived from the NL-Means method [Buades et al. (2005b)]. The NL-Means method relies on a weighted average of similar patches in a given neighborhood to denoise each patch. The NL-Bayes improves the denoising performance by considering a covariance matrix to estimate the variability in a group of similar patches. These similar patches are employed to build the 3D block of patches that is used in the Bayesian estimation [Masse et al. (2018)]. In the CNES imaging system [Delvit et al. (2019)], the NL-Bayes algorithm is chosen due to its simplicity concerning its adjustable parameters, high denoising performance, and its superiority in terms of computational efficiency when compared to other similar patch-based methods.

NL-Bayes is a two-step algorithm detailed in algorithms 2 and 3. Each step is composed

Table 2.1: List of parameters for NL-Bayes and optimal values selected by experts in the original implementation [Masse et al. (2018)].

Parameter		Optimal value		Description
Step 1	Step 2	Step 1	Step 2	
w_1	w_2	5	5	Patch size
k_1	k_2	27	25	Search area dimensions
N_1	N_2	74	30	Number of similar patches
β_1	β_2	1.0	1.6	Noise attenuation
	τ_2		2.5	Similarity threshold

of the same three parts (designated here as (a), (b), and (c)) [Masse et al. (2018)]. First in (a), the N most similar patches are localized and gathered in a 3D block. Consecutively in (b), the 1st- and 2nd-order statistics of the 3D block are computed, and Bayesian estimation is performed for each of its patches. Lastly, in (c), aggregation and weighting are performed, considering that the patches of the 3D blocks may overlap and lead to a variable number of denoising estimations per pixel. The second step takes benefit of the first step's result to improve its search of similar blocks. This procedure allows to obtain improved denoising since better mean vector, and covariance matrix estimations are performed with application of Bayes theory [Masse et al. (2018)].

A total of nine parameters are used for NL-Bayes parametrization: four for the first step, and five for the second. These parameters are used to adjust the patch size (w_1 and w_2), the search area dimensions (k_1 and k_2), the noise attenuation (β_1 and β_2), the number of similar patches selected for composing a 3D block (N_1 and N_2), and the minimum threshold to determine similar patches during the second stage (τ_2). These parameters are expressed in Table 2.1. [Masse et al. (2018)] introduced optimizations to the NL-Bayes algorithm to significantly reduce the required computation time. The computation time arises from both the number of overestimations, i.e., the number of operations necessary to estimate a denoised pixel, and the spatial extent of the search area while maintaining its high denoising performance. The authors thus identified two techniques: reducing the overestimations with a masking technique and modifying the search area window shape.

Algorithm 2: NL-Bayes algorithm: 1st step.

Input: $\mathbf{I}_n$ (noisy image);
Input: Set of parameters: w_1, k_1, N_1, β_1;
Result: $\mathbf{I}_b$ denoised intermediate image
for ***all*** pixel $i \in \mathbf{I}_n$ **do**
  $p \leftarrow$ patch of size ${w_1}^2$ centered on i;
  ${SA_p}^n \leftarrow$ search area of size ${k_1}^2$ centered on p;
  for ***all*** patch $q \in {SA_p}^n$ **do**
    Compute $s(p, q)$ (the similarity measure between patch p and q);
  end
  Find the N_1 most similar patches and gather them into a 3D block ${BP_n}^n$;
  Compute the statistics of ${BP_n}^n$: $\overline{\mu_p}$ and Σ_p;
  for ***all*** patch $q \in {BP_n}^n$ **do**
    $\hat{q} = \overline{\mu_p} + (\Sigma_p - \beta_1 \mathrm{I})\, {\Sigma_p}^{-1} (q - \overline{\mu_p})$ (Bayes estimation);
    Aggregate $\hat{q}$ in $\mathbf{I}_b$;
    Increment weighting buffer;
    Weight $\mathbf{I}_b$ with weighting buffer;
  end
  Weight $\mathbf{I}_b$ with weighting buffer;
end

Algorithm 3: NL-Bayes algorithm: 2nd step.

Input: $\mathbf{I}_n$ (noisy image);
Input: $\mathbf{I}_b$ (denoised intermediate image from the 1st step);
Input: Set of parameters: w_2, k_2, N_2, β_2, τ_2;
Result: $\hat{\mathbf{I}}_{nf}$ (denoised final image)
for ***all*** pixel $i \in \mathbf{I}_n$ **do**
  $p^n \leftarrow$ patch of size ${w_2}^2$ centered on i in $\mathbf{I}_n$;
  $p^b \leftarrow$ patch of size ${w_2}^2$ centered on i in $\mathbf{I}_b$;
  ${SA_p}^b \leftarrow$ search area of size ${k_1}^2$ centered on p;
  for ***all*** patch $q \in {SA_p}^b$ **do**
    Compute $s(p_b, q)$ (the similarity measure between patch p_b and q);
  end
  Find at most N_2 most similar patches satisfying $s(p_b, q) \leq \tau_2$and gather them into a 3D block ${BP_p}^b$;
  Compute the statistics of ${BP_p}^b$: $\overline{{\mu_p}^b}$ and ${\Sigma_p}^b$;
  for ***all*** patch $q \in {BP_p}^n$ **do**
    $\hat{q} = \overline{{\mu_p}^b} + {\Sigma_p}^b \left({\Sigma_p}^b - \beta_2 \mathrm{I}\right)^{-1} \left(q - \overline{{\mu_p}^b}\right)$ (Bayes estimation);
    Aggregate $\hat{q}$ in $\hat{\mathbf{I}}_{nf}$;
    Increment weighting buffer;
  end
  Weight $\hat{\mathbf{I}}_{nf}$ with weighting buffer;
end

2.5 Deconvolution

Once the noise has been sufficiently removed, deconvolution takes place to reverse the MTF effect. An important element in this part is the knowledge of the MTF. However, the estimation of the MTF is often not straightforward since the image acquisition system consists of many components, each with properties that may not be precisely characterized or exhibit changing characteristics. When the MTF is well-known, deconvolution consists in multiplying the image Fourier transform by a function close to MTF inverse. MTF is close to zero at Nyquist, and 1/MTF could take very large values, which would mean an oscillating deconvolution filter inducing ringing artefacts around transients [Delvit et al. (2019)]. Attention is necessary when performing deconvolution to denoised images. Such

a filter can also dramatically increase the remaining noise. Wiener-Tikhonov technique provides an adequate deconvolution filter [Tihonov (1963)], which can be defined as [Oh and Choi (2014)]:

$$W(v,w) = \frac{H^*(v,w)}{|H(v,w)|^2 + \frac{S_n(v,w)}{S_f(v,w)}}, \tag{2.16}$$

where $H^*(v,w)$ represents the conjugate complex of MTF and $S_n(v,w)/S_f(v,w)$ stands for the ratio of the power spectrum of the noise and the undistorted image. Generally, $S_n(v,w)/S_f(v,w)$ can be approximated by an inverse SNR [Oh and Choi (2014)].

2.6 Dataset of simulated images

The satellite images considered in this thesis are based on the Pléiades, which are high-resolution optical satellites developed by the French Space Agency (CNES) [Latry et al. (2012)]. Note that the image acquisition parameters are well known, e.g., the MTF response and noise model parameters (A, B). Consequently, precise models can be used to simulate realistic images. This becomes particularly interesting when the objective is to design and test the different processes carried out in a satellite image processing system separately.

The Pléiades satellites acquire images in both panchromatic and multispectral (Blue, Green, Red and Near Infra Red). We decided to consider only panchromatic images in the scope of this thesis since they effectively encapsulates the shape, structure, and texture of landscape objects in a single channel [Latry et al. (2012)]. The panchromatic acquisition is based upon Time Delay Integration (TDI) detectors with a maximum of 20 integration stages to satisfy the minimum SNR requirements. The multispectral mode depends on a classical charge-coupled device (CCD) detector [Latry et al. (2012)]. In the frame of this thesis, simulated images are used to benefit from an ideal reference and the intermediary images resulting from the different processes carried out. Those simulated Pléiades images are initially obtained from an airborne in a 10cm spatial resolution, but they are downsampled to 70cm.

Our image dataset is composed of 128 pairs of noise-free ($\mathbf{I_{nf}}$) and noisy ($\mathbf{I_n}$) 12-bit simulated Pléiades panchromatic images (of size 586×586) covering various landscapes, provided by the CNES. The instrumental noise is simulated according to [Delvit et al. (2019)]. For training and testing purposes, this dataset is splitted into a training dataset (116 images) and a test dataset (16 images). As an illustration, we can see images that compose the test dataset in figure 2.3. Note that in chapter 5, we also consider the same

Table 2.2: Noise model parameters (A, B).

A	B
1.423699	0.067198

landscapes but simulated for low average luminance to obtain a different scenario.



Figure 2.3: Simulated 12-bit Pléiades images.

Chapter 3

Image compression and denoising with neural networks

Contents

3.1 Generalities on neural networks . . . 26
3.2 Main neural architectures . . . 29
3.2.1 Feedforward neural networks (FNNs) x recurrent neural networks (RNNs) . . . 29
3.2.2 Convolutional neural networks (CNNs) . . . 30
3.2.3 Autoencoders (AE) . . . 33
3.2.4 Generative models . . . 34
3.3 Training neural networks . . . 35
3.3.1 Loss function and gradient descent . . . 35
3.3.2 Hyperparameters and model selection . . . 38
3.4 Compression with neural networks . . . 38
3.4.1 Generalities . . . 39
3.4.2 Analysis and synthesis transforms . . . 41
3.4.3 Loss function: rate distortion trade-off . . . 43
3.5 Denoising with neural networks . . . 49
3.5.1 General concepts . . . 49
3.5.2 Residual learning of deep CNN for image denoising (DnCNN) . . . 52

3.5.3 Toward a fast and flexible solution for CNN based Image denoising (FFDNet) 54
3.5.4 Image denoising using deep CNN with batch renormalization (BRDNet) 55
3.5.5 Neural networks for compression artifacts suppression 56
3.5.6 Denoising with GANs 58

Neural networks (DNNs) are particularly relevant for this thesis. DNNs were fast used for machine learning, then image processing: restoration, segmentation, denoising, and compression. Now, DNNs represent the state-of-the-art for many image processing tasks. Convolutional neural networks (CNNs) are particularly suited for image processing. This chapter introduces DNNs, then focuses on compression and denoising.

3.1 Generalities on neural networks

Neural networks (NNs) are highly inspired by neuroscience. Artificial neural networks represent an attempt of imitating the biological neural network of a brain to solve numerous problems. The research on NNs goes back to 1940s and an important milestone was the introduction of perceptrons by [Rosenblatt (1958)], which represent the basic unit of NNs. Individual neurons compose layers and sequences of layers form neural networks.

The perceptron was initially proposed to represent a mathematical model of the real neuron, which receives electrical impulses from its dendrites. When this specific neuron is polarized enough, it will propagate a signal to the other adjacent neurons. The perceptron takes an input, say $\mathbf{x}$. Each input (x_l) may not have the same significance to produce a desired output. Hence the concept of weights (w_l) is introduced, and they express the relevance of each input to output. Thus, the weighted contributions of each input is summed in a linear combination, and a bias term b is added as follows [Goodfellow et al. (2016)]:

$$y = \sigma\left(\sum_l x_l w_l + b\right). \tag{3.1}$$

The aggregated signal will be passed through a non-linear activation function ($\sigma(\cdot)$) to determine the stimulated neuronal response. The primary role of the activation function is to transform the weighted sum of the inputs into an output value to be fed to the next hidden layer or as output of the network. Thus, if the aggregated signal is strong enough according to a given threshold, then the neuron is activated, and a high value is passed forward to the

next layer or the network outputs. Otherwise, a low value or even no value is passed forward. When we consider only the linear combination expressed in Equation 3.1, its capacity to approximate complex models is rather limited. Thus, most real-world problems require non-linear operators because of their extended approximation capacities [Hinton et al. (2012)]. In practice, non-linearity is introduced by activation functions. Generally, each output of a layer has the same activation, but they may also consider different input formats, e.g., the generalized divisive normalization which will be introduced later. The choice of activation function may vary according to the application. We introduce three commonly adopted activation functions, namely sigmoid, tanh, and ReLU, which are illustrated in figure 3.1.

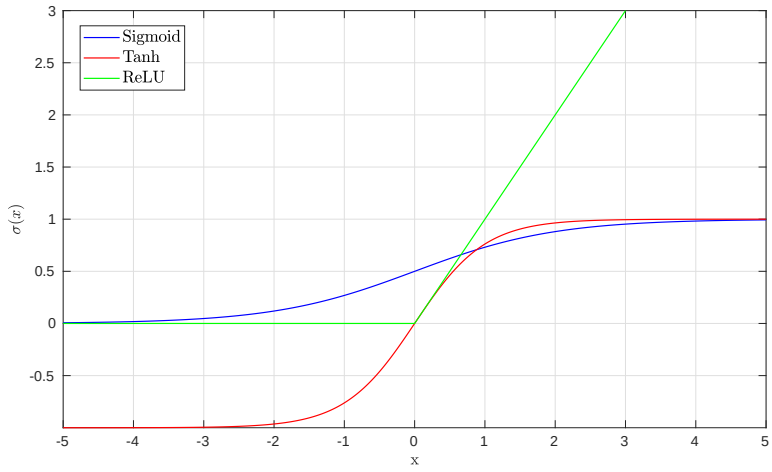


Figure 3.1: Three element-wise activation functions: sigmoid, tanh, and ReLU.

The sigmoid function has float output value that ranges between 0 and 1, according to:

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{3.2}$$

The sigmoid function can be employed at the output of a neural network for classification purposes, e.g., constraining the output value between 0 and 1 and then converting it to a discrete class label (0 or 1) by using a threshold such as 0.5.

Analogously to the sigmoid function, the hyperbolic tangent (tanh) constrains its output between -1 and 1, as defined in:

$$\sigma(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{3.3}$$

The rectified linear unit (ReLU) is defined as:

$$\sigma(x) = \begin{cases} 0 & \text{when } x \leq 0, \\ x & \text{when } x > 0. \end{cases} \tag{3.4}$$

[Glorot et al. (2011)] show that ReLU is more promising and presents more practical advantages for classification when compared to sigmoid and tanh. It is easier to implement and compute since only a single comparison with 0 is required. While in the sigmoid and tanh, the exponential function is harder to be computed. The ReLU also makes optimization easier since it is close to being linear, with two linear functions, making the gradient large and consistent. The SoftPlus is a smooth approximation of ReLU function and it is defined as:

$$\sigma(x) = log(1 + e^x). \tag{3.5}$$

The NN can dispose of multiple neurons and layers. The layers are designated as dense or fully connected layers when each neuron output is connected with all inputs. A multilayer perceptron (MLP) stands for a NN composed of at least two dense layers. Figure 3.2 displays an MLP consisting of four dense layers. The neurons' outputs are represented by y_i^l, where l stands for the layer index, and i is the output index. The three layers between the input and output layers are hidden layers since they are not accessible from outside the network. The dimensionality of these hidden layers defines the network's width. The number of layers defines the network depth. Note that the name "deep learning" arise from this terminology. When compared to the network with a single dense layer, MLP presents a stronger learning capability [LeCun et al. (2015)]. In practice, MLPs with several hidden layers are more flexible and easier to train than a single dense layer [Goodfellow et al. (2016)].

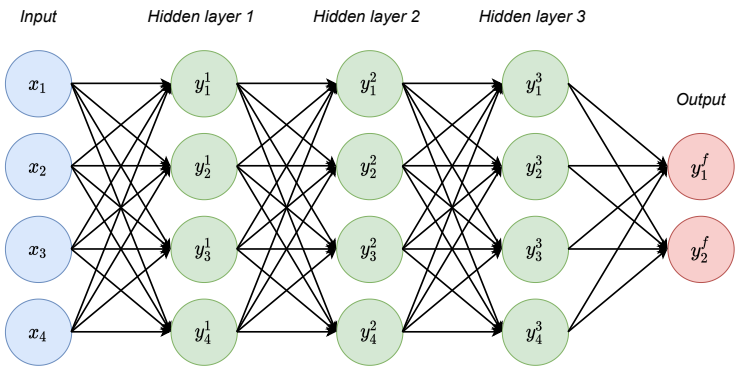


Figure 3.2: Example of an MLP with three hidden layers and one output layer.

3.2 Main neural architectures

3.2.1 Feedforward neural networks (FNNs) x recurrent neural networks (RNNs)

In the so-called feedforward neural networks (FNNs), information propagates in only one direction from the input nodes, through the hidden nodes (if any) towards the output nodes [Goodfellow et al. (2016)]. There are no feedback connections in which outputs of the model are provided back into itself. The network is thus associated with a directed acyclic graph.

Recurrent neural networks (RNNs) [Rumelhart et al. (1988)] are a family of neural networks designed for processing sequential data. Note that when dealing with sequential data, there is a temporal interaction among elements within the sequence. For example, a human reader can easily infer the next content of a small text fragment by only reading its beginning. RNNs should be able to effectively assimilate information contained in the sequential data and take into account the influence of old samples on the most recent ones. To attain this, they dispose of computational nodes featuring cycles that represent the influence of the past value of a variable on its present value. Therefore, each output member is produced as a function of the previous output members. Note that unfolding this graph results in sharing parameters across the network structure. When unfolding, the computational graph is not acyclic anymore. For example, we can consider the classical form of a dynamical system as follows [Goodfellow et al. (2016)]:

$$s^{(t)} = f(s^{(t-1)}), \tag{3.6}$$

where $s^{(t)}$ denotes the state of the system. Equation (3.6) expresses a recurrent mapping since the definition of s at time t refers back to the same definition at time $t-1$. For a finite number of time observations (τ_t), the graph can be unfolded by applying the definition $\tau_t - 1$ times. For example, the unfolded computation graph for $\tau_t = 3$ is illustrated in figure 3.3. RNNs are particularly well-suited to applications involving time-series data such as: handwriting recognition, speech recognition, image captioning, and language translation [Goodfellow et al. (2016)].

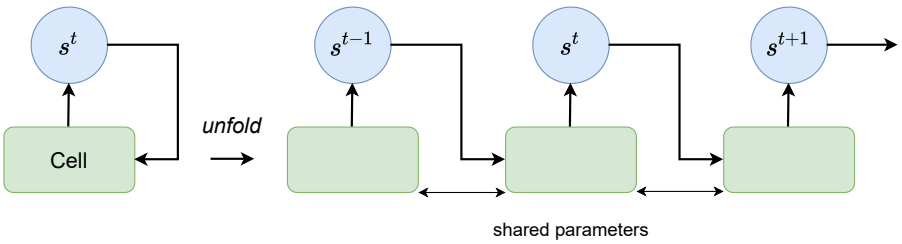


Figure 3.3: Example of an unfolded computation graph.

3.2.2 Convolutional neural networks (CNNs)

Convolutional neural networks (CNNs) are a class of ANNs most commonly applied to analyze visual imagery [Bengio (2009)]. They emerged as a need for processing grid-like topology and local features, such as those present in images, for instance. As a natural solution, they employ a mathematical operation called convolution in place of a general linear operation. The CNNs are known for being shift-invariant, based on the shared-weight architecture of the convolution kernels or filters. In CNNs, weight-sharing occurs since each filter is replicated across the entire visual field. These replicated units share the same parameterization (weight vector and bias), and they together operate to form a feature map. In other words, the convolutional filters slide along input features and provide translation-equivariant feature maps. Consequently, CNNs support the full-resolution processing of images, while conventional neural networks usually process partitioned blocks. Notably, CNNs have been successful in many computer vision applications [Bengio (2009)] such as classification [J. Hu et al. (2018)], object detection [Redmon et al. (2016)] and segmentation [Kaiser et al. (2017)].

Specifically, note that the basic unit composing the CNNs is the convolutional layer. The grid-like topology input data is arranged in 3 dimensions, and each channel consists of a single 2-dimensional slice of the depth dimension. $u_i(k,l)$ represents the ith input channel at spatial location (k,l). Each stage then begins with an affine convolution:

$$v_i(k,l) = \sum_j \left(h_{i,j} * u_j\right)(k,l) + c_i, \tag{3.7}$$

where $*$ represents 2D convolution, h_i denotes what is called the ith filter, j represent the input depth location, and c_i represents the bias term. Figure 3.4 presents an illustration of the 2D convolution operation. In this example, a single filter with kernel size $3 \times 3 \times 3$ is applied on an image $5 \times 5 \times 3$. The convolution is computed across the different channels.

The values obtained from each convolution stage are then summed up to generate the top-left value of the output. Note that the bias term is not present in this illustration. The stride (S) defines the distance between spatial locations where the convolution kernel is applied (here $S = 1$). Normally a larger stride is chosen ($S > 1$), resulting in fewer calculations and less memory usage.

Note that to ensure that boundary values are properly considered, the edges are usually padded with zeros (zero-padding), by reflecting the edges of the image (mirror-padding) or not filled, e.g., 'valid' as in the figure 3.4. The padding size is denoted as P. The output dimension (N_{out}) can be computed by:

$$N_{out} = \left\lfloor \frac{N_{in} - \kappa + 2P}{S} + 1 \right\rfloor, \tag{3.8}$$

where N_{in} represents the 2D input dimension (N_x or N_y), κ stands for the kernel size of the filter and $\lfloor \cdot \rfloor$ denotes the floor operation. The output's depth (number of channels) is defined by the number of filters. In figure 3.4, according to Equation (3.8), the output height/width is $(5 - 3 + 1) = 3$ and the depth is equal to 1 since there is one filter.

This operation is usually followed by downsampling:

$$w_i(k, l) = v_i\left(s \times k, s \times l\right), \tag{3.9}$$

where s represents the downsampling factor. Downsampling is motivated for reducing the spatial resolution of the feature maps, eliminate redundancy, and encouraging the network to extract the most significant features. A pooling layer performs downsampling. Pooling assumes that, for images, neighboring pixels are similar. The average pooling layer creates a downsampled (pooled) feature by dividing the input into rectangular regions and computing the average values of each of them. This operation adds a small amount of translation invariance and extracts smooth features. Whereas max-pooling mainly extracts features like edges by selecting the maximum pixel value of each region, which improves shift-invariance and makes it suitable for classification. Average pooling encourages the network to identify the complete extent of a specific region, while max pooling restricts that only to the most prominent features and may discard some details. Downsampling can also be performed during the convolutional operation when selecting S adequately but note that it can be combined with average or max pooling. Then each stage can conclude with an activation function.

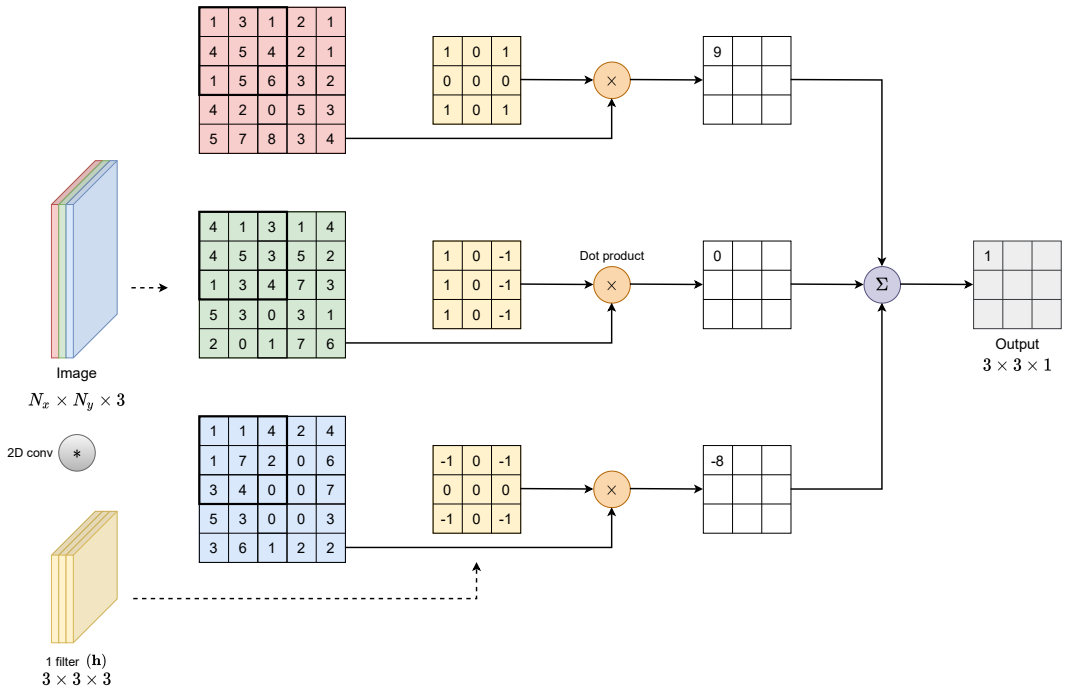


Figure 3.4: Illustration of the convolutional operation.

When CNNs aim to generate images, they are often built up from downsampled feature maps, low resolution, and high-level descriptions. In order to do this, we need some way to go from a lower resolution image to a higher one. We generally do this with the transposed convolution operation. In this case, this operation starts with upsampling:

$$\hat{v}_i(k,l) = \begin{cases} \hat{w}_i\left(k/\hat{s}, l/\hat{s}\right) & \text{if } k/\hat{s} \text{ and } l/\hat{s} \text{ are integers,} \\ 0 & \text{otherwise,} \end{cases} \tag{3.10}$$

where $\hat{s}$ is the upsampling factor. Then, this operation is followed by a convolution:

$$\hat{u}_i(k,l) = \sum_j \left(\hat{h}_{i,j} * \hat{v}_j\right)(k,l) + \hat{c}_i. \tag{3.11}$$

Note that the transposed convolutional layer does exactly what a standard convolutional layer does but on a upsampled input feature map ($\hat{v}_i(k,l)$). Analogously to the convolutional case, each stage can conclude with a non-linear activation function which is task dependent.

3.2.3 Autoencoders (AE)

Autoencoders (AEs) are motivated from the need to learn descriptive representation in lower-dimension for input data. Along with the reduction side, reconstruction is also desired. AEs don‘t aim not only learn latent representations of seen input features, but to generalize in a way that allows for an interpretation of unseen data and data features with slight variations [Bårli et al. (2021)]. AEs have been initially designed for data dimension reduction similar to, e.g., Principal Component Analysis [Bengio (2009)]. An encoder (E) is applied to the input data $\mathbf{x}$ to produce a learned representation $\mathbf{y} = E(\mathbf{x})$ at the bottleneck. Then, a decoder (D) is applied to reconstruct the input data from $\mathbf{y}$: $\hat{\mathbf{x}} = D(\mathbf{y})$. The architecture of a fully-connected autoencoder is illustrated in figure 3.5. Note that convolutional AEs represent a motivation for transposed convolutions. Autoencoders may also be employed to output reconstructed images from low-resolution or distorted images. In this case, they are trained with a pair degraded input/clean reference.

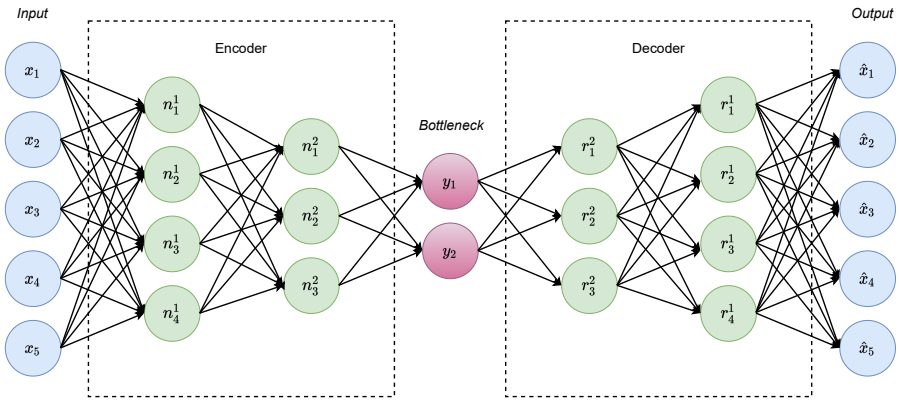


Figure 3.5: Illustration of a fully-connected autoencoder.

Sometimes, the input images may be noisy. In this case, when using noise-free images as reference, the autoencoder can be trained to denoise them, produce the hidden code representation, and then reconstruct the noise-free images [Vincent et al. (2010)]. Thus, it is designated as denoising autoencoder (DA). It can be seen as an extension of the basic autoencoders architecture suitable for image reconstruction. Note that DAs were not originally designed to automatically denoise an image. Their training is intended to stimulate their hidden layers to learn more robust filters. Thus, one strategy consists of adding additive random noise to the autoencoder inputs and then encouraging it to recover the original noise-free image. It reduces the risk of overfitting in the autoencoder [Vincent et

al. (2008)] and prevents it from learning a simple identity function [Vincent et al. (2010)]. Recent theoretical links between autoencoders and latent variable models have also placed autoencoders to the lead of generative modeling [Bengio (2009)].

3.2.4 Generative models

The variational autoencoder (VAE) consists of a particular generative model for unsupervised representation learning [Kingma and Welling (2013)]. A VAE provides a probabilistic manner for describing an observation in the latent space. Normally, a VAE consists of a standard autoencoder component which encodes the input data ($\mathbf{x}$) into a latent representation ($\mathbf{y}$) by minimizing the reconstruction error and forcing the posterior of the latent representation to match a prior distribution [Kingma and Welling (2013)]. For VAEs, the encoder is sometimes referred to as the inference network whereas the decoder is sometimes referred to as the generative network. In this case, the encoder also returns a distribution over the latent space. Note that a regularisation term is also added over that returned distribution in the loss function.

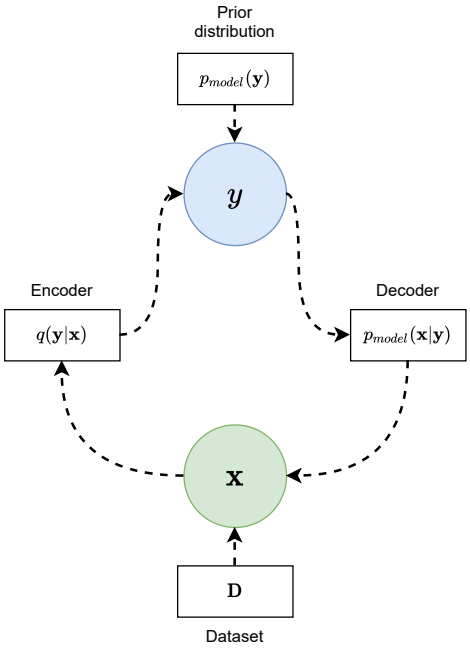


Figure 3.6: Illustration of the stochastic mappings learned by the VAE [Kingma and Welling (2019)].

3.3 Training neural networks

To train the weights or parameters of a NN, a loss function is usually minimized via stochastic gradient descent (SGD). When considering supervised training, we dispose of a training set of input data $\{\mathbf{x_i}\}$, where $i = 1, ..., N$ designate individual samples with a corresponding set of output data $\{\mathbf{y_i}\}$. The goal is to learn the parameter vector θ so that the network mapping function $f_\theta(\mathbf{x}) \backsimeq \mathbf{y}$. θ gathers the learnable parameters such as the weights or the filter coefficients in the case of convolutional networks and possibly parameters of the activation functions. Therefore, the training is cast as an optimization problem.

3.3.1 Loss function and gradient descent

The loss function $J(\mathbf{x}, \theta)$ measures the error, and the optimization algorithm minimizes it between the training labels and the neural network mapping with respect to θ. A simple and standard loss function is the mean square error (MSE), which calculates the mean squared difference between $f_\theta(\mathbf{x})$ and $\mathbf{y}$. The loss function generally follows:

$$J(\mathbf{x}, \theta) = \frac{1}{N} \sum_{i=1}^{N} D(f_\theta(\mathbf{x_i}), \mathbf{y_i}). \tag{3.12}$$

where $D(.)$ designates a divergence measure. For the minimization of the loss function, a gradient descent method is adopted. This requires the computation of the loss function gradient with respect to θ. The gradient of the loss function with respect to θ is computed with the backpropagation algorithm [Bengio (2009)]. The bakcpropagation algorithm computes the gradient of the loss function for a single weight by the chain rule. It efficiently computes one layer at a time, unlike a native direct computation. The back propagation algorithm is later presented in this section. method iteratively updates the set of learnable parameters with the aid of the gradient. Note that in practice, the gradient cannot be computed. The gradient is thus estimated on mini-batches. In a simple definition, the gradient descendent method can be expressed as:

$$\theta \leftarrow \theta - \rho \cdot \frac{\partial J(\mathbf{x}, \theta)}{\partial \theta}, \tag{3.13}$$

where ρ represents the learning rate [Bengio (2009)]. The gradient descent method does not necessarily reach the global minimum. Otherwise, it can reach a local minimum or it can get stucked in a saddle point in a worse case scenario. Moreover, a different initialization of the set of learnable parameters can result in a different solution. Consequently, the

initialization of θ is particularly important. [Glorot and Bengio (2010)] showed that a simple random initialization of the parameters is not suited for deep neural networks. Instead, they introduce a new heuristic for initializing θ. This heuristic takes into account the size of the previous layer (i.e., number of neurons arranged in the convolutional layers) n_{i-1} and the size n_i of the current layer i. The weights of layer i are thus initialized as follows [Glorot and Bengio (2010)]:

$$\theta_{\mathbf{i}} \sim \mathcal{U}[-\frac{\sqrt{6}}{\sqrt{n_{i-1}+n_i}}, \frac{\sqrt{6}}{\sqrt{n_{i-1}+n_i}}], \tag{3.14}$$

where $\mathcal{U}[-a, a]$ represents a uniform distribution between $(-a, a)$ and $\theta_{\mathbf{i}} = w^{(i)}$ represents the learnable parameters at layer i. The motivation of the weight initialisation is to avoid layer outputs from exploding or vanishing during the forward pass through a NN.

Every weight in θ is defined as $w_{j,k}^{(i)}$ where j describes the jth element composing the current layer (i) and k defines the connection unit in the previous layer. Figure 3.7 illustrates the unit j at the layer i, which corresponds to the $y_j^{(i)}$ and its respective K input connections showed in figure 3.2.

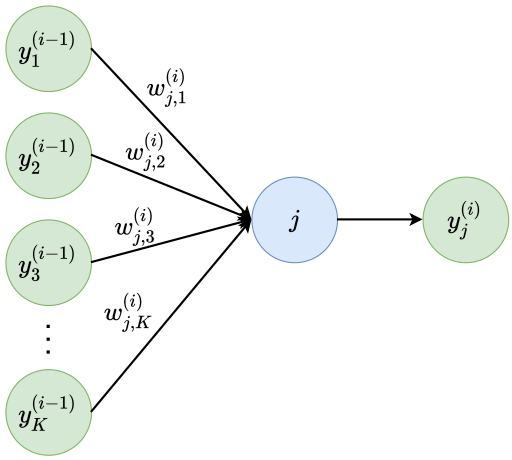


Figure 3.7: Example of one unit $y_j^{(i)}$ corresponding to the layer i.

Next, we introduce the backpropagation algorithm [Rumelhart et al. (1986) and LeCun et al. (2015)] which consists of an efficient method to compute the gradient with respect to the weights. To compute the gradient for each weight $(w_{j,k}^{(i)})$, the backpropagation algorithm disposes of the following recursive equation:

$$\frac{\partial J(\theta)}{\partial w_{j,k}^{(i)}} = \delta_j^{(i)} \cdot y_j^{(i)}, \tag{3.15}$$

where $\delta_j^{(i)}$ corresponds the partial derivative, $y_j^{(i)}$ is a single unit with respect to the layer i. σ_i designates the activation function of layer i and $z_j^{(i)} = \sum_{l=1}^{K} w_{j,l}^{(i)} \cdot y_l^{(i-1)}$ denotes the activation value from unit j in layer i. Thus, $y_j^{(i)} = \sigma_i(z_j^{(i)})$. To compute the partial derivative of the loss with respect to the layer output $\delta_j^{(i)} \cdot y_j^{(i)}$, we apply the chain rule as follows:

$$\delta_j^{(i)} = \begin{cases} \frac{\partial J(\theta)}{\partial y_j^{(i)}} \cdot \sigma_i'(z_j^{(i)}) & \text{if } i \text{ is the network output.} \\ \sum_{l=1}^{Q} \left(\delta_j^{(i+1)} \cdot w_{l,j}^{(i+1)}\right) \cdot \sigma_i'(z_l^{(i)}) & \text{if } i \text{ is a hidden layer.} \end{cases} \tag{3.16}$$

where Q defines the number of units in the layer $i+1$. Before initialize the backpropagation process, the forward propagation must be computed to produce the activation values $z_j^{(i)}$ and consecutively the output of $y_j^{(i)}$. These values are required for the backpropagation algorithm. Once all gradients are computed, the gradient descendent step can be applied, with the weights being individually updated as follows:

$$w_{j,k}^{(i)} \leftarrow w_{j,k}^{(i)} - \rho \cdot \frac{\partial J(\mathbf{x}, \theta)}{\partial w_{j,k}^{(i)}}. \tag{3.17}$$

Thus, the next iteration can be started. This process is repeated until the completion of a predefined maximal number of iterations, or when the loss value ($J(\mathbf{x}, \theta)$) reaches a value lower than a predefined threshold.

In DNNs, the gradients are approximated using a small subset of the dataset called mini-batch instead of using the whole dataset per gradient computation. Optimization algorithms form the basis on which DNNs can learn through observing examples. There are several ways learning is implemented with different kinds of optimization algorithms, such as Nesterov Momentum, Adagrad, RMSProp, and Adam [Zaheer and Shaziya (2019)]. Adam is the most popular algorithm in deep learning due to its high efficiency in optimizing DNNs. Adam is an algorithm for first-order gradient-based optimization of stochastic objective functions based on adaptive estimates of lower-order moments. This method is well adopted since it is straightforward to implement, is computationally efficient, has few memory requirements, and is well suited for problems that are large in terms of data and/or parameters. We adopted the Adam algorithm in the experiments conducted in this thesis, and we won't discuss the different optimization algorithms since it is not the focus of this thesis.

3.3.2 Hyperparameters and model selection

Hyperparameters are the variables that determine the NN structure (e.g., number of hidden units, number of layers, activation functions, etc.) and the variables which determine how the network is trained (e.g., training algorithm, learning rate, etc.). Hyperparameters are usually set before training. The NN performance can be significantly changed when adjusting them, and more precisely, the model capacity. A higher capacity model is expected to model more complex relationships between more variables than a model with a lower capacity. When considering neural networks, these parameters can be the learning rate (ρ) and the chosen architecture (e.g., the number of layers and the number of filters on each layer). Usually, these parameters are once fixed and not learned during the training. When considering a particular problem, the methodology adopted is to test different configurations of hyperparameters and prefer the one which presents the lowest generalization error. Note that this adjustment usually demands extensive empirical evaluations.

As we will also find later in this work, the neural network method architecture often depends on the desired application. Consequently, it is hard to find theoretical explanations for the proposed methods.

3.4 Compression with neural networks

Artificial NNs appeared as powerful data-driven tools to solve problems previously addressed with model-based methods in recent years. In particular, image processing has been widely impacted by convolutional neural networks (CNNs) such as segmentation, classification, and denoising. With the fast development of artificial NNs, end-to-end CNNs have been successfully employed for lossy image compression [Ballé et al. (2017), Theis et al. (2017), Rippel and Bourdev (2017), and Ballé et al. (2018)]. These proposals explore the great potential of NN to form learned image compression frameworks. However, the development of these learned methods presents significant differences compared to traditional compression methods. In the case of model-based compression methods, performance improvements are obtained by refinement of each element of the compression framework, e.g., in the Wavelet decomposition scheme and quantization strategy presented in 2.3.1. However, considerable human efforts are usually necessary. In some cases, even if the performance of an individual module is improved, the performance of the combined modules may not result in significant improvements. Thus, further improving the model-based methods is a difficult task [Y. Hu et al. (2021)].

When considering end-to-end learned compression, all the modules are trained together to reinforce the final objective. Two aspects are usually considered when designing such methods. First, more bit-rate can be saved in the entropy coder if the learned transform leads to less redundant coefficients [Y. Hu et al. (2021)]. Second, if the probability distribution of the learned coefficients can be precisely estimated, they can be more efficiently encoded by the entropy coder. Thus, such architectures jointly learn a non-linear transform and its underlying statistical distribution to optimize a rate-distortion trade-off. They are able to dramatically outperform traditional compression schemes regarding this trade-off. However, the performance improvements usually come at the cost of high computational complexity.

This section proposes an overview of the end-to-end learned compression composing the state-of-the-art. The different components and design options are presented, along with the challenges and limitations they encounter.

3.4.1 Generalities

Many different NN architectures have been proposed to image compression. They can be divided into two main categories: feedforward frameworks and multistage recurrent frameworks, described in the following paragraphs.

Feedforward frameworks In the literature, higher rate-distortion performance is attributed to the feedforward frameworks [Y. Hu et al. (2021)]. A typical neural network image compression framework is built upon a convolutional autoencoder [Ballé et al. (2017), Theis et al. (2017), and Ballé et al. (2018)]. The autoencoder transforms an image $\mathbf{x}$ into a learned representation $\mathbf{y}$. With the dimensional reduction and entropy constraints, the learned representation consists of less-redundant data. An entropy coder is used to generate a bitstream from the learned representation with the aid of an entropy model.

Feedforward frameworks have mainly been adopted for end-to-end learned image compression. [Ballé et al. (2017)] proposed the initial one. Note that a significant improvement in learned compression was the proposal of the generalized divisive normalization (GDN) (resp. inverse generalized divisive normalization (IGDN)) activation functions in [Ballé et al. (2016)], which have proven efficient for image compression.

Multistage recurrent frameworks In the multistage recurrent frameworks, the architecture design takes into account the fact that the image encoding/decoding will be pro-

gressive [Toderici et al. (2016) and Toderici et al. (2017)]. They perform successive passes to encode an original image and the resulting residual representations.

The compression networks [Toderici et al. (2016) and Toderici et al. (2017)] are composed of an encoding network G_a, a binarizer B and a decoding network G_s, where G_s and G_a contain recurrent network components, such as convolutional LSTMs [Xingjian et al. (2015)]. The decoder network creates an estimate of the original input image based on the received binary code. This procedure is repeated with the residual error. The network weights are shared between iterations, and the states in the recurrent components are propagated to the next pass. In each pass, B produces a bitstream $\mathbf{b}^t \in \{-1,1\}^m$, where m represents the number of bits produced after every pass, following [Toderici et al. (2016)]. The network generates a total of $m \cdot k$ bits after k passes, where m is a linear function of input size. For example, when considering image patches of 32×32, $m = 128$. The recurrent units used to create the encoder and decoder include two convolutional kernels: the "hidden kernel" and the "hidden convolution". The larger hidden kernels consistently resulted in improved compression curves [Toderici et al. (2017)]. In each stage, the composing recurrent layers take the current produced residual and the state from the previous stage as input, as previously explained in 3.2.1. Naturally, the variable rate is achieved when controlling the number of passes. More passes result in higher quality, however, at the cost of higher complexity.

Note that the recurrent-based frameworks require backpropagation through time (BPTT), rendering training more difficult besides the more complicated residual formulation that results in more complex frameworks [Y. Hu et al. (2021)]. In addition, the recurrent-based frameworks usually demand more time to encode and decode an image since the network runs multiple times. Regarding these considerations, this work focus on feedforward frameworks and their characteristics.

3.4.2 Analysis and synthesis transforms

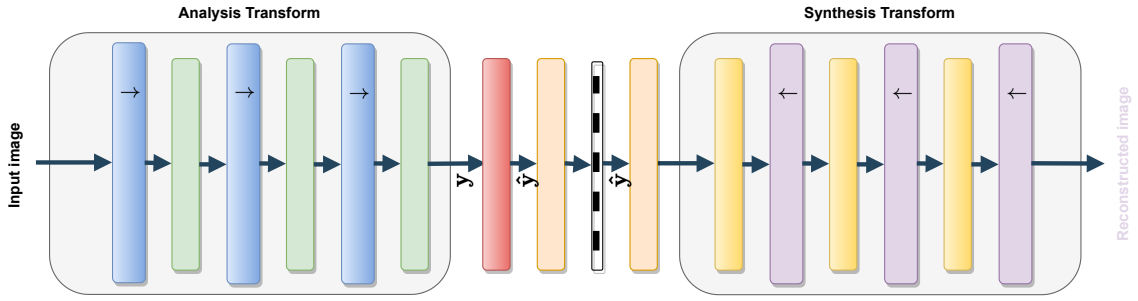


Figure 3.8: Architecture of the autoencoder [Ballé et al. (2017)].

In this thesis, we consider the reference architecture proposed in [Ballé et al. (2017)], displayed in figure 3.8. This architecture performs analysis and synthesis transforms using multiple convolutional layers that can include the generalized divisive normalization (GDN) (resp. inverse generalized divisive normalization (IGDN)) activation functions. In the analysis transform, the encoder reduces the spatial dimensions with the aid of stride convolutions [Ballé et al. (2017)]. Respectively, in the synthesis transform, the decoder increases the spatial dimensions using transposed convolutions. Based on [Ballé et al. (2017)], the authors proposed improvements in the synthesis and analysis transforms [Ballé et al. (2018)]. In [Theis et al. (2017) and Mentzer et al. (2018)], the authors adopted multiple residual blocks in both encoder and decoder, which allowed to increase the depth of the network. Moreover, some works adopt multiscale structure [Rippel and Bourdev (2017) and Cai et al. (2018)].

A deeper network is frequently associated with performance improvements in computer vision tasks, e.g., image recognition [He et al. (2016) and Szegedy et al. (2015)]. However, extending the architecture complexity does not result in significant performance improvements for learned image compression [Y. Hu et al. (2021)]. Although deeper architectures are more effective for image modeling, they are harder to train than their shallower counterparts devoted to compression [Y. Hu et al. (2021)].

3.4.2.1 Generalized divisive normalizations (GDNs)

In the context of transform coding, most compression methods are based on orthogonal linear transforms, chosen to reduce correlations in the data and thus to ease entropy coding. On the contrary, the joint statistics of linear filter outputs commonly exhibit strong

dependencies due to the convolutional operation. According to [Sinz and Bethge (2013), Lyu (2010), and Schwartz and Simoncelli (2001)], these redundancies may be significantly reduced through the use of joint local nonlinear gain control, inspired by models of visual neurons [Heeger (1992) and Carandini and Heeger (2012)]. Cascaded variants of beforementioned joint local nonlinear gain control modules have been used to obtain multiple stages of visual transformation, resembling a decomposition process [Simoncelli and Heeger (1998) and Mante et al. (2008)]. Some earlier results suggested that incorporating local normalization in linear block transform coding techniques can improve coding compression performance [(Malo et al., 2005; Malo and Laparra, 2010)]. However, the normalization parameters were not optimized for compression.

A GDN (resp. IGDN) transform with optimized parameters was first proposed in [Ballé et al. (2017)]. Contrarily to usual pointwise parameter-free activation functions (e.g., ReLU, sigmoid), GDN and IGDN are parametric functions that implement an adaptive normalization. In a given layer, the normalization operates through the different channels independently on each spatial location of the filter outputs. The GDN (resp. IGDN) has demonstrated an impressive capacity for modeling neural responses suitably to learned image compression. Since then, multiple end-to-end image frameworks have been proposed using GDN/IGDN [Ballé et al. (2017), Theis et al. (2017), and Ballé et al. (2018)]. If $v_i(k,l)$ denotes the spatial location indexed by (k,l) of the output of the ith filter, the GDN output is derived as follows:

$$GDN(v_i(k,l)) = \frac{v_i(k,l)}{(\beta_i + \sum_{j=1}^{N} \gamma_{ij} v_j^2(k,l))^{1/2}} \text{ for } i = 1, ..., N, \tag{3.18}$$

where N defines the number of channels. The IGDN is an approximate inverse of the GDN and is used in the decoder part, derived as follows:

$$IGDN(\hat{v}_i(k,l)) = \hat{v}_i(k,l) \left(\beta'_i + \sum_{j=1}^{N} \gamma'_{ij} \hat{v}_j^2(k,l) \right)^{1/2} \text{ for } i = 1, ..., N. \tag{3.19}$$

According to Equation (3.18) (resp. Equation (3.19)), the GDN (resp. IGDN) for channel i is defined by $N+1$ parameters denoted by β_i and γ_{ij} for $j = 1, ..., N$ (resp. β'_i and γ'_{ij} for $j = 1, ..., N$). Finally $N(N+1)$ parameters are required to define the GDN/IGDN in each layer. Note that the learning and the storage of these parameters are required. However, GDN has been shown to reduce statistical dependencies [Ballé (2018) and Lyu (2010)] and thus it appears particularly appropriate for transform coding. According to [Ballé (2018)],

the GDN better estimates the optimal transform than conventional activation functions for a wide range of rate-distortion trade-offs. GDN and IGDN, while intrinsically more complex than usual activation functions, are prone to boost the compression performance especially in case of a low number of layers, thus affording a lower global complexity for the network.

3.4.3 Loss function: rate distortion trade-off

The compression framework usually consists of an encoder and decoder pair (resp. analysis transform and synthesis transform). Given an input image $\mathbf{x}$ with distribution $p_{\mathbf{x}}(\mathbf{x})$, the encoder with an analysis transform (G_a), and a quantization function (Q), a discrete code $\hat{\mathbf{y}}$ is generated as:

$$\hat{\mathbf{y}} = Q\left(G_a(\mathbf{x})\right). \tag{3.20}$$

To obtain the reconstructed image, the corresponding decoder G_s reconstructs the image $\hat{\mathbf{x}}$ from the discrete code $\hat{\mathbf{y}}$ according to:

$$\hat{\mathbf{x}} = G_s(\hat{\mathbf{y}}). \tag{3.21}$$

Two different metrics, i.e., distortion D_r and bit-rate R_t, arise to the rate-distortion optimization $\lambda D_r + R_t$, which represents the main problem in lossy image compression. The distortion metric D_r measures how dissimilar the reconstructed image is from the original image, according to:

$$D_r = \mathbb{E}_{\mathbf{x} \backsim p_{\mathbf{x}}(\mathbf{x})}\left[D(\mathbf{x}, \hat{\mathbf{x}})\right], \tag{3.22}$$

where $\backsim$ means distributed according to, and $D(.)$ represents the distortion function. In practice, we consider the empirical mean squared error (MSE) as follows:

$$D_r = \frac{1}{N}\sum_{i=1}^{N} ||\mathbf{x} - \hat{\mathbf{x}}||_2^2. \tag{3.23}$$

R_t corresponds to the number of bits used to encode $\hat{\mathbf{y}}$, which is bounded according to the entropy of $\hat{\mathbf{y}}$ which can be estimated by computing the continuous differential entropy model of $\tilde{\mathbf{y}}$ ($p_{\tilde{\mathbf{y}}}(\tilde{\mathbf{y}})$), as it will be further explained in 3.4.3.1 and 3.4.3.3. The relaxed formulation of the differential entropy allows to optimize entropy within differential-based learning frameworks. The overall compression problem can be viewed as an optimization of the weighted sum of D_r and R_t. This problem can be addressed by minimizing the following

optimization as follows:

$$\hat{\theta}_{G_a}, \hat{\theta}_{G_s}, \hat{\theta}_{p_y} = \underset{\theta_{G_a}, \theta_{G_s}, \theta_{p_y}}{\arg\min} \ \lambda D(\mathbf{x}, \hat{\mathbf{x}}) + R(\hat{\mathbf{y}}), \tag{3.24}$$

where $\theta_{G_a}, \theta_{G_s}, \theta_{p_y}$ denotes the set of parameters for the analysis transform, synthesis transform and entropy model, respectively. The sets of parameters $\hat{\theta}_{G_a}$ and $\hat{\theta}_{G_s}$ consist of the weights of the convolutional filters and the own parameters of the GDNs and IGDNs. The entropy model, as it will be detailed in in 3.4.3.4, has its own set of parameters defined by θ_{p_y}. Transforms learned for different λ values are necessary to cover a wide rate-distortion range [Ballé et al. (2017) and Ballé et al. (2018)]. This may limit the adoption of such learned frameworks for variable-rate applications. Note that, usually, the rate-distortion performances of image compression are tuned by varying the quantization step size. In [Dumas et al. (2018)], the authors showed that comparable performances can be obtained with a unique learned transform. The different rate-distortion points are then reached by varying the quantization step size at test time.

In a multistage recurrent frameworks, during training, a L_1 loss is calculated on the weighted residuals generated at each iteration as follows:

$$\hat{\theta}_{G_a}, \hat{\theta}_{G_s} = \underset{\phi_{G_a}, \phi_{G_s}}{\arg\min} \ \beta \sum_t \left| \hat{\mathbf{r}}^{(t)} \right|, \tag{3.25}$$

where β stands for the weighting parameter.

3.4.3.1 Differentiable quantization

In the specific context of learned compression, a significant obstacle in training such autoencoder-based frameworks is that the derivative of the quantization function is zero everywhere except at integers, where it is undefined. Thus, its use is hampered in differential-based learning frameworks. Consequently, a quantization relaxation need to be adopted in the backward pass to overcome this limitation (i.e., when backpropagating the error gradient). One of the proposed methods in this sense includes approximating the quantization using additive uniform noise, $\eta \sim \mathcal{U}(-0.5, +0.5)$ and backpropagate it [Ballé et al. (2017)]. Independent uniform noise is often used as a quantization error model since it approximates the quantization error in terms of its marginal moments [Gray and Neuhoff (1998)]. Consequently, the same approximation can be extended to the distortion measure. For simplicity of notation, let's assume a scalar $y \in \mathbf{y}$. Thus, the resulting learned representation becomes $\tilde{y} = y + \eta$. Therefore, assuming that $p_y(y)$ represents the PDF of y, the PDF of $\tilde{y}$ can be

written as:

$$\begin{aligned} p_{\tilde{y}}(\tilde{y}) &= (p_y * \mathcal{U}(-0.5, +0.5))(\tilde{y}) \\ &= F_y(\tilde{y} + 0.5) - F_Y(\tilde{y} - 0.5) \end{aligned} \tag{3.26}$$

where $F_y(y) := \int_{-\infty}^{x} p_y(y)\, dy$ is defined as the cumulative distribution function (CDF) of $p_y(y)$. Moreover, the density function of the relaxed quantized representation ($\tilde{y}$) is a continuous relaxation of the probability mass function (PMF) of $\hat{y}$ ($p_{\hat{y}}(\hat{y})$) which implies that the differential entropy of $\tilde{y}$ ($p_{\tilde{y}}(\tilde{y})$) can be used as an approximation of the entropy of $\hat{y}$ in a gradient optimization framework. Bit-rate R_t can thus be estimated using differential entropy rather than discrete entropy during the training phase. In [Theis et al. (2017)], the authors proposed to replace the derivative of the quantization function with a smooth approximation. In both cases, the quantization is kept as it is in the forward pass (i.e., when processing input data).

3.4.3.2 Distortion

Considering that the goal is to output an image with better visual quality as perceived by a human observer, a chosen distortion measure D may account for image quality. The mean square error (MSE) 3.23 is the most adopted distortion measure across various fields, from regression to signal and image processing. It reunites desirable properties for optimization problems since it is convex and differentiable [Zhao et al. (2016)]. Note that other attractive property arises because MSE provides the maximum likelihood estimate when dealing with i.i.d. Gaussian noise. However, despite the widespread adoption of MSE, it is widely accepted that the MSE, and consequently the Peak Signal-to-Noise Ratio (PSNR), do not correlate well with image quality as perceived by a human viewer. PSNR (in dB) is defined as:

$$\begin{aligned} PSNR &= 20 \cdot \log_{10}\left(\frac{MAX_{\mathbf{x}}}{\sqrt{MSE}}\right) \\ &= 20 \cdot \log_{10}(MAX_{\mathbf{x}}) - 10 \cdot \log_{10}(MSE), \end{aligned} \tag{3.27}$$

where $MAX_{\mathbf{x}}$ is the maximum possible pixel value of the image. For example, when the pixels are represented using 12 bits per sample, this is $MAX_{\mathbf{x}} = 4095$. It turns out that MSE is not capable of capturing the elaborate characteristics of the human visual system (HVS). On the other hand, there is a vast literature of error measures that attempt to address the limitations of the simple MSE distortion. A popular perceptual distortion metric is

the structural similarity index (SSIM) [(Wang et al., 2004)]. SSIM evaluates image quality taking into account that the HVS is sensitive to changes in local structure.

To express SSIM formally [(Wang et al., 2004)], consider a pair of images $\{\mathbf{x}, \mathbf{x_r}\}$ of sizes $N_x \times N_y$. We aim to measure three aspects of similarities according to human perception: luminance (${}^{\beta}l(\mathbf{x}, \mathbf{x_r})$), contrast ($c(\mathbf{x}, \mathbf{x_r})$), and structure ($s(\mathbf{x}, \mathbf{x_r})$). These measures are quantified according to the summary of relative measures including mean (μ), variance (σ^2), and co-variance measured under sliding windows of size $\xi \times \xi$, with step size of 1 on both horizontal and vertical directions. For each sliding window, each perception subfunction for images $\mathbf{x}$ and $\mathbf{x_r}$ is computed as follows:

$$l(\mathbf{x}, \mathbf{x_r}) = \frac{2\mu_{\mathbf{x}}\mu_{\mathbf{x_r}} + C_1}{\mu_{\mathbf{x}}^2 + \mu_{\mathbf{x_r}}^2 + C_1}, \tag{3.28}$$

$$c(\mathbf{x}, \mathbf{x_r}) = \frac{2\sigma_{\mathbf{x}}\sigma_{\mathbf{x_r}} + C_2}{\sigma_{\mathbf{x}}^2 + \sigma_{\mathbf{x_r}}^2 + C_2}, \tag{3.29}$$

$$s(\mathbf{x}, \mathbf{x_r}) = \frac{2\sigma_{\mathbf{x}\mathbf{x_r}} + C_3}{\sigma_{\mathbf{x}}\sigma_{\mathbf{x_r}} + C_3}, \tag{3.30}$$

where $\{C_1, C_2, C_3\}$ are constants less than 1 to avoid potential zero division issue. Normally, $C_3 = \frac{1}{2}C_2$ [Y. Lu (2019)]. To enforce independence among those measures, the SSIM measured is defined as the product of those metrics as follows:

$$SSIM(\mathbf{x}, \mathbf{x_r}; \xi) = l(\mathbf{x}, \mathbf{x_r})^{\alpha} \cdot c(\mathbf{x}, \mathbf{x_r})^{\beta} \cdot s(\mathbf{x}, \mathbf{x_r})^{\gamma}, \tag{3.31}$$

where $\{\alpha, \beta, \gamma\}$ is a set of exponential weights [Y. Lu (2019)]. SSIM is extended in [Wang et al. (2003)] by observing the scale at which local structure should be analyzed as a function of factors such as image-to-observer distance. [Wang et al. (2003)] thus proposes MS-SSIM as a multi-scale version of SSIM that weighs SSIM computed at different scales according to the sensitivity of the HVS. Taking two images $\{\mathbf{x}, \mathbf{x_r}\}$ as the input, the system iteratively applies a low-pass filter and downsamples the filtered image by a factor of 2. The original image is indexed as Scale 1, and the highest scale as Scale M, which is obtained after $M-1$ iterations. At the j-th scale, the contrast 3.29 and the structure 3.30 are computed and denoted as $c_j(\mathbf{x}, \mathbf{x_r})$ and $s_j(\mathbf{x}, \mathbf{x_r})$, respectively. The luminance measure 3.28 is calculated on at Scale M and denoted as $l_M(\mathbf{x}, \mathbf{x_r})$. The MS-SSIM is thus obtained by combining the measurements at different scales as follows [Wang et al. (2003)]:

$$MS - SSIM(\mathbf{x}, \mathbf{x_r}; \xi) = l_M(\mathbf{x}, \mathbf{x_r})^{\alpha_M} \cdot \prod_{j=1}^{M} c_j(\mathbf{x}, \mathbf{x_r})^{\beta_j} \cdot s_j(\mathbf{x}, \mathbf{x_r})^{\gamma_j}, \tag{3.32}$$

where $\{\alpha_M, \beta_j, \gamma_j\}$ is a set of exponential weights used to adjust the importance of different components, analogously to [Y. Lu (2019)]. Note that MS-SSIM is differentiable [Zhao et al. (2016)]. However, it requires computing a pyramid of M levels of patch P, which is a computationally expensive operation given that it needs to be performed at each iteration.

3.4.3.3 Rate estimation

The rate R_t achieved by an entropy coder is lower-bounded by the entropy derived from the actual discrete probability distribution $m(\hat{\mathbf{y}})$ of the quantized vector $\hat{\mathbf{y}}$. The rate increase comes from the mismatch between the probability model $p_{\hat{\mathbf{y}}}(\hat{\mathbf{y}})$ required for the coder design and $m(\hat{\mathbf{y}})$. Consequently, the bit-rate is given by the Shannon cross entropy between the two distributions:

$$R(\hat{\mathbf{y}}) = H(\hat{\mathbf{y}}) = \mathbb{E}_{\hat{\mathbf{y}}\sim m}\left[-log_2 p_{\hat{\mathbf{y}}}(\hat{\mathbf{y}})\right], \tag{3.33}$$

The bit-rate is thus minimized if the distribution model $p_{\hat{\mathbf{y}}}(\hat{\mathbf{y}})$ is equal to the distribution $m(\hat{\mathbf{y}})$ arising from the actual distribution of the input image and from the analysis transform G_a. Note that the bit-rate estimation is derived from the differential entropy of $\tilde{\mathbf{y}}$ during training [Ballé et al. (2017) and Ballé et al. (2018)], according to Equation (3.26).

3.4.3.4 Entropy model

As stressed above, a key element in the end-to-end learned image compression frameworks is the entropy model defined through the probability model $p_{\hat{\mathbf{y}}}(\hat{\mathbf{y}})$ assigned to the quantized representation for coding. Since bit-rate estimation is derived from the differential entropy of $\tilde{\mathbf{y}}$, we considered the differential entropy here.

- Fully factorized model: For simplicity, in [Ballé et al. (2017) and Theis et al. (2017)], the approximated quantized representation was assumed independent and identically distributed within each channel and the channels were assumed independent of each other, resulting in a fully factorized distribution:

$$p_{\tilde{\mathbf{y}}|\boldsymbol{\psi}}(\tilde{\mathbf{y}}|\boldsymbol{\psi}) = \prod_i p_{\tilde{y}_i|\psi_i}(\tilde{y}_i), \tag{3.34}$$

  where index i runs over all elements of the representation, through channels and spatial locations, $\boldsymbol{\psi}_i$ is the distribution model parameter vector associated with each element. As mentioned previously, for backpropagation derivation during the training step, the

quantization process ($\hat{\mathbf{y}} = Q(\mathbf{y})$) is approximated by the addition of an i.i.d uniform noise η.

For generality, in [Ballé et al. (2017)], the distribution $p_{y_i|\psi_i}(y_i)$ is assumed non-parametric, namely without predefined shape. [Ballé et al. (2017) and Ballé et al. (2018)] use small NNs to model $p(x)$. In [Ballé et al. (2018)], a density $p : \mathbb{R} \to \mathbb{R}^+$ is defined using its CDF $c : \mathbb{R} \to [0, 1]$. Assuming the cumulative can be written as a composition of functions, then the density can be defined as a composition of functions using the chain rule of calculus as:

$$c = f_K \circ f_{K-1} \cdots f_1 \tag{3.35}$$

$$p = f_K^{'} \cdot f_{K-1}^{'} \cdots f_1^{'}, \tag{3.36}$$

where $f_k^{'}$ stands for the derivative of f_k. f_ks are allowed to be vector functions: $f_k : \mathbb{R}^{d_k} \to \mathbb{R}^{r_k}$. In general, the $f_k^{'}$ are Jacobian matrices, and $\cdot$ stands for matrix multiplications [Ballé et al. (2018)]. The domain of f_1 and the range of f_K need to be one dimensional ($d_1 = r_K = 1$) to ensure $p(x)$ is univariate. All the Jacobian elements are required to be non-negative to guarantee that $p(x)$ is a density [Ballé et al. (2018)]. According to [Ballé et al. (2018)], an effective choice of f_k is the following:

$$f_k(\mathbf{x}) = g_k\left(\mathbf{H}_k\mathbf{x} + \mathbf{b}_k\right) \qquad 1 \leq k < K \tag{3.37}$$

$$f_K(\mathbf{x}) = sigmoid\left(\mathbf{H}_K\mathbf{x} + \mathbf{b}_K\right) \tag{3.38}$$

where $\mathbf{H}_k$ are matrices, $\mathbf{b}_k$ are vectors and g_k are non-linearities defined as follows:

$$g_k(\mathbf{x}) = \mathbf{x} + \mathbf{a}_k \odot tanh(\mathbf{x}), \tag{3.39}$$

where $\mathbf{a}_k$ is a vector and $\odot$ denotes elementwise multiplication. Each univariate density model is thus associated to its own set of parameters $\{\mathbf{a}_k, \mathbf{b}_k, \mathbf{H}_k\}$. Note that these parameters together form $\boldsymbol{\psi}^{(i)}$. For all experiments performed in [Ballé et al. (2018)], the authors used $K = 4$, with the dimensionalities $r_1 = r_2 = r_3 = 3$. In [Ballé et al. (2018)], the authors found that this model fits well to arbitrary densities. Note that in [Ballé et al. (2017) and Theis et al. (2017)], the parameter vectors are learned from data during the training phase. This learning, performed once and for all, prohibits adaptivity to the input images during operational phase. Moreover, the simplifying hypothesis of a fully-factorized distribution is very strong and not satisfied in practice, since elements of $\hat{\mathbf{y}}$ can exhibit strong spatial dependency as observed in [Ballé

et al. (2018)]. To overcome these limitations and thus to obtain a more realistic and more adaptive entropy model, [Ballé et al. (2018)] proposed a hyperprior model, derived through a variational autoencoder, which takes into account possible spatial dependency in each input image.

- Hyperprior model: Auxiliary random variables $\tilde{\mathbf{z}}$, conditioned on which $\tilde{\mathbf{y}}$ elements are independent, are derived from $\tilde{\mathbf{y}}$ by an auxiliary autoencoder, connected in parallel with the bottleneck. The hierarchical model hyper-parameters are learned for each input image in operational phase. In [Ballé et al. (2018)], $\tilde{\mathbf{z}}$ distribution is assumed fully factorized and each representation element y_i, knowing $\tilde{\mathbf{z}}$, is modeled by a zero-mean Gaussian distribution with its own standard deviation σ_i. Finally, taking into account the quantization process, the conditional distribution of each $\tilde{y}_i$ is given by:

$$\tilde{y}_i|\tilde{\mathbf{z}} \sim \mathcal{N}\left(0, {\sigma_i}^2\right) * \mathcal{U}\left(-0.5, +0.5\right). \tag{3.40}$$

  The rate computation must take into account the prior distribution of $\tilde{\mathbf{z}}$.

3.5 Denoising with neural networks

Compared to standard denoising techniques, denoising CNNs are able to adapt to the dataset as well as to the noise model. Numerous works have been proposed in the past few years. We will only focus on residual learning of deep CNN for image denoising (DnCNN) [K. Zhang et al. (2017a)], a fast and flexible solution for CNN based image denoising (FFDNet) [K. Zhang et al. (2018)], and Image denoising using deep CNN with batch renormalization (BRDNet) [Tian et al. (2020)]. It is important to remember that we were not interested in deep networks for onboard compression due to their inherent high complexity. However, in denoising, deep networks constitute the state-of-the-art, so we turn our attention to them in this case.

3.5.1 General concepts

Residual learning (RL) In denoising neural networks, instead of learning a CNN that tries to output a noise-free image directly, residual learning considers a different learning perspective that targets predicting a residual image. The residual image is the difference between the corrupted and clean reference images. Residual learning [He et al. (2016)] in CNN was primitively conceived to address the performance degradation issues related to

training very deep neural networks, i.e., networks that present a high number of layers. The residual mapping is assumed to be much easier to be learned than the direct noise-free image, and this approach led to an easier training phase and improved accuracy for image classification and object detection using remarkably deep neural architectures [He et al. (2016)].

Batch normalization (BN) When training CNNs, mini-batch gradient descendent has been widely used. However, its training efficiency is negatively impacted by internal covariate shift [Ioffe and Szegedy (2015)], i.e., abrupt changes in the distributions of inputs during training. Thus, batch normalization (BN) [Ioffe and Szegedy (2015)] is used to ease the internal covariate shift. For a layer composed by a K-dimensional input $\mathbf{x} = (x_1 \dots x_K)$, each dimension, for $j = 1, ..., K$, is normalized as follows [Ioffe and Szegedy (2015)]:

$$\hat{x}_j = \frac{x_j - E\left[x_j\right]}{\sqrt{Var\left[x_j\right]}}, \tag{3.41}$$

where the expectation and variance are computed over the training data set. Note that simply normalizing each input of a layer may change what the layer can effectively represent, i.e., normalizing the inputs of an activation function such as the Sigmoid would constrain them to the linear regime of the non-linear activation function [Ioffe and Szegedy (2015)]. In order to address this issue, scale (γ_j) and shift (β_j) steps are integrated after the normalization step and preceding the nonlinearity in each layer as:

$$y_j = \gamma_j \hat{x}_j + \beta_j. \tag{3.42}$$

These parameters are learned during the training phase, along with the original architecture parameters. Thus, the proposed BN method can prevent exploding or vanishing gradients, accelerate the convergence of the network, and improve performance and stability [Ioffe and Szegedy (2015)]. Besides, BN also provides regularization. Therefore, a NN is expected to be robust enough to the internal covariate shift during training. Note that the batch normalization has to calculate mean and variance to normalize the previous outputs across the batch. This statistical estimation will be as much as more accurate if the batch size is reasonably large while keeps on decreasing as the batch size decreases. Since the batch normalization calculates the mean and variance to normalize the previous outputs across the batch, their statistical estimation will be as much as more accurate if the batch size is reasonably large. However, it keeps on decreasing as the batch size decreases. On the

other hand, BN is not effective for small mini-batches, which limits its applications, e.g., for image detection and video tracking [Ioffe (2017)].

Batch renormalization (BRN) Batch renormalization (BRN) was proposed to deal with the small mini-batch issue that affects BN [Ioffe (2017)]. BRN is applied over a mini-batch preceding the nonlinearity in each layer according to the Algorithm 4. During backpropagation, a standard chain rule is used. The values marked with *stop-gradient* are treated as constant for a given training step, and the gradient is not propagated through them.

Algorithm 4: Batch Renormalization implementation

Input: Values of x over a training mini-batch $B = \{x^{(1)} \dots x^{(m)}\}$;
γ and β: parameters;
μ and σ: current moving mean ad standard deviation, respectively;
α: moving average update rate;
r_{max} and d_{max}: maximum allowed corrections;
$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$;
$\sigma_B \leftarrow \sqrt{\epsilon + \frac{1}{m}\sum_{i=1}^{m}\left(x^{(i)} - \mu_B\right)}$;
$r \leftarrow \text{stop-gradient}\Big(clip_{[1/r_{max}, r_{max}]}\left(\frac{\sigma_B}{\sigma}\right)\Big)$;
$r \leftarrow \text{stop-gradient}\Big(clip_{[-d_{max}, d_{max}]}\left(\frac{\mu_B - \mu}{\sigma}\right)\Big)$;
$\hat{x}^{(i)} \leftarrow \frac{x^{(i)} - \mu_B}{\sigma_B} \cdot r + d$;
$y^{(i)} \leftarrow \gamma \hat{x}^{(i)} + \beta$;
$\mu := \mu + \alpha(\mu_B - \mu)$;
$\sigma := \sigma + \sigma(\sigma_B - \sigma)$;
Result: $y_i = BatchRenorm(x_i)$; updated μ and σ
Inference: $y \leftarrow \gamma \cdot \frac{x - \mu}{\sigma} + \beta$

BRN can effectively address the small mini-batch issue and non-independent identically distributed mini-batch problems since it uses individual samples to approximate the training data distribution. The non-independent identically distributed mini-batch issue occurs when samples in the same mini-batch are non-i.i.d, leading to poor performance of the trained neural network.

3.5.2 Residual learning of deep CNN for image denoising (DnCNN)

In [K. Zhang et al. (2017a)], the authors investigated the construction of feedforward denoising convolutional neural networks (DnCNNs) to embrace the progress in deep learning into image denoising. To architecture design, the VGG, which stands for Visual Geometry Group network [Simonyan and Zisserman (2014)], was modified to adapt it for image denoising. VGG consists of a classical CNN architecture that relies on 16 layers to increase its performance on ground-breaking object recognition applications. VGG's convolutional layers leverage a minimum kernel size, i.e., 3×3, the smallest possible size that still captures vertical and horizontal spatial information to minimize computational complexity impact. Note that the receptive field is a relevant concept in designing deep CNNs. The receptive field is defined as the region size in the input that produces the feature. It measures the association of an output feature (of any layer) to the input region (patch) [Araujo et al. (2019)]. Ideally, a higher receptive field is desired to ensure that no crucial information is not taken into account by the convolutional layers. The convolution stride is fixed at 1 pixel to keep the spatial resolution preserved after convolution and all pooling layers were removed. After convolution, the VGG architecture relies on the ReLU as its activation function. The VGG achieves almost 92.7% top-5 test accuracy in ImageNet. ImageNet is a dataset consisting of more than 14 million images belonging to nearly 1000 classes [J. Deng et al., 2009]. The VGG architecture surpasses baselines on many tasks and datasets beyond ImageNet, making it one of the most popular image recognition architectures. For learning the network, the authors adopted residual learning (RL) and batch normalization (BN), previously presented in 3.5.1 and 3.5.1 respectively, to improve the denoising performance and for fast training.

Rather than outputting the denoised image directly, the proposed DnCNN is designed to predict a residual image $F(\mathbf{I_n}) = \mathbf{I_n} - \mathbf{I_{nf}}$, i.e., the difference between the noisy image ($\mathbf{I_n}$) and the noise-free image ($\mathbf{I_{nf}}$), which is the principle of residual learning. Differently from the originally proposed residual network [He et al., 2016] that presents several residual connections, the proposed DnCNN adopted a single residual unit to extract the residual image [K. Zhang et al. (2017a)] from a noisy input image.

The architecture of the proposed DnCNN for learning the residual image is illustrated in figure 3.9.

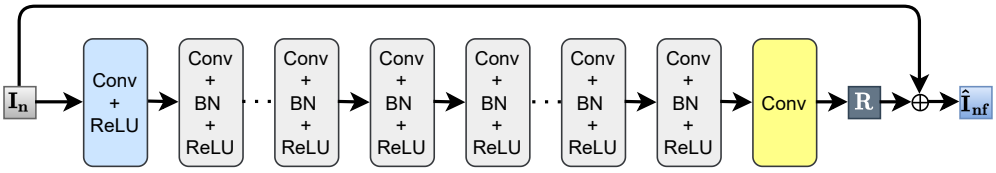


Figure 3.9: Architecture of the DnCNN network [K. Zhang et al. (2017a)].

A DnCNN architecture (with d layers), consists of three types of layers, shown in figure 3.9. (a) Conv + ReLU: for the first layer, 64 filters with kernel size 3×3 followed by ReLU nonlinear activation functions are used to generate 64 feature maps. (b) Conv + BN + ReLU: from layers 2 to $(d-1)$, 64 filters with kernel size 3×3 are used and batch normalization [Ioffe and Szegedy (2015)] is performed between convolution and ReLU. (c) Conv: in the last layer, 64 filters with kernel size 3×3 are used to predict the noise residual image $f(\hat{\mathbf{I}}_{\mathbf{n}})$.

Following the same principle in [Simonyan and Zisserman (2014)], the kernel size of convolutional filters was also set to 3×3 and all pooling layers were removed. Consequently, the receptive field associated to the DnCNN with depth d is $(2d+1) \times (2d+1)$ [K. Zhang et al. (2017a)]. Note that an increased receptive field allows the use of the information present in a larger image region. Setting a proper depth for DnCNN represents an important issue for finding a better trade-off between performance and complexity. It is worth mentioning that the receptive field size in denoising neural networks is analogous to the effective patch size of model-based denoising methods [Burger et al. (2012) and Jain and Seung (2008)]. For Gaussian denoising with a specific noise level, the number of layers composing the DnCNN architecture is set to 17, and it is set to 20 for other general image denoising tasks [K. Zhang et al. (2017a)]. The averaged MSE (3.23) between the desired residual images and the estimated ones from noisy input can be adopted as the loss function as follows:

$$J(\mathbf{x}, \theta) = \frac{1}{N_p} \sum_{i=1}^{Nt} ||F(\mathbf{I_n}^{(i)}) - (\mathbf{I_n}^{(i)} - \mathbf{I_{nf}}^{(i)})||^2, \tag{3.43}$$

where N_p represents the noisy/noise-free training image pairs.

In practice, various image denoising tasks can be implemented by employing the proposed DnCNN method. In [K. Zhang et al. (2017a)], the authors consider three specific tasks, i.e., blind Gaussian denoising, single image super-resolution problem, and JPEG deblocking. In the training stage, the authors used images with AWGN from a wide range of noise levels, downsampled images with multiple upscaling factors, and JPEG images with different

quality factors to train a single DnCNN architecture. Experimental results show that the learned DnCNN can provide great results for any of the three general image denoising tasks.

3.5.3 Toward a fast and flexible solution for CNN based Image denoising (FFDNet)

CNNs have demonstrated tremendous success for image denoising, notably using residual learning. This success is attributed to their large modeling capacity and advances in deep learning techniques. However, existing CNN-based denoising methods have been limited in flexibility, and the trained network usually doesn't perform well in handling different noise levels. For example, a single DnCNN trained for Gaussian denoising fails in generalizing well to real noisy images and works only if the noise level is within a preset range [K. Zhang et al. (2017a)]. Moreover, the existing residual learning-based methods are not able to deal with spatially variant noise.

To overcome the limitations mentioned above, the authors in [K. Zhang et al. (2018)] presented a fast and flexible denoising convolutional network (FFDNet). The FFDNet architecture consists of three types of layers, shown in figure 3.10. (a) Conv + ReLU: for the first layer. (b) Conv + BN + ReLU: for the middle layers, and (c) Conv: for the last convolutional layer.

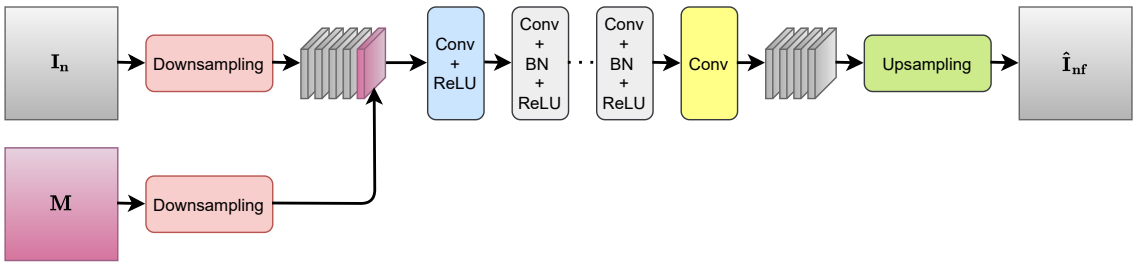


Figure 3.10: Architecture of the FFDNet network [K. Zhang et al. (2018)].

After the last convolution layer, an upscale operator is applied to produce the denoised image ($\mathbf{I_{nf}}$) as the reverse operator of the downsampling operator initially applied in the input image. Different from DnCNN [K. Zhang et al. (2017a)], FFDNet does not adopt residual learning to predict a noise map. The choice for operating on downsampled sub-images avoids the need to increase the receptive field further, e.g., by employing dilated convolutions [F. Yu and Koltun (2015)]. The FFDNet is formulated as a residual mapping, which is here expressed as $F(\mathbf{I_n}, \mathbf{M})$, where $\mathbf{M}$ denotes a noise level map. Consequently, in

the FFDNet, the noise level map is incorporated as an input, and the network parameters are invariant to the noise level. It is expected that the FDDNet performs well when the input noise map matches the noise degradation present in the noisy image. Furthermore, the noise level map also assumes the role of controlling the trade-off between noise reduction and detail preservation. The authors also introduced a reversible downsampling operation to reshape the input mage of size $N_x \times N_y \times C$ into four downsampled sub-images of size $N_x/2 \times N_y/2 \times 4C$. The authors also concatenate a tunable noise level map $\mathbf{M}$ with the downsampled sub-images to form a tensor $\mathbf{X_{in}}$ of size $N_x/2 \times N_y/2 \times (4C + 1)$. When considering spatially invariant AWGN with noise level σ, $\mathbf{M}$ is a uniform map with all elements equal to σ.

The authors set the number of convolutional filters as 15 for grayscale images and 12 for color images to balance complexity and performance. The authors claim that using fewer convolutional layers encourages the architecture to exploit the inter-channel dependency in color images. Concerning the number of feature maps defined by the number of filters, the authors chose 64 for grayscale images and 96 for color images. Experimentally, the authors also found that employing more filters per layer is beneficial when dealing with color images. Taking into account flexibility, efficiency and effectiveness, FDDNet consists on a practical solution to CNN denoising applications.

3.5.4 Image denoising using deep CNN with batch renormalization (BRDNet)

More recently in [Tian et al. (2020)], the authors proposed a novel deep learning framework designated batch-renormalization denoising network (BRDNet). BRDNet uses batch renormalization (BRN) [Ioffe (2017)] to deal with small mini-batch convergence issues in BN and adopts RL similarly to [K. Zhang et al. (2017a)]. Batch Renormalization is applied over a mini-batch preceding the nonlinearity in each layer according to the Algorithm 4.

BRDNet also blends two parallel sub-networks to obtain more relevant features for improving the denoising performance [Szegedy et al. (2015)]. Furthermore, the idea in this model was to increase the width of the network rather than the depth (e.g., number of layers) and thus avoid vanishing or exploding gradients issues during training, which mainly affect deeper networks. The proposed network is shown in figure 3.11.

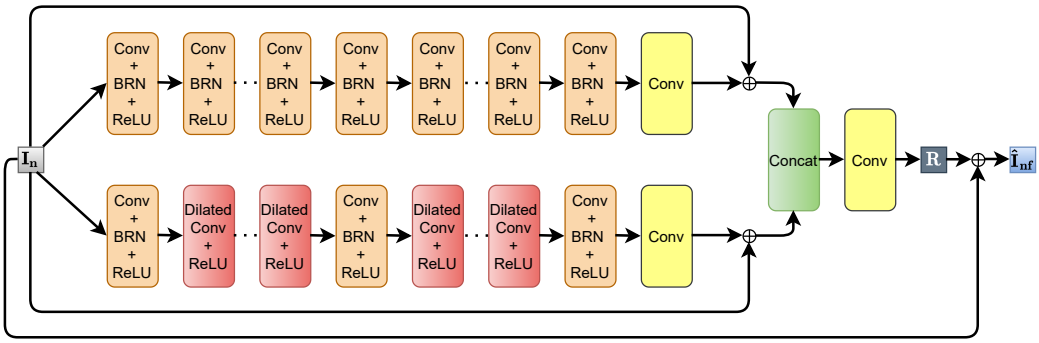


Figure 3.11: Architecture of the BRDNet network [Tian et al. (2020)].

$\mathbf{I_n}$ and $\hat{\mathbf{I}}_{\mathbf{nf}}$ denote the noisy image and the denoised image, respectively. We observe that the upper network mainly involves RL and BRN, while the lower network features RL, BRN, and dilated convolutions [F. Yu and Koltun (2015)]. The authors claim that dilated convolutions allow the extraction of more context information with relatively lower complexity when compared to conventional CNNs, mostly because they attain the same receptive field with few layers and parameters. In [Tian et al. (2020)], experiments demonstrated that the proposed BRDNet outperforms state-of-the-art deep learning denoising methods, e.g., DnCNN [K. Zhang et al. (2017a)], but also the fast and flexible denoising network (FFDNet) [K. Zhang et al. (2018)] and the image-restoration CNN (IRCNN) [K. Zhang et al. (2017b)].

3.5.5 Neural networks for compression artifacts suppression

Lossy image compression allows high compression rates by discarding information that introduces image distortion. The resulting image quality degradation may be acceptable in many applications, but the produced visual artifacts become unacceptable at elevated compression rates or in certain critical applications. Blocking, blurring, and ringing artifacts are common examples of image degradation arising from traditional lossy image compression methods.

All lossy compression algorithms produce ringing artifacts [Yang et al. (2005)]. They consist of a Gibbs type of oscillations around sharp-intensity transitions in the image. Consequently, unpleasing short vertical and horizontal "ridges" appear in the image, representing the ringing artifacts. These artifacts appear at high compression ratios in all transform-based codecs due to the low-pass nature of such systems as JPEG and JPEG2000. These

systems are by far the most popular compression systems. The classic JPEG compression algorithm produces blocking artifacts at high compression ratios. This artifact originates from the independent quantization of the block discrete cosine transform (DCT) coefficients used in the JPEG algorithm [Yang et al. (2005)]. When considering the JPEG2000 method, visual quality degradation commonly consists of prominent ringing and blurring artifacts [M. W. Marcellin et al. (2002)]. The JPEG2000 operates in the wavelet domain, endeavoring to represent an image as a sum of smooth oscillating waves. Blurring means that the image is smoother than originally. Texture information is usually discarded during lossy compression, while shape information is preserved. Blocking is mostly noticeable in low-frequency regions, while the ringing artifacts are especially well perceptible around sharp edges. Note that the blurring effect is less visually unpleasant compared to the blocking artifacts.

A wide variety of methods designed to reduce compression artifacts varies from relatively simple and fast handcrafted designed filters to fully probabilistic image restoration methods with complex priors [T.-S. Wong et al. (2009)] and new methods based on machine learning advances [Dong et al. (2014)]. Most image and video viewing software features simple deblocking and artifact removal post-processing filters. For example, the FFmpeg codec disposes of a simple post-processing filter (spp) [Nosratinia (1999)] that simply re-applies JPEG compression to the shifted versions of the previously compressed image, and averages the results. In Pointwise Shape-Adaptive DCT (SA-DCT) [Foi et al. (2007)], which is currently considered the state-of-the-art deblocking method among the learning-free methods, a local estimate of the signal within the adaptive form support is reconstructed from thresholded or attenuated transformation coefficients. However, similar to other deblocking methods, SA-DCT is not able to sharpen the edges and it excessively smooths images.

Convolutional networks have been used to suppress compression artifacts by [Dong et al. (2015)], who proposed the artifacts removing CNN network (AR-CNN) based on the super-resolution CNN (SRCNN) [Dong et al. (2014)], originally proposed to deal with the super-resolution problem. The SRCNN method consists of 3 feature extraction layers, a high dimensional mapping layer, and a final reconstruction layer. The SRCNN also adopts the so-called residual learning. Thus, residual learning also appears to be very important in super-resolution/compression artifacts suppression. AR-CNN [Dong et al. (2015)] extends the original SRCNN architecture [Dong et al. (2014)] with feature enhancement layers. The network training consist of two stages – a shallow network is trained first and used as an initialization for a final 4 layer CNN. According to the authors in [Dong et al. (2015)], the two stage training approach enhanced results when compared to training the 4 layer CNN

directly from scratch.

3.5.6 Denoising with GANs

GANs are used mostly as alternatives to traditional distortion measures such as MSE. They are capable of obtaining better perceptual image quality and visual performance. Recalling the concepts concerning generative adversarial networks (GANs) [Goodfellow et al. (2014)] presented in 3.2.4, we are now interested in denoising with GANs. In this case, the generator network is used to create solutions to an indistinguishable image with a ground truth noise-free image ($\mathbf{I_n f}$) by receiving a noisy image ($\mathbf{I_n}$) as input [Zhong et al. (2020)]. The discriminator network, in turn, receives the noise-free image ($\mathbf{I_n f}$) and is expected to distinguish the generated denoised image ($\hat{\mathbf{I_n}}$) from the ground truth noise-free image ($\mathbf{I_n f}$).

GANs have become famous for their ability to produce photorealistic images with high quality [Zhong et al. (2020)]. On the other hand, training GANs represents a great challenge due to problems such as vanishing gradients and mode colapse [Salimans et al. (2016)]. When used in image denoising, one major drawback for GANs is their tendency to synthesize some areas in the denoised image. Although presenting visually pleasing synthesized regions, this effect is undesirable in applications that rely on higher image fidelity. Consequently, they are not good candidates for satellite image denoising.

Chapter 4

Learned reduced-complexity onboard satellite image compression

This chapter is adapted from [Alves de Oliveira et al. (2021)]

Contents

4.1 Focus on the two reference learning-based architectures . . . 60
4.1.1 Recalling the main characteristics . . . 60
4.1.2 Statistical analysis of the learned transform . . . 67
4.2 Reduced-complexity variational autoencoder . . . 71
4.2.1 Simplified analysis and synthesis transforms . . . 71
4.2.2 Simplified entropy model . . . 72
4.3 Performance analysis . . . 74
4.3.1 Implementation setup . . . 75
4.3.2 Subjective image quality assessment . . . 76
4.3.3 Impact of the number of filter reduction . . . 78
4.3.4 Impact of the filter kernel support in the main autoencoder . . . 84
4.3.5 Impact of the GDN/IGDN replacement in the main autoencoder . . . 84
4.3.6 Impact of the number of layer reduction in the main autoencoder . . . 86
4.3.7 Impact of the entropy model simplification . . . 88
4.3.8 Discussion about complexity . . . 90
4.4 Conclusions . . . 91

This chapter starts with the description of two interesting frameworks [Ballé et al. (2017) and Ballé et al. (2018)] that we selected among the state-of-the-art to design a reduced-complexity framework adapted to satellite image compression. Besides improved performance, recall that the learned frameworks usually involve high computational complexity. Through cautious simplifications of [Ballé et al. (2018)], we seek an alternative solution with similar performance as [Ballé et al. (2018)] and similar or perhaps lower complexity than [Ballé et al. (2017)].

Note that most of the state-of-the-art deep learning architectures are briefly presented in the literature, and more emphasis is given to their performance. To achieve an effective complexity reduction, it is necessary first to understand the importance of each architecture element and, subsequently, how any simplification or replacement impacts the resulting performance. For this reason, each component of the original architecture, which includes filters, activation functions, and the entropy model, is studied in detail. Finally, complexity reductions are proposed based on these studies.

4.1 Focus on the two reference learning-based architectures

4.1.1 Recalling the main characteristics

In the context of image compression, autoencoders are used to learn a representation with low entropy after quantization. Autoencoders are composed of an analysis transform and a synthesis transform connected by a bottleneck. When devoted to compression, the bottleneck performs quantization and entropy coding. Please note that the dequantization process is integrated in the synthesis transform. An auxiliary autoencoder can also be used to infer the probability distribution of the representation as in [Ballé et al. (2018)].

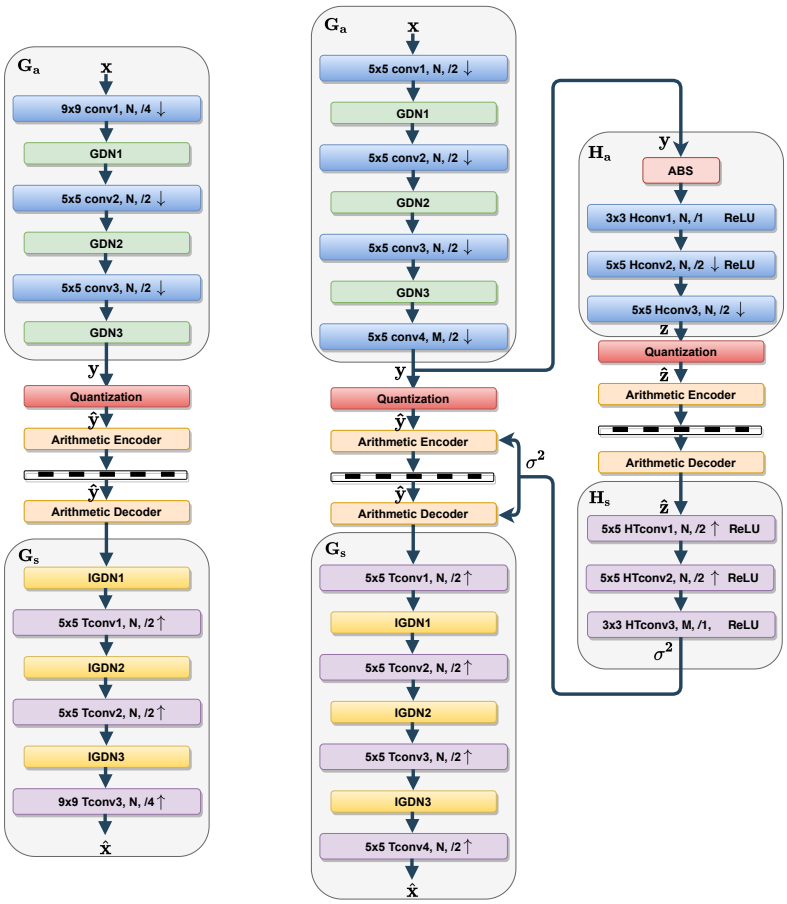


Figure 4.1: Architecture of the autoencoder [Ballé et al. (2017)] (**left**) and of the variational autoencoder [Ballé et al. (2018)] (**right**).

Here we focus on two reference architectures: [Ballé et al. (2017)] displayed in figure 4.1 (left) and [Ballé et al. (2018)] displayed in figure 4.1 (right). The reference architecture [Ballé et al. (2017)] is composed of a single autoencoder. The second reference architecture [Ballé et al. (2018)] is composed of a main autoencoder (slightly different than the one in [Ballé et al. (2017)]) and of an auxiliary one which aims to infer the probability distribution of the learned representation.

4.1.1.1 Analysis and synthesis transforms

In the main autoencoder (figure 4.1 (left) and left column of figure 4.1 (right)), the analysis transform G_a is applied to the input image $\mathbf{x}$ to produce a representation $\mathbf{y} = G_a(\mathbf{x})$. After the bottleneck, the synthesis transform G_s is applied to the quantized representation $\hat{\mathbf{y}}$

to reconstruct the image $\hat{\mathbf{x}} = G_s(\hat{\mathbf{y}})$. These representations are derived through several layers composed of filters, some of them followed by a non-linear activation function. The learned representation is multi-channel (the output of a particular filter is called a channel or a feature map) and non-linear. As previously mentioned, the analysis and synthesis transforms proposed in [Ballé et al. (2018)] result from improvements (mainly parameter adjustments) of the ones proposed in [Ballé et al. (2017)]. Thus, for brevity, the following description focuses on [Ballé et al. (2018)].

In [Ballé et al. (2018)], the analysis (resp. synthesis) transform G_a (resp. G_s) is derived through 3 convolutional layers each composed of N filters with kernel support 5×5 associated with parametric activation functions called Generalized Divisive Normalizations (GDN) (resp. Inverse Generalized Divisive Normalizations (IGDN)) and a downsampling (resp. upsampling) by a factor 2. These three convolutional layers are linked to the input (resp. output) of the bottleneck by a convolutional layer composed of $M > N$ (resp. N) filters with the same kernel support but without activation function. The fact that the last layer of the synthesis transform is composed of $M > N$ filters leads to a so-called wide bottleneck. According to [Ballé et al. (2018)], a wide bottleneck (large M) allows to attain comparable performance with low values of N and to accommodate higher bitrates more efficiently. [Ballé et al. (2018)] also stated that there exists a particular number of filters per layer that reaches a performance saturation level. Figure 4.2 (resp. figure 4.3) displays the before-mentioned four-layer analysis transform (G_a) (resp. four-layer synthesis transform (G_s)) [Ballé et al. (2018)]. These figures display the tensor-transforms performed along with the successive convolutional layers from the input to the recovered image. The activation functions GDN (resp. IGDN) are also shown. N_x and N_y define the width and height of the input image. Note that downsampling (resp. upsampling) operations accompany the convolutional layers. These operations lead to representations with low 2D dimensions at the bottleneck. Note that downsampling (resp. upsampling) is performed during the convolutional (resp. upsampling) operation by selecting the stride and padding adequately for computational efficiency [Ballé et al. (2017)].

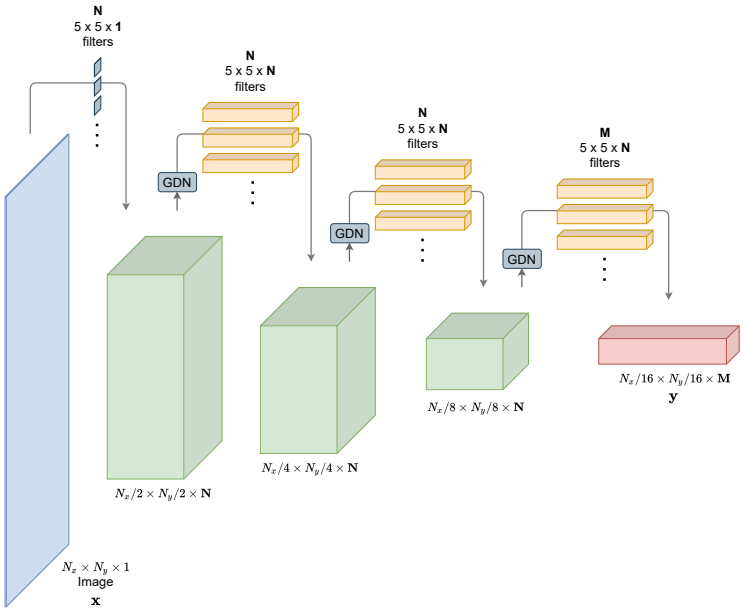


Figure 4.2: Four-layer analysis transform G_a [Ballé et al. (2018)].

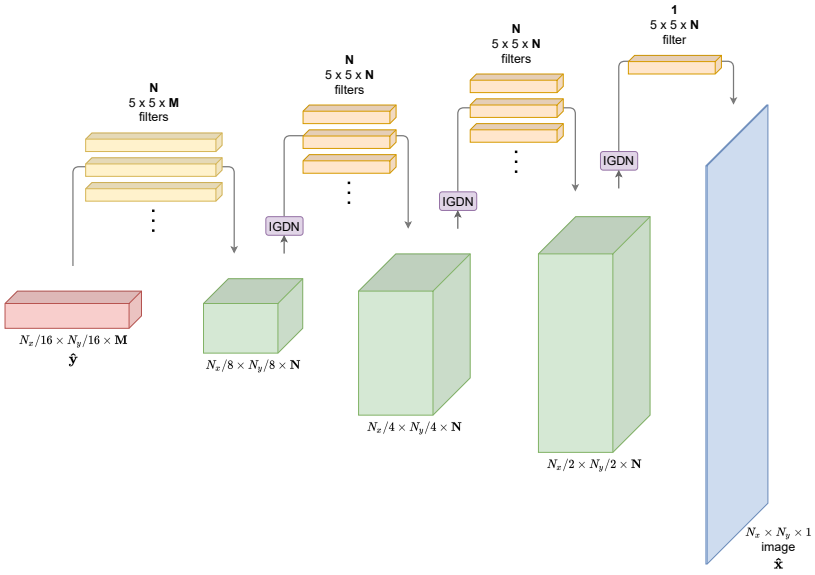


Figure 4.3: Four-layer synthesis transform G_s [Ballé et al. (2018)].

4.1.1.2 Loss function: rate distortion trade-off

The autoencoder parameters (filter weights, GDN/IGDN parameters and representation distribution model) are jointly learned through the optimization of a loss function involving the rate $R(\hat{\mathbf{y}})$ and the distortion $D(\mathbf{x}, \hat{\mathbf{x}})$ between the original image $\mathbf{x}$ and the reconstructed image $\hat{\mathbf{x}}$. The rate-distortion criterion, denoted as $J(\mathbf{x}, \hat{\mathbf{x}}, \hat{\mathbf{y}})$, writes as the weighted sum:

$$J(\mathbf{x}, \hat{\mathbf{x}}, \hat{\mathbf{y}}) = \lambda D(\mathbf{x}, \hat{\mathbf{x}}) + R(\hat{\mathbf{y}}), \tag{4.1}$$

where λ is a key parameter that tunes the rate-distortion trade-off. The loss function defined in Equation (4.1) is minimized through gradient descent with backpropagation [Bengio (2009)] on a representative image training set.

Distortion The distortion D is chosen to account for image quality as perceived by a human observer. Due to its many desirable computational properties, the mean square error (MSE) is generally selected. However, a measure of perceptual distortion may also be employed such as the multi-scale structural similarity index (MS-SSIM) [Wang et al. (2003)], as described in 3.4.3.2.

Rate The bit-rate is minimized if the entropy model ($p_{\hat{\mathbf{y}}}(\hat{\mathbf{y}})$) is equal to the actual distribution of the learned image representation $m(\hat{\mathbf{y}})$, as described in 3.4.3.3. After training, a range encoder losslessly compresses the quantized learned representation [Ballé et al. (2017)]. A range encoder can encode integer data into strings using cumulative distribution functions (CDF).

4.1.1.3 Entropy model

As previously presented in chapter 3, a critical element in the end-to-end learned image compression frameworks is the entropy model defined through the probability model $p_{\hat{\mathbf{y}}}(\hat{\mathbf{y}})$ assigned to the quantized representation. The entropy model is necessary to compress the quantized representation efficiently in a lossless manner. However, note that the bit-rate estimation is derived from the differential entropy of $\tilde{\mathbf{y}}$. The differential entropy consists of a continuous relaxation used only for training purposes that already incorporates the quantization effect by an additive uniform noise (η), as explained in 3.4.3.4. The resulting learned representation becomes $\tilde{\mathbf{y}} = \mathbf{y} + \eta$. For simplicity, we consider the differential entropy here in the following formulations.

- Fully factorized model: In [Ballé et al. (2017) and Theis et al. (2017)] for simplicity, the approximated quantized representation was assumed independent and identically distributed within each channel, and the channels were considered independent of each other, resulting in a fully factorized distribution:

$$p_{\tilde{\mathbf{y}}|\boldsymbol{\psi}}(\tilde{\mathbf{y}}|\boldsymbol{\psi}) = \prod_i p_{\tilde{y}_i|\psi^{(i)}}(\tilde{y}_i), \tag{4.2}$$

  where index i runs over all elements of the representation, through channels and through spatial locations, $\boldsymbol{\psi}^{(i)}$ is the distribution model parameter vector associated with each element. In [Ballé et al. (2017)], for generality purpose, the distribution $p_{\tilde{y}_i|\boldsymbol{\psi}^{(i)}}(\tilde{y}_i)$ is assumed non-parametric. This model assumes that each channel would present some unknown distribution with an arbitrary and potentially complex shape In [Ballé et al. (2017) and Theis et al. (2017)], the parameter vectors are learned during the training phase from data. Note that this learning process prohibits adaptivity during the operational phase since the entropy model parameters are assumed fixed once training is completed, no matter the input image.

- Hyperprior model: An additional set of auxiliary random variables $\tilde{\mathbf{z}}$ is derived from $\tilde{\mathbf{y}}$ by an auxiliary autoencoder, connected in parallel with the bottleneck (right column of figure 4.1 (right)). The conditional model hyper-parameters are thus learned for each input image in operational phase. First, the hyperprior transform analysis H_a produces a set of auxiliary random variables $\tilde{\mathbf{z}}$. Second, $\tilde{\mathbf{z}}$ is transformed by the hyperprior synthesis transform H_s into a second set of random variables $\boldsymbol{\sigma}$. Figure 4.4 (resp. figure 4.5) displays the before-mentioned three-layer hyperprior analysis transform (H_a) (resp. four-layer hyperprior synthesis transform (H_s)) [Ballé et al. (2018)]. These figures display the tensor-transforms performed along with the successive convolutional layers from $\tilde{\mathbf{y}}$ to the conditional entropy model parameters $\boldsymbol{\sigma}$. Note that downsampling (resp. upsampling) operations accompany the convolutional layers. In [Ballé et al. (2018)], $\tilde{\mathbf{z}}$ distribution is assumed fully factorized and each representation element y_i, knowing $\tilde{\mathbf{z}}$, is modeled by a zero-mean Gaussian distribution with its own standard deviation σ_i produced by H_s. Thus, taking into account the quantization process, the conditional distribution of each $\tilde{y}_i$ is expressed by:

$$\tilde{y}_i|\tilde{\mathbf{z}} \sim \mathcal{N}\left(0, {\sigma_i}^2\right) * \mathcal{U}\left(-\frac{1}{2}, \frac{1}{2}\right). \tag{4.3}$$

  During compression, the rate computation has to consider the entropy model of $\tilde{\mathbf{z}}$ since

$\tilde{\mathbf{z}}$ is compressed and transmitted to the decoder to perform the complete decompression of $\tilde{\mathbf{y}}$.

Note that the fully-factorized entropy model is very generic whereas the second one requires high computational complexity. Indeed, in neural network learning, especially in deep networks, complex methods are employed to handle a wide variety of data. However, for onboard compression purpose, some prior knowledge would be welcome to simplify the entropy model. In the following, we perform a statistical analysis of the learned transform on a representative set of satellite images. The objective is to propose an entropy model simpler than the two previous ones, while being adaptive to the input images.

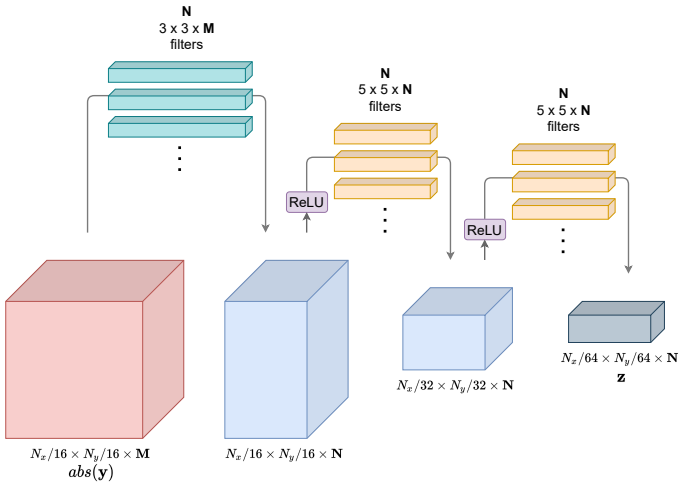


Figure 4.4: Three-layer hyperprior analysis transform H_a [Ballé et al. (2018)].

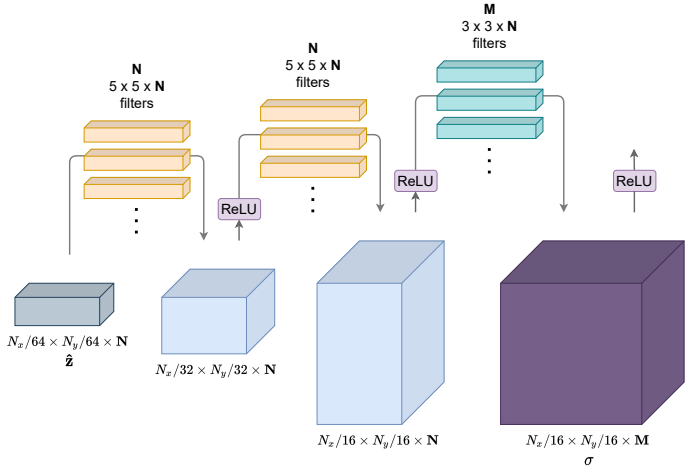


Figure 4.5: Three-layer hyperprior synthesis transform H_s [Ballé et al. (2018)].

4.1.2 Statistical analysis of the learned transform

Since the network must also learn the underlying distribution of the learned representation, we can try to provide it with a priori information to simplify its task. This subsection first performs a statistical analysis of each feature of the learned representation in the particular case of satellite images. A similar statistical analysis has been conducted in the case of natural images in [Dumas et al. (2018)] with the objective to properly design the quantization in the architecture of [Ballé et al. (2017)]. The probability density function related to each feature was estimated on a representative set composed of 24 natural images through a normalized histogram. The study showed that most features can be accurately modelled as Laplacian random variables. Interestingly, a similar result has also been analytically demonstrated in [Lam and Goodman (2000)] for block-DCT coefficients of natural images under the assumption that the variance is constant on each image block and that its values on the different blocks are distributed according to an exponential or a half-normal distribution. We conducted the statistical analysis on the representation obtained by the main autoencoder as defined in [Ballé et al. (2018)], with $N = 128$ and $M = 192$, but used alone, as in [Ballé et al. (2017)], without auxiliary autoencoder. First, the main autoencoder in [Ballé et al. (2018)] benefits from improvements concerning the one in [Ballé et al. (2017)]. Second, the auxiliary autoencoder is not necessary for this statistical study. This autoencoder is trained on a representative dataset of satellite images described in 2.6 and for rates between 2.5 bits/pixel and 3 bits/pixel. This rate range represents the target operating compression rates in satellite image compression. First, as an illustration, let consider the satellite image displayed in figure 4.6. This image of the city of Cannes (French Riviera) is a 12-bit simulated panchromatic Pléiades image with size 512×512 and resolution 70 cm.

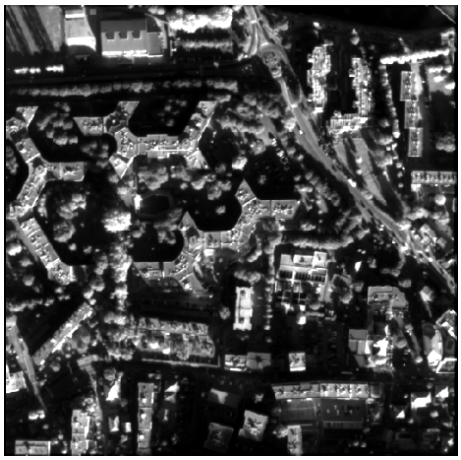
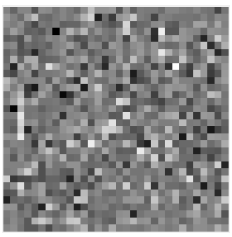
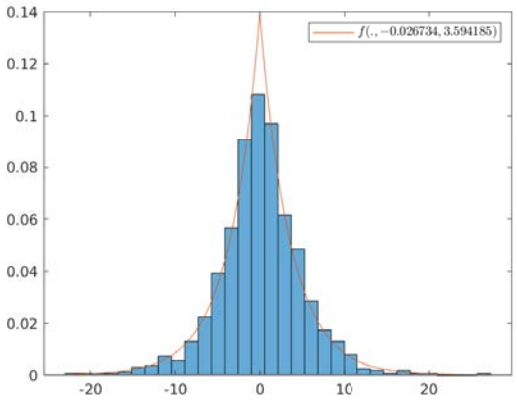


Figure 4.6: Simulated 12-bit Pléiades image of Cannes with size 512×512 and resolution 70 cm

Figure 4.7 shows the 1st 32×32 feature derived from the Cannes image and its normalized histogram with Laplacian fitting.



(**a**) 1st feature.



(**b**) Normalized histogram and Laplacian fitting.

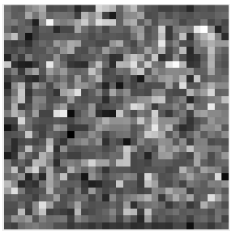
Figure 4.7: First feature of Cannes image representation, its normalized histogram with Laplacian fitting.

On this example, the fitting with an almost centered Laplacian seems appropriate. According to the Kolmogorov-Smirnov goodness-of-fit test [Pratt and Gibbons (1981)], 94% of the features derived from this image follow a Laplacian distribution with a significance level $\alpha = 5\%$. Recall that the Laplacian distribution Laplace(μ, b) is defined by the probability

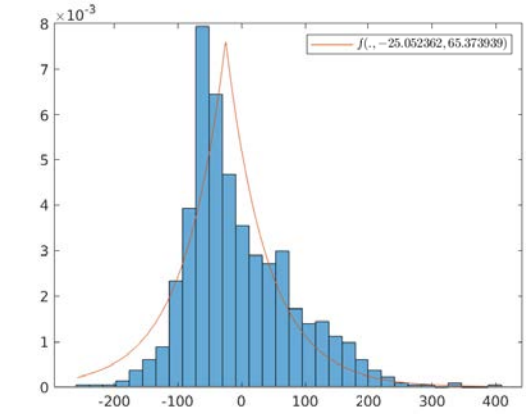
density function:

$$f(\zeta, \mu, b) = \frac{1}{2b} exp\left(-\frac{|\zeta - \mu|}{b}\right) \text{ for } \zeta \in \mathbb{R}, \tag{4.4}$$

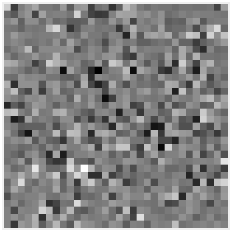
where μ is the mean value and $b > 0$ is a scale parameter related to the variance by $Var(\zeta) = 2b^2$. The remaining non-Laplacian feature maps (6% of the maps for this example) stay close to the Laplacian distribution. Figure 4.8 displays two representative examples of non-Laplacian feature maps. Please note that the first one (19th feature map) is a low-pass approximation of the input image. However, this is a very particular case: the second displayed feature has a typical behavior, not so far from a Laplacian distribution.



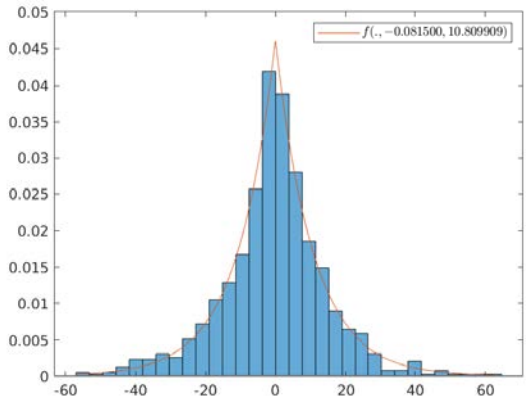
(**a**) 19th feature.



(**b**) Normalized histogram and Laplacian fitting.



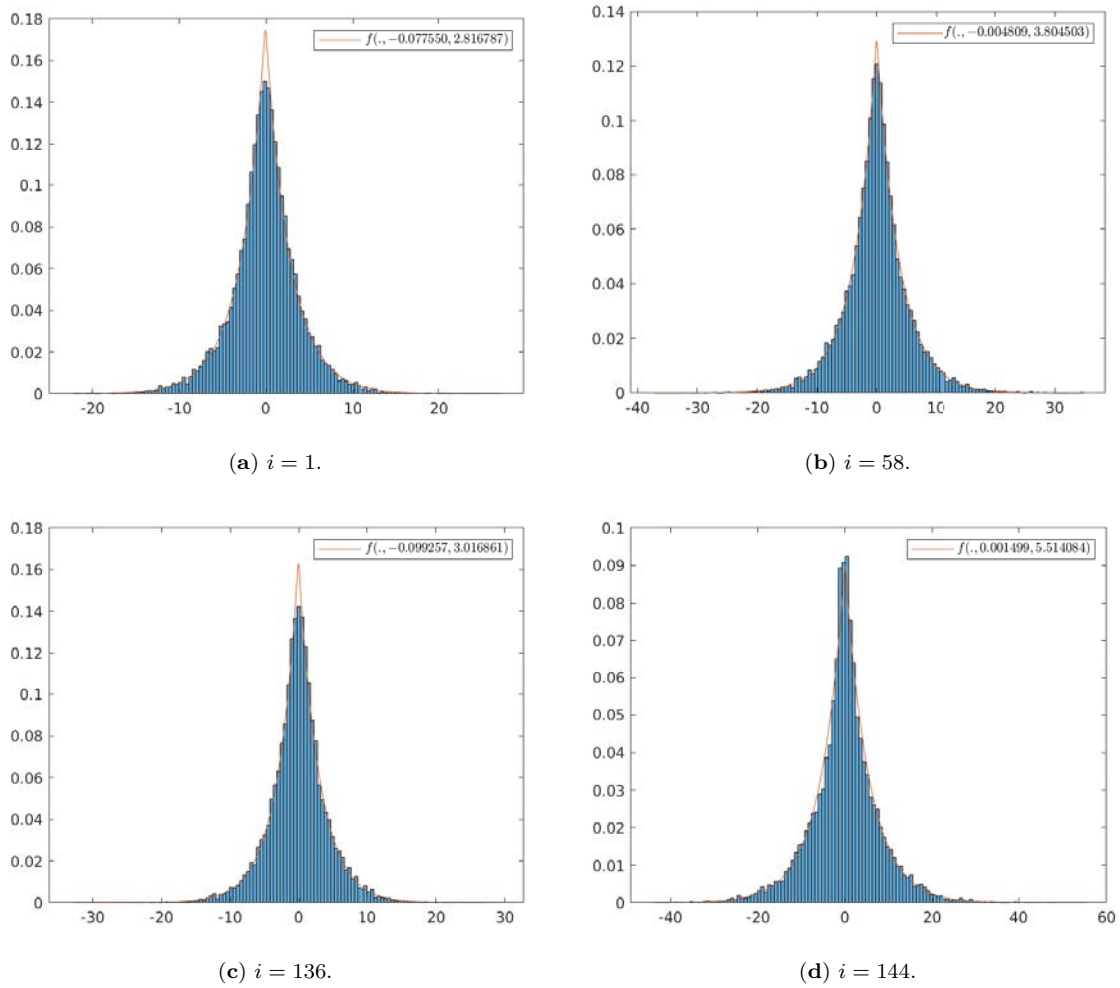
(**c**) 55th feature.



(**d**) Normalized histogram and Laplacian fitting.

Figure 4.8: Normalized histogram of the ith feature map and Laplacian fitting $f(., \mu, b)$.

To extend this result, the representation (composed of $M = 192$ feature maps) was derived for 16 simulated 512×512 Pléiades images. Figure 4.9 shows the normalized histograms of some feature maps derived from these 16 images and the Laplacian fitting.



(**a**) $i = 1$.

(**b**) $i = 58$.

(**c**) $i = 136$.

(**d**) $i = 144$.

Figure 4.9: Normalized histogram of the ith feature map and Laplacian fitting $f(., \mu, b)$.

We can also observe that most of the feature maps have a similar normalized histogram. Please note that the Gaussian distribution, $\mathcal{N}(\mu, \sigma^2)$ with a small value of μ, also fits the features albeit to a lesser extent. This statistical study will allow us to propose a more specific entropy model than those proposed in [Ballé et al. (2017) and Ballé et al. (2018)] and, therefore, more computationally efficient.

4.2 Reduced-complexity variational autoencoder

In the literature, the design of learning-based image compression frameworks hardly takes into account the computational complexity: the objective is merely to obtain the best performance in terms of rate-distortion trade-off. However, in the context of onboard compression, a trade-off between performance and complexity has also to be considered to take into account the strong computational constraints.

Our focus here is to propose a complexity-reduced alternative to the state-of-the-art architecture [Ballé et al. (2018)] while minimizing the impact on the performance. Please note that this complexity reduction must be essentially targeted at the coding part of the framework, which is subject to the onboard constraints. A meaningful indicator of the architecture complexity is the number of its parameters. Its reduction has a positive impact not only on the memory complexity, but also on the training difficulty. Indeed, the optimization process involved in the training step applies on fewer parameters. This is an advantage even if the training is performed on ground. The convergence is thus obtained with less iterations and thus faster. Moreover, the reduction of the number of parameters has also a positive impact on the ease to upload the final model to the spacecraft (once the training with real data completed) as the up-link transmission is severely limited. Note, however, that the time complexity is not proportional to the number of parameters as detailed in 4.4.

4.2.1 Simplified analysis and synthesis transforms

Our approach to reduce the complexity of the analysis and synthesis transforms, while maintaining an acceptable rate-distorsion trade-off, is to fine-tune the number of layers, the parameters of each layer (number of filters and filter sizes) and to consider the replacement of GDN/IGDN by simpler non-parametric activation functions.

In a first step, we propose a Simplified entropy model. The state-of-the-art frameworks generally involve a large number of filters with the objective of increasing the network approximation capability [Ballé et al. (2017) and Ballé et al. (2018)]. However, a high number of filters also implies a high number of parameters and operations in the GDN/IGDN, as it increases the depth of the tensors (equal to the number of filters) at their input. Apart from a harder training, an increased number of filters comes with a high number of operations and a high memory complexity, which is problematic in onboard compression. In [Ballé et al. (2018)], the number of filters at the bottleneck (M) and for the other layers (N) are

fixed according to the target bit-rate: the higher the target bit-rate, the higher M and N. Indeed, higher bit rates mean lower distortion and thus increased network approximation capabilities [Ballé et al. (2018)]. This principle also applies to the auxiliary autoencoder implementing the hyperprior illustrated in figure 4.1 (right column of the right part). In the present chapter, we propose and evaluate a reduction of the number of filters in each layer for different target rate ranges. In particular, we investigate the impact of M on the attainable performance when imposing a drastic reduction of N. The question is whether there is a real need for a high global number of filters (high N and M) or whether a high bottleneck size (low N and high M) is sufficient to achieve good performance at high rates. For that purpose, we impose a low value of N (typically $N = 64$) and we consider increasing values of M defined by $M = 2N, 3N, 4N, 5N$ to determine the minimum value of M that leads to an acceptable performance in a given rate range.

In a second step, we investigate the replacement of the GDN/IGDN by non-parametric activation functions. As previously mentioned, according to [Ballé (2018)], GDN/IGDN allow obtaining good performance even with a low number of layers. However, for the sake of completeness, we also test their replacement by ReLU functions. Finally, we propose to evaluate the effect of the filter kernel support. The main autoencoder in [Ballé et al. (2018)] is entirely composed of filters with kernel support $n \times n$. The idea then is to test different kernel supports that is $(n-2) \times (n-2)$ and $(n+2) \times (n+2)$. The case with a larger kernel size is tested to check if performance improvements can be obtained by only increasing the kernel size.

4.2.2 Simplified entropy model

The entropy model simplification aims at achieving a compromise between simplicity and performance while preserving the adaptability to the input image. In [Ballé et al. (2017)], the representation distribution is assumed fully factorized and the statistical model for each feature is non-parametric to avoid a prior choice of a distribution shape. This model is learned once, during the training. In [Ballé et al. (2018)], the strong independence assumption leading to a fully factorized model is avoided by the introduction of the hyperprior distribution, whose parameters are learned for each input image even in the operational phase. Both models are general and thus suitable to a wide variety of images; however the first one implies a strong hypothesis of independence and prohibits adaptivity to the input image while the second one is computationally expensive. Based on the previous analysis, we propose the following parametric model for each of the M features. Consider the jth

feature elements y_{i_j} for $i_j \in I_j$, where I_j denotes the set of indexes covering the jth feature:

$$\begin{aligned} y_{i_j} &\sim \text{Laplace}(0,\, b_j) \text{ (resp. } y_{i_j} \sim \mathcal{N}(0,\, \sigma_j^2)) \\ \text{with: } b_j &= \sqrt{Var(y_{i_j})/2} \text{ (resp. } \sigma_j^2 = Var(y_{i_j})). \end{aligned} \tag{4.5}$$

The problem then boils down to the estimation of a single parameter per feature referred to as the scale b_j (respectively the standard deviation σ_j) in the case of the Laplacian (resp. Gaussian) distribution. Starting from [Ballé et al. (2018)], this proposal reduces the complexity at two levels. First, the hyperprior autoencoder, including the analysis H_a and synthesis H_s transforms, is removed. Second, the side information initially composed of the compressed auxiliary random variable set ($\mathbf{z}$) of size $8 \times 8 \times M$ now reduces to a $M \times 1$ vector of variances.

As a fully factorized model, this model supposes that the learned representation elements are i.i.d. within a particular feature map, which is a simplifying but strong assumption. Concerning the mean values for each feature map, we can observe that they are mostly close to zero. Note that the data is automatically centered to zero within our software implementation, and the mean shift is included in the bit stream. Compared to the fully factorized non-parametric entropy model [Ballé et al. (2017)], the simplified entropy model is adaptive given that the scale parameter (b_j) is estimated for each feature of each input image. For the feature map that appears as a low-pass approximation of the input image, as observed in figure 4.8, it is difficult to find a general parametric distribution with a good fit. Moreover, it is not straightforward to find a priori its position. This feature map is also modelled using the Laplacian distribution, which is not bit-rate efficient. We expect, however, to keep the entropy model as simple as possible and that this particular feature modelling does not penalize too much the overall efficiency of our proposed model.

The auxiliary network simplification is displayed on the right part of figure 4.10.

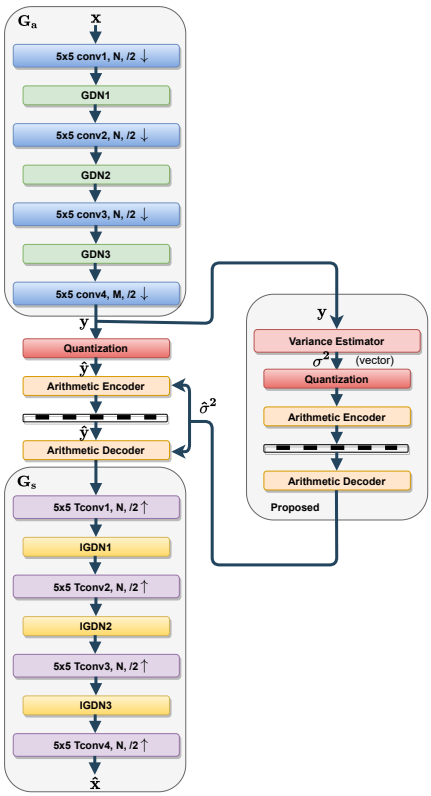


Figure 4.10: Proposed architecture after entropy model simplification: main autoencoder [Ballé et al. (2018)] (**left column**) and simplified auxiliary autoencoder (**right column**).

In the next section, the performance of these proposals and of their combinations are studied for different rate ranges.

4.3 Performance analysis

This section assesses the performance, in terms of rate-distortion, of the proposed architecture in comparison with the CCSDS 122.0-B [B. Book (2017)], JPEG2000 [Marcellin and Taubman (2002)] and with the reference methods [Ballé et al. (2017) and Ballé et al. (2018)]. Beforehand, a subjective image quality assessment is proposed. Although informal, it allows comparing the artefacts produced by learned compression and by the CCSDS 122.0-B [B. Book (2017)].

4.3.1 Implementation setup

To assess the relevance of the proposed complexity reductions, experiments were conducted using TensorFlow. The batch size (i.e., the number of training samples to work through before the parameters are updated) was set to 8 and up to 1M iterations were performed. Both training and validation datasets are composed of simulated 12-bit Pléiades panchromatic images provided by the CNES, covering various landscapes (i.e., desert, water, forest, industrial, cloud, port, rural, urban). These datasets are detailed in 2.6. The reference learned frameworks for image compression are designed to handle 8-bit RGB natural images and they generally target low rates (typically up to a maximum of 1.5 bits/pixel). In contrast, onboard satellite compression handles 12-bit panchromatic images and targets higher rates (from 2bits/pixel to 3.5 bits/pixel). The training dataset is composed of 8M of patches (of size 256×256) randomly cropped from 112 images (of size 585×585). The validation dataset is composed of 16 images (of size 512×512). MSE was considered to be the distortion metric for training. The rate and distortion measurements were averaged across the validation dataset for a given value of λ. Please note that the value of λ has to be set by trial and error for a targeted rate range.

In addition to the MSE, we also evaluate those results in terms of MS-SSIM. Both metrics were detailed in 3.4.3.2. Please note that they exhibit a similar behavior even if the models were trained for the MSE only. The proposed framework is compared with the CCSDS 122.0-B [B. Book (2017)], JPEG2000 [Marcellin and Taubman (2002)] and with the reference methods [Ballé et al. (2017) and Ballé et al. (2018)] implemented for values of N and M recommended by their authors for particular rate ranges.

When naming the different learning frameworks, we intend to distinguish between the main autoencoder architecture and the entropy model since we tested different combinations of both. In this case, we designate `AE-1` to the main autoencoder [Ballé et al. (2017)] and `AE-2` refers to the main autoencoder [Ballé et al. (2018)]. With respect to the entropy model, `-NP-` designates the non-parametric fully-factorized entropy model [Ballé et al. (2017)], `-H-` designates the hyperprior entropy model [Ballé et al. (2018)] and `-L-` designates the simplified Laplacian entropy model. Lastly, all the frameworks considered in this chapter terminates with `-C` which indicates that they were trained exclusively for compression. This suffix indication will be more evident in the following chapter. In this chapter, we consider both following learned framework baselines:

- `AE-1-NP-C` refers to the compression framework originally proposed in [Ballé et al. (2017)]. This framework is originally implemented for $N = 192$ (respectively $N = 256$)

for rates below 2 bits/pixel (respectively above 2bits/pixel). Note that the number of filters composing each layer is the same for all layers in this architecture.

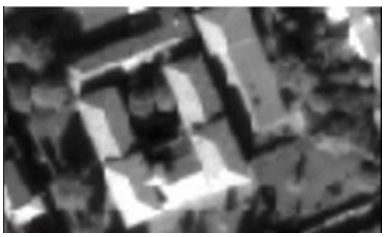
- `AE-2-H-C` refers to the compression framework proposed in [Ballé et al. (2018)]. This framework is implemented for $N = 128$ and $M = 192$ (respectively $N = 192$ and $M = 320$) for rates below 2 bits/pixel (respectively above 2 bits/pixel).

4.3.2 Subjective image quality assessment

At low rates, JPEG2000 is known to produce quite prominent blurring and ringing artifacts, which is particularly visible in high-frequency textures [Marcellin and Taubman (2002)]. This is also the case for the CCSDS 122.0-B [B. Book (2017)]. Figure 4.11a,b shows the original image of the city of Blagnac, which is a 12-bit simulated panchromatic Pléiades image with size 512 × 512 and resolution 70 cm. The figure also shows the image compressed by CCSDS 122.0 (c) and the image compressed by the reference learned compression architecture `AE-1-NP-C` (d), for a low compression rate (1.15 bits/pixel). The image obtained through learned compression appears closest to the original one than the image obtained through the CCSDS.



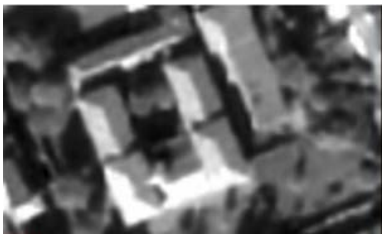
(**a**) Original image.



(**b**) Zoom on the original image.



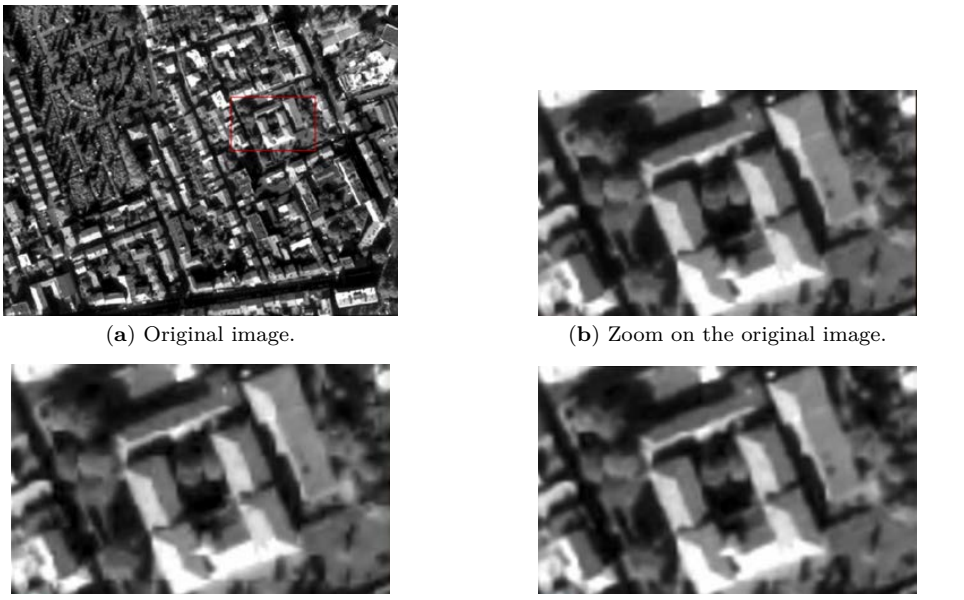
(**c**) Zoom on the CCSDS compressed image.



(**d**) Zoom on the end-to-end compressed image.

Figure 4.11: Subjective image quality analysis (R = 1.15 bits/pixel).

For medium luminances, far less artifacts, such as blurring and flattened effects are observed. In particular, the building edges are sharper. The same is true for low luminances, corresponding to shaded areas: the ground markings are sharp and less flatened areas are observed. As shown in figure 4.12, for a higher rate of 1.66 bits/pixel, the two reconstructed images are very close. However, the image obtained through learned compression remains closest to the original one, especially in shaded areas.



(**a**) Original image.

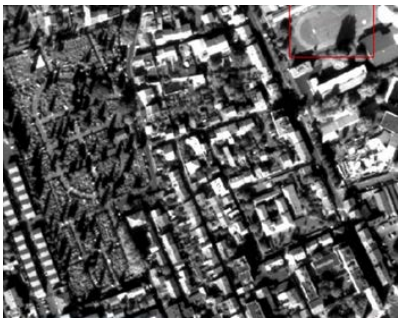
(**b**) Zoom on the original image.

(**c**) Zoom on the CCSDS compressed image.

(**d**) Zoom on the end-to-end compressed image.

Figure 4.12: Subjective image quality analysis (R = 1.66 bits/pixel).

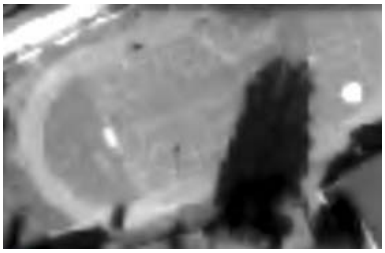
Finally, as shown in figure 4.13 for an even higher rate of 2.02 bits/pixel, CCSDS 122.0 still produces flattened effects, especially in low variance areas. The learned compression method leads to a reconstructed image that is closest to the original one. Even though the stadium ground markings slightly differs from the original, the image quality is overall preserved.



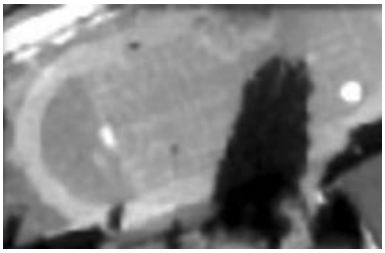
(**a**) Original image.



(**b**) Zoom on the original image.



(**c**) Zoom on the CCSDS compressed image.



(**d**) Zoom on the end-to-end compressed image.

Figure 4.13: Subjective image quality analysis (R = 2.02 bits/pixel).

Finally, for various rates, the learned compression method [Ballé et al. (2017)] does not suffer from the troublesome artifacts induced by the CCSDS 122.0, leading to a more uniform image quality. The same behavior was observed for the proposed simplified architecture.

In the following, an objective performance analysis is performed in terms of rate-distortion trade-off for the CCSDS 122.0-B [B. Book (2017)], JPEG2000, the reference architectures [Ballé et al. (2017)] and [Ballé et al. (2018)] and the proposed simplified ones.

4.3.3 Impact of the number of filter reduction

When considering the number of filters, it is essential to distinguish between the bottleneck size - defined by the number of filters M in the layer just before the bottleneck - and the other layer's number of filters N. At first, we consider a reduction of N, and then we evaluate the impact of the bottleneck size M.

4.3.3.1 Apart from the before-bottleneck layer

At Low Rates We first consider architectures devoted to low rates, say up to 2 bits/pixel. Starting from Ballé et al. (2018) denoted as `AE-2-H-C`, the number of filters N (for all layers,

Table 4.1: Detailed complexity of `AE-2-H-C (N=64, M=192)`

Layer	**Filter size**		**Channels**		**Output**		**Parameters**	**FLOPp**
	n	n	N_{in}	N_{out}	s_{out}	s_{out}		
conv1	5	5	1	64	256	256	1664	4.16×10^2
GDN1							4160	1.04×10^3
conv2	5	5	64	64	128	128	102464	6.40×10^3
GDN2							4160	2.60×10^2
conv3	5	5	64	64	64	64	102464	1.60×10^3
GDN3							4160	0.65×10^2
conv4	5	5	64	192	32	32	307392	1.2×10^3
Hconv1	3	3	192	64	32	32	110656	4.32×10^2
Hconv2	5	5	64	64	16	16	102464	1.00×10^2
Hconv3	5	5	64	64	8	8	102464	0.25×10^2
HTconv1	5	5	64	64	16	16	102464	1.00×10^2
HTconv2	5	5	64	64	32	32	102464	4.00×10^2
HTconv3	3	3	64	192	32	32	110784	4.32×10^2
Tconv1	5	5	192	64	64	64	307264	4.80×10^3
IGDN1							4160	0.65×10^2
Tconv2	5	5	64	64	128	128	102464	6.40×10^3
IGDN2							4160	2.60×10^2
Tconv3	5	5	64	64	256	256	102464	2.56×10^4
IGDN3							4160	1.04×10^3
Tconv4	5	5	64	1	512	512	1601	1.60×10^3
Total							1683969	5.2264×10^4

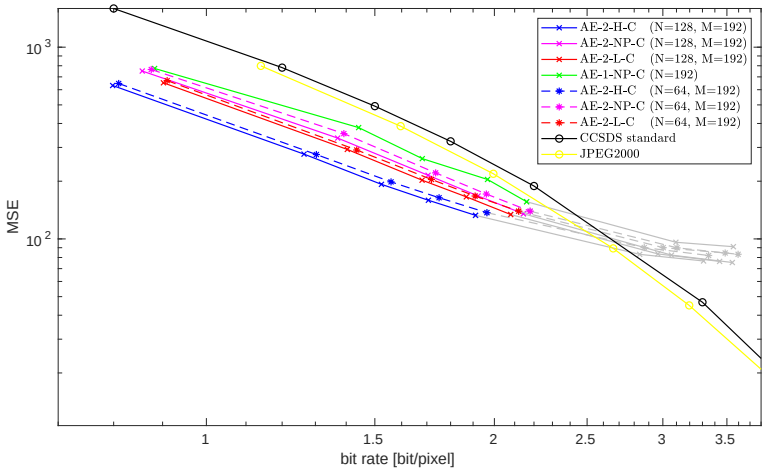
Table 4.2: Comparative complexity of the global architectures - Case of target rates up to 2 bits/pixel.

Method	**Parameters**	**FLOPp**	**Relative**
`AE-2-H-C (N=128, M=192)`	5055105	1.9115×10^5	1.00
`AE-2-H-C (N=64, M=192)`	1683969	5.2264×10^4	0.27
`AE-2-L-C (N=64, M=192)`	1052737	5.0774×10^4	0.265

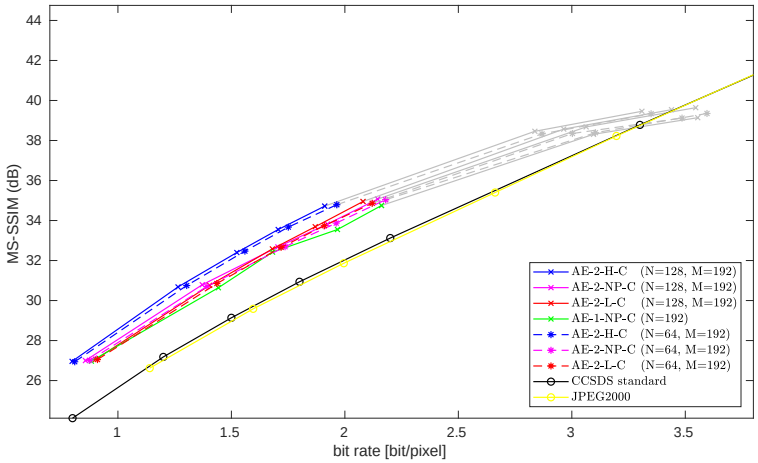
apart from the one just before the bottleneck) is reduced from $N = 128$ to $N = 64$, keeping $M = 192$ for the layer just before the bottleneck. This reduction is applied jointly to the main autoencoder and to the hyperprior one. The proposed simplified architecture is termed `AE-2-H-C (N=64, M=192)`. The complexity of this model is evaluated in terms of number of parameters and of floating point operation per pixel (FLOPp) in Table 4.1. Note that the complexity and FLOPp calculation is described in the Appendix A.1.

Table 4.2 compares the complexity of `AE-2-H-C (N=64, M=192)` to the reference `AE-2-H-C (N=128, M=192)`. The complexity of the proposed simplified architecture is 73% lower in terms of FLOPp with respect to the reference architecture. Now let consider the impact on compression performance of the reduction of N. Figure 4.14 displays the performance, in terms of MSE and MS-SSIM, of our different proposed solutions, of the

reference learned architectures (`AE-2-H-C (N=128, M=192)` and `AE-1-NP-C (N=192)`) and of the JPEG2000 [Marcellin and Taubman (2002)] and CCSDS 122.0-B [B. Book (2017)] standards. The gray curve portions indicate that the values of N and M are not recommended for this rate range (above 2 bits/pixel). In this first experiment, we are mainly concerned by the comparison of the blue lines: the solid one for the reference architecture `AE-2-H-C (N=128, M=192)` and the dashed one for the proposal `AE-2-H-C (N=64, M=192)`.



(**a**) Log–log scale. Distortion measure: MSE.



(**b**) Distortion measure: MS-SSIM (dB).

Figure 4.14: Rate-distortion curves for the considered learned frameworks and for the CCSDS 122.0-B [B. Book (2017)] and JPEG2000 [Marcellin and Taubman (2002)] standards in terms of MSE and MS-SSIM (dB) (derived as $-10\log_{10}(1-\text{MS-SSIM})$) - Case of rates up to 2 bits/pixel.

Table 4.3: Comparative complexity of the global architectures - Case of target rates above 2 bits/pixel.

Method	Parameters	FLOPp	Relative
`AE-2-H-C (N=192, M=320)`	11785217	4.3039×10^5	1.00
`AE-2-H-C (N=64, M=320)`	1683969	5.6966×10^4	0.13
`AE-2-L-C (N=64, M=320)`	1052737	5.4774×10^4	0.1273

As expected, `AE-2-H-C (N=64, M=192)` achieves a rate-distortion performance close to the one of `AE-2-H-C (N=128, M=192)` [Ballé et al. (2018)], both in terms of MSE and MS-SSIM, for rates up to 2 bits/pixel. We can conclude that the decrease in performance is very small, keeping in mind the huge complexity reduction. Please note that our proposal outperforms by far CCSDS 122.0-B [B. Book (2017)], JPEG2000 [Marcellin and Taubman (2002)] standards as well as `AE-1-NP-C (N=192)` [Ballé et al. (2017)].

At High Rates Now let consider the architectures devoted to higher rates, say above 2 bits/pixel. For such rates, the reference architectures involve a high number of filters ($N = 256$ in [Ballé et al. (2017)], $N = 192$ and $M = 320$ in [Ballé et al. (2018)]). Starting from [Ballé et al. (2018)], we reduced the number of filters to $N = 64$ in all layers except the one before the bottleneck, keeping $M = 320$. The proposal `AE-2-H-C (N=64, M=320)` is compared to the reference `AE-2-H-C (N=192, M=320)` but also to `AE-1-NP-C (N=256)` and to JPEG2000 [Marcellin and Taubman (2002)] and CCSDS 122.0-B [B. Book (2017)] standards. Table 4.3 compares the complexity of `AE-2-H-C (N=64, M=320)` to the reference `AE-2-H-C (N=192, M=320)`.

The complexity of the proposed simplified architecture is 87% lower in terms of FLOPp with respect to the reference method. Now let consider the impact on compression performance of the reduction of N. figure 4.15 displays the performance, in terms of MSE only, of our different proposed solutions, of the reference learned methods and of the JPEG2000 and CCSDS standards. The MS-SSIM shows the same behaviour.

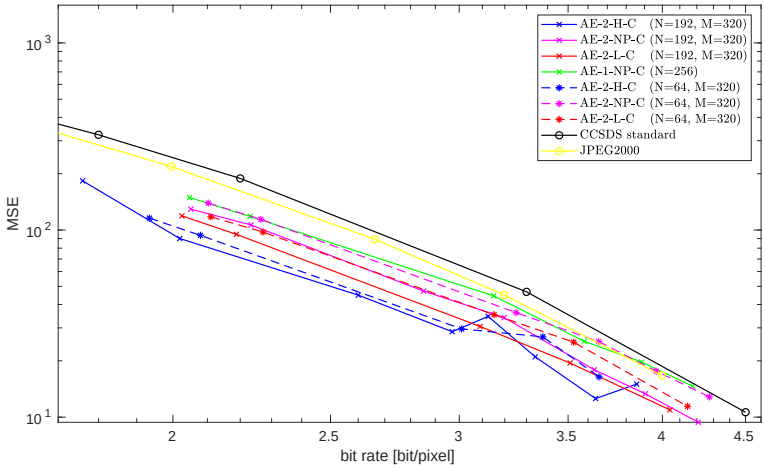
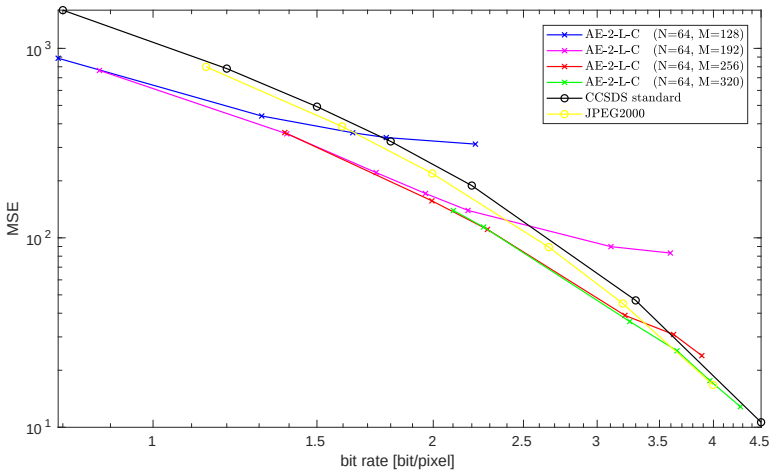


Figure 4.15: Rate-distortion curves at higher rates for learned frameworks and for the CCSDS 122.0-B [B. Book (2017)] and JPEG2000 [Marcellin and Taubman (2002)] standards for MSE in log-log scale - Case of high rates (above 2 bits/pixel).

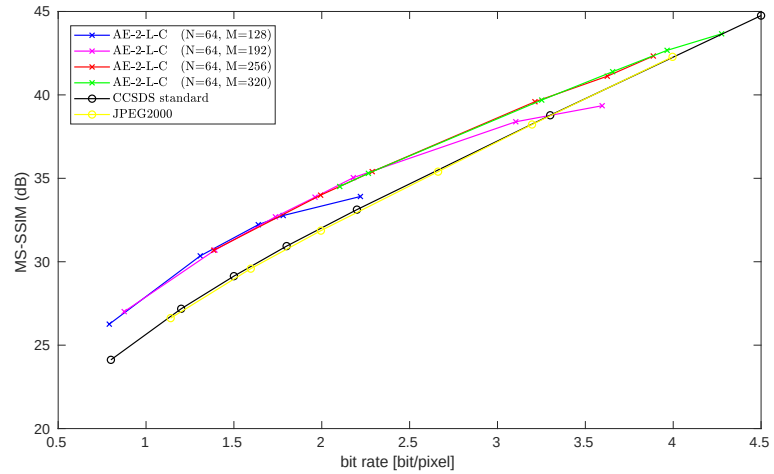
Theses curves show that the simplified architectures (e.g., resulting from a decrease of N) by far outperform the JPEG2000 and CCSDS standard even at high rates, while showing a low decrease in performance with respect to the reference architectures. Note however that, for both the reference (`AE-2-H-C (N=192, M=320)`) and the simplified (`AE-2-H-C (N=64, M=320)`) architectures with hyperpriors, a training of 1M iterations seems insufficient for the highest rates. Indeed, due to the auxiliary autoencoder implementing the hyperprior, the training has conceivably to be longer, which can be a disadvantage in practice. This may be an additional argument to propose a simplified entropy model.

4.3.3.2 The before-bottleneck layer

As previously highlighted, the bottleneck size M plays a key role in the performance of the considered architectures. Thus, we now consider a fixed low value of N ($N = 64$) and then we vary the bottleneck size ($M = 128, 192, 256$ and 320). This experiment, performed on the proposed architecture integrating the simplified entropy model `AE-2-L-C (N=64)`, allows quantifying the impact of M on the performance in terms of both MSE and MS-SSIM for increasing values of the target rate on the proposed architecture. Figure 4.16 shows the rate-distortion averaged over the validation dataset. According to the literature, high bit rates require a large global number of filters [Ballé et al. (2018)].



(**a**) Log–log scale. Distortion measure: MSE.



(**b**) Distortion measure: MS-SSIM (dB).

Figure 4.16: Impact of the bottleneck size in terms of MSE and MS-SSIM (dB) (derived as $-10\log_{10}(1-\text{MS-SSIM})$) .

Figure 4.16 shows that increasing the bottleneck size M only, while keeping N very small, allows maintaining the performance as the rate increases. As displayed in figure 4.16, while the performance reaches a saturation point for a given bottleneck size, it is possible to renew its dynamic by increasing M only. This result is consistent since the number of output channels (M), just before the bottleneck, corresponds to the number of features that must be compressed and transmitted. It therefore makes sense to produce more features at high rates for a better reconstruction of the compressed images. Interestingly, this figure

allows establishing in advance the convolution layer dimensions (N and M) for a given rate range, taking into account a complexity concern.

4.3.3.3 Summary

As an intermediary conclusion, for either low or high bit rates, a drastic reduction of N starting from the reference architecture [Ballé et al. (2018)], does not decrease significantly the performance, both in MSE and in MS-SSIM, while it leads to a complexity decrease of more than 70%. These results are interesting since it was mentioned in [Ballé et al. (2017), Ballé et al. (2018), and Ballé (2018)] that structures of reduced complexity would not be able to perform well at high rates.

4.3.4 Impact of the filter kernel support in the main autoencoder

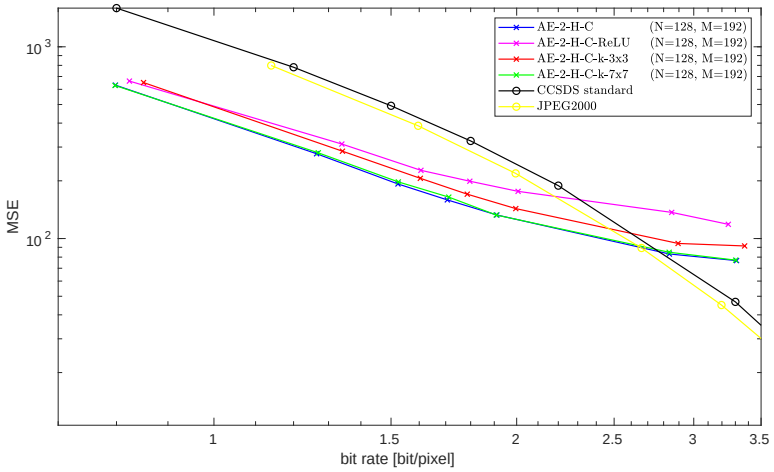
The original framework `AE-2-H-C (N=128, M=192)`[Ballé et al. (2018)] is also tested when replacing the 5×5 filters composing the convolutional layers of the main autoencoder by 3×3 and 7×7 filters. The architectures considered in this part share the same entropy model obtained through the same auxiliary autoencoder in terms of number of filters and kernel supports. According to figure 4.17, a kernel support reduction from 5×5 to 3×3 leads to a performance decrease. This result is expected in the sense that filters with a smaller kernel support correspond to a reduced approximation capability. On the other hand, a kernel support increase from 5×5 to 7×7 does not lead to a significant performance improvement. This result indicates that the approximation capability obtained with a kernel support 5×5 is sufficient.

4.3.5 Impact of the GDN/IGDN replacement in the main autoencoder

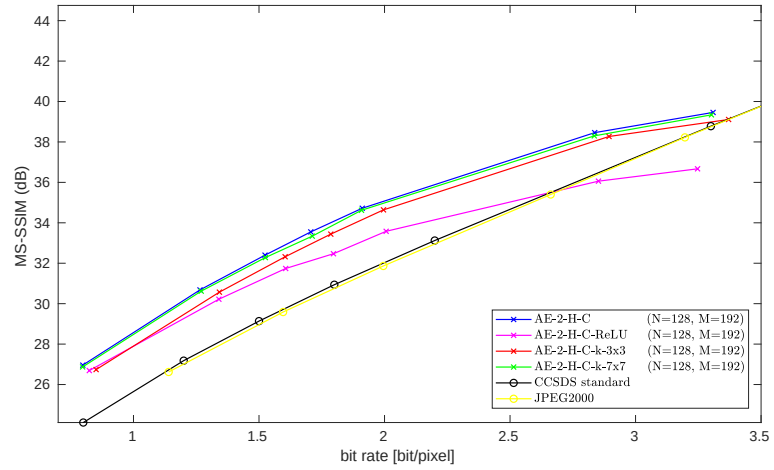
The original architecture of `AE-2-H-C (N=128, M=192)` [Ballé et al. (2018)], involving GDN/IGDN non-linearities, is compared with the architecture obtained after a full ReLU replacement, except for the last layer of the synthesis part. Indeed, this layer involves a sigmoid activation function for constraining the output interval mapping between 0 and 1 before quantization. Figure 4.17 shows the rate-distortion averaged over the validation dataset in terms of both MSE and MS-SSIM (dB).

As claimed in [Ballé (2018)], GDN/IGDN perform better than ReLU for all rates and especially at high rates. Thus, although GDN/IGDN increase the number of parameters to be learned and stored, as well as the number of FLOPp, on one side this increase

represents a small percentage of the overall structure with respect to conventional non-linearities [Ballé (2018)]. On the other side, GDN/IGDN lead to a dramatic performance boost. In view of these considerations, the complexity reduction in this chapter does not target the GDN/IGDN. However, their replacement by simpler activation functions can be envisioned in future work to take into account onboard hardware requirements.



(**a**) Log–log scale. Distortion measure: MSE.



(**b**) Distortion measure: MS-SSIM (dB).

Figure 4.17: Impact of the GDN/IGDN replacement and of the filter kernel support on performance in terms of MSE and MS-SSIM (dB) (derived as $-10\log_{10}(1-\text{MS-SSIM})$).

4.3.6 Impact of the number of layer reduction in the main autoencoder

The original framework `AE-2-H-C (N=128, M=192)` [Ballé et al. (2018)] is also compared to the framework obtained when the number of layers composing the main autoencoder (4 layers for the encoder and 4 layers for the decoder) is reduced to 3 layers and 2 layers successively. Note that all the simplified architectures considered in this part share the same entropy model obtained through the same auxiliary autoencoder in terms of the number of filters and kernel supports. Surprisingly, according to figure 4.18, reducing the number of layers leads to a performance improvement for high rates. When reducing to 2 layers, there is a slight deterioration in performance. However, the two-layer architecture overcame the saturation level achieved by the original architecture for higher bit rates. This result tends to be misleading since suppressing an intermediate layer does not necessarily imply a reduction in complexity, as we will see later in detail. The complexity of a convolutional layer is not directly proportional to the number of its parameters, according to the complexity calculation detailed in the Appendix A.1. Its complexity is also associated with the dimensions of its input, downsampling factors (resp. upsampling), and consequently, the dimensions of its output.

To better understand the attained performances, we now turn to the complexity of the considered architectures, which is shown in Table 4.4. We observe that the complexity in FLOPp is increased by simply reducing the number of layers. Note that this trend cannot be generalized since the architecture complexity also depends on other configurations. Although this architectural modification reduces the number of parameters, the latent representation ($\mathbf{y}$) has a larger spatial dimension since downsampling operations have been suppressed. According to Equation A.2, the resulting complexity for the filtering operation is expressed as a function of the spatial dimensions of the output intermediary representation ($s_{out} \times s_{out}$). Consequently, a larger spatial dimension implies higher complexity, especially at the layer preceding the bottleneck, which contains $M = 192$ filters that also operate in a greater spatial extent ($s_{in} \times s_{in}$). One way to compensate for this complexity increase would be to increase the downsampling (resp. upsampling) factor from 2 to 4, for example. However, a higher downsampling factor can imply abrupt losses in the spatial dimension. Following this logic, this becomes even more explicit when we look at the complexity of the secondary autoencoder that implements the hyperprior in Table 4.5. Despite maintaining the same number of parameters in this part, the complexity in FLOPp is considerably increased since it acts in a latent representation with a larger spatial dimension (multiplied by 2 and multiplied by 4 for one-layer and two-layer reduction, respectively).

Table 4.4: Comparative complexity of the considered architectures - Case of reducing the number of layers.

Method	Parameters	FLOPp	Relative to FLOPp
`AE-2-H-C (N=128, M=192)-4-layers (original)`	5055105	1.9115×10^5	1.00
`AE-2-H-C (N=64, M=192) -4-layers`	1683969	5.2264×10^4	0.27
`AE-2-H-C (N=128, M=192)-3-layers`	4202625	2.0732×10^5	1.08
`AE-2-H-C (N=64, M=192) -3-layers`	1470721	6.6605×10^4	0.34
`AE-2-H-C (N=128, M=192)-2-layers`	3350145	2.7201×10^5	1.42

Table 4.5: Comparative complexity of the considered architectures on the hyperprior part-Case of reducing the number of layers.

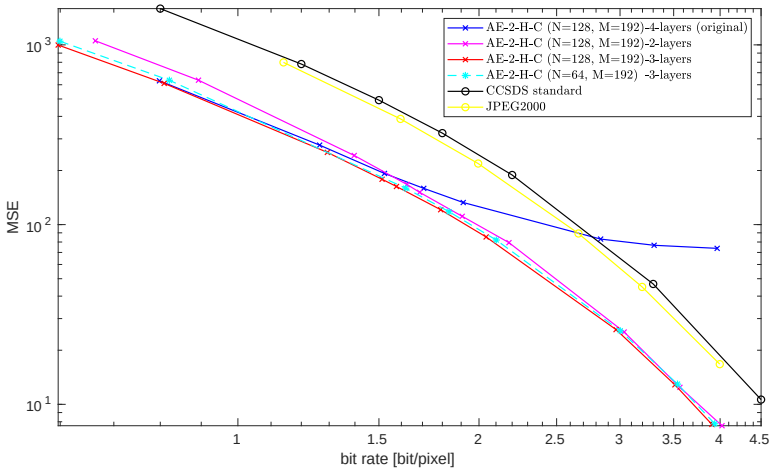
Method	Parameters	FLOPp	Relative to FLOPp
`AE-2-H-C (N=128, M=192)-4-layers (original)`	2081472	4230	1.00
`AE-2-H-C (N=64, M=192) -3-layers`	2081472	16920	4.00
`AE-2-H-C (N=64, M=192) -2-layers`	2081472	67680	16.00

When reducing the number of layers composing the main autoencoder (the symmetric pair of layers part of the encoder and decoder), each reduction represents one less downsampling (resp. upsampling) operation adopted. Each downsampling operation (resp. upsampling) implies the discarding (resp. interpolation) of information present in the spatial dimensions of the intermediary tensors along the whole autoencoder. Note that this process may be one of the sources of visual quality degradations in the resulting compressed image. In this case, there is a possible smaller loss of information in encoding and decoding processes.

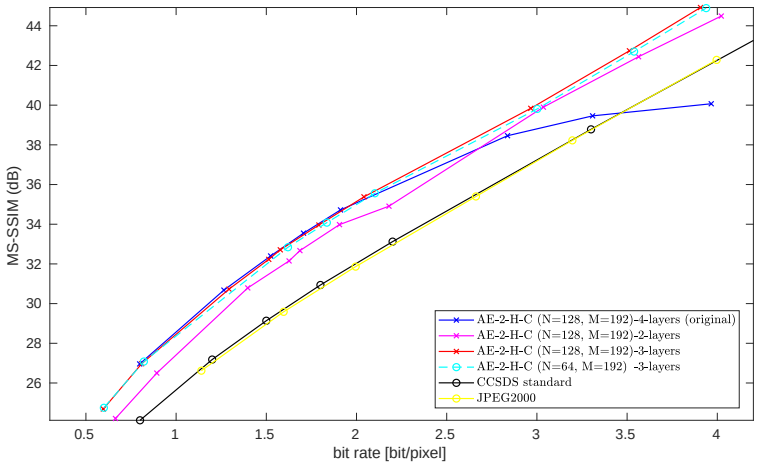
To further explore the number of layer reduction, experiments were also carried out with a single architecture that brings together the number of layer reduction (one layer reduction) and the number of filter reduction ($N = 64$ instead of $N = 192$). This architecture maintained the same high performance compared to its counterpart with more layers and filters and even for higher bit rates. Concerning the complexity, we observed that this architecture presents a lower complexity in terms of FLOPp with respect to the two-layer one. However, it is still greater to the one of the architecture featuring the number of filters reduction only (`AE-2-H-C (N=64, M=192)`).

Reducing the number of layers allows high compression performance at high rates without using a wider bottleneck, as seen in previous experiments. Thus, modifying the number of layers can be a different design choice to attain better performance at higher rates. However, this does not necessarily mean that reducing the number of layers is better than increasing the bottleneck size. Indeed, reducing the number of layers implies a complexity increase in FLOPp. However, reducing the number of layers allows reducing the number of parameters,

which may be relevant in a limited storage context.



(**a**) Log–log scale. Distortion measure: MSE.



(**b**) Distortion measure: MS-SSIM (dB).

Figure 4.18: Impact of the number of layer reduction on performance in terms of MSE and MS-SSIM (dB) (derived as $-10\log_{10}(1-\text{MS-SSIM})$).

4.3.7 Impact of the entropy model simplification

To evaluate the impact of the entropy model simplification, the autoencoder implementing the hyperprior in the reference compression framework [Ballé et al. (2018)] is replaced by the simplified entropy model presented in 4.2.2, according to figure 4.10.

Table 4.6: Reduction of the encoder complexity induced by the Laplacian entropy model on the coding part - Case of rates up to 2 bits/pixel.

Method	Parameters	FLOPp	Relative
`AE-2-H-C (N=64, M=192)`	1157696	1.25×10^4	1
`AE-2-L-C (N=64, M=192)`	526464	1.09×10^4	0.87

At Low Rates For rates up to 2 bits/pixel, the architectures `AE-2-L-C (N=128, M=192)` (with the Laplacian entropy model) and `AE-2-L-C (N=64, M=192)` (combining the reduction of the number of filters to $N = 64$ and the Laplacian entropy model) are compared with the non parametric reference method `AE-1-NP-C (N=192)` [Ballé et al. (2017)], with the hyperprior reference method `AE-2-H-C (N=128, M=192)` [Ballé et al. (2018)], with its version after reduction of the number of filters `AE-2-H-C (N=64, M=192)`, with the architecture denoted as `AE-2-NP-C (N=128, M=192)` (combining the main auto-encoder in [Ballé et al. (2018)] and the non-parametric entropy model in [Ballé et al. (2017)]) and its version after reduction of the number of filters `AE-2-NP-C (N=64, M=192)`. Table 4.6 shows that the coding part complexity of `AE-2-L-C (N=64, M=192)` is 13% lower than the one of `AE-2-H-C (N=64, M=192)`.

Figure 4.14 shows the rate-distortion averaged over the test dataset for the trained architectures for both MSE and MS-SSIM quality measures. Recall that the architectures were trained for MSE only. The proposed simplified entropy model (`AE-2-L-C (N=64, M=192)`) achieves an intermediate performance between the hyperprior model (`AE-2-H-C (N=64, M=192)`) and the non parametric model (`AE-2-NP-C (N=64, M=192)`). Obviously, due to the entropy model simplification, `AE-2-L-C (N=64, M=192)` underperforms the more general and thus more complex `AE-2-H-C (N=64, M=192)` model. However, the proposed entropy model, even if simpler, preserves the adaptability to the input image, unlike the models `AE-2-NP-C (N=128, M=192)` and `AE-1-NP-C (N=192)` [Ballé et al. (2017)]. Please note that the simplified Laplacian entropy model perform close to the hyperprior model at relatively high rates. One possible explanation for this behaviour can be the high amount of side information required by the hyperprior model [Ballé et al. (2018)] for these rates [Y. Hu et al. (2020)].

At High Rates For high rates (above 2 bits/pixel), the proposed architectures `AE-2-L-C (N=192, M=320)` (with the Laplacian entropy model) and `AE-2-L-C (N=64, M=320)` (combining the reduction of the number of filters to $N = 64$ and the Laplacian entropy model) are compared with the non parametric reference architecture `AE-1-NP-C (N=M=256)` [Ballé

Table 4.7: Reduction of the encoder complexity induced by the Laplacian entropy model on the coding part - Case of rates above 2 bits/pixel.

Method	Parameters	FLOPp	Relative
`AE-2-H-C (N=64, M=320)`	1715008	1.3979×10^4	1
`AE-2-L-C (N=64, M=320)`	731392	1.1787×10^4	0.8432

et al. (2017)], with the hyperprior reference architecture `AE-2-H-C (N=192, M=320)` [Ballé et al. (2018)], with its version after reduction of the number of filters `AE-2-H-C (N=64, M=320)`, with the architecture denoted as `AE-2-NP-C (N=192, M=320)` (combining the main auto-encoder in [Ballé et al. (2018)] and the non-parametric entropy model in [Ballé et al. (2017)]) and its version after reduction of the number of filters `AE-2-NP-C (N=64, M=320)`. Figure 4.15 displays the rate-distortion averaged over the validation dataset for the trained architectures in terms of MSE. The proposed Laplacian entropy method `AE-2-L-C (N=64, M=320)` achieves an intermediate performance between the hyperprior model (`AE-2-H-C (N=64, M=320)`) and the non parametric model `AE-2-NP-C (N=64, M=320)`, similarly to the models targeting lower rates in figure 4.14. Table 4.7 shows that the coding part complexity of `AE-2-L-C (N=64, M=320)` is around 16% lower than the one of `AE-2-H-C (N=64, M=320)`.

4.3.7.1 Summary

For either low or high bit rates, the proposed entropy model simplification leads to intermediary performance when compared to the non parametric and the hyperprior entropy models with the reference architectures [Ballé et al. (2017) and Ballé et al. (2018)], both in MSE and in MS-SSIM. Moreover, it leads to a coding part complexity decrease of more than 10% with respect to [Ballé et al. (2018)].

4.3.8 Discussion about complexity

According to the previous performance analysis, the computational complexity of the proposed simplified architecture combining the number of filters reduction and the Laplacian entropy model is significantly lower than the one of the reference learned architecture featuring the hyperprior model [Ballé et al. (2018)]. However, around 10 kFLOPp, the attained complexity is at least 2 orders of magnitude higher than the ones of the CCSDS and JPEG2000 [Marcellin and Taubman (2002)] standards. Indeed, the complexity of CCSDS 122.0 is around 140 FLOPp (without optimizations), or 70 MAC (Multiplication Accumulation). The JPEG2000 is 2 to 3 times more complex depending on the optimizations. Note,

however, that the CCSDS 122.0 dates back to 2008 when onboard technologies were limited to radiation-hardened (Rad-Hard) components dedicated to space, with the objectives of 1 Msample/s/W (as specified in the CCSDS 122.0 green book [G. Book (2015)]), to process around 50 Mpixels/s. Space technologies, currently developed for the next generation of CNES Earth observation satellites, rather target 5–10 Msample/s/W. Nowadays, the use of commercial off-the-shelf (COTS) components or of dedicated hardware accelerators is envisioned: based on a thinner silicon technology node, they allow higher processing frequencies with consistently lower consumption. For instance, the Movidius Myriade 2 announces 1 TFLOPS/W. The 10 kFLOPS of the current network would lead to 100 Mpixels/s/W on this component. Therefore, the order of magnitude of the proposed simplified architecture is not incompatible with an embedded implementation, taking into account the technological leap from the component point of view. Consequently, the complexity increase with respect to the CCSDS one, which we limited as far as possible, is expected to be affordable after computation device upgrading. Please note that, in addition, manufacturers of components dedicated to neural networks provide software suites (for example Xilinx) to optimize the portings. Finally, before onboard implementation, a network compression (including pruning, quantization, or tensor decomposition for instance) can be envisioned. However, this is out of the scope of this thesis.

4.4 Conclusions

This chapter proposed different solutions to adapt the reference learned image compression architectures [Ballé et al. (2017) and Ballé et al. (2018)] to onboard satellite image compression, taking into account their computational complexity. We first reduced the number of filters composing the convolutional layers of the main and auxiliary autoencoders, applying a special treatment to the bottleneck. The impact of the bottleneck size, under a drastic reduction of the overall number of filters, was investigated. This study allowed identifying the lowest global number of filters for each rate. For the sake of completeness, we also called into question the other design options of the reference architectures, and especially the parametric activation functions. We also reduced the number of layers composing the encoder (resp. decoder) to understand its impact on the performance. Counterintuitively, reducing the number of layers increases complexity. The latent representation has larger spatial dimensions, which results in more complex operations in the bottleneck part. Second, in order to simplify the entropy model, we also performed a statistical analysis of the learned representation. This analysis showed that most features follow a Laplacian distribu-

tion. We thus proposed a simplified parametric entropy model, involving a single parameter to be estimated. To preserve the adaptivity and thus the performance, this parameter is estimated in the operational phase for each feature of the input image. This entropy model, although far simpler than non-parametric or hyperprior models, brings comparable performance. In a nutshell, by combining the reduction of the global number of filters, and the simplification of the entropy model, we developed a reduced-complexity compression architecture for satellite images that outperforms the CCSDS 122.0-B [B. Book (2017)], in terms of rate-distortion trade-off, while maintaining a competitive performance for medium to high rates in comparison with the reference learned image compression models [Ballé et al. (2017) and Ballé et al. (2018)]. Thereupon, while more complex than traditional CCSDS 122.0 and JPEG 2000 standards, the proposed solutions offer a good compromise between complexity and performance. Thus, we can recommend their use, subject to the availability of suitable onboard devices.

Chapter 5

Satellite image compression and denoising with neural networks

This chapter is adapted from [Alves de Oliveira et al. (2022)]

Contents

5.1 Selected methods from state-of-the-art . . . 94
5.1.1 End-to-end trainable autoencoder for image compression . . . 94
5.1.2 Denoising with BRDNet . . . 95
5.2 Combining compression and denoising . . . 97
5.2.1 Joint compression and denoising . . . 97
5.2.2 Sequential compression and denoising . . . 98
5.2.3 Denoising as a post-processing after joint compression and denoising . 98
5.3 Performance analysis . . . 99
5.3.1 Implementation setup . . . 99
5.3.2 Compression-only performance of the joint compression and denoising method . . . 100
5.3.3 Compression and denoising performance . . . 101
5.4 Conclusions . . . 108

In this chapter, we take advantage of CNNs to address satellite image compression and denoising. We aim to outperform the current satellite imaging system [Delvit et al. (2019)], described in chapter 2, both in compression and denoising without manual parameter set-

ting, without a priori knowledge on the noise statistical model and without tricky procedures (like VST or instrumental noise restitution) to fit the noise to a given model. Besides, we aim to propose possible onboard denoising whereas it is currently mainly performed on ground as a post-processing. After a short review of state-of-the-art architectures, we first propose an onboard joint compression and denoising approach with a single neural architecture based on [Ballé et al. (2018)]. Second, we propose a modular neural architecture, that performs sequentially onboard compression based on [Ballé et al. (2018)] and on ground denoising based on [Tian et al. (2020)]. This sequential approach allows to lighten the onboard computational load if required, especially since it is compatible with every complexity reduction proposed in chapter 4.

5.1 Selected methods from state-of-the-art

5.1.1 End-to-end trainable autoencoder for image compression

In the context of image compression, autoencoders are used to learn a representation with low entropy after quantization [Ballé et al. (2017) and Ballé et al. (2018)]. In this chapter, we focus on the reference architecture [Ballé et al. (2018)] displayed on figure 5.1. In the main autoencoder (left part of figure 5.1), an analysis transform (G_a) is applied to the input image $\mathbf{I}$ to produce a learned representation $\mathbf{y} = G_a(\mathbf{I})$ at the bottleneck. Then, a synthesis transform (G_s) is applied to the quantized representation $\hat{\mathbf{y}}$ to reconstruct the input image $\hat{\mathbf{I}} = G_s(\hat{\mathbf{y}})$, as detailed in chapter 3. G_a and G_s are obtained through multiple convolutional layers composed of filters followed by non-linear activation functions. N defines the number of filters in each layer, except in the last layer before the bottleneck composed by M filters. Indeed, $M > N$ has to be maintained, following the so-called wide bottleneck strategy [Ballé et al. (2018)]. GDN (resp. IGDN) activation functions are used to implement a local adaptive normalization. The resulting learned representation is thus multi-channel and non-linear.

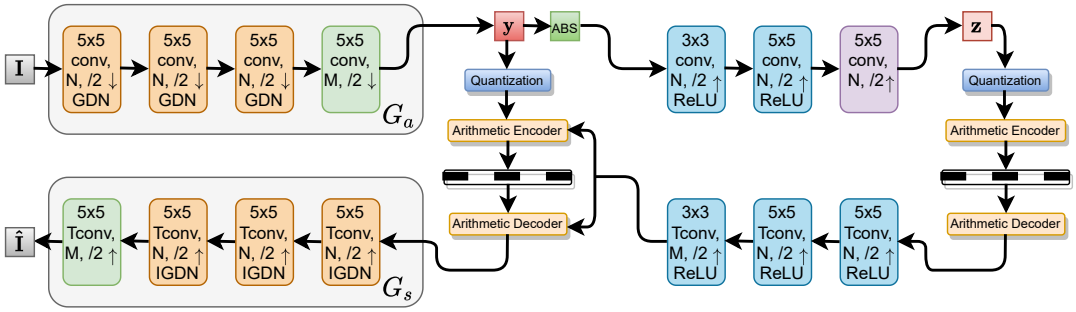


Figure 5.1: Architecture of the variational autoencoder [Ballé et al. (2018)].

A side autoencoder (right part of figure 5.1) is used to estimate the hyper-parameters of each input image representation distribution even during the operational phase. This model takes into account possible spatial dependency in each input image representation. Chapter 4 showed that a simpler fully-factorized Laplacian model can be successfully used to compress satellite images. This simplified model exploits a statistical analysis of the learned representation for satellite images presented in chapter 4. The computationally expensive auxiliary autoencoder (right part of figure 5.1) in [Ballé et al. (2018)] was thus replaced by the simple estimation of the scale parameter related to the Laplacian distribution.

5.1.2 Denoising with BRDNet

Deep CNN-based methods have become very successful in image denoising, as introduced in chapter 3. From the beginning, [K. Zhang et al. (2017a)] proposed a discriminative denoising model learning named DnCNN that adopts RL and BN to improve the denoising performance.

More recently, [Tian et al. (2020)] proposed a novel deep learning framework designated batch-renormalization denoising network (BRDNet), detailed in chapter 3, whose main characteristics are recalled subsequently. BRDNet uses batch renormalization (BRN) [Ioffe (2017)] to deal with small mini-batch convergence issues in BN and adopts RL similarly to [K. Zhang et al. (2017a)]. BRN is applied over a mini-batch according to the Algorithm 4 presented in 3.5.1. BRDNet also blends two parallel sub-networks to obtain more relevant features for improving the denoising performance [Szegedy et al. (2015)]. Furthermore, the idea was to increase the width of the network rather than the depth (e.g., number of layers) and thus to avoid vanishing or exploding gradients issues during training, which mainly affect deeper networks. The proposed network is shown in figure 5.2.

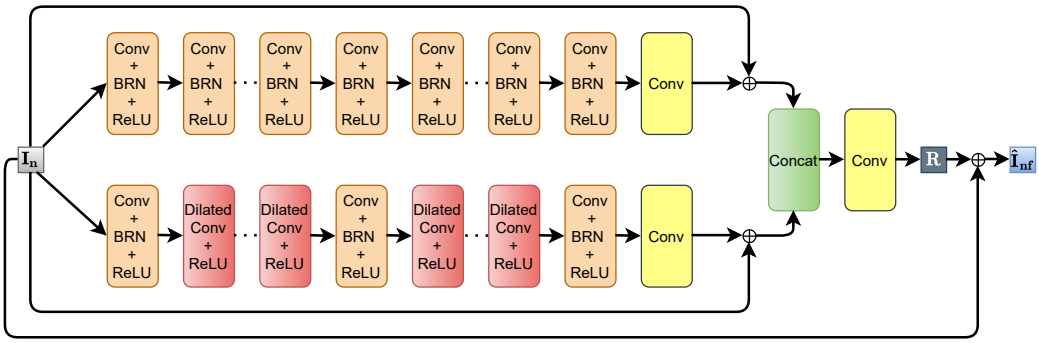


Figure 5.2: Architecture of the BRDNet network [Tian et al. (2020)].

$\mathbf{I_n}$ and $\hat{\mathbf{I}}_{\mathbf{nf}}$ denote the noisy image and the denoised image, respectively. We observe that the upper network mainly involves RL and BRN, while the lower network features RL, BRN, and dilated convolutions [F. Yu and Koltun (2015)]. The authors claim that dilated convolutions enable the extraction of more context information with relatively lower complexity when compared to conventional CNNs, mostly because they attain the same receptive field with few layers and parameters. In [Tian et al. (2020)], experiments demonstrated that the proposed BRDNet outperforms state-of-the-art deep learning denoising methods, e.g., DnCNN [K. Zhang et al. (2017a)], but also the fast and flexible denoising network (FFDNet) [K. Zhang et al. (2018)] and the image-restoration CNN (IRCNN) [K. Zhang et al. (2017b)].

Contrarily to autoencoders, BRDNet uses residual learning to predict a residual image $f(\mathbf{I_n}) \backsimeq \mathbf{I_n} - \mathbf{I_{nf}}$, where $f(\mathbf{I_n})$ denotes the noise prediction by the DNN. In figure 5.2, BRDNet takes $\mathbf{I_n}$ as input and the predicted denoised image ($\hat{\mathbf{I}}_{\mathbf{nf}}$) as output. The BRDNet parameters (filter weights) are learned by optimizing a loss function taking account the distortion between $\mathbf{I_{nf}}$ and $\hat{\mathbf{I}}_{\mathbf{nf}}$. The distortion criterion is defined by:

$$J(\theta) = \sum_{\mathbf{I_{nf}}, \hat{\mathbf{I}}_{\mathbf{nf}} \in dataset} D(\mathbf{I_{nf}}, \hat{\mathbf{I}}_{\mathbf{nf}}). \tag{5.1}$$

The loss function defined in Equation (5.1) is minimized through gradient descent with back-propagation [Bengio (2009)] on a representative data set composed of pairs of noisy ($\mathbf{I_n}$) and noise-free ($\mathbf{I_{nf}}$) images.

5.2 Combining compression and denoising

Currently, as presented in chapter 2, the compression is performed onboard the satellite whereas the denoising is performed on ground, because of its prohibitive computational cost [Delvit et al. (2019)]. However, the evolution of satellite computing capacities enables onboard denoising to be reasonably envisioned [Delvit et al. (2019)]. This chapter addresses data-driven approaches for satellite image compression and denoising, possibly both performed onboard. The aim is to attain high performance while dispensing from manual parameter setting, a priori knowledge of the noise model, or tricky intermediary steps (like VST or instrumental noise restitution). The first proposed approach takes advantage of the compression-dedicated architecture, proposed in [Ballé et al. (2018)] and adapted to satellite in chapter 4, to jointly perform compression and denoising onboard. The second proposed approach sequentially combines a compression-dedicated architecture and a denoising-dedicated one. Thanks to its modular structure, this sequential approach allows to choose the best architectures for compression [Ballé et al. (2018)] and for denoising [Tian et al. (2020)] respectively. Moreover, this approach facilitates consideration of the onboard hardware constraints since all the complexity-reductions proposed in chapter 4 can be applied to the compression-dedicated architecture in this case. The question of computational complexity is less crucial for the on ground denoising-dedicated architecture. Finally, note that whether the joint or the sequential approach are expected to suppress compression artifacts together with instrumental noise [Delvit et al. (2019)]. The transmission is assumed not to introduce additional degradations [Carlavan et al. (2012)]. Realistic simulation of satellite images provide both $\mathbf{I_n}$ and $\mathbf{I_{nf}}$ for the same scene, which makes architecture training and validation possible.

5.2.1 Joint compression and denoising

In this part, we consider the compression-dedicated architecture displayed in figure 5.1, with the number of filters reduction proposed in chapter 4. The input is the noisy acquired image ($\mathbf{I_n}$). In order to jointly perform denoising and compression, the architecture parameters are learned through the optimization of a specific loss function (different from the one used in chapter 4): the rate $R(\hat{\mathbf{y}})$ is the same but the distortion $D(\mathbf{I_{nf}}, G_s(G_a(\mathbf{I_n})))$ now measures the similarity between the reconstructed image $\hat{\mathbf{I}}_{\mathbf{nf}} = G_s(G_a(\mathbf{I_n}))$ and the reference noise-free image $\mathbf{I_{nf}}$ (instead of the input image $\mathbf{I_n}$). The reconstructed image is thus expected to be denoised. As joint compression and denoising may require a greater approximation capacity, we first consider the more sophisticated hyperprior entropy model [Ballé et al.

(2018)] (right part of figure 5.1) that delivers the best performance/modeling for compression. However, some supplementary tests were also performed for the simplified Laplacian entropy model proposed in chapter 4.

5.2.2 Sequential compression and denoising

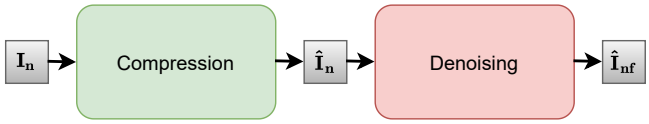


Figure 5.3: Sequential compression and denoising scheme.

The sequential compression and denoising approach exploits two architectures: the compression-dedicated one detailed in 5.1.1 (with two versions denoted respectively `AE-2-H-C` when featuring the hyperprior [Ballé et al. (2018)] and `AE-2-L-C` when featuring the Laplacian entropy model [Alves de Oliveira et al. (2021)]), and the denoising-dedicated architecture BRDNet, detailed in 3.5.4 [Tian et al. (2020)], from which we expect also compression artifact reduction.

5.2.3 Denoising as a post-processing after joint compression and denoising

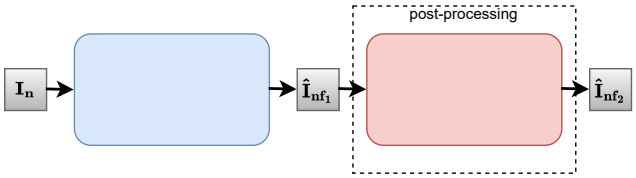


Figure 5.4: Joint compression and denoising followed by denoising as a post-processing scheme.

When combining the benefits of both joint and sequential approaches, BRDNet can also be employed as a post-processing module to joint compression and denoising. The resulting approach is expected to suppress artifacts arising from the previous compression as well as remaining noise after denoising. This scheme is illustrated in figure 5.4. This approach benefits from two separate denoising stages with one deployed on the ground. The on

ground architecture is trained on the images that have already been jointly compressed and denoised.

5.3 Performance analysis

To assess the relevance of the learned proposed compression and denoising approaches, experiments were conducted using TensorFlow. The reference CNES imaging system of [Delvit et al. (2019)] is considered as our baseline for compression and denoising performance.

5.3.1 Implementation setup

5.3.1.1 Datasets

Our training (resp. test) dataset is composed of 112 (resp. 16) pairs of noisy ($\mathbf{I_n}$) and noise-free ($\mathbf{I_{nf}}$) 12-bit simulated Pléiades panchromatic images (of size 586×586) provided by the CNES, covering various landscapes. The instrumental noise is simulated according to [Delvit et al. (2019)]. For training our compression architectures, we use patches randomly cropped from the noisy images ($\mathbf{I_n}$) composing our training dataset. For training our joint compression and denoising architectures, we use pairs of patches randomly cropped from image pairs $\mathbf{I_n}/\mathbf{I_{nf}}$ composing our training dataset. In this case, $\mathbf{I_n}$ is the input of the network, and $\mathbf{I_{nf}}$ is the reference for distortion computation. For the denoising-only architectures, the training has the same configurations except that the input images are the noisy compressed images $\hat{\mathbf{I}}_\mathbf{n}$ (resp. $\hat{\mathbf{I}}_\mathbf{nf}$) when performed after compression-only (resp. after joint compression and denoising). Analogously and following the same logic, full images (resp. pairs of images) from the test dataset are used to assess the performance.

5.3.1.2 Reference methods

For the joint compression and denoising and denoising-only architectures, the autoencoder architecture proposed in [Ballé et al. (2018)] is considered. The number of filters was set to $N = 64$ and $M = 320$ following the architecture simplifications in terms of number of filters presented in chapter 4. The considered architectures are designated as:

- `AE-2-H-CD`: joint learned compression and denoising architecture featuring the hyperprior.

- `AE-2-H-C`: learned compression architecture featuring the hyperprior presented in chapter 4 [Alves de Oliveira et al. (2021)].
- `AE-2-L-C`: learned compression architecture featuring the simplified Laplacian entropy model presented in chapter 4 [Alves de Oliveira et al. (2021)].
- `CNES-C`: CNES compression algorithm [Thiebaut et al. (2016)].

For the sequential denoising, the BRDNet is trained following the architecture description in [Tian et al. (2020)]. Each convolutional layer is composed of 64 filters with kernel size 3×3. Note that both joint and sequential compression and denoising approaches do not require the instrumental noise restitution and the variance stabilizing transform (VST) used in the reference CNES imaging system [Anscombe (1948)].

5.3.1.3 Training parameters

For learning the compression-dedicated and the joint compression and denoising architectures, the training patch size was set to 256×256, the batch size was set to 8 and up to 2 million iterations were performed. For learning the denoising-dedicated architecture, the training patch size was set to 50×50, the batch size was set to 20 and up to 500000 iterations were performed [Tian et al. (2020)]. MSE was used as the distortion metric for training in all cases.

5.3.2 Compression-only performance of the joint compression and denoising method

In this part, the different methods are tested for input images with noise ($\mathbf{I_n}$), and their output images are compared with the same noisy images ($\mathbf{I_n}$) for the compression performance assessment. The proposed joint compression and denoising architecture `AE-2-H-CD` is compared with its corresponding compression-only architecture `AE-2-H-C`, trained to minimize $||\hat{\mathbf{I}}_\mathbf{n} - \mathbf{I_n}||_2^2$, and the CNES compression method [Thiebaut et al. (2016)].

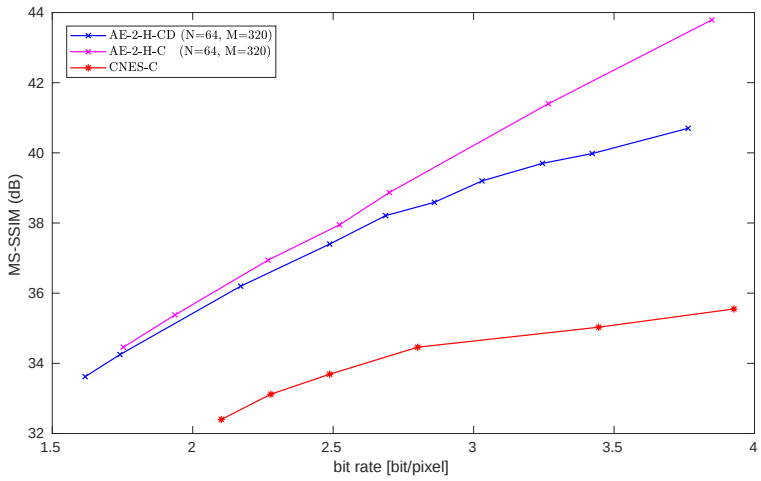


Figure 5.5: Curves for MS-SSIM (dB) between the output image and the noisy image ($\mathbf{I_n}$).

Figure 5.5 shows the performance averaged over the test dataset in terms of bit rate and MS-SSIM (dB) for the considered methods. In figure 5.5, it can be seen that `AE-2-H-CD` compresses more than denoises since the distortion arising from compression is expected to be more outstanding than the instrumental noise in lower bit rates. However, for higher bit rates, there is a greater tendency of this architecture to suppress noise compared to the compression-only architecture `AE-2-H-C`, which has no vocation to denoising. In this way, the joint compression and denoising architecture can implicitly take into account the different portions of distortion arising from compression and noise present in the input images ($\mathbf{I_n}$) for higher bit rates.

5.3.3 Compression and denoising performance

The considered joint compression and denoising framework `AE-2-H-CD` and the sequential compression and denoising approach for `AE-2-H-C+BRDNet` and `AE-2-L-C+BRDNet` are compared with the CNES compression method [Thiebaut et al. (2016)] in conjunction with the denoising algorithm NL-Bayes parametrized by CNES [Delvit et al. (2019)]. The two different compression and denoising strategies are tested for input images with noise ($\mathbf{I_n}$), but their output images are compared with the reference noise-free images ($\mathbf{I_{nf}}$) for the quality assessment. The compression/denoising performance for the same corresponding compression-only methods is also displayed. The performance curve for the `CNES-C` method in conjunction with `BRDNet` is also shown.

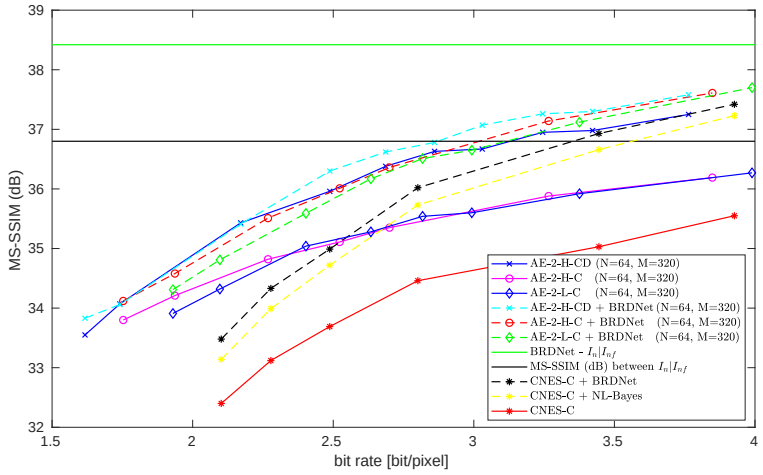


Figure 5.6: Curves for MS-SSIM (dB) between the output image and the noise-free image ($\mathbf{I_{nf}}$).

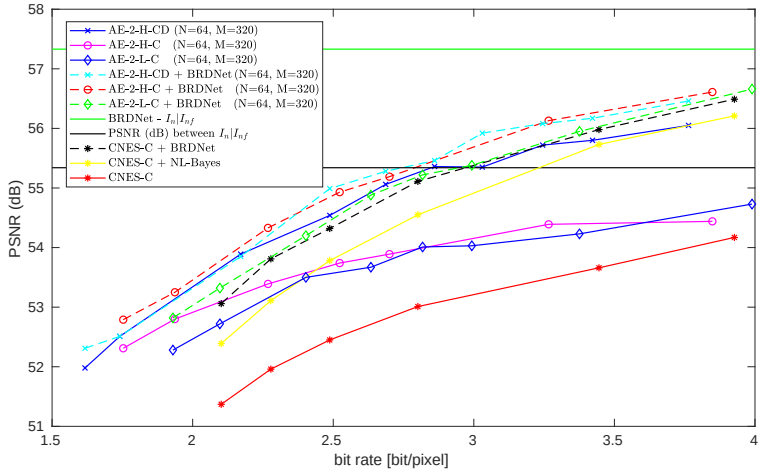


Figure 5.7: Curves for PSNR (dB) between the output image and the noise-free image ($\mathbf{I_{nf}}$).

Figures 5.6 and 5.7 show the performance averaged over the test dataset in terms of bit rate, MS-SSIM (dB) and PSNR (dB) for the considered methods. Table 5.1 compares the complexity of the autoencoder-based considered methods `AE-2-H-CD`, `AE-2-H-C` and `AE-2-L-C` to the `BRDNet`. The complexity of the `BRDNet` is about 21× higher in terms of FLOPp with respect to the considered joint compression and denoising `AE-2-H-CD`. Concerning the

Table 5.1: Comparative complexity of the considered architectures

Method	Parameters	FLOPp
`AE-2-H-CD / AE-2-H-C (N=64, M=320)`	1683969	5.6966×10^4
`AE-2-L-C (N=64, M=320)` [chapter 4]	1052737	5.4774×10^4
`BRDNet` [Tian et al. (2020)]	1186816	1.1868×10^6

`BRDNet`, we will focus on time complexity. It is indeed the more critical criterion on ground since it determines the pipeline throughput. BRDNet denoises in average 2.192×10^6 pixels per second on an NVIDIA Tesla V100 GPU with 32 GB onboard memory, whereas the reference CNES-customized NL-Bayes denoises on average 0.073×10^6 pixel per second on an Intel i7-6700 HQ (2.6-3.5GHz) CPU with 8 GB RAM [Masse et al. (2018)]. Finally BRDNet, benefiting from the GPU massively parallel architecture, denoises approximately 30 times faster than the CNES-customized NL-Bayes.

These experiments showed that the proposed joint compression and denoising model `AE-2-H-CD` outperforms the CNES baseline over the considered bit range between 2 and 3.7 bits/pixel. The advantage of the proposed joint compression and denoising method reduces at higher rates. The CNES compression algorithm in conjunction with `BRDNet` showed better compression and denoising performance than the `CNES+NL-Bayes` baseline. However, it still has a lower performance than the learned joint and sequential compression-and-denoising methods. This result also shows that deep learning denoising can significantly improve a typical satellite imaging system.

The sequential compression and denoising approaches (`AE-2-H-C` or `AE-2-L-C` followed by `BRDNet`) perform similarly and even better than the joint compression and denoising approach (`AE-2-H-CD`), particularly for the highest bit rates. The approach that performs denoising as a post-processing to joint compression and denoising (`AE-2-H-CD` followed by `BRDNet`) performs slightly better for rates between 2.2 and 3.2 bits/pixel. Besides, for this bit rate range, it outperforms all the other approaches, whereas it performs similarly to the sequential approaches at high rates. Finally, note that simply replacing the NL-Bayes algorithm with `BRDNet` architecture leads to a gain in performance without any modification onboard.

5.3.3.1 Subjective image quality analysis



Figure 5.8: Example result on a test image. (a) `CNES-C+NL-Bayes` Delvit et al., 2019. (b) `AE-2-H-C+BRDNet`. (c) `AE-2-H-CD`. (d) Noise-free image ($\mathbf{I_{nf}}$).

Figure 5.8 provides an example of the uncompressed and denoised image, obtained with the CNES baseline and with the proposed approaches for similar rates. Even if all the methods perform satisfactorily, the image obtained with the joint compression and denoising approach `AE-2-H-CD` is visually the closest to the noise-free reference whereas the sequential approach (`AE-2-H-C+BRDNet`) tends to produce a slightly smoothed image. Note that the current CNES pipeline also adds noise in areas where intensity is low, as is not the case for the proposed approaches.

5.3.3.2 Impact of the simplified Laplacian entropy model in the joint compression and denoising

Experiments were also carried out with the end-to-end joint compression and denoising architecture featuring the simplified Laplacian entropy model proposed in chapter 4. The considered model is thus designated as:

- `AE-2-L-CD`: joint learned compression and denoising architecture featuring the simplified Laplacian entropy model [Alves de Oliveira et al. (2021)].

Figure 5.9 shows the performance averaged over the test datasets in terms of bit rate and MS-SSIM (dB) for the considered `AE-2-L-CD` model in comparison with its counterpart `AE-2-H-CD` that features the more complex hyperprior model. We observe that `AE-2-L-CD` didn't exhibit a comparable performance. A statistical analysis of the resulting feature maps for this model revealed that most of them can be well fitted by centered Laplacian distributions in a similar proportion when compared to its equivalent compression-only model as showed in chapter 4.

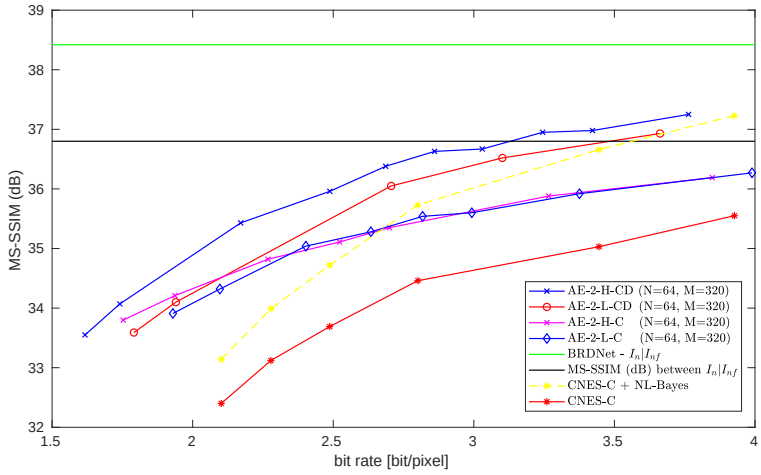


Figure 5.9: Curves for MS-SSIM (dB) between the output image and the noise-free image ($\mathbf{I_{nf}}$).

5.3.3.3 Impact of the number of filters increase in the joint compression and denoising

Experiments were also performed with the end-to-end joint compression and denoising method featuring the hyperprior model with a higher number of filters, analogously to the original compression framework proposed in [Ballé et al. (2017) and Ballé et al. (2018)] with $N = 192$. This model is designated as:

- `AE-2-H-CD (N=192, M=320)`: joint learned compression and denoising architecture featuring the hyperprior model with $N = 192$.

Figure 5.10 shows the performance averaged over the test datasets in terms of bit rate and MS-SSIM (dB) for the considered `AE-2-H-CD (N=192, M=320)` model in comparison with its counterpart `AE-2-H-CD (N=64, M=320)` that features a reduced number of filters. A higher number of filters doesn't result in performance improvements as we observed in the equivalent compression-only models in chapter 4. This result seems to be related to increased difficulty in training this architecture with an increased number of filters for joint compression and denoising.

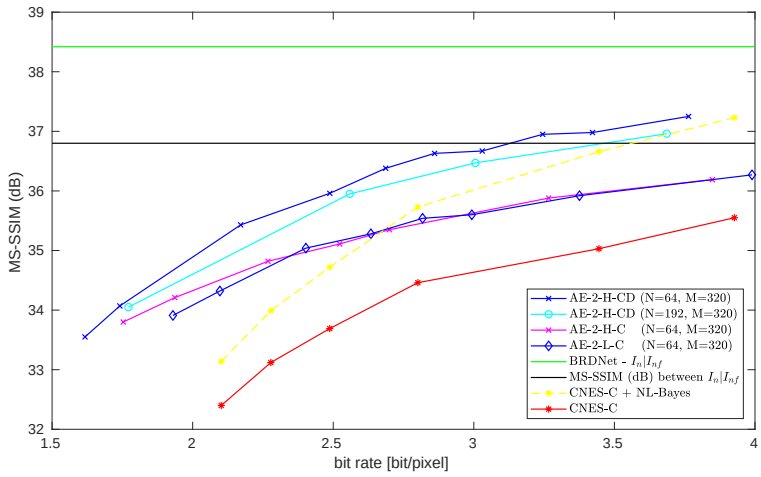


Figure 5.10: Curves for MS-SSIM (dB) between the output image and the noise-free image ($\mathbf{I_{nf}}$).

5.3.3.4 Lower average luminance images

The idea here is to verify how effective the learning-based joint and sequential compression and denoising approaches are effective in a different luminance scenario, e.g., with low average luminance images.

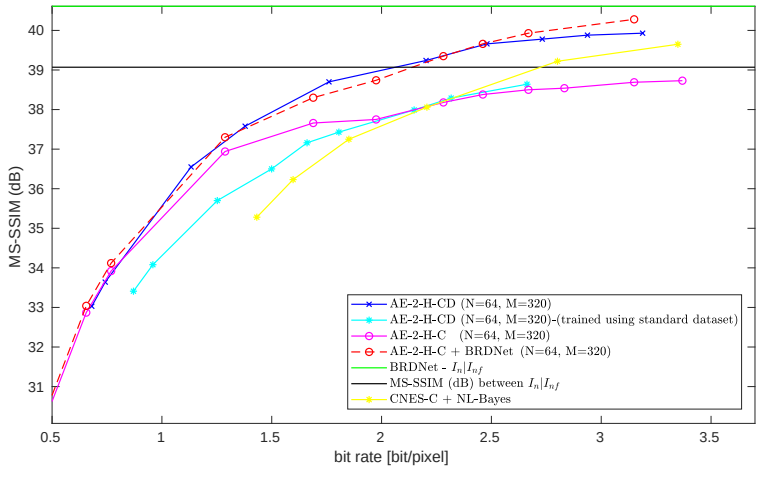


Figure 5.11: Curves for MS-SSIM (dB) between the output image and the noise-free image ($\mathbf{I_{nf}}$).

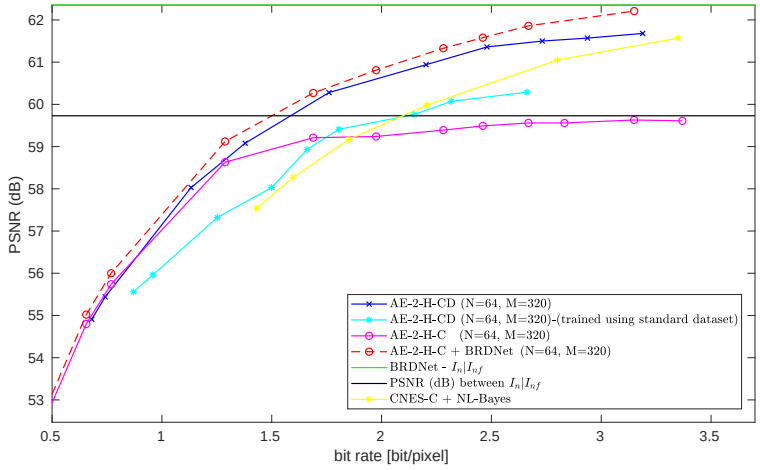


Figure 5.12: Curves for PSNR (dB) between the output image and the noise-free image ($\mathbf{I_{nf}}$).

Figures 5.11 and 5.12 show the performance averaged over the low-luminance images test dataset in terms of bit rate, MS-SSIM (dB) and PSNR (dB) for the considered methods. Analogously to what was observed in figures 5.6 and 5.7, the learned approaches `AE-2-H-CD` and `AE-2-H-C + BRDNet` presented a superior performance compared to the `CNES-C + NL-Bayes` for low luminance average images. The `AE-2-H-CD` model achieved better performance than the `CNES-C + NL-Bayes`, even at higher bit rates.

Note that the joint compression and denoising approach (`AE-2-H-CD`), even when still underperforming, reached a performance much closer to the one of the sequential approach when compared to the achieved performance on the standard average luminance images displayed on figures 5.6 and 5.7. Thus, the joint approach appears more attractive here for dealing with images with low average luminance due to its lower complexity. The sequential framework `AE-2-H-C + BRDNet` achieved considerably superior performance than the `CNES-C + NL-Bayes`. This result demonstrates how learned approaches can improve satellite image compression and denoising performance, even when dealing with images in a more challenging scenario. The figure 5.12 also displays the performance of the joint compression and denoising framework that was trained exclusively on the standard average luminance images. This framework was not able to achieve a great performance for compression and denoising in low average luminance images. However, it approached the curve of the reference `CNES-C + NL-Bayes` and indicated a tendency to compress well for this different scenario, showing a behavior close to that of the compression-only method `AE-2-H-C`.

5.4 Conclusions

In this chapter, two different learned approaches for satellite image compression and denoising were proposed. On one side, the joint approach performs compression and denoising with a single architecture. One advantage is that intermediary steps existing in the current CNES imaging system [Delvit et al. (2019)] can be eliminated. This approach is of interest for commercial applications since it provides noise-free images without on ground post-processing. On the other side, the sequential approach allows to consider a splitting compatible with all the architecture simplifications designed for onboard compression in chapter 4. The proposed approaches were shown to outperform the CNES baseline in terms of rate-distortion, visual quality and computational time.

Conclusion

This thesis mainly investigates the feasibility of deep learning techniques for onboard satellite compression. In this context, computational complexity reduction is required due to power limitations and hardware constraints onboard satellites. The major challenge with onboard compression is to obtain a good trade-off between rate and distortion while keeping computational complexity as low as possible.

In chapter 4, we proposed different approaches to adapt the reference learned image compression frameworks [Ballé et al. (2017) and Ballé et al. (2018)] to onboard satellite image compression. First, we studied the different choices regarding these autoencoder-based frameworks. Moreover, we conducted an analysis of the learned representation, which revealed that most of the feature maps exhibit a Laplacian distribution. This analysis led to the proposition of a simplified entropy model that requires estimating a single parameter for each feature map. The proposed simplified entropy model presents an intermediary performance between the more complex non-parametric entropy model [Ballé et al. (2017)] and the most performing hyperprior entropy model [Ballé et al. (2018)]. The combination of the number of filters reduction and the proposed simplified entropy model significantly reduces the complexity without penalizing the compression performance with respect to the reference model. In particular, it outperforms the CCSDS satellite image compression standard.

Note that learned satellite image compression was first considered as a separate module in chapter 4. Chapter 5 was devoted to satellite image compression and denoising based on machine learning approaches. In a typical satellite imaging system, compression is performed onboard the satellite, and denoising is performed on the ground due to the prohibitive complexity of the denoising algorithms. Two different approaches were proposed for satellite image compression and denoising. First, the joint approach was proposed to compress and denoise with a single autoencoder-based architecture. This approach allows to transmit noise-free images, which is particularly advantageous for commercial applications requiring fast obtention of exploitable satellite images. Complementary, the sequential approach allows combining two separate learned-based modules that benefit from different architec-

tures designed to perform the best on each task. Moreover, the denoising architecture can be plugged as a post-treatment module to the joint approach, which further improves the removal of remaining noise and compression artifacts. The proposed approaches for compression and denoising surpassed the CNES baseline in terms of rate-distortion, visual quality, and computational time. We show experimentally that data-driven compression and denoising approaches can significantly improve a typical satellite imaging system.

Of course, there is still a lack of understanding and reliability of ML in industrial applications. It is essential to consider the existing limitations of the machine learning approaches and carefully see them as candidate improvements rather than abrupt replacements. Ideally, it would be interesting to incorporate model-based methods into the learning process and design of data-driven approaches, but this remains a major challenge. Instead of the limitations, we can observe that the ML field progresses fast, and it is expected that ML will become less of a black box and more of a data-driven powerful box.

Perspectives

The possible prospects as a direct extension of this thesis concern four relevant objectives: consider a feature-dependent entropy model, desymmetrize the autoencoder to reduce onboard complexity, the hardware implementation of the proposed learned frameworks of this thesis, and extend the ML-based approaches to other functionalities of a typical satellite imaging system.

Feature-dependent entropy model

The thesis addressed different entropy models, however the choice a specific entropy model was made a priori so that a single model is considered at a time. One possibility would be to combine different entropy models to deal with the diversity of feature maps either adaptively during training or as a post-training adjustment.

Hardware implementation

The propositions of this thesis were developed and tested in specific frameworks for deep learning based on Python. However, it would be highly desirable to implement and test these propositions on real satellite hardware [Rapuano et al. (2021)]. Hardware implementation is particularly relevant to space-based applications. There is an evident need

to reduce the size and complexity of neural networks for deployment in highly constrained hardware platforms. Quantized models are those in which the learned frameworks are represented with lower precision, such as 8-bit integers instead of 32-bit float. Lower precision is required to leverage particular hardware. Some techniques have been proposed for this purpose, such as quantization-aware training and post-training quantization [Krishnamoorthi (2018)]. The hardware implementation of ML architectures can also be performed through dedicated hardware such as Field Programmable Gate Arrays (FPGAs) and GPU (Graphics Processing Units).

Desymmetrize the autoencoder to reduce onboard complexity

The considered autoencoder architectures are symmetric in relation to the analysis and synthesis transforms. One possibility to reduce the onboard complexity would be to reduce the complexity in the analysis transform which would eventually be compensated by an increase in complexity in the part responsible for the synthesis transform on earth.

Extend the ML-based approaches to other functionalities of a typical satellite imaging system

Another perspective within the scope of this thesis would be to use the features extracted for image compression in other ML-based functionalities onboard the satellite, such as classification or detection, avoiding additional feature extraction steps that result in more computational complexity. In particular, it would be interesting to dote the satellite system with some intelligence to prevent the compression and transmission of images that are not exploitable after passing through all the following steps on the ground. Among the non-exploitable images, there are cloud images and images of highly homogeneous zones that do not bring relevant information. For example, [Z. Zhang et al. (2019)] proposed cloud detection onboard satellites using a dedicated CNN architecture for image segmentation. A high cost is associated with transmitting images, so reducing it would be desirable.

The successful extension of the ML-based approach to denoising in chapter 4 opened up prospects for extending the ML to the on ground segment, for example, for the deconvolution or even constituting newer functionalities.

Appendix A

Complexity assessment

A.1 Complexity assessment for the end-to-end compression method

First, let characterize the computational complexity of a convolutional layer composing the analysis and synthesis transforms. Let N_{in} denote the number of features at the considered layer input. In the particular case of the network first layer, N_{in} is the number of channels of the input image ($N_{in} = 1$ for a panchromatic image) else N_{in} is the number of filters of the previous layer. Let N_{out} denote the number of features at this layer output, that is the number of filters of this layer. As detailed in chapter 4, in [Ballé et al. (2018)], $N_{out} = N$ for each layer of the analysis and synthesis transforms except for the last one of the main auto-encoder analysis transform and the last one of the auxiliary auto-encoder synthesis transform composed of M filters with $M > N$ and thus for these layers $N_{out} = M$. As in [Ballé et al. (2017) and Ballé et al. (2018)], we consider square filters with kernel size $n \times n$. The number of parameters associated to the filtering part of the layer is [Cheng et al. (2019)]:

$$\text{Param}^f = (n \times n \times N_{in} + \delta) \times N_{out}. \tag{A.1}$$

The term δ is equal to 1 when a bias is introduced and is equal to 0 otherwise. Note that this bias is rarely used in the considered architectures (except in Tconv3, as displayed in Figure 4.1). The filtering is applied to each input channel after downsampling (respectively upsampling). The downsampled (resp. upsampled) input channel if of size $s_{out} \times s_{out}$ with $s_{out} = s_{in}/D$ (respectively $s_{out} = s_{in} \times D$) where D denotes the downsampling (respectively upsampling) factor and $s_{in} \times s_{in}$ is the size of a feature at the filter input. Floating points operations Operation^f for the filtering operation is thus:

$$\text{Operation}^f = \text{Param}^f \times s_{out} \times s_{out}. \tag{A.2}$$

GDN/IGDN perform a normalization of a filter output with respect to the other filter outputs. According to chapter 4, the number of parameters and the number of operations of each GDN/IGDN are expressed by [Cheng et al. (2019)]:

$$\begin{aligned} \text{Param}^g &= (N_{out} + 1) \times N_{out} \\ \text{Operation}^g &= \text{Param}^g \times s_{out} \times s_{out}. \end{aligned} \tag{A.3}$$

Since the number of layers is already very low for the considered architectures, the reduction of the complexity of the analysis and synthesis transforms may target, according to the previous complexity assessment, the number of filters per layer, the size of these filters and the choice of the activation functions. Our proposal below details our strategy for complexity reduction.

A.2 Complexity assessment for the sequential denoising method

As detailed in chapter 5, in [Tian et al. (2020)], $N_{out} = 64$ for each layer of the BRDNet architecture, except in the last layer after the concatenation where $N_{out} = 128$. As in [K. Zhang et al. (2017a) and Tian et al. (2020)], we use standard square filters with kernel size $n \times n$. The number of parameters associated to the filtering part of the layer corresponds to Equation A.1. Note that the bias term is not used in the whole considered BRDNet architecture. The complexity associated to a dilated convolutional layer (also present) corresponds to the one of a standard convolutional layer, since only the filter layout changes and not its size. $s_{in} \times s_{in}$ stands for the size of a feature at the filter input. Note that the feature size corresponds to the input image size in the BRDNet architecture for all its layers. Consecutively, floating-points operations for the filtering operation correspond to A.2.

During inference, batch renormalization (BRN) performs a moving average operation of each filter output. According to section 5, the number of parameters and the number of operations of each BRN can be expressed by:

$$\begin{aligned} \text{Param}^b &= 4 \times N_{out} \\ \text{Operation}^b &= \text{Param}^{b,BRDNet} \times s_{out} \times s_{out}. \end{aligned} \tag{A.4}$$

Bibliography

Alves de Oliveira, V., M. Chabert, T. Oberlin, C. Poulliat, M. Bruno, C. Latry, M. Carlavan, S. Henrot, F. Falzon, and R. Camarero (2021). "Reduced-Complexity End-to-End Variational Autoencoder for on Board Satellite Image Compression." In: *Remote Sensing* 13.3 (cit. on pp. 4, 59, 98, 100, 104).

– (2022). "Satellite Image Compression and Denoising With Neural Networks." In: *IEEE Geoscience and Remote Sensing Letters* 19, pp. 1–5 (cit. on pp. 4, 93).

Alves de Oliveira, V., T. Oberlin, M. Chabert, C. Poulliat, B. Mickael, C. Latry, M. Carlavan, S. Henrot, F. Falzon, and R. Camarero (2020). "Simplified entropy model for reduced-complexity end-to-end variational autoencoder with application to on-board satellite image compression." In: *7th International Workshop on On-Board Payload Data Compression (OBPDC 2020)*, pp. 1–8 (cit. on p. 4).

Anscombe, F. J. (1948). "The transformation of Poisson, binomial and negative-binomial data." In: *Biometrika* 35.3/4, pp. 246–254 (cit. on pp. 17, 18, 100).

Araujo, A., W. Norris, and J. Sim (2019). "Computing receptive fields of convolutional neural networks." In: *Distill* 4.11, e21 (cit. on p. 52).

Ballé, J. (2018). "Efficient Nonlinear Transforms for Lossy Image Compression." In: *2018 IEEE Picture Coding Symposium (PCS)*, pp. 248–252 (cit. on pp. 42, 72, 84, 85).

Ballé, J., V. Laparra, and E. Simoncelli (2017). "End-to-end optimized image compression." In: *International Conference on Learning Representations (ICLR)* (cit. on pp. viii, 2, 38, 39, 41, 42, 44, 47, 48, 60–62, 64, 65, 67, 70–75, 78, 81, 84, 89–92, 94, 105, 109, 112).

Ballé, J., V. Laparra, and E. P. Simoncelli (2016). "Density modeling of images using a generalized normalization transformation." In: *International Conference on Learning Representations, ICLR* (cit. on p. 39).

Ballé, J., D. Minnen, S. Singh, S. J. Hwang, and N. Johnston (2018). "Variational image compression with a scale hyperprior." In: *International Conference on Learning Representations (ICLR)* (cit. on pp. viii, ix, 2, 38, 39, 41, 42, 44, 47–49, 60–63, 65–67, 70–76, 78, 81, 82, 84, 86, 88–92, 94, 95, 97–99, 105, 109, 112).

Bårli, E. M., A. Yazidi, E. H. Viedma, and H. Haugerud (2021). "DoS and DDoS mitigation using Variational Autoencoders." In: *Computer Networks* 199, p. 108399 (cit. on p. 33).

Bengio, Y. (2009). "Learning deep architectures for AI." In: *Foundations and trends in Machine Learning* 2.1, pp. 1–127 (cit. on pp. 2, 30, 33–35, 64, 96).

Book, B. (2017). "Consultative Committee for Space Data Systems (CCSDS), Image Data Compression CCSDS 122.0-B-2, ser. Blue Book." In: *CCSDS Secretariat* (cit. on pp. ix, 1, 17, 74–76, 78, 80–82, 92).

Book, G. (2015). “Consultative Committee for Space Data Systems (CCSDS), Image Data Compression CCSDS 120.1-G-2, ser. Green Book.” In: *CCSDS Secretariat* (cit. on pp. 8, 12, 14, 15, 91).

Braun, S., D. Ewins, S. S. Rao, and A. Leissa (2002). “Encyclopedia of Vibration: Volumes 1, 2, and 3.” In: *Appl. Mech. Rev.* 55.3, B45–B45 (cit. on p. 8).

Buades, A., B. Coll, and J.-M. Morel (2005a). “A non-local algorithm for image denoising.” In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 2. IEEE, pp. 60–65 (cit. on pp. 16, 19).

Buades, A., B. Coll, and J.-M. Morel (2005b). “A review of image denoising algorithms, with a new one.” In: *Multiscale modeling & simulation* 4.2, pp. 490–530 (cit. on p. 19).

Burger, H. C., C. J. Schuler, and S. Harmeling (2012). “Image denoising: Can plain neural networks compete with BM3D?” In: *2012 IEEE conference on computer vision and pattern recognition*. IEEE, pp. 2392–2399 (cit. on p. 53).

Cai, C., L. Chen, X. Zhang, and Z. Gao (2018). “Efficient variable rate image compression with multi-scale decomposition network.” In: *IEEE Transactions on Circuits and Systems for Video Technology* 29.12, pp. 3687–3700 (cit. on p. 41).

Carandini, M. and D. J. Heeger (2012). “Normalization as a canonical neural computation.” In: *Nature Reviews Neuroscience* 13.1, pp. 51–62 (cit. on p. 42).

Carlavan, M., L. Blanc-Féraud, M. Antonini, C. Thiebaut, C. Latry, and Y. Bobichon (2012). “A satellite imaging chain based on the Compressed Sensing technique.” In: *On-Board Payload Data Compression Workshop* (cit. on pp. 6, 97).

Chai, D. and A. Bouzerdoum (2001). “JPEG2000 image compression: an overview.” In: *The Seventh Australian and New Zealand Intelligent Information Systems Conference, 2001*, pp. 237–241 (cit. on pp. 13, 14).

Cheng, Z., H. Sun, M. Takeuchi, and J. Katto (2019). “Deep Residual Learning for Image Compression.” In: *2019 IEEE Conference on Computer Vision and Pattern Recognition Workshops* (cit. on p. 112).

Christopoulos, C., A. Skodras, and T. Ebrahimi (2000). “The JPEG2000 still image coding system: an overview.” In: *IEEE transactions on consumer electronics* 46.4, pp. 1103–1127 (cit. on pp. 12, 13).

Dabov, K., A. Foi, V. Katkovnik, and K. Egiazarian (2007). “Image denoising by sparse 3-D transform-domain collaborative filtering.” In: *IEEE Transactions on image processing* 16.8, pp. 2080–2095 (cit. on pp. 16, 19).

Delvit, J.-M., C. Thiebaut, C. Latry, G. Blanchet, and R. Camarero (2019). “A pipeline to improve compressed image quality.” In: *International Conference on Space Optics—ICSO 2018*. Vol. 11180. International Society for Optics and Photonics, p. 111807I (cit. on pp. viii, 2, 6, 17–19, 22, 23, 93, 97, 99, 101, 104, 108).

Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei (2009). “ImageNet: A large-scale hierarchical image database.” In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255 (cit. on p. 52).

Dong, C., Y. Deng, C. C. Loy, and X. Tang (2015). "Compression artifacts reduction by a deep convolutional network." In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 576–584 (cit. on pp. 2, 57).

Dong, C., C. C. Loy, K. He, and X. Tang (2014). "Learning a deep convolutional network for image super-resolution." In: *European conference on computer vision*. Springer, pp. 184–199 (cit. on p. 57).

Dumas, T., A. Roumy, and C. Guillemot (2018). "Autoencoder based image compression: can the learning be quantization independent?" In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1188–1192 (cit. on pp. 44, 67).

Foi, A., V. Katkovnik, and K. Egiazarian (2007). "Pointwise shape-adaptive DCT for high-quality denoising and deblocking of grayscale and color images." In: *IEEE transactions on image processing* 16.5, pp. 1395–1411 (cit. on pp. 17, 57).

Glorot, X. and Y. Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks." In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, pp. 249–256 (cit. on p. 36).

Glorot, X., A. Bordes, and Y. Bengio (2011). "Deep sparse rectifier neural networks." In: *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 315–323 (cit. on p. 28).

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep Learning*. MIT Press (cit. on pp. 26, 28, 29).

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014). "Generative adversarial nets." In: *Advances in neural information processing systems*, pp. 2672–2680 (cit. on p. 58).

Goyal, V. K. (2001). "Theoretical foundations of transform coding." In: *IEEE Signal Processing Magazine* 18.5, pp. 9–21 (cit. on p. 1).

Gray, R. M. and D. L. Neuhoff (1998). "Quantization." In: *IEEE transactions on information theory* 44.6, pp. 2325–2383 (cit. on p. 44).

Hayat, K., W. Puech, and G. Gesquiere (2009). "JPEG2000-Based Data Hiding and its Application to 3D Visualization." In: *Recent Advances in Signal Processing*. IntechOpen (cit. on pp. 9, 10, 13).

He, K., X. Zhang, S. Ren, and J. Sun (2016). "Deep residual learning for image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778 (cit. on pp. 41, 49, 50, 52).

Heeger, D. J. (1992). "Normalization of cell responses in cat striate cortex." In: *Visual neuroscience* 9.2, pp. 181–197 (cit. on p. 42).

Hinton, G., L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. (2012). "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups." In: *IEEE Signal processing magazine* 29.6, pp. 82–97 (cit. on p. 27).

Hu, J., L. Shen, and G. Sun (2018). "Squeeze-and-excitation networks." In: *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7132–7141 (cit. on pp. 2, 30).

Hu, Y., W. Yang, Z. Ma, and J. Liu (2020). "Learning End-to-End Lossy Image Compression: A Benchmark." In: *arXiv preprint arXiv:2002.03711* (cit. on p. 89).

– (2021). "Learning end-to-end lossy image compression: A benchmark." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (cit. on pp. 38–41).

Huang, B. (2011). "Satellite Data Compression." In: *Springer Science & Business Media* (cit. on p. 1).

Huang, B., A. Ahuja, H.-L. Huang, T. J. Schmit, and R. W. Heymann (2004). "Comparison of arithmetic coding and prefix coding with the CCSDS lossless compression recommendation for satellite data." In: *13th Conference on Satellite Meteorology and Oceanography P.* Vol. 1. Citeseer, p. 14 (cit. on p. 12).

Ioffe, S. (2017). "Batch renormalization: Towards reducing minibatch dependence in batch-normalized models." In: *arXiv preprint arXiv:1702.03275* (cit. on pp. 51, 55, 95).

Ioffe, S. and C. Szegedy (2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift." In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 448–456 (cit. on pp. 50, 53).

Jain, V. and S. Seung (2008). "Natural image denoising with convolutional networks." In: *Advances in neural information processing systems* 21 (cit. on p. 53).

Kaiser, P., J. D. Wegner, A. Lucchi, M. Jaggi, T. Hofmann, and K. Schindler (2017). "Learning aerial image segmentation from online maps." In: *IEEE Transactions on Geoscience and Remote Sensing* 55, pp. 6054–6068 (cit. on pp. 2, 30).

Khan, A., B. Nagendra, and N. Misra (2013). "Entropy Coding Technique for Compression of Satellite Vibration Test Data." In: (cit. on p. 12).

Kingma, D. P. and M. Welling (2013). "Auto-encoding variational bayes." In: *arXiv preprint arXiv:1312.6114* (cit. on p. 34).

– (2019). "An Introduction to Variational Autoencoders." In: (cit. on pp. viii, 34).

Krishnamoorthi, R. (2018). "Quantizing deep convolutional networks for efficient inference: A whitepaper." In: *arXiv preprint arXiv:1806.08342* (cit. on p. 111).

Lam, E. Y. and J. W. Goodman (2000). "A mathematical analysis of the DCT coefficient distributions for images." In: *IEEE Transactions on Image Processing* 9.10, pp. 1661–1666 (cit. on p. 67).

Latry, C., S. Fourest, and C. Thiebaut (2012). "Restoration technique for Pleiades-HR panchromatic images." In: *Int.. Arch. Photogram., Remote Sens. Spatial Inf. Sci.* 39, pp. 555–560 (cit. on pp. 6, 7, 23).

Lebrun, M., A. Buades, and J.-M. Morel (2013). "A nonlocal bayesian image denoising algorithm." In: *SIAM Journal on Imaging Sciences* 6.3, pp. 1665–1688 (cit. on pp. 16, 19).

LeCun, Y., Y. Bengio, and G. Hinton (2015). "Deep learning." In: *nature* 521.7553, pp. 436–444 (cit. on pp. 28, 36).

Lu, Y. (2019). "The Level Weighted Structural Similarity Loss: A Step Away from MSE." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01, pp. 9989–9990 (cit. on pp. 46, 47).

Lu, Z.-M. and S.-Z. Guo (2017). "Chapter 1 - Introduction." In: *Lossless Information Hiding in Images*. Ed. by L. Zhe-Ming and G. Shi-Ze. Syngress, pp. 1–68 (cit. on p. 12).

Lyu, S. (2010). "Divisive normalization: Justification and effectiveness as efficient coding transform." In: *Advances in neural information processing systems (NIPS 2010)* 23, pp. 1522–1530 (cit. on p. 42).

Makitalo, M. and A. Foi (2012). "Optimal inversion of the generalized Anscombe transformation for Poisson-Gaussian noise." In: *IEEE transactions on image processing* 22.1, pp. 91–103 (cit. on pp. 18, 19).

Mäkitalo, M. and A. Foi (2012). "Poisson-gaussian denoising using the exact unbiased inverse of the generalized anscombe transformation." In: *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, pp. 1081–1084 (cit. on p. 19).

Malo, J., I. Epifanio, R. Navarro, and E. P. Simoncelli (2005). "Nonlinear image representation for efficient perceptual coding." In: *IEEE Transactions on Image Processing* 15.1, pp. 68–80 (cit. on p. 42).

Malo, J. and V. Laparra (2010). "Psychophysically tuned divisive normalization approximately factorizes the PDF of natural images." In: *Neural computation* 22.12, pp. 3179–3206 (cit. on p. 42).

Mante, V., V. Bonin, and M. Carandini (2008). "Functional mechanisms shaping lateral geniculate responses to artificial and natural stimuli." In: *Neuron* 58.4, pp. 625–638 (cit. on p. 42).

Marcellin, M. W., M. A. Lepley, A. Bilgin, T. J. Flohr, T. T. Chinen, and J. H. Kasner (2002). "An overview of quantization in JPEG 2000." In: *Signal Processing: Image Communication* 17.1, pp. 73–84 (cit. on p. 57).

Marcellin and Taubman (2002). "JPEG2000: image compression fundamentals, standards, and practice." In: *International Series in Engineering and Computer Science, Secs* 642 (cit. on pp. ix, 8, 9, 11, 74–76, 80–82, 90).

Martin, G. (1979). "Range encoding: an algorithm for removing redundancy from a digitised message." In: *Video and Data Recording Conference, Southampton*, pp. 24–27 (cit. on p. 12).

Masse, A., S. Lefevre, R. Binet, S. Artigues, G. Blanchet, and S. Baillarin (2018). "Denoising very high resolution optical remote sensing images: Application and optimization of non-local bayes method." In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 11.3, pp. 691–700 (cit. on pp. x, 6, 16, 17, 19, 20, 103).

Mentzer, F., E. Agustsson, M. Tschannen, R. Timofte, and L. Van Gool (2018). "Conditional probability models for deep image compression." In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, p. 3 (cit. on p. 41).

Nosratinia, A. (1999). "Embedded post-processing for enhancement of compressed images." In: *Proceedings DCC'99 Data Compression Conference (Cat. No. PR00096)*. IEEE, pp. 62–71 (cit. on p. 57).

Oberlin, T., F. Malgouyres, and J.-Y. Wu (2019). "Loss functions for denoising compressed images: a comparative study." In: *2019 27th European Signal Processing Conference (EUSIPCO)*. IEEE, pp. 1–5 (cit. on p. 17).

Oh, E. and J.-K. Choi (2014). "GOCI image enhancement using an MTF compensation technique for coastal water applications." In: *Optics express* 22.22, pp. 26908–26918 (cit. on p. 23).

Ose, K., T. Corpetti, and L. Demagistri (2016). "2 - Multispectral Satellite Image Processing." In: *Optical Remote Sensing of Land Surface*. Ed. by N. Baghdadi and M. Zribi. Elsevier, pp. 57–124 (cit. on p. 7).

Panda, M. N., C. C. Mosher, and A. K. Chopra (Mar. 2000). "Application of Wavelet Transforms to Reservoir-Data Analysis and Scaling." In: *SPE Journal* 5.01, pp. 92–101 (cit. on p. 8).

Pratt, J. W. and J. D. Gibbons (1981). "Kolmogorov-Smirnov two-sample tests." In: *Concepts of Nonparametric Theory*, pp. 318–344 (cit. on p. 68).

Rapuano, E., G. Meoni, T. Pacini, G. Dinelli, G. Furano, G. Giuffrida, and L. Fanucci (2021). "An FPGA-Based Hardware Accelerator for CNNs Inference on Board Satellites: Benchmarking with Myriad 2-Based Solution for the CloudScout Case Study." In: *Remote Sensing* 13.8 (cit. on p. 110).

Redmon, J., S. Divvala, R. Girshick, and A. Farhadi (2016). "You only look once: Unified, real-time object detection." In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788 (cit. on pp. 2, 30).

Rippel, O. and L. Bourdev (2017). "Real-Time Adaptive Image Compression." In: *International Conference on Machine Learning (ICML)*, pp. 2922–2930 (cit. on pp. 2, 38, 41).

Rosenblatt, F. (1958). "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6, p. 386 (cit. on p. 26).

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). "Learning representations by back-propagating errors." In: *nature* 323.6088, p. 533 (cit. on p. 36).

Rumelhart, D. E., J. L. McClelland, P. R. Group, et al. (1988). *Parallel distributed processing*. Vol. 1. IEEE Massachusetts (cit. on p. 29).

Salimans, T., I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen (2016). "Improved techniques for training gans." In: *Advances in neural information processing systems* 29, pp. 2234–2242 (cit. on p. 58).

Schwartz, O. and E. P. Simoncelli (2001). "Natural signal statistics and sensory gain control." In: *Nature neuroscience* 4.8, pp. 819–825 (cit. on p. 42).

Shannon, C. E. (1948). "A mathematical theory of communication." In: *The Bell system technical journal* 27.3, pp. 379–423 (cit. on p. 12).

Simoncelli, E. P. and D. J. Heeger (1998). "A model of neuronal responses in visual area MT." In: *Vision research* 38.5, pp. 743–761 (cit. on p. 42).

Simonyan, K. and A. Zisserman (2014). "Very deep convolutional networks for large-scale image recognition." In: *arXiv preprint arXiv:1409.1556* (cit. on pp. 52, 53).

Sinz, F. H. and M. Bethge (2013). "What is the limit of redundancy reduction with divisive normalization?" In: *Neural Computation* 25.11, pp. 2809–2814 (cit. on p. 42).

Subramanya, A. (2001). “Image compression technique.” In: *IEEE Potentials* 20.1, pp. 19–23 (cit. on p. 7).

Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich (2015). “Going deeper with convolutions.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9 (cit. on pp. 41, 55, 95).

Taubman, D. (2000). “High performance scalable image compression with EBCOT.” In: *IEEE Transactions on Image Processing* 9.7, pp. 1158–1170 (cit. on p. 14).

Theis, L., W. Shi, A. Cunningham, and F. Huszár (2017). “Lossy image compression with compressive autoencoders.” In: *International Conference on Learning Representations (ICLR)* (cit. on pp. 2, 38, 39, 41, 42, 45, 47, 48, 65).

Thiebaut, C., C. Latry, R. Camarero, J.-M. Delvit, G. Blanchet, X. Delaunay, E. Bousquet, and G. Lauren (2016). “Performances of a CCSDS-based algorithm for quality- controlled compression on Earth observation missions.” In: *5th International Workshop on On-Board Payload Data Compression (OBPDC 2016)* (cit. on pp. 15, 17, 100, 101).

Tian, C., Y. Xu, and W. Zuo (2020). “Image denoising using deep CNN with batch renormalization.” In: *Neural Networks* 121, pp. 461–473 (cit. on pp. viii, ix, 49, 55, 56, 94–98, 100, 103, 113).

Tihonov, A. N. (1963). “Solution of incorrectly formulated problems and the regularization method.” In: *Soviet Math.* 4, pp. 1035–1038 (cit. on p. 23).

Toderici, G., S. M. O’Malley, S. J. Hwang, D. Vincent, D. Minnen, S. Baluja, M. Covell, and R. Sukthankar (2016). “Variable Rate Image Compression with Recurrent Neural Networks.” In: *International Conference on Learning Representations* (cit. on p. 40).

Toderici, G., D. Vincent, N. Johnston, S. J. Hwang, D. Minnen, J. Shor, and M. Covell (2017). “Full Resolution Image Compression with Recurrent Neural Networks.” In: *CVPR*, pp. 5435–5443 (cit. on p. 40).

Tretter, D., N. Memon, and C. Bouman (2005). *Multispectral image coding* (cit. on p. 8).

Vincent, P., H. Larochelle, Y. Bengio, and P.-A. Manzagol (2008). “Extracting and composing robust features with denoising autoencoders.” In: *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103 (cit. on p. 33).

Vincent, P., H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, and L. Bottou (2010). “Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion.” In: *Journal of machine learning research* 11.12 (cit. on pp. 33, 34).

Wang, Z., A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli (2004). “Image quality assessment: from error visibility to structural similarity.” In: *IEEE transactions on image processing* 13.4, pp. 600–612 (cit. on p. 46).

Wang, Z., E. P. Simoncelli, and A. C. Bovik (2003). “Multiscale structural similarity for image quality assessment.” In: *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003* 2, pp. 1398–1402 (cit. on pp. 46, 64).

Wong, S. M., S. Ong, and C. S. Chan (2020). “Regularization-based modulation transfer function compensation for optical satellite image restoration using joint statistical model

in curvelet domain." In: *Journal of Applied Remote Sensing* 14.3, p. 036506 (cit. on pp. 6, 7).

Wong, T.-S., C. A. Bouman, I. Pollak, and Z. Fan (2009). "A document image model and estimation algorithm for optimized JPEG decompression." In: *IEEE Transactions on Image Processing* 18.11, pp. 2518–2535 (cit. on p. 57).

Xing, Y., M. Kaaniche, B. Pesquet-Popescu, and F. Dufaux (2015). *Digital holographic data representation and compression*. Academic Press (cit. on p. 11).

Xingjian, S., Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo (2015). "Convolutional LSTM network: A machine learning approach for precipitation nowcasting." In: *Advances in neural information processing systems*, pp. 802–810 (cit. on p. 40).

Yang, Y., N. Galatsanos, and A. Katsaggelos (2005). "5.8 - Recovery Methods for Post-processing of Compressed Images." In: *Handbook of Image and Video Processing (Second Edition)*. Ed. by A. BOVIK. Second Edition. Communications, Networking and Multimedia. Burlington: Academic Press, pp. 761–774 (cit. on pp. 56, 57).

Yu, F. and V. Koltun (2015). "Multi-scale context aggregation by dilated convolutions." In: *arXiv preprint arXiv:1511.07122* (cit. on pp. 54, 56, 96).

Zaheer, R. and H. Shaziya (2019). "A study of the optimization algorithms in deep learning." In: *2019 Third International Conference on Inventive Systems and Control (ICISC)*. IEEE, pp. 536–539 (cit. on p. 37).

Zhang, K., W. Zuo, Y. Chen, D. Meng, and L. Zhang (2017a). "Beyond a gaussian denoiser: Residual learning of deep CNN for image denoising." In: *IEEE Transactions on Image Processing* 26.7, pp. 3142–3155 (cit. on pp. viii, 2, 49, 52–56, 95, 96, 113).

Zhang, K., W. Zuo, S. Gu, and L. Zhang (2017b). "Learning deep CNN denoiser prior for image restoration." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3929–3938 (cit. on pp. 56, 96).

Zhang, K., W. Zuo, and L. Zhang (2018). "FFDNet: Toward a fast and flexible solution for CNN-based image denoising." In: *IEEE Transactions on Image Processing* 27.9, pp. 4608–4622 (cit. on pp. viii, 49, 54, 56, 96).

Zhang, Z., A. Iwasaki, G. Xu, and J. Song (2019). "Cloud detection on small satellites based on lightweight U-net and image compression." In: *Journal of Applied Remote Sensing* 13.2, p. 026502 (cit. on p. 111).

Zhao, H., O. Gallo, I. Frosio, and J. Kautz (2016). "Loss functions for image restoration with neural networks." In: *IEEE Transactions on computational imaging* 3.1, pp. 47–57 (cit. on pp. 45, 47).

Zhong, Y., L. Liu, D. Zhao, and H. Li (2020). "A generative adversarial network for image denoising." In: *Multimedia Tools and Applications* 79.23, pp. 16517–16529 (cit. on p. 58).