REPUBLIC OF TÜRKİYE
YILDIZ TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

POST-TRAINING MODEL QUANTIZATION APPROACHES IN COMPRESSION OF MULTITEMPORAL MULTISPECTRAL IMAGES

Naser SADEGHI GHARAGOZ

MASTER OF SCIENCE THESIS
Department of Computer Engineering
Program of Computer Engineering

Assoc. Prof. Dr. Ali Can KARACA

March, 2025

REPUBLIC OF TÜRKİYE
YILDIZ TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

POST-TRAINING MODEL QUANTIZATION APPROACHES IN COMPRESSION OF MULTITEMPORAL MULTISPECTRAL IMAGES

A thesis submitted by Naser SADEGHI GHARAGOZ in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE THESIS** is approved by the committee on 04.03.2025 in Department of Computer Engineering , Program of Computer Engineering .

Assoc. Prof. Dr. Ali Can KARACA
Yildiz Technical University
Supervisor

Approved By the Examining Committee

Assoc. Prof. Dr. Ali Can KARACA, Supervisor
Yildiz Technical University _____

Prof. Dr. Gökhan BİLGİN, Member
Yildiz Technical University _____

Assoc. Prof. Dr. Süleyman EKEN, Member
Kocaeli University _____

I hereby declare that I have obtained the required legal permissions during data collection and exploitation procedures, that I have made the in-text citations and cited the references properly, that I haven't falsified and/or fabricated research data and results of the study and that I have abided by the principles of the scientific research and ethics during my Thesis Study under the title of "Post-Training Model Quantization Approaches in Compression of Multitemporal Multispectral Images " supervised by my supervisor, Assoc. Prof. Dr. Ali Can KARACA . In the case of a discovery of false statement, I am to acknowledge any legal consequence.

Naser SADEGHI GHARAGOZ

Signature

Dedicated to my family, my mother Fariba YAZDANI KHANESHAN,
and my beloved wife.

ACKNOWLEDGEMENTS

Pursuing this research in the field of remote sensing and multispectral image compression has been both a challenging and rewarding experience. Investigating novel approaches to enhance the efficiency of multitemporal satellite image compression has deepened my understanding of data-driven methods and strengthened my passion for developing innovative solutions in this domain.

I extend my sincere gratitude to my supervisor, Assoc. Prof. Dr. Ali Can Karaca, whose invaluable guidance, insightful feedback, and continuous support have been instrumental throughout this research. His expertise and encouragement have greatly contributed to the development and refinement of this work.

Finally, I am profoundly grateful to my family and my beloved wife for their unwavering support, patience, and encouragement throughout this journey. Their belief in me has been a constant source of motivation, making this endeavor even more fulfilling.

Naser SADEGHI GHARAGOZ

TABLE OF CONTENTS

LIST OF ABBREVIATIONS viii

LIST OF FIGURES x

LIST OF TABLES xiv

ABSTRACT xvi

ÖZET xvii

1 INTRODUCTION 1

1.1 Context and Significance 1
1.2 Problem Formulation 2
1.3 Research Goals 2
1.4 Overview of the Thesis 3
1.5 Ethics 4

2 BACKGROUND AND RELATED WORK 5

2.1 Background 5
2.2 Related Work 6
2.2.1 JPEG2000 (JP2K) 7
2.2.2 ResNet 8
2.2.3 U-Net 9
2.2.4 LinkNet 10
2.2.5 MultiTempGAN 11
2.2.6 Performance Comparison of the Related Works 12

3 Datasets and Data Preparation 20

3.1 Datasets 20
3.2 Data Preprocessing 22
3.2.1 Normalization 22
3.2.2 Concatenation 22
3.2.3 Image Splitting 22
3.2.4 Batch Preparation 22

3.2.5 Data Augmentation (Horizontal Flipping) 23

4 Quantization Approaches **24**
4.1 Quantization 24
4.2 Uniform Quantization 25
4.2.1 Quantizing All Weights 26
4.2.2 Linear Quantization 30
4.2.3 Fixed-Point Quantization 33
4.2.4 Stochastic Quantization 36
4.2.5 Symmetric Quantization 40
4.2.6 Asymmetric Quantization 42
4.2.7 Performance Comparison of All Uniform Approaches . . . 45
4.3 Mixed-precision Quantization 49
4.3.1 Quantizing Only Convolutional Layers 49
4.3.2 Quantizing Only Non-Convolutional Layers 52
4.3.3 Quantizing Convolutional and Non-Convolutional Layers to Different Bit Widths 54
4.3.4 Layer-Wise Precision Quantization (LWPQ) 57
4.3.5 Performance Comparison of All Mixed-precision Approaches 60
4.4 Non-uniform Quantization 64
4.4.1 Logarithmic Quantization 64
4.4.2 Dynamic Fixed-Point Quantization (DFPQ) 67
4.4.3 Learned Step Size Quantization (LSSQ) 69
4.4.4 Scalar Dead Zone Quantization (SDZQ) 72
4.4.5 Lloyd Max Quantization 77
4.4.6 K-Means Quantization 79
4.4.7 Piecewise Linear Quantization (PLQ) 81
4.4.8 Performance Comparison of All Non-Uniform Approaches . 86

5 Proposed Model: PLQ-MultiTempGAN **89**
5.1 PLQ-MultiTempGAN Details 89
5.1.1 Change Detection Maps (CDMs) 91

6 DISCUSSION **94**
6.1 Development Insights of PLQ-MultiTempGAN 94
6.2 Motivation and Design Choices 94
6.3 Quantization Segmentation 94
6.4 Limitations 95
6.5 Future Work 96

7 CONCLUSION **97**
7.1 Summary and Insights 97

REFERENCES **99**

A PROJECT ENVIRONMENT SETUP **103**

PUBLICATIONS FROM THE THESIS **104**

LIST OF ABBREVIATIONS

bpp	Bit-per-pixel
CPU	Central Processing Unit
CDM	Change Detection Map
CCSDS	Consultative Committee for Space Data Systems
CAE	Convolutional Autoencoder
CNN	Convolutional Neural Networks
DZW	Dead-zone Width
DWT	Discrete Wavelet Transform
DFPQ	Dynamic Fixed-Point Quantization
EDSR	Enhanced Deep Super-Resolution Network
GAN	Generative Adversarial Network
GPU	Graphical Processing Unit
HS	Hyperspectral
JP2K	JPEG2000
LMSE	Laplacian Mean Square Error
LWPQ	Layer-Wise Precision Quantization
LSSQ	Learned Step Size Quantization
MB	Megabytes
MS	Multispectral
PLQ	Piecewise Linear Quantization
PCA	Principal Component Analysis
ReLU	Rectified Linear Unit
ResNet	Residual Network

RNN	Residual Neural Network
SDZQ	Scalar Dead-Zone Quantization
SNR	Signal-to-noise Ratio
SRResNet	Super-Resolution Residual Network
TPU	Tensor Processing Unit

LIST OF FIGURES

Figure 2.1 Core structure of encoder and decoder in JP2K. 7
Figure 2.2 ResNet Architecture. 9
Figure 2.3 U-Net Architecture. 10
Figure 2.4 LinkNet Architecture. 11
Figure 2.5 MultiTempGAN architecture with the generator at the top and the discriminator at the bottom, processing concatenated real and generated images. Discriminator's convolutional layers are shown without parameter counts, as they are less critical for compression. The final layer outputs real/fake probabilities. . . . 12
Figure 2.6 Total runtime and generator loss per epoch for ResNet. 15
Figure 2.7 Total runtime and generator loss per epoch for U-Net. 16
Figure 2.8 Total runtime and generator loss per epoch for LinkNet. 17
Figure 2.9 Total runtime, generator, and discriminator loss per epoch for MultiTempGAN. 18
Figure 2.10 Visual comparison of reference, target, and generated images using ResNet, U-Net, LinkNet, and MultiTempGAN. 19
Figure 3.1 RGB representation of an MS image patch pair, where each set of three channels is visualized as a single image. 21
Figure 4.1 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with quantizing all weights approach. 30
Figure 4.2 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with linear quantization approach. 33
Figure 4.3 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with fixed-point quantization approach. 36
Figure 4.4 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with stochastic quantization approach. 39

Figure 4.5 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with symmetric quantization approach. 42
Figure 4.6 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with asymmetric quantization approach. 44
Figure 4.7 Visual Comparison of all deep learning-based models and uniformly quantized versions of MultiTempGAN using the best bit precision for each method. 46
Figure 4.8 Comparison of all uniform approaches, ResNet, U-Net, LinkNet, and MultiTempGAN based on Euclidean distance calculated with SNR and bpp metrics. 47
Figure 4.9 Q-MultiTempGAN Architecture with the generator structure is shown at the top, discriminator at the bottom left, and quantization process in the bottom right. 48
Figure 4.10 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with quantizing only convolutional layers weights approach. 51
Figure 4.11 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with quantizing only non-convolutional layers weights approach. . . 54
Figure 4.12 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with quantizing convolutional and non-convolutional layers weights to different bit widths approach. 56
Figure 4.13 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with LWPQ. . . 60
Figure 4.14 Visual Comparison of all deep learning-based models and quantized versions of MultiTempGAN using the mixed-precision approaches and the best bit precision for each approach. 62
Figure 4.15 Comparison of all mixed-precision approaches, ResNet, U-Net, LinkNet, and MultiTempGAN based on Euclidean distance calculated with SNR and bpp metrics. 63
Figure 4.16 MPQ-MultiTempGAN Architecture. 63
Figure 4.17 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with logarithmic approach. 67

Figure 4.18 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with DFPQ approach. 69
Figure 4.19 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with LSSQ approach. 72
Figure 4.20 Visualization of DZW based on both range and standard deviation of the model weights. 75
Figure 4.21 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with SDZQ approach. 77
Figure 4.22 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with Lloyd Max approach. 79
Figure 4.23 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with K-Means approach. 81
Figure 4.24 Visualization of PLQ's Segments for each of the three datasets. . 83
Figure 4.25 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with PLQ approach. 85
Figure 4.26 Visual comparison of all deep learning-based models and top 5 non-uniformly quantized versions of MultiTempGAN using the best bit precision for each method. 87
Figure 4.27 Comparison of all non-uniform approaches, ResNet, U-Net, LinkNet, and MultiTempGAN based on Euclidean distance calculated with SNR and bpp metrics. 88
Figure 5.1 Proposed Model (PLQ-MultiTempGAN) Architecture. 90
Figure 5.2 Comparison of the proposed model with other approaches based on SNR and bpp. Symm and Stoch denote symmetric and stochastic quantization respectively. 91
Figure 5.3 Comparison of the proposed model with other approaches based on LMSE and bpp. Symm and Stoch denote symmetric and stochastic quantization respectively. 91
Figure 5.4 CDM for the first three-channel RGB representation of the original reference and target images (patches), highlighting detected changes in white. 92

Figure 5.5 CDMs for the target image (patch), the generated results of deep-learning-based models, and quantized versions of MultiTempGAN including 8-bit DFPQ, LSSQ, Lloyd Max, K-Means, and the proposed 6-bit PLQ model. 93

LIST OF TABLES

Table 2.1 Performance metrics of the mentioned deep learning-based architectures across all three datasets. 14
Table 3.1 Details of the datasets used in this thesis work. 20
Table 4.1 Performance metrics of quantizing all weights approach for 2, 4, 6, 8, 16 and 32 bits averaged across all three datasets. 29
Table 4.2 Performance metrics of linear quantization approach for 6 and 8 bits across all three datasets. 32
Table 4.3 Performance metrics of the fixed-point quantization approach for 8 (2+6), 12 (3+9), and 16 (3+13) bits across all three datasets. . . 35
Table 4.4 Performance metrics of stochastic quantization approach for 6 and 8 bits across all three datasets. 39
Table 4.5 Performance metrics of symmetric quantization approach for 6 and 8 bits across all three datasets. 41
Table 4.6 Performance metrics of asymmetric quantization approach for 6 and 8 bits across all three datasets. 44
Table 4.7 Performance metrics of the best uniform approaches and all non-quantized deep learning-based models. SNR and LMSE values are averaged across all three datasets. The table includes bit precision, the number of side information parameters, memory usage (MB), and the compression ratio compared to MultiTempGAN, noted as Bits, S.I., Mem., and Ratio, respectively. 45
Table 4.8 Performance metrics of quantizing only convolutional layers approach for 2, 4, 6, 8, 16 and 32 bits averaged across all three datasets. 51
Table 4.9 Performance metrics of quantizing only non-convolutional layers approach for 2, 4, 6, 8, 16 and 32 bits averaged across all three datasets. 53
Table 4.10 Performance metrics of quantizing convolutional and non-convolutional layers to different bit widths approach where critical convolutional layers are quantized to 16 bits and other layers to 8 bits across all three datasets. 56

Table 4.11 Performance metrics of the LWPQ approach for different bit precision configurations across all three datasets. 59
Table 4.12 Performance metrics of the best mixed-precision approaches and all non-quantized deep learning-based models. SNR and LMSE values are averaged across all three datasets. The table includes bit precision, the number of side information parameters, memory usage (MB), and the compression ratio compared to MultiTempGAN, noted as Bits, S.I., Mem., and Ratio, respectively. 61
Table 4.13 Performance metrics of logarithmic quantization approach for 6 and 8 bits across all three datasets. 66
Table 4.14 Performance metrics of DFPQ quantization approach for 6 and 8 bits across all three datasets. 69
Table 4.15 Performance metrics of LSSQ quantization approach for 6 and 8 bits across all three datasets. 72
Table 4.16 Performance metrics of SDZQ approach for 4, 6, and 8 bits across all three datasets. 76
Table 4.17 Performance metrics of Lloyd Max approach for 6 and 8 bits across all three datasets. 79
Table 4.18 Performance metrics of K-Means approach for 6 and 8 bits across all three datasets. 81
Table 4.19 Performance metrics of PLQ approach for 4, 6, and 8 bits across all three datasets. 85
Table 4.20 Performance metrics of the best non-uniform approaches and all non-quantized deep learning-based models. SNR and LMSE values are averaged across all three datasets. The table includes bit precision, the number of side information parameters, memory usage (MB), and the compression ratio compared to MultiTempGAN, noted as Bits, S.I., Mem., and Ratio, respectively. S.I. for PLQ contains three different values due to the different number of quantization segments for each of the datasets. 86
Table A.1 Software dependencies and their corresponding versions used in this thesis work. 103

ABSTRACT

Post-Training Model Quantization Approaches in Compression of Multitemporal Multispectral Images

Naser SADEGHI GHARAGOZ

Department of Computer Engineering
Master of Science Thesis

Supervisor: Assoc. Prof. Dr. Ali Can KARACA

The vast amount of data generated by multispectral (MS) satellites makes image compression a critical task for remote sensing applications. Temporal correlations in multitemporal images, captured on different dates of the same scene, can be used for compression. This study builds upon MultiTempGAN, a lightweight generative adversarial network (GAN) designed for multitemporal MS image compression, and investigates the impact of three post-training quantization categories: uniform, mixed-precision, and non-uniform quantization. Representing model parameters with different bit-widths significantly reduces model size while preserving image quality. Experiments on Sentinel-2 satellite MS image pairs reveal that quantized models deliver competitive signal-to-noise ratio (SNR) and Laplacian mean square error (LMSE) while achieving improved bit-per-pixel (bpp) metrics compared to the original model, with minimal impact on image reconstruction and compression performance. These findings demonstrate the potential of post-training quantization to enhance GAN-based compression, ensuring efficient and scalable MS data handling without sacrificing accuracy.

Keywords: Multispectral image compression, generative adversarial networks, model quantization, model compression, multitemporal images.

YILDIZ TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

ÖZET

Çok Zamanlı Multispektral Görüntülerin Sıkıştırılmasında Eğitim Sonrası Model Kuantalama Yaklaşımları

Naser SADEGHI GHARAGOZ

Bilgisayar Mühendisliği Anabilim Dalı
Yüksek Lisans Tezi

Danışman: Doç. Dr. Ali Can KARACA

Çok bantlı (MS) uyduların ürettiği büyük veri miktarı, uzaktan algılama uygulamalarında görüntü sıkıştırmayı kritik hale getirmektedir. Aynı bölgenin farklı tarihlerde çekilmiş çok zamanlı görüntülerindeki zamansal korelasyonlar sıkıştırmada değerlendirilebilir. Bu çalışma, çok zamanlı MS görüntü sıkıştırma için hafif bir çekişmeli üretici ağ (GAN) olan MultiTempGAN'i temel alarak üç eğitim sonrası nicemleme yöntemini inceler: üniform, karışık hassasiyetli ve üniform olmayan nicemleme. Model parametrelerinin farklı bit değerleriyle temsil edilmesi, model boyutunu küçültürken görüntü kalitesini korur. Sentinel-2 MS görüntü çiftleriyle yapılan testler, nicemlenmiş modellerin orijinal modele kıyasla rekabetçi sinyal-gürültü oranı (SNR), Laplacian ortalama karesel hata (LMSE) sunduğunu, bit-başına piksel (bpp) metriklerini iyileştirdiğini ve sıkıştırmayla yeniden yapılandırma performansına minimum etki ettiğini göstermektedir. Sonuçlar, eğitim sonrası nicemlemenin GAN tabanlı sıkıştırmayı geliştirerek MS verilerinin verimli ve ölçeklenebilir işlenmesini sağladığını belirtmektedir.

Anahtar Kelimeler: Çok bantlı görüntü sıkıştırma, çekişmeli üretici ağlar, model nicemleme, model sıkıştırma, çok zamanlı görüntüler.

YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

1
INTRODUCTION

1.1 Context and Significance

Recently, the demand for remote sensing data that is used in various areas such as disaster monitoring, environmental change detection, and urban planning has resulted in an exponential growth in the size of multispectral (MS) and hyperspectral (HS) satellite image datasets. To demonstrate the extremely large size of the datasets created by space-borne satellites, it is worth mentioning that the Sentinel-2 satellite itself is predicted to produce more than 100 petabytes of data by 2030 [1]. As most of these applications require fast and accurate processing of image data, compression tends to be a vital element of them.

Accuracy and efficiency, particularly in terms of processing time, have always been crucial in image processing tasks. Challenges such as the storage, transmission, and processing of large MS data, especially in resource-constrained environments like satellite systems, have further emphasized the need for effective image data compression. Earlier studies conducted in the field of MS image compression have utilized traditional approaches such as JPEG2000 [2] with wavelet-based transforms and predictive approaches such as least mean squares (LMS) and recursive least squares (RLS) [3][4] recommended by the Consultative Committee for Space Data Systems (CCSDS) as the standard for compressing satellite images [5].

Recent studies have demonstrated important progress by utilizing deep learning approaches such as convolutional neural networks (CNNs), GANs, and autoencoders [6][7]. For instance, GAN-based approaches, such as the work by Agustsson et al. [8], showed effective compression at low bit rates, while CNN models like those by Valsesia and Magli [9] achieved efficient reconstruction consistent with standard compression protocols. These advances highlight the increasing importance of deep learning in tackling the challenges associated with large-scale MS data compression.

Advanced deep learning models, like Pix2Pix [10] and MultiTempGAN [11], use GANs to achieve high compression ratios for multitemporal MS data by taking the significance of temporal correlations between images that are captured over the same region but at different times unlike traditional MS image compression models like JPEG2000. MultiTempGAN, which is the paper that this study is built on, utilizes GANs to estimate the target image using the reference image. Figure 2.5 illustrates the structure of the generator and discriminator in MultiTempGAN.

1.2 Problem Formulation

While deep learning-based image to image translation models like U-Net [12], LinkNet [13], ResNet [14], Pix2Pix, and MultiTempGAN solve the issue of ignoring temporal correlations of multitemporal MS images present in traditional methods like JPEG2000, they still face limitations due to large parameter sizes and high computational demands. Especially in resource-constrained environments like space-borne satellites, where keeping the computational resources such as central processing units (CPUs) and graphical processing units (GPUs) up-to-date is impractical and costly, these limitations become more prominent. Similarly, transmitting these data to centers on Earth is also difficult due to their large size.

Briefly, in this study, different post-training quantization approaches were utilized to represent the model parameters (weights) with fewer bits to address the mentioned issues. This work evaluates the average performance of several quantization techniques that fall into the uniform, mixed-precision, and non-uniform quantization categories on three distinct Sentinel-2 satellite MS image datasets and introduces a much lighter model using piecewise linear quantization with 6-bit precision.

1.3 Research Goals

This thesis aims to address the limitations of deep learning models such as U-Net, LinkNet, and ResNet, as well as GAN-based models like MultiTempGAN, particularly their large parameter sizes and high computational demands, without compromising image quality or significantly impacting reconstruction and compression performance. The primary goal is to represent model parameters (weights) using fewer bits through various post-training quantization techniques, ensuring that the model remains efficient for resource-constrained environments such as satellite systems.

A significant goal of this study is to evaluate the effectiveness of three quantization categories: uniform, mixed-precision, and non-uniform quantization in compressing GAN models. By comparing these categories, the study seeks to determine which approach excels in general and to identify the best-performing methods and the most appropriate bit precision within each category. Metrics such as signal-to-noise ratio (SNR), bit-per-pixel (bpp), and Laplacian mean square error (LMSE) were used to assess the impact of quantization on image quality and compression efficiency, complemented by visual evaluation of regenerated images after quantization, compared to the reference and target images, as well as the generated images of other models, to ensure perceptual quality.

The study investigates how the weight distribution of each dataset influences the selection of quantization segments in piecewise linear quantization and its subsequent impact on metrics such as SNR, bpp, and LMSE after applying quantization. By analyzing these relationships, the study provides insights into optimizing quantization parameters based on dataset-specific weight distributions, ensuring improved performance and generalization for various remote sensing applications.

1.4 Overview of the Thesis

The structure of this thesis is as follows:

Section 2 covers the background and related work. It aims to provide an overview of the methods used in the literature.

Section 3 introduces the dataset and the data preparation process for this thesis.

Section 4 presents the quantization approaches and discusses the pros and cons of each thoroughly.

Section 5 presents the proposed model in this thesis and the reasons behind this decision.

Section 6 discusses the limitations and potential future work of this work.

Finally, Section 7 summarizes the entire process, highlights key achievements, and discusses expectations for future researches.

1.5 Ethics

The ethical principles were prioritized throughout the development and composition phases of this study. All three datasets used in this work are freely available on the Copernicus Open Access website and were previously utilized in the MultiTempGAN paper. Additionally, no personally identifiable information was included in the case studies, ensuring that the insights derived from this research do not compromise the integrity, privacy, or security of the original data. All interpretations were conducted ethically and without bias, supported by both mathematical calculations and visual results, minimizing the potential for errors.

2
BACKGROUND AND RELATED WORK

2.1 Background

Traditional image compression techniques, such as JPEG2000 (JP2K), have been widely used for MS image compression due to their ability to efficiently utilize spatial and spectral redundancies. JP2K employs the Discrete Wavelet Transform (DWT) to provide multi-resolution image representation, significantly improving compression performance while preserving image quality [15]. Configurations like combining DWT with JP2K or integrating Principal Component Analysis (PCA) with JP2K have further optimized the compression process. DWT + JP2K improves compression by highlighting fine details and reducing redundancy [16]. PCA + JP2K reduces the dimensionality of the data, reducing storage requirements without substantial loss of information [2].

Modern approaches to HS and MS image compression have shifted towards deep learning models to enhance compression efficiency while maintaining high image quality. For example, Kong et al. proposed an end-to-end multispectral image compression framework based on adaptive multiscale feature extraction, where CNNs extract spatial-spectral features and adaptively adjust feature weights, outperforming traditional methods like JP2K and 3D-SPIHT [17].

Similarly, Barman et al. present a method that compresses multiple images simultaneously to generate a common codebook. Using a deep learning-based autoencoder, the size of this codebook is reduced, leading to improved compression ratios while maintaining image quality. The experimental results demonstrate a 20% reduction in codebook size and an overall performance improvement of approximately 1.5% [18].

In the work of Anuradha et al., a hybrid deep learning-based compression approach for remote sensing images is presented. The authors propose a convolutional autoencoder (CAE) architecture that effectively reduces image dimensionality while

preserving essential features. The method integrates deep feature extraction with traditional entropy coding to enhance compression performance. The experimental results demonstrate that the proposed model achieves better compression ratios and reconstruction quality compared to conventional methods such as JP2K and SPIHT, making it a promising solution for efficient storage and transmission of remote sensing data [19].

Despite the effectiveness of the mentioned approaches, they do not explicitly account for temporal correlations in multitemporal MS data, limiting their compression efficiency when applied to datasets consisting of MS image pairs similar to those in this study. To address this limitation, Karaca et al. [11] proposed MultiTempGAN, a GAN-based model specifically designed to incorporate temporal information in multitemporal MS image compression tasks. MultiTempGAN evaluates different architectures, including U-Net [12], LinkNet [13], ResNet [14], and Pix2Pix [10], within its framework to determine the most effective model for MS image compression. U-Net and ResNet are widely used for extracting spatial features and reconstructing high-quality compressed images, while LinkNet's lightweight structure makes it particularly suitable for real-time applications. Pix2Pix, a conditional GAN model, specializes in image-to-image translation, achieving high compression performance even at low bit rates. MultiTempGAN builds on Pix2Pix by reducing the number of convolutional layers and introducing support for 12-channel input images. This modern technique addresses the limitations of traditional methods, particularly in handling the complexities of large-scale and temporally rich MS datasets.

2.2 Related Work

Since this thesis builds upon MultiTempGAN, the details of previous studies conducted in the literature, covering both traditional methods and deep learning-based architectures, including GANs and CNNs, are provided below, with a particular focus on those explored within MultiTempGAN. During the development of this thesis, additional deep learning-based approaches, including ResNet, U-Net, LinkNet, and MultiTempGAN, were re-trained and tested. The detailed results for the three datasets used in this study, together with their overall average, are reported in Section 2.2.6.

2.2.1 JPEG2000 (JP2K)

JP2K is a widely recognized image compression standard that employs DWT to provide multi-resolution image representation. Its encoder-decoder structure includes several key steps: the input image undergoes wavelet decomposition, followed by quantization and entropy coding in the encoder, while the decoder reverses these steps to reconstruct the image. [15] Although JP2K supports both lossy and lossless compression modes, it is mainly used as a lossy compression method due to its efficiency in reducing data size while maintaining acceptable image quality. The encoder-decoder structure of JP2K is illustrated in Figure 2.1. It is worth mentioning that the quantization used in JP2K, while serving the same general purpose as model quantization (namely, reducing the precision of numerical values to save storage and computational resources), is fundamentally different in its application. In JP2K, quantization is applied to wavelet coefficients, which represent the image in the transform domain. In contrast, this thesis work applies quantization to the parameters (weights) of the GAN model or neural network to reduce the model size, as explained in Section 4.

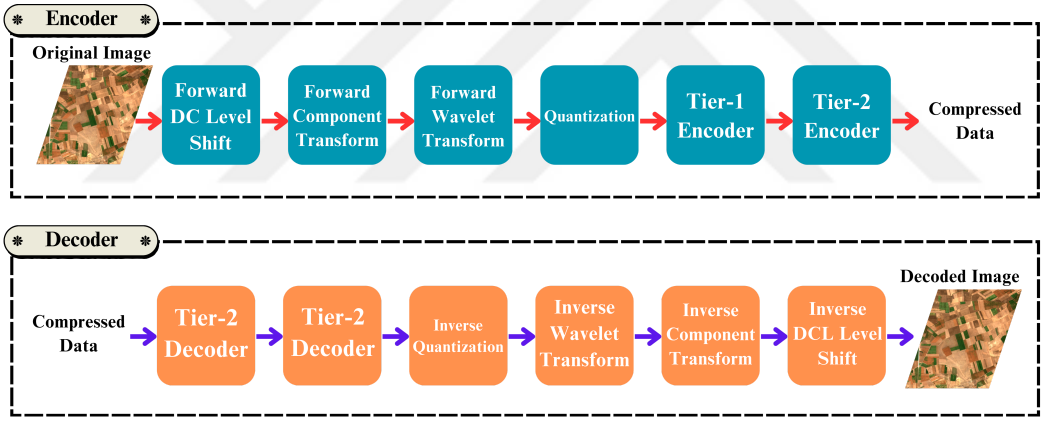


Figure 2.1 Core structure of encoder and decoder in JP2K [15].

Enhanced configurations of JP2K have been developed to address specific compression challenges. The combination of DWT with JP2K (JP2K+DWT) [16] improves compression performance by leveraging the ability of DWT to highlight fine details and reduce spatial redundancies. This method improves the rate distortion efficiency of JP2K by improving the representation of important image characteristics, making it particularly useful for datasets with high spatial variability.

Similarly, PCA has been integrated with JP2K (JP2K+PCA) [2] to further optimize compression for MS images. PCA reduces the dimensionality of the spectral data by retaining only the most significant principal components, minimizing storage

requirements without substantial loss of information. This technique complements JP2K by reducing redundancy in spectral bands, leading to better compression ratios and rate distortion performance.

In the context of MS image compression, JP2K has been widely applied to leverage its ability to handle both spatial and spectral redundancies. The work of Karaca et al. [20] highlights the effectiveness of the method in compressing MS images by combining JP2K with additional preprocessing techniques to improve spectral decorrelation and overall compression performance. Additionally, they explored the use of super-resolution methods, such as Super-Resolution Residual Network (SRResNet) and Enhanced Deep Super-Resolution Network (EDSR), after JP2K compression to improve the resolution of compressed images and reduce artifacts. This approach is particularly suited for scenarios where large volumes of MS data must be efficiently stored and transmitted while preserving essential information for analysis.

2.2.2 ResNet

Residual Networks (ResNet) [14] are commonly used in image processing, including MS image compression, due to their ability to overcome the vanishing gradient problem. ResNet achieves this by incorporating residual blocks, which directly add the input to the output after convolutional operations, enabling deeper architectures and better gradient flow.

The ResNet generator for MS image compression, as shown in Figure 2.2, consists of three main sections: encoding, residual processing, and decoding. The encoding section extracts features using convolutional layers, while the core of the network (the residual blocks) processes these features. Each residual block includes two convolutional layers, batch normalization, and rectified linear unit (ReLU) activation, with a skip connection that combines the input and output of the block. This design ensures efficient learning and robust feature extraction.

The architecture demonstrated in Figure 2.2 illustrates the generator structure of ResNet, featuring a total of 2,884,044 parameters. It consists of convolutional layers (conv1–conv3), deconvolutional layers (deconv1–deconv3), and residual blocks (RB1–RB9), where each residual block contains two convolutional layers.

For each block or layer, the parameters are represented by three distinct numbers: the number of weights displayed at the top, the number of biases shown in orange below or next to it, and the number of batch normalization parameters at the bottom under the layer illustration. This parameter representation is consistently

used across all other architectures, including U-Net, LinkNet, MultiTempGAN, and PLQ-MultiTempGAN.

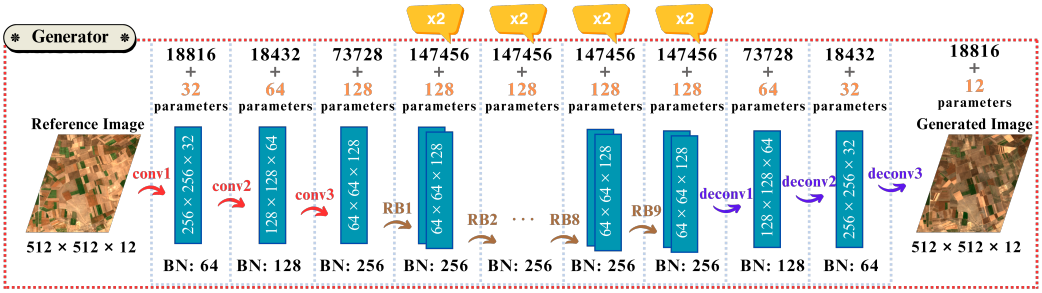


Figure 2.2 ResNet Architecture. [11].

The decoding section reconstructs the compressed image using deconvolutional layers. This structure enables ResNet to effectively compress MS images while maintaining high-quality reconstruction. To optimize the model for MS datasets, parameters like the number of filters were adjusted to achieve a balanced bit rate (0.0665 bpp), which, despite being slightly above MultiTempGAN and significantly higher than the bpp result achieved in this thesis, is still comparable and acceptable. Detailed ResNet performance metrics are demonstrated in Section 2.2.6

2.2.3 U-Net

U-Net is a popular deep learning architecture with a primary focus on addressing the challenge of limited annotated data in the medical field and was introduced in the work of Ronneberger et al. [12] for the first time. It is widely used in segmentation, image-to-image translation, and satellite imaging tasks. As illustrated in Figure 2.3, its generator structure featuring 2,560,252 parameters consists of convolutional and deconvolutional layers with concatenation-based skip connections marked as yellow arrows. Skip connections play a significant role in effectively combining features from the encoder and decoder paths. The advantages of having skip connections can be summarized like below:

- Preserving Spatial Information: Skip connections reintroduce high-resolution spatial details from the encoder (convolutional layers) to the decoder (corresponding deconvolutional layers or relevant convolutional layers), compensating for the information lost during down-sampling.
- Enabling Accurate Reconstruction: By merging early feature maps with decoder features, skip connections help in recovering fine-grained details for precise output generation.

- Mitigating Vanishing Gradient Problems: Skip connections facilitate smoother gradient flow, stabilizing training, and enhancing convergence in deeper networks.
- Combining Semantic and Contextual Information: Skip connections integrate high-level semantic and low-level contextual features, ensuring that outputs are both meaningful and detailed.
- Avoiding Information Bottlenecks: Directly linking encoder and decoder layers bypasses the compression bottleneck, allowing efficient information transfer for improved performance.

The benefits of concatenation-based skip connection have encouraged the authors of MultiTempGAN [11] paper to use this in the architecture of the proposed model's generator, since MultiTempGAN's generator (and consequently the generator in this study) is based on U-Net. It is worth mentioning that unlike LinkNet where the features of each encoder layer are directly added to its corresponding decoder layer through addition-based skip connection, in U-Net, Pix2Pix, MultiTempGAN, and this study concatenation-based skip connections along the channel dimension are used that focus on transferring as much information as possible at the cost of increased computational complexity and memory usage. These settings result in a bpp of 0.059 that is almost 11% lower than the bpp of ResNet. Detailed U-Net performance metrics are shown in Section 2.2.6.

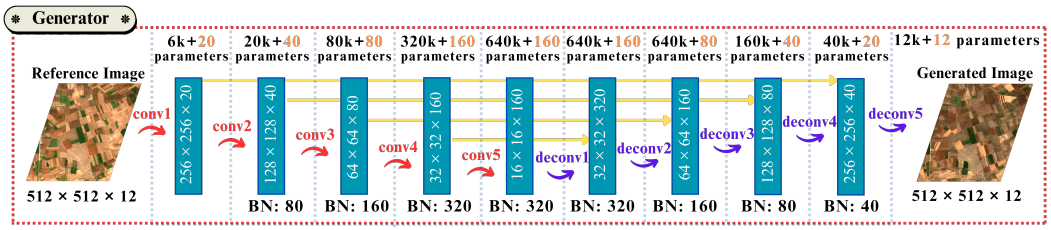


Figure 2.3 U-Net Architecture [11].

2.2.4 LinkNet

LinkNet is a relatively light-weight deep neural network architecture that is designed to be used mainly in semantic segmentation. Its generator structure is so close to that of U-Net's with this difference that it uses element-wise addition-based skip connections between encoder (convolutional layers) and decoder (deconvolutional layers). The generator structure which consists of 2,580,876 parameters is depicted in Figure 2.4. Although its parameter count is lower than ResNet's, it is slightly higher than U-Net's. This slight increase in

parameter count, although negligible, results in slightly higher memory usage and bpp (0.0593) values compared to U-Net. However, the marginal increase in the bpp metric was not the primary reason for not selecting this architecture for the generator of this thesis work; rather, it was the weaker performance in terms of SNR and LMSE compared to MultiTempGAN [11], as shown in Section 2.2.6.

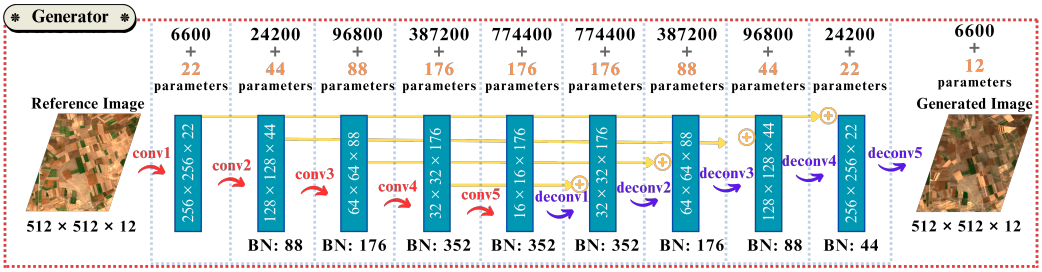


Figure 2.4 LinkNet Architecture[11].

2.2.5 MultiTempGAN

MultiTempGAN [11] is a lightweight and accurate GAN model designed for the compression of multitemporal MS images, as proposed by Karaca et al. As mentioned in the previous sections, MultiTempGAN's generator structure is based on both Pix2Pix and U-Net, with key modifications including a significant reduction in the number of convolutional layers to lower complexity and the addition of support for input images with dimensions of $512 \times 512 \times 12$ (12 channels) to handle MS satellite images. With this model, the number of parameters in the original Pix2Pix model, which was nearly 54 million for $256 \times 256 \times 3$ images, is reduced to 2,580,876 for $512 \times 512 \times 12$ images. This reduction makes the model much more efficient and easier to implement, particularly for the image compression task. Lowering the original Pix2Pix model's convolutional layers from 8 to 5 in the generator and using a 5×5 filter with a stride of 2 for each convolutional layer are other notable aspects of this model's architecture.

As demonstrated in Figure 2.5, this approach not only minimizes the number of generator parameters, as discussed in Section 2.2.6, but also achieves the best performance among the deep learning-based approaches mentioned so far. In MultiTempGAN and similar GAN-based image compression models, the generator is the main focus because it directly impacts the model's efficiency and performance during inference. The discriminator, in contrast, is only relevant during training, so its parameter count is less critical and does not affect the final compression performance metrics such as bpp or memory usage, and that is why it is not included.

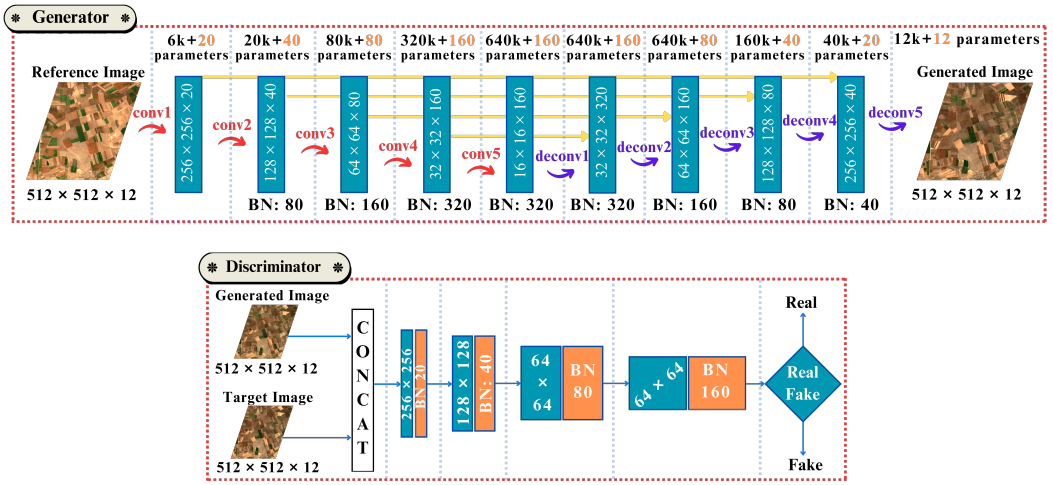


Figure 2.5 MultiTempGAN architecture with the generator at the top and the discriminator at the bottom, processing concatenated real and generated images. Discriminator's convolutional layers are shown without parameter counts, as they are less critical for compression. The final layer outputs real/fake probabilities [11].

2.2.6 Performance Comparison of the Related Works

As mentioned above, for the purpose of performance comparison between other deep learning-based models and this work, the models were implemented, and their performance metrics are reported in Table 2.1.

The key metrics that play an important role in the selection of one model over another are SNR, bpp, memory, and LMSE. Below, the formulas for these metrics are presented along with their descriptions.

Signal-to-Noise Ratio (SNR): SNR measures the ratio of the signal power to the noise power, indicating the quality of the reconstructed image. It is given by:

$$SNR = 10 \times \log_{10} \left(\frac{\sum_{i=1}^{N} x_i^2}{\sum_{i=1}^{N} (x_i - \hat{x}_i)^2} \right) \tag{2.1}$$

where x_i represents the pixel values of the original image, $\hat{x}_i$ represents the pixel values of the reconstructed (generated) image, and N is the total number of pixels. The total number of pixels is calculated as:

$$N = \#\text{pixels} \times \#\text{images} \tag{2.2}$$

For this study, where each image has $512 \times 512 \times 12$ dimensions and each of the

three datasets contains 441 images, the total number of pixels becomes $N = (512 \times 512 \times 12) \times 441 = 1,387,266,048$.

Bits Per Pixel (bpp): bpp measures the number of bits used to encode each pixel in the compressed image and is calculated as:

$$bpp = \frac{\#\text{parameters} \times \text{bit precision}}{\#\text{pixels} \times \#\text{images}} \tag{2.3}$$

Here, #parameters refers to the total number of generator parameters, each stored with a specific precision (by default with 32 bits). #pixels represents the total number of pixels in an MS image (not the total number of pixels that was represented with N in Equations 2.1 and 2.2), calculated as $512 \times 512 \times 12$, and #images refers to the total number of pictures (patches) in the dataset, which is 441.

Memory Usage: Memory refers to the total storage required for the model parameters and is computed as:

$$\text{Memory (MB)} = \frac{\#\text{parameters} \times \text{bit precision}}{8 \times 1024 \times 1024} \tag{2.4}$$

To represent memory in megabytes (MB), the total storage in bits is divided by $8 \times 1024 \times 1024$.

Laplacian Mean Squared Error (LMSE): LMSE evaluates the error between the original and reconstructed images based on Laplacian-filtered pixel values, providing insight into fine details. It is given by:

$$LMSE = \log_{10}\left(\frac{\sum_{i=1}^{N}(\mathcal{L}(x_i) - \mathcal{L}(\hat{x}_i))^2}{N}\right) \tag{2.5}$$

where $\mathcal{L}(x_i)$ and $\mathcal{L}(\hat{x}_i)$ are the Laplacian-filtered original and reconstructed pixel values, respectively, and N is the total number of pixels.

These metrics collectively provide a comprehensive evaluation of the performance of the models under consideration, helping to assess their trade-offs between compression efficiency and image quality.

Table 2.1 Performance metrics of the mentioned deep learning-based architectures across all three datasets.

Method	Dataset	Memory (MB)	bpp	SNR	LMSE
ResNet (Parameters: 2,884,044)	MSI Pair-1	11.00	0.0665	23.65	0.71
	MSI Pair-2	11.00	0.0665	22.10	0.60
	MSI Pair-3	11.00	0.0665	20.94	0.62
	Avg.	**11.00**	**0.0665**	**22.23**	**0.65**
U-Net (Parameters: 2,560,252)	MSI Pair-1	9.76	0.0590	27.31	0.51
	MSI Pair-2	9.76	0.0590	24.85	0.40
	MSI Pair-3	9.76	0.0590	24.30	0.43
	Avg.	**9.76**	**0.0590**	**25.49**	**0.45**
LinkNet (Parameters: 2,580,876)	MSI Pair-1	9.84	0.0595	26.10	0.62
	MSI Pair-2	9.84	0.0595	24.80	0.41
	MSI Pair-3	9.84	0.0595	23.57	0.44
	Avg.	**9.84**	**0.0595**	**24.82**	**0.49**
MultiTempGAN (Parameters: 2,560,252)	MSI Pair-1	9.76	0.0590	27.87	0.46
	MSI Pair-2	9.76	0.0590	26.43	0.32
	MSI Pair-3	9.76	0.0590	24.72	0.39
	Avg.	**9.76**	**0.0590**	**26.34**	**0.39**

The performance metrics presented in Table 2.1 clearly demonstrate that MultiTempGAN outperforms all other models in terms of SNR, bpp, and LMSE. Additionally, it has the minimum number of generator parameters (shared with U-Net), making it the most suitable choice as the base model for this study. As previously mentioned, the number of discriminator parameters is not highly significant in a compression problem. However, it is worth noting that incorporating the discriminator into the structure affects runtime, although this impact is negligible as it only applies during training. Figures 2.6 to 2.9 show the training time for each deep learning-based approach over 150 epochs across the three datasets.

As demonstrated in Figure 2.9, in MultiTempGAN, the generator and discriminator losses do not necessarily need to reach the lowest possible values; instead, they need to stabilize, maintaining a balance during training. Despite slightly higher losses compared to other models, MultiTempGAN achieves better performance (higher SNR and lower LMSE) due to its modified Pix2Pix structure, which is designed to

preserve spatial and spectral consistency, resulting in superior feature representation and generalization for MS image datasets.

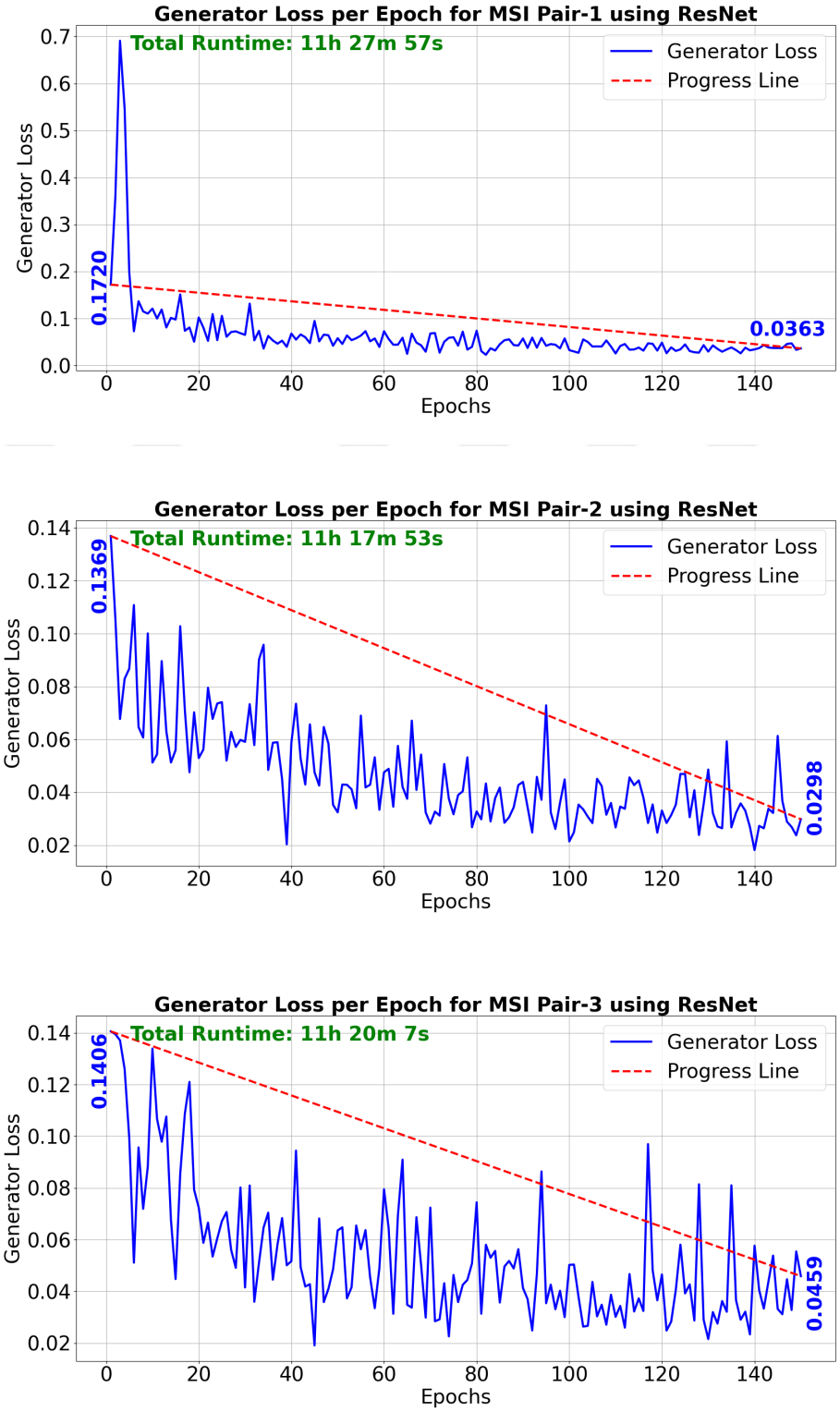


Figure 2.6 Total runtime and generator loss per epoch for ResNet.

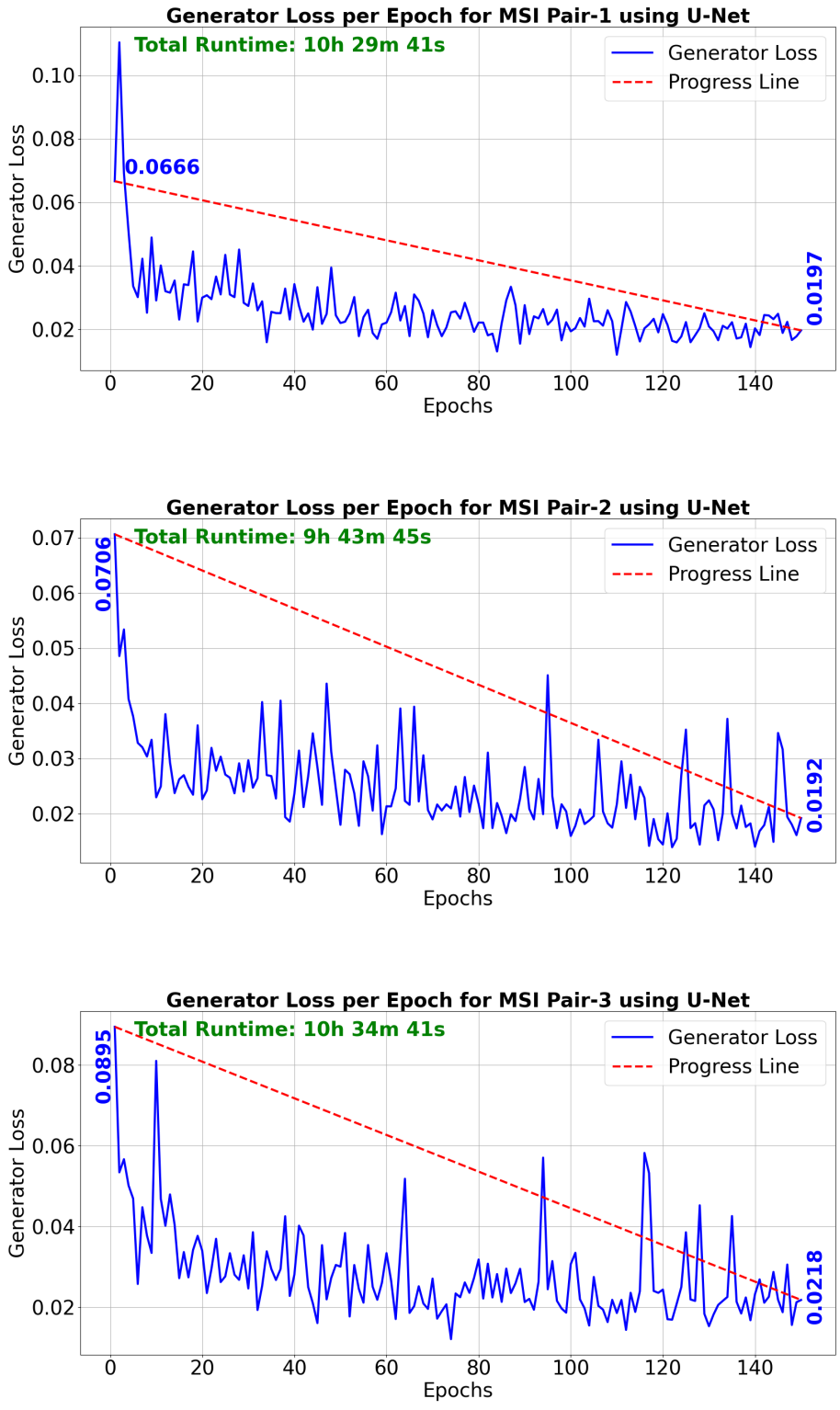


Figure 2.7 Total runtime and generator loss per epoch for U-Net.

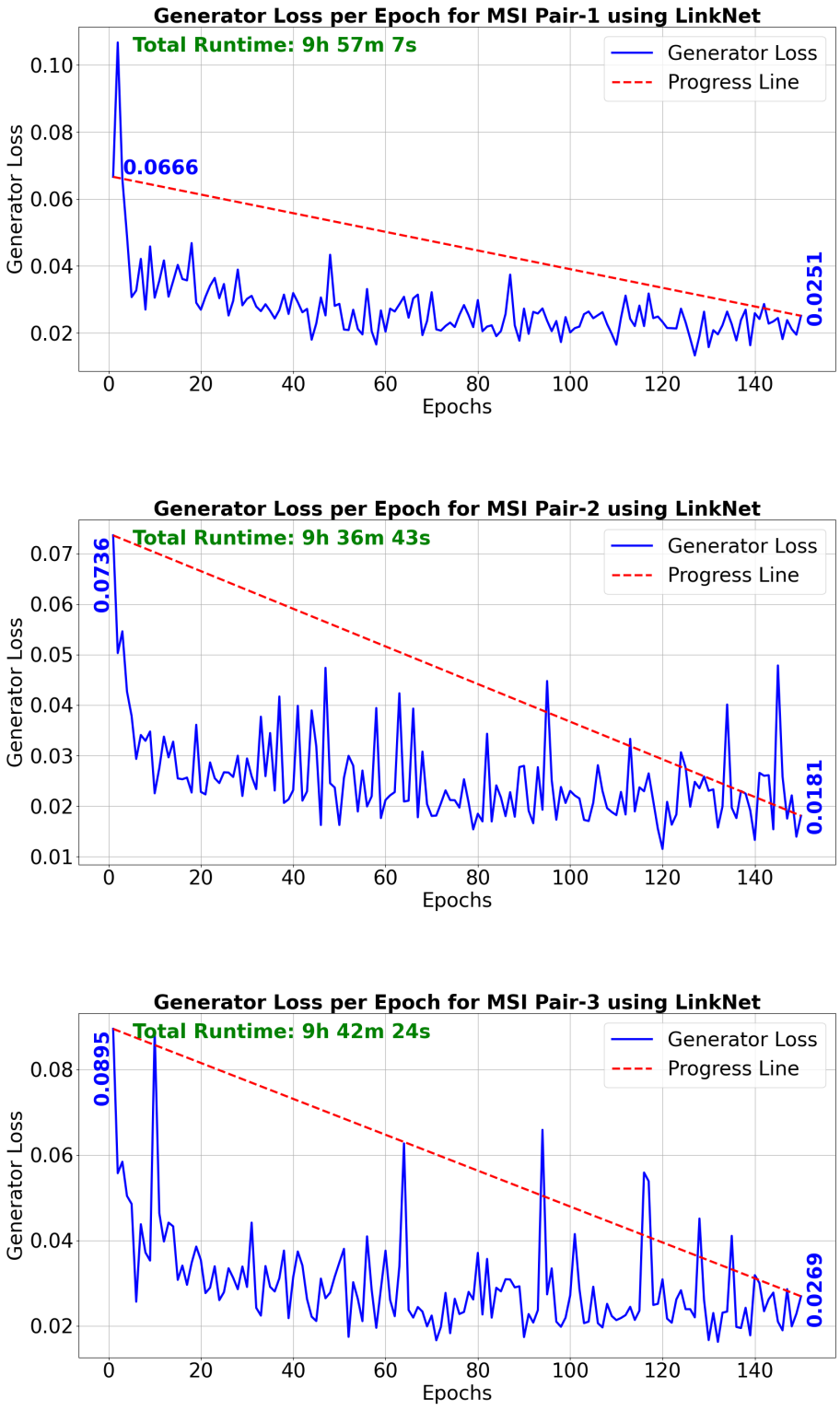


Figure 2.8 Total runtime and generator loss per epoch for LinkNet.

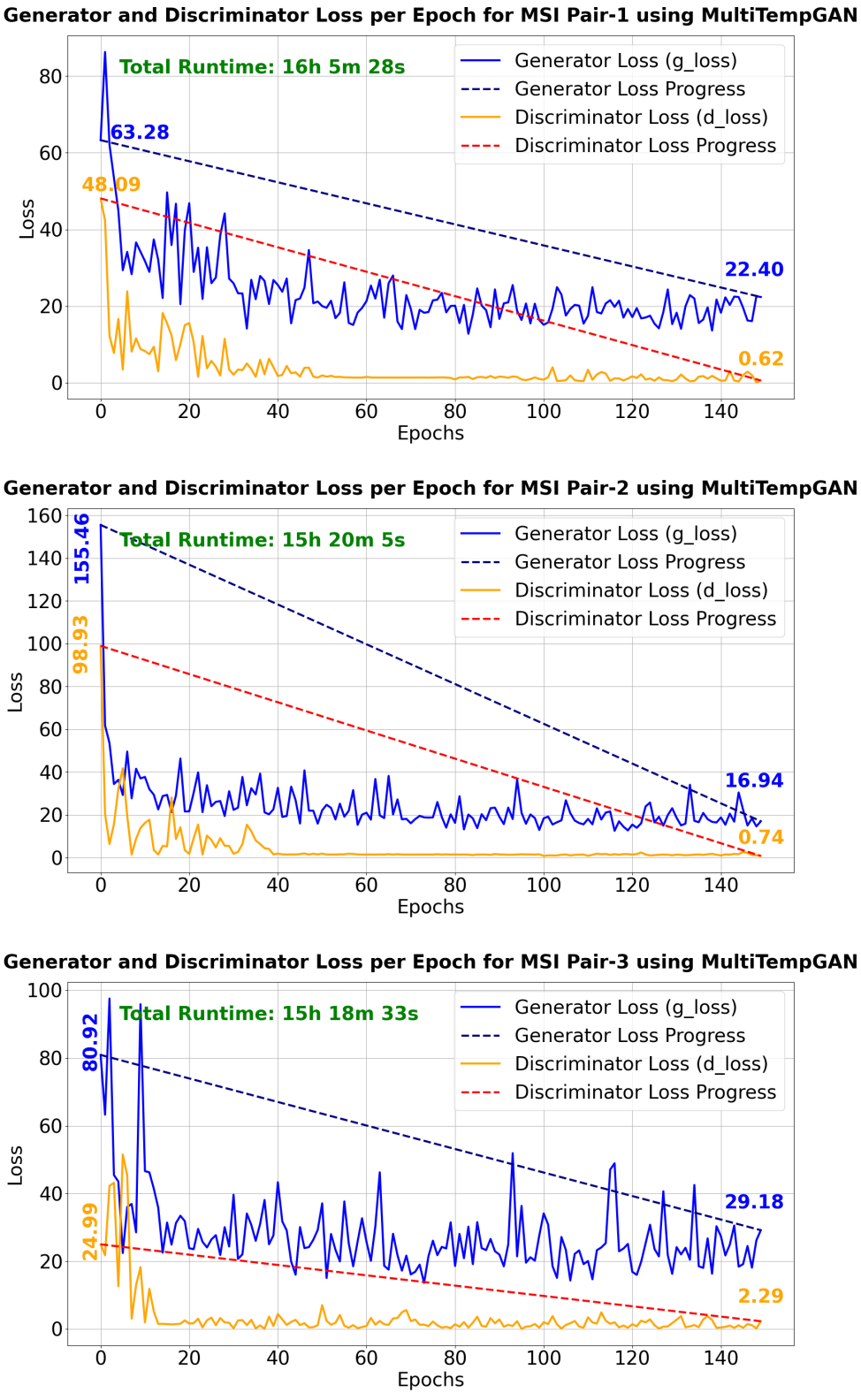
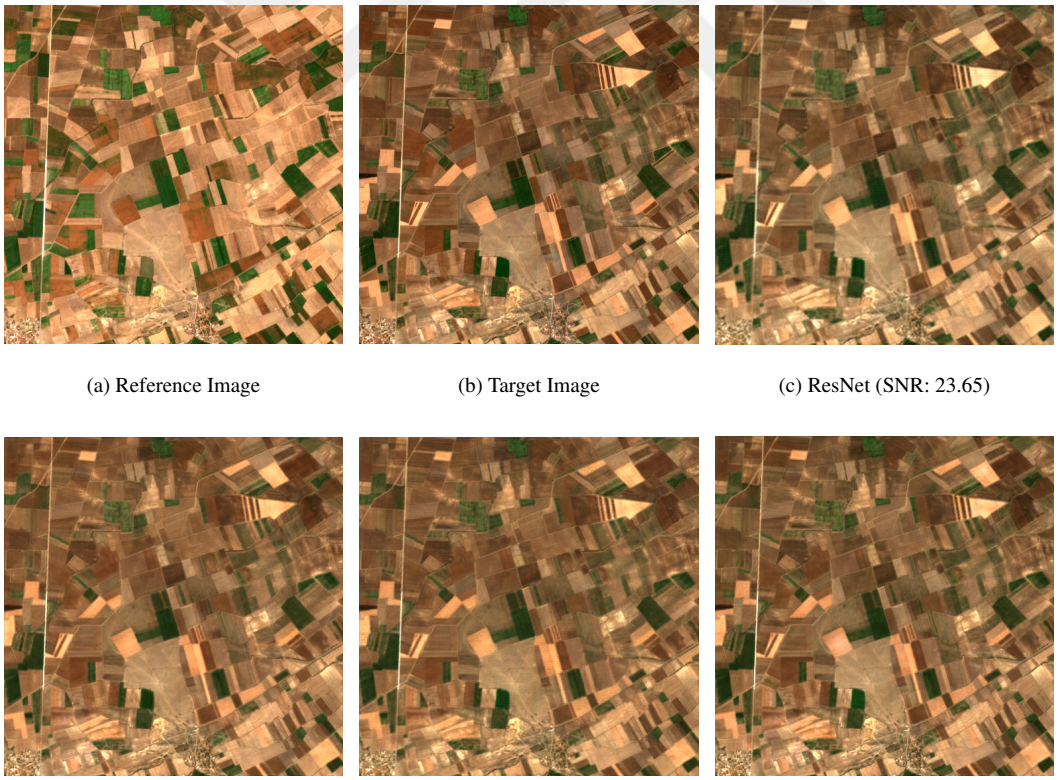


Figure 2.9 Total runtime, generator, and discriminator loss per epoch for MultiTempGAN.

To sum up the related work section, it is worth mentioning that the superior performance of MultiTempGAN is also visually observable in Figure 2.10, which

depicts the reference and target images for a randomly selected sample from one of the datasets, along with the generated images using ResNet, U-Net, LinkNet, and MultiTempGAN for the same sample. Of course, only the first three channels are visualized due to the impracticality of displaying all 12 channels. All of the mentioned details played a significant role in selecting MultiTempGAN as the basis of this thesis work and trying to make it even more compact using different post-training quantization approaches with minimum impact on the quality of the generated images.

In Figure 2.10, the SNR value displayed under each image corresponds to the specific SNR of the dataset and the approach from which the image (patch) is derived. Consequently, these values may differ from the average SNR reported in Table 2.1. This applies to all visual comparisons presented throughout this thesis. Also, it is worth mentioning that all visual results in this work, after training and sampling, have undergone a uniform lightening process by multiplying pixel values by 8 and clipping to $[0, 1]$ to ensure the intensity remains within the valid range, without affecting compression performance.



(a) Reference Image (b) Target Image (c) ResNet (SNR: 23.65)

(d) U-Net (SNR: 27.31) (e) LinkNet (SNR: 26.10) (f) MultiTempGAN (SNR: 27.87)

Figure 2.10 Visual comparison of reference, target, and generated images using ResNet, U-Net, LinkNet, and MultiTempGAN.

3
Datasets and Data Preparation

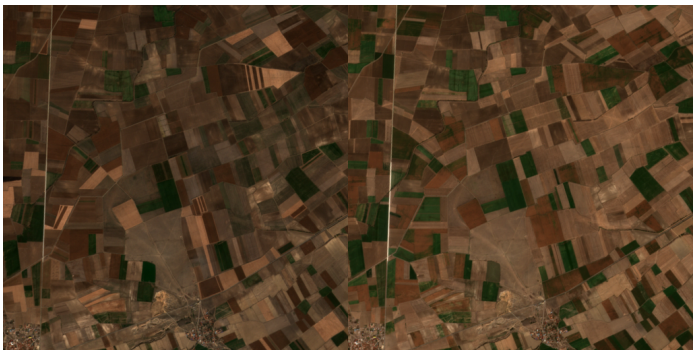
3.1 Datasets

In this work, the same three datasets that were used to train the models in MultiTempGAN were utilized. Each of the datasets consists of a single Sentinel-2 satellite MS image pair captured on different dates from various regions of Türkiye. Each MS image pair has a reference image on the right and a target image on the left. It is worth mentioning that the reference image is sensed on an earlier date than the target image. As Sentinel-2 satellite images are so large, each of them is first up-sampled to 10 m spatial resolution [21] using the SNAP toolbox [22] and then divided into 441 patches that do not overlap. Each patch has $512 \times 512 \times 12$ dimension and the pair $1024 \times 512 \times 12$ as a result of the combination. For better visual representation, each pair is saved as four $1024 \times 512 \times 3$ RGB images in PNG format where each RGB image represents 3 channels of the image.

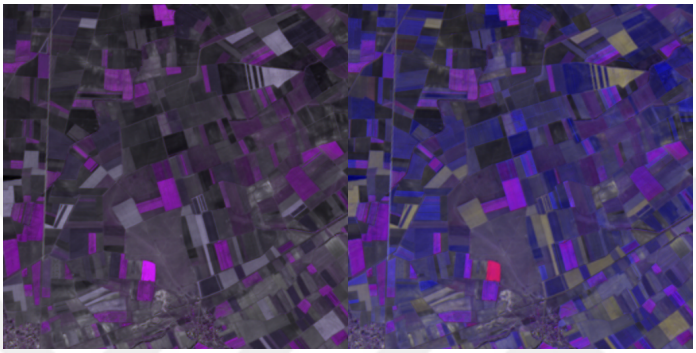
Figure 3.1 demonstrates the four $1024 \times 512 \times 12$ RGB representations for a sample patch pair chosen randomly from one of the datasets, and Table 3.1 contains additional details about all three datasets. These images are freely accessible and can be downloaded from the Copernicus Open Access Hub [23].

Table 3.1 Details of the datasets used in this thesis work.

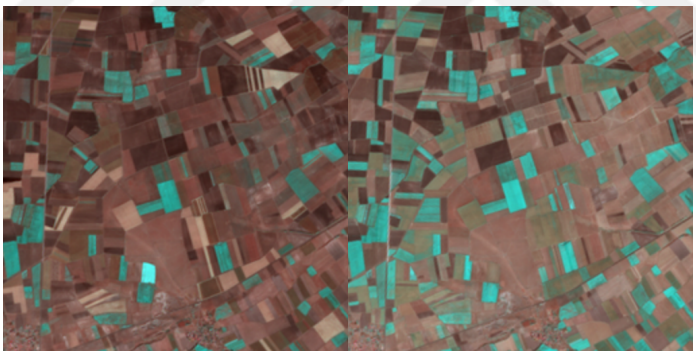
Dataset	Sensing Region	Sensing Dates (Reference - Target)
MSI Pair-1	Between Eskişehir and Konya	28.09.2020 – 18.10.2020
MSI Pair-2	Between Denizli and Muğla	14.05.2021 – 19.05.2021
MSI Pair-3	Between Balıkesir and İzmir	05.08.2020 – 25.08.2020



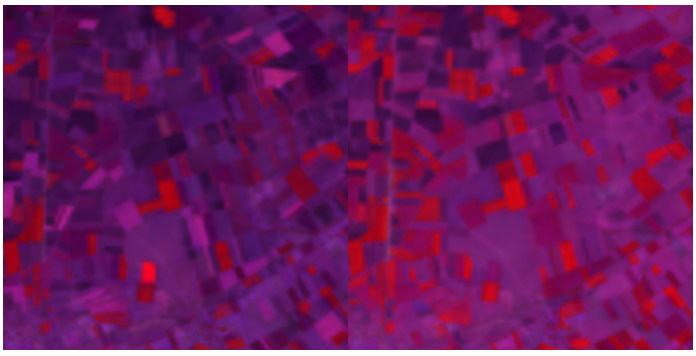
(a) Channels 1, 2, and 3 of the sample patch pair.



(b) Channels 4, 5, and 6 of the sample patch pair.



(c) Channels 7, 8, and 9 of the sample patch pair.



(d) Channels 10, 11, and 12 of the sample patch pair.

Figure 3.1 RGB representation of an MS image patch pair, where each set of three channels is visualized as a single image.

3.2 Data Preprocessing

The following preprocessing approaches have been implemented in all three datasets prior to the training process.

3.2.1 Normalization

The pixel values of the image patches were normalized to the range [-1, 1]. This helps the model to potentially train more efficiently by standardizing input distributions and avoiding numerical instabilities.

3.2.2 Concatenation

During the model training process, all four $512 \times 512 \times 3$ RGB representations of the reference image patch and four $512 \times 512 \times 3$ RGB representation of the target image patch were concatenated along the channel axis (axis = 2) to shape a final tensor of $512 \times 512 \times 24$ with the first 12 channels representing the target and the second 12 channels the reference patch.

The data loading function processes these image patches by first extracting individual RGB components of the reference and target images from multiple input files. Each image is normalized by scaling its pixel values to the range $[-1, 1]$ using a factor of 8191.5 to ensure numerical stability during training. Finally, the normalized images are concatenated along the channel axis to form the final tensor, preserving the spatial dimensions while integrating spectral and temporal information.

3.2.3 Image Splitting

Every 3-channel representation of each image patch pair with dimensions of $1024 \times 512 \times 3$ is split into two halves where the left half is related to the target and the right half to the reference image patch.

3.2.4 Batch Preparation

Batch preparation ensures that the data have the correct shape for the model by adding a batch dimension, converting a single sample image (patch pair) from the shape (512, 512, 24) to (1, 512, 512, 24). This step aligns the data format with the model's requirement to process batches, even if the batch size is just one image. It also enables easy scaling for training multiple images simultaneously.

3.2.5 Data Augmentation (Horizontal Flipping)

Horizontally flipping the image patches with probability 50% during training enhances the datasets and introduces variations with the potential to improve the generalization of the model.

4
Quantization Approaches

4.1 Quantization

Quantization in deep learning-based image processing models, such as ResNet, LinkNet, U-Net, or MultiTempGAN, refers to reducing the precision of numerical values (e.g., weights, biases, and activations) in a neural network from higher precision (commonly 32-bit floating-point, FP32) to lower precision (e.g., 16-bit floating-point, FP16, or 8-bit integers, INT8) [24]. This process greatly improves model efficiency by reducing its memory footprint and computational requirements.

In a broad categorization, quantization can be divided into two main categories: Post-Training Quantization [25] and Quantization-Aware Training [26]. These categories differ in their application and the stage at which quantization is introduced. Post-Training Quantization is applied to a fully trained model, making it a quick and efficient option, while Quantization-Aware Training incorporates quantization during the training process, allowing the model to adapt to precision loss and improve accuracy. Below, each approach is described in detail:

- **Post-Training Quantization**: This method involves applying quantization to a fully trained model without retraining. It is simple and hardware-friendly, making it ideal for deployment on resource-constrained devices. It is particularly useful for deployment on devices such as mobile phones, edge devices, embedded systems, and satellite systems.
- **Quantization-Aware Training**: This method introduces quantization during the training process. The model simulates low-precision arithmetic during training, allowing it to adapt to precision loss and minimize accuracy degradation.

For the sake of this study, Post-Training Quantization was implemented and the base model of MultiTempGAN was quantized using various post-training

quantization techniques that can be mainly categorized to uniform, mix-precision and non-uniform quantization. The explanation of each quantization technique, the model architectures, the performance metrics for each category, and their specific techniques, as well as the visual results for each technique, are presented in the upcoming sections.

It is worth noting that post-training quantization introduces additional parameters such as scales, zero points, and related values, referred to as side information. While performance metrics such as memory, SNR, and LMSE are calculated using the same equations mentioned in Section 2.2.6 for the quantized model, the calculation of bpp is slightly modified due to the inclusion of side information parameters. These additional parameters, represented with 32-bit precision, are accounted for in the calculation as shown below.

$$bpp = \frac{(\#\text{q_params} \times \text{bit precision}) + (\#\text{side_info_params} \times 32)}{\#\text{pixels} \times \#\text{images}} \tag{4.1}$$

Here, #q_params refers to the total number of quantized generator parameters, each stored with a chosen bit precision (e.g., 6 bits). #side_info_params represents the total number of additional side information parameters introduced during post-training quantization, each stored with 32-bit precision. #pixels represents the total number of pixels in an MS image, calculated as $512 \times 512 \times 12$, and #images refers to the total number of pictures (patches) in the dataset, which is 441.

The side information parameters also slightly affect the memory usage of the model. Therefore, the formula given in Equation 2.4 gets updated as follows:

$$\text{Memory (MB)} = \frac{(\#\text{q_params} \times \text{bit precision}) + (\#\text{side_info_params} \times 32)}{8 \times 1024 \times 1024} \tag{4.2}$$

#q_params refers to the total number of quantized generator parameters, each stored with a chosen bit precision (e.g., 6 bits). To represent memory in megabytes (MB), the total storage in bits is divided by $8 \times 1024 \times 1024$.

4.2 Uniform Quantization

Uniform quantization refers to quantizing all model parameters or weights to the same fixed bit width across the entire model [24]. This method is relatively straightforward to implement and is highly compatible with hardware, since it

requires minimal customization. However, it can negatively affect layers that are more sensitive to precision loss.

The model architecture for uniform quantization (with symmetric quantization and 8-bit precision, which has the best performance among all uniform approaches) is illustrated in Figure 4.9. In this thesis work, several uniform quantization approaches were implemented and their performances are reported in Table 4.7 that shows the detailed performance metrics of all uniform techniques tested during the development of this work.

4.2.1 Quantizing All Weights

Quantizing all weights involves applying uniform quantization to every parameter in the model. This method simplifies the implementation, making it highly compatible with hardware architectures. However, it may degrade performance in layers that are more sensitive to precision loss due to uniform quantization.

- **Bit-width Selection**: The desired bit precision (e.g., 8 bits) is set to determine the precision of quantization.
- **Scaling Factor**: The weights are scaled using a factor derived from the maximum value (α) of the weights. This ensures that the values are within the quantization range.
- **Clipping**: The scaled weights are clipped to the valid range of the chosen bit precision, preventing outliers from exceeding the representable range.
- **Reconstruction**: The quantized weights are dequantized back to floating-point values using the same scaling factor for further processing.

4.2.1.1 Quantizing All Weights Algorithm

Algorithm 1 provided below demonstrates the process step by step using this technique.

Algorithm 1 Quantizing All Weights

1: **Input:** Model checkpoint (`checkpoint_dir`), bit-width (`bitval`)
2: **Output:** Quantized model checkpoint with updated weights
3: Load the model checkpoint from `checkpoint_dir`
4: Retrieve the list of variables in the checkpoint
5: Initialize an empty list `assign_ops` to store weight updates
6: **for** each variable `var` in the checkpoint **do**
7: Load the tensor values of `var` from the checkpoint
8: Compute the maximum value $\alpha = \max(\texttt{var})$
9: Calculate the scaling factor: $\texttt{scale} = \frac{(2^{\texttt{bitval}}-1)}{\alpha}$
10: Quantize weights: $\texttt{quantized_weights} = \texttt{round}(\texttt{var} \times \texttt{scale})$
11: Clip quantized weights to the range: $[-2^{\texttt{bitval}-1}, 2^{\texttt{bitval}-1} - 1]$
12: Dequantize weights: $\texttt{rescaled_weights} = \frac{\texttt{quantized_weights}}{\texttt{scale}}$
13: Convert `rescaled_weights` to `float32` (required due to TensorFlow's limitation of saving variables as `float32`)
14: Create a TensorFlow variable with the updated weights and append it to `assign_ops`
15: **end for**
16: Initialize a TensorFlow session and run `assign_ops` to update all variables
17: Save the quantized model checkpoint at a specified path

In lines 11 and 12 of the algorithm above, dequantization is required due to TensorFlow 1.15's limitation in only accepting variables in float32 format. However, despite the dequantization and the re-conversion of the quantized variables to float32, the quantization remains effective because the values stored as float32 are still quantized and retain the reduced precision introduced during the quantization process. During inference, hardware such as Tensor Processing Units (TPUs) or GPUs can utilize these quantized values in lower bit precision (e.g., INT8,), resulting in reduced memory usage and faster computations.

For example, an original weight value of 0.00655927 is considered, with a maximum value $\alpha = 0.3763316$. A bit width of 8 bits is assumed and the scaling factor is calculated as:

$$\text{scale} = \frac{2^8 - 1}{\alpha} = \frac{255}{0.3763316} \approx 258.058 \tag{4.3}$$

The weight value is then quantized as:

$$\text{Quantized Value} = \text{round}(0.00655927 \times 258.058) = \text{round}(1.692) = 2 \quad (4.4)$$

Next, the quantized value is clipped to fit within the 8-bit range, $[-128, 127]$. Since 2 is already within the range, no clipping is necessary. During dequantization, the clipped value is scaled back to float32 as follows:

$$\text{Dequantized Value} = \frac{2}{258.058} \approx 0.00775 \quad (4.5)$$

This dequantized value (0.00775) is stored as float32 in the checkpoint but still reflects the quantized representation. Consequently, the quantization remains effective, reducing computational complexity and improving efficiency during inference. Furthermore, the model becomes more compact in its effective representation, as fewer bits are used to quantize the weights, which is reflected in a drop in the bpp metric, even though the checkpoint size may not shrink significantly due to the storage format being float32.

Table 4.1 contains the performance metrics of quantized MultiTempGAN using the quantizing all weights approach, and Figure 4.1 visually compares the reference, target, MultiTempGAN's result, and quantized MultiTempGAN's results with 2, 4, 6, 8, 16, and 32 bit precision using the same approach. The reason for quantizing model weights to 32 bits, even though the default bit-width is already 32 bits, is to evaluate the performance of the quantization process and assess how closely the results match the original values. With the inclusion of multiple bit precisions (2, 4, 6, 8, 16, and 32), only average metrics are included instead of detailed metrics for each dataset in this approach and the two other mixed-precision approaches (quantizing only convolutional layers and quantizing only non-convolutional layers) to keep the performance metrics table concise as the number of rows increased.

It is worth mentioning that in TensorFlow, model variables represent named tensors stored in memory. These variables can refer to weight matrices (e.g., layer kernels, biases), batch normalization statistics (e.g., moving_mean, moving_variance, gamma, beta), or any other learnable or non-learnable values. On the other hand, a parameter refers to the individual numerical elements (e.g., floating-point numbers) that make up the tensors associated with variables.

In the MultiTempGAN generator structure, the total number of variables is 36, excluding those associated with batch normalization ("moving_mean" and

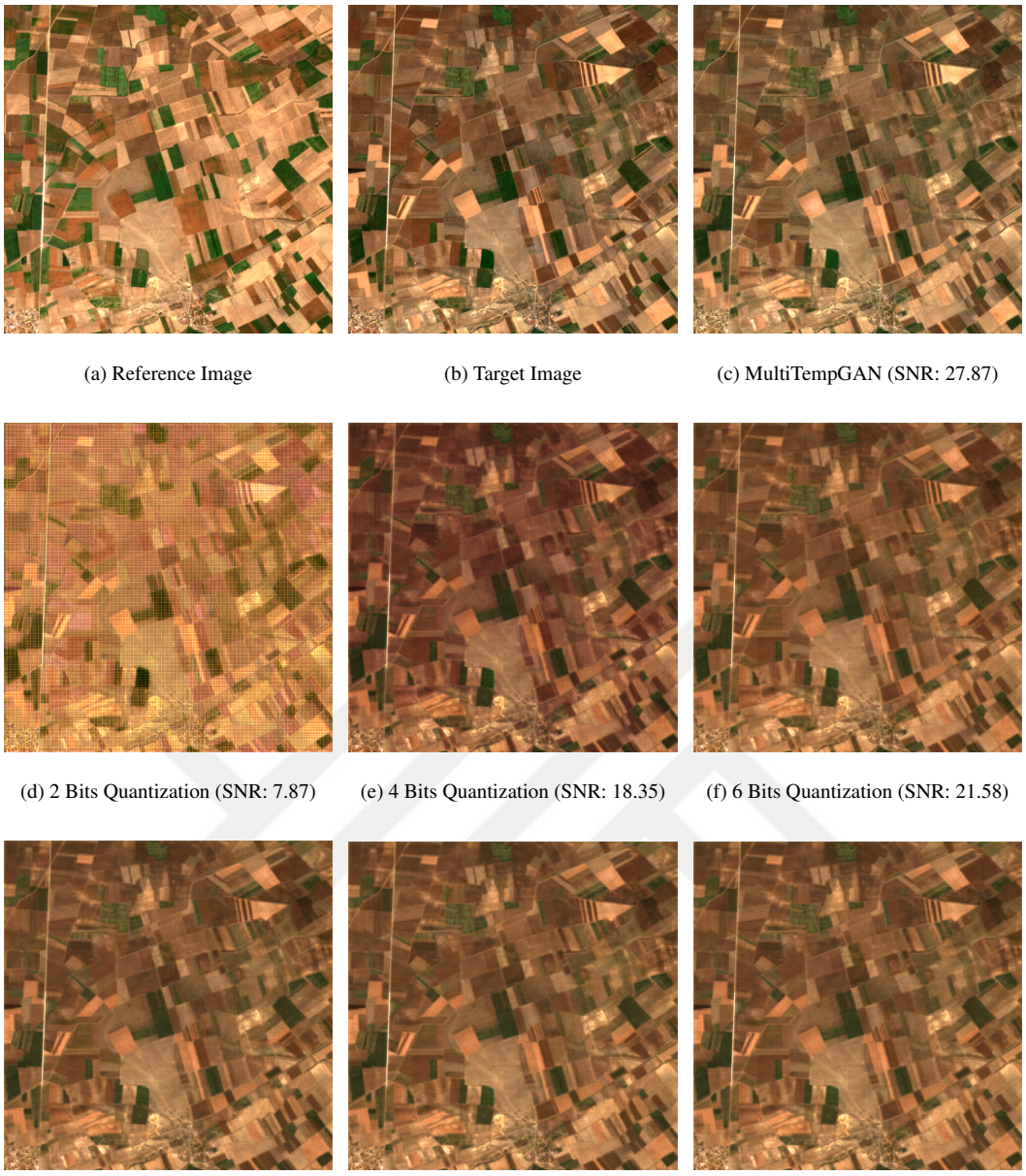
"moving_variance"). These variables are also omitted when counting the generator parameters, and since their impact on bpp and memory is minimal, this exclusion is justified. The total number of parameters related to the mentioned 36 variables in the generator is 2,560,252. During the quantization process, each variable requires a scaling factor, meaning the number of side information parameters equals the number of unique variables in the model. As a result, this approach requires 36 side information parameters.

Table 4.1 Performance metrics of quantizing all weights approach for 2, 4, 6, 8, 16 and 32 bits averaged across all three datasets.

Bit Precision	Memory (MB)	bpp	SNR	LMSE	Ratio
2	0.61	0.0037	6.51	103.01	15.99
4	1.22	0.0074	17.70	14.44	7.99
6	1.83	0.0111	20.06	2.15	5.33
8	2.44	0.0148	20.23	1.37	3.99
16	4.88	0.0295	20.21	1.31	1.99
32	9.76	0.0590	20.21	1.31	0.99

As reported in Table 4.1 and demonstrated in Figure 4.1, the 6-bit and 8-bit precision levels produce the best results when considering the average SNR, LMSE metrics and the visual results. Therefore, 2-bit and 4-bit precisions are excluded due to their significantly high LMSE values and low SNR results. Similarly, 16-bit precision is eliminated as its performance is nearly identical to that of 8-bit precision, making it inefficient to use twice the number of bits. Finally, as previously mentioned, 32-bit precision is included only for testing and evaluation purposes and is not considered in practical scenarios.

For the remaining categories and approaches, only 6-bit and 8-bit precision results are reported, as these provide a good balance between performance and efficiency. However, in cases where 6-bit precision results are exceptionally close to the original (32-bit) results, 4-bit precision is also included to explore the possibility of representing the model with even fewer bits without significant loss in accuracy.



(a) Reference Image (b) Target Image (c) MultiTempGAN (SNR: 27.87)

(d) 2 Bits Quantization (SNR: 7.87) (e) 4 Bits Quantization (SNR: 18.35) (f) 6 Bits Quantization (SNR: 21.58)

(g) 8 Bits Quantization (SNR: 21.63) (h) 16 Bits Quantization (SNR: 21.67) (i) 32 Bits Quantization (SNR: 21.67)

Figure 4.1 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with quantizing all weights approach.

4.2.2 Linear Quantization

Linear quantization is a simple yet widely used method to quantize the weights and parameters of the model. In this approach, the range of values is divided uniformly into a fixed number of levels, determined by the chosen bit precision (e.g., 6-bit or 8-bit quantization). This technique is computationally efficient and straightforward to implement[24] [27]. Linear quantization is particularly suitable for scenarios that require fast and hardware-friendly implementations.

- **Bit-width Selection**: The desired bit precision (e.g., 8 bits) is selected to

determine the number of quantization levels.

- **Scaling Factor**: A scale factor is calculated based on the maximum absolute value (α) of the variable to ensure that all values fit within the quantization range.
- **Quantization**: The weights are quantized by multiplying them by the scale factor, rounding them to the nearest integers, and clipping the results to fit within the valid range of the chosen bit precision.
- **Reconstruction**: The quantized weights are dequantized back to floating-point values using the same scaling factor for further processing or storage.

4.2.2.1 Linear Quantization Algorithm

Algorithm 2, shows how Linear Quantization technique works in details.

Algorithm 2 Linear Quantization

1: **Input:** Model checkpoint (`checkpoint_dir`), bit-width (`bitval`)
2: **Output:** Quantized model checkpoint with updated weights
3: Load the model checkpoint from `checkpoint_dir`
4: Retrieve the list of variables in the checkpoint
5: Initialize an empty list `assign_ops` to store updated variables
6: **for** each variable `var` in the checkpoint **do**
7: Load the tensor values of `var` from the checkpoint
8: Compute the maximum absolute value: $\alpha = \max(|\texttt{var}|)$
9: Calculate the scaling factor: $\texttt{scale} = \frac{(2^{\texttt{bitval}}-1)}{\alpha}$
10: Quantize weights: $\texttt{quantized_weights} = \texttt{round}(\texttt{var} \times \texttt{scale})$
11: Clip quantized weights to the range: $[-2^{\texttt{bitval}-1}, 2^{\texttt{bitval}-1} - 1]$
12: Dequantize weights: $\texttt{rescaled_weights} = \frac{\texttt{quantized_weights}}{\texttt{scale}}$
13: Convert `rescaled_weights` to `float32` (required due to TensorFlow's limitation of saving variables as `float32`)
14: Create a TensorFlow variable with the updated weights and append it to `assign_ops`
15: **end for**
16: Initialize a TensorFlow session and run `assign_ops` to update all variables
17: Save the quantized model checkpoint at a specified path

Similar to the quantizing all weights approach, in linear quantization the number of quantized variables (tensors) is 36 corresponding to 2,560,252 parameters and

since one scale factor is required per variable, 36 side information parameters will be added. Table 4.2 contains the performance metrics of the linear quantization method and Figure 4.2 illustrates the images generated for the randomly selected sample from one of the datasets using this approach. As mentioned in Section 4.2.1 for linear quantization and the rest of the approaches, only the 6-bit and 8-bit precisions are included.

Table 4.2 Performance metrics of linear quantization approach for 6 and 8 bits across all three datasets.

Bit	Dataset	Memory (MB)	bpp	SNR	LMSE	Ratio
6	MSI Pair-1	1.83	0.0111	21.97	2.39	5.33
	MSI Pair-2	1.83	0.0111	20.64	2.14	5.33
	MSI Pair-3	1.83	0.0111	18.92	3.22	5.33
	Avg.	**1.83**	**0.0111**	**20.51**	**2.58**	**5.33**
8	MSI Pair-1	2.44	0.0148	22.25	0.85	3.99
	MSI Pair-2	2.44	0.0148	21.48	0.44	3.99
	MSI Pair-3	2.44	0.0148	19.14	0.78	3.99
	Avg.	**2.44**	**0.0148**	**20.96**	**0.69**	**3.99**



(a) Reference Image (b) Target Image (c) MultiTempGAN (SNR: 27.87)



(d) 6 Bits Quantization (SNR: 21.97) (e) 8 Bits Quantization (SNR: 22.25)

Figure 4.2 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with linear quantization approach.

4.2.3 Fixed-Point Quantization

Fixed-point quantization transforms floating-point numbers into a fixed-point representation by using predefined integer and fractional bits. This approach ensures compatibility with hardware acceleration while maintaining computational efficiency[24] [28]. For example, weights are scaled by $2^{\text{fractional_bits}}$, quantized, and clipped to fit within a fixed-point range. This method is particularly suitable for resource-constrained environments where operations on fixed-point numbers are more efficient than on floating-point numbers.

- **Scaling Factor**: The weights are scaled using a factor of $2^{\text{fractional_bits}}$, converting them to fixed-point representation.
- **Quantization**: The scaled weights are rounded to the nearest integers and clipped to ensure they remain within the representable range of the chosen fixed-point format.
- **Reconstruction**: The quantized weights are dequantized back to floating-point values by dividing them by the same scaling factor.

4.2.3.1 Fixed-Point Quantization Algorithm

Fixed-Point Quantization approach is demonstrated in Algorithm 3 as follows:

Algorithm 3 Fixed-Point Quantization

1: **Input:** Model checkpoint (`checkpoint_dir`), integer bit-width (`integer_bits`), fractional bit-width (`fractional_bits`)
2: **Output:** Quantized model checkpoint with updated weights
3: Load the model checkpoint from `checkpoint_dir`
4: Retrieve the list of variables in the checkpoint
5: Initialize an empty list `assign_ops` to store updated variables
6: **for** each variable `var` in the checkpoint **do**
7: Load the tensor values of `var` from the checkpoint
8: Compute the scaling factor: $\texttt{scale} = 2^{\texttt{fractional_bits}}$
9: Quantize weights: $\texttt{quantized_weights} = \texttt{round}(\texttt{var} \times \texttt{scale})$
10: Clip quantized weights to the range: $[-2^{\texttt{integer_bits + fractional_bits}-1}, 2^{\texttt{integer_bits + fractional_bits}-1} - 1]$
11: Dequantize weights: $\texttt{rescaled_weights} = \frac{\texttt{quantized_weights}}{\texttt{scale}}$
12: Convert `rescaled_weights` to `float32` (required due to TensorFlow's limitation of saving variables as `float32`)
13: Create a TensorFlow variable with the updated weights and append it to `assign_ops`
14: **end for**
15: Initialize a TensorFlow session and run `assign_ops` to update all variables
16: Save the quantized model checkpoint at a specified path

This approach requires 36 side information parameters to reconstruct the quantized weights. In fixed-point quantization, numerical values are represented using a fixed number of bits for the integer part and another set of bits for the fractional part. However, the total bit-width used for quantization is the sum of these two parts. For example, if the integer part is allocated 2 bits and the fractional part 6 bits, the overall quantized representation would be considered as 8-bit fixed-point quantization [28][29].

Table 4.3 presents the performance metrics of the fixed-point quantization approach, while Figure 4.3 visually compares its results with the reference, target, and MultiTempGAN-generated image (patch).

Table 4.3 Performance metrics of the fixed-point quantization approach for 8 (2+6), 12 (3+9), and 16 (3+13) bits across all three datasets.

Total Bit Precision	Dataset	Memory (MB)	bpp	SNR	LMSE	Ratio
8 (2+6)	MSI Pair-1	2.44	0.0148	25.50	3.73	3.99
	MSI Pair-2	2.44	0.0148	23.64	2.18	3.99
	MSI Pair-3	2.44	0.0148	21.70	5.27	3.99
	Avg.	**2.44**	**0.0148**	**23.61**	**3.73**	**3.99**
12 (3+9)	MSI Pair-1	3.66	0.0221	27.82	0.54	2.66
	MSI Pair-2	3.66	0.0221	26.33	0.35	2.66
	MSI Pair-3	3.66	0.0221	24.65	0.44	2.66
	Avg.	**3.66**	**0.0221**	**26.27**	**0.45**	**2.66**
16 (3+13)	MSI Pair-1	4.88	0.0295	27.87	0.46	2.00
	MSI Pair-2	4.88	0.0295	26.43	0.32	2.00
	MSI Pair-3	4.88	0.0295	24.72	0.39	2.00
	Avg.	**4.88**	**0.0295**	**26.34**	**0.39**	**2.00**



(a) Reference Image (b) Target Image (c) MultiTempGAN (SNR: 27.87)



(d) 8 (2+6) Bits Quantization (SNR: 25.50) (e) 12 (3+9) Bits Quantization (SNR: 27.82) (e) 16 (3+13) Bits Quantization (SNR: 27.87)

Figure 4.3 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with fixed-point quantization approach.

During evaluating this method, several different combinations of bit precisions were tested, and it was observed that using lower bit widths for integer values while allocating more bits for fractional values generate the best results. This makes sense as fractional values require more precision to maintain accuracy. Among all combinations tested, the three reported results were the most promising. The performance metrics of this approach demonstrate relatively good and accurate results, especially for 12 (3 +9) bits. However, since the bit precision required for fixed-point quantization was higher than those of other superior approaches, it resulted in increased memory usage and bpp metrics. At the same time, the SNR values of other methods were not significantly lower, so it was decided not to use this approach in the proposed model.

4.2.4 Stochastic Quantization

Stochastic quantization introduces randomness into the quantization process by probabilistically rounding weights based on their proximity to quantization levels. This method helps reduce quantization artifacts and improves generalization by adding controlled noise to the training process [24]. Unlike deterministic

quantization methods, stochastic quantization varies rounding decisions, which can result in improved robustness in model performance.

- **Scaling Factor**: The weights are scaled using a factor derived from the maximum absolute value (α) of the weights and the maximum quantization level ($Q = 2^{\text{bitval}-1} - 1$). This ensures that the values fall within the range $[-Q, Q]$.
- **Rounding Decision**: The scaled weights are stochastically rounded to either the lower or upper quantization levels based on their proximity to these levels. The probability of rounding up is proportional to the fractional distance from the lower level.
- **Clipping**: After rounding, the quantized weights are clipped to fit within the representable range $[-Q, Q]$, avoiding overflow.
- **Reconstruction**: The quantized weights are dequantized back to floating-point values using the scaling factor, ensuring compatibility with TensorFlow's variable format.

4.2.4.1 Stochastic Quantization Algorithm

Details about Stochastic Quantization technique can be found in the algorithm 4 provided below:

Algorithm 4 Stochastic Quantization

Input: Model checkpoint (`checkpoint_dir`), bit-width (`bitval`)
Output: Stochastically quantized model checkpoint with updated weights
Load the model checkpoint from `checkpoint_dir`
Retrieve the list of variables in the checkpoint
Initialize an empty list `assign_ops` to store updated variables
Compute the maximum quantization level: $Q = 2^{\texttt{bitval}-1} - 1$
for each variable `var` in the checkpoint **do**
    Load the tensor values of `var` from the checkpoint
    Compute the maximum absolute value: $\alpha = \max(|\texttt{var}|)$
    Calculate the scaling factor: $\texttt{scale} = Q/\alpha$
    Scale the weights: $\texttt{scaled_var} = \texttt{var} \times \texttt{scale}$
    Compute lower and upper quantization levels:
    $\texttt{lower_bound} = \lfloor \texttt{scaled_var} \rfloor$,
    $\texttt{upper_bound} = \texttt{lower_bound} + 1$
    Calculate the rounding probability:
    $\texttt{prob} = \texttt{scaled_var} - \texttt{lower_bound}$
    Generate a random tensor:
    $\texttt{stochastic_rounding} = \texttt{rand}(*\texttt{var.shape}) < \texttt{prob}$
    Perform stochastic rounding:
    $\texttt{quantized_weights} = \texttt{lower_bound} + \texttt{stochastic_rounding}$
    Clip quantized weights to the range: $[-Q, Q]$
    Dequantize weights: $\texttt{rescaled_weights} = \frac{\texttt{quantized_weights}}{\texttt{scale}}$
    Convert `rescaled_weights` to `float32` (required due to TensorFlow's limitation of saving variables as `float32`)
    Create a TensorFlow variable with the updated weights and append it to `assign_ops`
end for
Initialize a TensorFlow session and run `assign_ops` to update all variables
Save the quantized model checkpoint at a specified path

This method also requires 36 side information parameters for quantizing all generator parameters. Table 4.4 contains the performance metrics of the stochastic quantization method and Figure 4.4 depicts the images generated using the quantization approach for the same randomly selected sample from one of the datasets that has also been used to test the previous models.

Table 4.4 Performance metrics of stochastic quantization approach for 6 and 8 bits across all three datasets.

Bit	Dataset	Memory (MB)	bpp	SNR	LMSE	Ratio
6	MSI Pair-1	1.83	0.0111	22.69	11.32	5.33
	MSI Pair-2	1.83	0.0111	19.24	8.94	5.33
	MSI Pair-3	1.83	0.0111	16.31	29.28	5.33
	Avg.	**1.83**	**0.0111**	**19.41**	**16.51**	**5.33**
8	MSI Pair-1	2.44	0.0148	27.28	0.89	3.99
	MSI Pair-2	2.44	0.0148	25.02	1.20	3.99
	MSI Pair-3	2.44	0.0148	23.96	1.15	3.99
	Avg.	**2.44**	**0.0148**	**25.42**	**1.08**	**3.99**



(a) Reference Image



(b) Target Image



(c) MultiTempGAN (SNR: 27.87)



(d) 6 Bits Quantization (SNR: 22.69)



(e) 8 Bits Quantization (SNR: 27.28)

Figure 4.4 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with stochastic quantization approach.

Stochastic quantization performs pretty well, especially for 8-bit precision. However, as there are other methods that outperformed this both in 6-bit and 8-bit precisions, this method was eliminated when the proposed model was being

designed. Furthermore, the difference between SNR and LMSE metrics of this technique with 8-bit was significantly larger.

4.2.5 Symmetric Quantization

Symmetric quantization ensures that quantization levels are symmetric around zero, with positive and negative values equally spaced. This approach simplifies the quantization and dequantization process and is computationally efficient. It is particularly useful for balanced data distributions and is widely supported by popular quantization toolkits, such as TensorFlow and PyTorch [24].

4.2.5.1 Symmetric Quantization Algorithm

Detailed explanation about the Symmetric Quantization approach can be found in algorithm 5.

Algorithm 5 Symmetric Quantization

1: **Input:** Model checkpoint (`checkpoint_dir`), bit-width (`bitval`)
2: **Output:** Quantized model checkpoint with updated weights
3: Load the model checkpoint from `checkpoint_dir`
4: Retrieve the list of variables in the checkpoint
5: Initialize an empty list `assign_ops` to store updated variables
6: Compute the maximum quantization level: $Q = 2^{\texttt{bitval}-1} - 1$
7: **for** each variable `var` in the checkpoint **do**
8: Load the tensor values of `var` from the checkpoint
9: Compute the maximum absolute value: $\alpha = \max(|\texttt{var}|)$
10: Calculate the symmetric scaling factor: $\texttt{scale} = Q/\alpha$
11: Quantize weights symmetrically: $\texttt{quantized_weights} = \texttt{round}(\texttt{var} \times \texttt{scale})$
12: Clip quantized weights to the symmetric range: $[-Q, Q]$
13: Dequantize weights: $\texttt{rescaled_weights} = \frac{\texttt{quantized_weights}}{\texttt{scale}}$
14: Convert `rescaled_weights` to `float32` (required due to TensorFlow's limitation of saving variables as `float32`)
15: Create a TensorFlow variable with the updated weights and append it to `assign_ops`
16: **end for**
17: Initialize a TensorFlow session and run `assign_ops` to update all variables
18: Save the quantized model checkpoint at a specified path

Similar to all other techniques mentioned up to here, 36 side information parameters

are essential for symmetric quantization in reconstruction stage, as each variable requires one scale (unlike asymmetric quantization, the zero point is always equal to zero in this approach). Table 4.5 reports the performance metrics of the symmetric quantization approach and Figure 4.5 illustrates the generated images using this quantization method for the same randomly selected sample from one of the datasets that has been used to test previous models as well.

Table 4.5 Performance metrics of symmetric quantization approach for 6 and 8 bits across all three datasets.

Bit	Dataset	Memory (MB)	bpp	SNR	LMSE	Ratio
6	MSI Pair-1	1.83	0.0111	23.92	4.79	5.33
	MSI Pair-2	1.83	0.0111	21.28	7.06	5.33
	MSI Pair-3	1.83	0.0111	20.38	6.47	5.33
	Avg.	**1.83**	**0.0111**	**21.86**	**6.11**	**5.33**
8	MSI Pair-1	2.44	0.0148	27.45	0.74	3.99
	MSI Pair-2	2.44	0.0148	25.67	0.65	3.99
	MSI Pair-3	2.44	0.0148	24.22	0.77	3.99
	Avg.	**2.44**	**0.0148**	**25.78**	**0.72**	**3.99**



(a) Reference Image (b) Target Image (c) MultiTempGAN (SNR: 27.87)



(d) 6 Bits Quantization (SNR: 23.92) (e) 8 Bits Quantization (SNR: 27.45)

Figure 4.5 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with symmetric quantization approach.

Symmetric quantization outperforms all other uniform methods (and also mixed-precision approaches covered in Section 4.3). Therefore, if the proposed model were to be developed using a uniform approach, this method could have been chosen with 8-bit precision, as shown in Figure 4.9.

4.2.6 Asymmetric Quantization

Asymmetric quantization adjusts quantization levels to handle data that are not centered around zero. Unlike symmetric quantization, it works well with skewed data, where most values are either positive or negative. This approach is especially useful for improving accuracy in tasks where the data has an uneven distribution, as it provides a more accurate representation of the data range [24].

4.2.6.1 Asymmetric Quantization Algorithm

Asymmetric Quantization technique is described with formulas in details in Algorithm 6.

Algorithm 6 Asymmetric Quantization

Input: Model checkpoint (`checkpoint_dir`), bit-width (`bitval`)
Output: Quantized model checkpoint with updated weights
Load the model checkpoint from `checkpoint_dir`
Retrieve the list of variables in the checkpoint
Initialize an empty list `assign_ops` to store updated variables
Compute the maximum quantization level: $Q = 2^{\texttt{bitval}} - 1$
for each variable `var` in the checkpoint **do**
    Load the tensor values of `var` from the checkpoint
    Compute the minimum and maximum values of the tensor:
    $\texttt{min_val} = \min(\texttt{var}), \texttt{max_val} = \max(\texttt{var})$
    Calculate the scale and zero-point for quantization:

$$\texttt{scale} = \frac{\texttt{max_val} - \texttt{min_val}}{Q}, \quad \texttt{zero_point} = \texttt{round}\left(-\frac{\texttt{min_val}}{\texttt{scale}}\right)$$

    Clip the zero-point to the valid range:

$$\texttt{zero_point} = \texttt{clip}\big(\texttt{zero_point}, 0, Q\big)$$

    Quantize weights asymmetrically:

$$\texttt{quantized_weights} = \texttt{clip}\left(\texttt{round}\left(\frac{\texttt{var}}{\texttt{scale}} + \texttt{zero_point}\right), 0, Q\right)$$

    Dequantize weights:

$$\texttt{rescaled_weights} = \big(\texttt{quantized_weights} - \texttt{zero_point}\big) \times \texttt{scale}$$

    Convert `rescaled_weights` to `float32` (required due to TensorFlow's limitation of saving variables as `float32`)
    Create a TensorFlow variable with the updated weights and append it to `assign_ops`
end for
Initialize a TensorFlow session and run `assign_ops` to update all variables
Save the quantized model checkpoint at a specified path

Considering that asymmetric quantization needs one scale factor and one zero-point per variable, this quantization method requires 72 side information parameters (twice the other uniform techniques) because the total number of generator variables that get quantized is 36. Table 4.6 represents the key performance metrics for this approach, and Figure 4.6 shows the visual comparison between the images generated using this method and MultiTempGAN along with the reference and target image.

Table 4.6 Performance metrics of asymmetric quantization approach for 6 and 8 bits across all three datasets.

Bit	Dataset	Memory (MB)	bpp	SNR	LMSE	Ratio
6	MSI Pair-1	1.83	0.0111	19.91	4.18	5.33
	MSI Pair-2	1.83	0.0111	17.42	4.10	5.33
	MSI Pair-3	1.83	0.0111	16.51	7.42	5.33
	Avg.	**1.83**	**0.0111**	**17.95**	**5.24**	**5.33**
8	MSI Pair-1	2.44	0.0148	21.87	0.88	3.99
	MSI Pair-2	2.44	0.0148	19.08	0.53	3.99
	MSI Pair-3	2.44	0.0148	17.08	0.73	3.99
	Avg.	**2.44**	**0.0148**	**19.09**	**0.72**	**3.99**



(a) Reference Image



(b) Target Image



(c) MultiTempGAN (SNR: 27.87)



(d) 6 Bits Quantization (SNR: 19.91)



(e) 8 Bits Quantization (SNR: 21.08)

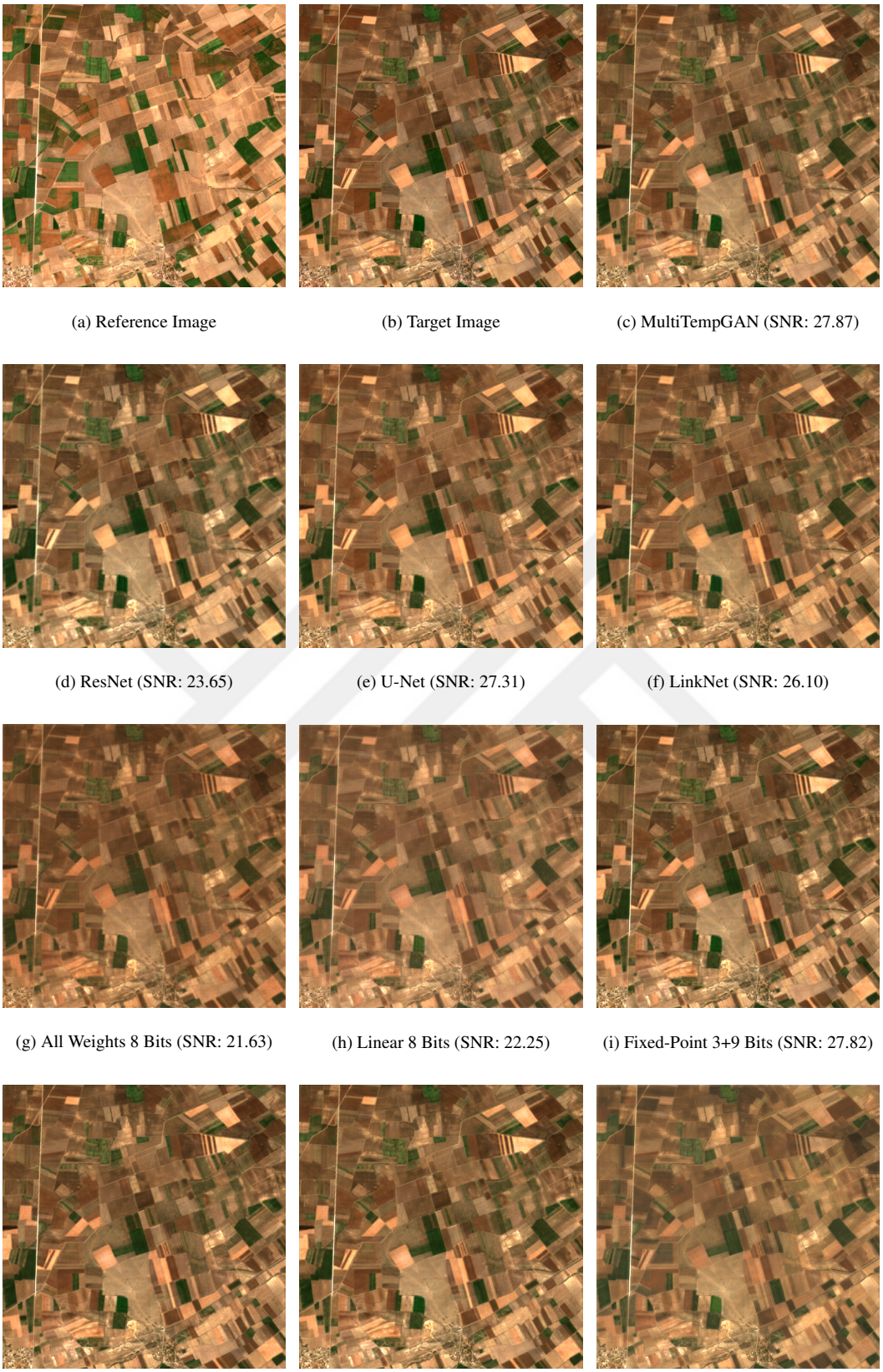
Figure 4.6 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with asymmetric quantization approach.

4.2.7 Performance Comparison of All Uniform Approaches

In this section, in order to make the comparisons between all deep learning-based models mentioned in the MultiTempGAN paper (namely, ResNet, U-Net, LinkNet, and MultiTempGAN itself), and the quantized versions of MultiTempGAN that use the explained uniform approaches easier (both visually and numerically), from each approach the bit precision that produces the best results in terms of model size, bpp, and SNR is summarized once again in Table 4.7. Furthermore, while Figure 4.7 illustrates the results generated using the mentioned uniform approaches and other deep learning-based models, Figure 4.8 plots the SNR against bpp with the Euclidean distance highlighted between the result of each uniform approach, MultiTempGAN, and other deep learning-based models. The Euclidean distance was computed as demonstrated in Equation 4.6.

Table 4.7 Performance metrics of the best uniform approaches and all non-quantized deep learning-based models. SNR and LMSE values are averaged across all three datasets. The table includes bit precision, the number of side information parameters, memory usage (MB), and the compression ratio compared to MultiTempGAN, noted as Bits, S.I., Mem., and Ratio, respectively.

	Method	**Bits**	**S.I.**	**Mem.**	**bpp**	**SNR**	**LMSE**	**Ratio**
Deep Learning Approaches	ResNet	-	-	11.00	0.0665	22.23	0.65	0.88
	U-Net	-	-	9.76	0.0590	25.49	0.45	1.00
	LinkNet	-	-	9.84	0.0595	24.82	0.49	0.99
	MultiTempGAN	-	-	9.76	0.0590	26.34	0.39	1.00
Uniformly Quantized Versions	All Weights	8	36	2.44	0.0148	20.23	1.37	3.99
	Linear	8	36	2.44	0.0148	20.96	0.69	3.99
	Fixed-Point	12	36	3.66	0.0221	26.27	0.45	2.66
	Stochastic	8	36	2.44	0.0148	25.42	1.08	3.99
	Symmetric	8	36	2.44	0.0148	25.78	0.72	3.99



(a) Reference Image (b) Target Image (c) MultiTempGAN (SNR: 27.87)

(d) ResNet (SNR: 23.65) (e) U-Net (SNR: 27.31) (f) LinkNet (SNR: 26.10)

(g) All Weights 8 Bits (SNR: 21.63) (h) Linear 8 Bits (SNR: 22.25) (i) Fixed-Point 3+9 Bits (SNR: 27.82)

(j) Stochastic 8 Bits (SNR: 27.2) (k) Symmetric 8 Bits (SNR: 27.45) (l) Asymmetric 8 Bits (SNR: 27.08)

Figure 4.7 Visual Comparison of all deep learning-based models and uniformly quantized versions of MultiTempGAN using the best bit precision for each method.

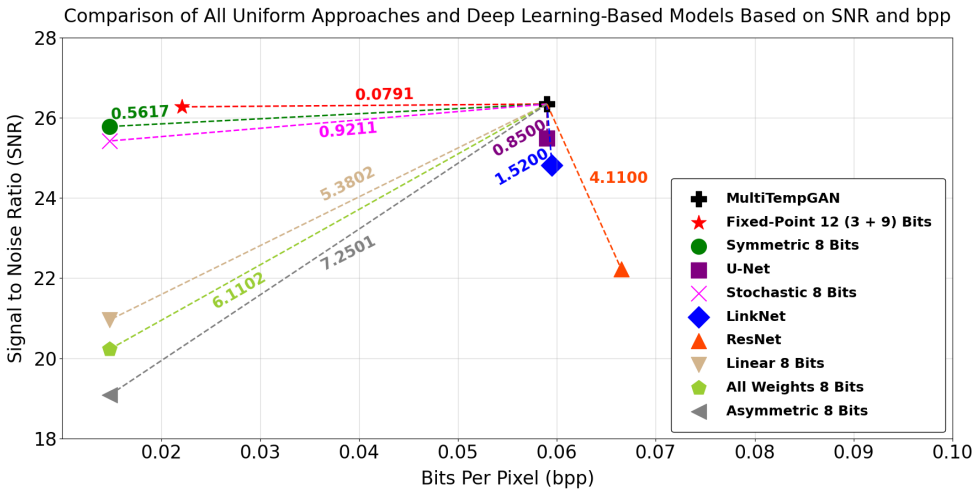


Figure 4.8 Comparison of all uniform approaches, ResNet, U-Net, LinkNet, and MultiTempGAN based on Euclidean distance calculated with SNR and bpp metrics.

Above, in Figure 4.8 the Euclidean distance is calculated as follows:

$$d = \sqrt{(bpp - bpp_{ref})^2 + (SNR - SNR_{ref})^2} \tag{4.6}$$

where:

- d : The Euclidean distance between a given quantization method and the original (reference) model, which is MultiTempGAN.
- bpp : The bpp value of the evaluated quantization method.
- bpp_{ref} : The bpp value of the reference model.
- SNR : The SNR value of the evaluated quantization method.
- SNR_{ref} : The SNR of the reference model.

The Euclidean distance is computed using the raw values of bpp and SNR, instead of their normalized counterparts. Since lower bpp values indicate higher compression efficiency and higher SNR values indicate better reconstruction quality, the quantization method with the smallest d value is considered the most similar in performance to MultiTempGAN.

Figure 4.8 shows that after the fixed-point method with 12-bit precision (3 bits for integer values and 9 bits for fractional values), the smallest distance belongs

to symmetric quantization with 8-bit precision. However, since the impact of extremely small differences in bpp values decreases when calculating the Euclidean distance, and given that the main goal of this study is to compress the model without significantly sacrificing accuracy, the approach with the second smallest distance after fixed-point quantization was chosen to design the architecture illustrated in Figure 4.9. This selected approach is symmetric quantization with 8-bit precision. This architecture is created under the assumption that the choice of quantization approach is limited to uniform methods only and is called Q-MultiTempGAN.

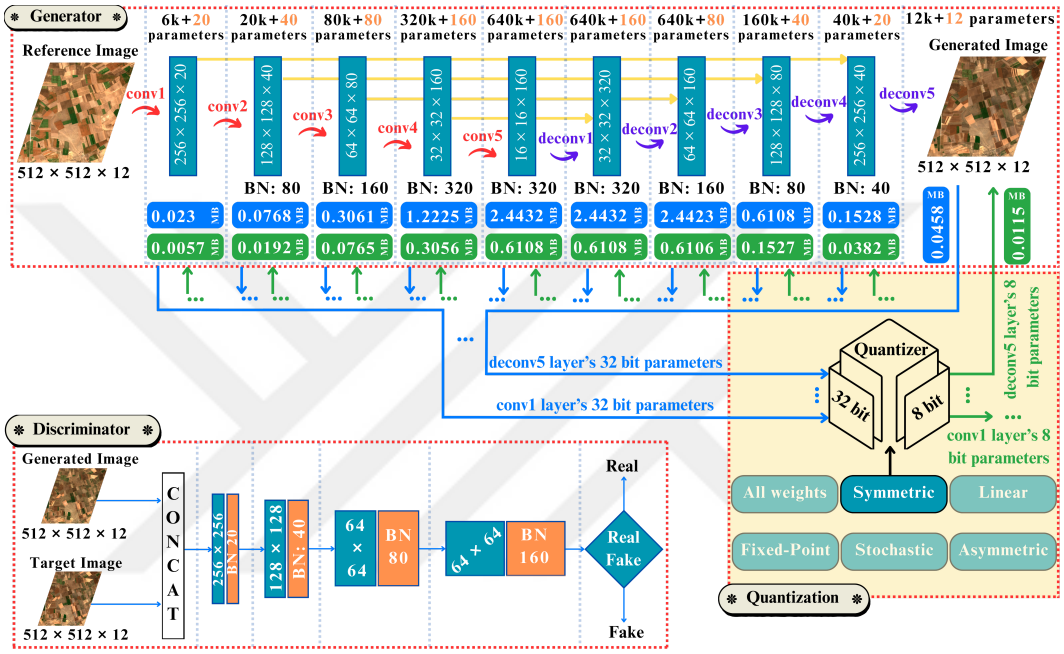


Figure 4.9 Q-MultiTempGAN Architecture with the generator structure is shown at the top, discriminator at the bottom left, and quantization process in the bottom right.

4.3 Mixed-precision Quantization

In mixed-precision quantization, various components of the model, such as layers or parameters, are assigned different bit widths depending on their sensitivity to quantization errors [30]. In this category, four main approaches were tested for the sake of this study and are detailed in the following.

4.3.1 Quantizing Only Convolutional Layers

In this approach, the weights of the convolutional layers are represented with a lower bit width using the linear quantization approach, while weights of the non-convolutional layers remain in the default 32-bit precision. The algorithm identifies convolutional layers for quantization by examining variable names and selecting the variables that contain substrings such as "kernel" or "weight" or end with "w". It preserves the precision of fully connected and normalization layers, as these layers are more sensitive to quantization errors.

4.3.1.1 Quantizing Only Convolutional Layers Algorithm

Algorithm 7 depicts all the details and information about the process carried out using the Quantizing Only Convolutional Layers method.

Algorithm 7 Quantizing Only Convolutional Layers

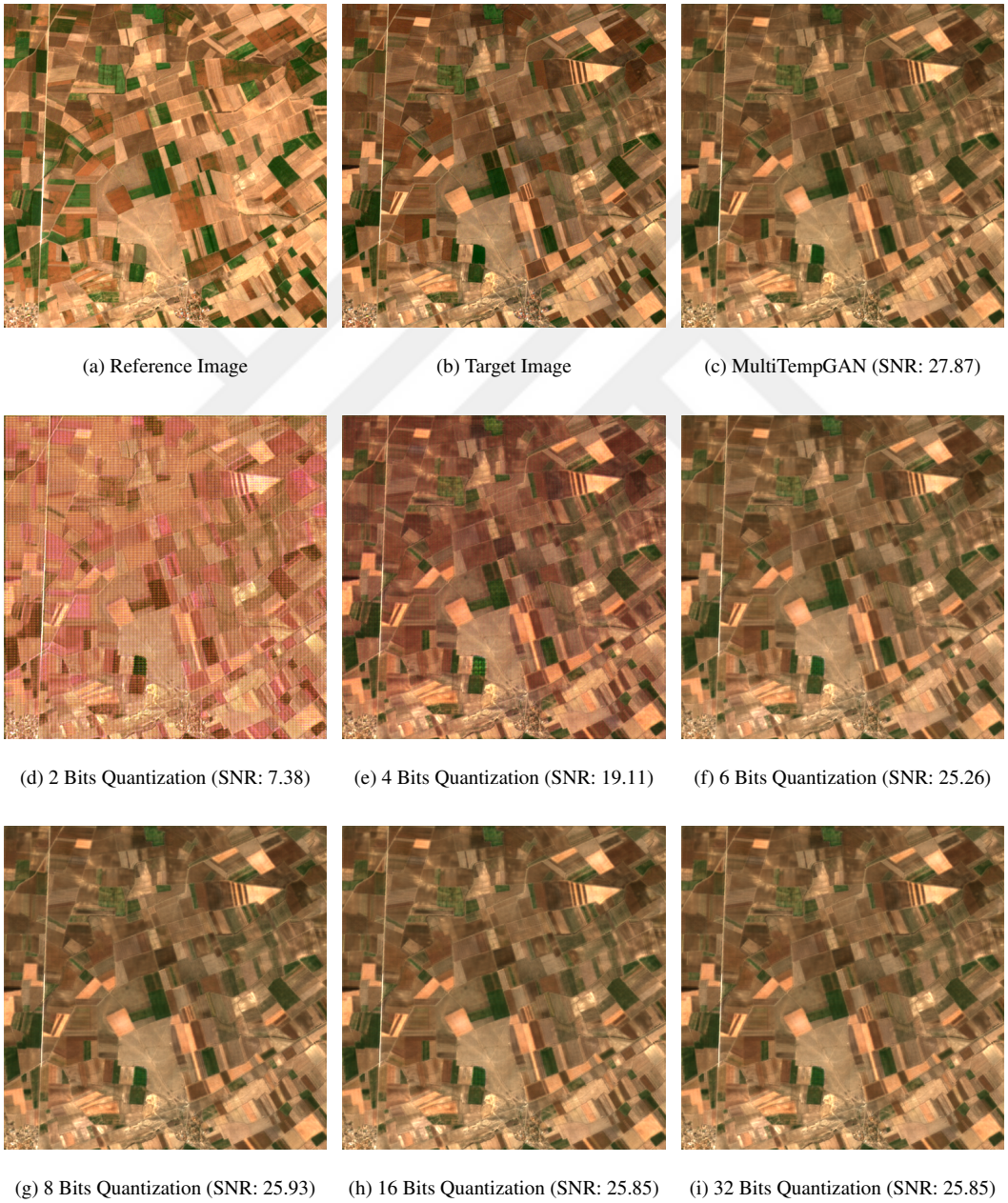
Input: Model checkpoint (`checkpoint_dir`), bit-width (`bitval`)
Output: Quantized model checkpoint with updated convolutional layer weights
Load the model checkpoint from `checkpoint_dir`
Retrieve the list of variables in the checkpoint
Initialize an empty list `assign_ops` to store weight updates
for each variable `var` in the checkpoint **do**
  if `var` belongs to a convolutional layer **then**
    Load the tensor values of `var` from the checkpoint
    Compute the maximum value $\alpha = \max(\texttt{var})$
    Calculate the scaling factor: $\texttt{scale} = \frac{(2^{\texttt{bitval}}-1)}{\alpha}$
    Quantize weights: $\texttt{quantized_weights} = \texttt{round}(\texttt{var} \times \texttt{scale})$
    Clip quantized weights to the range: $[-2^{\texttt{bitval}-1}, 2^{\texttt{bitval}-1} - 1]$
    Dequantize weights: $\texttt{rescaled_weights} = \frac{\texttt{quantized_weights}}{\texttt{scale}}$
    Convert `rescaled_weights` to `float32` (required due to TensorFlow's limitation of saving variables as `float32`)
    Create a TensorFlow variable with the updated weights and append it to `assign_ops`
  end if
end for
Initialize a TensorFlow session and run `assign_ops` to update all convolutional layer variables
Save the quantized model checkpoint at a specified path

Table 4.8 and Figure 4.10 present numerical and visual comparisons of key performance metrics across different bit widths for this approach. As with uniform quantization, the 2, 4, 6, 8, 16, and 32-bit precisions were tested in the first two mixed-precision methods. Based on the superior SNR and LMSE metrics observed for 6-bit and 8-bit quantization, the remaining bit-widths were skipped in the remaining mixed-precision approaches.

The quantization of the generator parameters in this approach applies lower bit widths to 10 variables (such as the encoder variable with the name `generator/g_e1_conv/w`, the decoder variable named `generator/g_d2/w`, etc.), covering 2,558,000 of the total 2,560,252 generator parameters. Since linear quantization requires one scale factor per quantized variable, this method introduces 10 additional side information parameters.

Table 4.8 Performance metrics of quantizing only convolutional layers approach for 2, 4, 6, 8, 16 and 32 bits averaged across all three datasets.

Bit Precision	Memory (MB)	bpp	SNR	LMSE	Ratio
2	0.61	0.0037	4.27	101.43	15.79
4	1.22	0.0074	16.72	23.83	7.95
6	1.83	0.0111	23.93	2.41	5.31
8	2.44	0.0148	25.19	0.66	3.98
16	4.88	0.0295	25.23	0.57	1.99
32	9.76	0.0590	25.23	0.57	0.99



(a) Reference Image (b) Target Image (c) MultiTempGAN (SNR: 27.87)

(d) 2 Bits Quantization (SNR: 7.38) (e) 4 Bits Quantization (SNR: 19.11) (f) 6 Bits Quantization (SNR: 25.26)

(g) 8 Bits Quantization (SNR: 25.93) (h) 16 Bits Quantization (SNR: 25.85) (i) 32 Bits Quantization (SNR: 25.85)

Figure 4.10 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with quantizing only convolutional layers weights approach.

4.3.2 Quantizing Only Non-Convolutional Layers

This algorithm ensures that only non-convolutional layers undergo quantization while convolutional layers remain in their original 32-bit precision.

4.3.2.1 Quantizing Only Non-Convolutional Layers Algorithm

The process of this quantization approach is explained in Algorithm 8 as follows:

Algorithm 8 Quantizing Only Non-Convolutional Layers

Input: Model checkpoint (`checkpoint_dir`), bit-width (`bitval`)
Output: Quantized model checkpoint with updated non-convolutional layer weights
Load the model checkpoint from `checkpoint_dir`
Retrieve the list of variables in the checkpoint
Initialize an empty list `assign_ops` to store weight updates
for each variable `var` in the checkpoint **do**
    Load the tensor values of `var` from the checkpoint
    if `var` corresponds to non-convolutional weights (e.g., bias, dense, beta, gamma) **then**
        Compute the maximum absolute value: $\alpha = \max(|\texttt{var}|)$
        Calculate the scaling factor: $\texttt{scale} = \frac{(2^{\texttt{bitval}}-1)}{\alpha}$
        Quantize weights: $\texttt{quantized_weights} = \texttt{round}(\texttt{var} \times \texttt{scale})$
        Clip quantized weights to the range: $[-2^{\texttt{bitval}-1}, 2^{\texttt{bitval}-1} - 1]$
        Dequantize weights: $\texttt{rescaled_weights} = \frac{\texttt{quantized_weights}}{\texttt{scale}}$
        Convert `rescaled_weights` to `float32` to comply with TensorFlow's storage requirements
        Create a TensorFlow variable with the updated weights and append it to `assign_ops`
    else
        Store the original `var` without quantization
    end if
end for
Initialize a TensorFlow session and run `assign_ops` to update all variables
Save the quantized model checkpoint at a specified path

In this method, 26 generator variables undergo quantization and a total of 2,252 parameters are represented with a lower bit width. Table 4.9 and Figure 4.11 contain the numeric and visual comparison of the results of this approach with different bit precisions.

Table 4.9 Performance metrics of quantizing only non-convolutional layers approach for 2, 4, 6, 8, 16 and 32 bits averaged across all three datasets.

Bit Precision	Memory (MB)	bpp	SNR	LMSE	Ratio
2	0.61	0.0037	19.79	0.44	15.79
4	1.22	0.0074	20.93	0.40	7.95
6	1.83	0.0111	21.44	0.40	5.31
8	2.44	0.0148	21.47	0.40	3.98
16	4.88	0.0295	21.50	0.40	1.99
32	9.76	0.0590	21.50	0.40	0.99

Although the LMSE values seem to remain the same in Table 4.9 after 4-bit precision, it should be noted that since these values are rounded (truncated) to two decimal places, minor changes are invisible.



(a) Reference Image (b) Target Image (c) MultiTempGAN (SNR: 27.87)



(d) 2 Bits Quantization (SNR: 21.69) (e) 4 Bits Quantization (SNR: 22.54) (f) 6 Bits Quantization (SNR: 23.27)



(g) 8 Bits Quantization (SNR: 23.16) (h) 16 Bits Quantization (SNR: 23.21) (i) 32 Bits Quantization (SNR: 23.21)

Figure 4.11 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with quantizing only non-convolutional layers weights approach.

4.3.3 Quantizing Convolutional and Non-Convolutional Layers to Different Bit Widths

In this approach, the parameters of the critical convolutional layers are quantized to a specific bit width. Unlike quantizing only convolutional layers method, this approach also quantizes other parameters instead of keeping them in 32-bit precision. The key distinction is that convolutional and non-convolutional layers are quantized to different bit widths.

4.3.3.1 Quantizing Convolutional and Non Convolutional Layers to Different Bit Widths Algorithm

Algorithm 9, depicts all the steps taken in the Quantizing Convolutional and Non-Convolutional Layers to Different Bit Widths technique in details.

Algorithm 9 Mixed-Precision Quantization of Convolutional and Non-Convolutional Layers to different bit widths

Input: Model checkpoint (`checkpoint_dir`), high precision bit-width (`higher_precision_bitval`), low precision bit-width (`lower_precision_bitval`)
Output: Quantized model checkpoint with mixed-precision weights
Load the model checkpoint from `checkpoint_dir`
Retrieve the list of variables in the checkpoint
Initialize an empty list `assign_ops` to store quantized weight updates
for each variable `var` in the checkpoint **do**
    Load the tensor values of `var` from the checkpoint
    if `var` corresponds to a sensitive layer (e.g., batch norm layers or first encoder layer) **then**
        Set `bitval` to `higher_precision_bitval`
    else
        Set `bitval` to `lower_precision_bitval`
    end if
    Compute the maximum absolute value: $\alpha = \max(|\texttt{var}|)$
    Calculate the scaling factor: $\texttt{scale} = \frac{(2^{\texttt{bitval}}-1)}{\alpha}$
    Quantize weights: $\texttt{quantized_weights} = \texttt{round}(\texttt{var} \times \texttt{scale})$
    Clip quantized weights to the range: $[-2^{\texttt{bitval}-1}, 2^{\texttt{bitval}-1} - 1]$
    Dequantize weights: $\texttt{rescaled_weights} = \frac{\texttt{quantized_weights}}{\texttt{scale}}$
    Convert `rescaled_weights` to `float32` (due to TensorFlow's limitation)
    Create a TensorFlow variable with updated weights and append it to `assign_ops`
end for
Initialize a TensorFlow session and run `assign_ops` to update all variables
Save the quantized model checkpoint at the specified path

In this method, 18 variables of the generator are quantized to a higher bit width and the remaining 18 variables to a lower bit width. The first group of variables corresponds to 7,500 and the second group to 2,552,752 parameters. So, 36 side information parameters are needed. Table 4.10 and Figure 4.12 represent the

performance of this approach numerically and visually, respectively. It is worth mentioning that several bit precisions were tested in this approach and only the mentioned one showed acceptable performance.

Table 4.10 Performance metrics of quantizing convolutional and non-convolutional layers to different bit widths approach where critical convolutional layers are quantized to 16 bits and other layers to 8 bits across all three datasets.

Bit Precision	Dataset	Memory (MB)	bpp	SNR	LMSE	Ratio
16 - 8	MSI Pair-1	2.44	0.0148	22.28	0.85	3.98
	MSI Pair-2	2.44	0.0148	21.68	0.44	3.98
	MSI Pair-3	2.44	0.0148	19.06	0.77	3.98
	Avg.	**2.44**	**0.0148**	**21.01**	**0.69**	**3.98**



(a) Reference Image (b) Target Image (c) MultiTempGAN (SNR: 27.87)



(d) 16 - 8 Bits (SNR: 22.28)

Figure 4.12 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with quantizing convolutional and non-convolutional layers weights to different bit widths approach.

4.3.4 Layer-Wise Precision Quantization (LWPQ)

This approach applies different bit widths to convolutional and non-convolutional layers, ensuring that layers more sensitive to quantization errors retain a higher bit precision. Specifically, parameters of certain layers such as batch normalization (`generator/g_bn`) and the first encoder layer (`generator/g_e1`) are assigned a higher precision, while the parameters of the remaining layers are quantized to a lower bit width. This strategy balances model compression and accuracy retention, requiring additional side information parameters for each quantized variable.

4.3.4.1 LWPQ Algorithm

The details and explanation of this technique are provided in Algorithm 10 below:

Algorithm 10 Layer-Wise Precision Quantization (LWPQ)

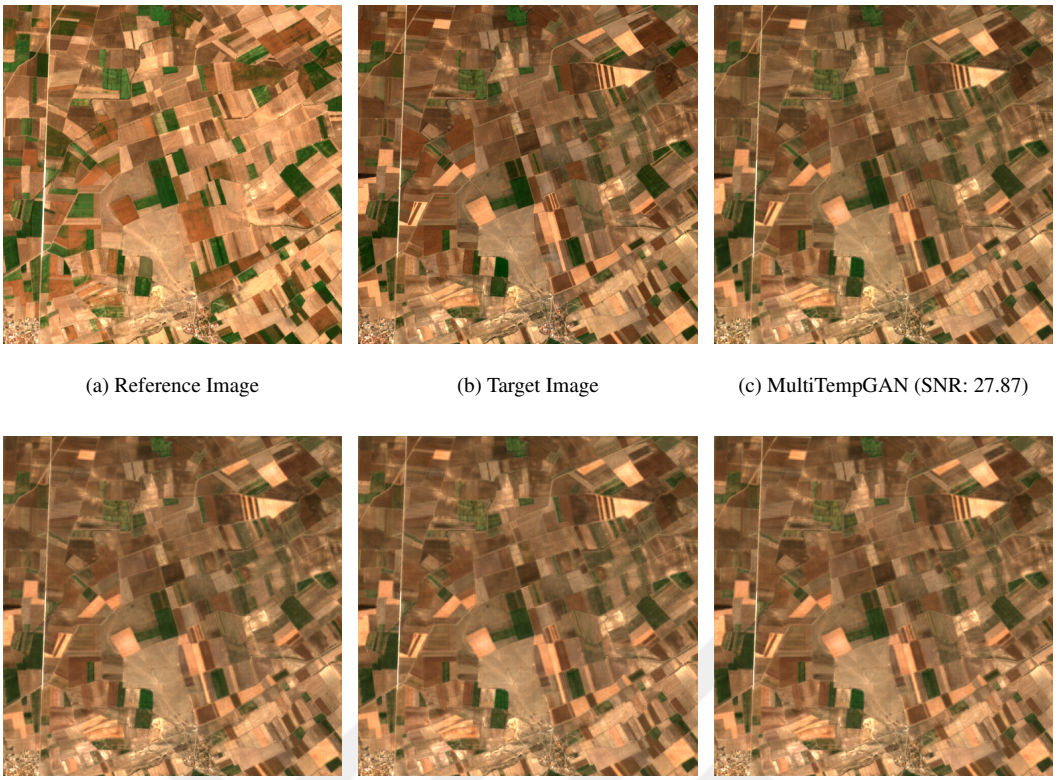
Input: Model checkpoint (`checkpoint_dir`), bit-widths for critical convolutional layers (`critical_conv_bitval`), regular convolutional layers (`conv_bitval`), and non-convolutional layers (`non_conv_bitval`)
Output: Quantized model checkpoint with layer-wise mixed-precision weights
Load the model checkpoint from `checkpoint_dir`
Retrieve the list of variables in the checkpoint
Initialize an empty list `assign_ops` to store quantized weight updates
for each variable `var` in the checkpoint **do**
    Load the tensor values of `var` from the checkpoint
    if `var` corresponds to a convolutional layer **then**
        if `var` belongs to critical layers (e.g., first and last convolutional layers) **then**
            Set `bitval` to `critical_conv_bitval`
        else
            Set `bitval` to `conv_bitval`
        end if
    else if `var` corresponds to batch normalization parameters **then**
        Skip quantization for batch norm parameters
    else
        Set `bitval` to `non_conv_bitval`
    end if
    Compute the maximum absolute value: $\alpha = \max(|\texttt{var}|) + 10^{-8}$ (avoid division by zero)
    Calculate the scaling factor: $\texttt{scale} = \frac{(2^{\texttt{bitval}}-1)}{\alpha}$
    Quantize weights: $\texttt{quantized_weights} = \texttt{round}(\texttt{var} \times \texttt{scale})$
    Clip quantized weights to the range: $[-2^{\texttt{bitval}-1}, 2^{\texttt{bitval}-1} - 1]$
    Dequantize weights: $\texttt{rescaled_weights} = \frac{\texttt{quantized_weights}}{\texttt{scale}}$
    Convert `rescaled_weights` to `float32` (due to TensorFlow's limitation)
    Create a TensorFlow variable with updated weights and append it to `assign_ops`
end for
Initialize a TensorFlow session and run `assign_ops` to update all variables
Save the quantized model checkpoint at the specified path

This approach divides the model layers into critical convolutional, regular

convolutional, and non-convolutional layers, each containing 18,000, 2,440,000, and 102,252 parameters, respectively. For these layer types, 2, 8, and 26 side information parameters are required, summing to a total of 36 side information parameters. Table 4.11 contains the performance metrics of this approach for different combinations of bit precision and Figure 4.13 depicts the visual results for the generated images using each combination.

Table 4.11 Performance metrics of the LWPQ approach for different bit precision configurations across all three datasets.

Bit Precision	Dataset	Memory (MB)	bpp	SNR	LMSE	Ratio
16 - 8 - 4	MSI Pair-1	2.45	0.0145	25.12	0.75	3.97
	MSI Pair-2	2.45	0.0145	25.68	0.39	3.97
	MSI Pair-3	2.45	0.0145	23.67	0.55	3.97
	Avg.	**2.45**	**0.0145**	**24.83**	**0.56**	**3.97**
16 - 8 - 6	MSI Pair-1	2.45	0.0147	25.54	0.78	3.97
	MSI Pair-2	2.45	0.0147	25.66	0.39	3.97
	MSI Pair-3	2.45	0.0147	23.59	0.55	3.97
	Avg.	**2.45**	**0.0147**	**24.93**	**0.57**	**3.97**
16 - 10 - 6	MSI Pair-1	3.06	0.0182	25.53	0.78	3.18
	MSI Pair-2	3.06	0.0182	25.66	0.39	3.18
	MSI Pair-3	3.06	0.0182	23.60	0.55	3.18
	Avg.	**3.06**	**0.0182**	**24.93**	**0.57**	**3.18**



(a) Reference Image (b) Target Image (c) MultiTempGAN (SNR: 27.87)

(d) 16 - 8 - 4 (SNR: 25.12) (e) 16 - 8 -6 (SNR: 25.54) (f) 16 - 10 - 6 (SNR: 25.53)

Figure 4.13 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with LWPQ.

4.3.5 Performance Comparison of All Mixed-precision Approaches

In Table 4.12 the bit precisions or the bit precisian combinations that generate the best performance metrics in terms of SNR, bpp and memory are summarized. Also, Figure 4.14 visualizes the generated images (patches) using these bit widths, and finally, Figure 4.15 depicts the comparison of all mixed-precision approaches along with other deep learning-based models such as ResNet, U-Net, LinkNet, and MultiTempGAN that is a GAN-based model.

Table 4.12 Performance metrics of the best mixed-precision approaches and all non-quantized deep learning-based models. SNR and LMSE values are averaged across all three datasets. The table includes bit precision, the number of side information parameters, memory usage (MB), and the compression ratio compared to MultiTempGAN, noted as Bits, S.I., Mem., and Ratio, respectively.

	Method	**Bits**	**S.I.**	**Mem.**	**bpp**	**SNR**	**LMSE**	**Ratio**
Deep Learning Approaches	ResNet	-	-	11.00	0.0665	22.23	0.65	0.88
	U-Net	-	-	9.76	0.0590	25.49	0.45	1.00
	LinkNet	-	-	9.84	0.0595	24.82	0.49	0.99
	MultiTempGAN	-	-	9.76	0.0590	26.34	0.39	1.00
Mixed-Precision Quantized Versions	Conv Only	8	10	2.44	0.0148	25.19	0.66	3.98
	Non-Conv Only	8	26	2.44	0.0148	21.47	0.40	3.98
	Conv - Non-Conv	16 8	36	2.44	0.0148	21.01	0.69	3.98
	LWPQ	16 8 6	36	2.45	0.0147	24.93	0.57	3.97



(a) Reference Image (b) Target Image (c) MultiTempGAN (SNR: 27.87)



(d) ResNet (SNR: 23.65) (e) U-Net (SNR: 27.31) (f) LinkNet (SNR: 26.10)



(g) Only Conv 8 Bits (SNR: 25.93) (h) Only Non-conv 8 Bits (SNR: 23.16) (i) Mixed 16 - 8 (SNR: 22.28)



(j) LWPQ 16 - 8 - 6 (SNR: 25.54)

Figure 4.14 Visual Comparison of all deep learning-based models and quantized versions of MultiTempGAN using the mixed-precision approaches and the best bit precision for each approach.

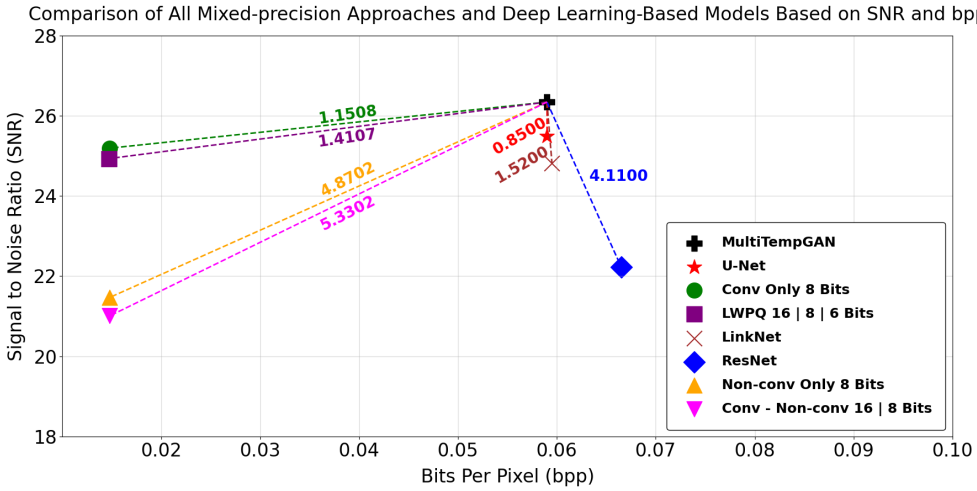


Figure 4.15 Comparison of all mixed-precision approaches, ResNet, U-Net, LinkNet, and MultiTempGAN based on Euclidean distance calculated with SNR and bpp metrics.

The plot in Figure 4.15 proves that quantizing only convolutional layers to 8-bit precision yields the best performance in this category of quantization. Therefore, if the proposed model had to be developed assuming that only mixed-precision approaches were available, it would have looked as demonstrated in Figure 4.16. This model is called MPQ-MultiTempGAN.

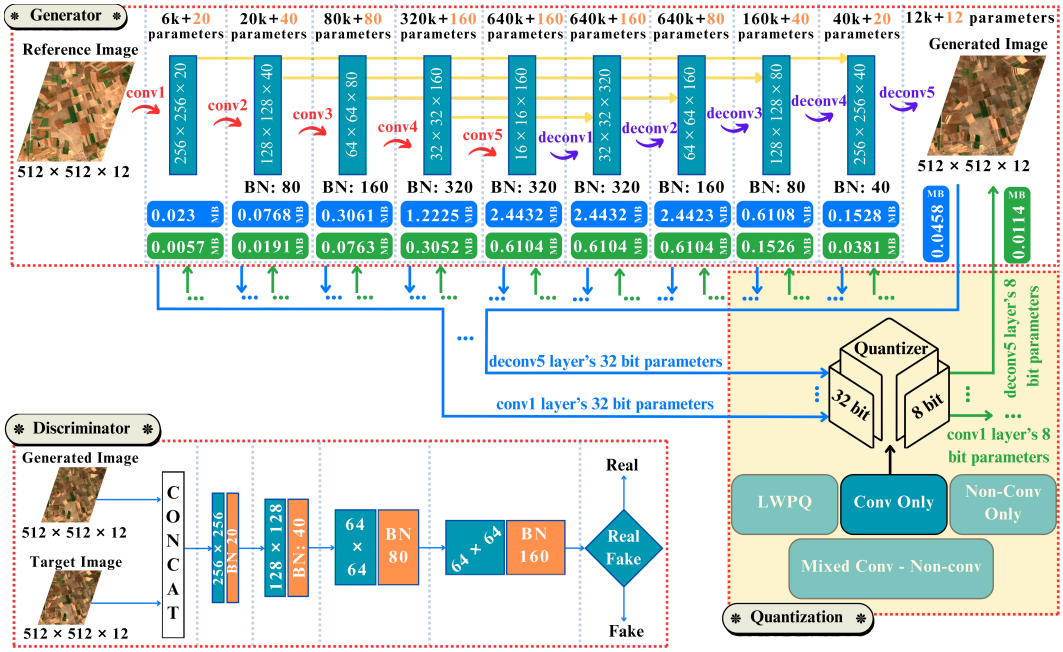


Figure 4.16 MPQ-MultiTempGAN Architecture.

4.4 Non-uniform Quantization

Non-uniform quantization is a technique where quantization levels are distributed unevenly across the range of input values, unlike uniform quantization which uses equally spaced levels. This approach is particularly beneficial when dealing with data that has a non-uniform distribution, allowing for more efficient representation and reduced quantization error.

In non-uniform quantization, smaller quantization steps are allocated to regions where the input data is more densely populated, and larger steps are used where the data is sparser. This method aligns the quantization levels more closely with the statistical properties of the input data, leading to improved performance in applications such as signal processing and neural network compression.

Uniform quantization performs well and remains highly acceptable, especially when symmetric quantization is applied. However, the primary motivation for exploring non-uniform approaches in this study was to further reduce the bit width, which in turn decreases bpp while maintaining or even improving SNR metrics. In this category, several approaches were evaluated and their details can be found in the following sections.

4.4.1 Logarithmic Quantization

Logarithmic quantization assigns quantization levels that are spaced logarithmically, making it particularly effective for datasets with high dynamic range where smaller magnitudes occur more frequently. Unlike uniform quantization, which maintains equal intervals between levels, logarithmic quantization compresses the range of larger values while maintaining finer granularity for smaller values. This results in better representation for small-valued parameters while still preserving the dynamic range [31].

4.4.1.1 Logarithmic Quantization Algorithm

Logarithmic Quantization approach is briefly explained in Algorithm 11.

Algorithm 11 Logarithmic Quantization

Input: Model checkpoint (`checkpoint_dir`), bit-width (`bitval`)
Output: Logarithmically quantized model checkpoint
Load the model checkpoint from `checkpoint_dir`
Retrieve the list of variables in the checkpoint
Initialize an empty list `assign_ops` to store weight updates
Compute the number of quantization levels: $\texttt{max_levels} = 2^{\texttt{bitval}}$
for each variable `var` in the checkpoint **do**
    Load the tensor values of `var` from the checkpoint
    Flatten `var` into a 1D array for processing
    if `var` contains only zeros or insufficient unique values **then**
        Skip quantization and retain original precision
    end if
    Compute the sign of the weights: $\texttt{sign} = \texttt{sign(var)}$
    Compute absolute values: $\texttt{var_abs} = |\texttt{var}|$
    Avoid log(0) by setting a small epsilon: $\texttt{var_abs} = \max(\texttt{var_abs}, \epsilon)$
    Compute logarithmic levels: $\texttt{log_levels} = \texttt{linspace}(\log(\min(\texttt{var_abs})), \log(\max(\texttt{var_abs})), \texttt{max_levels})$
    Convert variable to log scale: $\texttt{log_var} = \log(\texttt{var_abs})$
    Assign each value to the closest log level: $\texttt{quantized_indices} = \texttt{digitize}(\texttt{log_var}, \texttt{log_levels}) - 1$
    Clip indices to ensure valid range: $\texttt{quantized_indices} = \texttt{clip}(\texttt{quantized_indices}, 0, \texttt{max_levels} - 1)$
    Map indices back to logarithmic values: $\texttt{quantized_values} = \exp(\texttt{log_levels}[\texttt{quantized_indices}])$
    Restore original signs: $\texttt{quantized_weights} = \texttt{sign} \times \texttt{quantized_values}$
    Reshape `quantized_weights` to the original tensor shape
    Convert `quantized_weights` to `float32` and append to `assign_ops`
end for
Initialize a TensorFlow session and run `assign_ops` to update all variables
Save the quantized model checkpoint at a specified path

This method quantizes all 36 generator variables and their corresponding 2,560,252 parameters. Based on bit precision, the number of logarithmic levels is calculated as: $N_{\text{logarithmic_levels}} = 2^{\text{bit_precision}}$. These levels must be stored for reconstruction purpose along with `log_min` and `log_max` (two parameters per variable), which

means the total number of side information parameters will be calculated as follows:

$$N_{\text{side_information_parameters}} = N_{\text{variables}} \times (N_{\text{logarithmic_levels}} + 2) \tag{4.7}$$

Observations from uniform and mixed-precision quantization were persuasive enough to start evaluating non-uniform approaches with 6-bit and 8-bit precision, and only if 6-bit precision yields results closely matching the original model, proceed to 4-bit and lower. Therefore, 6 and 8-bit precisions require 2,376 and 9,288 side information parameters, respectively. It is worth noting that unlike uniform and mixed-precision approaches, here the bit precisions affect the number of side information parameters, and as a result bpp and memory. Table 4.13 provides the performance metrics of this quantization technique, and the visual results are displayed in Figure 4.17.

Table 4.13 Performance metrics of logarithmic quantization approach for 6 and 8 bits across all three datasets.

Bit	Dataset	Memory (MB)	bpp	SNR	LMSE	Ratio
6	MSI Pair-1	1.84	0.0111	20.05	1.05	5.31
	MSI Pair-2	1.84	0.0111	18.86	0.78	5.31
	MSI Pair-3	1.84	0.0111	17.72	1.17	5.31
	Avg.	**1.84**	**0.0111**	**18.88**	**1.00**	**5.31**
8	MSI Pair-1	2.47	0.015	26.33	0.50	3.94
	MSI Pair-2	2.47	0.015	25.47	0.35	3.94
	MSI Pair-3	2.47	0.015	23.57	0.44	3.94
	Avg.	**2.47**	**0.015**	**25.12**	**0.43**	**3.94**



(a) Reference Image (b) Target Image (c) MultiTempGAN (SNR: 27.87)



(d) 6 Bits Quantization (SNR: 20.05) (e) 8 Bits Quantization (SNR: 26.33)

Figure 4.17 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with logarithmic approach.

4.4.2 Dynamic Fixed-Point Quantization (DFPQ)

DFPQ adjusts the fixed-point representation dynamically based on the range of values within each layer or dataset. Unlike standard fixed-point quantization, where scaling factors are predetermined, this approach dynamically recalculates the scaling factor to better capture variations in the distribution of weights [32]. This makes DFPQ more flexible and adaptive, particularly for deep learning models with large variations in parameter magnitudes.

4.4.2.1 DFPQ Algorithm

The flexible and adaptive DFPQ is explained completely in Algorithm 12.

Algorithm 12 Dynamic Fixed-Point Quantization

Input: Model checkpoint (`checkpoint_dir`), bit-width (`bitval`)
Output: Dynamically fixed-point quantized model checkpoint
Load the model checkpoint from `checkpoint_dir`
Retrieve the list of variables in the checkpoint
Initialize an empty list `assign_ops` to store weight updates
Compute the maximum representable value: $\texttt{max_val} = 2^{\texttt{bitval}-1} - 1$
for each variable `var` in the checkpoint **do**
    Load the tensor values of `var` from the checkpoint
    Flatten `var` into a 1D array for processing
    if `var` contains only zeros or insufficient unique values **then**
        Skip quantization and retain original precision
    end if
    Compute the maximum absolute value: $\texttt{max_abs_val} = \max(|\texttt{var}|)$
    Compute dynamic scaling factor: $\texttt{scaling_factor} = \frac{\texttt{max_abs_val}}{\texttt{max_val}}$ (if $\texttt{max_abs_val} \neq 0$, otherwise set to 1)
    Quantize the weights: $\texttt{quantized_weights} = \texttt{round}(\frac{\texttt{var}}{\texttt{scaling_factor}})$
    Clip quantized values to the range: $[-\texttt{max_val}, \texttt{max_val}]$
    Dequantize weights: $\texttt{dequantized_weights} = \texttt{quantized_weights} \times \texttt{scaling_factor}$
    Reshape `dequantized_weights` to match the original tensor shape
    Convert `dequantized_weights` to `float32` and append to `assign_ops`
end for
Initialize a TensorFlow session and run `assign_ops` to update all variables
Save the dynamically fixed-point quantized model checkpoint at a specified path

DFPQ, like all non-uniform quantization techniques in this study, quantizes all 2,560,252 generator parameters across 36 variables (tensors) that store these parameters. So, 36 side information parameters are needed to make the quantized model reconstructable. Table 4.14 represents the numerical performance metrics of DFPQ technique, while Figure 4.18 depicts the visual comparison of the results when this approach is applied with different bit precisions.

Table 4.14 Performance metrics of DFPQ quantization approach for 6 and 8 bits across all three datasets.

Bit	Dataset	Memory (MB)	bpp	SNR	LMSE	Ratio
6	MSI Pair-1	1.83	0.0111	23.92	4.79	5.33
	MSI Pair-2	1.83	0.0111	21.28	7.06	5.33
	MSI Pair-3	1.83	0.0111	20.38	6.47	5.33
	Avg.	**1.83**	**0.0111**	**21.86**	**6.11**	**5.33**
8	MSI Pair-1	2.44	0.0148	27.45	0.74	3.99
	MSI Pair-2	2.44	0.0148	25.67	0.65	3.99
	MSI Pair-3	2.44	0.0148	24.22	0.77	3.99
	Avg.	**2.44**	**0.0148**	**25.78**	**0.72**	**3.99**



(a) Reference Image (b) Target Image (c) MultiTempGAN (SNR: 27.87)



(d) 6 Bits Quantization (SNR: 23.92) (e) 8 Bits Quantization (SNR: 27.45)

Figure 4.18 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with DFPQ approach.

4.4.3 Learned Step Size Quantization (LSSQ)

This approach incorporates learnable step sizes into the quantization process, where the step size determines the resolution of quantized values by scaling

the floating-point weights into discrete levels. Unlike conventional quantization methods that use predefined step sizes, this method optimizes step sizes during training to minimize reconstruction errors [33]. By allowing step sizes to be trained alongside model parameters, it enhances adaptability and minimizes information loss. This technique is particularly useful for end-to-end learning in deep neural networks.

4.4.3.1 Determining Step Size in LSSQ

Step size in quantization is the smallest difference between two consecutive quantization levels. In LSSQ, it is dynamically optimized during training to minimize reconstruction errors and improve quantization efficiency.

In the work of Esser et al. [33] the authors propose a method to train deep networks with low-precision operations by learning the quantizer's step size during training. However, as in this thesis work post-training quantization approaches are utilized, the step size is not learned during training, but instead computed directly from the weight statistics after training using a previously saved model checkpoint [24] [34]. The initial step size for LSSQ is calculated as demonstrated in Equation 4.8

$$initial_step_size = \frac{2 \times \text{Mean}(|w|)}{\sqrt{2^b - 1}} \tag{4.8}$$

where:

- Mean$(|w|)$ represents the mean absolute value of the model weights.
- b is the bit width used for quantization.
- $2^b - 1$ represents the total number of quantization levels.

This formulation ensures that the step size is adapted to the weight distribution, allowing efficient and stable quantization. Also, using `step_size = tf.Variable(initial_step_size, trainable=True, dtype=tf.float32, name='step_size')` in the quantization code ensures that TensorFlow can update the `step_size` during optimization to minimize quantization error.

4.4.3.2 LSSQ Algorithm

The algorithm 13 provides a rough demonstration of the utilization of the LSSQ approach.

Algorithm 13 Learned Step Size Quantization

Input: Model checkpoint (`checkpoint_dir`), bit-width (`bitval`)
Output: Quantized model checkpoint with optimized step size
Load the model checkpoint from `checkpoint_dir`
Retrieve the list of variables in the checkpoint
Initialize step size variable `step_size` = `initial_step_size`
Initialize an empty list `assign_ops` to store quantized weight updates
for each variable `var` in the checkpoint **do**
    Load the tensor values of `var`
    if `var` has insufficient unique values **then**
        Skip quantization and retain original values
    else
        Compute quantized values:

$$\texttt{quantized} = \texttt{round}\left(\frac{\texttt{var}}{\texttt{step_size}}\right) \times \texttt{step_size}$$

        Clip values to the valid quantization range:

$$\texttt{quantized} = \texttt{clip}(\texttt{quantized}, -\texttt{max_val}, \texttt{max_val} - \texttt{step_size})$$

        Store quantized weights in `assign_ops`
    end if
end for
Initialize a TensorFlow session and run `assign_ops` to update all variables
Save the quantized model checkpoint at a specified path

In this technique, 36 side information parameters are required after quantization. Table 4.15 presents the LSSQ performance metrics with the initial step size calculated as 0.014891, 0.014101, and 0.015079 for MSI Pair-1, MSI Pair-2, and MSI Pair-3, respectively, in 6-bit precision, and 0.007401, 0.007009, and 0.007495 in 8-bit precision. Figure 4.19 illustrates the visual results for the images (patches) generated using this approach for a randomly selected patch from the MSI Pair-1 dataset.

Table 4.15 Performance metrics of LSSQ quantization approach for 6 and 8 bits across all three datasets.

Bit	Dataset	Memory (MB)	bpp	SNR	LMSE	Ratio
6	MSI Pair-1	1.83	0.0111	23.02	4.26	5.33
	MSI Pair-2	1.83	0.0111	19.58	2.33	5.33
	MSI Pair-3	1.83	0.0111	19.22	2.72	5.33
	Avg.	**1.83**	**0.0111**	**20.60**	**3.10**	**5.33**
8	MSI Pair-1	2.44	0.0148	27.14	1.22	3.99
	MSI Pair-2	2.44	0.0148	25.75	0.79	3.99
	MSI Pair-3	2.44	0.0148	24.03	1.00	3.99
	Avg.	**2.44**	**0.0148**	**25.64**	**1.00**	**3.99**



(a) Reference Image (b) Target Image (c) MultiTempGAN (SNR: 27.87)

(d) 6 Bits Quantization (SNR: 23.02) (e) 8 Bits Quantization (SNR: 27.14)

Figure 4.19 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with LSSQ approach.

4.4.4 Scalar Dead Zone Quantization (SDZQ)

This approach introduces a dead-zone around zero to enhance the handling of sparse weight distributions. Unlike standard quantization methods, which treat all values

uniformly, scalar dead-zone quantization sets values close to zero to exactly zero. This reduces noise in near-zero values and improves sparsity preservation, leading to better compression and efficiency [35].

4.4.4.1 Determining Dead-Zone Width (DZW) in SDZQ

There is no universally fixed formula to determine the DZW in SDZQ, but there are practical methods and heuristics based on the distribution of the weights and the quantization objectives. In the following, two main methods to determine an appropriate dead-zone width are explained.

- **Range-Based Method:** In this approach, the DZW is defined as a fraction of the range of the weights:

$$DZW = \alpha \times (w_{\text{max}} - w_{\text{min}}) \tag{4.9}$$

  where α is a small fraction, w_{max} is the maximum weight, and w_{min} is the minimum weight. This method considers the overall spread of the weight values.

- **Standard Deviation-Based Method:** Alternatively, this approach sets the DZW proportional to the standard deviation (σ) of the weights:

$$DZW = k \times \sigma \tag{4.10}$$

  where k is a scaling factor. This method adapts the DZW based on the dispersion of the weight values.

The selection of k and α depends on the experimental analysis and quantization performance based on:

- **Data Distribution:** Higher variance in weights may require a larger dead-zone width to effectively reduce noise.
- **Desired Sparsity:** A larger dead-zone width increases sparsity, which improves compression but may impact accuracy.
- **Experimental Tuning:** Typical values for k range between 0.1 and 0.5, while α is usually set between 0.01 and 0.05 based on experimental validation. In this study, k is set to 0.15 and α to 0.02 after observing the results of various combinations.

4.4.4.2 SDZQ Algorithm

The algorithm 14 gives an overview of the SDZQ approach.

Algorithm 14 Scalar Dead-Zone Quantization

Input: Model checkpoint (`checkpoint_dir`), bit-width (`bitval`), dead-zone width (`dead_zone_width`)
Output: Quantized model checkpoint with dead-zone applied
Load the model checkpoint from `checkpoint_dir`
Retrieve the list of variables in the checkpoint
Compute the maximum representable value: $\texttt{max_val} = 2^{(\texttt{bitval}-1)} - 1$
Initialize an empty list `assign_ops` to store quantized weight updates
for each variable `var` in the checkpoint **do**
    Load the tensor values of `var`
    if `var` has insufficient unique values **then**
        Skip quantization and retain original values
    else
        Compute the scaling factor: $\texttt{scale} = \frac{\max(|\texttt{var}|)}{\texttt{max_val}}$
        Apply dead-zone quantization:

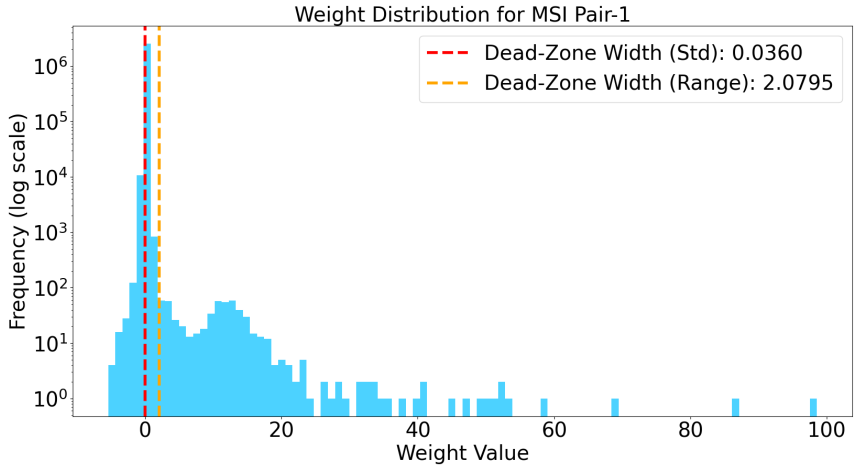
$$\texttt{quantized} = \begin{cases} 0, & \text{if } |\texttt{var}| < \texttt{dead_zone_width} \\ \texttt{round}\left(\frac{\texttt{var}}{\texttt{scale}}\right) \times \texttt{scale}, & \text{otherwise} \end{cases}$$

        Clip values to the valid quantization range:

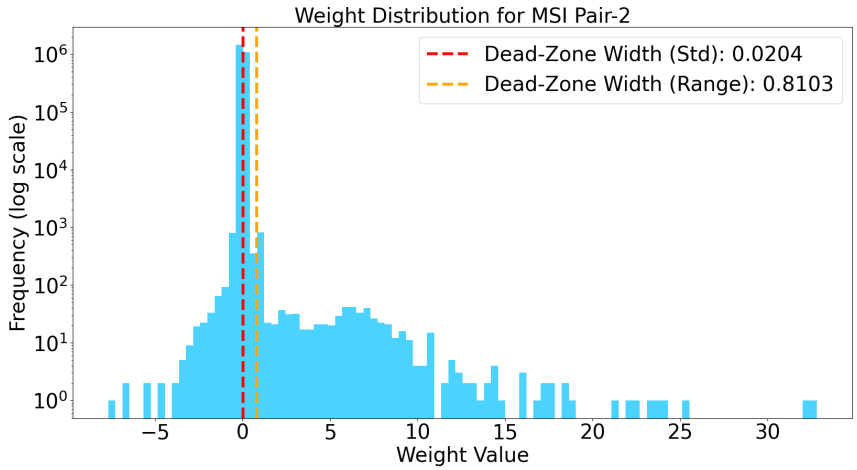
$$\texttt{quantized} = \texttt{clip}(\texttt{quantized}, -\texttt{max_val} \times \texttt{scale}, \texttt{max_val} \times \texttt{scale})$$

        Store quantized weights in `assign_ops`
    end if
end for
Initialize a TensorFlow session and run `assign_ops` to update all variables
Save the quantized model checkpoint at a specified path

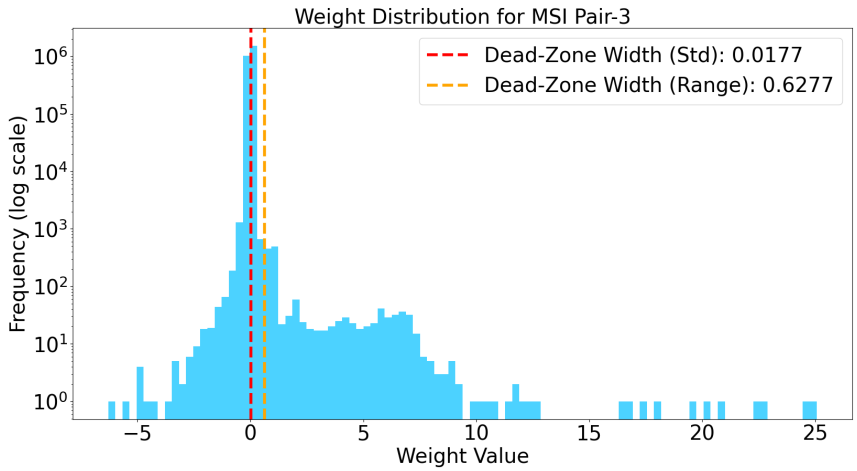
SDZQ quantizes all generator variables and also needs to store a DZW value for reconstruction. Thus, a total of 37 side information parameters are required when the model is quantized using this approach. Figure 4.20 shows the selected DZW values for each of the datasets based on each of the mentioned heuristics and Table 4.16 represents the performance metrics for each bit precision using each of these DZW values. Finally, the generated images (patches) for a sample randomly selected from the MSI Pair-1 dataset are demonstrated for each bit precision and the DZW selection heuristic in Figure 4.21.



(a) MSI Pair-1



(b) MSI Pair-2



(c) MSI Pair-3

Figure 4.20 Visualization of DZW based on both range and standard deviation of the model weights.

In MSI Pair-1 dataset, DZW = 0.036 produces the best results in terms of SNR among all other evaluated DZW values, including range-based DZW for 4, 6, and 8-bit precisions. This means that the distribution of weights of MSI Pair-1 dataset are more compatible with selecting DZW based on the standard deviation of the weights.

In MSI Pair-2 dataset, DZW = 0.0204 which is selected based on the standard deviation results in the best SNR value for 4 and 8-bit precisions, while in 6-bit, experimentally chosen DZW = 0.075 gives the best SNR metric.

In MSI Pair-3 dataset, the SNR results for 4 and 8-bits precisions peak when DZW = 0.1 and the SNR results for 6-bit precision reach the best value among all the other scenarios that were evaluated when DZW = 0.05 (which is also selected experimentally).

These findings suggest that while standard deviation-based DZW selection performs well in several cases, the heuristics mentioned in Section 4.4.4.1 are not always optimal for every dataset and bit precision configuration. Instead, dataset-specific tuning of the DZW is necessary to achieve the best trade-off between quantization efficiency and signal reconstruction quality. The variability in optimal DZW values across different datasets indicates that a universal approach may not be suitable, and adaptive methods that dynamically adjust DZW based on the characteristics of the dataset could be more effective.

Table 4.16 Performance metrics of SDZQ approach for 4, 6, and 8 bits across all three datasets.

Bit	Dataset	DZW	Memory (MB)	bpp	SNR	LMSE	Ratio
4	MSI Pair-1	0.036	1.22	0.0074	11.29	78.25	7.99
	MSI Pair-2	0.0204	1.22	0.0074	22.87	0.50	7.99
	MSI Pair-3	0.036	1.22	0.0074	19.49	0.48	7.99
	Avg.	-	**1.22**	**0.0074**	**17.88**	**26.41**	**7.99**
6	MSI Pair-1	0.036	1.83	0.0111	27.23	0.48	5.33
	MSI Pair-2	0.075	1.83	0.0111	24.86	0.35	5.33
	MSI Pair-3	0.05	1.83	0.0111	20.83	0.45	5.33
	Avg.	-	**1.83**	**0.0111**	**24.31**	**0.43**	**5.33**
8	MSI Pair-1	0.036	2.44	0.0148	26.03	0.46	3.99
	MSI Pair-2	0.0204	2.44	0.0148	25.83	0.33	3.99
	MSI Pair-3	0.1	2.44	0.0148	18.17	0.61	3.99
	Avg.	-	**2.44**	**0.0148**	**23.34**	**0.47**	**3.99**



(a) Reference Image (b) Target Image (c) MultiTempGAN (SNR: 27.87)



(d) 4 Bits and Dead Zone Width:0.036 (SNR: 11.29) (e) 6 Bits and Dead Zone Width:0.036 (SNR: 27.23) (f) 8 Bits and Dead Zone Width:0.036 (SNR: 26.03)

Figure 4.21 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with SDZQ approach.

4.4.5 Lloyd Max Quantization

The Lloyd–Max quantization algorithm, independently developed by Stuart P. Lloyd [36] and Joel Max [37] in the late 1950s, is a foundational method in signal processing for optimal quantization. This algorithm iteratively optimizes quantization levels and decision thresholds to minimize distortion, making it particularly effective for signals with Gaussian or uniform distributions. The process involves partitioning the input data into regions and assigning representative values (centroids) to each region, similar to the k-means clustering approach. Using k-means clustering to determine optimal quantization levels, it efficiently compresses weights while preserving accuracy.

4.4.5.1 Lloyd Max Quantization Algorithm

The algorithm 15 illustrates the application of the Lloyd Max approach in a simplified manner.

Algorithm 15 Lloyd-Max Quantization

Input: Model checkpoint (`checkpoint_dir`), bit-width (`bitval`)
Output: Quantized model checkpoint using Lloyd-Max optimization
Load the model checkpoint from `checkpoint_dir`
Retrieve the list of variables in the checkpoint
Compute the number of quantization levels: $\texttt{max_levels} = 2^{\texttt{bitval}}$
Initialize an empty list `assign_ops` to store quantized weight updates
for each variable `var` in the checkpoint **do**
    Load the tensor values of `var`
    Flatten `var` to a 1D array
    Determine the number of unique values and adjust `num_levels`
    if `num_levels` > 1 **then**
        Apply k-means clustering to determine optimal quantization levels
        Assign values to nearest cluster centers (quantization)
        Replace values with their corresponding cluster centers (dequantization)
    else
        Retain original variable values (skip quantization)
    end if
    Reshape quantized values back to original tensor shape
    Store quantized weights in `assign_ops`
end for
Initialize a TensorFlow session and run `assign_ops` to update all variables
Save the quantized model checkpoint at a specified path

In this approach, the bit precision which is noted as *b* in the Equation 4.11 affects the number of side information parameters, as the count of quantization levels depends on it. Thus, the number of side information parameters is calculated as shown in the following resulting in 2,304 and 9,216 parameters for 6 and 8-bit precisions respectively:

$$N_{\text{side_information_parameters}} = N_{\text{quantized_variables}} \times 2^b \tag{4.11}$$

Table 4.17 shows the performance metrics for each dataset in two different bit precisions using this approach and also the average SNR and LMSE over all datasets for each bit-width. In addition, Figure 4.22 indicates the visual results of the Lloyd Max technique.

Table 4.17 Performance metrics of Lloyd Max approach for 6 and 8 bits across all three datasets.

Bit	Dataset	Memory (MB)	bpp	SNR	LMSE	Ratio
6	MSI Pair-1	1.84	0.0111	26.53	1.58	5.30
	MSI Pair-2	1.84	0.0111	24.91	0.77	5.30
	MSI Pair-3	1.84	0.0111	23.60	0.90	5.30
	Avg.	**1.84**	**0.0111**	**25.01**	**1.08**	**5.30**
8	MSI Pair-1	2.47	0.015	27.57	0.58	3.94
	MSI Pair-2	2.47	0.015	26.03	0.42	3.94
	MSI Pair-3	2.47	0.015	24.36	0.65	3.94
	Avg.	**2.47**	**0.015**	**25.99**	**0.55**	**3.94**



(a) Reference Image (b) Target Image (c) MultiTempGAN (SNR: 27.87)

(d) 6 Bits Quantization (SNR: 26.53) (e) 8 Bits Quantization (SNR: 27.57)

Figure 4.22 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with Lloyd Max approach.

4.4.6 K-Means Quantization

K-Means quantization is a technique that applies K-Means clustering to determine quantization levels by grouping similar weight values into clusters. Instead of assuming a specific distribution, K-Means adapts to the actual distribution of the weights, making it suitable for complex or multimodal distributions. Each weight

value is assigned to the nearest centroid, and the centroid values replace the original weights. This method effectively reduces precision while preserving key patterns in weight distributions, improving efficiency with minimal loss in accuracy [38].

4.4.6.1 K-Means Quantization Algorithm

The algorithm 16 offers a high-level depiction of the K-Means Quantization approach.

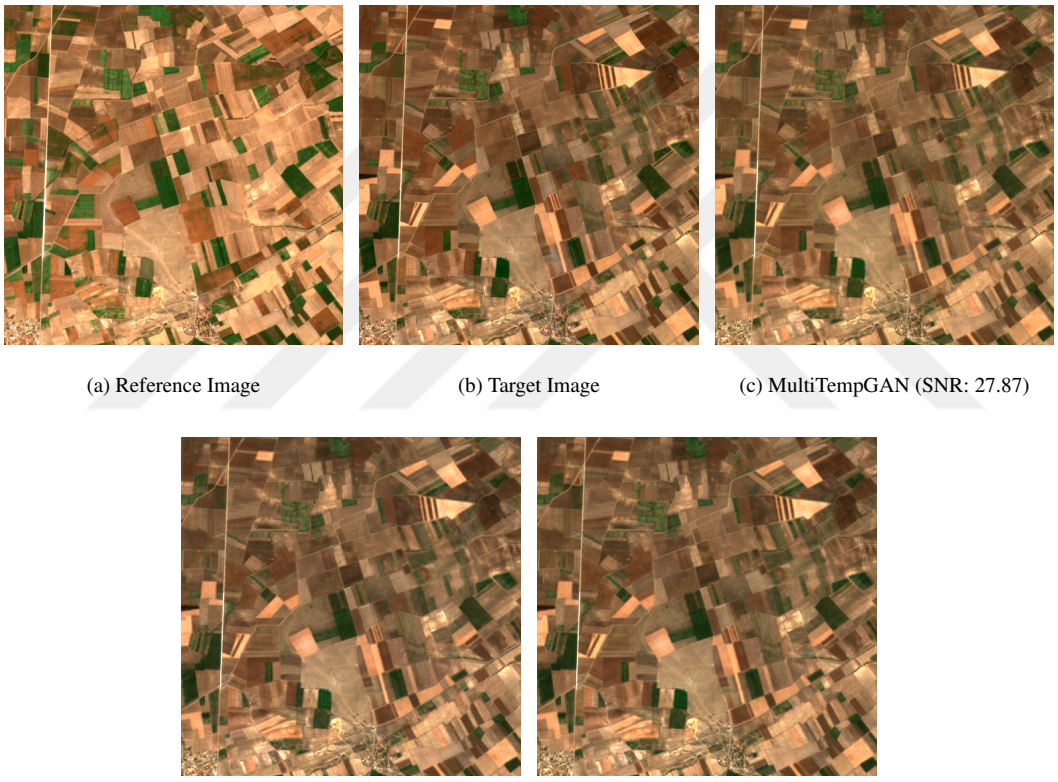
Algorithm 16 K-Means Quantization

Input: Model checkpoint (`checkpoint_dir`), bit-width (`bitval`)
Output: Quantized model checkpoint using K-Means clustering
Load the model checkpoint from `checkpoint_dir`
Retrieve the list of variables in the checkpoint
Compute the number of quantization levels: $\texttt{max_levels} = 2^{\texttt{bitval}}$
Initialize a dictionary `tf_vars` to store quantized variables
for each variable `var` in the checkpoint **do**
    Load the tensor values of `var`
    Flatten `var` to a 1D array
    Determine the number of unique values and adjust `num_levels`
    if `num_levels` > 1 **then**
        Apply K-Means clustering to determine quantization centroids
        Assign each value in `var` to the closest centroid
        Replace values with corresponding cluster centers (dequantization)
    else
        Retain original variable values (skip quantization)
    end if
    Reshape quantized values back to original tensor shape
    Store quantized weights in `tf_vars`
end for
Initialize a TensorFlow session and save the quantized model

The number of side information parameters in K-Means quantization is calculated similar to Lloyd Max using Equation 4.11. This means that quantizing to 6 and 8-bit precisions requires 2,304 and 9,216 side information parameters, respectively. Table 4.18 and Figure 4.23 contain the numerical and visual performance metrics of this approach.

Table 4.18 Performance metrics of K-Means approach for 6 and 8 bits across all three datasets.

Bit	Dataset	Memory (MB)	bpp	SNR	LMSE	Ratio
6	MSI Pair-1	1.84	0.0111	26.45	1.51	5.30
	MSI Pair-2	1.84	0.0111	24.97	0.76	5.30
	MSI Pair-3	1.84	0.0111	23.36	1.49	5.30
	Avg.	**1.84**	**0.0111**	**24.93**	**1.25**	**5.30**
8	MSI Pair-1	2.47	0.015	27.65	0.57	3.94
	MSI Pair-2	2.47	0.015	25.98	0.46	3.94
	MSI Pair-3	2.47	0.015	24.41	0.64	3.94
	Avg.	**2.47**	**0.015**	**26.01**	**0.56**	**3.94**



(a) Reference Image (b) Target Image (c) MultiTempGAN (SNR: 27.87)

(d) 6 Bits Quantization (SNR: 26.45) (e) 8 Bits Quantization (SNR: 27.65)

Figure 4.23 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with K-Means approach.

4.4.7 Piecewise Linear Quantization (PLQ)

PLQ is a technique that divides the data range into multiple segments and applies linear quantization within each segment. This method is particularly effective for handling non-linear weight distributions, as it reduces quantization-induced errors in data with complex variations [39]. By independently processing different parts

of the data range, PLQ balances simplicity and efficiency, making it a valuable approach when uniform quantization leads to significant distortions in certain regions.

4.4.7.1 Determining the Number of Segments in PLQ

The number of segments for PLQ is determined by the variability of the weight values using Equation 4.12:

$$N_{\text{segments}} = \max\left(2, \left\lfloor \text{base_scale} + \text{variability_scale} \cdot \log_{10}\left(\frac{w_{\max} - w_{\min}}{\sigma + \epsilon} + 1\right)\right\rfloor\right) \tag{4.12}$$

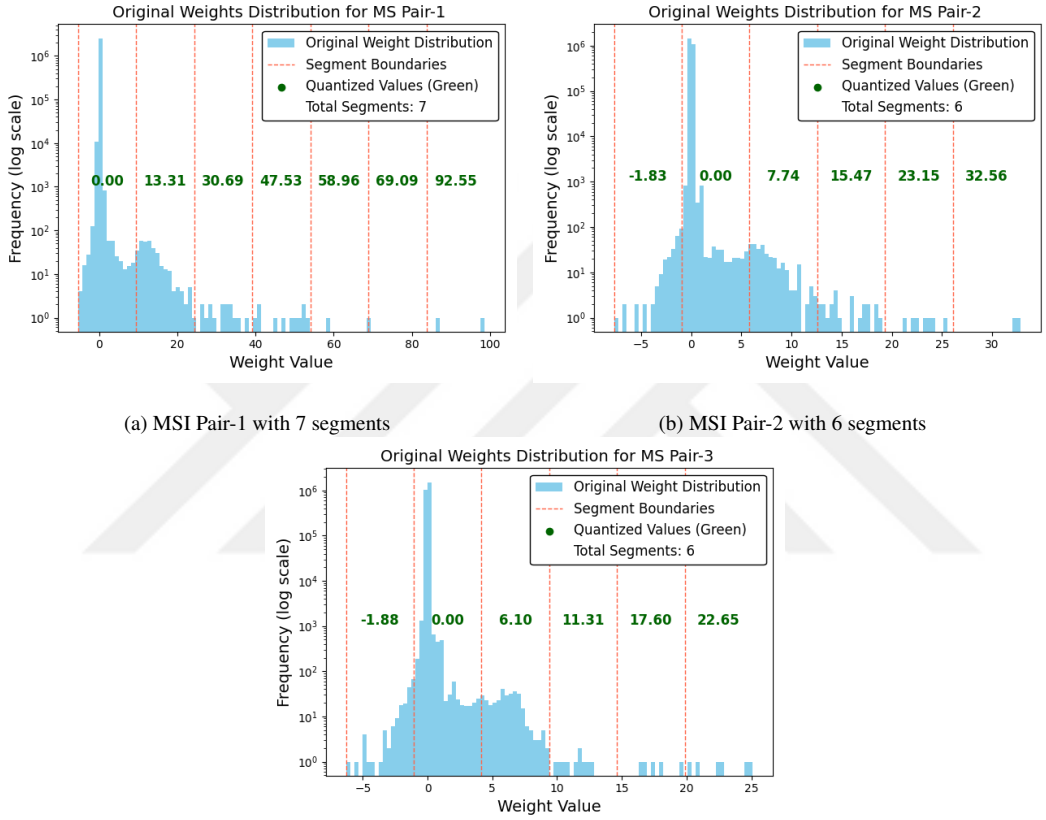
In this equation, the maximum function is used to ensure that even if the weight distribution is highly uniform, at least two segments are used to prevent shifting to uniform quantization because of having only one segment. Additionally,

- N_{segments}, $w_{\max}$, and $w_{\min}$ denote the number of segments required for PLQ, the maximum weight value, and the minimum weight value, respectively.
- σ represents the standard deviation of the weight values.
- ϵ is a small constant to prevent division by zero that is set to 1×10^{-8} in this work.
- **Base Scale:** Ensures a minimum number of segments for stable quantization. In this study, we set base_scale $= 1.5$ to maintain at least two quantization segments, ensuring that even when the weight distribution is narrow, the segmentation remains effective. This value prevents excessive merging of regions with distinct characteristics.
- **Variability Scale:** Adjusts the number of segments based on the variability of the weight distribution. The variability_scale $= 2.0$ was selected to make the number of segments more adaptive to different datasets. A higher variability scale increases the sensitivity of the segmentation to changes in the weight distribution, while a lower value results in a more uniform segmentation. The choice of 2.0 balances adaptation and prevents over-segmentation in cases with low variance.

After determining the number of segments, all weight parameters within a given segment are represented by a single value, as illustrated in Figure 4.24. Since

defining N segments requires $N+1$ boundaries and all 36 generator variables (tensors) are quantized in this approach, the total number of side information parameters, computed as shown in Equation 4.13, is $36 \times (7+1) = 288$ for the MSI Pair-1 dataset and $36 \times (6+1) = 252$ for the MSI Pair-2 and MSI Pair-3 datasets.

$$N_{\text{side_information_parameters}} = N_{\text{quantized_variables}} \times (N_{\text{segments}} + 1) \tag{4.13}$$



(a) MSI Pair-1 with 7 segments

(b) MSI Pair-2 with 6 segments

(c) MSI Pair-3 with 6 segments

Figure 4.24 Visualization of PLQ's Segments for each of the three datasets.

4.4.7.2 PLQ Algorithm

The algorithm 17 presents a simplified implementation of the PLQ approach.

Algorithm 17 Piecewise Linear Quantization

Input: Model checkpoint (`checkpoint_dir`), bit-width (`bitval`), number of segments (`num_segments`)
Output: Quantized model checkpoint with piecewise linear quantization
Load the model checkpoint from `checkpoint_dir`
Retrieve the list of variables in the checkpoint
Initialize an empty list `assign_ops` to store weight updates
for each variable `var` in the checkpoint **do**
    Load the tensor values of `var` from the checkpoint
    Flatten `var` into `var_flat` for processing
    Compute the min and max values of `var_flat`: $\texttt{min_val}, \texttt{max_val}$
    Define segment boundaries: $\texttt{segment_edges} = \texttt{linspace}(\texttt{min_val}, \texttt{max_val}, \texttt{num_segments} + 1)$
    Initialize `quantized_weights` with zeros
    for each segment i in `num_segments` **do**
        Determine segment bounds: $\texttt{lower_bound}, \texttt{upper_bound}$
        Identify weights within segment: `mask`
        if segment has valid weights **then**
            Compute quantization scale: $\texttt{scale} = \frac{\texttt{upper_bound} - \texttt{lower_bound}}{2^{\texttt{bitval}} - 1}$
            Quantize weights: $\texttt{quantized_segment} = \texttt{round}(\frac{\texttt{segment_values} - \texttt{lower_bound}}{\texttt{scale}}) \times \texttt{scale} + \texttt{lower_bound}$
            Store quantized values in `quantized_weights`
        end if
    end for
    Reshape `quantized_weights` back to the original shape of `var`
    Convert `quantized_weights` to `float32`
    Create a TensorFlow variable with the updated weights and append it to `assign_ops`
end for
Initialize a TensorFlow session and run `assign_ops` to update all variables
Save the quantized model checkpoint at a specified path

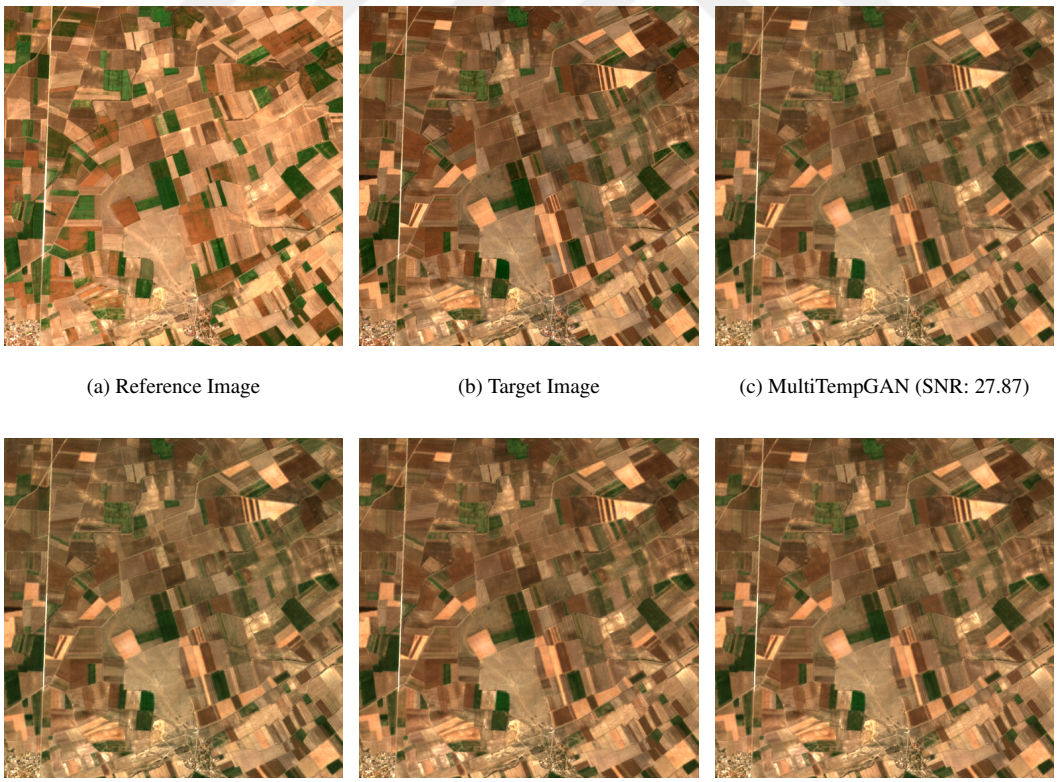
Table 4.19 demonstrates the performance metrics of the PLQ technique and it is worth noting despite the impact of the number of segments on the number of side information parameters which consequently affects memory and compression ratio, considering that the values are rounded (truncated) to two decimal places, the average values seem to remain the same while they actually show a slight change.

Figure 4.25 depicts the visual comparison of the generated images (patches) using

Table 4.19 Performance metrics of PLQ approach for 4, 6, and 8 bits across all three datasets.

Bit	Dataset	Segments	Memory (MB)	bpp	SNR	LMSE	Ratio
4	MSI Pair-1	7	1.22	0.0074	26.46	2.06	7.99
	MSI Pair-2	6	1.22	0.0074	24.55	1.75	7.99
	MSI Pair-3	6	1.22	0.0074	22.38	4.35	7.99
	Avg.	-	**1.22**	**0.0074**	**24.46**	**2.72**	**7.99**
6	MSI Pair-1	7	1.83	0.0111	27.84	0.57	5.33
	MSI Pair-2	6	1.83	0.0111	26.23	0.41	5.33
	MSI Pair-3	6	1.83	0.0111	24.60	0.57	5.33
	Avg.	-	**1.83**	**0.0111**	**26.22**	**0.52**	**5.33**
8	MSI Pair-1	7	2.44	0.0148	27.88	0.46	3.99
	MSI Pair-2	6	2.44	0.0148	26.42	0.33	3.99
	MSI Pair-3	6	2.44	0.0148	24.73	0.40	3.99
	Avg.	-	**2.44**	**0.0148**	**26.35**	**0.40**	**3.99**

each bit width for a randomly selected patch from the MSI Pair-1 dataset with the reference, target and MultiTempGAN's generated image.



(a) Reference Image (b) Target Image (c) MultiTempGAN (SNR: 27.87)

(d) 6 Bits and 7 segments (SNR: 26.46) (e) 8 Bits and 7 segments (SNR: 27.84) (f) 8 Bits and 7 segments (SNR: 27.88)

Figure 4.25 Visual comparison of reference, target, and generated images using MultiTempGAN and its quantized version with PLQ approach.

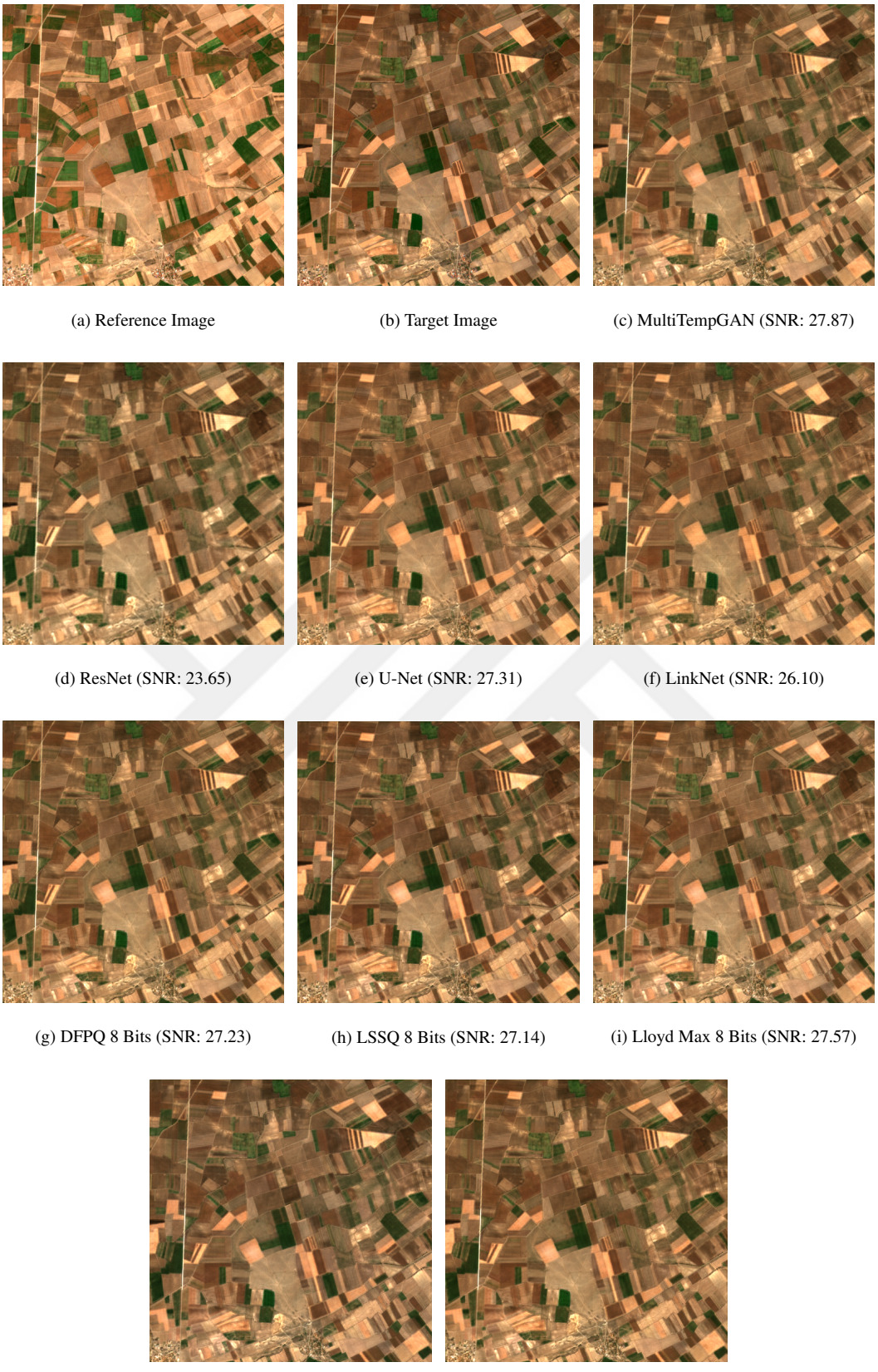
4.4.8 Performance Comparison of All Non-Uniform Approaches

It was previously shown that among all uniform and mixed-precision approaches, symmetric quantization with 8-bit precision outperformed the others, achieving an average SNR of 25.78 and an average LMSE of 0.72. When evaluating non-uniform approaches, it was observed that the Lloyd-Max, K-Means, and PLQ methods improved performance at the same bit precision as symmetric quantization (8-bit). Furthermore, PLQ with 6-bit precision not only enhanced both SNR and LMSE compared to the result of the symmetric quantization approach with 8-bit precision, but also outperformed the 8-bit precision results of all other approaches, leading to a 25% improvement in the bpp metric as well.

Table 4.20 summarizes the performance metrics averaged across all datasets with the bit precision that generates the best results in terms of SNR, LMSE, and bpp for each of the techniques evaluated in the non-uniform quantization category and Figure 4.26 demonstrates the visual comparison of results using the top five quantization approach in this category.

Table 4.20 Performance metrics of the best non-uniform approaches and all non-quantized deep learning-based models. SNR and LMSE values are averaged across all three datasets. The table includes bit precision, the number of side information parameters, memory usage (MB), and the compression ratio compared to MultiTempGAN, noted as Bits, S.I., Mem., and Ratio, respectively. S.I. for PLQ contains three different values due to the different number of quantization segments for each of the datasets.

	Method	**Bits**	**S.I.**	**Mem.**	**bpp**	**SNR**	**LMSE**	**Ratio**
Deep Learning Approaches	ResNet	-	-	11.00	0.0665	22.23	0.65	0.88
	U-Net	-	-	9.76	0.0590	25.49	0.45	1.00
	LinkNet	-	-	9.84	0.0595	24.82	0.49	0.99
	MultiTempGAN	-	-	9.76	0.0590	26.34	0.39	1.0
Non-uniformly Quantized Versions	Logarithmic	8	9288	2.47	0.015	25.12	0.43	3.94
	DFPQ	8	36	2.44	0.0148	25.78	0.72	3.99
	LSSQ	8	36	2.44	0.0148	25.64	1.00	3.99
	SDZQ	6	37	1.83	0.0111	24.31	0.43	5.33
	Lloyd Max	8	9216	2.47	0.015	25.99	0.55	3.94
	K-Means	8	9216	2.47	0.015	26.01	0.56	3.94
	PLQ	6	288 252 252	1.83	0.0111	26.22	0.52	5.33



(a) Reference Image (b) Target Image (c) MultiTempGAN (SNR: 27.87)

(d) ResNet (SNR: 23.65) (e) U-Net (SNR: 27.31) (f) LinkNet (SNR: 26.10)

(g) DFPQ 8 Bits (SNR: 27.23) (h) LSSQ 8 Bits (SNR: 27.14) (i) Lloyd Max 8 Bits (SNR: 27.57)

(j) K-Means 8 Bits (SNR: 27.65) (k) PLQ 6 Bits (SNR: 27.84)

Figure 4.26 Visual comparison of all deep learning-based models and top 5 non-uniformly quantized versions of MultiTempGAN using the best bit precision for each method.

As demonstrated below in Figure 4.27 PLQ with 6-bit precision outperforms all other quantization approaches and deep learning-based methods mentioned in this study. This makes it the best choice for designing the proposed model architecture given in Chapter 5.

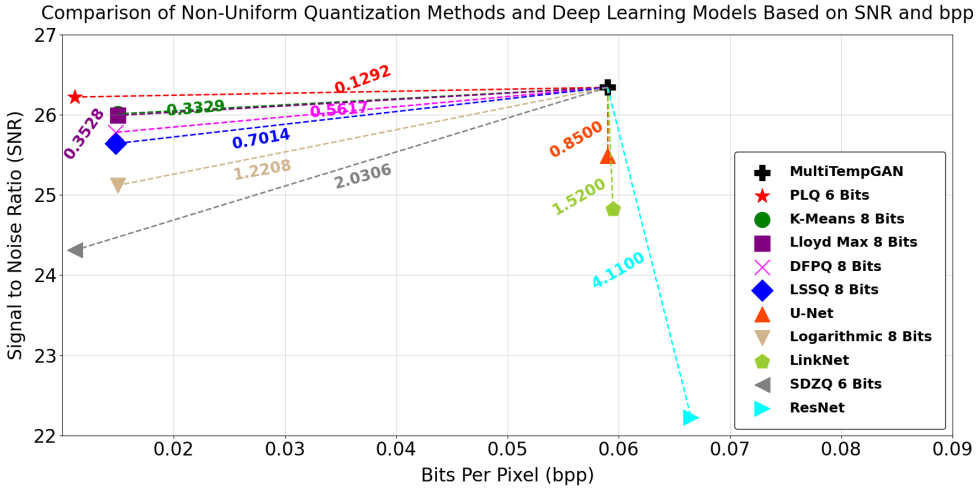


Figure 4.27 Comparison of all non-uniform approaches, ResNet, U-Net, LinkNet, and MultiTempGAN based on Euclidean distance calculated with SNR and bpp metrics.

5
Proposed Model: PLQ-MultiTempGAN

5.1 PLQ-MultiTempGAN Details

In Chapter 4, after assessing the numerical and visual performance metrics of various quantization methods, it was determined that PLQ with 6-bit precision achieved the highest ranking among all approaches in the three categories of quantification examined: uniform, mixed precision, and nonuniform quantization.

Following training of the model for 150 epochs using the same configurations specified in the study of Karaca et al. [11] and saving checkpoints for each of the three datasets, PLQ with a bit-width of 6 was applied to all generator weights by reloading the model from the saved checkpoint. After performing post-training quantization to 6-bit precision, the model was stored as a new checkpoint, which was then used for sampling across all datasets.

It should be noted that the weight distribution plays an important role in the selection of the number of segments in PLQ, and it is included in the proposed model structure illustrated in Figure 5.1. This architecture shows how post-training quantization affects the weights of different convolutional and deconvolutional layers in the original model.

In the structure named PLQ-MultiTempGAN and demonstrated in Figure 5.1, similar to the previously mentioned architectures pointed out in Sections 4.2.7 and 4.3.5, the non-uniformly quantized generator structure is shown at the top and the discriminator is shown at the bottom left. The quantization process is detailed at the bottom right. The quantized discriminator is not illustrated because this structure does not affect the sampling process after training is completed. The main difference between this structure and the previous ones is that 6-bit quantization is utilized, which results in a more compact GAN model.

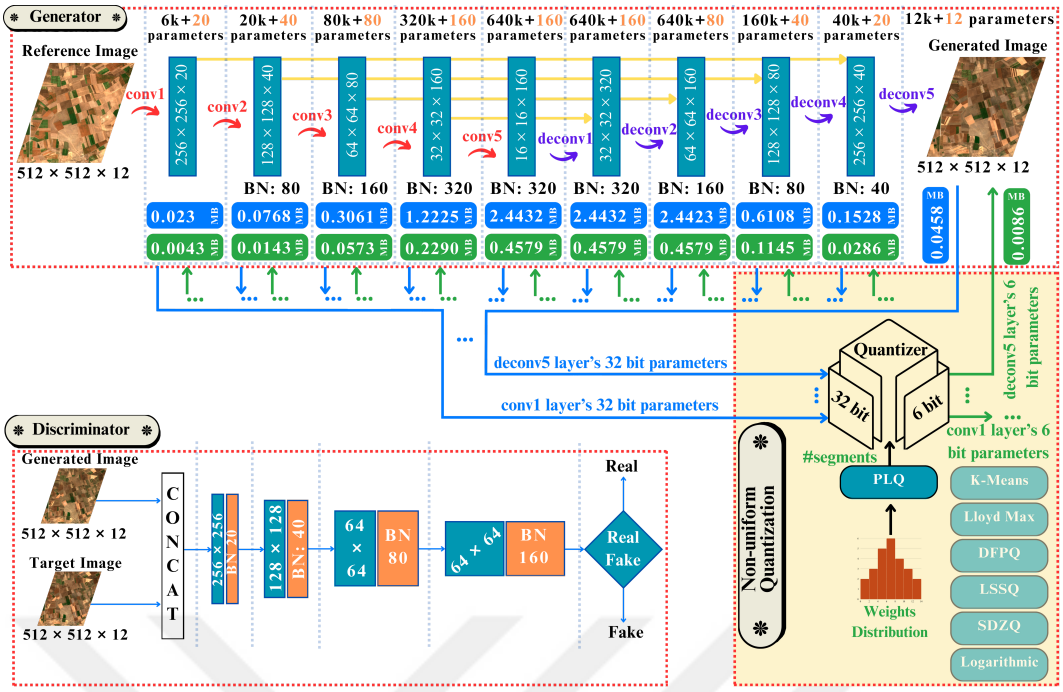


Figure 5.1 Proposed Model (PLQ-MultiTempGAN) Architecture.

Figure 5.2 illustrates the comparison of PLQ-MultiTempGAN with other quantized versions of MultiTempGAN using different approaches, as well as deep learning-based MS image compression models such as ResNet, U-Net, and LinkNet. The figure focuses on SNR and bpp metrics across all evaluated quantization techniques that demonstrated notable performance in this study. Although PLQ with 8-bit precision achieves slightly better performance than its 6-bit equivalent, the latter is selected because the primary objective is to maximize model compression while minimizing its impact on MS image compression accuracy. Consequently, the approximately 0.49% decrease in SNR from 26.35 to 26.22 is considered an acceptable trade-off for the 25% reduction in bpp from 0.0148 to 0.0111.

Similarly, Figure 5.3 compares the proposed model with other models based on LMSE and bpp. While in Figure 5.2, models closer to the top left are more accurate, in Figure 5.3, models closer to the bottom left perform better. As 6-bit PLQ meets these criteria in both plots, it demonstrates superior performance in terms of both SNR and LMSE.

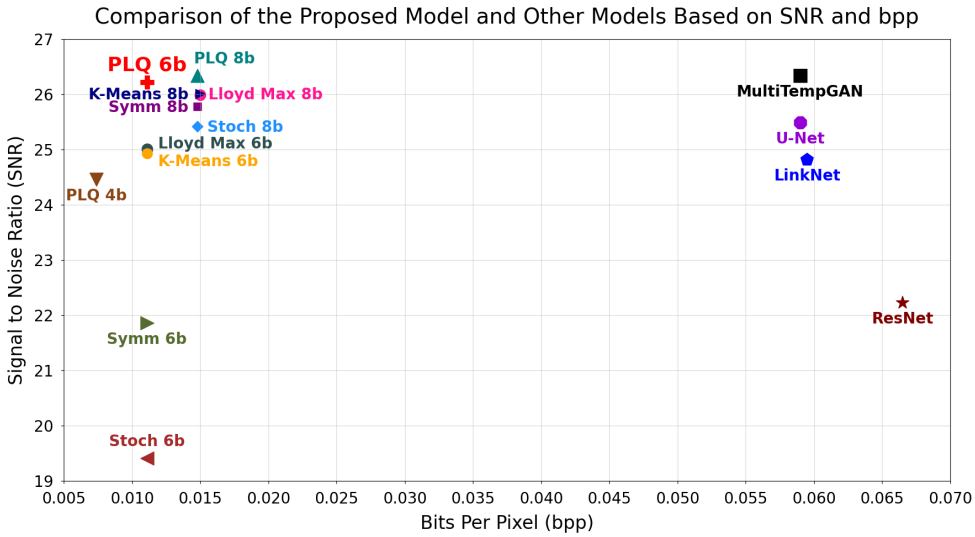


Figure 5.2 Comparison of the proposed model with other approaches based on SNR and bpp. Symm and Stoch denote symmetric and stochastic quantization respectively.

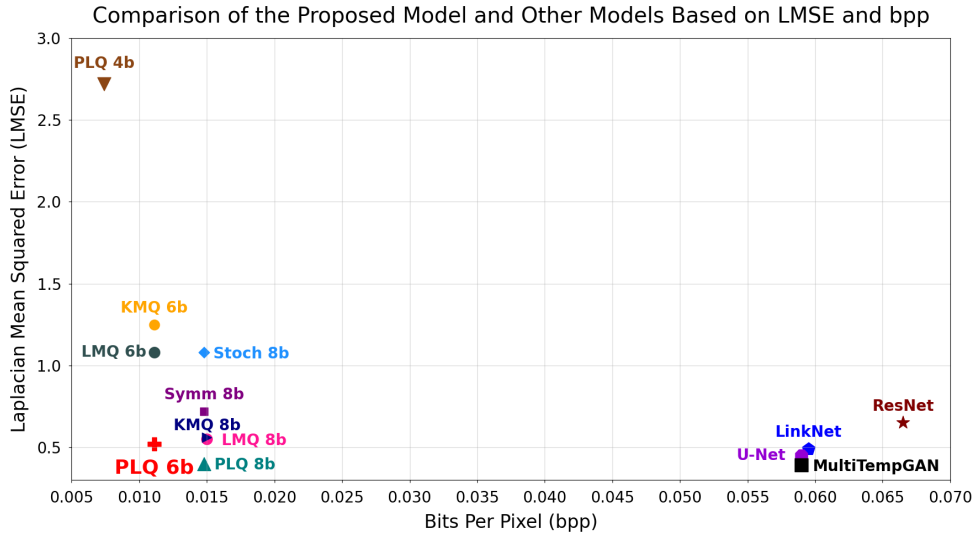


Figure 5.3 Comparison of the proposed model with other approaches based on LMSE and bpp. Symm and Stoch denote symmetric and stochastic quantization respectively.

5.1.1 Change Detection Maps (CDMs)

Despite acceptable performance metrics and visual results, the importance of CDMs should be emphasized. A CDM provides an output representation that highlights the differences between two images captured at different times. These maps are widely utilized in remote sensing, surveillance, medical imaging, and environmental monitoring. Since changes may not be easily distinguishable in visual comparisons (as shown in Figure 4.26), CDMs offer a more structured approach to identifying

alterations in image datasets.

Normally, the change detection method computes differences between two images taken at different times to identify regions of change, and this can be easily applied to the reference and target images (patches) in this work. However, it has also been applied to the target and generated images of various deep learning-based methods, as well as the best-performing non-uniformly quantized versions of MultiTempGAN, to assess changes and evaluate effectiveness of the models in regenerating the target image (patch).

In CDMs, both images are first converted to grayscale, and an absolute difference operation is performed to highlight variations. A thresholding step then filters out minor noise, producing a binary change detection map where white regions indicate significant changes, and black regions represent areas with no change. As with all visual results in this work, the images underwent a uniform lightening process by multiplying pixel values by 8 and clipping them to $[0, 1]$ before the CDM process.

Figure 5.4 presents the CDM between the original reference images and target images (patches). Since the original MS patches have dimensions of $512 \times 512 \times 12$, for practical visualization, the comparison is performed in the same manner as the dataset storage format, where each set of three channels from the reference patch is compared against the corresponding three channels of the target patch. However, for brevity, only the first three channels are included in the visualization.

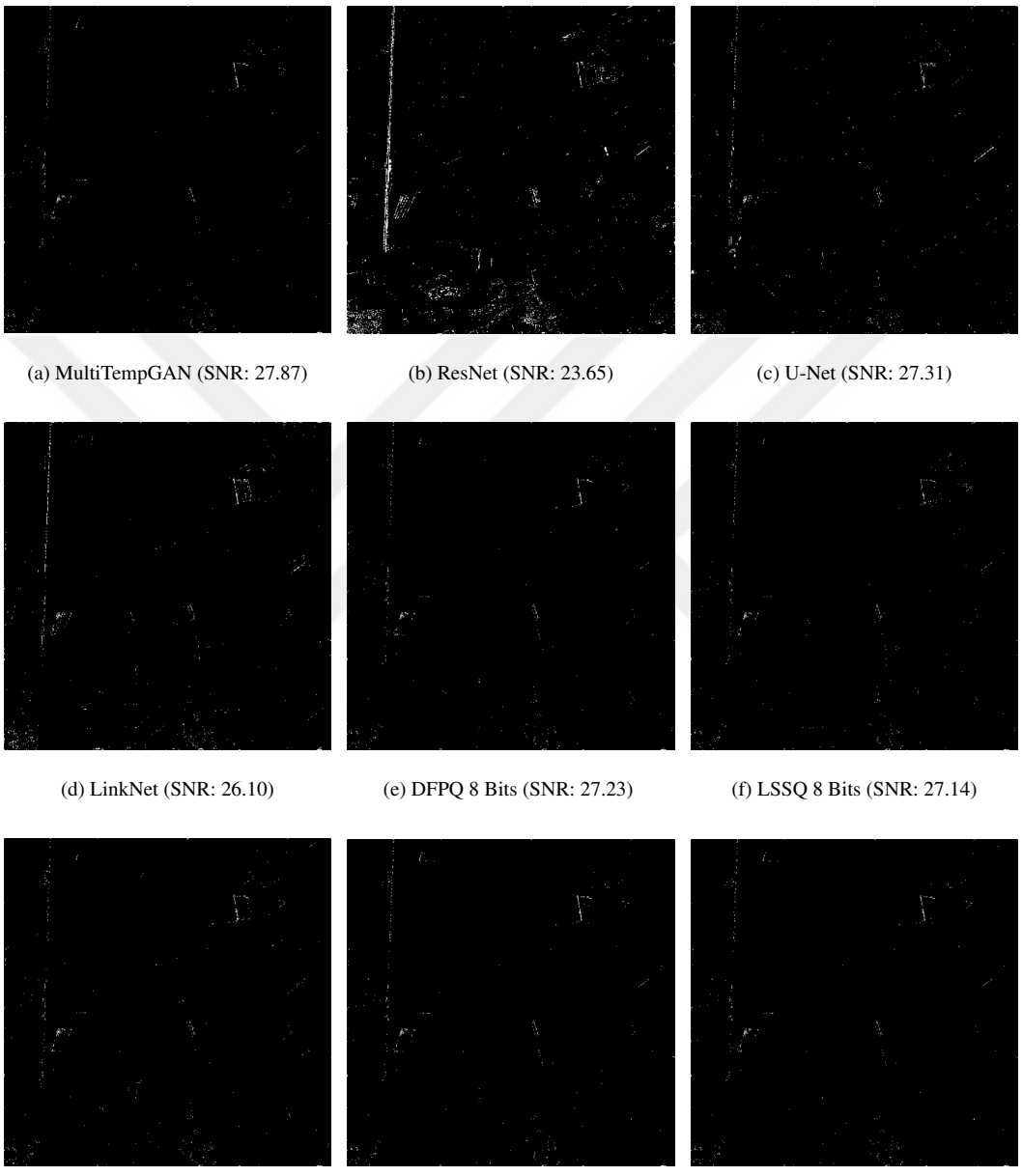


(a) Reference Image (b) Target Image (c) Computed CDM

Figure 5.4 CDM for the first three-channel RGB representation of the original reference and target images (patches), highlighting detected changes in white.

It is worth noting that this patch was randomly selected from the MSI Pair-1 dataset, where the temporal difference between the sensing dates is approximately 20 days. Given this time frame, observing significant changes is expected, particularly due to vegetation dynamics in the region.

Additionally, Figure 5.5 illustrates the CDMs computed for the first three channels of the RGB representation of the original target patch and the generated results of deep-learning-based models, along with the best-performing quantized versions of MultiTempGAN, including DFPQ, LSSQ, Lloyd Max, and K-Means (all at 8-bit precision), as well as the proposed model (PLQ with 6-bit precision).



(a) MultiTempGAN (SNR: 27.87) (b) ResNet (SNR: 23.65) (c) U-Net (SNR: 27.31)

(d) LinkNet (SNR: 26.10) (e) DFPQ 8 Bits (SNR: 27.23) (f) LSSQ 8 Bits (SNR: 27.14)

(g) Lloyd Max 8 Bits (SNR: 27.57) (h) K-Means 8 Bits (SNR: 27.65) (i) PLQ 6 Bits (SNR: 27.84)

Figure 5.5 CDMs for the target image (patch), the generated results of deep-learning-based models, and quantized versions of MultiTempGAN including 8-bit DFPQ, LSSQ, Lloyd Max, K-Means, and the proposed 6-bit PLQ model.

Figure 5.5 shows that the CDM results align with the performance metrics and visual comparisons discussed in the previous sections.

6
DISCUSSION

6.1 Development Insights of PLQ-MultiTempGAN

In this section, the logic behind adopting PLQ is explained, along with key design choices made to ensure optimal performance while maintaining efficient compression.

6.2 Motivation and Design Choices

The motivation for adopting PLQ is the need to achieve effective model compression while minimizing accuracy degradation in MS image compression tasks. Unlike uniform quantization, which applies a fixed step size across all weight values, PLQ dynamically partitions the weight distribution into segments, allowing finer quantization in regions where weights are more concentrated. This approach ensures that weight distributions specific to each dataset are better preserved, leading to improved quantization efficiency.

The choice of PLQ is particularly relevant for resource-constrained environments such as satellite-based image processing, where storage and computational resources are limited. By reducing the bit precision of the model while maintaining performance comparable to that of the original full-precision MultiTempGAN, PLQ-MultiTempGAN addresses the critical challenge of balancing compression and reconstruction accuracy.

6.3 Quantization Segmentation

A key factor in the effectiveness of PLQ is the adaptive selection of quantization segments, which is determined based on the variability of the weight distributions in each dataset. The segmentation strategy is guided by the standard deviation and range of the weights, ensuring that each segment captures meaningful variations in weight values. The number of segments is computed using a logarithmic scaling

function that considers both the base scale and the variability scale, as defined in Equation 4.12.

Experimental results demonstrate that selecting the appropriate number of segments prevents excessive quantization error while maintaining significant compression gains. By applying PLQ to all generator weights and using post-training quantization, the model is efficiently stored and reloaded with reduced memory requirements without requiring extensive re-training.

6.4 Limitations

Despite its advantages, PLQ-MultiTempGAN has certain limitations that should be considered for future improvements.

One limitation is the need for a dataset-specific tuning of the segmentation parameters, namely the base scale and the variability scale, which introduces additional complexity. Unlike uniform quantization, where a fixed bit-width is applied across the model, PLQ requires analyzing the weight distribution of each dataset separately to determine optimal segmentation.

Another limitation is the additional side information required to store segment boundaries. While PLQ reduces the overall bit-width of the quantized model, the need to store segment information as metadata slightly increases storage overhead compared to simpler quantization techniques like uniform quantization. For instance, while 8-bit symmetric quantization requires only 36 side information parameters, which is 1,152 storage bits, PLQ for the MSI Pair-1 dataset, which resulted in 7 segments, requires 288 side information parameters, corresponding to 9,216 storage bits.

A further limitation is that the effectiveness of PLQ-MultiTempGAN is highly dataset-specific. The findings, including the optimal number of segments and bit precision, are based on the datasets used in this study, meaning that they may not directly generalize to unseen datasets with significantly different weight distributions. Applying this approach to a new dataset would require re-evaluating and tuning the segmentation parameters, which adds an additional step before deployment.

6.5 Future Work

The following steps may be considered in a future study that focuses on enhancing the performance of the model proposed in this work.

One promising direction is the integration of mixed-precision quantization strategies that dynamically adjust bit-widths across different layers and quantization segments rather than applying a fixed bit-width uniformly to all weights. This could further optimize the trade-off between compression and accuracy by allocating higher precision to critical segments while using lower precision for less sensitive regions.

Additionally, this study was conducted using TensorFlow 1.15, which does not provide native support for quantization-aware training [26]. Future work could leverage newer TensorFlow versions that include quantization-aware training, allowing quantization to be incorporated during training rather than applied post-training. Unlike post-training quantization, quantization-aware training enables the model to learn and adapt to lower-precision weights throughout the training process, helping to minimize accuracy loss while achieving a more effective balance between compression and performance.

7
CONCLUSION

7.1 Summary and Insights

This study explored various quantization techniques for compressing a GAN-based multitemporal MS image compression model while maintaining high reconstruction quality. Among the evaluated methods, PLQ with 6-bit precision proved to be the most effective, offering a good balance between compression and accuracy. The results showed that adjusting quantization segments based on the dataset's weight distribution plays a significant role in minimizing performance loss. Although PLQ requires extra storage for segment boundaries, the overall reduction in bpp makes this trade-off acceptable. These findings emphasize the need for adaptive quantization strategies tailored to specific datasets for efficient MS image compression using deep learning.

As mentioned in the uniform and mixed-precision quantization sections, in some approaches in these two categories, 16-bit and 32-bit precision were also explored. However, the results showed that these higher bit-widths provided nearly identical SNR values in most cases. This can be attributed to floating-point precision: when weights are quantized to 16 or 32 bits and later dequantized back to floating-point (float32), the precision is effectively restored. The quantization error becomes negligible in both cases, as they are capable of capturing very fine details. This observation reinforces the idea that beyond a certain bit-width, further increasing precision does not yield significant benefits in terms of reconstruction accuracy while unnecessarily increasing storage and computational costs.

In some cases, the quantized model's SNR even showed slight improvements over the original model at lower bit precision. For instance, PLQ with 8-bit precision achieved an SNR of 27.88 for the MSI Pair-1 dataset, compared to 27.87 in the original model, while for MSI Pair-3, the 8-bit PLQ model attained an SNR of 24.73, slightly surpassing the original model's 24.72. This improvement can be attributed to the better adjustment of the weight values to the specific bit-width,

reducing numerical noise and enhancing generalization. Additionally, quantization can act as a form of regularization, preventing over-fitting and leading to more robust representations. These findings indicate that, in certain cases, carefully designed quantization strategies may not only preserve but even slightly enhance model performance.

REFERENCES

[1] B. Huang, *Satellite Data Compression*, B. Huang, Ed. Springer New York, 2011.

[2] Q. Du, J. E. Fowler, "Hyperspectral image compression using jpeg2000 and principal component analysis," *IEEE Geoscience and Remote Sensing Letters*, vol. 4, pp. 201–205, 2007.

[3] M. Klimesh, "Low complexity lossless compression of hyperspectral imagery via adaptive filtering," *Interplanetary Network Progress Report*, vol. 42, pp. 1–10, 2005.

[4] J. Song, Z. Zhang, X. Chen, "Lossless compression of hyperspectral imagery via rls filter," *IET Electronics Letters*, vol. 49, pp. 992–994, 2013.

[5] T. C. C. for Space Data Systems (CCSDS), *CCSDS123.0-B-2 Recommended Standard: Blue Book*. 2019.

[6] D. Liu, Y. Li, J. Lin, H. Li, F. Wu, "Deep learning-based video coding: A review and a case study," *ACM Computing Surveys*, vol. 53, pp. 1–35, 2020.

[7] T. Dumas, A. Roumy, C. Guillemot, "Autoencoder-based image compression: Can the learning be quantization independent?" In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2018, pp. 1188–1192.

[8] E. Agustsson, M. Tschannen, F. Mentzer, R. Timofte, L. V. Gool, "Generative adversarial networks for extreme learned image compression," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Seoul, Korea, 2019.

[9] D. Valsesia, E. Magli, "High-throughput onboard hyperspectral image compression with ground-based cnn reconstruction," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, pp. 9544–9553, 2019.

[10] P. Isola, J.-Y. Zhu, T. Zhou, A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, USA: IEEE, 2017, pp. 5967–5976, isbn: 978-1-5386-0457-1. doi: 10.1109/CVPR.2017.632.

[11] A. C. Karaca, O. Kara, M. K. Güllü, "MultiTempGAN: Multitemporal multispectral image compression framework using generative adversarial networks," *Journal of Visual Communication and Image Representation*, vol. 81, p. 103 385, 2021. doi: 10.1016/j.jvcir.2021.103385.

[12] O. Ronneberger, P. Fischer, T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, Springer, Cham, 2015, pp. 234–241.

[13] A. Chaurasia, E. Culurciello, "LinkNet: Exploiting encoder representations for efficient semantic segmentation," in *2017 IEEE Visual Communications and Image Processing (VCIP)*, IEEE, 2017, pp. 1–4.

[14] K. He, X. Zhang, S. Ren, J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[15] D. Chai, A. Bouzerdoum, "JPEG2000 Image Compression: An Overview," in *Proceedings of The Seventh Australian and New Zealand Intelligent Information Systems Conference (ANZIIS)*, Perth, WA, Australia, 2001, pp. 18–21, isbn: 1-74052-061-0. doi: 10.1109/ANZIIS.2001.974083.

[16] S. Fazli, S. Toofan, Z. Mehrara, "JPEG2000 Image Compression Using SVM and DWT," *International Journal of Science and Engineering Investigations*, vol. 1, no. 3, Apr. 2012, Research Institute of Physiology and Biotechnology, University of Zanjan, Iran; Electrical Engineering Department, Faculty of Engineering, Zanjan University, Zanjan, Iran, issn: 2251-8843.

[17] F. Kong, S. Zhao, Y. Li, D. Li, "End-to-end multispectral image compression framework based on adaptive multiscale feature extraction," *Journal of Electronic Imaging*, vol. 30, no. 1, p. 013 010, Feb. 2021. doi: 10.1117/1.JEI.30.1.013010.

[18] D. Barman, A. Hasnat, S. Begum, B. Barman, "A deep learning based multi-image compression technique," *Signal, Image and Video Processing (SIViP)*, vol. 18, pp. 407–416, Aug. 2024. doi: 10.1007/s11760-024-03163-8.

[19] D. Anuradha, G. C. Sekhar, A. Mishra, P. Thapar, T. D. Y. A. B. El-Ebiary, M. Syamala, "Efficient compression for remote sensing: Multispectral transform and deep recurrent neural networks for lossless hyperspectral imaging," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 15, no. 2, 2024, Accessed: 2024-01-29.

[20] A. C. Karaca, İ. Uçurmak, M. K. Güllü, "Efficient Resolution Enhancement of JPEG2000 Compressed Multispectral Images Using Deep Super-resolution Methods," in *Proceedings of the 2021 International Conference on INnovations in Intelligent SysTems and Applications (INISTA)*, Kocaeli, Turkey: IEEE, 2021. doi: 10.1109/INISTA52262.2021.9548453.

[21] "Sentinel-2 mission information." Accessed: 2024-01-29. (2024), [Online]. Available: https://sentiwiki.copernicus.eu/web/s2-mission.

[22] "Snap software download - esa." Accessed: 2024-01-29. (2024), [Online]. Available: https://step.esa.int/main/download/snap-download/.

[23] "Copernicus data space browser." Accessed: 2024-01-29. (2024), [Online]. Available: https://browser.dataspace.copernicus.eu/.

[24] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *arXiv preprint arXiv:1806.08342*, Jun. 2018. arXiv: 1806.08342 [cs.CV].

[25] TensorFlow Authors. "Post-training quantization." Accessed: 2025-01-29. (2024), [Online]. Available: https://www.tensorflow.org/model_optimization/guide/quantization/post_training.

[26] TensorFlow Authors. "Quantization aware training." Accessed: 2025-01-29. (2024), [Online]. Available: https://www.tensorflow.org/model_optimization/guide/quantization/training.

[27] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, K. Keutzer, "A survey of quantization methods for efficient neural network inference," in *Low-Power Computer Vision*, 1st, Chapman and Hall/CRC, 2022, p. 36, isbn: 9781003162810.

[28] B. Jacob *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, USA: Computer Vision Foundation / IEEE, 2018.

[29] S. Gupta, A. Agrawal, K. Gopalakrishnan, P. Narayanan, "Deep learning with limited numerical precision," in *Proceedings of the 32nd International Conference on Machine Learning*, F. Bach, D. Blei, Eds., ser. Proceedings of Machine Learning Research, vol. 37, Lille, France: PMLR, Jul. 2015, pp. 1737–1746.

[30] P. Micikevicius *et al.*, "Mixed precision training," in *International Conference on Learning Representations*, Published as a conference paper at ICLR 2018, 2018.

[31] D. V. Ess. "Algorithm - logarithmic signal companding - not just a good idea - it is μ-law." Accessed: 2024-01-29. (2024).

[32] L. Kummer, K. Sidak, T. Reichmann, W. Gansterer, "Adaptive precision training (adapt): A dynamic quantized training approach for dnns," Aug. 2021, Accessed: 2024-01-29.

[33] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, D. S. Modha, "Learned step size quantization," in *Proceedings of the International Conference on Learning Representations (ICLR)*, Accessed: 2024-01-29, San Jose, California, USA, 2020.

[34] M. Nagel, M. van Baalen, T. Blankevoort, M. Welling, "Data-free quantization through weight equalization and bias correction," 2024, Accessed: 2024-01-29.

[35] J. Yu, "Advantages of uniform scalar dead-zone quantization in image coding system," in *Proceedings of the 2004 International Conference on Communications, Circuits and Systems (ICCCAS)*, Chengdu, China, 2004, isbn: 0-7803-8647-7. doi: 10.1109/ICCCAS.2004.1346303.

[36] S. Lloyd, “Least squares quantization in pcm,” *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, Mar. 1982, issn: 0018-9448. doi: 10.1109/TIT.1982.1056489.

[37] J. Max, “Quantizing for minimum distortion,” *IRE Transactions on Information Theory*, vol. 6, no. 1, pp. 7–12, Mar. 1960, issn: 0096-1000. doi: 10.1109/TIT.1960.1057548.

[38] W. Lei, H. Chen, Y. Wu, “Compressing deep convolutional networks using k-means based on weights distribution,” in *Proceedings of the 2nd International Conference on Intelligent Information Processing (ICIIP ’17)*, 2017, pp. 1–6. doi: 10.1145/3144789.3144803.

[39] J. Fang, A. Shafiee, H. Abdel-Aziz, D. Thorsley, G. Georgiadis, J. Hassoun, “Post-training piecewise linear quantization for deep neural networks,” 2024, Accessed: 2024-01-29.

A PROJECT ENVIRONMENT SETUP

Table A.1 Software dependencies and their corresponding versions used in this thesis work.

Package Name	**Package Version**
TensorFlow	1.15
imageio	2.31.2
Keras-Applications	1.0.8
Keras-Preprocessing	1.1.2
matplotlib	3.5.3
numpy	1.21.6
opencv-python	4.10.0.84
pandas	1.1.5
Pillow	9.5.0
scikit-image	0.19.3
scikit-learn	1.0.2
scipy	1.7.3
six	1.16.0

PUBLICATIONS FROM THE THESIS

Conference Papers

1. N. S. Gharagoz and A. C. Karaca, "Q-MultiTempGAN: Post-training Quantization of Multitemporal Multispectral Image Compression Models," in Proceedings of the 5th International Conference on Signal Processing and Machine Learning (CONF-SPML 2025), Portsmouth, UK, 2025. doi: 10.54254/2755-2721/2025.20699.