Silesian
University
of Technology

FINAL PROJECT

Where deep learning meets Earth observation satellites: Compression and analysis of hyperspectral images

Michał Dominik RAJZER

Student identification number: 306081

Jakub SANECKI

Student identification number: 306085

Programme: Informatics

Specialisation: Computer Graphics and Software

SUPERVISOR

Jakub Nalepa, PhD, DSc

DEPARTMENT of Algorithmics and Software

Faculty of Automatic Control, Electronics and Computer Science

Gliwice 2026

Thesis title

Where deep learning meets Earth observation satellites: Compression and analysis of hyperspectral images

Abstract

Hyperspectral images (HSIs) provide detailed spectral information across hundreds of bands but generate significantly larger data volumes than traditional RGB images, creating challenges for storage and transmission in resource-constrained environments such as satellites. This thesis explores deep learning-based compression methods for HSIs, implementing and evaluating four models: LineRWKV for lossless compression, and RCAE2D1D, RCAE3D, and RCGDNAE for lossy compression. The models were trained and tested on the HySpecNet11k dataset, with performance assessed using metrics such as Bits Per Pixel Per Band (BPPPB), Peak Signal-to-Noise Ratio (PSNR), and Structural Similarity Index Measure (SSIM). Another important aspect of this work is the evaluation of the feasibility of these models for downstream tasks, specifically semantic segmentation, which was assessed using a separate model trained on the original, uncompressed data. The results indicate that deep learning-based compression techniques can effectively reduce HSI data sizes while maintaining differing quality for both visual assessment and semantic segmentation tasks depending on the model with some models excelling in visual quality and others in segmentation performance.

Key words

Deep Learning, Compression, Hyperspectral Images

Tytuł pracy

Gdzie uczenie głębokie spotyka się z satelitami obserwacji Ziemi: kompresja i analiza obrazów hiperspektralnych

Streszczenie

Obrazy hiperspektralne (HSI) dostarczają szczegółowych informacji spektralnych w setkach pasm, ale generują znacznie większe wolumeny danych niż tradycyjne obrazy RGB, co stwarza wyzwania związane z przechowywaniem i transmisją w środowiskach o ograniczonych zasobach, takich jak satelity. Niniejsza praca bada metody kompresji oparte na uczeniu głębokim dla HSI, implementując i oceniając cztery modele: LineRWKV do kompresji bezstratnej oraz RCAE2D1D, RCAE3D i RCGDNAE do kompresji stratnej. Modele zostały wytrenowane i przetestowane na zbiorze danych HySpecNet11k, a wydajność oceniono za pomocą metryk takich jak bity na piksel na pasmo (BPPPB), szczytowy stosunek sygnału do szumu (PSNR) oraz wskaźnik podobieństwa strukturalnego (SSIM). Kolejnym ważnym aspektem tej pracy jest ocena wykonalności tych modeli dla zadań następczych, w szczególności segmentacji semantycznej, która została oceniona za pomocą oddzielnego modelu wytrenowanego na oryginalnych, nieskompresowanych danych. Wyniki wskazują, że techniki kompresji oparte na uczeniu głębokim mogą skutecznie zmniejszyć rozmiary danych HSI, zachowując różną jakość zarówno do oceny wizualnej, jak i zadań segmentacji semantycznej w zależności od modelu, przy czym niektóre modele wyróżniają się jakością wizualną, a inne wydajnością segmentacji.

Słowa kluczowe

Uczenie Głębokie, Kompresja, Obrazy Hiperspektralne

Contents

Chapter 1

Introduction

1.1 Introduction, thesis goal

Earth observation began as an exclusively scientific endeavor and has developed into an essential service aimed at the observation, interpretation, and management of the various Earth systems. The widespread use of satellites and other remote earth observation platforms results in the generation of a massive amount of data daily. This data is the backbone of various applications including agriculture [4], forestry [1], surveillance [26], geology [24] and environmental monitoring [9].

Traditionally, a vast majority of the optical data has been acquired by panchromatic or multispectral cameras. Of these, the most common type of camera is the RGB (Red, Green, Blue), which mimic human eye by capturing data in three primary color channels: red (635 nm–700 nm), green (490 nm–560 nm), and blue (450 nm–490 nm) [3]. Although RGB images provide enough information for visual analysis and simple categorizations, they lack the depth of information required for more advanced applications.

This limitation has driven the development and deployment of hyperspectral imaging sensors (HSIS). Unlike their traditional counterparts, HSIS behave like an optical spectrometer on a per-pixel basis. HSIs go beyond three bands by providing data across dozens or even hundreds of very narrow spectral bands, which may include wavelengths outside the visible spectrum — from the ultraviolet (UV) and visible to the near-infrared (NIR) and short-wave infrared (SWIR) regions, typically ranging from 400 nm to 2.5 μm, with each band only a few nanometers wide [14]. The resulting dataset is a three-dimensional data cube that contains two dimensions of spatial information (x, y) and one dimension of spectral information (λ), which contains a distinct spectral signature or “fingerprint” for each imaged material. These signatures enable the differentiation of materials and objects that cannot be distinguished from each other in conventional images [24].

The use that can be made with this level of spectral detail is unparalleled. In agriculture applications, the use of HSIs helps to monitor nutrient deficiencies and diseases before they become visible to the naked eye [4]. In the management of forest resources, HSIs help to assess species distribution, weight, and pest infestations [1]. Geology applications include mineral mapping and soil composition analysis [24]. Additionally, HSIs play critical roles in surveillance and environmental applications by enabling the detection of camouflaged objects and monitoring of pollution levels [26, 9]. Most uses directly involve the analysis of the spectral signatures present in the data, allowing for more accurate and detailed assessments than traditional imaging methods. A common technique used in HSI analysis is semantic segmentation, where each pixel in the image is classified into different categories based on its spectral signature [21].

However, the informational density of HSIs comes at a cost. A single HSI pixel no longer contains three values, but a vector of hundreds [1]. This results in significantly larger data sizes compared to traditional images, posing challenges for storage, transmission, and processing, especially in resource-constrained environments like satellites. To mitigate these challenges, effective compression techniques are essential. Over the years, a variety of compression methods have been proposed, ranging from adaptations of traditional compression algorithms to deep learning-based approaches specifically designed for HSI [14].

In this thesis, the focus is placed on the exploration and evaluation of selected HSI compression methods based on deep learning, which already exist in the literature. The goal is to compare these methods in terms of Compression Ratio (CR), reconstruction quality, and computational requirements for both lossy and lossless compression. Another goal is to check the feasibility of these methods by including a simple semantic segmentation model in the evaluation pipeline to assess the impact of compression on downstream tasks. To achieve this, a custom data pipeline was developed to facilitate the training and evaluation of the selected methods on a standard dataset [7]. We divided the work so that one author focused on two lossy compression methods while the other focused on one lossless and one lossy compression method, and both authors collaborated on the data pipeline and the semantic segmentation model.

1.2 Chapter overview

The thesis is structured into several chapters. This section provides the structure of the thesis and a brief description of each chapter.

1. **Introduction** Introduces the topic of hyperspectral image compression, outlines the goals of the thesis, and summarizes the contributions of the authors.

2. **Literature** This chapter reviews existing literature on HSI compression, covering lossy and lossless methods, traditional and deep learning-based approaches, and relevant metrics for evaluating compression performance.

3. **Dataset** Description of the dataset used for training and testing the models, and a custom visualization tool developed to better understand the dataset.

4. **External specifications** Description of the tools and libraries used in the implementation of the models. Details instalation process and system requirements.

5. **Internal specifications** Description of the custom data pipeline developed for training and evaluation of the models. Details the architecture and functionality of the pipeline as well as its components and data structure.

6. **Machine learning model implementation** This chapter introduces the fundamental concepts of machine learning. It is focused on structures and the mathematics behind neural networks. It details activation functions and explains the use of optimizers.

7. **Lossless compression model: LineRWKV** In this chapter, the LineRWKV model for lossless compression is introduced. It details the model's structure, its implementation, and training process.

8. **Lossy compression model: Residual Convolutional Autoencoders** This chapter focuses on residual convolutional autoencoders for lossy compression. It presents the implementation and training results for two variants, a 3D RCAE and a hybrid 2D-1D RCAE.

9. **Lossy compression model: Reduced Complexity General Divisive Normalisation Autoencoder** This chapter presents the RCGDNAE, a lossy compression model for HSI, that enhances a standard autoencoder with fixed KLT preprocessing and GDN activation layers.

10. **Verification and validation** This chapter is focused on describing the methods and procedures used for verification and validation of the implemented models. It details the evaluation metrics used to assess the performance of the models.

11. **Conclusion** Chapter that summarizes the results of the presented compression models while acknowledging limitations and proposing future improvement.

1.3 Contribution of the authors

Table 1.1 summarizes the contributions of each author to different parts of the thesis. It highlights the division of work between the authors in terms of chapters and specific topics covered.

Table 1.1: Summary of the contribution of the authors.

Part of the thesis	Author
Chapter 1. Introduction	MR, JS
Chapter 2. Literature	MR, JS
Chapter 3. Dataset	MR, JS
Chapter 4. External specifications	MR, JS
Chapter 5. Internal specifications	MR, JS
Chapter 6. Machine learning model implementation	MR, JS
Chapter 7. Lossless compression model: LineRWKV	MR
Chapter 8. Lossy compression model: Residual Convolutional Autoencoders	JS
Chapter 9. Lossy compression model: Reduced Complexity General Divisive Normalisation Autoencoder	MR
Chapter 10. Verification and validation	MR, JS
Chapter 11. Comparison	MR, JS
Chapter 12. Conclusion	MR, JS
LineRWKV implementation	MR
Residual Convolutional Autoencoders implementation	JS
RCGDNAE implementation	MR
Data pipeline implementation	MR, JS
SmallSeg implementation	MR, JS

Chapter 2

Related literature

The purpose of this thesis is to create and validate an integrated deep learning pipeline for compression of HSIs, with particular focus on Earth observation satellites. Therefore, before developing the models, it is crucial to understand what are the particularities of working with HSIs, the challenges they present, and the existing methods for their compression. The literature review provides an overview of metrics used in image compression evaluation, general concepts of image compression, and a survey of existing HSI compression models. Reviewed literature includes scientific articles and conference papers, and is described in more detail in the following sections. These publications were selected for their relevance to the topic of this thesis and were screened to ensure only current research that has not been superseded was used. These papers served as the first step in the development of model implementations.

2.1 Processing challenges of hyperspectral images

Hyperspectral image acquisition enables the capture of detailed spectral information, but it introduces challenges related to data volume and transmission. A single HSI can be comprised of hundreds of spectral bands for each spatial pixel, leading to substantial file sizes. This poses significant challenges for storage, processing, and transmission, especially in resource-constrained environments such as satellites and remote sensing platforms. Furthermore, as the technology advances, the number of spectral bands and the spatial resolution of hyperspectral images continue to increase, exacerbating the problem. From the signal processing perspective, HSIs exhibit high correlation across both spatial and spectral dimensions [5]. This correlation can be exploited for compression, but it also means that traditional compression techniques may not be optimal.

2.2 Approaches to image compression

Image compression is a technique used to reduce the size of images for storage and transmission purposes. The main objective of image compression is to minimize the amount of data required to represent an image. The most important aspects of image compression are the compression ratio (CR) and the quality of the reconstructed image. The compression ratio is defined as the ratio of the original image size to the compressed image size. A higher CR means a greater reduction in size [14]. Compression can be broadly categorized into two types: lossy and lossless [14]. Lossless compression techniques allow for the exact reconstruction of the original image from the compressed data. These techniques are essential in applications where any loss of information is unacceptable [11]. However, lossless compression typically achieves lower compression ratios compared to lossy methods or takes more computational resources. Lossy compression techniques, on the other hand, allow for some loss of information in exchange for higher compression ratios [11]. These techniques are commonly used in applications where some loss of quality is acceptable, such as web images and videos. Lossy compression methods allow for higher compression ratios than lossless methods, but they may introduce artifacts and degrade image quality. The choice between lossy and lossless compression depends on the specific requirements of the application, including the acceptable level of quality loss and the desired compression ratio.

2.3 Hyperspectral image compression

Hyperspectral image compression has been an active area of research, with various approaches proposed over the years. These methods can be loosely categorized into traditional compression techniques and deep learning-based methods. Traditional compression techniques include, but are not limited to, methods such as statistical coding, wavelet transforms, and predictive coding [2]. These methods often rely on hand-crafted features and may not fully exploit the complex correlations present in hyperspectral data. In recent years, deep learning-based methods have gained prominence due to their ability to learn complex representations from data [14]. Autoencoders [7], convolutional neural networks (CNNs) [27], and generative adversarial networks (GANs) [6] have been employed for hyperspectral image compression. These methods can adaptively learn to compress hyperspectral images based on the data distribution, potentially leading to better performance compared to traditional methods. Several studies have demonstrated the effectiveness of deep learning-based compression methods for hyperspectral images [7, 27]. These methods are known to exploit the high spatial and spectral correlation present in HSIs, leading to improved compression ratios and reconstruction quality [14].

2.3.1 Algorithmic approaches

HSI compression algorithms can be distinguished based on the techniques they approach. One of the approaches is prediction-based, which relies on sequentially estimating pixel values based on previously encoded values. Due to its simplicity, this approach is often used in lossless compression algorithms [11]. The CCSDS 123.0-B-2 standard is an example of a predictive coding algorithm, which is the standard for lossless and near-lossless compression in space applications due to their low computing requirements [11]

A major recent breakthrough is a deep learning model called LineRWKV [23]. This model uses a predictive approach but replaces the fixed formula with a neural network. It became the first deep learning (DL) model proven to beat CCSDS 123.0-B-2 standard by achieving better compression while being efficient enough to run on a satellite [23]. Another approach is transformation-based compression. Instead of looking at pixels individually, this approach looks at the image as a whole and applies a mathematical transformation to repackage the entire image's information into a new, more compact form.

One of the most prominent examples of such transformation-based methods in deep learning are autoencoders. Autoencoders are neural networks trained to squeeze an image into a smaller "bottleneck" representation. This bottleneck is called latent space, and it contains the most important features of the original image in a compressed form. The process of squeezing is called encoding, and the reverse process of reconstructing the image to its original form is called decoding [16]. While autoencoders can obtain high compression ratios, due to the small size of the latent space representation, the information in it is heavily rounded off, which makes the process lossy.

2.4 Semantic segmentation of hyperspectral images

Semantic segmentation is a computer vision task that involves classifying each pixel in an image into predefined categories based on image content. In the context of HSIs, semantic segmentation is a particularly powerful tool due to the rich spectral information available for each pixel. This, in turn, allows for more accurate and detailed classification compared to traditional RGB images. HSI semantic segmentation has found applications in various fields, including agriculture [9], forestry [1], geology [24], surveillance [26], and environmental monitoring [4]. For example, in agriculture applications, semantic segmentation of HSIs is used to identify crop types, monitor nutrients, and detect diseases [4]. Thus, semantic segmentation is crucial for further utilization of HSIs in various applications. However, the high dimensionality and complexity of HSIs create a significant computational challenge for doing semantic segmentation in real-time or resource-constrained environments. Therefore, it is essential to ensure swift and efficient transmission and storage of HSIs, which can be achieved through different compression techniques. In order to evaluate the impact of compression on downstream tasks such as semantic segmentation, it is important to include a segmentation model in the evaluation pipeline. Due to the focus of this thesis being on compression techniques, a simple segmentation model is used for evaluation purposes.

Chapter 3

Dataset

3.1 Choosing the dataset

Among the many difficulties that need to be addressed when creating machine learning models, one of the most significant is sthe election of the dataset. The dataset must be relevant to the problem, of sufficient size, and of high quality. An unbalanced dataset or a dataset with too many items of one class can lead to biased models with poor generalization capabilities. A poorly preprocessed dataset with noise can lead to models that behave unpredictably in real-world scenarios.

The size of the dataset is another important factor. Large datasets are generally preferred as they provide the possibility to learn complex patterns, but it is also important to choose a dataset with diverse examples to cover a wide range of scenarios. Choosing a large dataset comes with increased computational requirements, longer training time, and storage needs. Selecting a dataset that is too small carries its own risks. A model trained on an insufficient dataset may not learn crucial patterns and quickly overfit to the training data, resulting in poor performance on unseen data. Taking these factors into account, the HySpecNet-11k dataset was chosen, which is described in Section 3.2. The dataset was accessed in November 2025, and it has been publicly available since its publication in 2023 [7]. This, however, changed in 2026, when the dataset was made unavailable due to licensing issues [8].

3.2 HySpecNet 11k dataset

For training, validation, and testing of models, the HySpecNet-11k dataset was used [7]. The dataset contains 11,483 nonoverlapping image patches. Each patch is 128×128 with 224 spectral bands. All of the images were captured in the real world by the Environmental Mapping and Analysis Program (EnMAP) satellite with a ground sampling distance of 30 meters. The number of bands is reduced from 224 to 202 by removing bands in the ranges [127-141] and [161-167] due to high water vapor absorption.

The dataset is split into two categories, easy and hard. The easy split was used for training and testing the models, as the primary task is data compression rather than image segmentation or classification. Each category is divided into training, validation, and testing subsets with a ratio of 70%, 20%, and 10 The dataset is distributed in the form of geotiff files, with each file containing several bands of a single image patch. To work with the dataset, first, the patches had to be reconstructed from the GeoTIFF files into 3D NumPy arrays of shape (128, 128, 202). To facilitate this, the code provided by the dataset authors was used [7].

3.3 Data visualization

To better understand the dataset and visualize the hyperspectral images, a custom visualization tool was developed. The tool allows selecting individual image patches from the dataset and displaying them in single-band and RGB formats. Figures 3.1 and 3.2 show examples of visualizations generated by the tool.

- Figure 3.1 present an RGB visualization of three different image patches obtained by connecting bands 43, 28 and 10 at wavelengths 634 nm, 550 nm and 463 nm.
- Figure 3.2 present a set of 4 images showing individual bands from a single image patch.

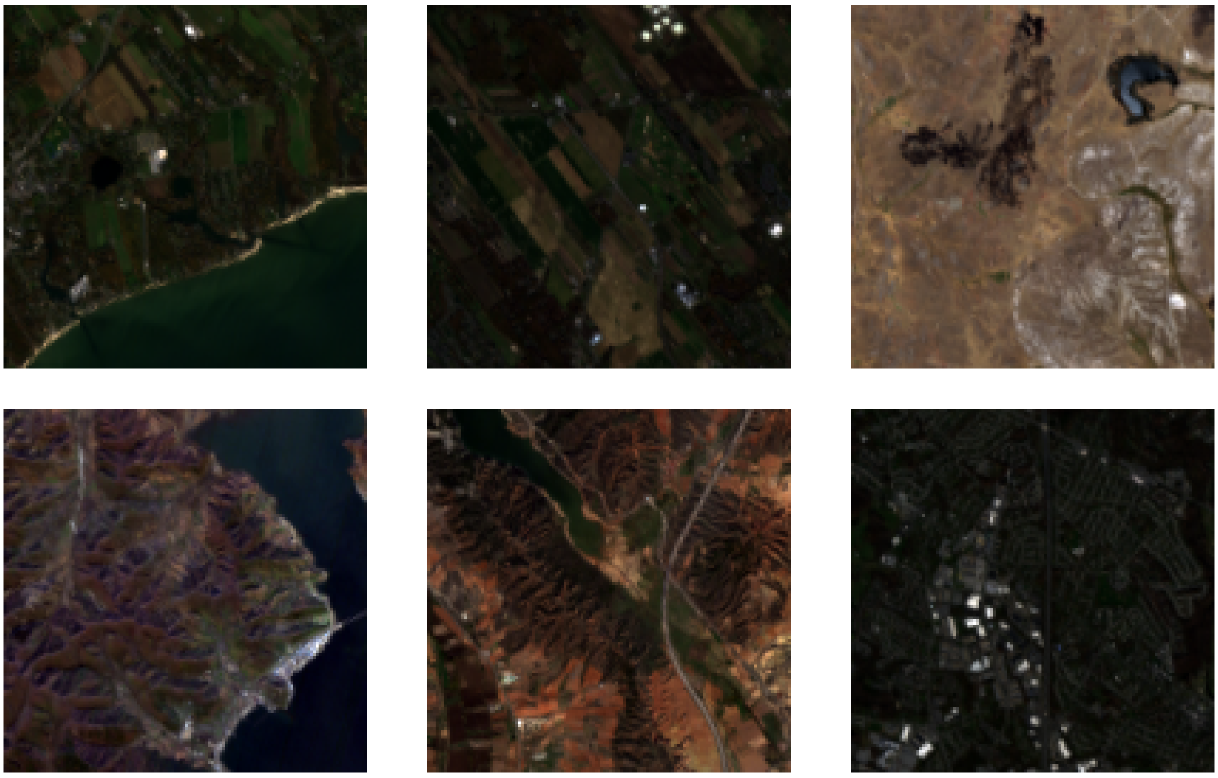
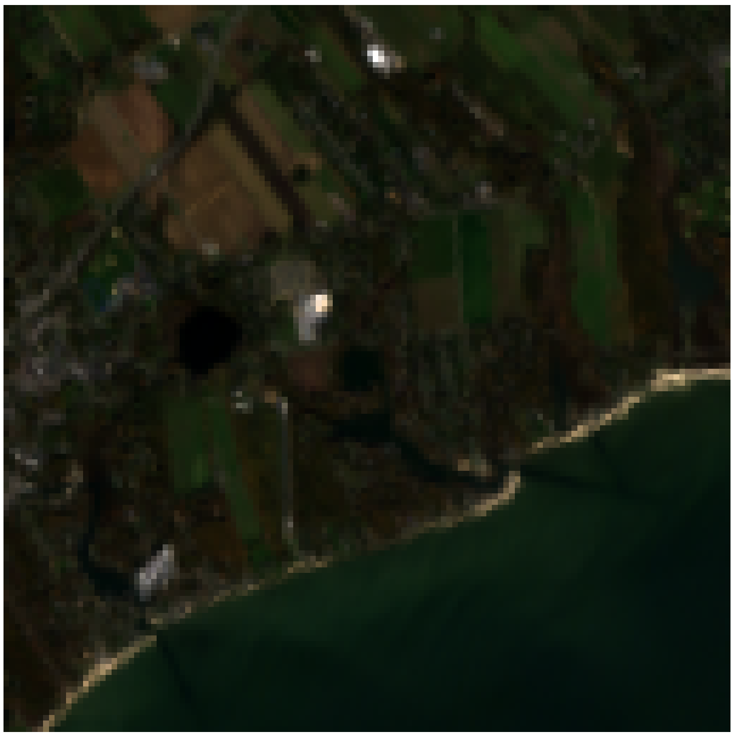
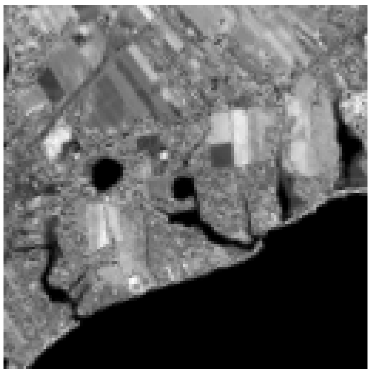


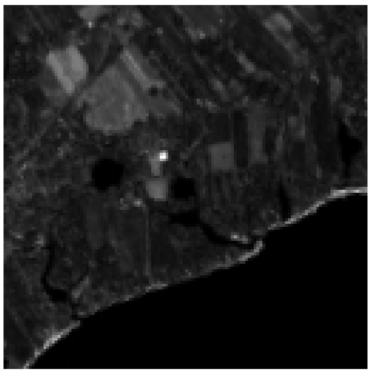
Figure 3.1: Example images from the dataset colored to RGB using bands 43, 28 and 10 corresponding to wavelengths 634 nm, 550 nm and 463 nm respectively.



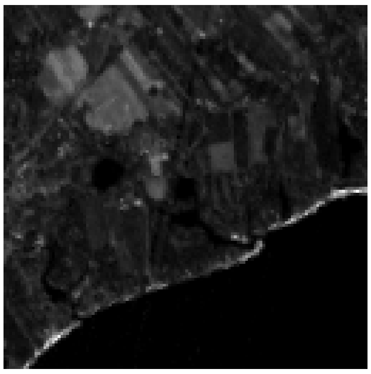
(a) RGB visualization using bands 43, 28 and 10 corresponding to wavelengths 634 nm, 550 nm and 463 nm respectively.



(b) Single band visualization of band 100



(c) Single band visualization of band 150



(d) Single band visualization of band 200

Figure 3.2: Example image patch from the dataset visualized in RGB using bands 43, 28 and 10 3.2a and three individual bands: band 100 3.2b, band 150 3.2c and band 200 3.2d.

Chapter 4

External specification

4.1 Software

4.1.1 Operating system

Project was developed and tested on Linux operating system. Installation instructions for the tools used in this thesis are provided for computers with Linux operating system.

4.1.2 Programming language

The code was implemented using Python programming language version 3.12. To manage project dependencies and environment, virtual environment was used. For installation of required libraries, built-in Package Installer for Python (pip) was used in version 25.2, that allows to install additional libraries from Python Package Index (PyPI).

4.1.3 IDE

Whole code was written using Visual Studio Code (VS Code) version 1.81.1. While VS Code in itself do not provide full IDE capabilities, it can be extended using various extensions. For this project, the following extensions were used:

- Python extension by Microsoft
- Pylez
- Jupyter

4.1.4 Drivers

To utilize GPU acceleration for model training, NVIDIA GPU drivers in version 580.95.05 were installed as well as CUDA Package in version 13.1.1-1. These are required to use tensorcores on NVIDIA GPUs for faster computation.

4.2 Libraries

4.2.1 Machine learning frameworks

As the main framework for building and training the models, TensorFlow in version 2.20.0 was used. It is a Python library for machine learning and artificial intelligence. It provides a comprehensive ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML, and developers easily build and deploy ML-powered applications. In this thesis, TensorFlow was used mainly for:

- Multidimensional arrays
- Distributed processing
- Model construction and training

Other libraries

- **Keras** (version 3.12.0) is a high-level API of TensorFlow that provides an easy-to-use interface for machine learning problems. It covers the workflow from data processing to model deployment.
  Main advantages of Keras are:
  - Simple interface
  - Clear error messages
  - Automated multiprocessing
  - Highly integrated GPU acceleration
- **NumPy** (version 2.3.5) is a fundamental mathematical and vector operation library for Python. It provides many mathematical functions and data structures useful for performing mathematical operations. In this thesis, it was mainly used due to the format of the HSIs and for efficient array manipulations.
- **Pandas** (version 2.3.3) is a data analysis and manipulation library. It was used for loading and processing CSV files containing dataset splits.
- **Matplotlib** (version 3.10.8) is a plotting library used for creating visualizations of results and training progress.
- **TensorBoard** (version 2.20.0) is a visualization toolkit for TensorFlow that provides tools for tracking metrics during training.
- **Pillow** (version 12.0.0) is an imaging library used for image processing operations.

- **scikit-learn** provides the PCA and IncrementalPCA implementations used for the Karhunen-Loève Transform (KLT).
- **pytz** is used for timezone handling when generating timestamps for experiment logs and model checkpoints.
- **os** provides bindings for the operating system. It is a built-in library in Python. It was used mainly for working with the filesystem.

4.3 Hardware

The training process was performed on NVIDIA graphics cards due to their CUDA and tensor core capabilities, due to this 2 NVIDIA GPUs were used. The first one was NVIDIA GeForce RTX 4080 with 16 GB of VRAM. The second one was NVIDIA GeForce RTX 2080 Ti with 11 GB of VRAM. Both of the GPUs support CUDA 13.1, and they are equipped with Tensor Cores, which significantly improve machine learning performance. Some parts of the network training and testing were performed on the CPU when GPU memory was insufficient. One of the CPUs used was an AMD Ryzen 7 7700X with 8 cores equipped with 32 GB of RAM. The other one was a dual-socket Intel Xeon E5-2667 v4 equipped with 512 GB of RAM.

4.4 Environment setup

4.4.1 Installation

The project source code can be found in the repository: `https://github.com/PiHalves/HSI-Compression`. To run the project, Python 3.12 is required. The first step is to create a virtual environment to isolate the project dependencies from the system ones. The next step is to install the required libraries. The required libraries are listed in the `requirements.txt` file in the main repository directory. Both of these steps can be done using the following command in the terminal provided in Figure 4.1. The complete list of

```
Python3.12 -m venv .env
source .env/bin/activate
pip install -r requirements.txt
```

Figure 4.1: Creating virtual environment and installing required libraries.

dependencies is specified in the `requirements.txt` file and includes all necessary packages for TensorFlow operations, data processing, and visualization. Key version requirements include:

Python	$\geq$ 3.12
TensorFlow	$\geq$ 2.20.0
Keras	$\geq$ 3.12.0
NumPy	$\geq$ 2.3.5
CUDA	$\geq$ 13.1 (for GPU acceleration)

4.4.2 Dataset

The dataset used in this project can be downloaded from the following link: `https://datadryad.org/dataset/doi:10.5061/dryad.fttdz08zh` The downloaded dataset will need to be unpacked. To do that, use this command in the terminal:

```
    tar -xzf hyspecnet-11k-*.tar.gz
```

Figure 4.2: Unpacking the dataset.

After unpacking, it is necessary to preprocess the data using the script provided in this repository: `https://git.tu-berlin.de/rsim/hyspecnet-tools`. To do that, download the repository, open the file `tif_to_npy.ipynb` and set variable `in_directory` to the path of the unpacked dataset in the second cell of the file. Run all the cells in the notebook to preprocess the data. After preprocessing, the folder structure should look like in Figure 4.3.

Best Models

In the course of the thesis, several models were trained. The best weights of these models are provided in the directory structure shown in Figure 4.4. This folder structure is placed in the main repository of the project.

```
hyspecnet-11k
├─ patches
│  ├─ tile_001
│  │  ├─ tile_001-patch_01
│  │  │  ├─ tile_001-patch_01-DATA.npy
│  │  │  ├─ tile_001-patch_01-QL_PIXELMASK.TIF
│  │  │  └─ ...
│  │  ├─ tile_001-patch_02
│  │  │  ├─ tile_001-patch_02-DATA.npy
│  │  │  ├─ tile_001-patch_02-QL_PIXELMASK.TIF
│  │  │  └─ ...
│  │  └─ ...
│  ├─ tile_002
│  │  ├─ tile_002-patch_01
│  │  │  ├─ tile_002-patch_01-DATA.npy
│  │  │  ├─ tile_002-patch_01-QL_PIXELMASK.TIF
│  │  │  └─ ...
│  │  └─ ...
│  └─ ...
└─ splits
   ├─ easy
   │  ├─ train.csv
   │  ├─ val.csv
   │  └─ test.csv
   └─ hard
      ├─ train.csv
      ├─ val.csv
      └─ test.csv
```

Figure 4.3: Ready to use dataset structure.

```
results
├─ lossy
│  ├─ case2d1d.keras
│  ├─ case3d.keras
│  └─ GNN.weights.h5
└─ lossless
   └─ lineRWKV.weights.h5
```

Figure 4.4: Directory tree structure.

Chapter 5

Internal specification

5.1 File structure

The project is organized into several directories and files, each serving a specific purpose. The repository is structured to clearly separate lossless and lossy compression pipelines, isolate non-learned signal processing components (such as KLT), and decouple training logic from model definitions. This separation allows independent development and testing of each component while maintaining clear interfaces between them. The structure is shown in Figure 5.1:

Description of the elements in the repository shown in Figure 5.1:

- Code/lossless: Contains the implementation of the LineRWKV model for lossless compression.
- Code/lossy: Contains the implementation of the lossy compression models (RCG-DNAE, RCAE3D, RCAE2D1D).
- Code/segmentation: Contains the implementation of segmentation models for hyperspectral data classification (small_segmenter).
- Code/TFDataloader: Contains the data loader implementation for loading and pre-processing the hyperspectral image dataset.
- Code/metrics: Contains the implementation of various evaluation metrics used to assess model performance.
- Code/nonML: Contains the KLT implementation needed for the RCGDNAE model.
- Code/main.py: The main script to run training and evaluation of the models.
- Latex: Contains all the files related to the thesis document.
- results: Contains the best model weights and results from training.

```
Repository
├─ Code
│  ├─ lossless
│  ├─ lossy
│  ├─ segmentation
│  ├─ TFDataloader
│  ├─ metrics
│  ├─ nonML
│  └─ main.py
├─ Latex
│  └─ Thesis files
├─ results
└─ requirements.txt
```

Figure 5.1: Main project files and directories.

- requirements.txt: Lists all the Python dependencies required to run the project.

5.2 Model overview

Table 5.1 summarizes the key characteristics of each implemented model.

Table 5.1: Comparison of implemented compression models.

Model	**Type**	**Convolution**	**Key Feature**
LineRWKV	Lossless	—	Autoregressive
RCGDNAE	Lossy	2D + KLT	Rate–distortion
RCAE (3D)	Lossy	3D	Joint spatial–spectral
CAE2D1D	Lossy	2D + 1D	Separated processing
Small Seg	Segmentation	3D + 2D	Spectral pooling

5.3 Classes and functions

5.3.1 LineRWKV model

The lineRWKV model is implemented in the file `Code/lossless/lineRWKV.py` with a trainer class defined in `Code/lossless/lineRWKV_trainer.py`. There are three classes in the `lineRWKV.py` file: LineRWKVBlock, LineRWKVPredictor, and LineRWKVLosslessCodec. All of these classes describe the basic building blocks and functionality of the LineRWKV model.

LineRWKVBlock

The LineRWKVBlock class implements a single block of the LineRWKV model, inheriting from `tf.keras.layers.Layer`. This block encapsulates the core RWKV attention mechanism adapted for line-based processing of hyperspectral data, managing the recurrent state updates and attention computations within a single processing unit.

LineRWKVPredictor

The LineRWKVPredictor class encapsulates the autoregressive prediction mechanism required for lossless coding of hyperspectral images, inheriting from `tf.keras.Model`. The predictor processes data line-by-line to maintain causality during encoding and decoding—each line prediction depends only on previously processed lines. Preprocessing is integrated within the model to ensure consistent data transformation across training and inference. The `predict_line_by_line` method provides explicit causal prediction, while `compute_residuals` separates residual computation for downstream entropy coding. This design allows the same model to be used for both training (batch processing) and inference (sequential line processing).

LineRWKVLosslessCodec

The LineRWKVLosslessCodec class provides a complete encode/decode interface for lossless compression, wrapping the LineRWKVPredictor. This separation of concerns allows the predictor to focus on prediction quality while the codec handles the compression workflow—computing residuals during encoding and reconstructing from residuals during decoding. The codec ensures that the encode and decode operations are exact inverses, guaranteeing lossless reconstruction.

LineRWKVTrainer

The LineRWKVTrainer class handles training of the LineRWKV model, decoupling training logic from the model definition. This separation allows the same model architecture to be trained with different loss functions, optimizers, or training strategies without modifying the model code.

The trainer implements a custom prediction residual loss function that directly optimizes for compression performance by minimizing the entropy of prediction residuals. This is more appropriate for lossless compression than standard reconstruction losses, as it targets the actual quantity that determines compressed file size.

The training loop follows a standard pattern: per-step gradient updates, per-epoch metric aggregation, and checkpoint saving. Metrics (PSNR, SSIM, Spectral Angle) are computed during validation to monitor reconstruction quality without affecting the training objective.

5.3.2 RCGDNAE model

The RCGDNAE model is implemented in the file `Code/lossy/RCGDNAE.py` with a trainer class defined in `Code/lossy/RCGDNAE_trainer.py`. One additional file `Code/nonML/KLT.py` contains the implementation of the Karhunen–Loève Transform (KLT) needed for the RCGDNAE model.

KLT

The Karhunen–Loève Transform (KLT) module in `KLT.py` provides spectral decorrelation as a preprocessing step for the RCGDNAE model. KLT is the optimal linear transform for decorrelating multivariate data, making it particularly effective for hyperspectral images where adjacent spectral bands are highly correlated.

The module addresses memory constraints through incremental training: rather than loading the entire dataset to compute the covariance matrix, `train_klt_incremental` uses scikit-learn's IncrementalPCA to process data in batches. This is essential when working with large hyperspectral datasets that exceed available RAM.

All transform operations (`apply_klt_tf`, `inverse_klt`) are implemented using TensorFlow operations to ensure compatibility with the neural network pipeline and enable GPU acceleration. The `cluster_klt_bands` function groups transformed bands based on their variance contribution, allowing the RCGDNAE model to allocate more capacity to bands carrying more information.

FullCompressor class

The RCGDNAE model is implemented as the FullCompressor class, inheriting from `tf.keras.Model`. The encoder and decoder networks are constructed separately via dedicated builder methods, allowing architectural modifications without affecting the overall compression pipeline. This modular design facilitates experimentation with different encoder/decoder configurations while maintaining a consistent interface for training and inference.

GDN

The Generalized Divisive Normalization (GDN) layer implements a learned normalization operation specifically designed for image compression. Unlike batch normalization, GDN learns channel-wise divisive interactions that help decorrelate features and improve rate–distortion performance. The layer supports both forward (GDN) and inverse (IGDN) modes, used in the encoder and decoder, respectively, controlled by an inverse flag during initialization.

MaskedConv2D

The MaskedConv2D layer implements autoregressive convolutions by applying a learned mask to the convolutional kernel. This ensures that predictions for each spatial location depend only on previously processed locations, maintaining the causal structure required for entropy modeling. The mask is constructed during the build phase based on the kernel size and remains fixed during training.

Quantization function

Quantization is the primary source of information loss in the lossy compression pipeline. The `quantize` function implements differentiable quantization using the straight-through estimator approach.

During training, uniform noise is added instead of rounding, which provides a differentiable approximation that allows gradients to flow through the quantization operation. During inference, standard rounding is applied. This technique, common in learned image compression, enables end-to-end training while still producing integer-valued latent codes at test time.

Gaussian likelihood function

The `gaussian_likelihood` function estimates the bitrate of quantized latent codes by modeling their distribution as Gaussian. This likelihood estimate forms the rate term in the rate–distortion loss: lower likelihood (higher surprise) indicates more bits required to encode a value. By training the encoder to produce latent codes with high likelihood under the learned prior, the model learns to generate easily compressible representations.

5.3.3 NonNegativeMinimum constraint

The NonNegativeMinimum constraint ensures that certain network weights remain positive and above a specified minimum threshold. This is particularly important for the GDN layer, where the normalization parameters must be strictly positive to avoid division by zero and ensure numerical stability.

The constraint is applied during training after each gradient update, clipping any values that fall below the minimum threshold. This approach is more stable than using activation functions like ReLU on the weights, as it preserves gradient flow while enforcing the constraint.

RCGDNAETrainer class

The RCGDNAETrainer class handles training of the RCGDNAE model, following the same training loop structure as LineRWKVTrainer but with a fundamentally different optimization objective. While LineRWKVTrainer minimizes prediction residual entropy for lossless compression, RCGDNAETrainer optimizes a rate–distortion trade-off that balances reconstruction quality against bitrate. The `compute_rate_distortion_loss` method implements this objective by combining a distortion term (reconstruction error) with a rate term (estimated bitrate from entropy modeling). This allows controlling the compression ratio by adjusting the rate–distortion trade-off parameter.

5.3.4 RCAE model (3D Residual Convolutional Autoencoder)

The RCAE model is implemented in `Code/lossy/rcae3d.py` as the ResidualConv3DAutoencoder class. This model uses 3D convolutions to jointly process spatial and spectral information, treating hyperspectral images as volumetric data.

ResidualConv3DAutoencoder class

The ResidualConv3DAutoencoder class inherits from `tf.keras.Model` and implements an encoder–decoder architecture with residual connections. The design choice to use 3D convolutions allows the network to learn joint spatial–spectral features, capturing correlations that exist across both dimensions simultaneously. Residual blocks with batch normalization and leaky ReLU activation help with gradient flow during training of deep networks.

The encoder applies strided 3D convolutions for spatial downsampling while preserving spectral resolution, reducing the data to a compact latent representation. The decoder mirrors this structure using transposed convolutions for upsampling. Separate `encode` and `decode` methods allow independent access to encoding and decoding operations, useful for analyzing the latent space or performing partial reconstruction.

5.3.5 2D1D RCAE model

The 2D1D RCAE model is implemented in `Code/lossy/rcae2D1D.py` as the RCAE2D1D class. This model explicitly separates spatial and spectral processing, using 2D convolutions for spatial features and 1D convolutions for spectral features.

RCAE2D1D class

The RCAE2D1D class inherits from `tf.keras.Model` and implements a two-stage compression approach. Unlike the 3D autoencoder that jointly processes spatial and spectral dimensions, this architecture processes them sequentially—first extracting spatial features with 2D convolutions, then compressing spectral information with 1D convolutions.

This separation offers two advantages: reduced computational cost compared to full 3D convolutions, and the ability to apply dimension-specific processing strategies. The encoder first applies spatial residual blocks to reduce spatial redundancy, reshapes the output to a sequence format, and then applies spectral residual blocks. The decoder reverses this process.

SpatialReshape class

The SpatialReshape layer handles the transition between 2D spatial and 1D spectral processing stages. It reshapes tensors while tracking the original spatial dimensions, ensuring correct reconstruction during decoding.

5.3.6 Small Segmenter model

The Small Segmenter model is implemented in `Code/segmentation/small_seg.py` as the small_segmenter class. This model provides a memory-efficient architecture for per-pixel classification of hyperspectral data, designed to run on GPUs with limited VRAM (approximately 1–2 GB for batch size of 1).

small_segmenter class

The small_segmenter class inherits from `tf.keras.Model` and implements a 3D-to-2D segmentation architecture. The model takes 5D input tensors of shape $(B, H, W, D, 1)$ where D is the spectral dimension (202 bands for HySpecNet), and outputs 2D spatial segmentation masks of shape (B, H, W, C) where C is the number of classes.

The architecture follows a progressive spectral reduction strategy:

1. **Initial feature extraction**: A 3D convolution with kernel size $(3, 3, 7)$ extracts low-level spatial–spectral features with a larger receptive field in the spectral dimension.

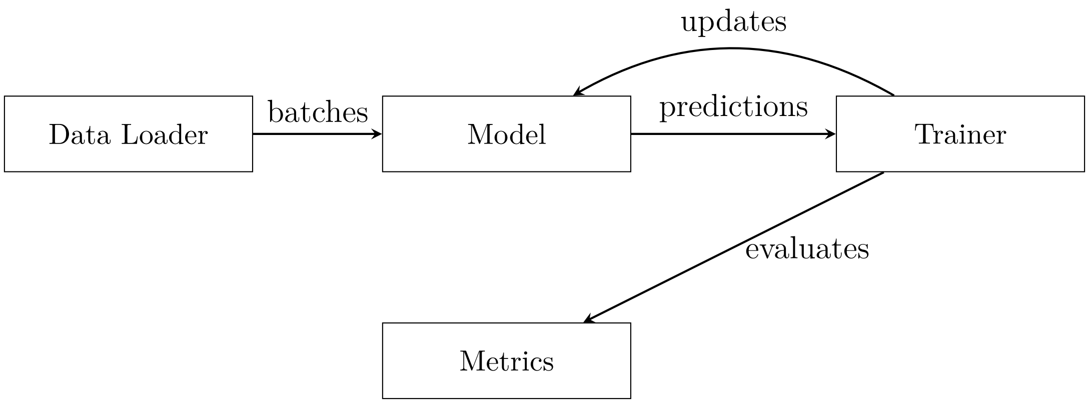


Figure 5.2: Component interaction during training.

2. **Encoder blocks**: Multiple 3D convolutional blocks with spectral-only max pooling progressively reduce the spectral dimension while preserving full spatial resolution.
3. **Spectral collapse**: A 1×1×1 convolution followed by mean reduction over the spectral axis collapses the spectral dimension.
4. **2D refinement**: Two 2D convolutional layers refine the spatial features for classification.
5. **Classification**: A final 1×1 convolution with softmax activation produces per-pixel class probabilities.

The model uses batch normalization after each convolution and dropout before the final classification layer for regularization. Configurable hyperparameters include `base_filters` (initial filter count), `depth` (number of encoder blocks), and `dropout_rate`.

Memory efficiency considerations

The small_segmenter is specifically designed for memory-constrained environments. By using spectral-only pooling (pool size $(1, 1, 2)$), the model reduces spectral dimensionality without losing spatial resolution, which is critical for dense prediction tasks. The `estimate_memory_usage` utility function provides rough estimates of GPU memory requirements for different batch sizes and configurations.

5.3.7 Component interactions

Figure 5.2 illustrates how the main components interact during training and inference.

The data loader provides batched hyperspectral images to the model. The trainer orchestrates the training loop: it calls the model's forward pass, computes the loss, performs backpropagation, and updates model weights. Metrics are computed periodically (typically each epoch) on the validation set to monitor training progress without affecting the optimization.

This separation ensures that each component has a single responsibility:

- **Loader**: I/O and batching
- **Model**: Forward computation
- **Trainer**: Optimization loop and checkpointing
- **Metrics**: Evaluation

5.3.8 Metrics

All metrics are implemented in `Code/metrics/metrics.py`.

CompressionMetrics class

The CompressionMetrics class provides static methods for evaluating hyperspectral image compression quality. The metrics fall into two categories:

- **Spatial quality metrics**: MSE, RMSE, PSNR, SNR, and SSIM measure reconstruction fidelity in the spatial domain.
- **Spectral quality metrics**: Spectral Angle Mapper (SAM) measures preservation of spectral signatures, which is critical for hyperspectral applications where spectral fidelity matters more than spatial appearance.

Additional methods compute compression-specific metrics: compression ratio (CR), bits per pixel (bpp), and bits per pixel per band (bpppb).

Metrics and loss layer classes

In addition to the static methods, the `Code/metrics/metrics.py` file also contains several classes that implement the metrics as Keras layers. These were created to speed up the computation of metrics during model training and evaluation. The classes are: `MeanSquaredError`, `PeakSignalToNoiseRatio`, `StructualSimilarity` and `SpectralAngle`. Each of these classes inherits from the `tf.keras.layers.Layer` class and implements the respective metric in the `call` method.

5.3.9 Data Loader

The data loader is implemented in `Code/TFDataloader/TFdataloader.py` as the TFHySpecNetLoader class. This class utilizes TensorFlow's `tf.data` pipeline to efficiently load and preprocess the HySpecNet-11k dataset.

The loader assumes hyperspectral data stored as NumPy files (`.npy`) with band-major ordering, following the preprocessed format described in the external specification. Dataset splits are defined by CSV files containing patch identifiers for training, validation, and test sets.

To avoid I/O bottlenecks during training, the loader uses parallel mapping (`tf.data.AUTOTUNE`) to load multiple files concurrently and prefetching to overlap data loading with model computation. This design ensures that GPU utilization remains high even when loading large hyperspectral images from disk.

5.3.10 Configuration and extensibility

Model hyperparameters are stored in JSON configuration files (e.g., `config/lineRWKV_config.json`) rather than being hardcoded. This separation of configuration from code enables:

- Running experiments with different hyperparameters without code changes
- Reproducing experiments by storing configuration alongside model weights
- Version control of experimental settings

To add a new compression model, the following components need to be implemented:

1. A model class inheriting from `tf.keras.Model` with `encode` and `decode` methods
2. A trainer class following the pattern established by LineRWKVTrainer or RCG-DNAETrainer
3. A configuration file specifying model hyperparameters

The existing metrics and data loader can be reused without modification, as they are model-agnostic.

Chapter 6

Machine learning model implementation

6.1 Machine learning basics

6.1.1 Neural networks

Artificial neural networks (ANNs) are function approximators inspired by how people believed the human brain works. They consist of layers of neurons that are interconnected via weighted connections. Each neuron receives multiple inputs, applies a bias term and an activation function, which produces an output. The simplified mathematical representation of a single neuron is given by equation 6.1 :

$$y = \phi \left(\sum_{i=1}^{n} w_i x_i + b \right) , \tag{6.1}$$

where y is the output of the neuron, x_i are the inputs, w_i are the weights associated with each input, b is the bias term and ϕ is the activation function. ϕ can be any non-linear differentiable function, as linear functions would simplify the model to only having one layer, while it needs to be differentiable due to the use of gradient-based optimization methods discussed in Section 6.1.2.

Typical activation functions include Rectified Linear Unit (ReLU), Sigmoid or Tanh. In these, the models mostly used ReLU, Gaussian Error Linear Unit (GeLU), Leaky ReLU, and Sigmoid. The ReLU, GeLU, and Leaky ReLU are similar functions that simply introduce non-linearity. The most basic of these is the ReLU, which outputs zero for any negative input and the input itself for any positive input. Leaky ReLU introduces a small slope for negative inputs instead of outputting zero. GeLU is a smoother version of ReLU that approximates the output using the Gaussian error function, as shown in equation

6.2:

$$\mathrm{GeLU}(x) = \frac{1}{2}x\left(1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right), \tag{6.2}$$

where erf is the error function shown in equation 6.3,

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt. \tag{6.3}$$

The Sigmoid function, on the other hand, maps input values to a range between 0 and 1 which is useful for probabilistic outputs or other problems where a certain range of outputs is appropriate. The Sigmoid function is given by the formula in equation 6.4:

$$\mathrm{Sigmoid}(x) = \frac{1}{1 + e^{-x}}. \tag{6.4}$$

6.1.2 Training neural networks

Neural networks are typically trained using a process called backpropagation. This involves calculating the gradient of a loss function with respect to the network's weights and biases using the chain rule of calculus. These gradients can then be used to update the weights and biases of the neurons using optimization algorithms. One of the most commonly used optimization algorithms is stochastic gradient descent (SGD), especially in the form of the Adam optimizer [12], which was used in this thesis. Adam combines the advantages of two other extensions of SGD: Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp). It computes adaptive learning rates for each parameter based on estimates of the first and second moments of the gradients. However, to use Adam or any other gradient-based optimization method a suitable loss function must be used to quantify the difference between the predicted outputs of the network and the true target values. Common loss functions for regression problems include Mean Squared Error (MSE) and Mean Absolute Error (MAE) [22]. In image compression tasks, MSE is often used as it directly measures the pixel-wise differences between the original and reconstructed images. This doesn't always correlate well with human perception of image quality, so other metrics like those discussed in Section 11.1 may also be considered during model evaluation.

Chapter 7

Lossless compression model: LineRWKV

7.1 Introduction

The LineRWKV model is a neural network designed for lossless compression of HSIs. It is based on the RWKV (Receptance Weighted Key Value) architecture, which combines the advantages of recurrent neural networks (RNNs) while incorporating the parallelizability of transformers. The LineRWKV model is specifically adapted to process HSIs in a line-by-line manner, making it suitable for exploiting the spatial and spectral correlations present in hyperspectral data. This model is considered lossless because it is designed to predict each pixel or line of the image based on the context provided by the preceding pixels or lines, while having an explicit recurrence that allows the decoder to exactly reproduce the predictions made by the encoder. This allows the model to achieve near lossless performance, which, when combined with the entropy coding of the prediction residuals, results in effective lossless compression.

7.2 Related works

The RWKV architecture was introduced by Peng et al. [18] as a novel neural network architecture that combines the strengths of RNNs and transformers. RWKV achieves linear complexity $O(n)$ compared to the quadratic complexity $O(n^2)$ of attention-based models, making it more efficient for long sequences. The RWKV was then adapted for HSI lossless compression by Valsesia et al. [23] who created the original LineRWKV model. The implementation presented in this thesis is based on their work but modified to better suit the specific dataset and requirements of this project.

7.3 Model architecture

The model takes as input a hyperspectral image of shape $[batch, 128, 128, 202]$ where 202 is the number of spectral bands, and 128x128 is the spatial dimension of the image patch. The input is first projected to a working dimension (default $dim = 128$) using a dense layer. This projected input is then processed through a stack of LineRWKVBlock layers (default: 4 blocks). Finally, the output is projected back to the original number of channels using another dense layer to produce the predicted image. The models architecture is present in Figure 7.1.

7.3.1 Overall structure

The LineRWKV model processes the input image in a sequential manner, line by line along the height dimension. For each line, the model processes the pixels from left to right, maintaining a state that captures the context from the previous pixels in the line. The context not only includes information that the model has inferred but also the residuals, allowing the model to make accurate predictions. Processing images line-by-line ensures that the model maintains causality, allowing the decoder to exactly reproduce the predictions made by the encoder. This explicit recurrence is crucial for achieving lossless compression. The main downside of this approach is its sequential processing along the height dimension, which limits parallelization across lines. This may be seen as a limitation in synthetic scenarios, such as processing images on GPUs, where parallelization is key for performance. However, in practical applications such as satellite data transmission, where images are often gathered line by line, it is a reasonable trade-off to make.

7.3.2 RWKV block

The core component of the LineRWKV model is the LineRWKVBlock, which implements the RWKV mechanism adapted for line-by-line processing. The block consists of three main projections: Value, Receptance, and Channel Mix. The Value projection determines what information to store in the state, while the Receptance projection acts as a gate controlling how much of the state influences the output. The RWKV recurrence is implemented as follows:

$$s_t = \text{decay} \cdot s_{t-1} + (1 - \text{decay}) \cdot v_t, \tag{7.1}$$

$$y_t = x_t + r_t \cdot s_t \tag{7.2}$$

where s_t is the accumulated state at position t, v_t is the value at position t, r_t is the receptance (gate) at position t, and decay is a fixed parameter per block. The recurrence is implemented using TensorFlow's `tf.scan` function to ensure explicit left-to-right processing along the width dimension of the image line.

7.3.3 Predictor model

The LineRWKVPredictor class encapsulates the overall model architecture. It handles the line-by-line processing of the input image using a `tf.while_loop` to iterate over the height dimension. Within each iteration, the current line is processed through all RWKV blocks, maintaining the state across lines to capture temporal and spatial context.

7.4 Model implementation

One of the greatest differences between this implementation and the original LineRWKV implementation by Valsesia et al. [23] is the lack of using external programming languages such as CUDA for the RWKV block implementation. Instead, this implementation relies solely on TensorFlow operations to ensure compatibility and ease of use. This makes the implementation more accessible and easier to modify or extend. The LineRWKV model is implemented in the file `lineRWKV.py` which contains the main classes: `LineRWKVBlock`, `LineRWKVPredictor`, and `LineRWKVLosslessCodec`. The training logic is encapsulated in the `LineRWKVTrainer` class located in the file `Code/lossless/lineRWKV_trainer.py`.

7.5 Training

The LineRWKV model was trained using the Adam optimizer with a learning rate of 0.0001. Mean Squared Error (MSE) was used as the loss function to quantify the difference between the original and predicted images. The training process involved monitoring the loss on both the training and validation datasets to ensure proper convergence and to prevent overfitting. The training curves for both training and validation metrics are shown in Figure 7.2.

The training was conducted over 44 epochs, with the model showing consistent improvement in all monitored metrics. Such a specific number of epochs was chosen due to time constraints rather than convergence criteria. The training was stopped early because of limited computational resources and time constraints. Due to such early termination of training, none of the metrics started to plateau, indicating that the model could benefit from further training to achieve even better performance.

7.6 Summary

The LineRWKV model presents a paper-faithful implementation of the RWKV architecture tailored for hyperspectral image lossless compression. The model effectively captures the spatial and spectral correlations present in HSI data while maintaining the causality necessary for lossless compression. It processes images line-by-line, allowing it to exploit the inherent structure of hyperspectral images.

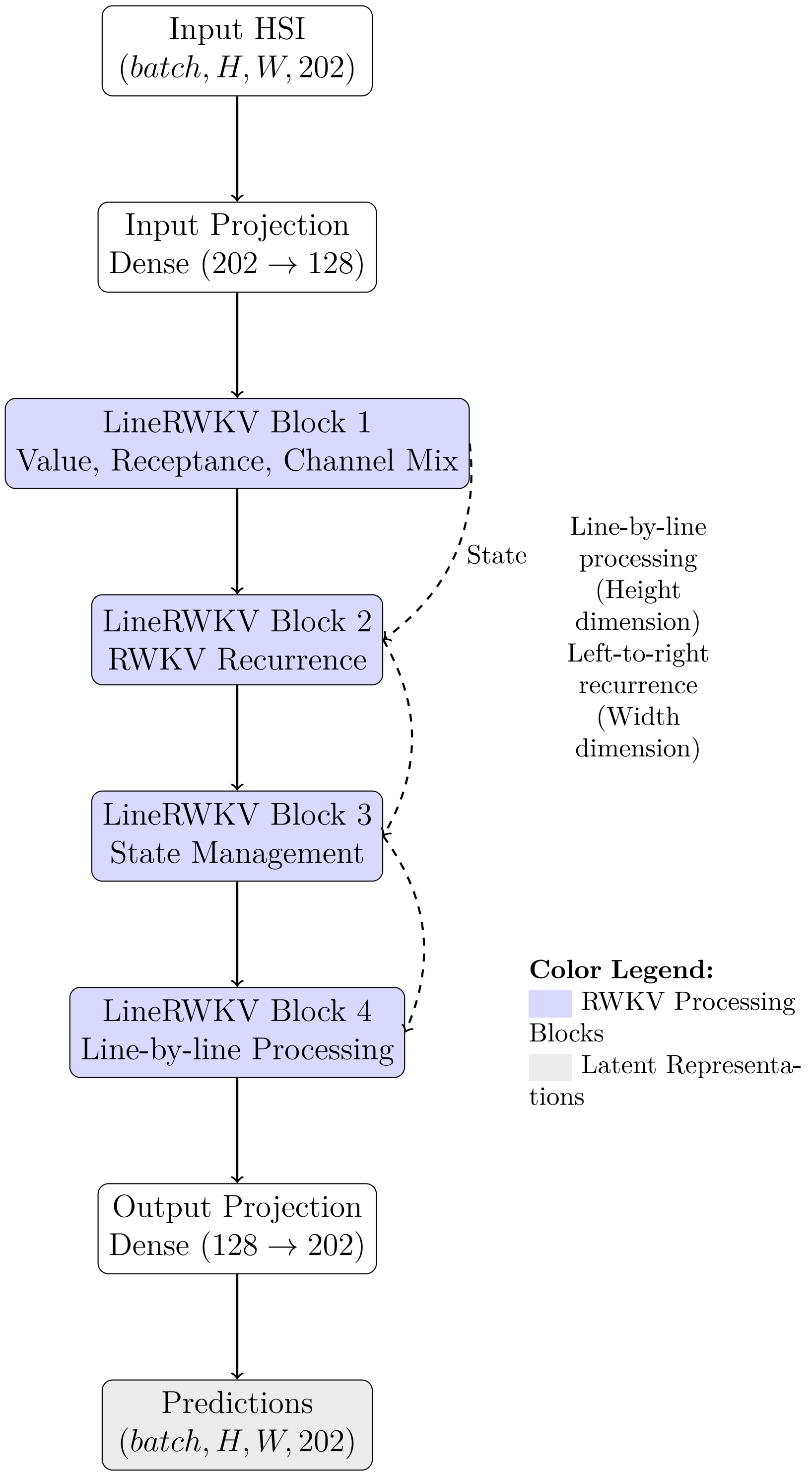
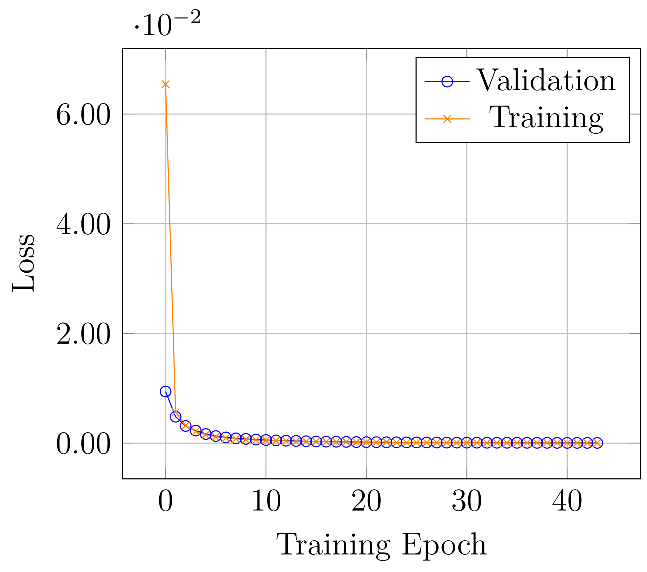
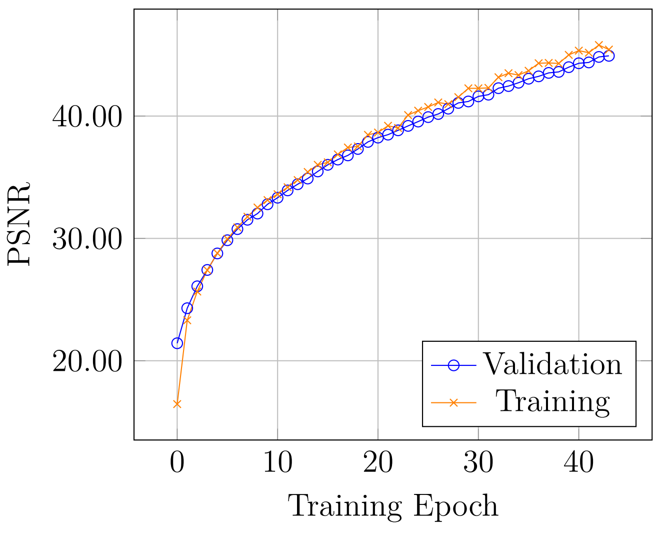


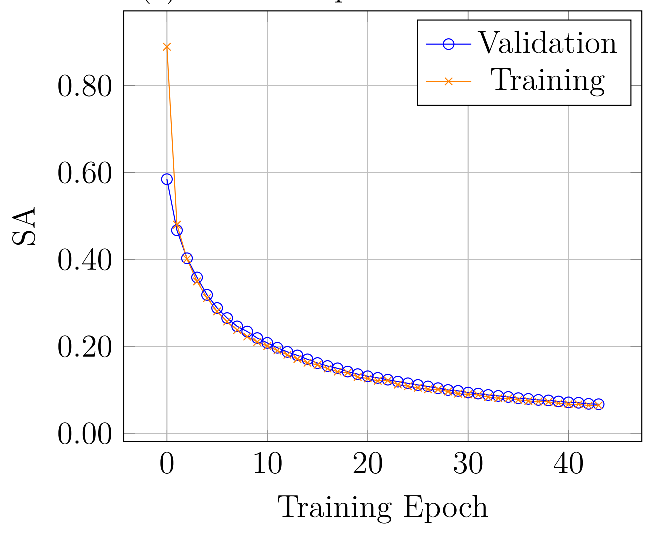
Figure 7.1: LineRWKV architecture for lossless HSI compression. The model processes images line-by-line with explicit recurrence maintaining state across RWKV blocks.



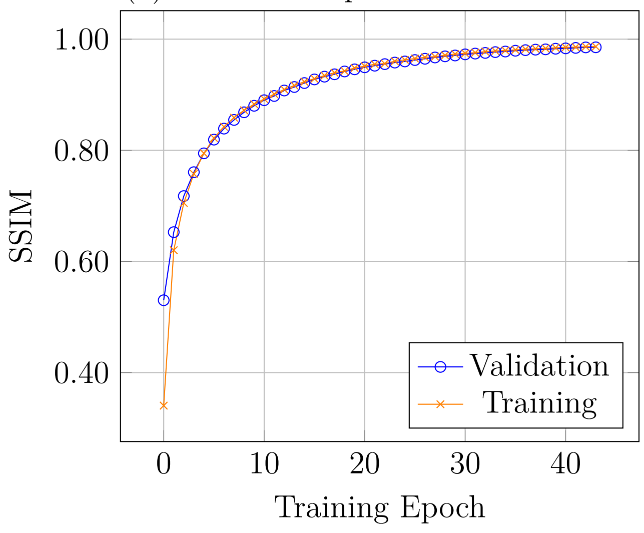
(a) Loss over Epochs



(b) PSNR over Epochs



(c) SA over Epochs



(d) SSIM over Epochs

Figure 7.2: Training and Validation Metrics over Epochs for LineRWKV model

Chapter 8

Lossy compression model: Residual Convolutional Autoencoders

8.1 Introduction

An autoencoder is a type of neural network used for unsupervised learning, with the primary application in data compression and representation [13, 16]. Its structure mainly consists of two main parts: an encoder and a decoder. The encoder compresses the high-dimensional input data into a lower latent representation called the bottleneck. The decoder, on the other hand, reconstructs compressed data back to its original form. The network is trained to minimize the difference between the input data and its reconstructed output, typically using a loss function such as MSE. For the purpose of this thesis, two custom autoencoder architectures were implemented that were loosely inspired by existing solutions

8.2 Related works

For image data, the standard autoencoder architecture is inefficient due to the high dimensionality and spatial structure of images. Convolutional Autoencoders (CAEs) overcome these limitations by replacing fully connected layers with convolutional layers in both the encoder and decoder. In an encoder, convolutional layers progressively reduce the spatial dimensions of the image while increasing the depth. The decoder performs the reverse operation using transposed convolutional layers or upsampling layers to reconstruct the image to its original size. This architecture is very effective in capturing spatial patterns and features in images, including HSI [16].

8.3 Model architecture

The Residual Convolutional Autoencoder (RCAE) enhances the standard CAE architecture by incorporating residual blocks, popularized by ResNet models [10]. The residual blocks incorporate skip connections that allow for bypassing one or more layers. These blocks learn the residual mapping, which is the difference between the input and the output, rather than a full transformation. This helps to mitigate the vanishing gradient problem and allows for training deeper networks more effectively. The designed architectures are presented in Figure 8.1 for 3D CNN, and in Figure 8.4 for the hybrid 2D-1D CNN.

8.4 Residual blocks

The core functionality of residual blocks is the shortcut connection. In a traditional network without residual connections, it is designed to learn a direct mapping from input x to output $H(x)$. The residual block, instead of learning $H(x)$ directly, learns the residual function $F(x) = H(x) - x$. The original input x is then added to the output of these layers, making the final output $H(x) = F(x) + x$. This approach creates a clear pathway for information and gradients to flow through the network.

8.5 Residual Convolutional Autoencoder architecture

Two main different RCAE models were implemented in this thesis: a RCAE3D with a residual block and a hybrid 2D-1D RCAE also with residual blocks.

8.5.1 RCAE3D

The architecture that was implemented in this thesis is an unsymmetrical RCAE. It follows the general structure of an autoencoder with an encoder and a decoder, with the addition of residual blocks between convolutional layers. The network uses 3D convolutional layers to capture both spatial and spectral features of HSI data. While the overall structure is similar to traditional CAEs, some modifications were made that allow it to distinguish itself from standard implementations. The first of them is the use of 3D convolutional kernels with a depth of 1, which makes the model process each spectral band separately, treating spectral dimensions as separate channels. This approach reduces computational complexity. The second modification is the use of sigmoid activation in the latent space to constrain the bottleneck values between 0 and 1. The last modification is the use of double upsampling in the decoder part to better reconstruct the image as well as to reduce checkerboard artifacts.

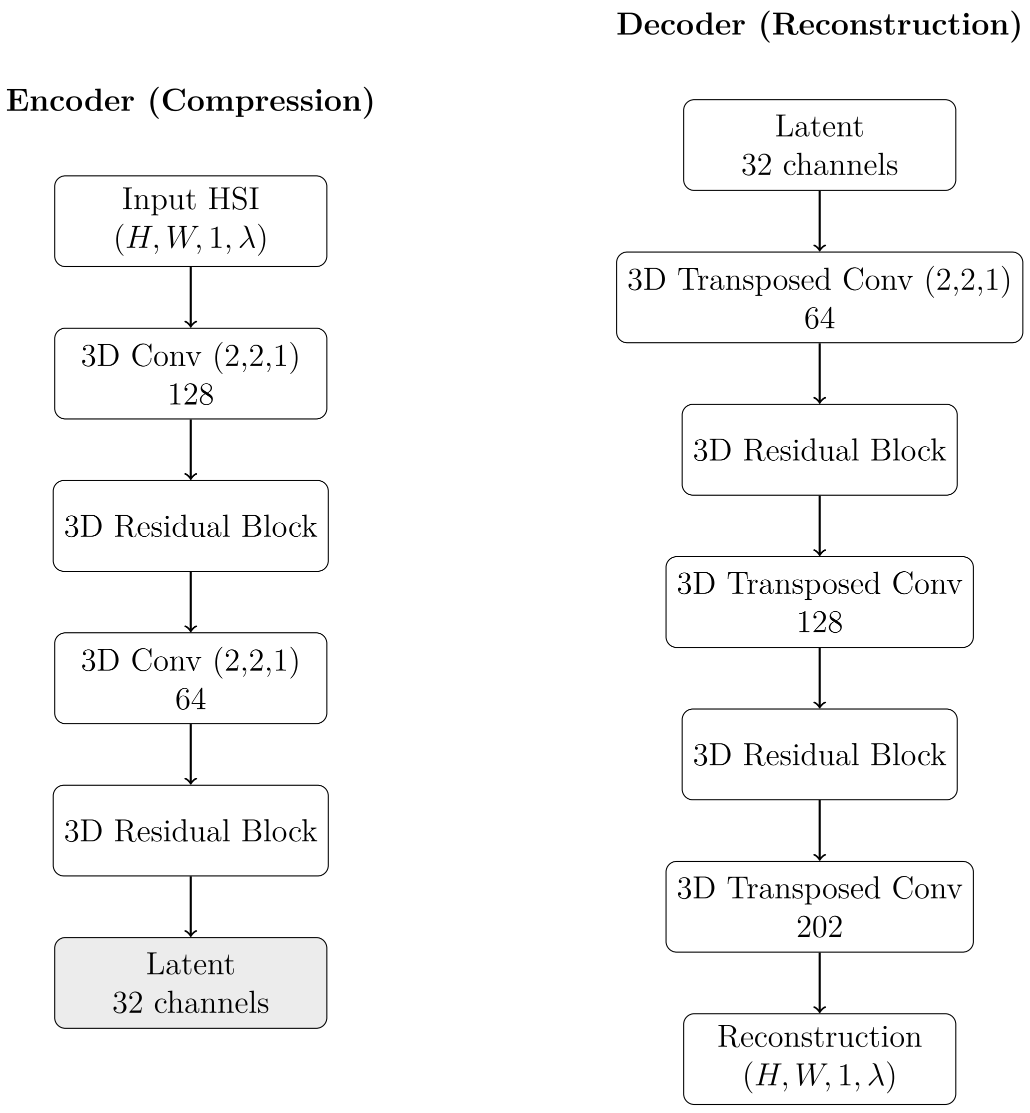


Figure 8.1: RCAE3D architecture. Left: encoder. Right: decoder. λ is the number of spectral bands.

8.5.2 RCAE2D1D

The hybrid RCAE2D1D model combines 2D and 1D convolutional layers to capture both spatial and spectral features of HSI data. The encoder processes spatial patterns using 2D convolutional layers with residual blocks while reducing the spatial dimensions. It then reshapes the data to treat flattened spatial dimensions as a sequence and applies 1D convolutional layers to capture spectral correlations, compressing the input spectral channels down to specified latent dimensions. The latent representation uses sigmoid activation to constrain values between 0 and 1.

The decoder reverses this process, starting with 1D transposed convolutional layers to expand the latent representation back to the original spectral channels. It then reshapes the data back to its spatial dimensions and uses 2D transposed convolutional layers with residual blocks to reconstruct the original image size. The model includes residual blocks in both spatial and spectral processing paths to enhance feature learning and mitigate vanishing gradient issues

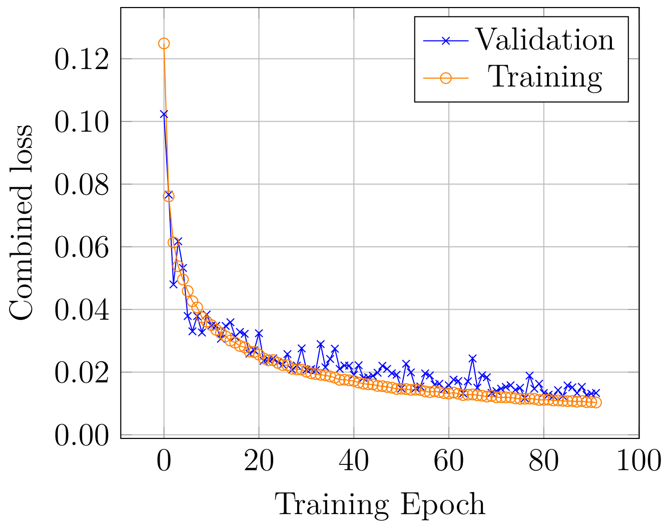
8.5.3 Model training

Both models were trained using the Adam optimizer with initial learning rate of 1×10^{-4}, combined with a cosine decay learning rate schedule. The decay schedule gradually reduces the learning rate over the course of training while maintaining a small non-zero learning rate, which helps to prevent convergence and improve generalization. The Adam optimizer was configured with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1 \times 10^{-7}$. Additionally, gradient clipping with a norm of 0.5 was applied to stabilize training and prevent exploding gradients. Both models were trained with a latent space of 128. As the primary loss function, combined loss was used, which is a sum of MSE and SSIM with equal weights. Early stopping was implemented and triggered for both models around 90 epochs after 15 consecutive epochs without improvement in validation loss in both models.

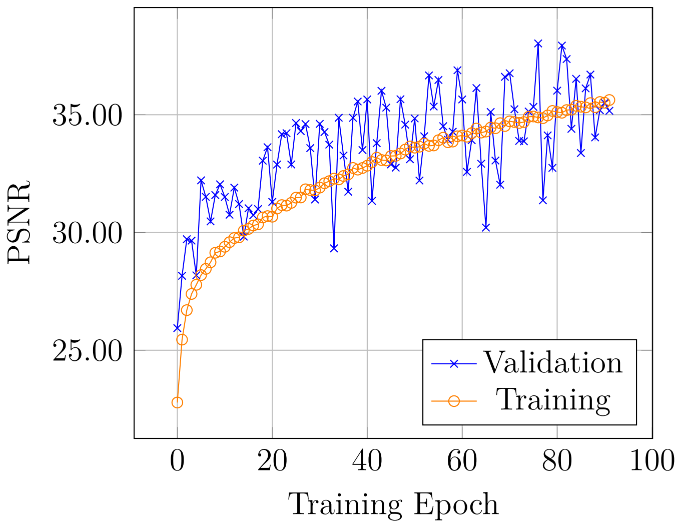
The training and validation curves demonstrate stable convergence for both architectures. A consistent decrease in training and validation loss indicates effective learning. SSIM and PSNR metrics show improvement in reconstruction quality over epochs. Training and validation curves for 3D Convolutional Autoencoder are presented in Figure 8.2. Similarly, training and validation curves for 2D-1D Convolutional Autoencoder are presented in Figure 8.3.

8.6 Summary

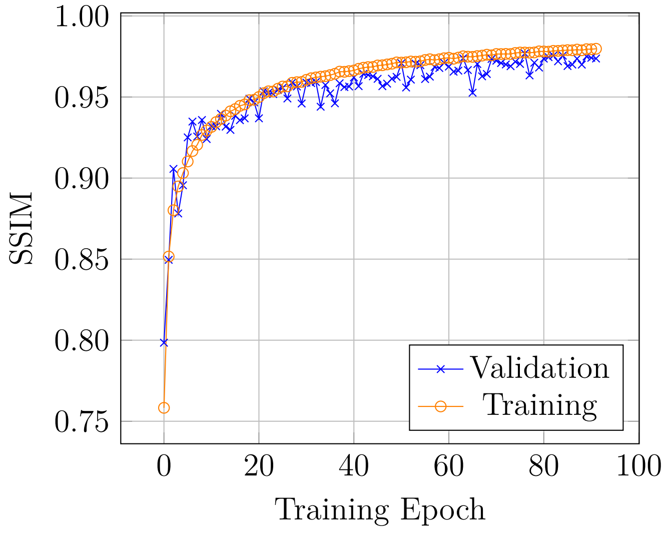
This chapter presented a lossy HSI compression framework based on Residual Convolutional Autoencoders. The fundamental principles of autoencoders were explained, along with the explanation of the residual connections. Two different architectures were presented: a RCAE3D and RCAE2D1D. The RCAE3D model with a kernel depth of one, which treats the spectral dimension as channels, reduces computational complexity. In contrast, the RCAE2D1D model separates spatial and spectral processing into different layers, allowing for more specialized feature extraction. Both models were trained using the same optimization and loss functions, and their training metrics were presented. Training results demonstrate that both models are capable of effectively compressing and reconstructing HSI data.



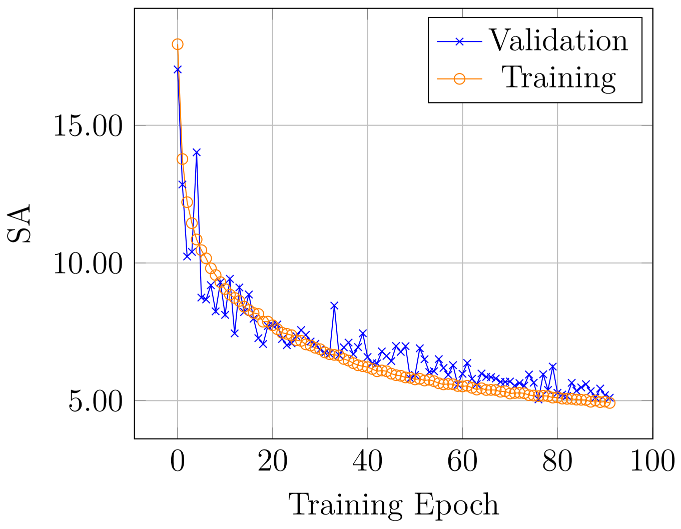
(a) Combined loss over Epochs



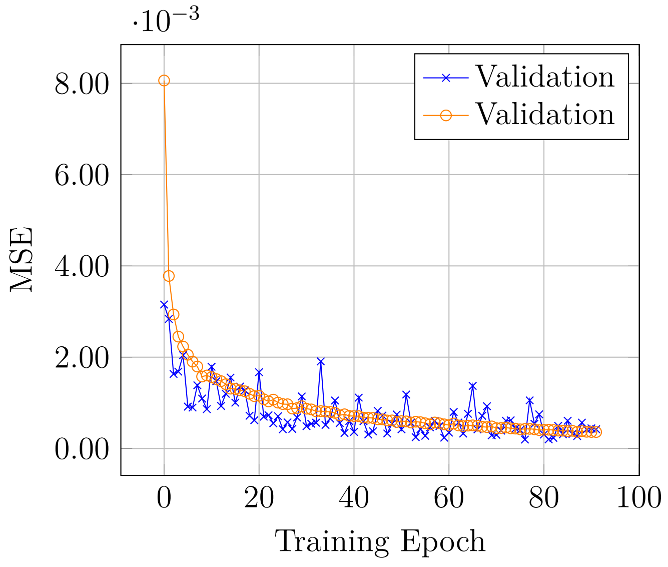
(b) PSNR over Epochs



(c) SSIM over Epochs

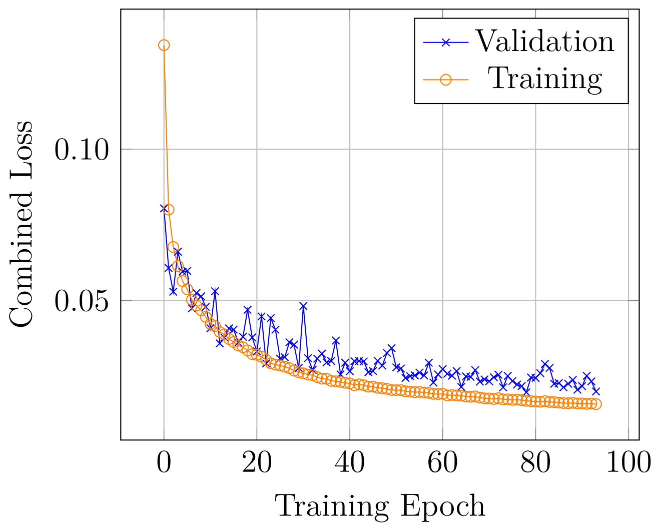


(d) Spectral Angle over Epochs

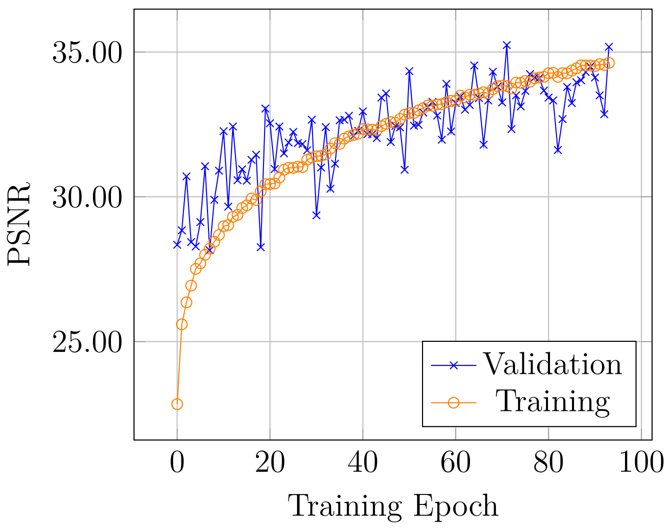


(e) MSE over Epochs

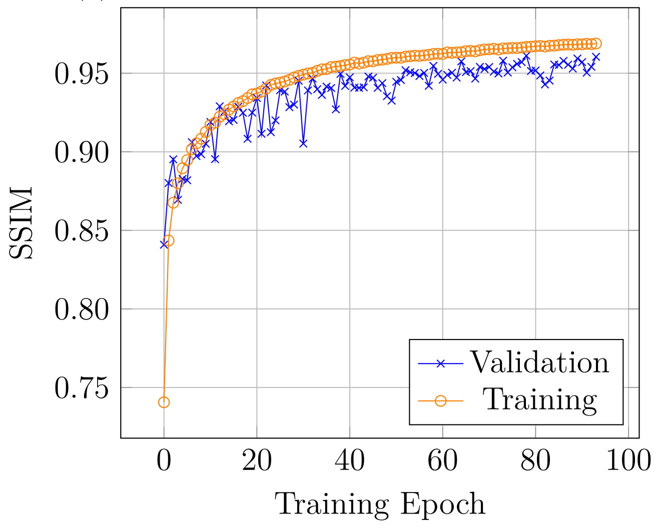
Figure 8.2: Training and Validation Metrics over Epochs for RCA3D



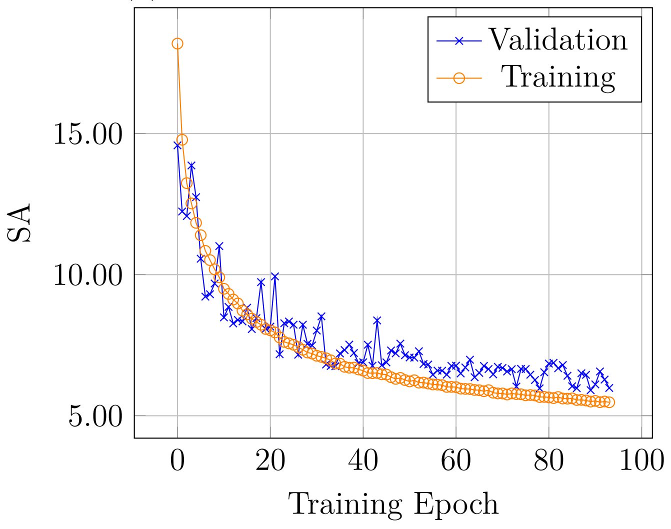
(a) Combined loss over Epochs



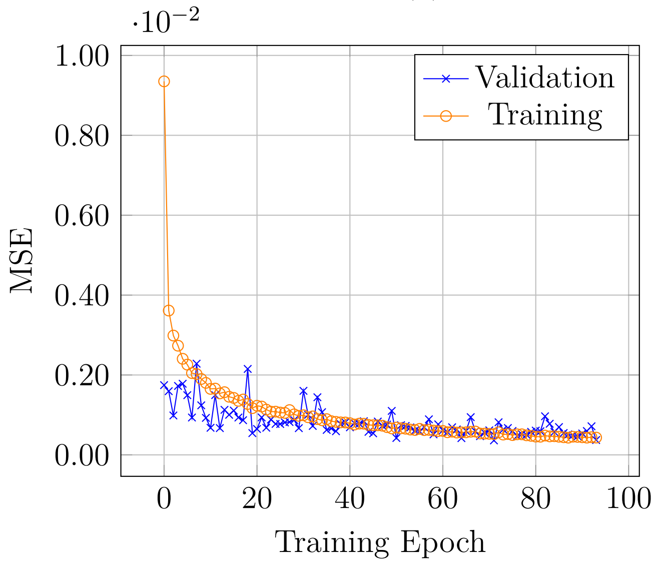
(b) PSNR over Epochs



(c) SSIM over Epochs



(d) Spectral Angle over Epochs



(e) MSE over Epochs

Figure 8.3: Training and Validation Metrics over Epochs for RCA2D1D

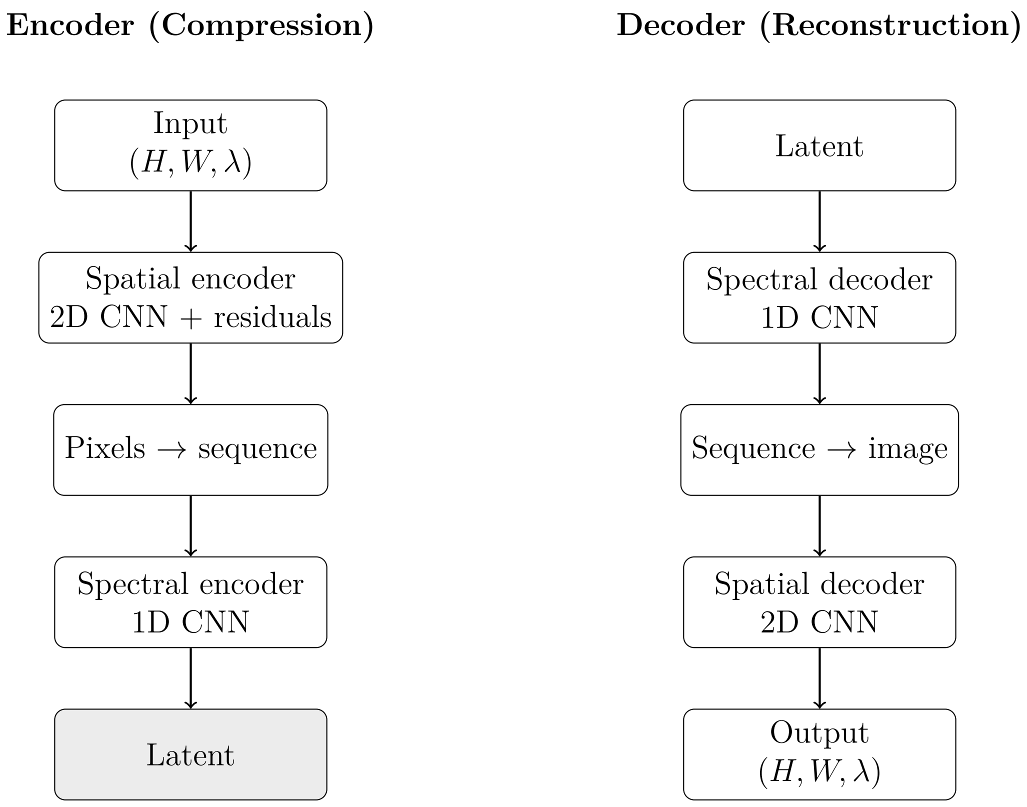


Figure 8.4: RCAE2D1D architecture. Left: encoder. Right: decoder. λ is the number of spectral bands.

Chapter 9

Lossy compression model: Reduced Complexity General Divisive Normalization Autoencoder

9.1 Introduction

This model implements a lossy compression method based on the autoencoder architecture. It varies from the 3D Autoencoder described in Sections 8 by using General Divisive Normalization (GDN) layers instead of standard convolutional layers. It also applies the Karhunen-Loève Transform (KLT) to the input data before feeding it into the autoencoder. Yet another difference is the use of a hyperprior network to model the distribution of the latent representation for better compression performance. Lastly, the model also incorporates an autoregressive context model to further improve the entropy coding of the latent representation.

9.2 Related works

This model was greatly inspired by the work of Mijares i Verdú et al. [25] who proposed the use of GDN layers in autoencoders for hyperspectral image compression. They also demonstrated that first applying the KLT to the data as a preprocessing step can significantly reduce the computational complexity of the model while preserving both compression performance and reconstruction quality. The work presented in this thesis builds upon these ideas by implementing a similar architecture and training procedure, but adapting it to the specific dataset and requirements of this thesis.

9.3 Model architecture

9.3.1 Overall structure

The architecture of the Reduced Complexity General Divisive Normalization Autoencoder (RCGDNAE) consists of an encoder and a decoder, both utilizing GDN layers. When encoding the input data, the model first applies the KLT to decorrelate the spectral bands. The encoder then compresses the decorrelated data into a lower-dimensional representation using a series of GDN layers. The decoder reconstructs the original data from this compressed representation by applying inverse GDN layers followed by the inverse KLT.

9.3.2 General Divisive Normalisation

The GDN layer applies a non-linear transformation to the data at its input, which helps to capture complex relationships between the input features. The GDN transformation's mathematical definition is given by equation 9.1:

$$y_i = \frac{x_i}{\left(\beta_i + \sum_j \gamma_{ij} |x_j|^\alpha\right)^{\frac{1}{\alpha}}} \tag{9.1}$$

where y_i is the output of the GDN layer, x_i is the input, β_i and γ_{ij} are learnable parameters, and α is a hyperparameter that controls the non-linearity of the transformation. The GDN function is particularly effective for image compression tasks as it has been shown to improve the entropy of the latent representation which, in turn, leads to higher compression ratios.

9.3.3 Karhunen-Loève Transform

The KLT is a linear transformation that decorrelates the input data by projecting it onto a new surface defined by the eigenvectors of the data's covariance matrix [15]. This transformation is useful as it maximizes the variance along each new axis, allowing for more efficient compression. The KLT is defined mathematically by equation 9.2:

$$y = W^T(x - \mu) \tag{9.2}$$

where y is the transformed data, W is the matrix of eigenvectors, x is the input data, and μ is the mean of the input data. To ensure the model has a low computational complexity, the W matrix used was precomputed from the training dataset and is fixed during training of the rest of the model.

9.3.4 Hyperprior network

The hyperprior network is an additional neural network that models the distribution of the points in the latent space produced by the encoder. It consists of two parts: a hyperprior analysis network and a hyperprior synthesis network. The hyperprior analysis network takes the latent representation from the encoder and produces a set of parameters that describe the distribution of the latent space. The hyperprior synthesis network then uses these parameters to generate a set of mean and standard deviation values that are used to model the distribution of the latent representation for entropy coding. This approach allows for better compression performance by providing more accurate estimates of the latent representation's distribution for entropy coding.

9.3.5 Autoregressive context model

The autoregressive context model is used to further improve the entropy coding of the latent representation. It models the dependencies between different elements in the latent space by conditioning the probability of each element on the previous elements. This is achieved using a series of convolutional layers that process the latent representation in a sequential manner. The output of the context model is then combined with the output of the hyperprior synthesis network to produce the final mean and standard deviation values used for entropy coding.

9.3.6 Variable Encoding Rate

To enable variable encoding rates, the model incorporates a rate-distortion trade-off mechanism. During training, a Lagrangian multiplier λ is introduced to balance the trade-off between the bitrate and the distortion of the reconstructed image. By adjusting λ, the model can be trained to prioritize either lower bitrate or higher reconstruction quality, allowing for flexible encoding rates based on application requirements. The lambda value is incorporated into the loss function as follows:

$$L = D + \lambda R \tag{9.3}$$

where L is the total loss, D is the distortion (measured by MAE), R is the estimated bitrate, and λ is the Lagrangian multiplier. Three versions of the model were trained with different λ values: 0 (pure MAE loss), 0.001, and 0.01. These different models allow for the evaluation of the trade-off between compression rate and reconstruction quality.

9.3.7 Model implementation

The RCGDNAE model was implemented using the TensorFlow framework. The model, and all of the non-standard layers are defined in the `RCGDNAE.py` file. The encoder itself was constructed using four GDN layers interleaved with convolutional layers that progressively reduce the spatial dimensions of the input data. The decoder mirrors this structure using inverse GDN layers and transposed convolutional layers to reconstruct the original data. The hyperprior network and context model were also implemented using convolutional layers, with the hyperprior analysis network reducing the spatial dimensions of the latent representation and the hyperprior synthesis network increasing them back to match the latent representation's dimensions. The context model processes the latent representation using masked convolutions to ensure the autoregressive property is maintained.

9.4 Training

The model was trained in three different configurations corresponding to different target bitrates, achieved by varying the Lagrangian multiplier λ in the loss function. The three configurations are referred to as RCGDNAE-Low, RCGDNAE-Medium, and RCGDNAE-High, corresponding to λ values of 0.01, 0.001, and 0, respectively. The training procedure for all three configurations was identical, with the only difference being the value of λ used in the loss function. The training consisted of 100 epochs and a learning rate of 0.0001 for the RCGDNAE-Medium and RCGDNAE-High models, while the RCGDNAE-Low model was trained for 500 epochs with a learning rate of 0.00001. The curves for the monitored metrics for each model can be seen in Figures 9.4, 9.3, and 9.2.

The training and validation metrics lack the usual plateau patterns often seen in overfitting scenarios. This suggests that the model is still learning and has not yet reached their optimal performance. This is a positive indication that the model has the potential for further improvement with continued training. However, it wasn't possible to explore longer training due to computational resource constraints.

9.5 Summary

The RCGDNAE model demonstrates effective training behavior, as evidenced by the consistent improvement in loss and evaluation metrics over the training epochs. The model itself is quite simple computationally, with a modest number of parameters and a straightforward architecture. The RCGDNAE model was implemented with 4 main components: the encoder, decoder, hyperprior network, and context model. The encoder and decoder both consist of a series of convolutional layers followed by GDN or inverse GDN layers. The hyperprior network consists of convolutional layers that process the latent representation to produce the parameters for the distribution modeling. The context model also consists of convolutional layers that process the latent representation in a sequential manner. The overall architecture of the model is designed to efficiently compress hyperspectral images while maintaining high reconstruction quality.

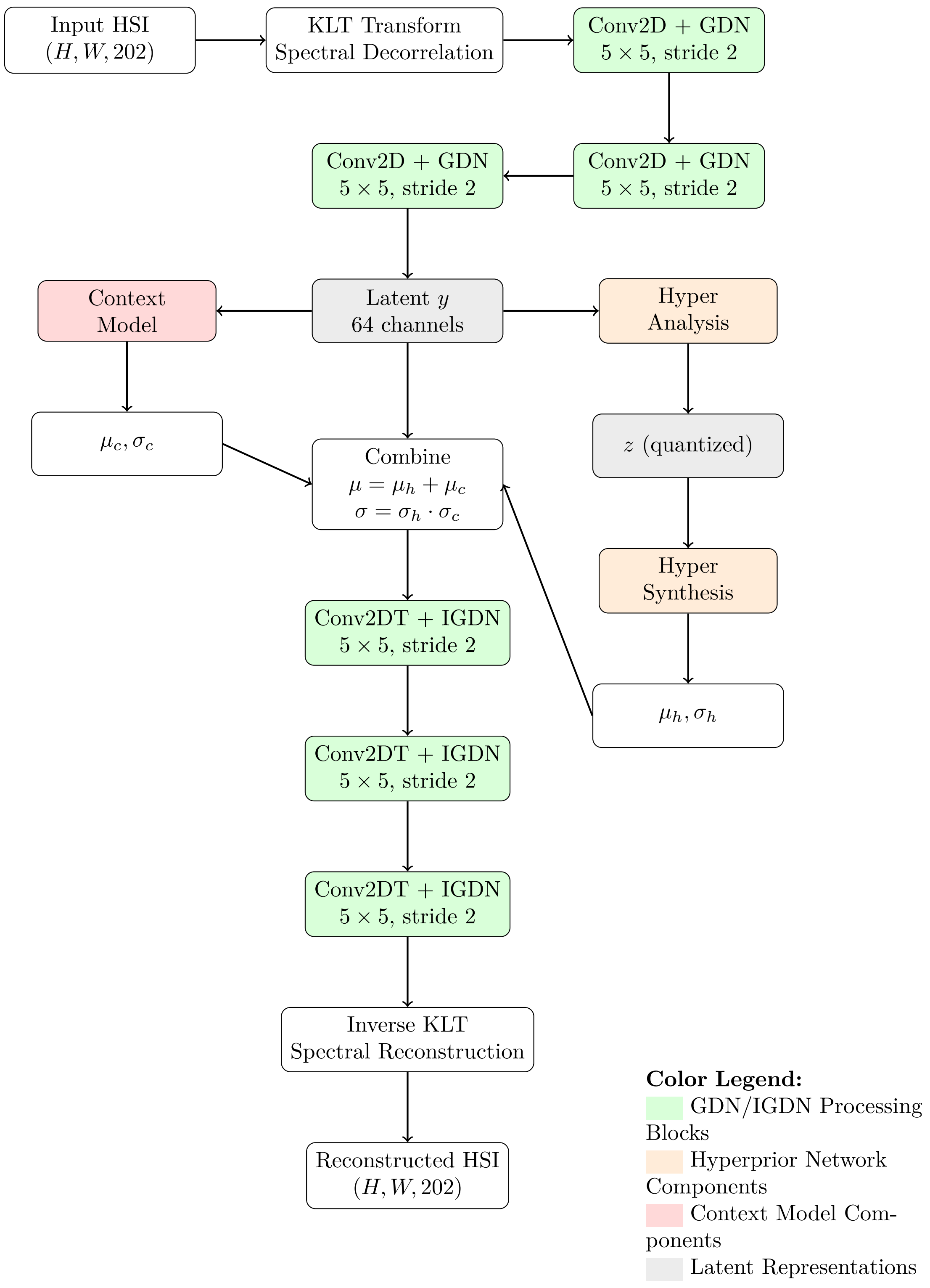
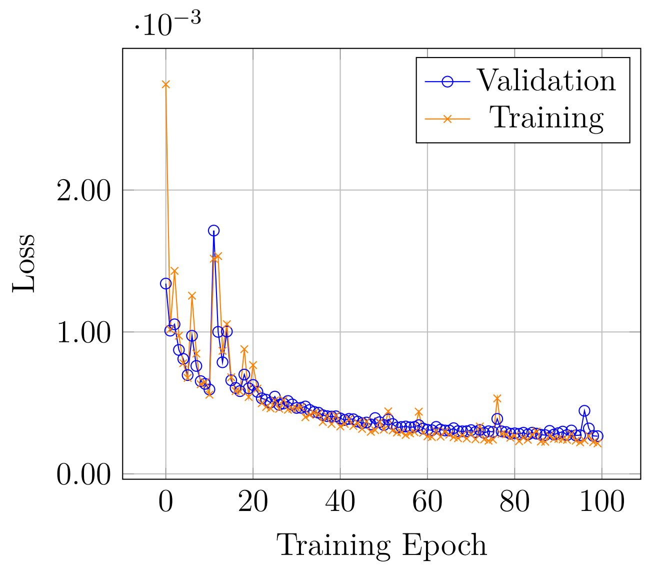
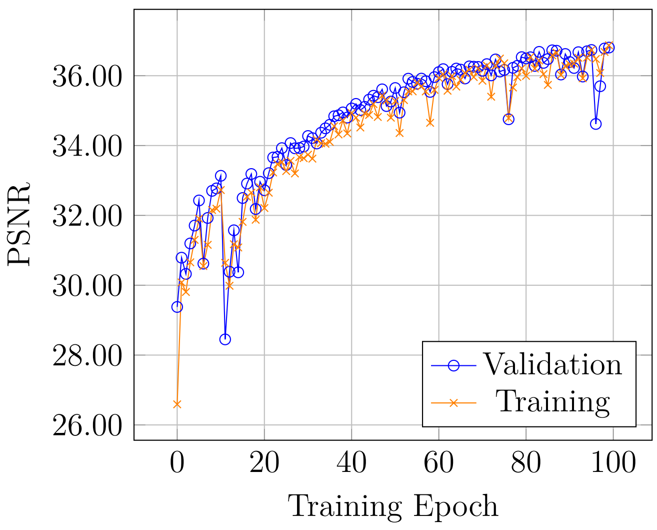


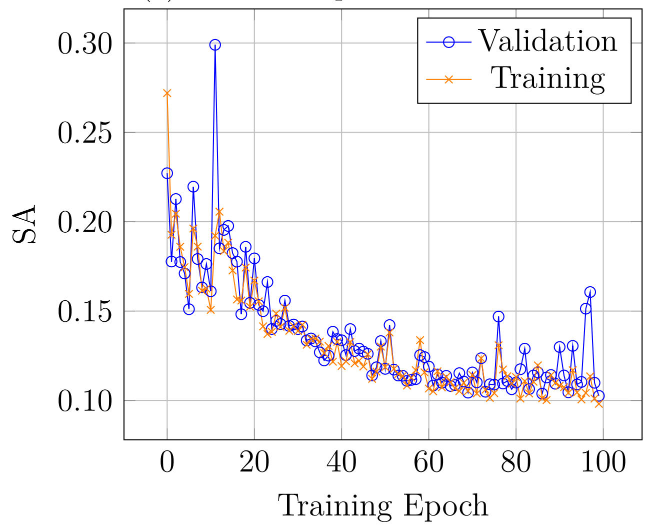
Figure 9.1: RCGDNAE architecture with KLT preprocessing, GDN/IGDN layers, hyperprior network, and context model for variable-rate lossy HSI compression.



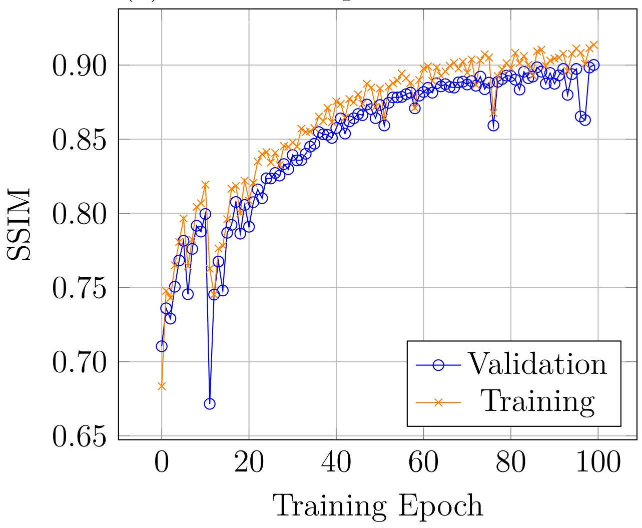
(a) Loss over Epochs



(b) PSNR over Epochs

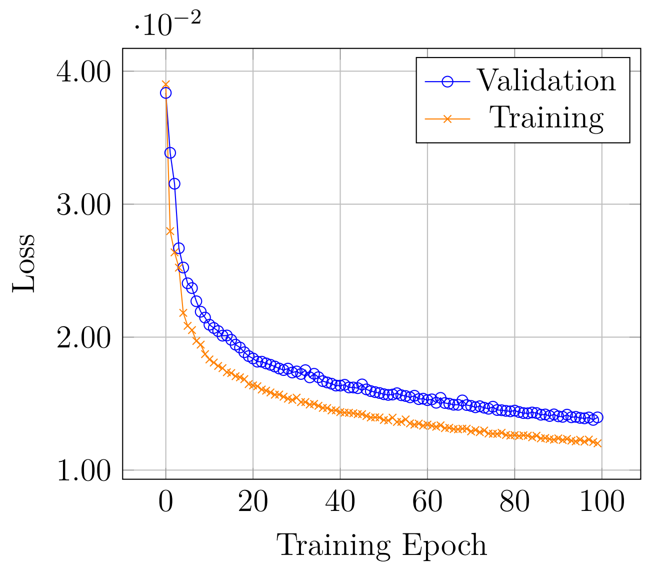


(c) SA over Epochs

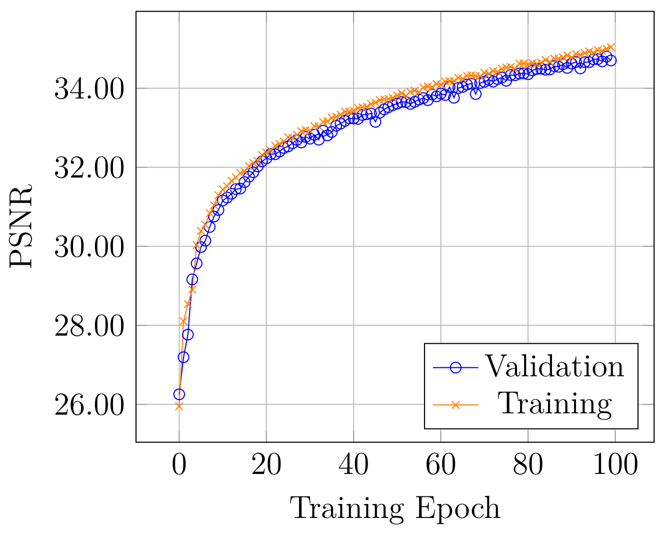


(d) SSIM over Epochs

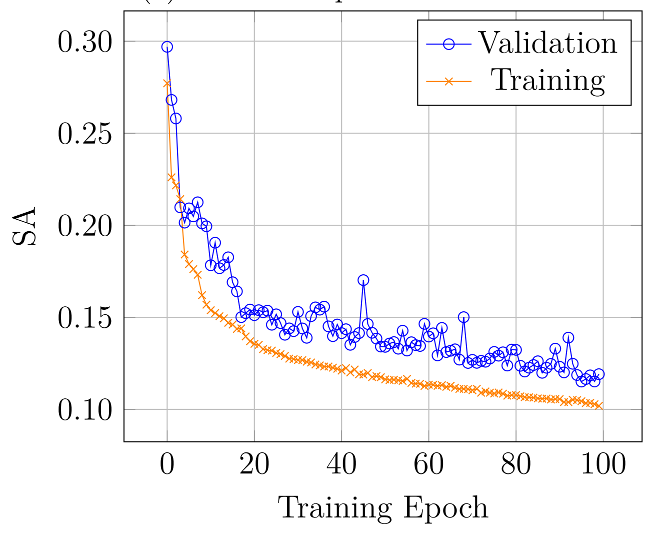
Figure 9.2: Training and Validation Metrics over Epochs for RCGDNAE high bitrate model



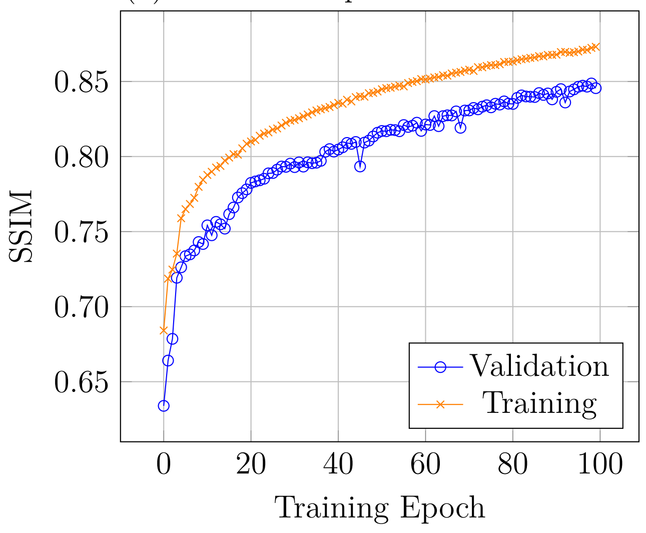
(a) Loss over Epochs



(b) PSNR over Epochs

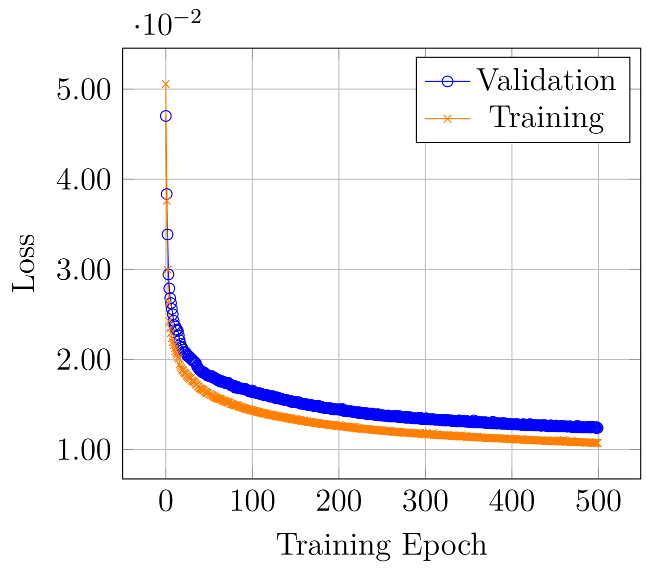


(c) SA over Epochs

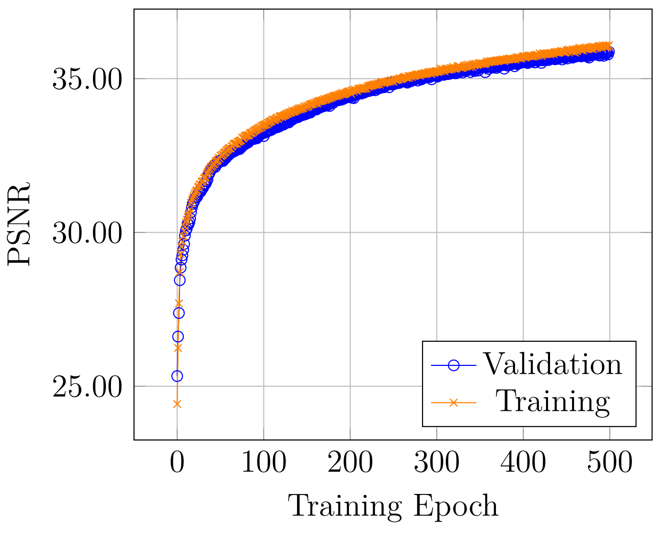


(d) SSIM over Epochs

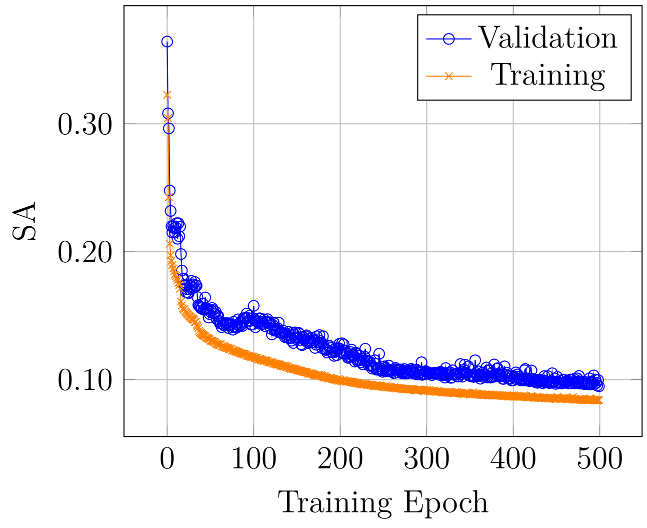
Figure 9.3: Training and Validation Metrics over Epochs for RCGDNAE medium bitrate model.



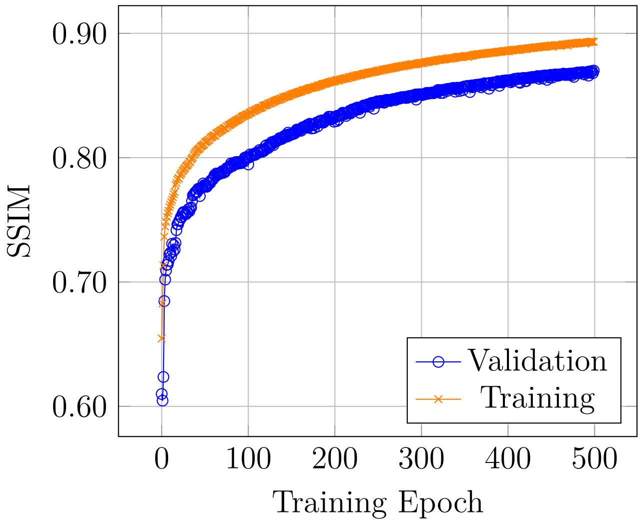
(a) Loss over Epochs



(b) PSNR over Epochs



(c) SA over Epochs



(d) SSIM over Epochs

Figure 9.4: Training and Validation Metrics over Epochs for RCGDNAE low bitrate Model

Chapter 10

Small Segmentation model

In addition to the aforementioned compression models, a small segmentation model was also implemented as part of this thesis work. The segmentation model is designed specifically for memory-efficient per-pixel classification of hyperspectral data, requiring approximately 1–2 GB of VRAM for a batch size of 1.

10.1 Model architecture

The small_segmenter takes 5D input tensors of shape $(B, H, W, D, 1)$ where B is the batch size, $H \times W$ is the spatial resolution (128×128), and D is the number of spectral bands (202 for HySpecNet). The model outputs 2D spatial segmentation masks of shape (B, H, W, C) where C is the number of classes.

Unlike traditional segmentation architectures such as U-Net [20] that use spatial downsampling and upsampling with skip connections, this model employs a progressive spectral reduction strategy that preserves full spatial resolution throughout the network. This design choice is motivated by the nature of hyperspectral segmentation, where maintaining spatial detail is critical for accurate per-pixel classification. The designed Small Segmentation architecture is presented in Figure 10.1.

10.2 Model implementation

The model uses a series of 3D convolutions to ensure both spatial and spectral features are utilized. Next, the architecture collapses the spectral dimension using a $1 \times 1 \times 1$ convolution followed by mean reduction, and then applies 2D convolutions to refine spatial features before the final classification layer. Finally, a softmax activation produces per-pixel class probabilities. Additionally, batch normalization is applied after each convolution, and a dropout layer is used before the final classification layer to ensure regularization.

10.3 Training

The model was trained on hyperspectral images with corresponding segmentation maps using categorical cross-entropy loss and the Adam optimizer. The training process was set to run for 80 epochs, with a learning rate of 0.001. The training and validation metrics over epochs are shown in Figure 10.2. It is clear that both the training and validation loss decrease over time, while the accuracy metrics increase, indicating that the model is learning effectively without signs of overfitting. There is, however, still some room for improvement, as the validation loss and accuracy metrics have not yet plateaued, suggesting that further training could yield better performance. However, due to both time and computational resource constraints, it was not feasible to continue training for more epochs to explore this potential improvement.

10.4 Summary

The small segmentation model works well as a symbolic representation of the next step in the data processing pipeline, which uses compression as a pre-processing step for downstream tasks such as segmentation. The model itself is simple with a straightforward architecture that still manages to use both spatial and spectral features effectively.

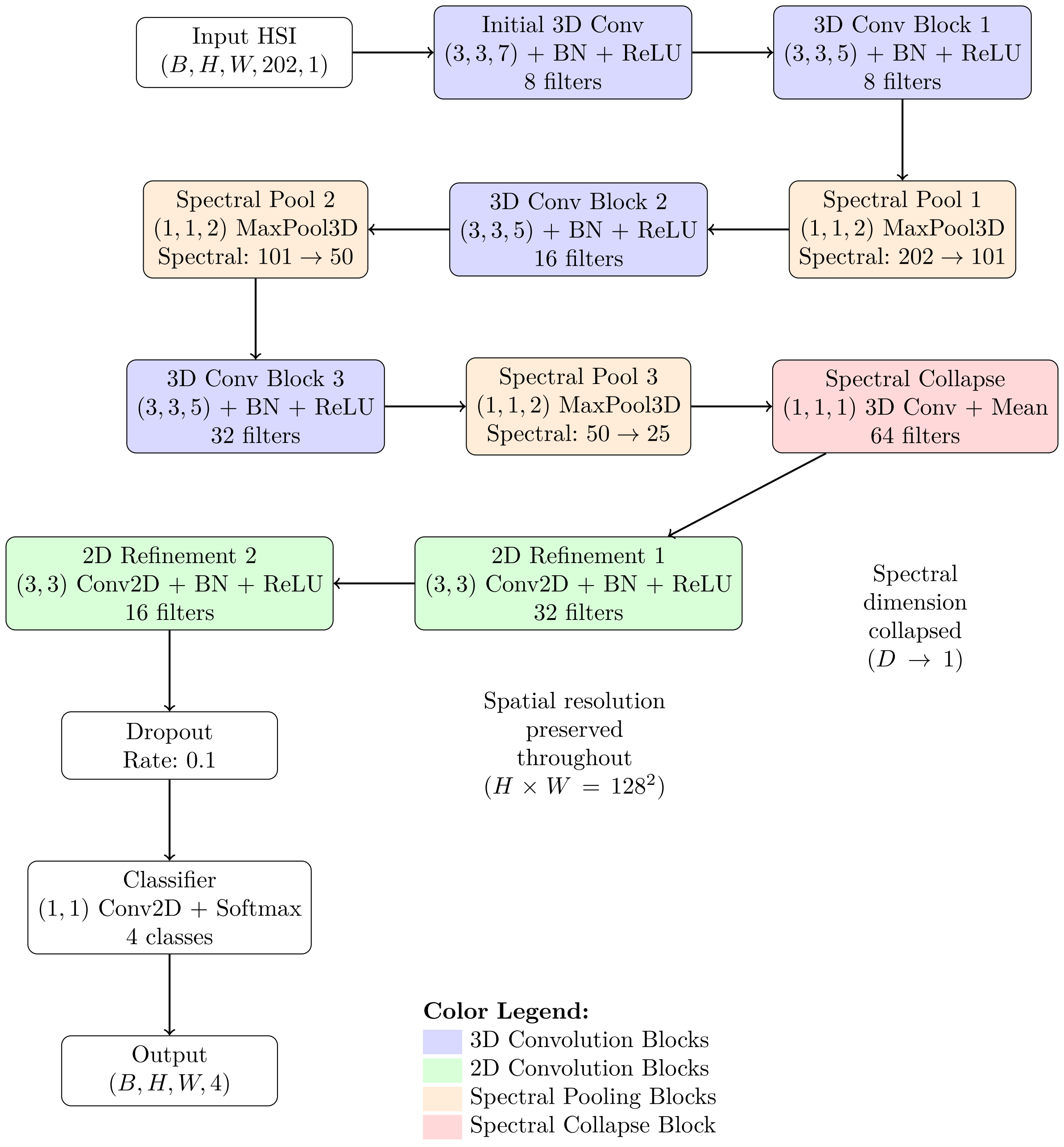
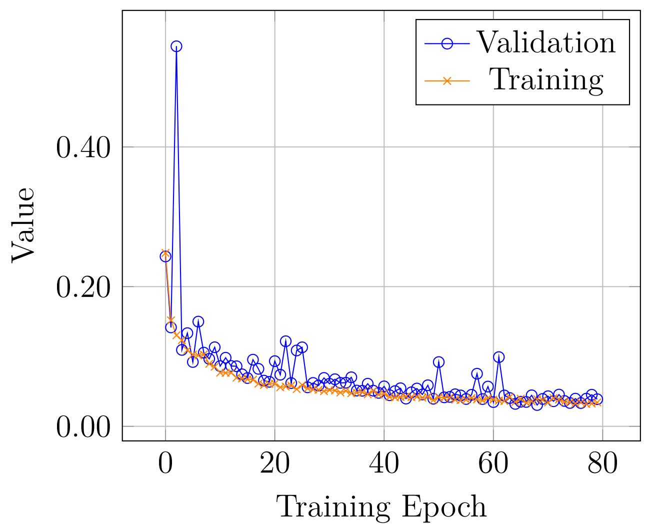
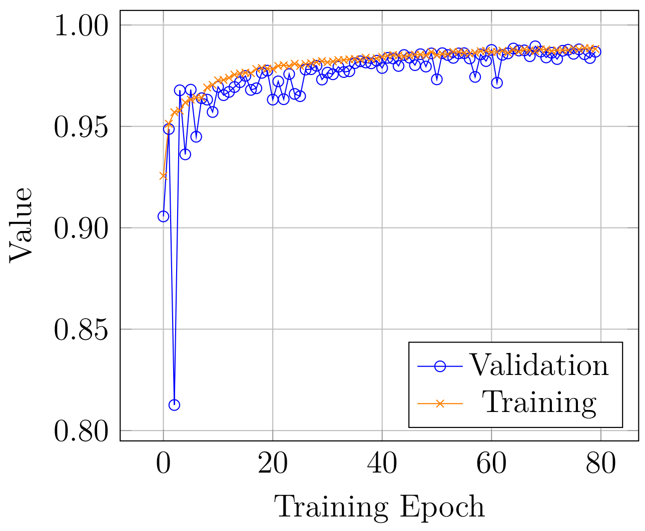


Figure 10.1: Small segmentation model architecture for memory-efficient HSI segmentation. The model progressively reduces spectral dimensionality while preserving full spatial resolution for per-pixel classification.



(a) Loss over Epochs



(b) Accuracy over Epochs

Figure 10.2: Training and Validation Metrics over Epochs for the Small Segmentation model

Chapter 11

Verification and validation

11.1 Evaluation metrics for image compression

For evaluating the performance of training and validating image compression models, appropriate metrics are needed to quantify the quality of compressed images. Some of the most common metrics used in HSI compression are Peak Signal-to-Noise Ratio (PSNR), Structural Similarity Index Measure (SSIM), Mean Squared Error (MSE), and Spectral Angle (SA)[17, 19]. These metrics help to quantify the differences between the original and reconstructed images, providing insights into the effectiveness of models used for compression. Two metrics are particularly well suited for examining HSI compression performance, SA and SSIM. PSNR and MSE are more general metrics often used in image processing tasks. MSE measures the average squared difference between the original and reconstructed pixel values, where PSNR is MSE logarithmic scale representation expressed in decibels (dB). [17] Unlike PSNR and the MSE, SSIM is a metric designed to align more closely with visual perception. SSIM evaluates the similarity between two images based on their structural information, luminance, and contrast. [17] SA measures the spectral similarity between the original and reconstructed image by calculating the angle between their spectral vectors. [19]

11.1.1 Structural Similarity Index Measure

The Structural Similarity Index Measure (SSIM) is a metric that quantifies many image quality aspects. It combines data about the differences in luminance, contrast, and structure between two images into a single value. The formula for calculating SSIM between two images x and y is given by equation 11.1:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{11.1}$$

where μ_x and μ_y are the average pixel values of images x and y, σ_x^2 and σ_y^2 are the variances, σ_{xy} is the covariance between the two images, and C_1 and C_2 are constants to ensure no division by zero takes place. The SSIM value theoretically ranges from -1 to 1, where a value of 1 indicates perfect similarity between the two images, while values close to 0 indicate low similarity and negative values indicate very opposite similarity. In practice, for image compression tasks, SSIM values are typically between 0 and 1, with higher values indicating better reconstruction quality.

11.1.2 Spectral Angle

The formula for calculating the SA between two spectral vectors **a** and **b** is given by equation 11.2:

$$\mathrm{SA}(\mathbf{a}, \mathbf{b}) = \arccos\left(\frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\|\|\mathbf{b}\|}\right) \tag{11.2}$$

where $\mathbf{a} \cdot \mathbf{b}$ is the dot product of the vectors, and $\|\mathbf{a}\|$ and $\|\mathbf{b}\|$ are the magnitudes of the vectors. SA is typically measured in degrees, with smaller angles indicating greater similarity between the spectral signatures.

11.1.3 Mean Squared Error

Mean Squared Error (MSE) is a common metric often used in machine learning. It measures the average square of the difference of each point. In the case of images, it measures the average squared difference between the original and reconstructed pixel values. The formula for calculating MSE between two images x and y is shown in equation 11.3:

$$\mathrm{MSE}(x, y) = \frac{1}{N}\sum_{i=1}^{N}(x_i - y_i)^2 \tag{11.3}$$

where N is the total number of pixels in the images, and x_i and y_i are the pixel values at position i. The MSE value is non-negative, with lower values indicating better reconstruction quality.

11.1.4 Peak Signal-to-Noise Ratio

Peak Signal-to-Noise Ratio (PSNR) is a metric used to measure the quality of reconstructed images compared to the original images. It is expressed in decibels (dB) which are logarithmic units. This, in turn, means that small changes in PSNR can represent significant differences in image quality. The formula for calculating PSNR between two images x and y is given by equation 11.4:

$$\text{PSNR}(x, y) = 10 \cdot \log_{10}\left(\frac{MAX_I^2}{\text{MSE}(x, y)}\right) \tag{11.4}$$

where MAX_I is the maximum possible pixel value of the image. The PSNR value is better when it is higher, with values above 30 dB generally considered good enough for most image compression tasks, while values above 45 dB are considered excellent. It is important to note that PSNR is a logarithmic measure, so a small increase in PSNR can correspond to a significant improvement in image quality.

11.1.5 Bits per pixel per channel

Another metric specific to HSIs is the bits per pixel per channel (bpppc), which represents the average number of bits used to encode each pixel in each spectral channel. The formula for calculating bpppc is given by equation 11.5:

$$\text{bpppc} = \frac{\text{Total Bits in Compressed Image}}{\text{Number of Pixels} \times \text{Number of Spectral Channels}} \tag{11.5}$$

The bpppc metric is particularly important for evaluating the efficiency of compression algorithms, as it directly relates to the amount of data required to represent the image. Lower bpppc values indicate more efficient compression, while higher values indicate less efficient compression.

11.1.6 Error histogram

Besides numerical metrics, visualizing the distribution of reconstruction errors was employed to gain insights into the typical errors made by the compression models. Visualizing the distribution of reconstruction errors can be achieved in the form of a histogram. The histograms present the frequency of errors over some interval, which provides a way to identify patterns in the errors. For instance, a histogram that is heavily skewed toward zero indicates that most reconstruction errors are small, suggesting that the model is performing well. A histogram with a peak away from zero indicates that the model is frequently making larger errors, which may suggest areas for improvement. The best error histogram would ideally show a high frequency of errors close to zero, indicating that the model is accurately reconstructing the images with minimal errors.

11.1.7 Error map

An additional method for visualizing reconstruction errors is through error maps. Error maps are the result of calculating the difference between the original and reconstructed images on a pixel-by-pixel basis. These maps provide a spatial representation of where the model is performing well and where it is struggling. The error maps presented in this thesis use a color scale to represent the magnitude of the errors. Pixels with negative errors are shown in shades of blue, while pixels with positive errors are shown in shades of red, and black represents zero error. This color scheme allows for easy identification of areas where the model is overestimating or underestimating pixel values.

11.2 Evaluation of semantic segmentation

To evaluate the performance of the semantic segmentation model, several common metrics were used, including the confusion matrix, accuracy, positive predictive value (PPV), recall, F1-score, Intersection over Union (IoU), and Area under Curve (AUC). These metrics provide a comprehensive assessment of the model's ability to correctly assign classes to each pixel in the image.

11.2.1 Confusion matrix

A confusion matrix is a table that summarizes the performance of a classification model by comparing the predicted class labels with the true class labels. Each row of the matrix corresponds to the true class (as labeled in the dataset), while each column corresponds to the predicted class (as output by the model). The entries in the matrix represent the number of pixels that were classified into each combination of true and predicted classes. The perfect model would produce a diagonal confusion matrix, where all pixels are correctly classified into their respective classes.

The confusion matrix provides a detailed view of the model's performance, allowing for the identification of specific classes that may be more challenging to classify correctly. Another useful aspect of the confusion matrix is that it can be used to calculate other evaluation metrics such as accuracy, precision, recall, and F1-score. This can be done by summing the appropriate entries in the matrix to obtain the necessary counts for each metric. All of these summations fall into four categories: true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). True positives are the number of pixels that were correctly classified as belonging to a particular class. True negatives are the number of pixels that were correctly classified as not belonging to a particular class. False positives are the number of pixels that were incorrectly classified as belonging to a particular class when they do not. False negatives are the number of pixels that were incorrectly classified as not belonging to a particular class when they do.

11.2.2 Accuracy

Accuracy is a metric that measures the overall correctness of the model's predictions. It is calculated as the ratio of correctly classified pixels to the total number of pixels in the image. The formula for calculating accuracy is given by equation 11.6:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{11.6}$$

The accuracy value ranges from 0 to 1, with higher values indicating better performance.

11.2.3 Positive predictive value

PPV is a metric that directly measures the proportion of positive identifications that were actually correct. It is calculated as the ratio of true positives to the sum of true positives and false positives. The formula for calculating PPV is given by equation 11.7:

$$\text{PPV} = \frac{TP}{TP + FP} \tag{11.7}$$

The PPV value ranges from 0 to 1, with higher values indicating better performance.

11.2.4 Recall

Recall is a metric that measures the model's ability to correctly identify all relevant instances of a particular class. It is calculated as the ratio of true positives to the sum of true positives and false negatives. The formula for calculating recall is given by equation 11.8:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{11.8}$$

The recall value ranges from 0 to 1, with higher values indicating better performance.

11.2.5 F1-score

F1-score is a metric that combines PPV and recall into a single value. It is the harmonic mean of PPV and recall, providing a balanced measure of the model's performance. The formula for calculating F1-score is given by equation 11.9:

$$\text{F1-score} = 2 \cdot \frac{\text{PPV} \cdot \text{Recall}}{\text{PPV} + \text{Recall}} \tag{11.9}$$

The F1-score value ranges from 0 to 1, with higher values indicating better performance.

11.2.6 Intersection over Union

IoU is a metric that measures the overlap between the predicted and true class labels. It is calculated as the ratio of the intersection of the predicted and true class labels to the union of the predicted and true class labels. The formula for calculating IoU is given by equation 11.10:

$$\text{IoU} = \frac{TP}{TP + FP + FN} \tag{11.10}$$

The IoU value ranges from 0 to 1, with higher values indicating better performance.

11.2.7 Area under Curve

AUC is a metric that measures the model's ability to distinguish between different classes. It is calculated as the area under the Receiver Operating Characteristic (ROC) curve, which plots the true positive rate against the false positive rate at various threshold settings. AUC values range from 0 to 1, with higher values indicating better performance. It is particularly useful for evaluating models in imbalanced datasets, where one class may be more prevalent than others. AUC is calculated using numerical integration methods, such as the Riemann sum or trapezoidal rule, to estimate the area under the ROC curve. The formula for calculating AUC is given by equation 11.11:

$$\text{AUC} = \int_0^1 \text{TPR}(t)\, d\text{FPR}(t) \tag{11.11}$$

The perfect model would have an AUC of 1, which means that the model is able to distinguish classes with perfect accuracy and a random model would have an AUC of 0.5 (for a balanced dataset), which means that the model is no better than random guessing.

11.3 Segmentation model evaluation

To evaluate the performance of the semantic segmentation model, the metrics mentioned in Section 11.2 were calculated using the model's predictions on the test dataset. These metrics were first calculated for the uncompressed images to establish a baseline performance for the model. Subsequently, the same metrics were calculated for the images reconstructed from each compression model, which will be further discussed in the section about each model. In addition to numerical metrics, confusion matrices were generated for both the uncompressed and reconstructed images to visually compare the model's performance across different classes. This comprehensive evaluation approach allowed for a thorough assessment of the impact of compression on the semantic segmentation model's performance. The model achieved the results shown in Table 11.2 when evaluated on the uncompressed test dataset. These were calculated from the confusion matrix shown in Figure 11.1. The metrics over all classes are in the Table 11.1.

Table 11.1: Semantic segmentation model evaluation metrics on uncompressed test dataset over all classes

Metric	Average Value
Accuracy	0.9893
F1-Score	0.9042
IoU	0.8446
AUC	0.8735

Table 11.2: Semantic segmentation model evaluation metrics on uncompressed test dataset per class

Metric	Value for class 0	Value for class 1	Value for class 2
PPV	0.8389	0.9911	0.9876
Recall	0.6608	0.9966	0.9715
F1-Score	0.7393	0.9939	0.9795
IoU	0.5864	0.9878	0.9598

An interesting observation from the confusion matrix in Figure 11.1 is that class 3 does not appear in the test set at all. This means that the model was never evaluated on this class, which could potentially skew the overall performance. Upon closer inspection, it was found that it doesn't appear once in the whole dataset. This results in our inability to calculate any meaningful metrics for this class, and thus it was omitted from both Table 11.2 and any future per class metrics tables. Another observation is that class 0 has a significantly lower occurrence rate compared to classes 1 and 2. This class imbalance is reflected in the model's performance, with class 0 having notably lower PPV, recall, F1-score, and IoU compared to the other two classes that exist in the dataset. Overall, the model performs admirably on the uncompressed test dataset, achieving high scores across all metrics.

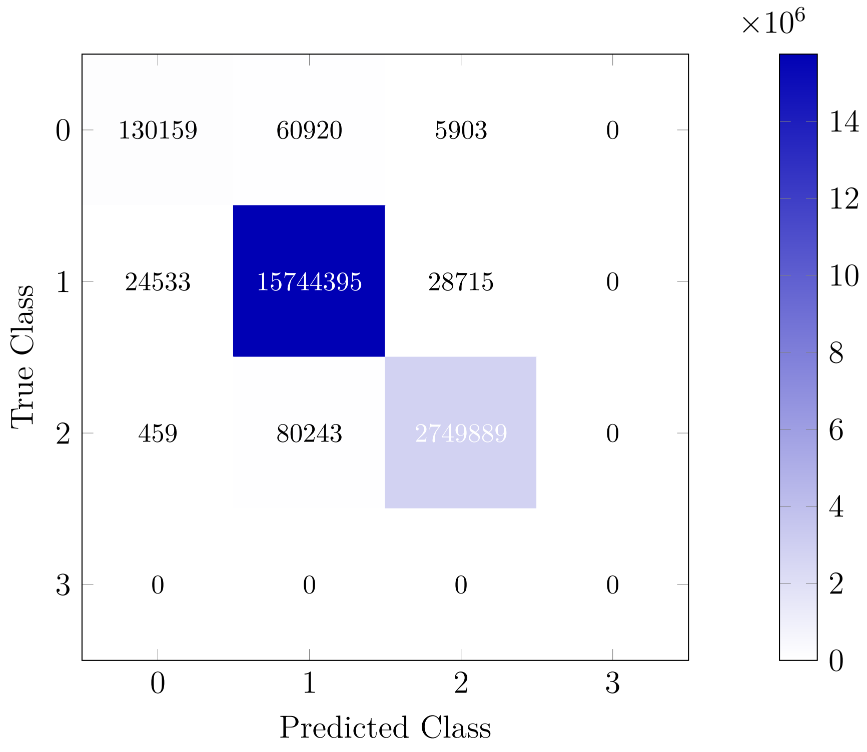
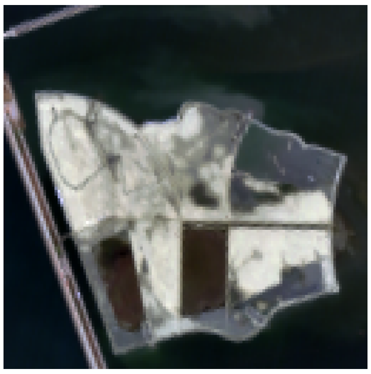
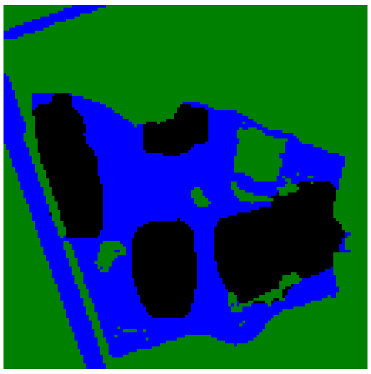


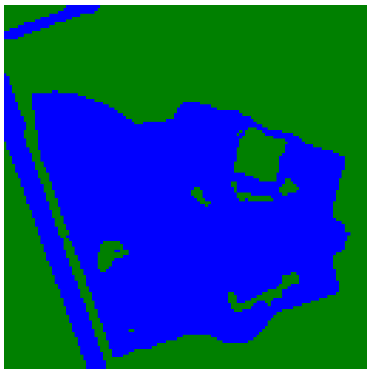
Figure 11.1: Confusion matrix heatmap for baseline segmentation on uncompressed images vs ground truth

11.4 Compression models evaluation

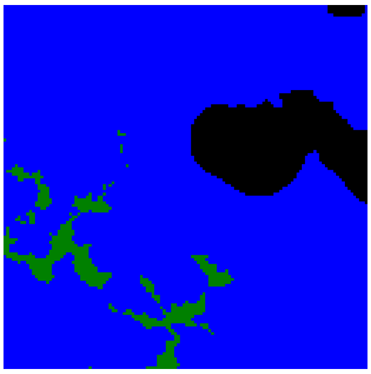
The following subsections describe each model with metrics described in Section 11.1. All of the models were evaluated on the test dataset provided in the HySpecNet-11k dataset. All of the models were assessed using the same set of metrics for consistency and comparability. Most models were evaluated using the easy split of the dataset, except for the RCGDNAE model, which was evaluated using the hard split. This choice was made to ensure all of the models were tested under similar conditions, but also to ensure that no data leakage occurred between training and testing phases. The evaluation metrics used are described in Section 11.1 and include PSNR, SSIM, MSE, SA, bpppc, and compression ratio (CR). All of the metrics were calculated as mean values over the entire test dataset. In addition to numerical metrics and histograms of reconstruction errors, several examples of images from the test dataset were selected to visually compare the original and reconstructed images from each model. Another important aspect of the evaluation was to assess the performance of the semantic segmentation model on the reconstructed images from each compression model. This was done to measure the impact of compression on downstream tasks such as segmentation. To this end, the segmentation model described in Chapter 10 was used to predict class labels for the reconstructed images from each compression model. The segmentation model's performance was then evaluated using the metrics described in Section 11.2. This comprehensive evaluation approach allowed for a

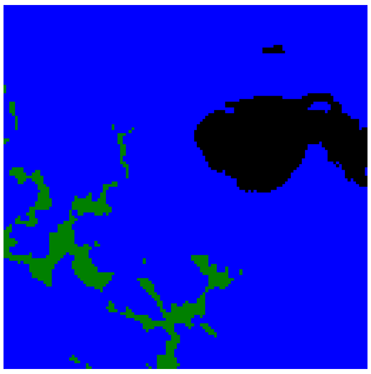

(a) Original image

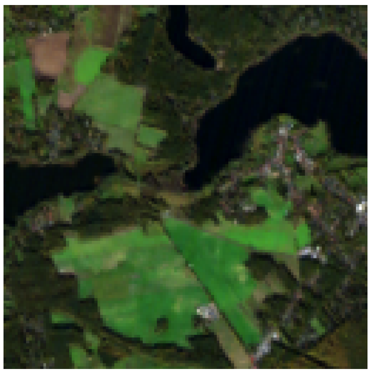

(b) Ground truth

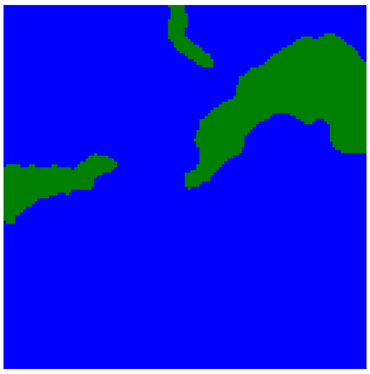

(c) Predicted segmentation

(d) Original image


(e) Ground truth


(f) Predicted segmentation


(g) Original image


(h) Ground truth

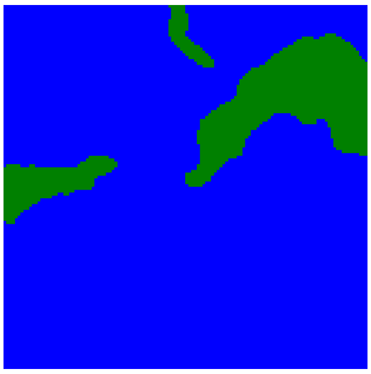

(i) Predicted segmentation

Figure 11.2: Examples of semantic segmentation model predictions on uncompressed test images

thorough assessment of each compression model's performance, both in terms of image quality and its impact on downstream tasks.

11.4.1 LineRWKV

The metrics calculated for the LineRWKV model are shown in Table 11.3. The histogram of reconstruction errors for the model is shown in Figure 11.3. The error map for the best reconstructed image is shown in Figures 11.4.

Table 11.3: LineRWKV evaluation metrics

Metric	Value
PSNR (dB)	44.87
SSIM	0.985
MSE	3.9×10^{-5}
SA (degrees)	3.987
bpppc	2.00
CR	8.00

By analyzing the results, it can be assessed that the model is behaving moderately well. It is able to reconstruct images with high fidelity, as indicated by the high PSNR and SSIM values. The relatively low PSNR values compared to what would be expected from a lossless or near-lossless compression method stem from the low number of learning epochs used during training. However, by applying the residuals to the base model predictions, the model is able to achieve near-lossless reconstruction quality. This, however, comes at a cost to the compression ratio, which is lower than that of other models presented in this thesis.

The error histogram in Figure 11.3 shows that the majority of reconstruction errors are below 0.5×10^{-2}, which indicates that the model is able to reconstruct most images with minimal error. The error histograms in Figures 11.4d, 11.5d, and 11.6d show that the per pixel errors are generally small and in the shape of a normal distribution. This is generally a good sign, as it shows that the model is good at minimizing large errors, although there exists a slight offset in the error distribution, which can be attributed to the fact that the model is not trained for a sufficient number of epochs to fully converge, which would likely result in a more centered error distribution. An additional observation from both the histograms and the error maps shown in Figures 11.4c, 11.5c, and 11.6c is that there are a few relatively large errors that occur in the reconstructed images, which can be seen in the tails of the error histograms and the red and blue areas in the error maps. These large errors mostly occur in the beginnings of the images, which is expected, as the model operates on the principle of predicting the next pixel based on the previous ones, which makes the beginning of the images more difficult to reconstruct accurately.

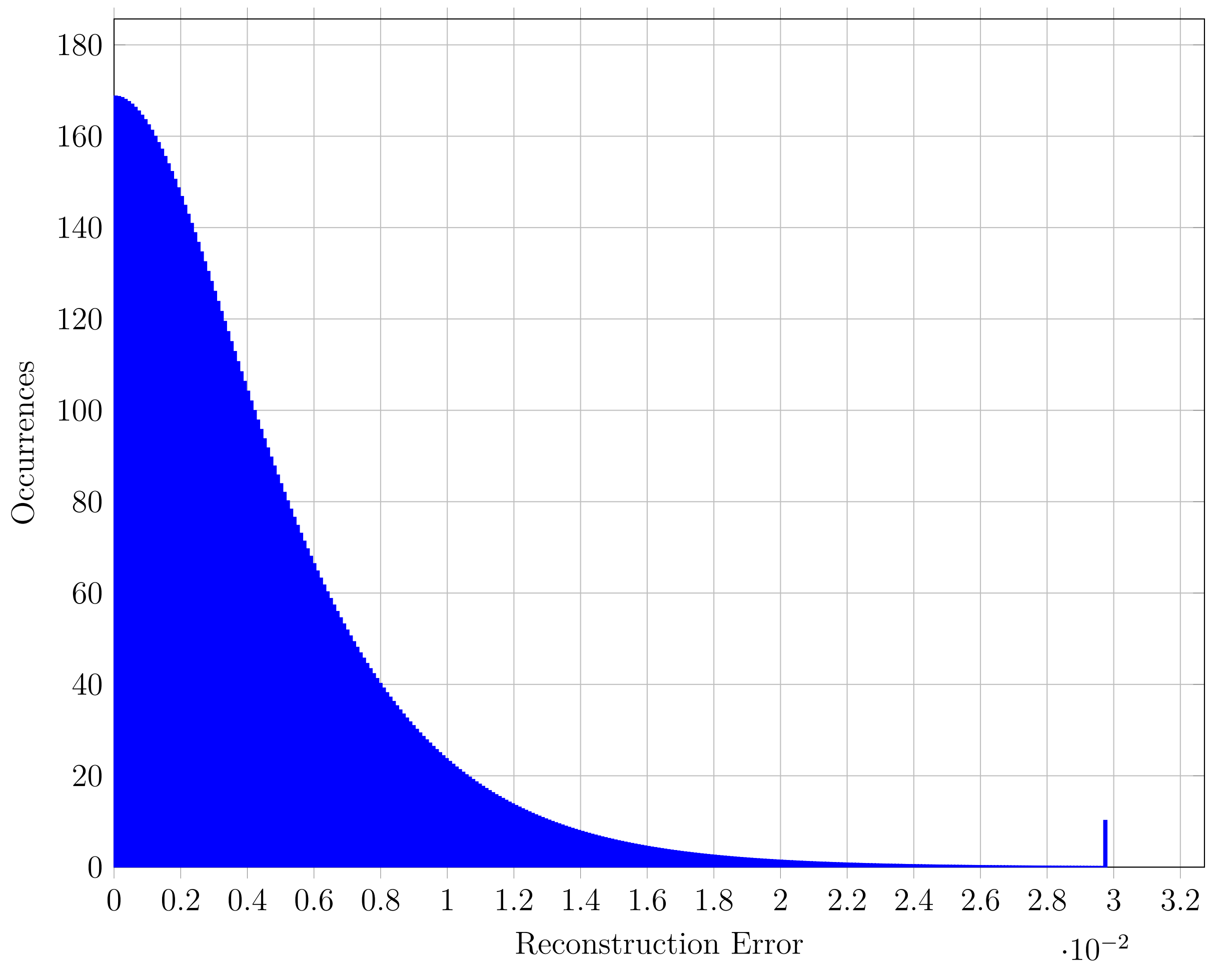
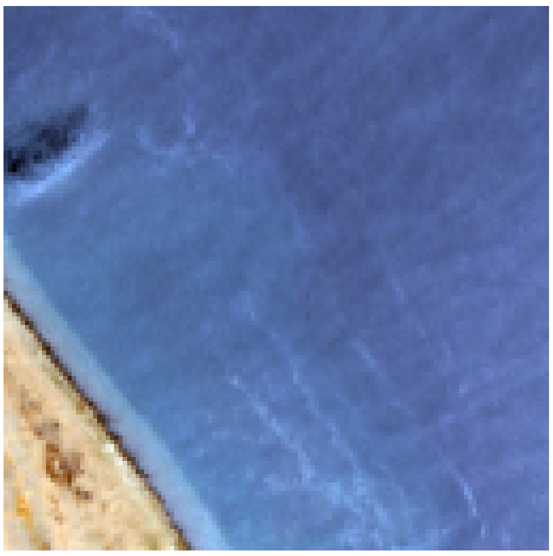
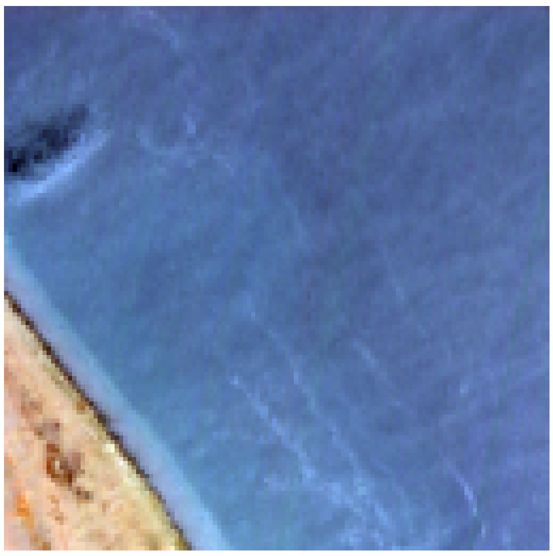
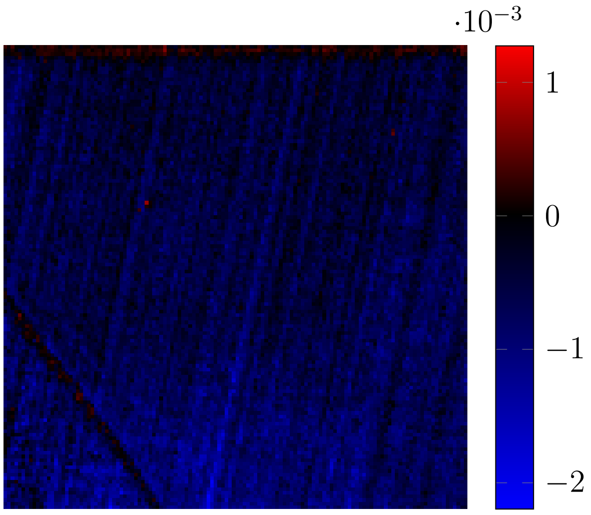


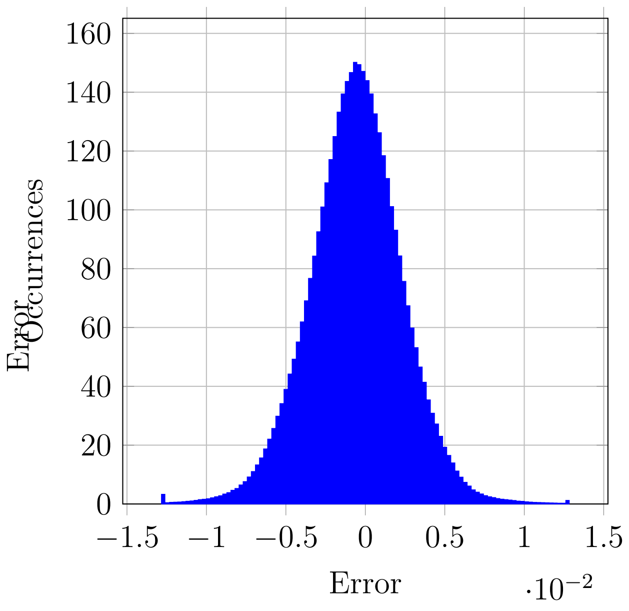
Figure 11.3: Histogram reconstruction error for LineRWKV


(a) Original image


(b) Reconstructed image



(c) Per pixel error map normalized to between -2.1962×10^{-3} and 1.2729×10^{-3}

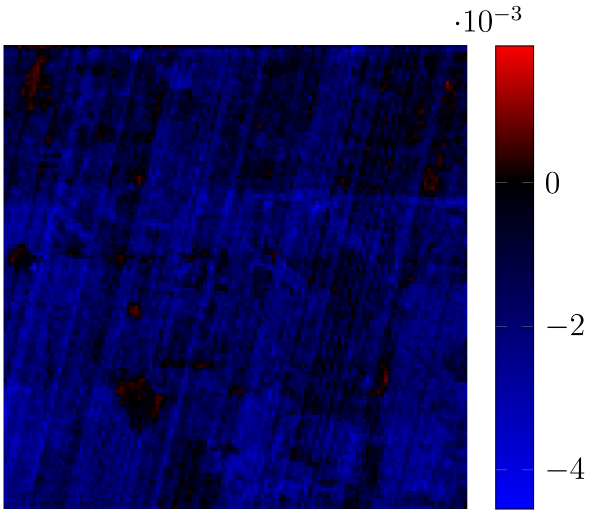


(d) Per pixel error histogram

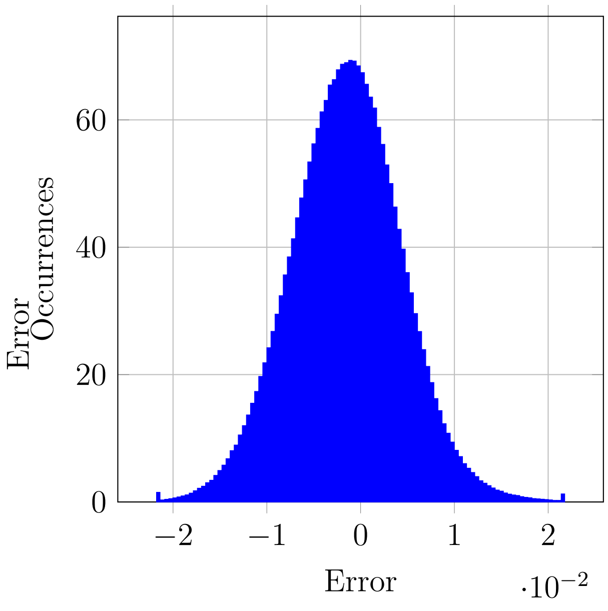
Figure 11.4: Best image reconstruction for LineRWKV model

(a) Original image

(b) Reconstructed image

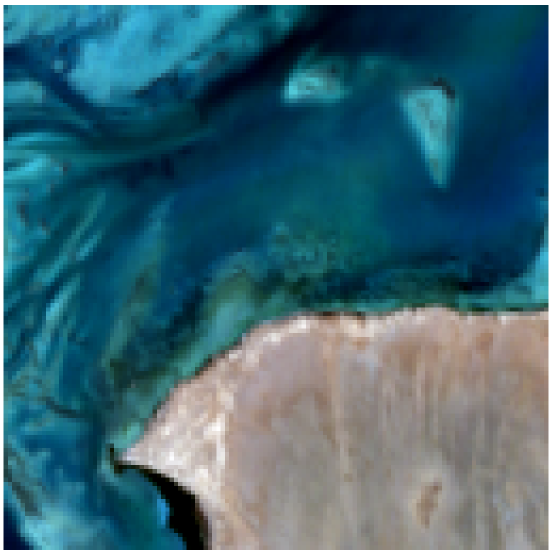


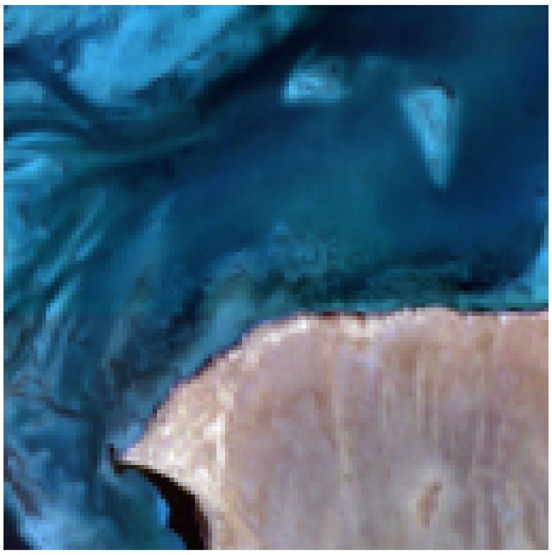
(c) Per pixel error map normalized to between -4.5452×10^{-3} and 1.9049×10^{-3}

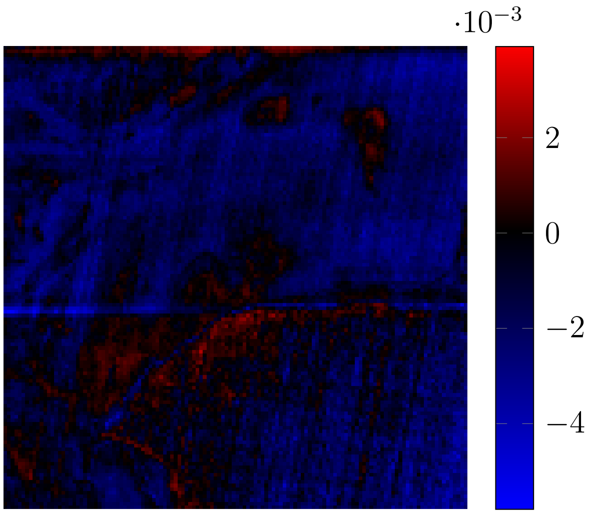


(d) Per pixel error histogram

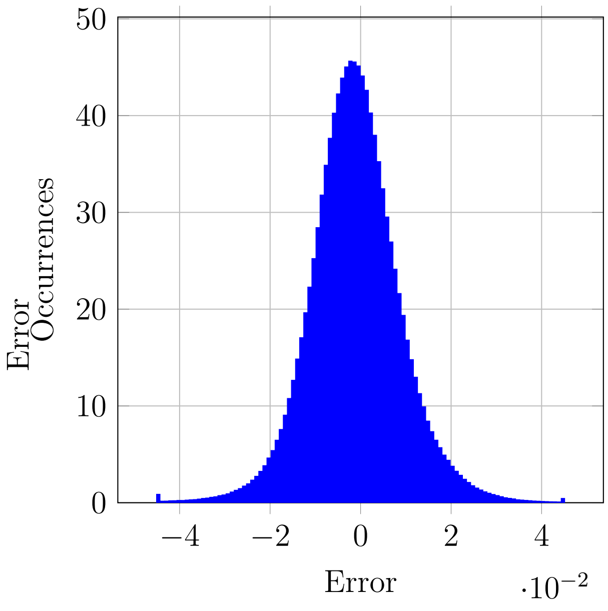
Figure 11.5: Median image reconstruction for LineRWKV model


(a) Original image


(b) Reconstructed image



(c) Per pixel error map normalized to between -5.7977×10^{-3} and 3.9121×10^{-3}



(d) Per pixel error histogram

Figure 11.6: Worst image reconstruction for LineRWKV model

The metrics for the segmentation model using the images after compressing and decompressing with the LineRWKV model are shown in tables 11.4 and 11.5. These metrics were calculated from the confusion matrix shown in Figure 11.7. The results indicate that the segmentation model performs well on the processed images, albeit with a slight decrease in performance compared to the baseline results on uncompressed images. This suggests that the LineRWKV model is able to preserve important features in the images that are crucial for downstream tasks such as segmentation.

Table 11.4: Average segmentation metrics for LineRWKV reconstructed images

Metric	Value
Accuracy	0.9645
F1-score	0.8394
IoU	0.7490
AUC	0.8674

Table 11.5: Per class segmentation metrics for LineRWKV reconstructed images

Metric	Value for class 0	Value for class 1	Value for class 2
PPV	0.6863	0.9701	0.9478
Recall	0.5974	0.9882	0.8573
F1-Score	0.6388	0.9791	0.9003
IoU	0.4693	0.9590	0.8187

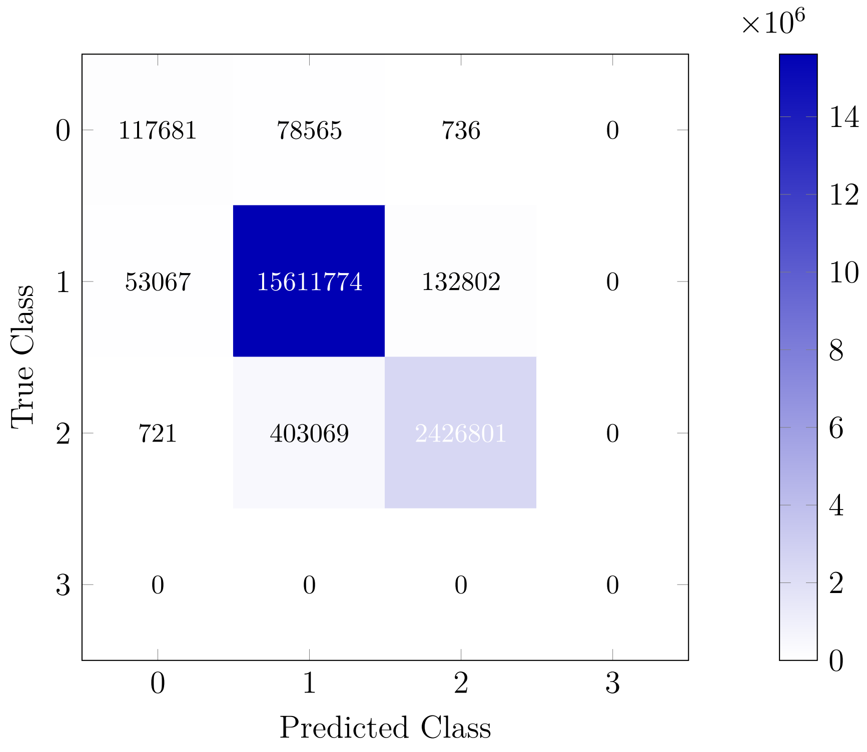


Figure 11.7: Confusion matrix heatmap for LineRWKV reconstructed images vs ground truth

11.4.2 Residual Convolutional 3D Autoencoder

The reconstruction results for RCAE3D model are shown in Table 11.6. The histogram of reconstruction errors for RCAE3D model is shown in Figure 11.8. The best reconstructed image as well as its error map are shown in Figure 11.9.

Table 11.6: RCAE3D evaluation metrics

Metric	Value
PSNR (dB)	38.1504
SSIM	0.9774
MSE	1.89×10^{-4}
SA (degrees)	5.138
bpppc	1.2673
CR	12.6253

High SSIM and PSNR values indicate that the reconstructed images are very similar to the original ones, both in terms of structure and pixel accuracy. A low MSE value further confirms that the average squared difference between the original and reconstructed images is minimal. The SA value indicates that the spectral similarity between the original and reconstructed images is also quite high.

At the same time, the histogram 11.8 shows that most reconstruction errors are concentrated close to zero, with only a few instances of larger errors. Most of the errors are

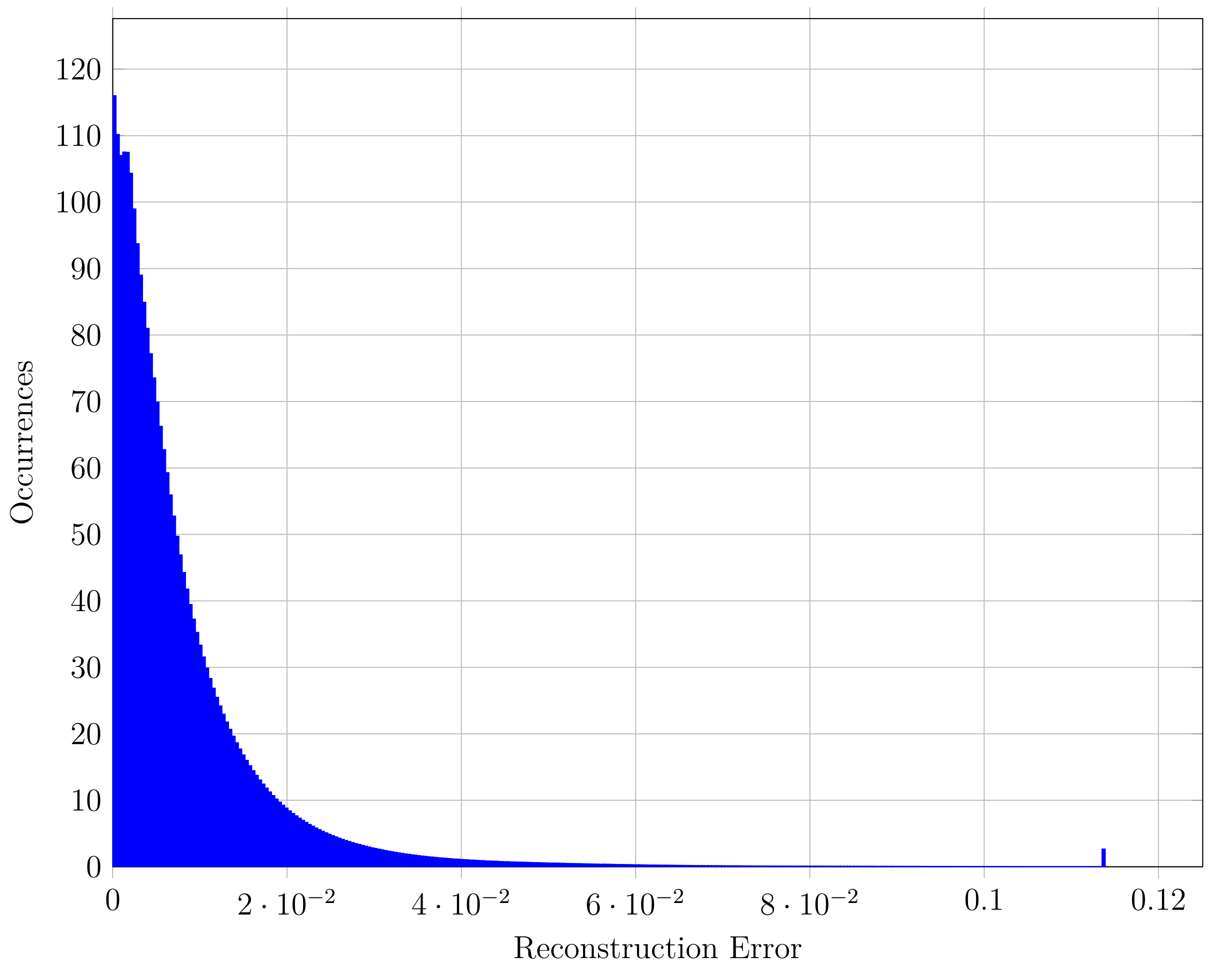
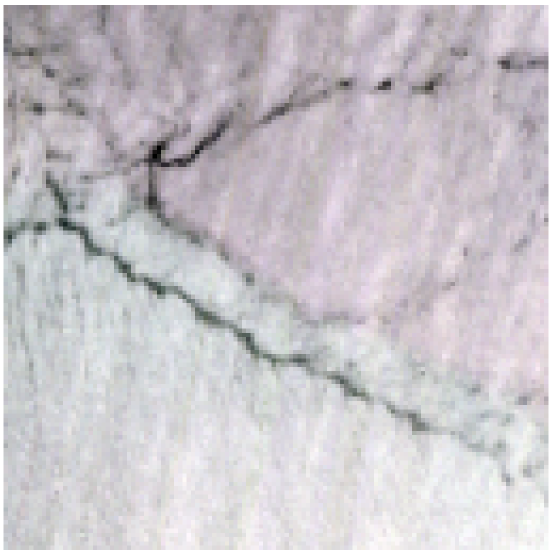
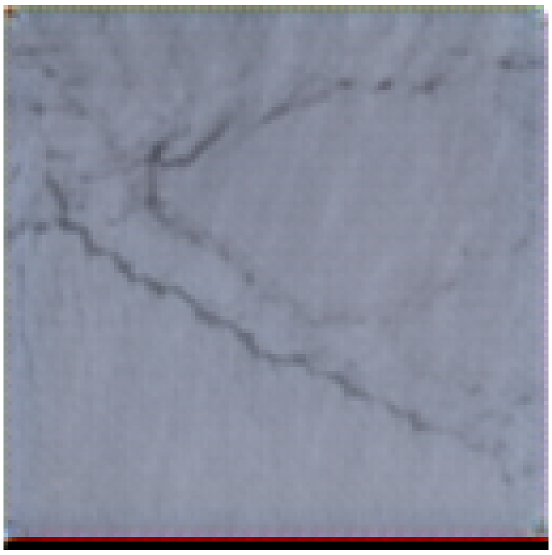
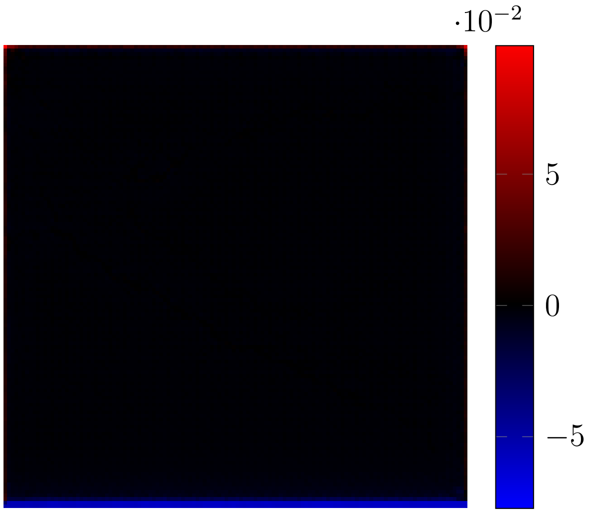


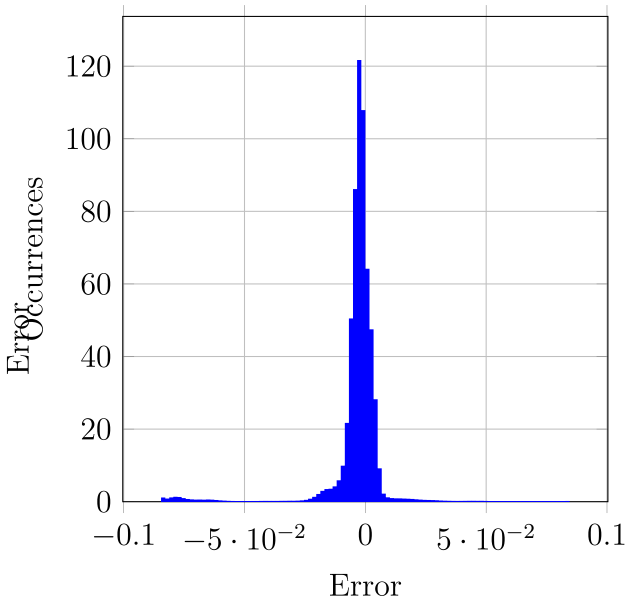
Figure 11.8: Histogram reconstruction error for RCAE3D


(a) Original image


(b) Reconstructed image

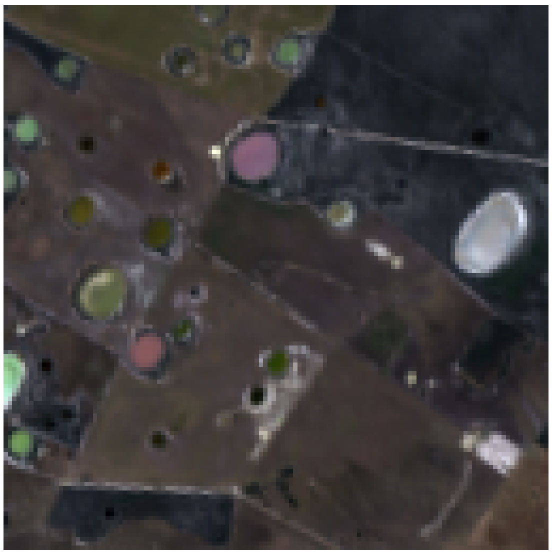


(c) Per pixel error map normalized to between -7.7278×10^{-2} and 9.8919×10^{-2}

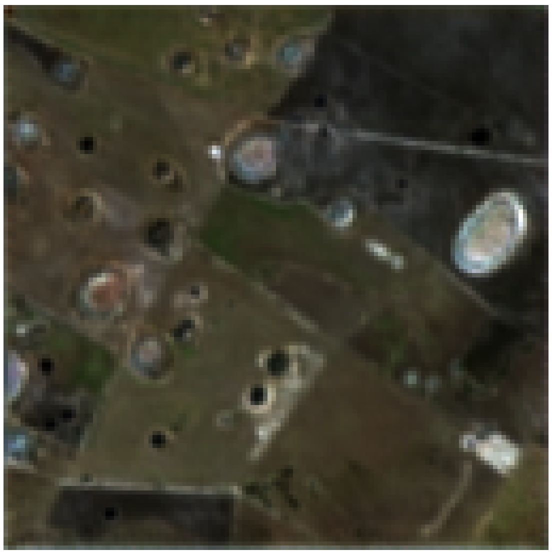


(d) Per pixel error histogram

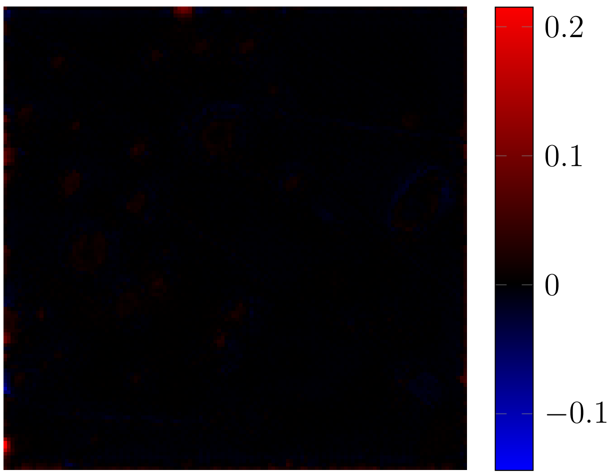
Figure 11.9: Best image reconstruction for RCAE3D model



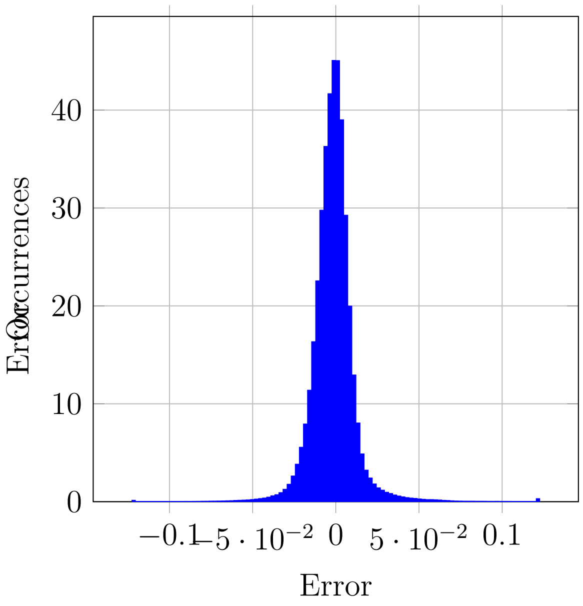
(a) Original image



(b) Reconstructed image

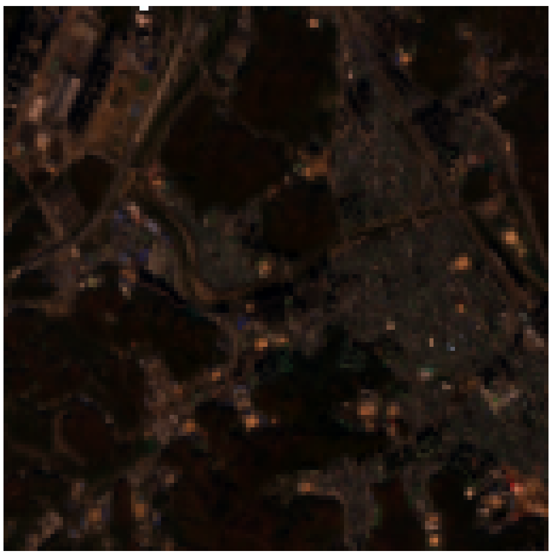


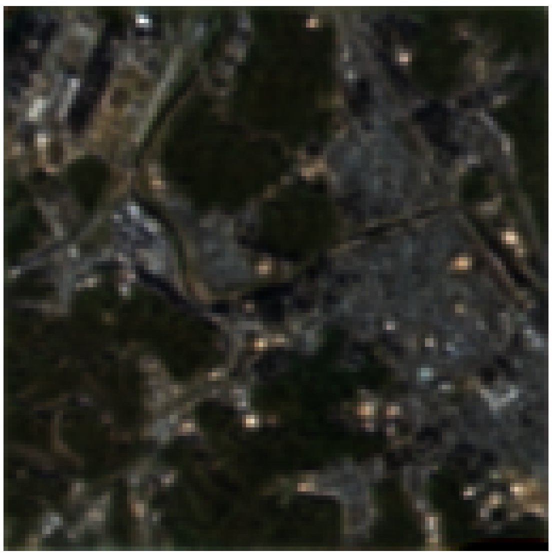
(c) Per pixel error map normalized to between -1.4386×10^{-1} and 2.1522×10^{-1}

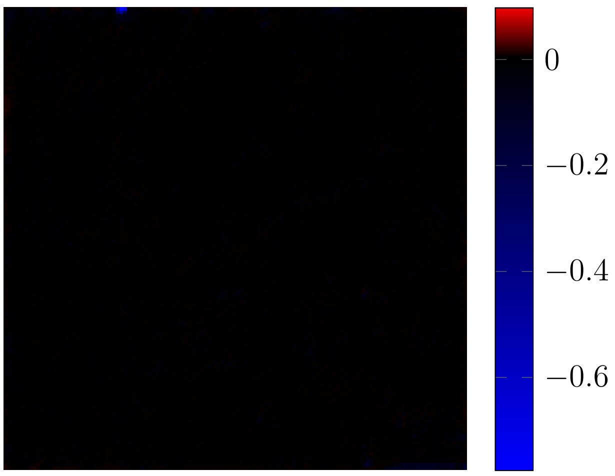


(d) Per pixel error histogram

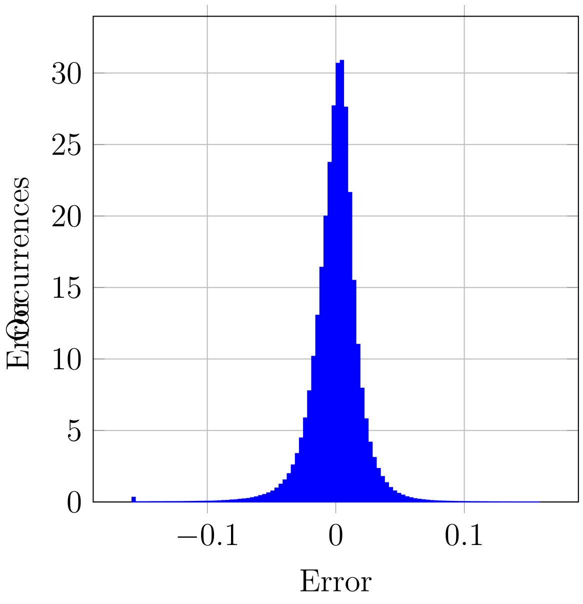
Figure 11.10: Median image reconstruction for RCAE3D model


(a) Original image


(b) Reconstructed image

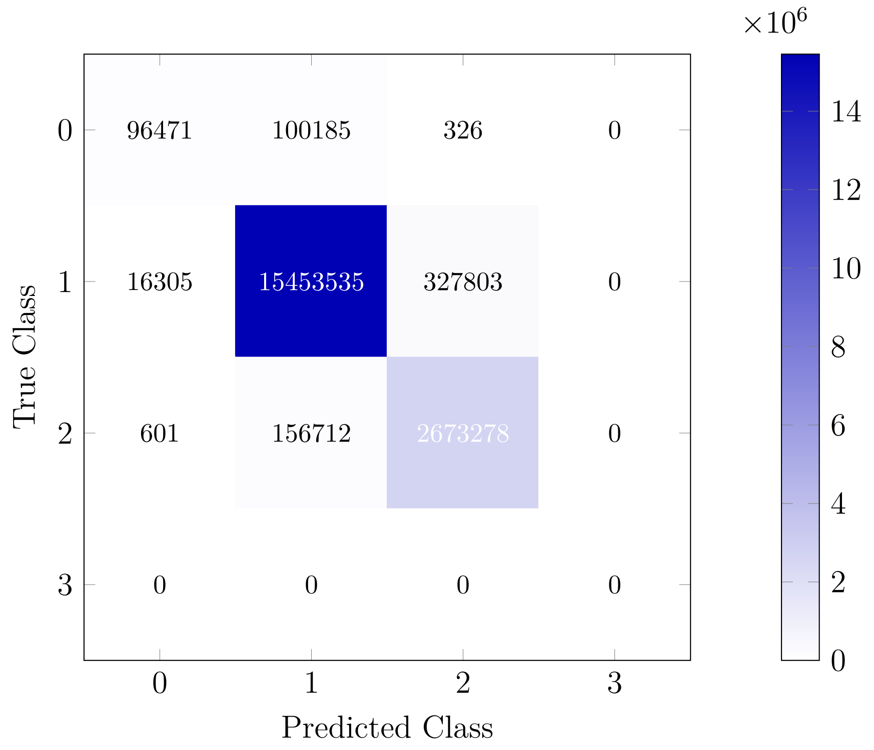


(c) Per pixel error map normalized to between -7.7572×10^{-1} and 9.7556×10^{-2}



(d) Per pixel error histogram

Figure 11.11: Worst image reconstruction for RAE 3D model



True \ Predicted	0	1	2	3
0	96471	100185	326	0
1	16305	15453535	327803	0
2	601	156712	2673278	0
3	0	0	0	0

Figure 11.12: Confusion matrix heatmap for RCAE3D reconstructed images vs ground truth

below 4×10^{-2} with a small additional peak around 0.11 which can be attributed to a few outlier images such as the one shown in Figure 11.11. The best reconstructed image shown in Figure 11.9 has a very low error, as can be seen in the error map and the error histogram, which shows that most of the pixel errors are in the range of about $\pm 1 \times 10^{-2}$. However, there exists a row of outlier pixels with errors around 0.1, which can be clearly seen in the error map in Figure 11.9c. These errors most probably occur due to the padding strategy used in the convolutional layers of the model, which can cause some artifacts in the reconstructed images, especially near the borders. A similar phenomenon can be observed in the other reconstructed images shown in Figures 11.10 and 11.11, where the highest errors occur near the borders of the images, which further supports the hypothesis that these errors are caused by the padding strategy.

In summary, the RCAE3D model demonstrates strong performance in reconstructing images from moderate compression of 1.2673 bpppc and a compression ratio of 12.625, maintaining high fidelity to the original images across multiple evaluation metrics. The model also has some space for improvement, the validation loss is still decreasing and the model is not overfitted yet. The performance of the segmentation model on the images reconstructed by the RCAE3D model is summarized in tables 11.7 and 11.8. These metrics were calculated from the confusion matrix shown in Figure 11.12. The results indicate that the segmentation model handles data processed by the RCAE3D model exquisitely, with only a slight decrease in performance when compared to the baseline. This suggests that the RCAE3D model is quite effective at preserving spatio-spectral features that are crucial for accurate segmentation, which means that it is suitable for use in practical applications where both compression and segmentation are required.

Table 11.7: Average segmentation metrics for RCAE3D reconstructed images

Metric	Value
Accuracy	0.9680
F1-score	0.8398
IoU	0.7533
AUC	0.8659

Table 11.8: Per class segmentation metrics for RCAE3D reconstructed images

Metric	Value for class 0	Value for class 1	Value for class 2
PPV	0.8509	0.9836	0.8907
Recall	0.4897	0.9782	0.9444
F1-Score	0.6217	0.9809	0.9168
IoU	0.4510	0.9626	0.8463

11.4.3 Residual Convolutional 2D1D Autoencoder

The reconstruction metrics for the RCAE2D1D model are shown in Table 11.9. The histogram of reconstruction errors for the RCAE2D1D model is shown in Figure 11.13. The best reconstructed image, as well as its error map are shown in Figure 11.14.

Based on the obtained metrics, it can be concluded that the model is performing well, but not as well as the RCAE3D model. The SSIM and PSNR values are slightly lower than those of the RCAE3D model, indicating that the reconstructed images are less similar to the original ones in terms of structure and pixel accuracy. The MSE value is higher, which confirms that the average squared difference between the original and reconstructed images is larger. The SA value is at 6.13 degrees, indicating a moderate level of spectral similarity between the original and reconstructed images. The compression ratio of this model is 25.09.

Table 11.9: RCAE2D1D evaluation metrics

Metric	Value
PSNR (dB)	34.24
SSIM	0.9619
MSE	0.000487
SA (degrees)	6.13
bpppc	0.6337
CR	25.09

In summary, the RCAE2D1D model demonstrates decent performance with a slightly higher compression ratio of 25.09 and a bitrate of 0.6337 bpppc, but with a trade-off in reconstruction quality compared to the RCAE3D model. The histogram 11.13 shows that most errors are concentrated close to zero, with a small additional peak around 0.13 which can be attributed to the same outlier images as in the RCAE3D model, such as the one shown in Figure 11.16. The histograms and error maps for the best, median, and worst reconstructed images shown in Figures 11.14, 11.15, and 11.16 respectively, show a similar trend to the RCAE3D model, where the highest errors accumulate near the borders of the images, which can be attributed to the padding strategy used in the convolutional layers of the model. An additional observation, particularly in the median reconstructed image shown in Figure 11.15, is that there are some noticeable errors with sudden changes in pixel values, which can be attributed to the fact that the RCAE2D1D model processes the spectral and spatial dimensions separately, which can lead to some loss of information and less accurate reconstructions compared to the RCAE3D model, which processes all dimensions together. Yet another observation in the error histograms is the skewness of the distribution of errors towards negative values, which can be a result of the model picking up some bias in the training data. The model is not overfitted; the SSIM and

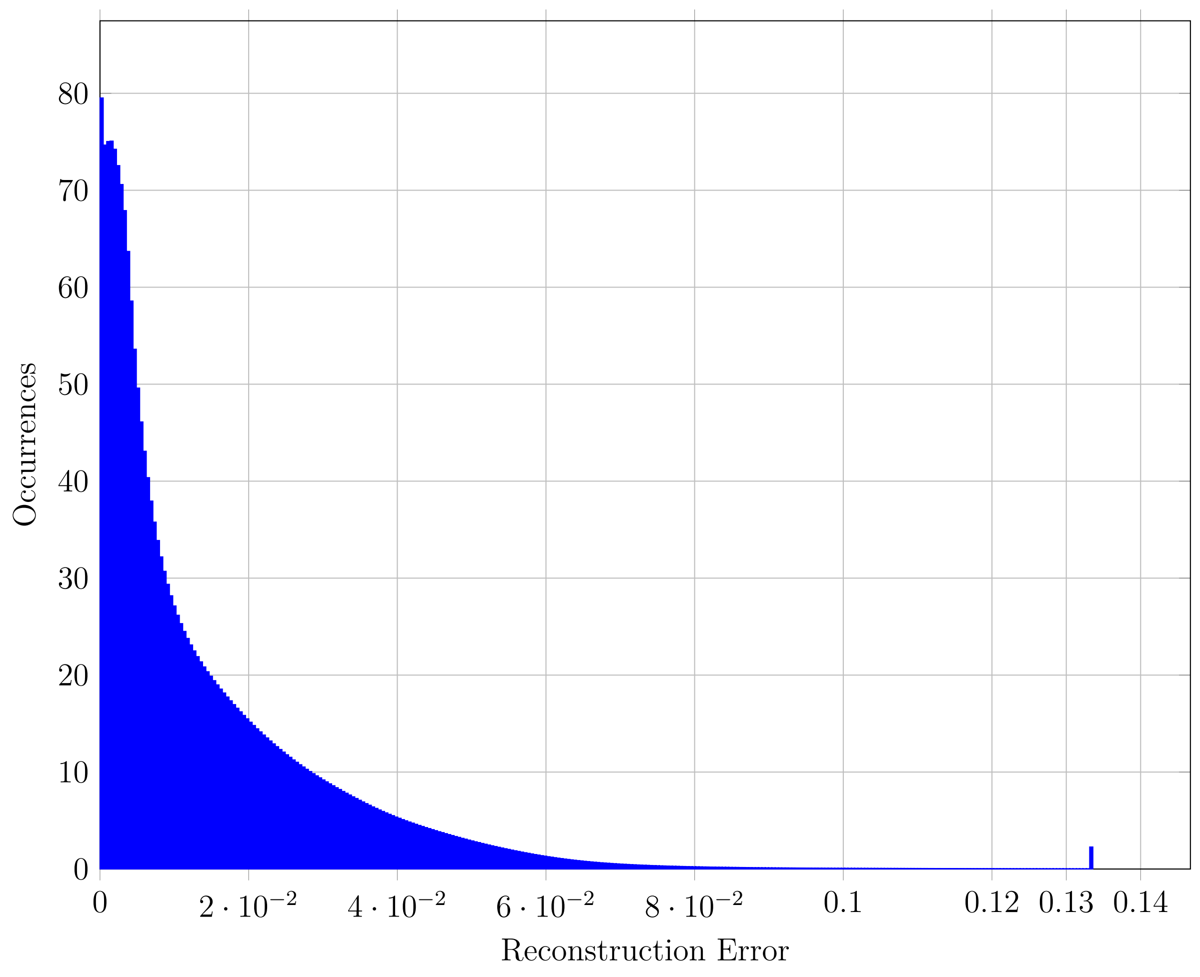
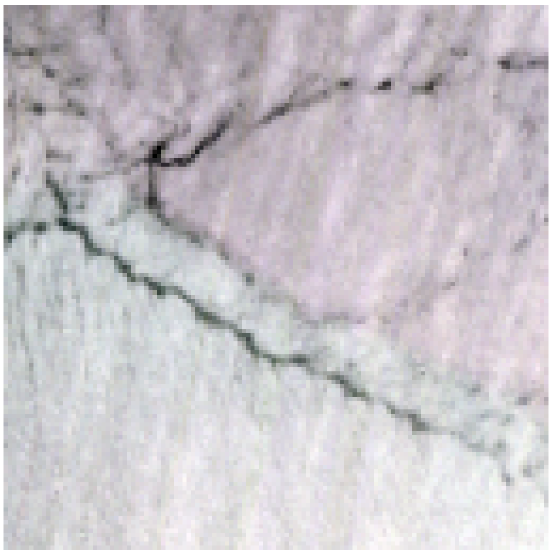
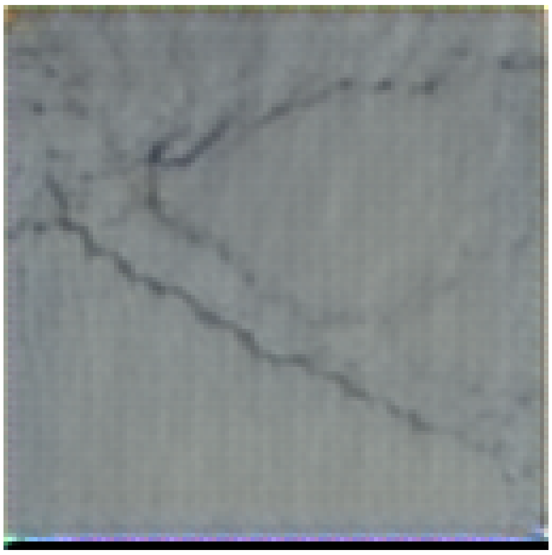
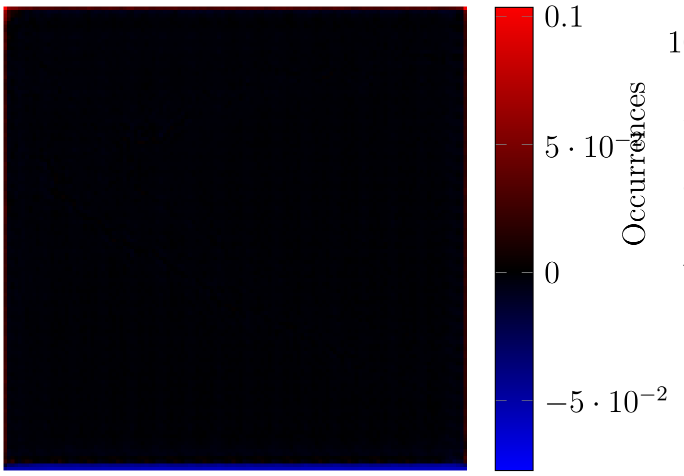


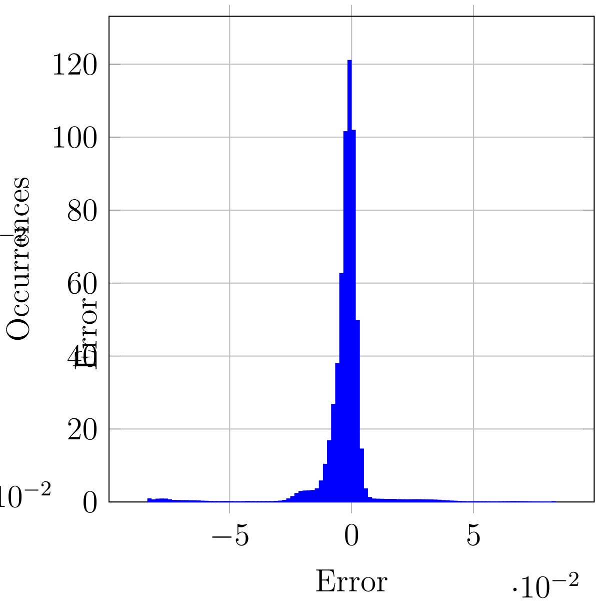
Figure 11.13: Histogram reconstruction error for Residual Convolutional 2D1D Autoencoder


(a) Original image


(b) Reconstructed image

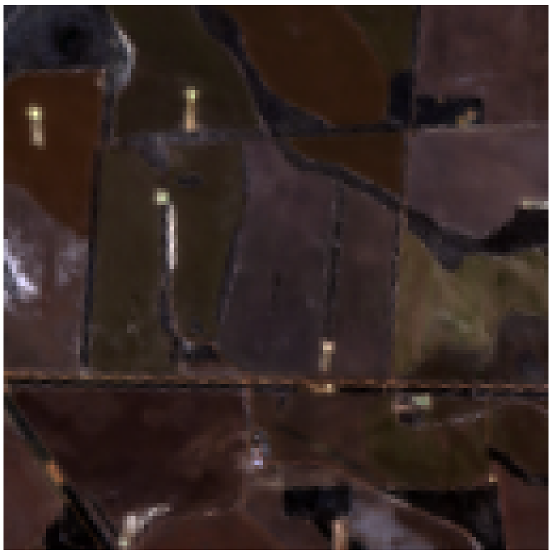


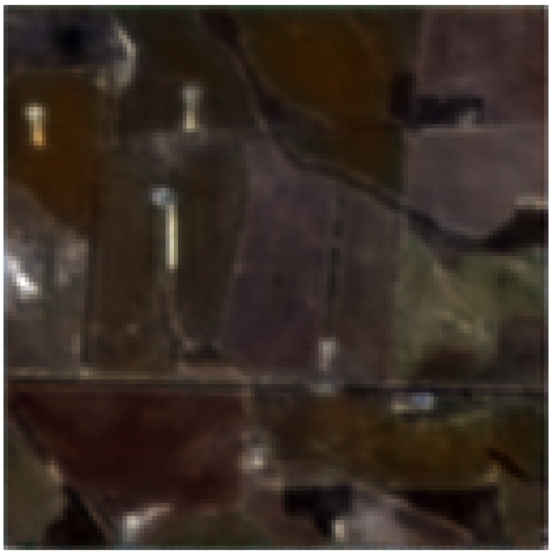
(c) Per pixel error map normalized to between -7.7258×10^{-2} and 1.0350×10^{-1}

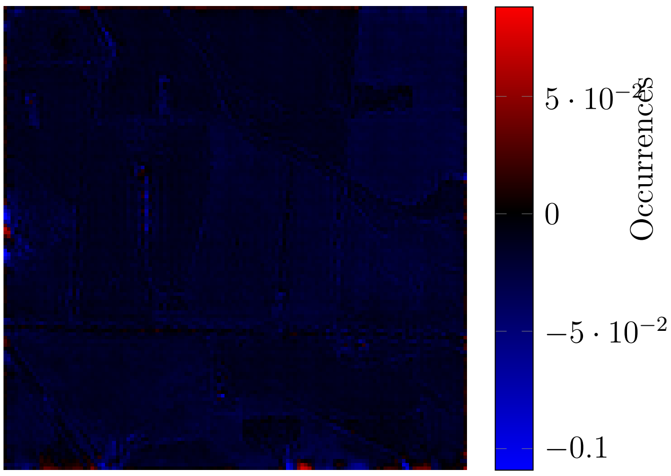


(d) Per pixel error histogram

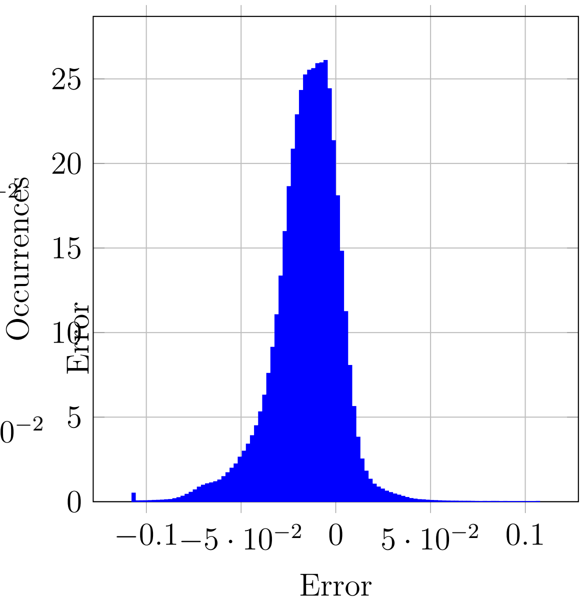
Figure 11.14: Best image reconstruction for RCAE2D1D model


(a) Original image


(b) Reconstructed image

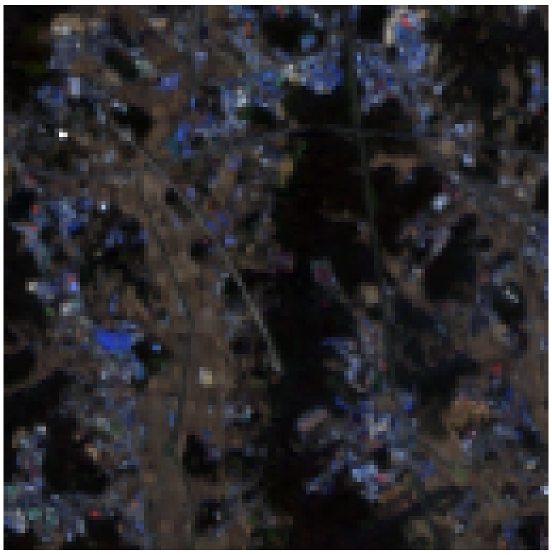


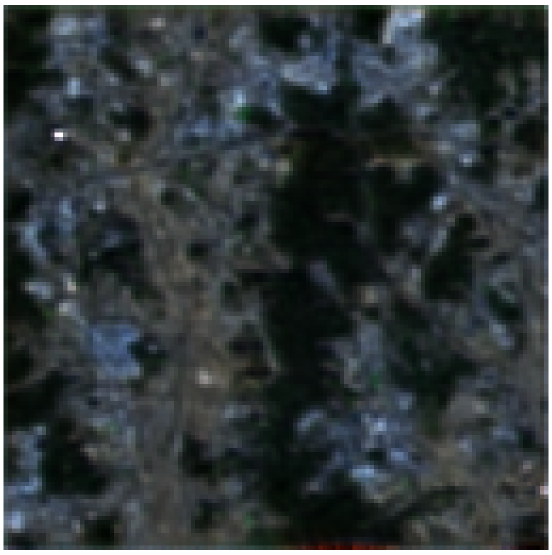
(c) Per pixel error map normalized to between -1.0914×10^{-1} and 8.8060×10^{-2}

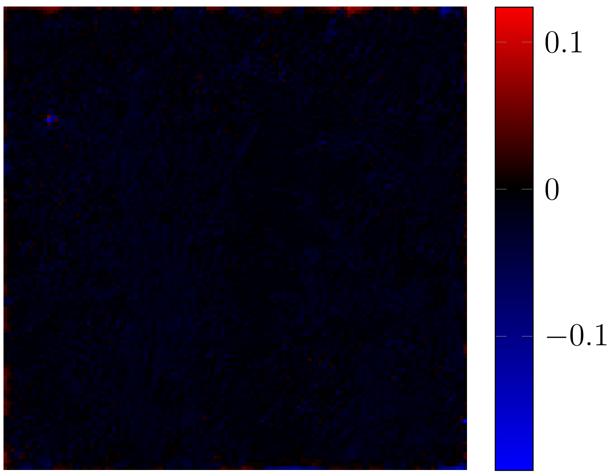


(d) Per pixel error histogram

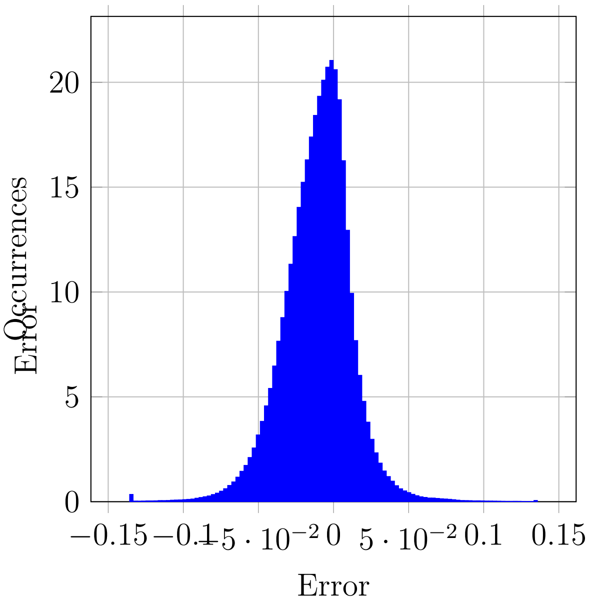
Figure 11.15: Mid error image reconstruction for RCAE2D1D model


(a) Original image


(b) Reconstructed image



(c) Per pixel error map normalized to between -1.9113×10^{-1} and 1.2371×10^{-1}



(d) Per pixel error histogram

Figure 11.16: Worst image reconstruction for RCAE2D1D model

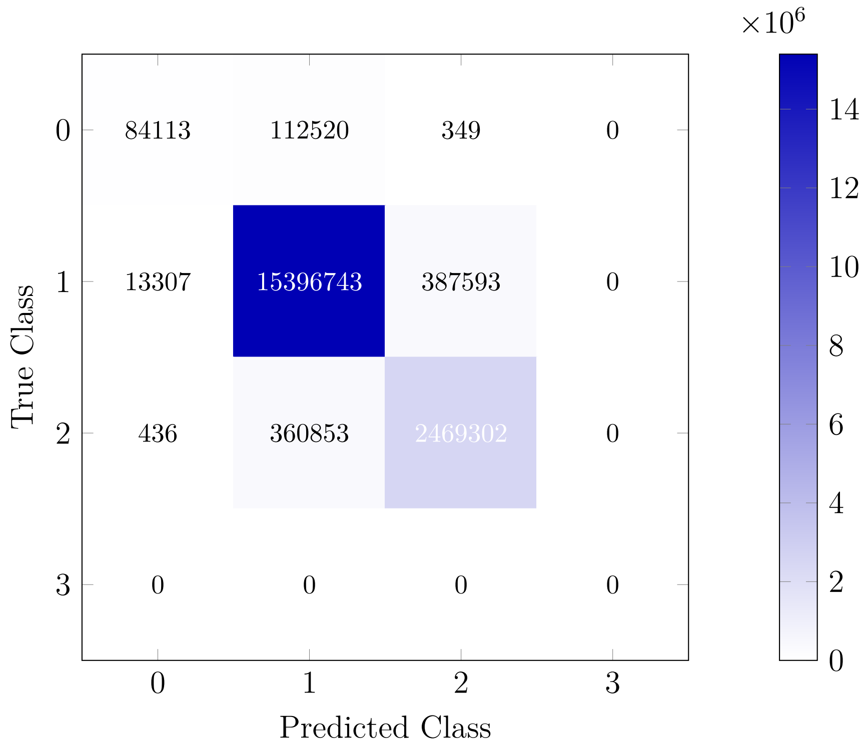


Figure 11.17: Confusion matrix heatmap for RCAE2D1D reconstructed images vs ground truth

PSNR values are still on the rise, and MSE as well as SA values are still decreasing, indicating that there is still some room for improvement in the model's performance.

Table 11.10: Average segmentation metrics for RCAE2D1D reconstructed images

Metric	Value
Accuracy	0.9535
F1-score	0.8037
IoU	0.7042
AUC	0.8610

The results of the segmentation model on the images reconstructed by the RCAE2D1D model are summarized in tables 11.10 and 11.11. These metrics were calculated from the confusion matrix shown in Figure 11.17. The results indicate that the segmentation model handles data processed by the RCAE2D1D model reasonably well, although there is a more noticeable decrease in performance compared to the baseline than with the RCAE3D model. This suggests that while the RCAE2D1D model is effective at preserving essential image features for segmentation, it may not retain them as well as other models. This, however, doesn't disqualify it from practical applications as the results are still acceptable for many use cases where both compression and segmentation are required.

Table 11.11: Per class segmentation metrics for RCAE2D1D reconstructed images

Metric	Value for class 0	Value for class 1	Value for class 2
PPV	0.8596	0.9702	0.8642
Recall	0.4270	0.9746	0.8724
F1-Score	0.5706	0.9724	0.8683
IoU	0.3992	0.9463	0.7672

11.4.4 RCGDNAE

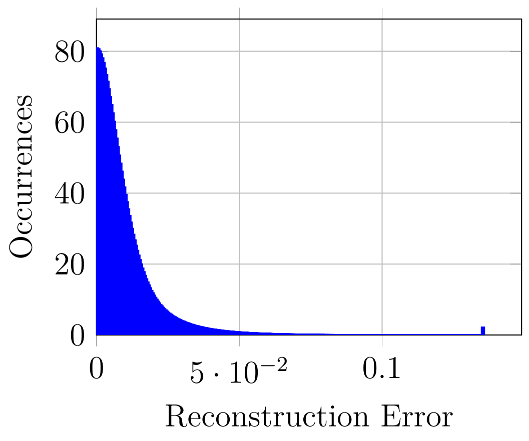
The RCGDNAE model was evaluated in three different versions, each trained with a different λ parameter. The models were trained with λ values of 0, 0.001, and 0.01. These models are called the high, medium, and low bitrate models, respectively. The reconstruction metrics for RCGDNAE model are shown in Table 11.12. The histogram of reconstruction errors for RCGDNAE model is shown in Figure 11.18 The error map for the best reconstructed image is shown in Figure 11.19c.

Table 11.12: RCGDNAE evaluation metrics

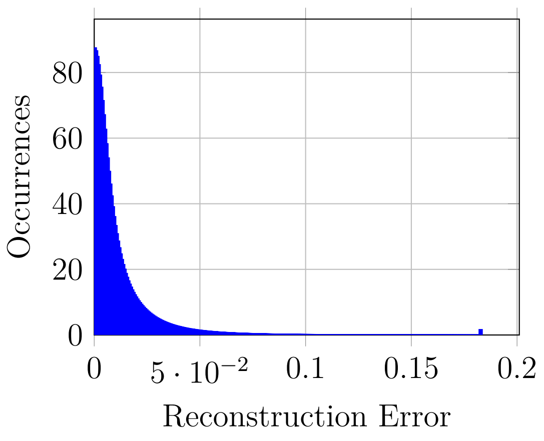
Metric	Value (High)	Value (Medium)	Value (Low)
PSNR (dB)	38.01	36.68	37.74
SSIM	0.889	0.866	0.896
MSE	0.000376	0.000469	0.000285
SA (degrees)	7.47	9.27	8.89
bpppc	0.015	0.0218	$2.58 * 10^{-5}$
CR	1,036	129,799	538,667

The model can achieve extremely high compression ratios, up to 538,667:1 for the low bitrate model. This, however, comes at a high cost to reconstruction quality. The extremely low bpppc values initially suggested that there might be an error in the implementation, such as the model being overtrained or data leakage between training and testing datasets. This is why the model was evaluated using the hard split of the dataset to ensure that no data leakage occurred. After careful review of the implementation and evaluation process, it was confirmed that the results are accurate and valid. The high compression ratios achieved by the RCGDNAE model are a result of its architecture, especially the use of the entropy coder, which allows for efficient encoding of the latent representations.

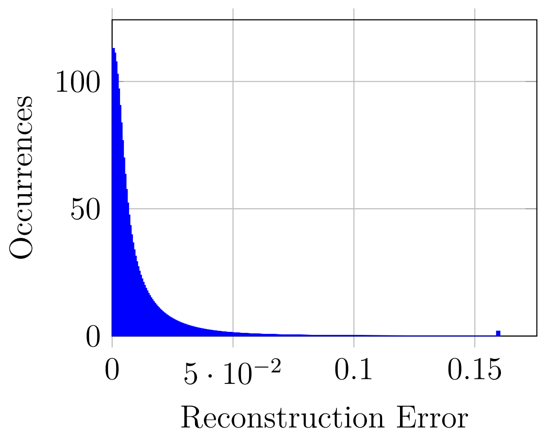
However, the trade-off between compression ratio and reconstruction quality is certainly evident in the results. All of the reconstructed images show significant loss of detail and introduction of artifacts, especially at lower bitrates. Upon analysis of the error histograms in Figure 11.18 it is clearly visible that even though the images visually appear to be of very low quality, the majority of errors are actually quite small, with a long tail of larger errors. This is further reinforced by the distribution of errors in histograms in



(a) High bitrate

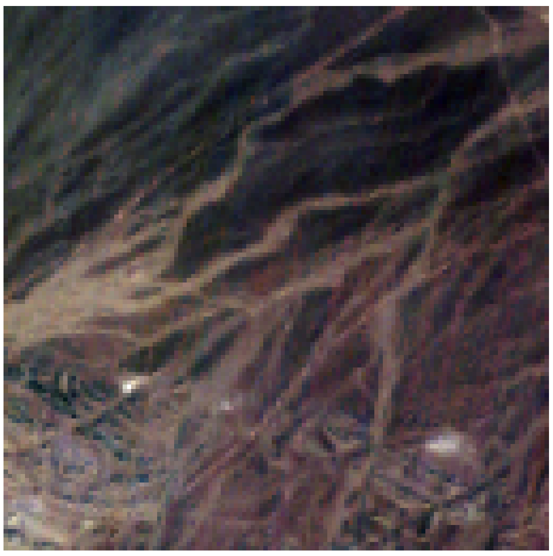


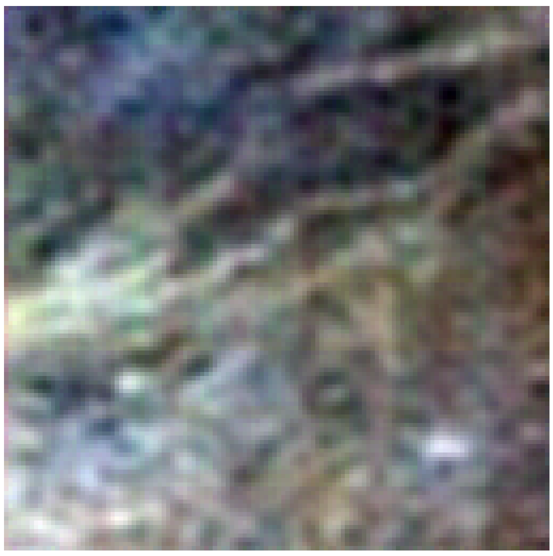
(b) Medium bitrate

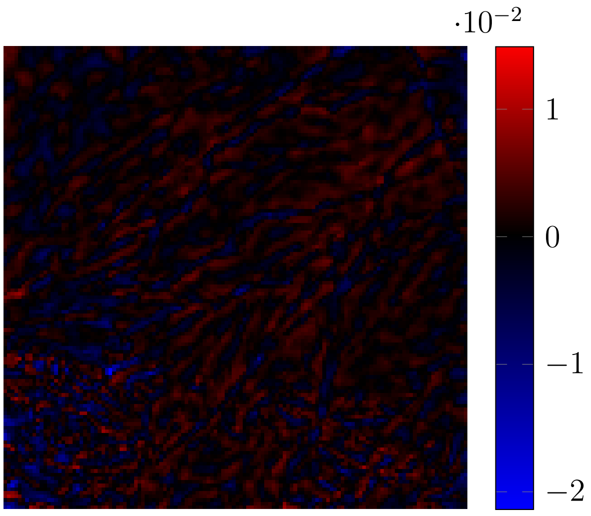


(c) Low bitrate

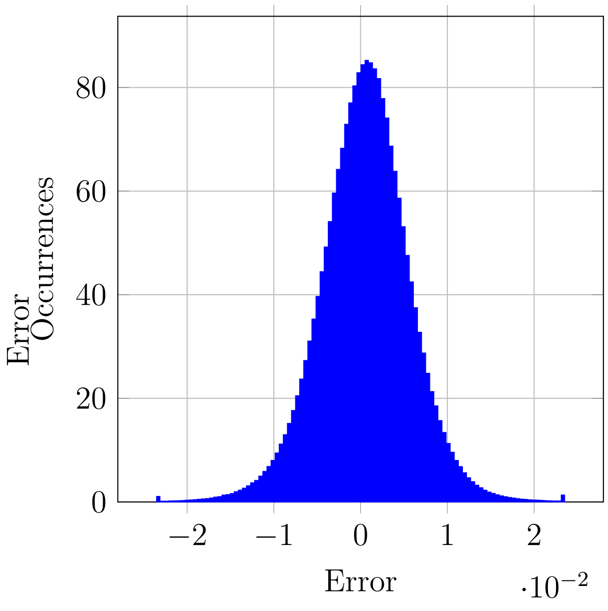
Figure 11.18: Mean Aboslute Error histogram for RCGDNAE model per image patch


(a) Original image


(b) Reconstructed image

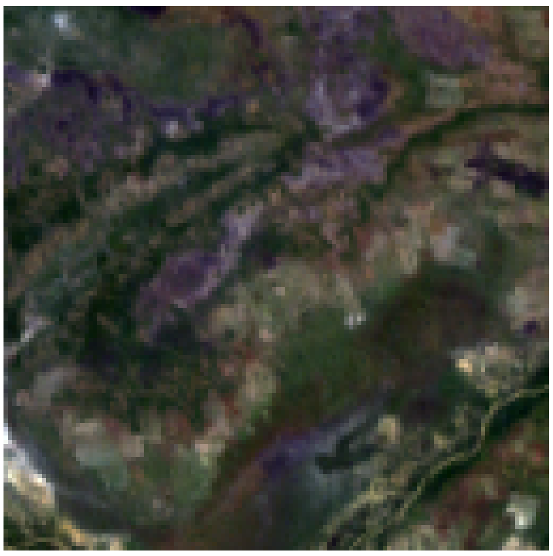


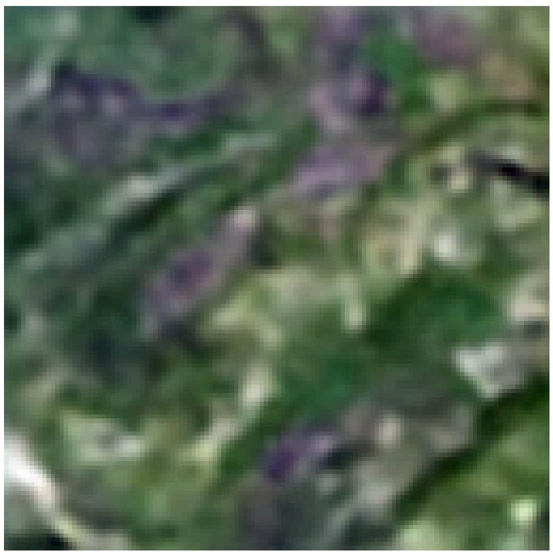
(c) Per pixel error map normalized to between -2.1382×10^{-2} and 1.4893×10^{-2}

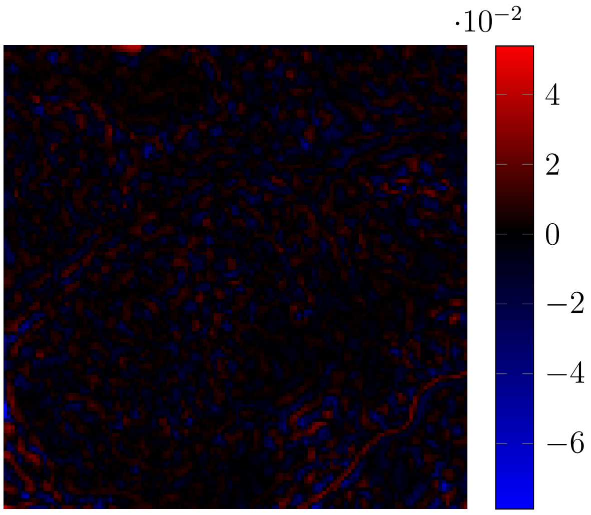


(d) Per pixel error histogram

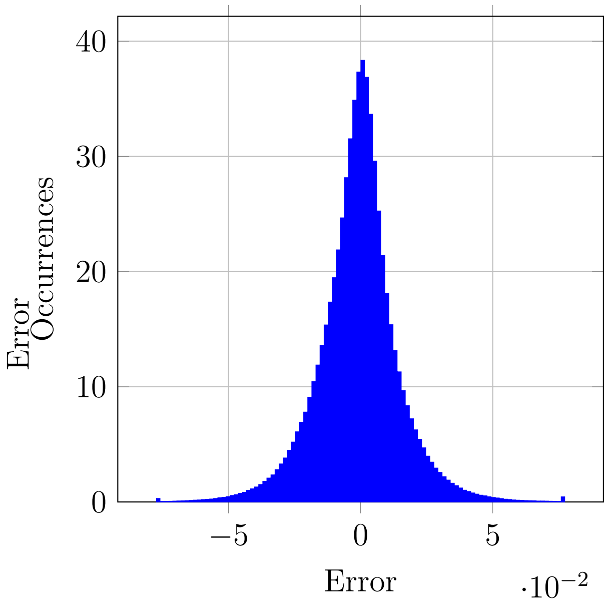
Figure 11.19: Best image reconstruction for RCGDNAE model (low bitrate)


(a) Original image


(b) Reconstructed image



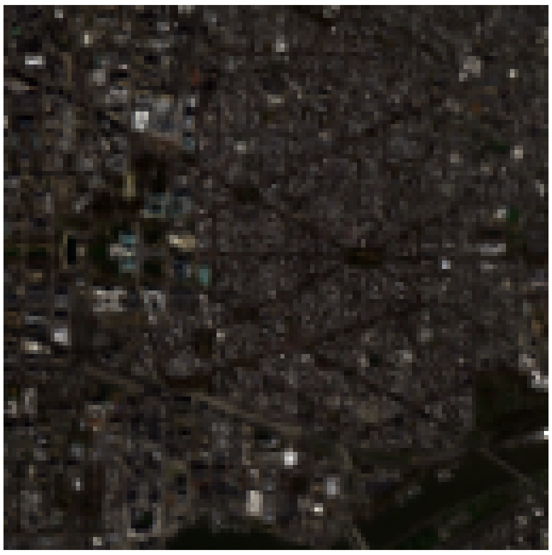
(c) Per pixel error map normalized to between -7.8742×10^{-2} and 5.3919×10^{-2}

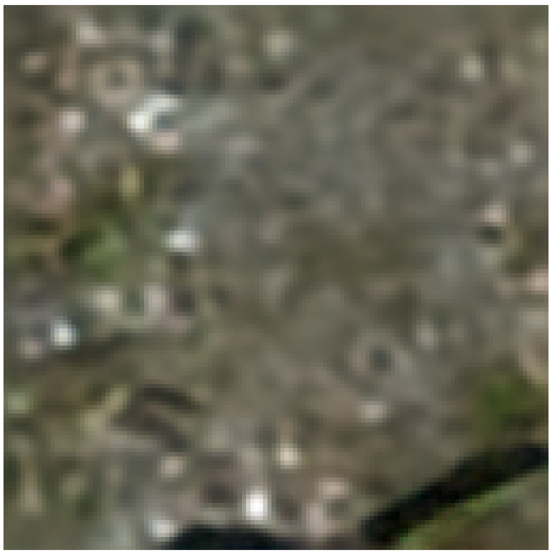


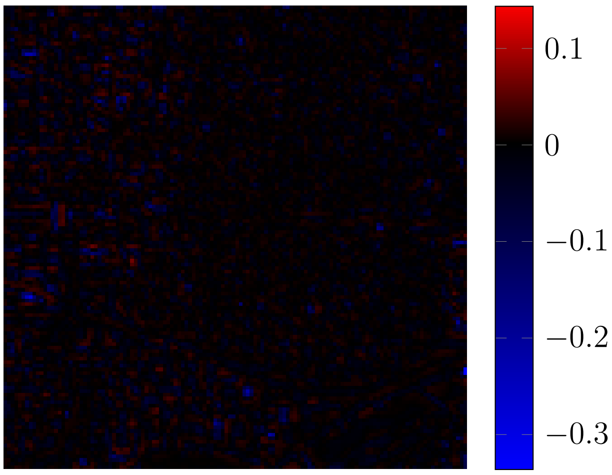
(d) Per pixel error histogram

Figure 11.20: Median image reconstruction for RCGDNAE model (low bitrate)

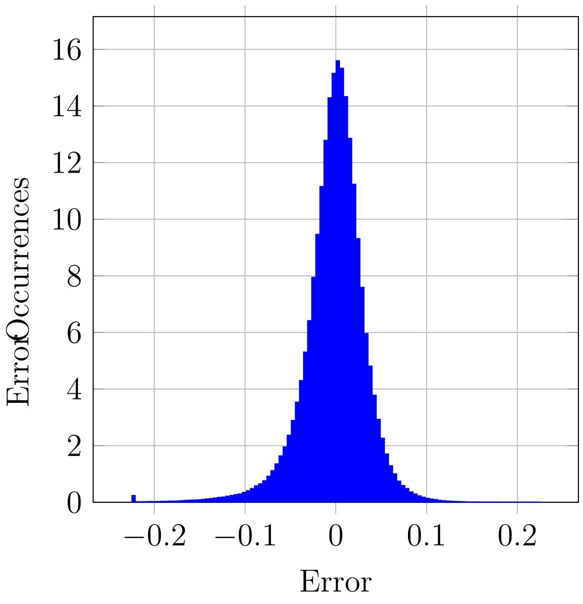
Figure 11.21: Worst image reconstruction for RCGDNAE model (low bitrate)


(a) Original image

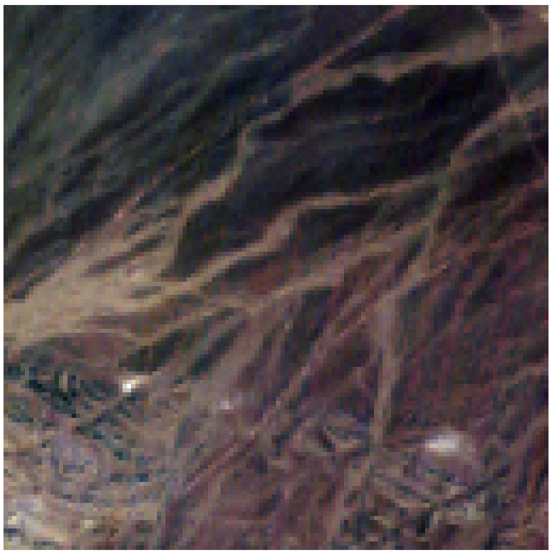

(b) Reconstructed image

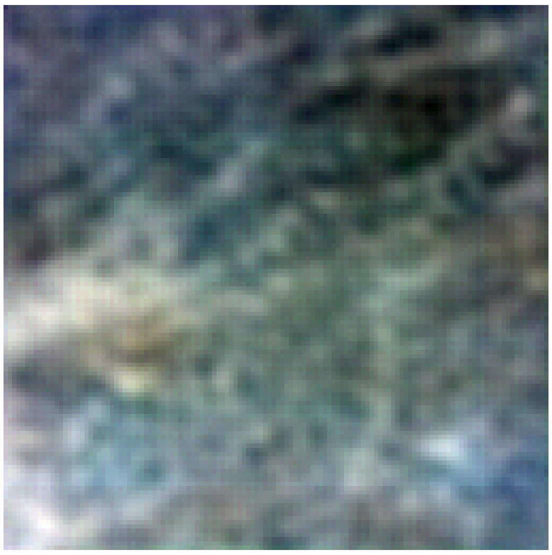


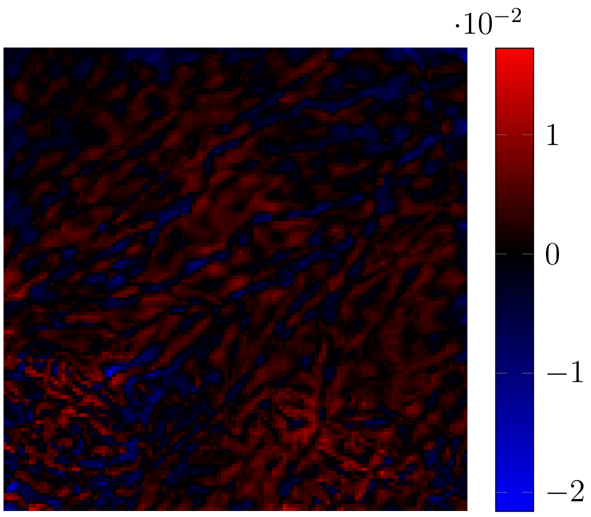
(c) Per pixel error map normalized to between -3.3618×10^{-1} and 1.4364×10^{-1}



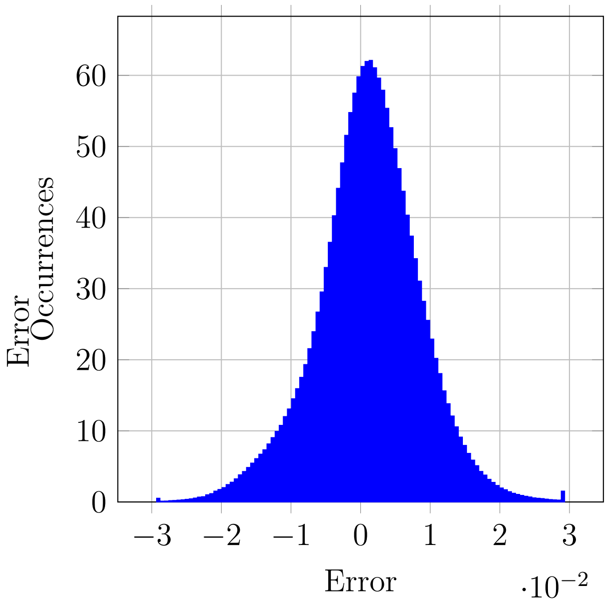
(d) Per pixel error histogram


(a) Original image


(b) Reconstructed image

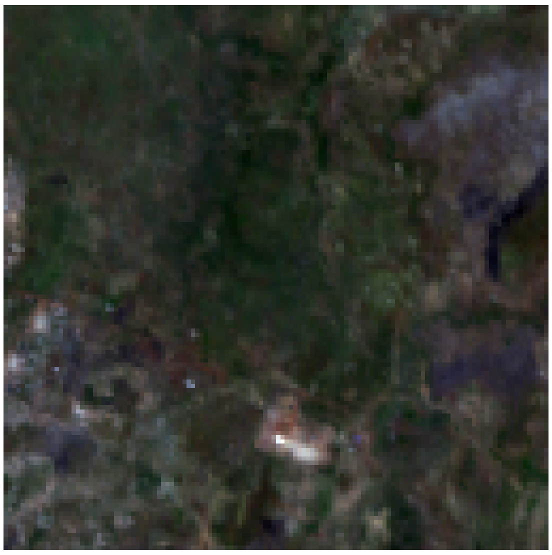


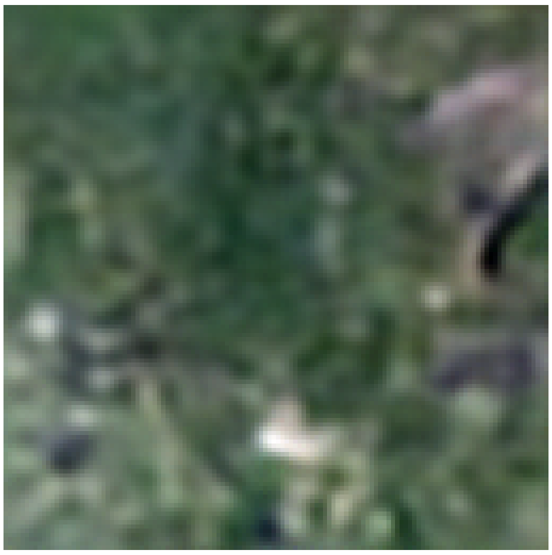
(c) Per pixel error map normalized to between -2.1598×10^{-2} and 1.7263×10^{-2}

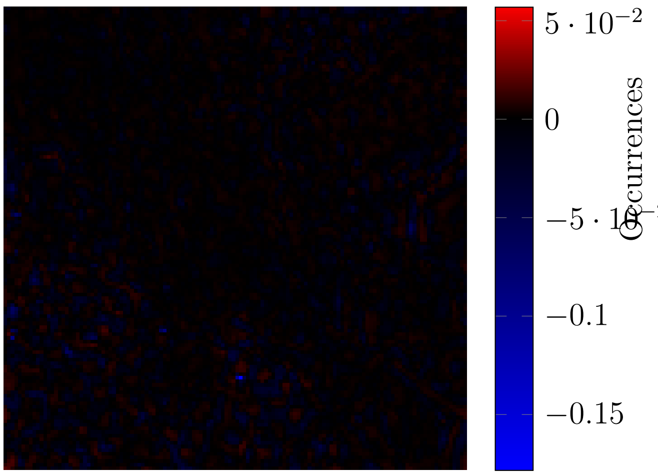


(d) Per pixel error histogram

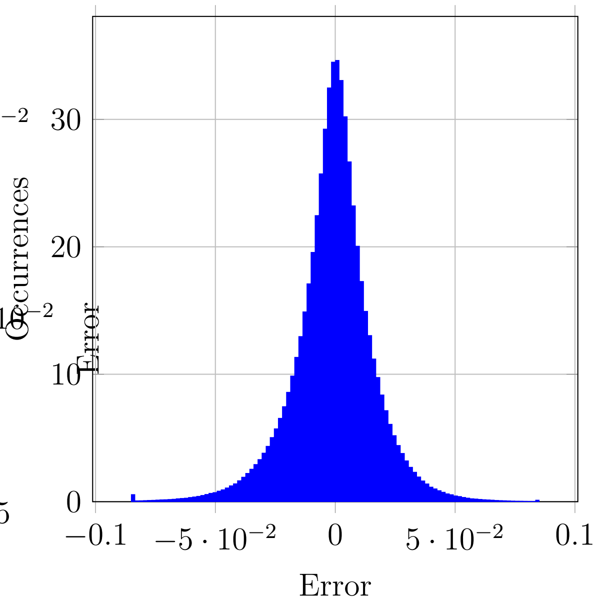
Figure 11.22: Best image reconstruction for RCGDNAE model (medium bitrate)


(a) Original image


(b) Reconstructed image

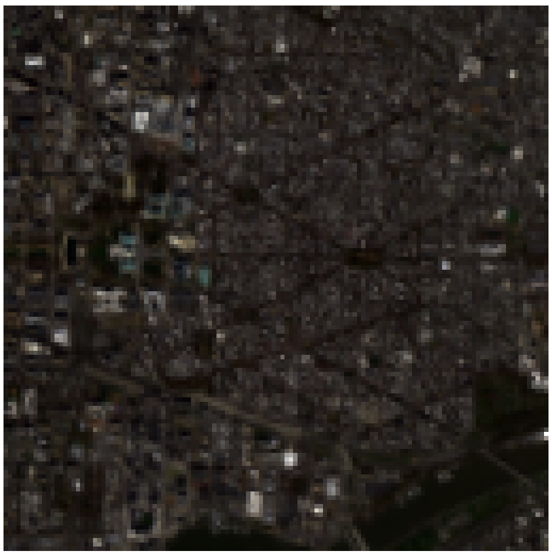


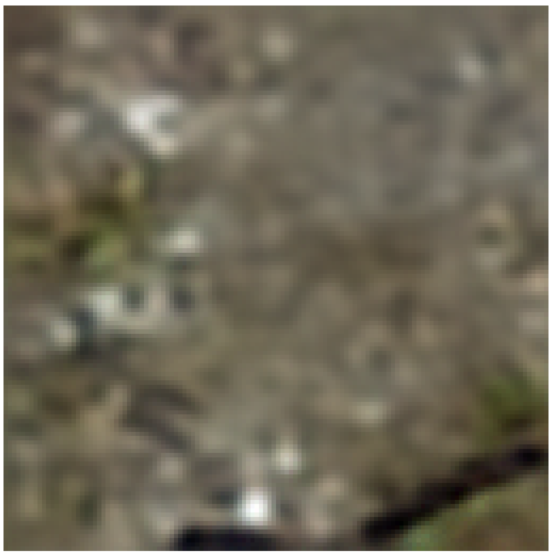
(c) Per pixel error map normalized to between -1.7838×10^{-1} and 5.6898×10^{-2}

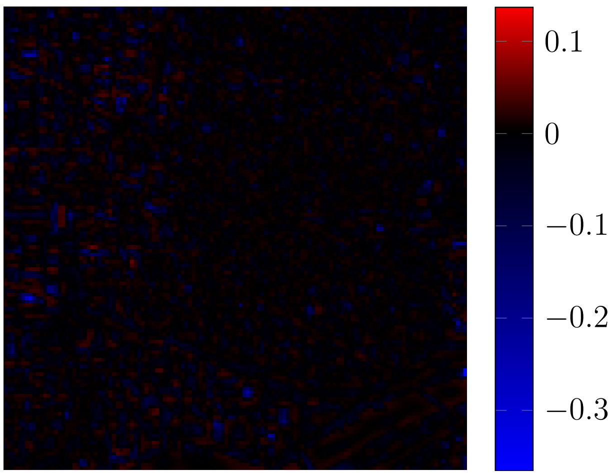


(d) Per pixel error histogram

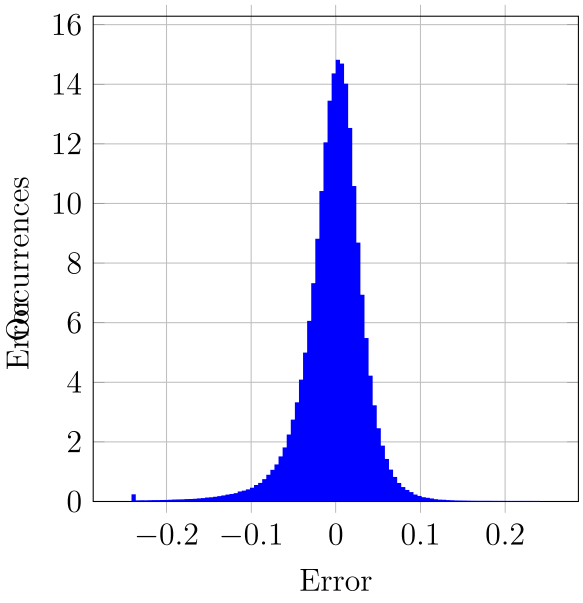
Figure 11.23: Median image reconstruction for RCGDNAE model (medium bitrate)


(a) Original image


(b) Reconstructed image

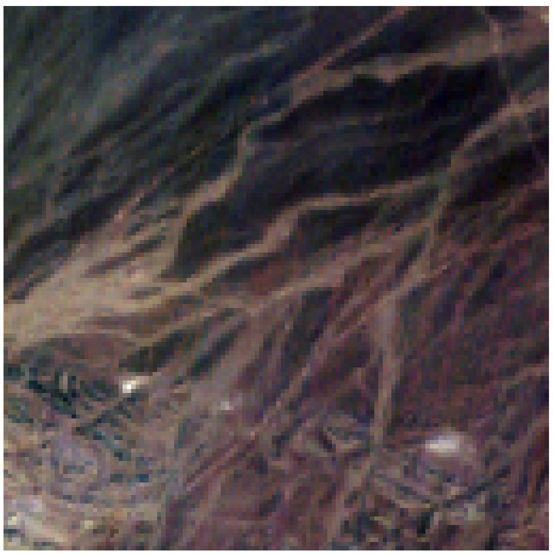


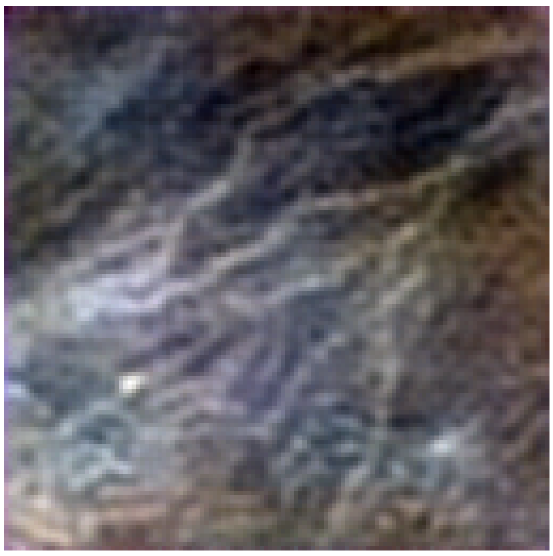
(c) Per pixel error map normalized to between -3.6513×10^{-1} and 1.3683×10^{-1}

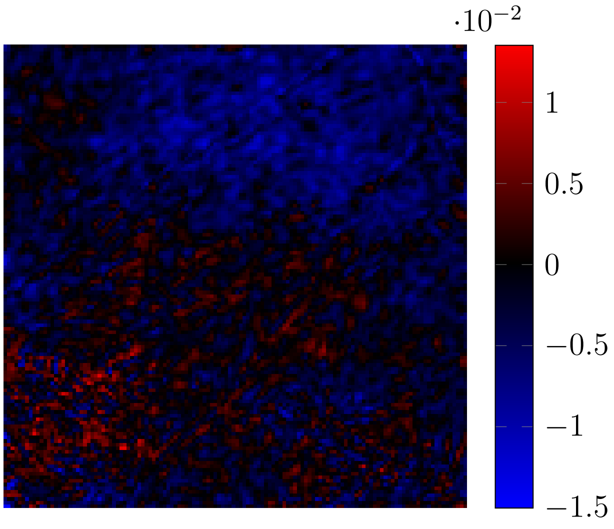


(d) Per pixel error histogram

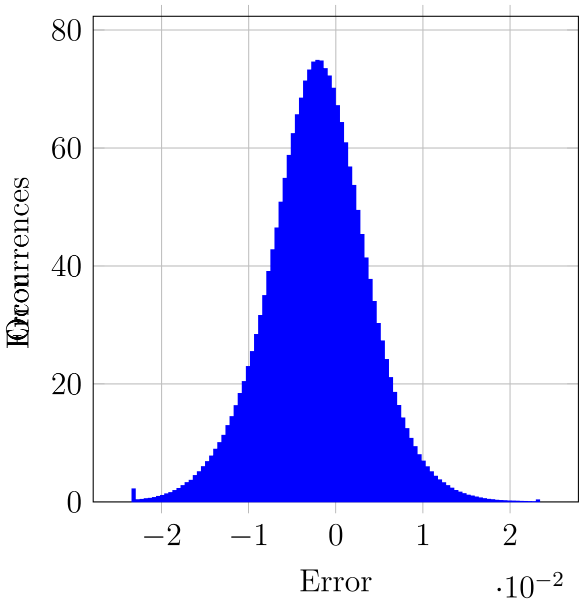
Figure 11.24: Worst image reconstruction for RCGDNAE model (medium bitrate)


(a) Original image


(b) Reconstructed image

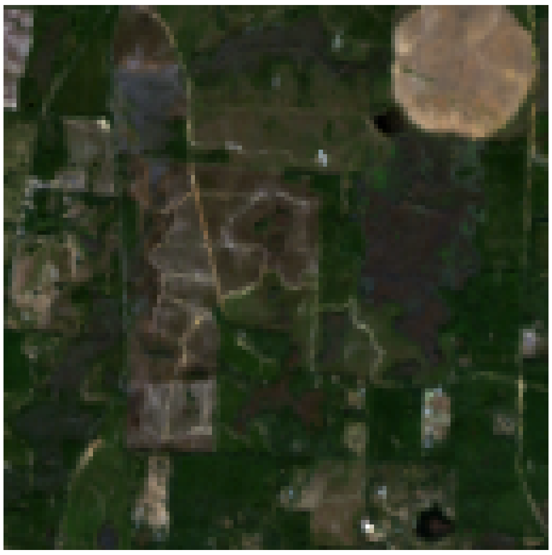


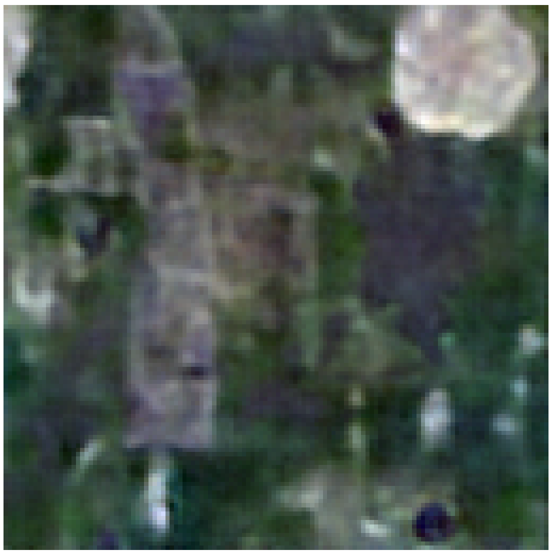
(c) Per pixel error map normalized to between -1.5011×10^{-2} and 1.3521×10^{-2}

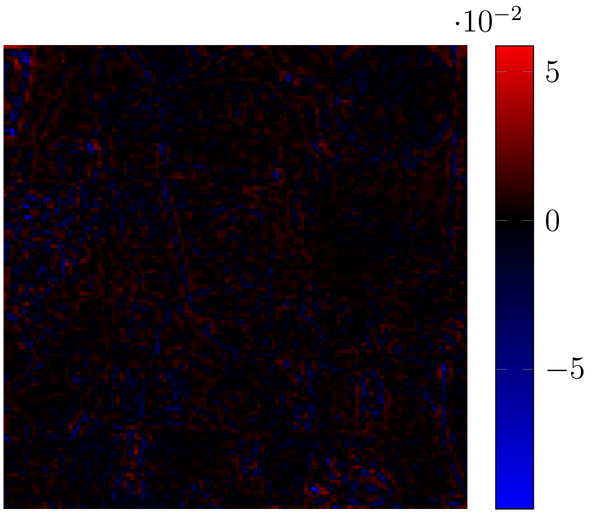


(d) Per pixel error histogram

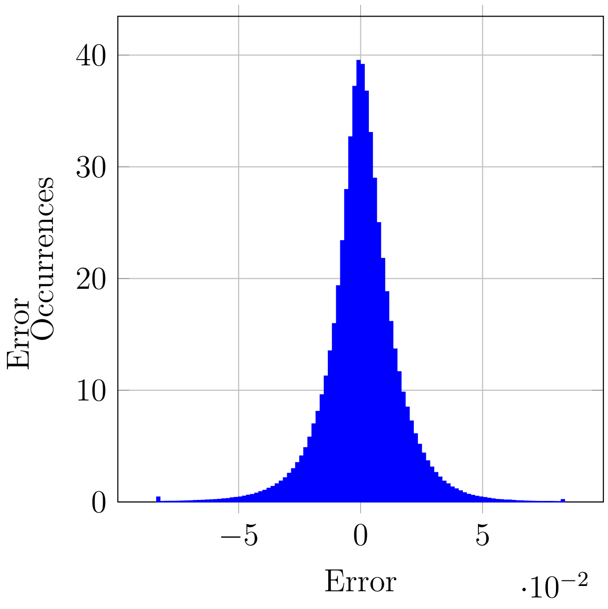
Figure 11.25: Best image reconstruction for RCGDNAE model (high bitrate)


(a) Original image


(b) Reconstructed image

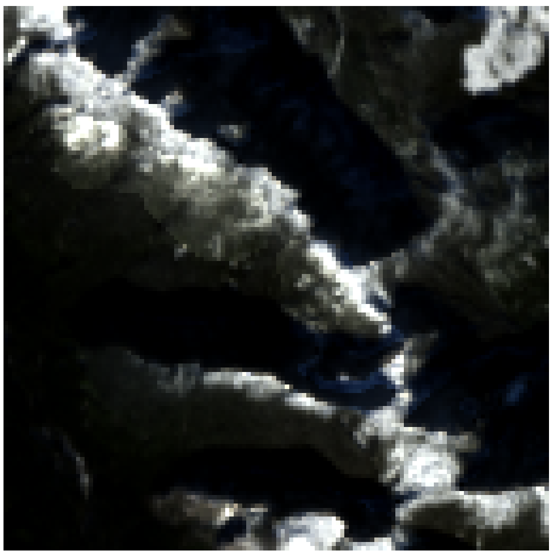


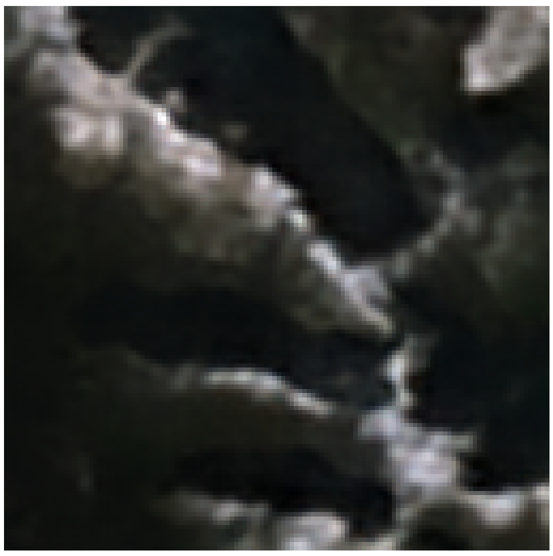
(c) Per pixel error map normalized to between -9.6863×10^{-2} and 5.8626×10^{-2}

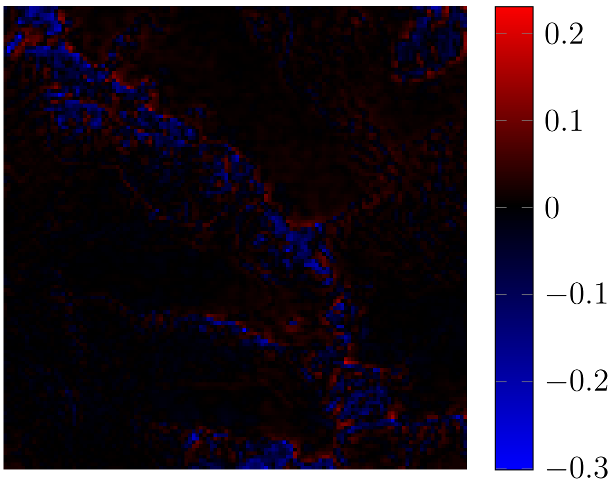


(d) Per pixel error histogram

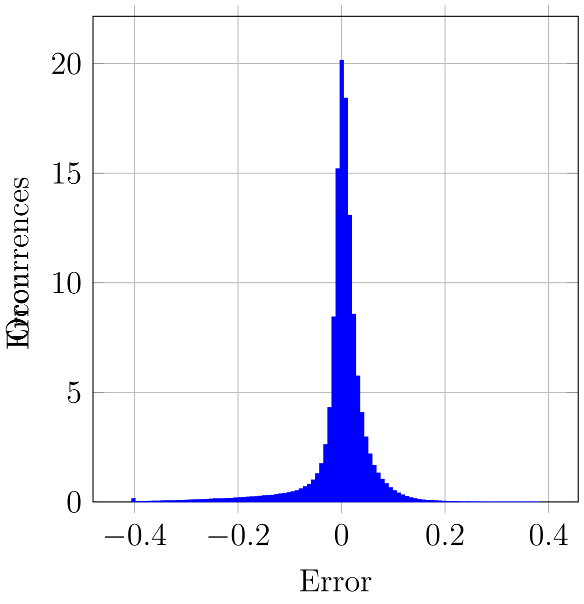
Figure 11.26: Median image reconstruction for RCGDNAE model (high bitrate)


(a) Original image


(b) Reconstructed image



(c) Per pixel error map normalized to between -3.0125×10^{-1} and 2.3062×10^{-1}



(d) Per pixel error histogram

Figure 11.27: Worst image reconstruction for RCGDNAE model (high bitrate)

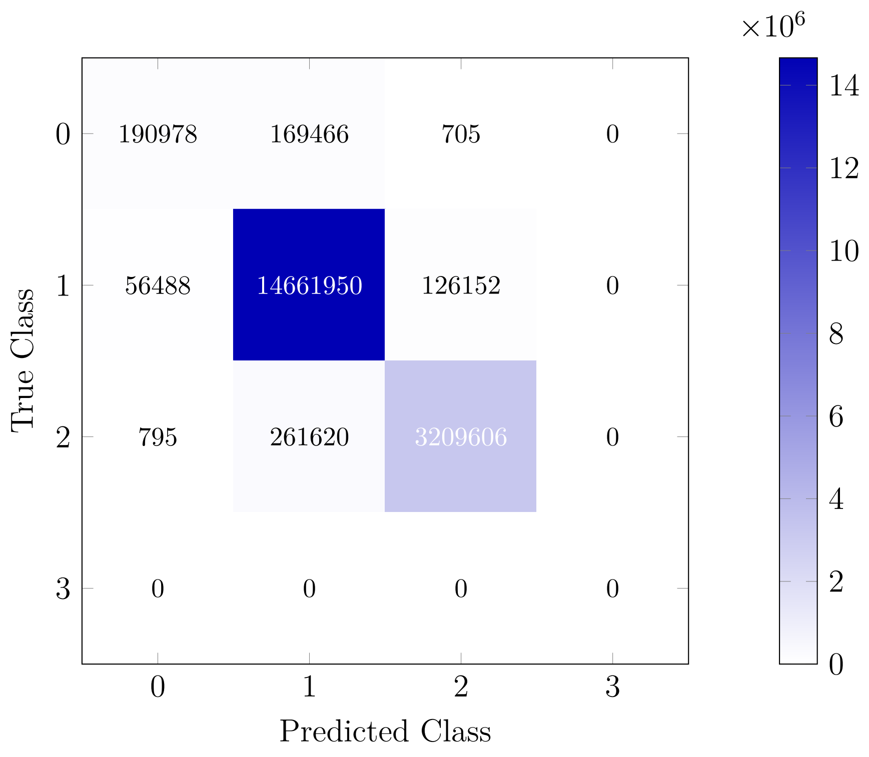


Figure 11.28: Confusion matrix heatmap for RCGDNAE (low bitrate) reconstructed images vs ground truth

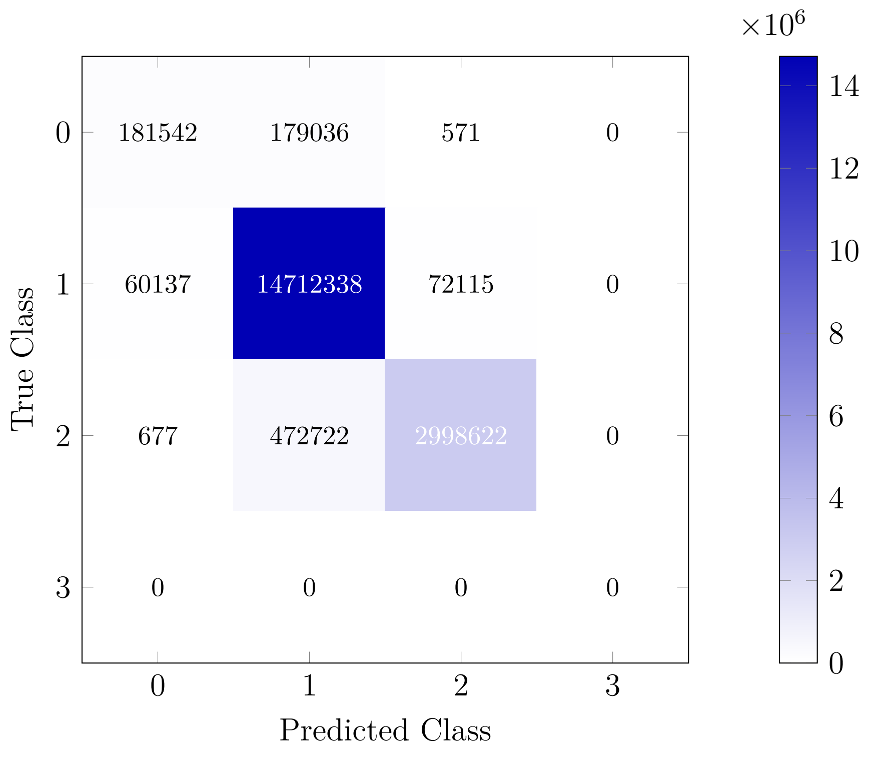


Figure 11.29: Confusion matrix heatmap for RCGDNAE (mid bitrate) reconstructed images vs ground truth

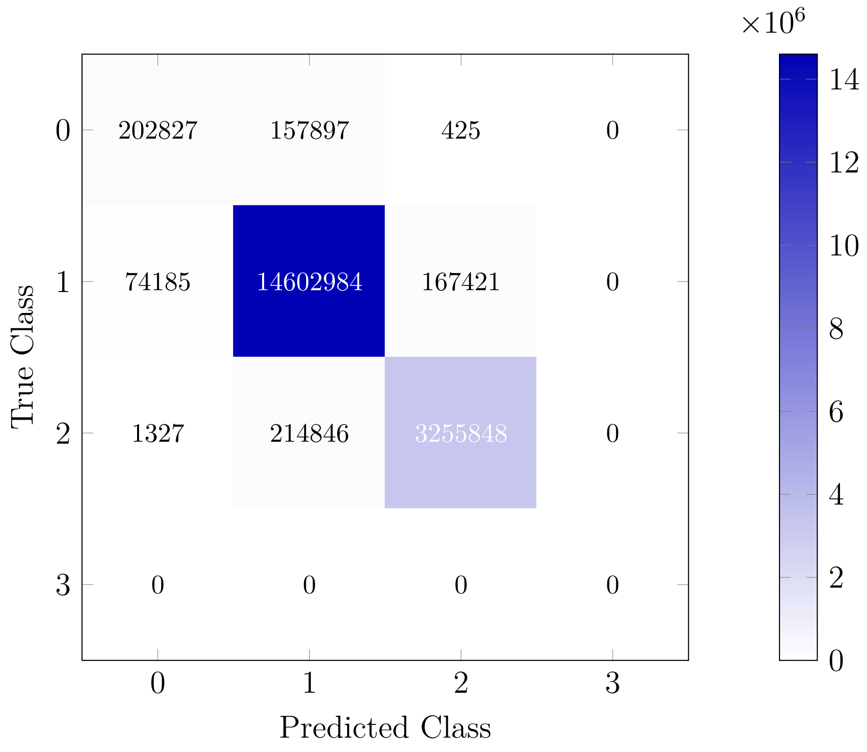


Figure 11.30: Confusion matrix heatmap for RCGDNAE (high bitrate) reconstructed images vs ground truth

Figures 11.20d, 11.21d, 11.22d, 11.23d, 11.24d, 11.25d, 11.26d, and 11.27d. Upon visual inspection of the error maps in Figures 11.20c, 11.21c, 11.22c, 11.23c, 11.24c, 11.25c, 11.26c, and 11.27c it is clear that the errors oscillate around the true pixel values. This is in contrast to the error maps for the other convolutional autoencoder models, where the errors seem to mostly come from specific regions of the images, such as edges or areas with high intensity gradients. The reconstructed images themselves show a lack of high frequency details and a general blurring effect, which is consistent with the observed error patterns.

The segmentation model was used to quantify the impact of these reconstruction artifacts on downstream tasks. The confusion matrices for the RCGDNAE reconstructed images against the ground truth segmentations are shown in Figures 11.28, 11.29, and 11.30 for the high, medium, and low bitrate RCGDNAE models, respectively. The segmentation metrics are reported in Tables 11.13, 11.15, and 11.17. The results also include per class metrics in Tables 11.14, 11.16, and 11.18. The results show that all compression levels lead to a degradation in segmentation performance compared to the original images. The error is however much smaller than expected given the visual quality of the reconstructed images, with the low bitrate model achieving an overall accuracy of 0.9671 and an F1-score of 0.8497. This suggests that while the reconstructed images may look visually poor, they still contain enough information for the segmentation model to perform reasonably well. This shows that while the usual metrics for image reconstruction quality, such as SSIM and SA, may indicate a significant loss of quality, the reconstructed images can still be useful for downstream tasks such as segmentation. This is especially important in the context of satellite imagery, where it is conceivable to achieve very high compression ratios while still retaining enough information for certain applications, which can be crucial for efficient storage and transmission of large volumes of satellite data.

Table 11.13: Average segmentation metrics for RCGDNAE (high bitrate) reconstructed images

Metric	Value
Accuracy	0.9670
F1-score	0.8527
IoU	0.7729
AUC	0.8688

Table 11.14: Per class segmentation metrics for RCGDNAE (high bitrate) reconstructed images

Metric	Value for class 0	Value for class 1	Value for class 2
PPV	0.7287	0.9751	0.9510
Recall	0.5616	0.9837	0.9377
F1-Score	0.6343	0.9794	0.9443
IoU	0.4645	0.9596	0.8945

Table 11.15: Average segmentation metrics for RCGDNAE (medium bitrate) reconstructed images

Metric	Value
Accuracy	0.9580
F1-score	0.8307
IoU	0.7419
AUC	0.8668

Table 11.16: Per class segmentation metrics for RCGDNAE (medium bitrate) reconstructed images

Metric	Value for class 0	Value for class 1	Value for class 2
PPV	0.7491	0.9576	0.9763
Recall	0.5027	0.9911	0.8637
F1-Score	0.6016	0.9740	0.9165
IoU	0.4302	0.9494	0.8459

Table 11.17: Average segmentation metrics for RCGDNAE (low bitrate) reconstructed images

Metric	Value
Accuracy	0.9671
F1-score	0.8497
IoU	0.7694
AUC	0.8691

Table 11.18: Per class segmentation metrics for RCGDNAE (low bitrate) reconstructed images

Metric	Value for class 0	Value for class 1	Value for class 2
PPV	0.7693	0.9714	0.9620
Recall	0.5288	0.9877	0.9244
F1-Score	0.6268	0.9795	0.9428
IoU	0.4564	0.9598	0.8918

11.4.5 Literature reference models

The models presented in the original papers for each of the reference architectures were used as the baseline for comparison with the models developed in this thesis. The models presented in the HySpecNet-11k paper[7] were used, namely the 1D-Convolutional Autoencoder (1D-CAE) (I), Extended 1D-Convolutional Auto-encoder (1D-CAE-Ext) (II), 3D Convolutional Auto-encoder (3D-CAE)(III), and Spectral Signals Compressor Network (SSCNet) (IV). Additionally, the results presented in the LineRWKV paper[23](V) were used as well as for the results presented in the A Scalable Reduced-Complexity Compression of Hyperspectral Remote Sensing Images Using Deep Learning paper[25](VI). For each of the HySpecNet-11k models, the provided models were run in the validation mode on the easy split of the dataset and the metrics. For the LineRWKV and Reduced Complexity models, we used the values presented in the original papers. To keep the comparison simple, we focused on the models with the lowest bpppc values from each paper. The models were evaluated using the same test dataset and metrics as the models developed in this thesis. The results of the reference models are presented in Table 11.19

Table 11.19: Reference models evaluation metrics

Metric	I	II	III	IV	V	VI
PSNR (dB)	52.52	43.22	39.06	43.91	53*	61.96
SSIM	0.997	0.964	0.934	0.969	N/A	N/A
SA (degrees)	1.86	5.94	5.80	3.13	N/A	N/A
bpppc	2.06	8.08	1.00	1.00	1.4*	0.1
CR	7.77	1.98	16.00	16.00	11.43	160.00

* Value estimated from graph in the original paper.

It is clearly visible that the models developed in this thesis underperform compared to the reference models from the HySpecNet-11k paper. This is due to the limited resources available for training and testing the models. An additional factor that skews the results is the much lower bpppc values achieved by the models developed in this thesis. The LineRWKV and the Reduced Complexity models could not be directly compared as there had been some difficulties in reproducing the results from the original papers.

Chapter 12

Comparison and conclusions

12.1 Compression comparison

The results obtained from the different models were compared based on the evaluation metrics discussed in Section 11.4. For context, a table comparing the results from the models developed in this thesis against selected reference models from the literature is provided in Table 12.1. The models are as follows: LineRWKV (I), Extended 1D-Convolutional Auto-encoder (1D-CAE-Ext) (II), 3D Convolutional Auto-encoder (3D-CAE)(III), Scalable Reduced-Complexity Compression Model (IV), LineRWKV reproduction (V), RCAE3D (VI), RCAE2D1D (VII), RCGDNAE medium bitrate model (VIII). Of these models, models I to IV are reference models from the literature, while models V to VIII were developed in this thesis. The comparison focused on both the models' quantitative performance, as measured by PSNR, SSIM, and SA, as well as qualitative aspects such as reconstruction quality and computational efficiency. The models constructed in the thesis demonstrated varying degrees of success in compressing and reconstructing hyperspectral images from the HySpecNet-11k dataset. The models in this thesis still achieved reasonable performance, with the best model in terms of PSNR being the LineRWKV reproduction, achieving a PSNR of 44.87 dB and SSIM of 0.9850. This model was then followed by the convolutional autoencoder models, which achieved PSNR values in the range of 34 to 38.15 dB and SSIM values between 0.866 and 0.977. One particularity worth noting is that the RCGDNAE model repeatedly achieved higher PSNR values compared to the other autoencoder models; however, it consistently had the lowest SSIM values among them. This suggests that while the RCGDNAE model was effective at minimizing pixel-wise errors, it may have struggled to preserve the overall structural integrity of the images as perceived by human observers. Overall, the weaker performance compared to published works indicates that there is still room for improvement in the models developed in this thesis.

Table 12.1: Comparison of evaluation metrics for reference models from the literature and models developed in this thesis.

Metric	I	II	III	IV	V	VI	VII	VIII
PSNR (dB)	53*	43.22	39.06	61.96	44.87	38.15	34.24	36.68
SSIM	N/A	0.964	0.934	N/A	0.985	0.977	0.961	0.866
SA (deg.)	N/A	5.94	5.80	N/A	3.987	5.137	6.131	9.27
bpppc	1.4*	8.08	1.00	0.1	2.00	1.267	0.633	0.0218
CR	11.43	1.98	16.00	160.00	8.00	12.62	25.09	129799

* Value estimated from graph in the original paper.
I LineRWKV
II Extended 1D-Convolutional Auto-encoder (1D-CAE-Ext)
III 3D Convolutional Auto-encoder (3D-CAE)
IV Scalable Reduced-Complexity Compression Model
V LineRWKV reproduction
VI RCAE3D
VII CAE2D1D
VIII RCGDNAE medium bitrate model

12.2 Segmentation performance

All of the models developed in this thesis are useful for downstream tasks that require hyperspectral image compression, especially in resource-constrained environments such as satellites. The performance of the models was evaluated based on their ability to perform semantic segmentation on the reconstructed images. The segmentation metrics for the different models are summarized in Table 12.2. With the model names corresponding to those in Table 12.1. An additional model was added (IX), which is the Simple Segmentation Model evaluated on the original uncompressed images serving as a baseline. This is demonstrated by the use of the Simple Segmentation Model, which was able to perform semantic segmentation on the reconstructed images from all of the models with reasonable accuracy. Some models achieved better segmentation accuracy than others, notably the RCAE2D1D model (VII), which achieved the lowest accuracy of 0.953, with an F1-score of 0.803 and IoU of 0.704. This indicates that while the model was able to compress the images effectively, it may have lost some important features necessary for accurate segmentation. On the other hand, the RCAE3D reproduction model (VI) achieved the highest accuracy of 0.968, with an F1-score of 0.839 and IoU of 0.753, indicating that it was able to preserve more of the important features necessary for accurate segmentation. Still, all models achieved reasonably high accuracy, F1-scores, IoU, and AUC values, indicating that they were able to retain important features for segmentation despite the compression. Of some interest is that the segmentation performance does not directly correlate with the SSIM or SA metrics. This suggests that while these metrics are useful for evaluating image quality, they may not fully capture the aspects of the images that are most relevant for segmentation tasks.

Table 12.2: Segmentation metrics comparison on reconstructed images from different compression models.

Model	V	VI	VII	VIII	IX
Accuracy	0.964	0.968	0.953	0.957	0.989
F1-score	0.839	0.839	0.803	0.830	0.904
IoU	0.748	0.753	0.704	0.741	0.844
AUC	0.867	0.865	0.861	0.866	0.873

V LineRWKV reproduction
VI RCAE3D
VII RCAE2D1D
VIII RCGDNAE medium bitrate model
IX Simple Segmentation Model on original uncompressed images (baseline)

12.3 Possible improvements and future work

The main issues affecting the performance were likely the limited computational resources available during development and training, as well as time constraints. Future work could focus on optimizing the model architectures further, especially allowing for much longer training epochs in excess of the 100 to 500 epochs used in this thesis. Additionally, exploring more advanced techniques such as transfer learning could potentially enhance model performance. Another avenue for future research could involve investigating hybrid models that combine the strengths of different architectures, such as integrating elements of LineRWKV with convolutional autoencoders. Yet another area for improvement could be the integration of the downstream segmentation task directly into the training process of the compression models. This could involve designing a multi-task learning framework where the model is trained to optimize both compression and segmentation performance simultaneously. Lastly, further research could focus on using the compressed representations for other downstream tasks beyond semantic segmentation, such as object detection or anomaly detection in hyperspectral images.

12.4 Conclusions

This thesis has explored the application of deep learning techniques for hyperspectral image compression using the HySpecNet-11k dataset. Four distinct compression models were developed and evaluated, each demonstrating varying degrees of success in compressing and reconstructing hyperspectral images. The models were assessed based on quantitative metrics such as PSNR, SSIM, and SA, as well as their performance in a downstream semantic segmentation task. While the models developed in this thesis did not surpass the performance of existing state-of-the-art methods, they still achieved reasonable results, indicating their potential utility in resource-constrained environments. The findings suggest that deep learning-based compression techniques hold promise for HSI compression, but further research and optimization are needed to fully realize their capabilities.

Bibliography

[1] Telmo Adão, Jonáš Hruška, Luís Pádua, José Bessa, Emanuel Peres, Raul Morais and Joaquim João Sousa. 'Hyperspectral Imaging: A Review on UAV-Based Sensors, Data Processing and Applications for Agriculture and Forestry'. In: *Remote Sensing* 9.11 (2017). ISSN: 2072-4292. DOI: 10.3390/rs9111110. URL: https://www.mdpi.com/2072-4292/9/11/1110.

[2] K. Subhash Babu, V. Ramachandran, K. K. Thyagharajan and Geeta Santhosh. 'Hyperspectral Image Compression Algorithms—A Review'. In: *Artificial Intelligence and Evolutionary Algorithms in Engineering Systems*. Ed. by L Padma Suresh, Subhransu Sekhar Dash and Bijaya Ketan Panigrahi. New Delhi: Springer India, 2015, pp. 127–138. ISBN: 978-81-322-2135-7.

[3] Venkatesh Bharadwaj. 'COLOURS: A SCIENTIFIC APPROACH'. In: *International Journal of Research -GRANTHAALAYAH* 2.3SE (2014), pp. 1–6. DOI: 10.29121/granthaalayah.v2.i3SE.2014.3543. URL: https://www.granthaalayahpublication.org/journals/granthaalayah/article/view/IJRG14_CC11_141.

[4] Madeleine F. Dupont, Aaron Elbourne, Daniel Cozzolino, James Chapman, Vi Khanh Truong, Russell J. Crawford and Kay Latham. 'Chemometrics for environmental monitoring: a review'. In: *Anal. Methods* 12 (38 2020), pp. 4597–4620. DOI: 10.1039/D0AY01389G. URL: http://dx.doi.org/10.1039/D0AY01389G.

[5] Wei Fu, Shutao Li, Leyuan Fang and Jón Atli Benediktsson. 'Adaptive Spectral–Spatial Compression of Hyperspectral Image With Sparse Representation'. In: *IEEE Transactions on Geoscience and Remote Sensing* 55.2 (2017), pp. 671–682. ISSN: 1558-0644. DOI: 10.1109/TGRS.2016.2613848.

[6] Martin Hermann Paul Fuchs, Akshara Preethy Byju, Alisa Walda, Behnood Rasti and Begüm Demir. 'Generative Adversarial Networks for Spatio-Spectral Compression of Hyperspectral Images'. In: *2024 14th Workshop on Hyperspectral Imaging and Signal Processing: Evolution in Remote Sensing (WHISPERS)*. IEEE, Dec. 2024, pp. 1–5. DOI: 10.1109/whispers65427.2024.10876535. URL: http://dx.doi.org/10.1109/WHISPERS65427.2024.10876535.

[7] Martin Hermann Paul Fuchs and Begüm Demir. 'HySpecNet-11k: a Large-Scale Hyperspectral Dataset for Benchmarking Learning-Based Hyperspectral Image Compression Methods'. In: *IGARSS 2023 - 2023 IEEE International Geoscience and Remote Sensing Symposium*. IEEE, July 2023. DOI: 10.1109/igarss52108.2023.10283385. URL: http://dx.doi.org/10.1109/IGARSS52108.2023.10283385.

[8] Martin Hermann Paul Fuchs. *HySpecNet Home Page*. URL: https://hyspecnet.rsim.berlin/.

[9] M Govender, K Chetty and H Bulcock. 'A review of hyperspectral remote sensing and its application in vegetation and water resource studies'. In: *Water SA* 33.2 (2009). DOI: 10.4314/wsa.v33i2.49049. URL: https://www.ajol.info/index.php/wsa/article/view/49049.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. URL: https://arxiv.org/abs/1512.03385.

[11] Miguel Hernández-Cabronero, Aaron Kiely, Matthew Klimesh, Ian Blanes, Jonathan Ligo, Enrico Magli and Joan Serra-Sagristà. 'The CCSDS 123.0-B-2 Low-Complexity Lossless and Near-Lossless Multispectral and Hyperspectral Image Compression Standard: A comprehensive review'. In: *IEEE Geoscience and Remote Sensing Magazine* PP (Feb. 2021). DOI: 10.1109/MGRS.2020.3048443.

[12] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[13] Lianfa Li, Ying Fang, Jun Wu and Jinfeng Wang. *Autoencoder Based Residual Deep Networks for Robust Regression Prediction and Spatiotemporal Estimation*. 2018. arXiv: 1812.11262 [cs.LG]. URL: https://arxiv.org/abs/1812.11262.

[14] Shumin Liu, Fahad Saeed, Zhenghui Yang and Jie Chen. 'A Comprehensive Review on Hyperspectral Image Lossless Compression Algorithms'. In: *Remote Sensing* 17.24 (2025). ISSN: 2072-4292. DOI: 10.3390/rs17243966. URL: https://www.mdpi.com/2072-4292/17/24/3966.

[15] Claudio Maccone. 'A simple introduction to the KLT (Karhunen—Loève Transform)'. In: Jan. 2009. ISBN: 978-3-540-72942-6. DOI: 10.1007/978-3-540-72943-3_10.

[16] Umberto Michelucci. *An Introduction to Autoencoders*. 2022. arXiv: 2201.03898 [cs.LG]. URL: https://arxiv.org/abs/2201.03898.

[17] Shima Mohammadi, Mohsen Jenadeleh, Jon Sneyers, Dietmar Saupe and João Ascenso. *Evaluation of Objective Image Quality Metrics for High-Fidelity Image Compression*. 2025. arXiv: 2509.13150 [cs.MM]. URL: https://arxiv.org/abs/2509.13150.

[18] Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Leon Derczynski, Xingjian Du, Matteo Grella, Kranthi Gv, Xuzheng He, Haowen Hou, Przemyslaw Kazienko, Jan Kocon, Jiaming Kong, Bartłomiej Koptyra, Hayden Lau, Jiaju Lin, Krishna Sri Ipsit Mantri, Ferdinand Mom, Atsushi Saito, Guangyu Song, Xiangru Tang, Johan Wind, Stanisław Woźniak, Zhenyuan Zhang, Qinghua Zhou, Jian Zhu and Rui-Jie Zhu. 'RWKV: Reinventing RNNs for the Transformer Era'. In: *Findings of the Association for Computational Linguistics: EMNLP 2023*. Ed. by Houda Bouamor, Juan Pino and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 14048–14077. DOI: `10.18653/v1/2023.findings-emnlp.936`. URL: `https://aclanthology.org/2023.findings-emnlp.936/`.

[19] Alfie Roddan, Tobias Czempiel, Chi Xu, Daniel S. Elson and Stamatia Giannarou. 'SAMSA: Segment Anything Model Enhanced with Spectral Angles for Hyperspectral Interactive Medical Image Segmentation'. In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2025*. Springer Nature Switzerland, Sept. 2025, pp. 478–488. ISBN: 9783032051141. DOI: `10.1007/978-3-032-05114-1_46`. URL: `http://dx.doi.org/10.1007/978-3-032-05114-1_46`.

[20] Olaf Ronneberger, Philipp Fischer and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: `1505.04597 [cs.CV]`. URL: `https://arxiv.org/abs/1505.04597`.

[21] Vít Růžička, Gonzalo Mateo-Garcia, Luis Gómez-Chova, Anna Vaughan, Luis Guanter and Andrew Markham. 'Semantic segmentation of methane plumes with hyperspectral machine learning models'. In: *Scientific Reports* 13.1 (2023), p. 19999. ISSN: 2045-2322. DOI: `10.1038/s41598-023-44918-6`. URL: `https://doi.org/10.1038/s41598-023-44918-6`.

[22] Juan Terven, Diana-Margarita Cordova-Esparza, Julio-Alejandro Romero-González, Alfonso Ramírez-Pedraza and E. A. Chávez-Urbiola. 'A comprehensive survey of loss functions and metrics in deep learning'. In: *Artificial Intelligence Review* 58.7 (Apr. 2025). ISSN: 1573-7462. DOI: `10.1007/s10462-025-11198-7`. URL: `http://dx.doi.org/10.1007/s10462-025-11198-7`.

[23] Diego Valsesia, Tiziano Bianchi and Enrico Magli. 'Onboard deep lossless and near-lossless predictive coding of hyperspectral images with line-based attention'. In: *IEEE Transactions on Geoscience and Remote Sensing* (2024).

[24] Freek D. van der Meer, Harald M.A. van der Werff, Frank J.A. van Ruitenbeek, Chris A. Hecker, Wim H. Bakker, Marleen F. Noomen, Mark van der Meijde, E. John M. Carranza, J. Boudewijn de Smeth and Tsehaie Woldai. 'Multi- and hyperspectral geologic remote sensing: A review'. In: *International Journal of Applied Earth Observation and Geoinformation* 14.1 (2012), pp. 112–128. ISSN: 1569-8432.

DOI: https://doi.org/10.1016/j.jag.2011.08.002. URL: https://www.sciencedirect.com/science/article/pii/S0303243411001103.

[25] Sebastià Mijares i Verdú, Johannes Ballé, Valero Laparra, Joan Bartrina-Rapesta, Miguel Hernández-Cabronero and Joan Serra-Sagristà. 'A Scalable Reduced-Complexity Compression of Hyperspectral Remote Sensing Images Using Deep Learning'. In: *Remote Sensing* 15.18 (2023). ISSN: 2072-4292. DOI: 10.3390/rs15184422. URL: https://www.mdpi.com/2072-4292/15/18/4422.

[26] P WT Yuen and M Richardson. 'An introduction to hyperspectral imaging and its application for security, surveillance and target acquisition'. In: *The Imaging Science Journal* 58.5 (2010), pp. 241–253. DOI: 10.1179/174313110X12771950995716. eprint: https://doi.org/10.1179/174313110X12771950995716. URL: https://doi.org/10.1179/174313110X12771950995716.

[27] Lei Zhang, Longsheng Zhang, Chengpeng Song and Peng Zhang. 'Hyperspectral Image Compression Sensing Network With CNN–Transformer Mixture Architectures'. In: *IEEE Geoscience and Remote Sensing Letters* 21 (2024), pp. 1–5. ISSN: 1558-0571. DOI: 10.1109/LGRS.2024.3403828.

Appendices

Index of abbreviations and symbols

Adam	Adaptive Moment Estimation (optimizer)
AuC	Area under Curve
dB	decibel
deg.	degree
CAE	Convolutional Autoencoder
CNN	convolutional neural network
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture (NVIDIA)
CR	compression ratio
DL	deep learning
e.g.	exempli gratia (for example)
EnMAP	Environmental Mapping and Analysis Program (satellite)
FN	false negative
FP	false positive
GAN	generative adversarial network
GDN	Generalized Divisive Normalization
GeLU	Gaussian Error Linear Unit
GeoTIFF	geographic Tagged Image File Format
GPU	Graphics Processing Unit
HSI	hyperspectral image

HSIS	hyperspectral image sensor
IDE	integrated development environment
IGDN	Inverse Generalized Divisive Normalization
IoU	Intersection over Union
KLT	Karhunen–Loève Transform (principal component-like transform)
Keras	high-level neural networks API
MSE	Mean Squared Error
MAE	Mean Absolute Error
N/A	Not Applicable
NumPy	Numerical Python library
pip	Package Installer for Python
PSNR	Peak Signal-to-Noise Ratio
PyPI	Python Package Index
PPV	Positive Predictive Value
RCAE	Residual Convolutional Autoencoder
RCGDNAE	Residual Convolutional Generalized Divisive Normalization Autoencoder
RMSProp	Root Mean Square Propagation
ROC	Receiver Operating Characteristic
RWKV	Receptance-Weighted Key-Value
RNN	recurrent neural network (RNNs = plural)
RGB	red–green–blue
SA	Spectral Angle
SGD	stochastic gradient descent
SSIM	Structural Similarity Index Measure
TF	TensorFlow

TN	true negative
TP	true positive
bpppc	bits per pixel per channel
μm	micrometer (10^{-6} meter)
nm	nanometer (10^{-9} meter)
UV	ultraviolet
VS Code	Visual Studio Code
NIR	near-infrared
SWIR	short-wave infrared

List of additional files in electronic submission

Additional files uploaded to the system include:

- source code of the project, containing the implementation of the proposed models

List of Figures

List of Tables