Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Sakaria Hussein Ali

Transform-based Lossless and Lossy Compression of Hyperspectral Images

Master's thesis in Electronics Systems Design and Innovation
Supervisor: Milica Orlandic
Co-supervisor: Sivert Bakken

July 2023

NTNU
Norwegian University of
Science and Technology

Sakaria Hussein Ali

Transform-based Lossless and Lossy Compression of Hyperspectral Images

Master's thesis in Electronics Systems Design and Innovation
Supervisor: Milica Orlandic
Co-supervisor: Sivert Bakken
July 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

NTNU
Norwegian University of
Science and Technology

Abstract

This thesis explores transform-based methods to improve lossy and lossless compression of hyperspectral images as an alternative to and in combination with the existing CCSDS standards. In addition, it explores using state-of-the-art compression standards for RGB images, such as JPEG XL, for hyperspectral images. Experiments were designed to measure the performance of these methods. The most exciting result from the experiments for lossless compression was a slight improvement on the CCSDS-123 standard by first spectrally decorrelating the hyperspectral images with the reversible Cohen–Daubechies–Feauveau 9/7 integer wavelet. The improvement was 10.8 % for the CCSDS test dataset and 7.6 % for the HYPSO dataset. This thesis is the first time compression experiments of this scale have been conducted on HYPSO images. For lossy compression, a combination of JPEG 2000 and the Karhunen-Loeve transform achieved the highest quality for all bit rates for both the PSNR and SSIM metrics. The JPEG XL standard was likely too optimized for the human visual system to work well on non-visual transform coefficients.

Sammendrag

Denne avhandlingen utforsker transformasjonsbaserte metoder for å forbedre komprimering, med og uten tap, av hyperspektrale bilder som et alternativ til og i kombinasjon med de eksisterende CCSDS-standardene. I tillegg utforskes bruk av de nyeste komprimeringsstandardene for vanlige bilder, som JPEG XL, på hyperspektrale bilder. Eksperimenter ble designet for å måle ytelsen til metodene. Det mest spennende resultatet fra eksperimentene for tapsfri komprimering var en liten forbedring av CCSDS-123-standarden ved først å spektralt dekorrelere de hyperspektrale bildene med den reversible Cohen-Daubechies-Feauveau 9/7 integer wavelet. Forbedringen var $10,8$ % for CCSDS-testdatasettet og $7,6$ % for HYPSO-datasettet. Denne avhandlingen er første gang komprimeringseksperimenter i denne skalaen er utført på HYPSO-bilder. For komprimering med tap oppnådde en kombinasjon av JPEG 2000 og Karhunen-Loeve-transformasjonen den høyeste oppnådde kvaliteten, uavhengig antall bits per piksel, for både PSNR- og SSIM-målene. JPEG XL-standarden var sannsynligvis for optimalisert for menneskes synsystem til å fungere godt på ikke-visuelle transformasjonskoeffisienter.

Acknowledgement

I would like to thank my supervisors, Milica Orlandic and Sivert Bakken, for providing theoretical guidance and motivation when needed. Especially Sivert Bakken for frequent discussions on relevant topics.

Furthermore, I would like to thank everybody who spent countless hours on the fifth floor of the electrical building, both blocks C and D, over the last two years. For always having someone take a break or chat with.

Lastly, I would like to thank all of the people I've met in Trondheim over the last five years for giving me years of fun, learning and endless memories.

Sakaria Hussein Ali
Lunde, Telemark
17.07.2023

Preface

This document is a master's thesis written for the Department of electronic systems at the Norwegian University for Science and Technology NTNU. The thesis is the culmination of 5 years of studying electronic system design and innovation and concludes my degree.

The sections 1, 2 and 3 are based on a project thesis written by me for the subject TFE4580. The thesis is not published so I can therefore not cite the thesis.

Contents

Introduction 1

Earth-observing satellites are used to monitor and explore the health of our world. To enhance the performance of these satellites, specialized algorithms are designed to perform tasks onboard the satellite. This thesis aims to find better compression algorithms that hopefully would allow for capturing more images per satellite. If we want to explore the images further than the hardware on the satellites allows, we need to transmit the images to a ground station.

1.1 Motivation

In recent years the cost of satellite missions has decreased drastically, mainly due to reduced launch costs and more standardization in the satellite industry [32]. This has increased the need and spurred an interest in computational methods and algorithms specialized for use in satellites. Compression algorithms are important for satellite imaging missions because they allow for increased transmission of images to the ground, allowing the satellite to take more images.

Earth observation from satellites is an integral part of monitoring the environment. With an accelerating climate change crisis, the need and importance of earth observation could not be clearer. The melting of the ice caps, forest fires and algae blooms are examples of phenomena that can be detected through hyperspectral earth observation.

United Nations developed 17 goals to improve human life and protect the environment. Hyper-spectral imaging satellites contribute to several of these goals. Three goals stick out as the three goals were hyperspectral imaging satellites can contribute the most. They are goal 6 - clean water and sanitation, goal 13 - climate action, and goal 14 life below water. Figure 1.1 shows the official UN illustrations of these goals.

Fig. 1.1: Illustration of the relevant UN sustainable development goals.

1.2 Satellite Design

For the Hypso-1 satellite, the downlinking capabilities are the bottleneck limiting the potential operational capabilities of the satellite. Other satellites have experienced the same problem [27]. This became clear in the early in the lifetime of the satellite. In general, the HYPSO-1 satellite gets more power from the solar panels than it needs to perform its functionalities. Therefore there is an opportunity to employ more power-demanding algorithms to get more performance from the satellite.

1.3 HYPSO Mission

HYPSO, or HYPer-spectral Smallsat for ocean Observation, is a project at NTNU that designs, builds and operate satellites for monitoring ocean health [23]. The project is part of the Center for Autonomous Marine Operations at NTNU. The center includes other projects that focus on other autonomous vehicles and operations. The goal is that in the future these vehicles can work in tandem to give a complete picture of the ocean health in an area. The first HYPSO satellite is already in orbit, HYPSO-1, and one more is currently being developed, and funding secured for more in the future. Figure 1.2 shows two captures taken with the HYPSO-1 satellite.

1.4 Problem Statement

This thesis explores transform-based methods alternative to or in combination with the CCSDS-123 standard to compress hyperspectral images in a lossless and lossy way by using transforms to decorrelate the spectral pixels. In addition, using modern codecs intended for RGB images, such as the JPEG XL standard. Primarily seeking to

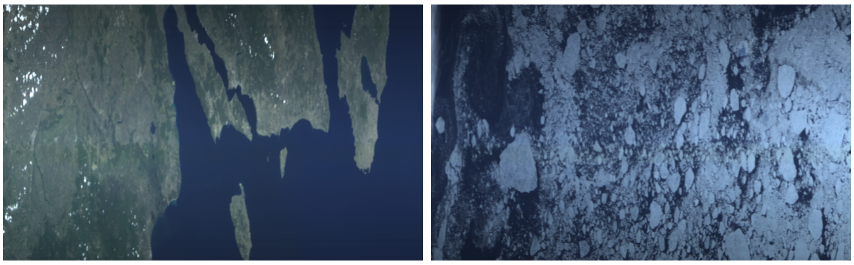

Fig. 1.2: Two images captured by the HYPSO-1 satellite. The image to the left is from the Dardanelles by the northwestern coast of Turkey, and the image to the right is an image of Newfoundland, Canada.

achieve higher compression while maintaining the same or higher quality. The idea is to use more computationally heavy methods to try and trade power and time for compression performance. This will give the opportunity for hyper-spectral imaging satellite missions with a positive power budget, such as HYPSO-1, to capture more scenes.

1.5 Thesis Structure

The main body of the thesis consists of seven chapters. The first is the introduction 1, which gives the motivation and defines the problem statement. Followed by theory 2, where the necessary theoretical background for image compression is presented. Chapter 3, method, explains how the compression experiments were conducted. Chapter 4 presents the results of compression experiments and highlights the most interesting findings. Chapter 5 discusses the results found in the earlier chapter and deliberates over the strengths and weaknesses of both methodology and transforms. In addition to giving recommendations for future work on the subject. The final chapter 6 concludes and summarizes the thesis.

Theory 2

2.1 Image Compression

Image compression aims to reduce the amount of data needed to represent a digital image. When an image is captured, each sample is stored as a fixed-length binary number. This representation of visual information stores redundant and irrelevant data for natural images. Creating larger data files to store and transmit. Even more so for hyperspectral images that have a much higher degree of spectral resolution than normal RGB images. Image compression algorithms find a representation of the information that reduces the amount of redundant and irrelevant data.

2.1.1 Fundamentals

Image compression is mainly divided into two distinct categories, lossless and lossy. After lossless compression, the original image can be reconstructed without any errors. While for lossy compression some error is accepted to further decrease the amount of data needed to represent the image. Lossless compression finds a representation that minimizes redundant data, while lossy compression minimizes redundant and irrelevant data. Near-lossless is often used term for lossy compression where the error in the reconstructed image is small and can be controlled directly at the encoding stage.

Redundancy

A raw image, where all the samples from the camera sensor are stored directly, will contain a lot of redundant data. In this context, redundant data refers to data that represents information that can be derived from other parts of the image or data that can be coded in a more efficient way. Redundant data due to inefficient coding is called coding redundancy, while redundant data due to representing the same information as other samples is called spatial, spectral or temporal redundancy [39].

In this context, coding means the strategy that gives numerical values a binary code. The raw representation codes the sample values as fixed-length binary code and therefore needs to allow for the coding of all of the possible values. An example of coding redundancy is an image containing many repeated values and only a small percentage of the values that the coding allows. A more efficient way to code the sample values in this image would be to use a variable length code that only assigns codes to values present in the image. This creates the need for a codebook for the decoder to be able to reconstruct the original image that needs to be stored and transmitted with the image. Often used coding strategies for image compression are Huffman coding [30], arithmetic coding [41] and bit-plane coding [40].

Another form of redundancy occurs when the same information is represented in multiple samples. For natural images, this is usually the case for neighboring samples along the spatial and spectral axis of an image. In literature, redundancy along the spatial axis is called spatial redundancy, and redundancy along the spectral axis is called spectral redundancy. For videos, the neighboring frames will also carry a lot of the same information, and the redundancy between these frames is called temporal redundancy [39]. RGB images sample the visual spectrum at only three fairly spaced out points and will therefore have a low degree of spectral redundancy. However, hyperspectral images have a high resolution along the spectral axis, and it follows that there exists more redundancy along this axis.

Methods are designed to decorrelate the samples in images to remove redundancy. By decorrelating the samples, the same information will no longer be carried through multiple samples, creating less redundant information in the image. Two main strategies exist for decorrelating samples, transforms and prediction [19]. This thesis will focus on the transform methods, and how they work will be explained more in 2.2. Methods developed for hyperspectral images usually focus more on decorrelating along the spectral axis than methods developed for RGB images because more redundancy exists along this axis [13].

Relevancy

Generally, the simplest way to compress an image is to discard irrelevant information. The use case of the image defines what information is deemed irrelevant. The most common use case for images is to be viewed by humans. Therefore most common strategies for reducing irrelevant information focus on eliminating information that is ignored by the human visual system. However, hyperspectral images are mainly for scientific purposes and are not generally intended to be viewed by humans. For

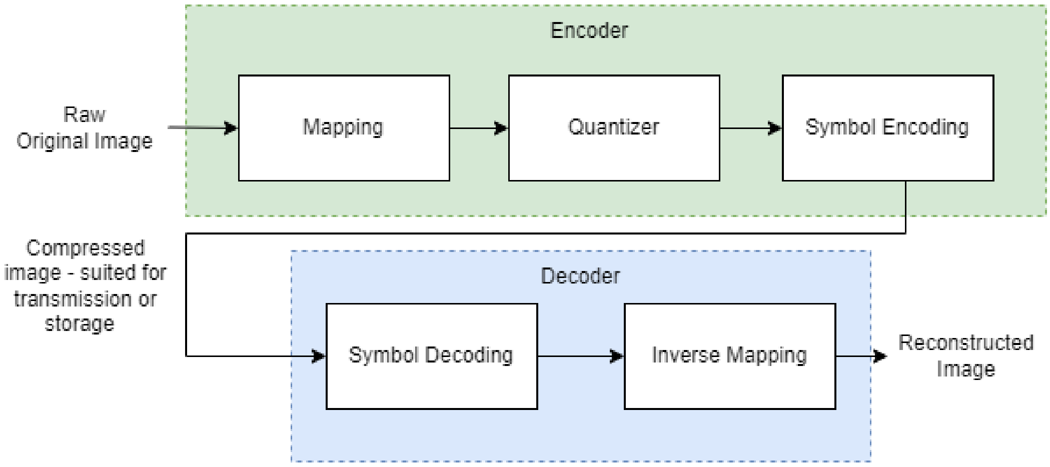

Fig. 2.1: A diagram that shows the typical stages of compressing and decompressing an image. The diagram is inspired by a similar diagram in [39].

scientific use generally want to keep as much information as possible because new ways to exploit and interpret the information are being developed constantly. A typical use case for hyperspectral images is anomaly detection, classification and training of similar machine learning methods [6].

2.1.2 Compression Model

Figure 2.1 describes the typical stages of compressing and decompressing an image. The stage-wise breakdown of the encoder and decoder is only a model, and not every compression scheme will fit the model exactly. However, it allows us to describe a compression scheme generally and compare the strengths of weaknesses of different methods [39].

The model consists of two complementary parts, the encoder and the decoder. The encoder creates a compressed representation of the image that is suitable for transmission and storage. While the decoder reconstructs the original image as closely as possible. The encoder generally consists of three stages, mapping, quantization and symbol encoding. Conversely, the decoder consists of an inverse mapper and a symbol decoder. The quantization in the encoder does not have a corresponding step in the decoder because quantization removes information that is unrecoverable. It follows that lossless compression schemes do not have a quantization step.

Mapping

The purpose of the mapping stage is to remove as much spatial and spectral redundancy as possible. And temporal redundancy for video. Often used methods are transforms or prediction. Transforming the image into a set of less redundant transform coefficients allow for higher compression performance. Prediction can remove redundancy by mapping the samples to the prediction errors, which are less correlated than the original samples.

Quantization

The quantizer's purpose is to remove irrelevant information. This is achieved by restricting the possible values of the samples. Which allows for a more efficient symbol encoding. The possible values the samples can take are restricted by essentially mapping the sample values to a smaller set of possible values. Since the resulting set is smaller, multiple values are mapped to the same value in the smaller set, and therefore the mapping is irreversible. Different quantization methods differ in how the values in the smaller set are defined and how the mapping is applied. How aggressive the quantization is can generally be set when compressing an image.

Encoding

The third and final stage is the encoder. It's purpose is to remove the coding redundancy. In an efficient manner, the sample values are given shorter codes that require fewer bits. Often a variable length code is used, which gives shorter codes to more frequent values, which in turn produces a shorter average code length. This process is reversible. However, one generally requires a codebook to be stored with data for decoding the codes. As mentioned earlier, Huffman coding, Golomb/Rice coding [18], arithmetic coding and bit-plane coding are examples of encoding techniques. After this stage, the compressed image should have a smaller file size and be suitable for storage and transmission.

2.1.3 Metrics

To evaluate lossy compression techniques, a metric is needed to quantify the effect that removing information from the original image has on the quality of the image. In this context, quality means the usefulness of the image in the setting that it will be used. Different use cases will need different levels of quality and different metrics to evaluate the quality.

Mean Square Error

The most intuitive way to design a quality metric for reconstructed images is to use the difference between the original and reconstructed image. The difference is the error defined in 2.1, where $f(x, y, z)$ is the original image and $\hat{f}(x, y, z)$ is the reconstructed image.

$$E(x, y, z) = \hat{f}(x, y, z) - f(x, y, z) \tag{2.1}$$

The average error for all the samples in all dimensions is taken to get one metric for the entire image. The errors are squared to ensure that the errors don't cancel each other out. The resulting metric is called the *mean square errror* and is defined in 2.2.

$$MSE(x, y, z) = \frac{1}{NMK} \sum_{N} \sum_{M} \sum_{K} (\hat{f}(x, y, z) - f(x, y, z))^2 \tag{2.2}$$

PSNR

The Peak Signal to Noise Ratio expands on the mean square error by considering the reconstructed image as the sum of the original image and the error. Then the error can be considered as noise added when reconstructing the image. This ratio between the noise and the signal is often used in signal processing to measure the quality of a signal [38]. The most common signal-to-noise ratio used for image compression is the peak-signal-to-noise ratio, defined in 2.3. This metric is the ratio between the peak value of the signal and the mean square error. It is usually used in the decibel scale since the ratio can span several orders of magnitude.

$$PSNR = 10 \cdot \log_{10} \frac{MAX^2}{MSE} \tag{2.3}$$

SSIM

The Structural Similarity Index Measure, or SSIM, is a widely-used metric in the field of image processing for evaluating the similarity between two images, with consideration given to perceptual quality degradation and structural similarities [46]. A value of 1 indicates that images are identical, while a zero indicates no correlation. Negative values indicate a negative correlation between the images. The SSIM metric measures how closely the reconstructed image resembles the original, considering factors such as luminance, contrast, and structural similarities. This allows for a metric that evaluates quality degradation closer to how the human visual system would, compared to PSNR or MSE. The equation 2.4 below shows how the metric is computed.

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{2.4}$$

The x and y are windows of size $N \cdot N$ in the images that are being compared. The size N can be set to an arbitrary size when computing the metric. The equation to compute the SSIM consists of three underlying comparison metrics luminance, contrast and structure. The functions for these three metrics are given respectively in 2.5, 2.6 and 2.7. The three comparison metrics are all based on different statistical measures of the two windows, x and y, being compared. The luminance metric is based on the sample means in the windows x and y, the contrast is based on the variance and the structure metric is based on the covariance. Otherwise, the metrics have similar algebraic structures. To improve stability when the statistical measures are close to zero the constants c_1, c_2 and c_3 are added to the denominator and numerator. The original SSIM [46] paper sets these values to $c_1 = (k_1L)^2$, $c_2 = (k_2L)^2$ and $c_3 = \frac{c_2}{2}$, where $k_1 = 0.01$ and $k_2 = 0.03$.

$$l(x,y) = \frac{2\mu_x\mu_y + c_1}{\mu_x^2 + \mu_y^2 + c_1} \tag{2.5}$$

$$c(x,y) = \frac{2\sigma_x\sigma_y + c_2}{\sigma_x^2 + \sigma_y^2 + c_2} \tag{2.6}$$

$$s(x,y) = \frac{\sigma_{xy} + c_3}{\sigma_x\sigma_y + c_3} \tag{2.7}$$

Butteraugli distance

Butteraugli distance is an advanced metric created by Google to further optimize the lossy compression of images intended to be viewed by the human visual system [1]. The metric does not have an explicit formula but rather an algorithm that is performed on the images. The steps in the algorithm are outlined under:

1. Convert the images: The first step is to convert the images into linear-light sRGB color space. The rest of the algorithm assumes that this is the color space of the images.
2. Downsample: The images are down-sampled and a blur filter is applied to imitate the human visual system.
3. Apply masking: Apply a mask to images. The masking map is created by comparing the local gradients and the visual impact of different frequency components.
4. Calculate the sample differences: Take the difference between the two images after the masking is applied.
5. Aggregate the sample differences: Aggregate the sample differences into a single value, by combining the mean and max pooling.
6. Non-linear mapping: Take the aggregated value and apply a non-linear mapping to better emulate the human visual system.

As can be seen from the steps outlined the algorithm to obtain the Butteraugli distance is quite involved making it computationally intensive to compute. The benefit is that the Butteraugli distance when employed in an iterative optimization strategy can reduce the file size by 20% [1].

2.1.4 Entropy

Information theory serves as the mathematical foundation for data compression techniques. At the core of this field lies the concept of ***entropy***, which can be mathematically defined as:

$$H(X) = -\sum_{x \in X} p(x) \log_2 p(x) = E[-\log_2 p(X)] \tag{2.8}$$

In this equation, the random variable X represents the information source, and $p(x)$ denotes the probability density function associated with it. Entropy quantifies the inherent uncertainty of the information source, indicating the level of difficulty in predicting specific values from the source [39]. It is worth noting that entropy is maximized when the probability density function is uniform. Intuitively, this implies that when every outcome is equally likely, the task of predicting the outcome becomes exceedingly challenging. This observation aligns directly with the objective of data compression. For instance, consider an image composed of pixels randomly selected from a uniform source. In this case, even if one knows the underlying distribution, it would be nearly impossible to predict the value of a pixel. However, for a natural image, we expect the distribution to include a couple of distinct peaks, which makes it easier to predict the next pixel.

Another perspective on entropy is to view it as a measure of information. This concept may initially seem counter-intuitive, as demonstrated in the earlier example where a random image exhibited higher entropy than a natural image. However, one can reconcile this discrepancy by understanding that greater uncertainty regarding the outcome of an event corresponds to a higher amount of information gained upon learning the outcome. Furthermore, entropy establishes a lower limit on the achievable lossless compression of data from an information source [39]. This can be intuitively explained by considering that a lossless compressed representation must retain the same information as the original data. Therefore, the entropy, or information measure, defines the theoretical lower bound for lossless compression.

2.2 Transform Coding

A transform is a mathematical term defined as a mapping between vector spaces. A transform can have a wider definition in other contexts, but this definition is wide enough for the field of digital image processing. The equation defines it with mathematical symbols.

$$A : V \to U \tag{2.9}$$

where V and U are vector spaces, and A is the transform. For images, the vector spaces are almost always R^n or a subspace of R^n, where n is the dimension of the space [25] and R are the real numbers. For lossless compression, we need an inverse transform that reverses the mapping, defined under

$$A^{-1} : U \to V \tag{2.10}$$

A transform is essentially a change of basis [19]. A basis is defined as a set of vectors in a vector space that spans the vector space. The span of a set of vectors is the space of all vectors that are linear combinations of the vectors in the set.

$$b = \{v_1, v_2, ..., v_n\}, v_n \in V \tag{2.11}$$

$$x = \sum_{n=1}^{N} a_n v_n, a_n \in R \tag{2.12}$$

Where x is an arbitrary vector in V. Most vector spaces will have many different bases b that span the space. The scalars a_n can be seen as a representation of the vector with basis b [25]. A transform is a process that maps the arbitrary vector x into the scalars a_n, often called the transform coefficients. In image compression, we want to transform the image into coefficients that are more easily compressible [38].

Digital images are described by discrete values, and hence digital image processing transforms are generally discrete transforms.

We can group transforms into two distinct groups, data-dependent transforms and data-independent transforms. Data-dependent transforms construct a basis by performing some operation on the input data. Often to make the transform coefficients achieve some desired property. Data-independent transforms use a fixed basis for all possible inputs. In contrast to the data-dependent transforms, after the data-independent transform the coefficients do not generally achieve some property, but rather yield a new basis with some advantage when compressing [20].

Since the data-independent transforms have different bases depending on the input, the basis also needs to be stored and transmitted with the compressed image to allow inverse transforms to be applied. The extra data created by data-dependent transforms are called side-channel information or data. In addition, acquiring the transform basis is computationally expensive and time-consuming. For a data-dependent transform to be worthwhile in a compression scheme, it needs to improve the compression more than the extra side-channel information it creates. Furthermore, the application scenario should allow for the extra computational cost.

2.2.1 KLT

The Karhunen-Loeve transform KLT, also known as the Principal Component Analysis [16], is a data-dependent transform with many desirable properties for image compression. These properties hold when coding Gaussian sources. The most important of these properties are listed under

1. Minimizes the mean square error when the coefficients are truncated [21]
2. Decorrelates the data [16], [27]
3. Creates the representation of the data with minimum representation entropy [12].

The properties make it the optimal transform for image compression in cases with jointly Gaussian sources.

The transform is performed in three steps. First, the covariance matrix of the input sample vector is computed. Then the eigendecomposition of the covariance matrix is found, and lastly, the transform is applied using the basis given by the eigendecomposition [42].

There exists multiple ways for the Karhunen-Loeve Transform to be applied, but in the rest of the thesis, we will focus on the variation often used in hyperspectral image compression as seen in the papers [37],[24].

Computing The Covariance Matrix

The covariance matrix is computed by taking the covariance of all the dimensions of the random sample vector. For a 3x3 random vector, the covariance matrix would look like this.

$$C_x = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_y^2 & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_z^2 \end{bmatrix} \tag{2.13}$$

and the covariance of random variables A and B are

$$\sigma_{AB} = E\left[(A - E[A]) \cdot (B - E[B])\right] \tag{2.14}$$

where the E[] is the expected value operator. The covariance matrix is always symmetrical because $\sigma_{AB} = \sigma_{BA}$.

For hyperspectral images, the KLT is often performed over the spectral range where all of the samples from different bands but the same spatial location are regarded as one pixel.

Eigendecomposition

The covariance matrix C_x will always be a real-valued NxN symmetrical matrix. Therefore, in any case, the covariance matrix can be factorized into $C_x = W\Lambda W^T$, where the W is a unitary matrix with all of the eigenvectors of C_x, and Λ is a diagonal matrix with all of the eigenvalues along the diagonal [25]. The eigenvalues in Λ are ordered so that the eigenvalues with the highest values come first in the matrix. And the corresponding eigenvectors to the eigenvalues are ordered in the same way. This ensures that the truncation of the basis will always remove the dimensions with the least impact over the values in C_x [16].

Application

When computing the Karhunen-Loeve transform of a typically sized hyper-spectral image, over 100 000 covariance matrices must be computed. In addition, each covariance matrix will need to be subjected to eigenvalue decomposition to find the optimal transform basis. To reduce the amount of eigenvector problems that need to be solved a proposed speed-up of the transform is to compute an average covariance matrix over all of the spectral pixels so that only one eigenvector problem needs to be computed. This greatly reduces the computational cost of the transform [37]. Another proposed speed-up is to compute the average covariance matrix with a randomly selected subset of the spectral pixels [36]. Depending on the size of the subset, this can greatly improve the time it takes to compute the average covariance matrix. This technique is called covariance sub-sampling, and evaluation of the technique done in [36] and [37] show that a subset of a tenth or a hundredth of the spectral pixels both give comparable results to using all of the spectral pixels.

Integer Reversebility

For lossless compression, the transforms need to be integer-to-integer transforms. The normal Karhunen-Loeve-transform is not integer reversible by default. An integer-to-integer approximation of the Karhunen-Loeve transform has been developed by [24] and further developed by [15]. They propose an algorithm for factoring the transform matrix into triangular elementary reversible matrices, or TERMs, as long as the matrix is non-singular.

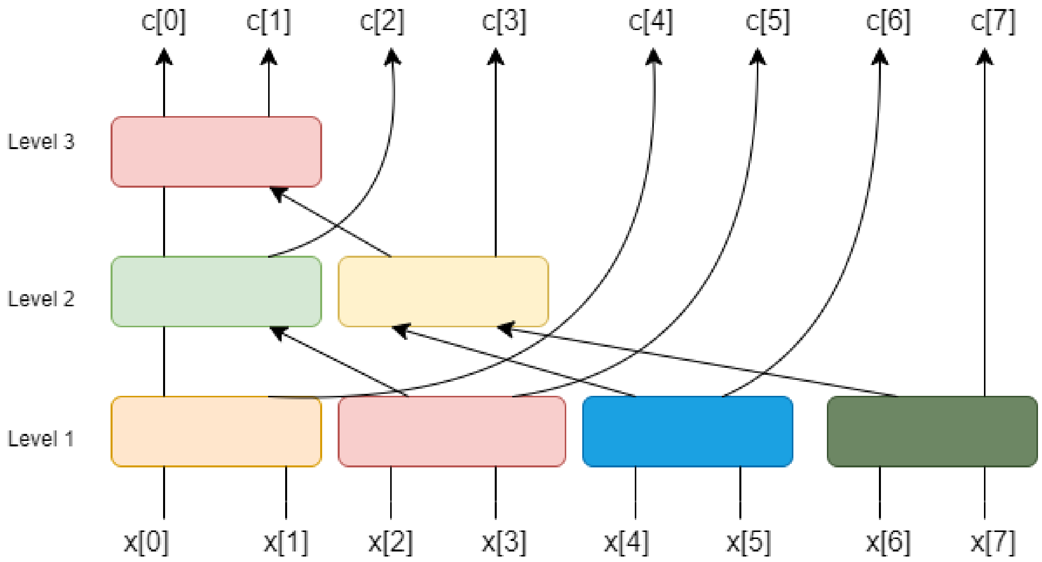

Fig. 2.2: A diagram illustrating how the input vector x[n], with length eight, is processed by Pairwise Orthogonal Transform. The transform coefficients are indicated as c[n]. This figure is inspired by a similar figure in the original POT paper [8].

2.2.2 Pairwise Orthogonal Transform

Several approximations to the Karhunen-Loeve Transforms have been proposed [7] [4] to try and get similar performance but significantly lower the computational cost. The pairwise orthogonal transform, POT, is one such proposal [8]. Many of these approximations are based on using a divide-and-conquer algorithm to reduce the complexity of the KLT transform. The Pairwise orthogonal transform decomposes the input in pairs of two values and performs a two-component KLT on the pair of values as shown in figure 2.2. A two-component KLT simplifies the eigenvector problem to a direct calculation of the transform matrix from the covariance matrix. In addition, the transform application is simplified because of the integer lifting decomposition, to ensure lossless integer reversibility, is simplified into a known structure. After the pair is transformed, the first principal component is transformed again in the same two-component transform with the principal component of the neighboring transformed pair. This process is continued for $log_2(n)$, where n is the size of the input vector since the size n is halved for each level.

2.2.3 Wavelets

Wavelet transforms are a family of transforms based on wavelets, wave-like functions with zero mean. The transform basis consists of one of these wave-like functions, called the mother wavelet $\Psi(t)$, scaled and shifted. In contrast to sines and cosines, the wavelets functions are square-integrable. Which means they are limited in both time and frequency. This means that the transformed representation contains both the frequency and time characteristics of the signal [39].

The scaling and shifting operation resemble convolution to such a degree that the wavelet transforms can be implemented as a series of low-pass and high-pass filters. First, the signal is simultaneously decomposed by low-pass and high-pass filters. The output of the high-pass filter is called detail coefficients, while the output of the low-pass filter is called approximation coefficients. Both sets of coefficients are downsampled by a factor of two. After the downsampling, the detail coefficients are stored, the approximation coefficients are sent through filters again, and the process is repeated [43]. The amount of the process repeated is called the level of the wavelet transform.

Figure 2.3 shows examples of three commonly used mother wavelets, the Morlet, Haar and Debauchies [9] wavelets. For a discrete application, these functions are discretely sampled [16].

2.2.4 Discrete Cosine Transform

The discrete cosine transform is a discrete transform that transforms a signal from the time or spatial domain to the frequency domain. The formula to acquire the transformed sequence is given in 2.15. In the formula, x[n] is the original signal and $x[k]$ is the transform coefficients representing the frequency domain signal.

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot \cos\left(\frac{\pi}{N} \cdot (n + 0.5) \cdot k\right) \tag{2.15}$$

The transform is similar to the well-known Discrete Fourier Transform that also transforms a signal to the frequency domain. The difference is that the discrete cosine transform basis only consists of cosines, not cosines and sines, such as the Fourier transform. The discrete cosine transform is preferred for image compression because the DCT compacts the energy of the signal in a small number of coefficients [39]. The first DCT coefficient will be the mean of the input signal, and the next coefficients will code higher and higher frequencies. Since most of the energy and information in natural images are contained in lower frequencies, the DCT will compact the energy into a few components. This feature of the DCT allows a compression scheme to discard the higher-frequency components or code them with lower accuracy.

2.3 Compression Codecs for Satellites

The consultative committee for space data systems CCSDS publishes recommended standards for use on computer systems in space. The standards emphasize low complexity because of the limited hardware on satellites and other spacecraft. The CCSDS has published a standard [26] for image compression of hyperspectral images. The standard aims to give the best possible compression ratio while simultaneously adhering to the constraints of satellite computer systems, mainly limited power and memory. The standard is named CCSDS-123 and has a couple of different variations and iterations.

2.3.1 CCSDS-123

The compression scheme uses prediction to reduce the amount of data needed to describe the image. A predictor tries to predict the value of a pixel by using the neighboring pixels. The error, the difference between the actual and predicted pixels, is quantized and encoded. The entropy encoder used is a Golomb/Rice encoder [26]. The idea here is that the prediction errors will be encoded more efficiently than the pixels themselves because the frequency of the same values showing up will increase, which generally gives a better encoding efficiency for entropy encoders[28].

2.3.2 CCSDS-121

The CCSDS-121 is a precursor to the CCSDS-123 standard. The encoder is essentially the same, but a more rudimentary decorrelator is used [10].

2.4 JPEG

The JPEG standards are the most commonly used compression standards in a wide area of applications. The standards were developed by JPEG (Joint Photography Expert Group) committee originally for still image photography. The committee consists of many different industry leaders, researchers and universities. The most widely used JPEG standards are JPEG 2000 and JPEG 1.

2.4.1 JPEG-1

JPEG 1 is the most used image compression standard. Even though it was created in 1992, it still performs well enough that it is preferred for most applications. It uses a Discrete Cosine Transform DCT to transform the data into space where the image data is more compacted into a smaller part of the space. The data is then quantized and then encoded. A combination of Run-length coding and Huffman coding is used to encode the images. Run-length coding is used on the zero values, while Huffman coding is used on the rest. The transform is performed on $8 * 8$ square of the image, which causes unwanted blocking effects with low bit rates. In addition, the JPEG standard is inherently lossy [29].

2.4.2 JPEG-LS

JPEG-LS is a fast and efficient lossless to near-lossless compression standard developed and standardized in 1998. The standard is prediction-based and employs a fairly simple predictor given in the expression 2.16. Here X is the predicted value, A is the sample to the left of the arbitrary pixel being predicted, B is over the sample, and C is neighboring over to the left of the sample.

$$X = \begin{cases} \min(A, B) & \text{if } C \geq \max(A, B) \\ \max(A, B) & \text{if } C \leq \min(A, B) \\ A + B - C & \text{otherwise} \end{cases} \tag{2.16}$$

After the prediction, the prediction errors are coded with Golomb/Rice coding. In addition, large homogeneous areas are coded using Run-length coding. The compression algorithm used is called the LOCO-I - Low Complexity Lossless Compression for images [48].

2.4.3 JPEG-2000

The JPEG 2000 standard is proposed as an improvement on JPEG 1. It adds several new features, such as dynamic bit rates, lossless compression, rate-distortion control and no blocking effects. These features are made possible by the change in transform to Discrete Wavelet Transform DWT [45]. The transform is performed on the entire image, not blocks such as JPEG 1. The encoder used is called an entropy encoder embedded block coding with optimized truncation [44]. The codes allow for truncation at any point in an optimal with regard to rate and distortion. JPEG 2000 has been applied to compress hyperspectral images.

2.4.4 JPEG-XL

The JPEG XL is a modern compression standard created to achieve better performance when compressing images both with and without loss. One of the main features is that it allows for transcoding JPEG images to JPEG XL and further compressing the image without loss. [47]. The standard involves both a prediction, using the neighboring samples, and transform coding. JPEG XL uses a variable discrete cosine transform, which essentially means that it varies the sizes of the blocks used in applying the DCT. The blocks vary from $2x2$ to $32x32$, which allows for more efficient coding. JPEG XL is very optimized for the human visual system and employs several tricks to remove information that the human visual system does not notice [1].

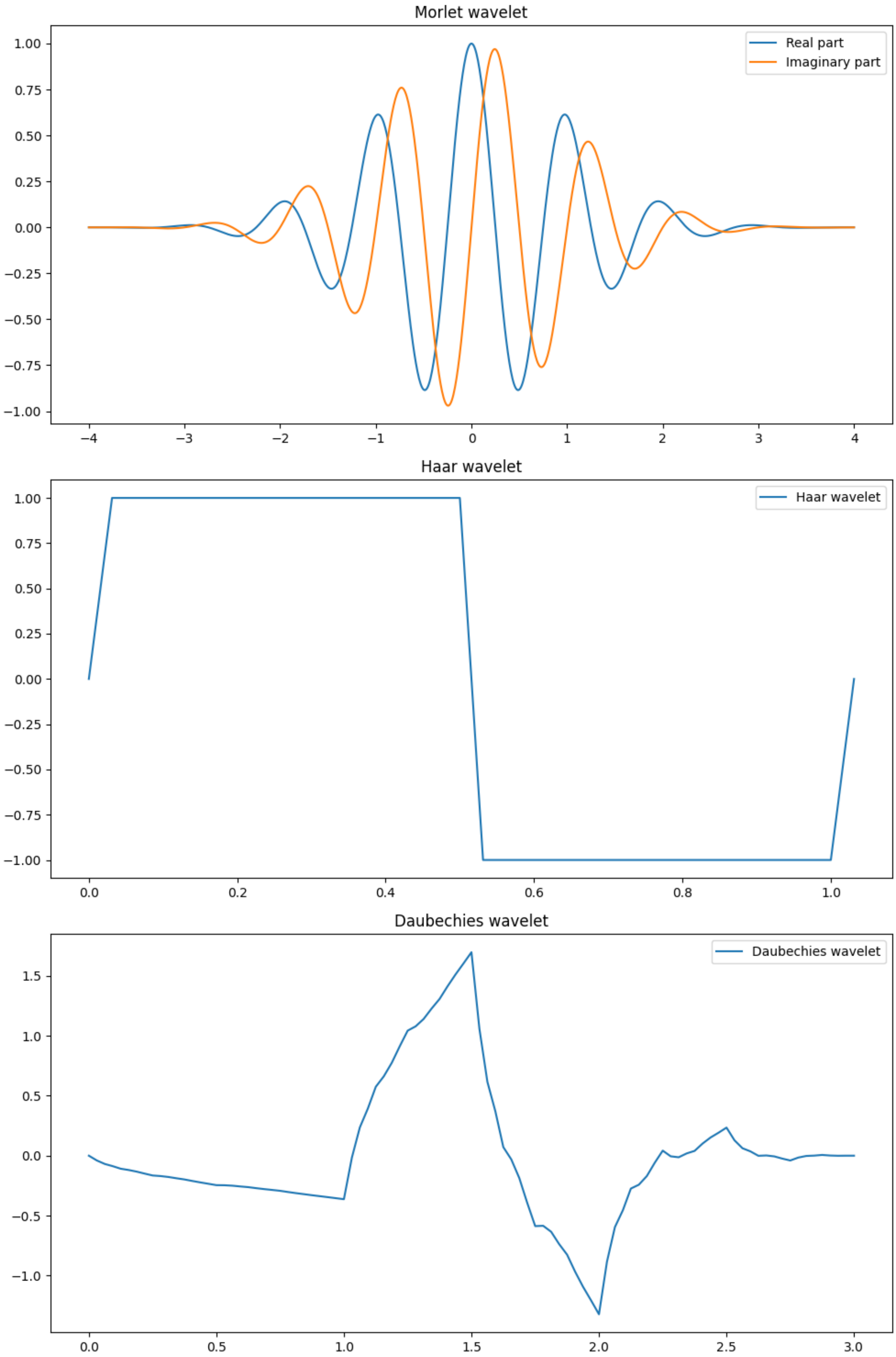

Fig. 2.3: Example of mother wavelets. The Morlet, Haar and Daubechies wavelets are depicted.

Methodology 3

The chapter outlines the methodology used to tackle the problem statement described in 1.4. In addition, it describes how the results in section 4 are attained and creates the possibility for others to reproduce the results. Furthermore, it gives proper context to understand and discuss the results presented.

3.1 Compression experiments

Several experiments were designed to test which methods that gave the best compression performance when compressing hyperspectral images. A handful of transforms and codecs were chosen and combined to create compressed representations of the images. The transforms and codecs chosen were the ones deemed to have the most potential to achieve good results going of the relevant literature. The images were decompressed and the reconstructions were compared with the original images using different metrics. The experiments conducted were separated into two different categories, lossless and lossy.

3.1.1 Lossless Experiments

Figure 3.1 shows how the lossless experiment was conducted. All of the datasets described in section 3.3 were used to conduct these experiments. The experiment was conducted with a combination of all of the transforms and codecs. First, the images are transformed along the spectral axis, meaning the axis that contains the samples from the same spatial location for different wavelengths. Then a constant is added to all of the transform coefficients so all coefficients are non-negative. Then all of the codecs are used to compress the images, and the resulting compression ratios are stored. In addition, the images are decompressed, and the inverse transforms are applied to ensure the compression scheme is actually lossless.

The list below lists the five transforms that were used in the experiment and the abbreviations that will be used to refer to the transforms from now on. The imple-

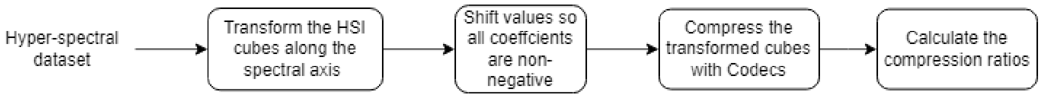
Hyper-spectral dataset
Transform the HSI cubes along the spectral axis
Shift values so all coeffcients are non-negative
Compress the transformed cubes with Codecs
Calculate the compression ratios

Fig. 3.1: A diagram showcasing the steps in the lossless compression experiment.

mentations and parameter settings used are described in 3.1.4. All of the transforms are integer-to-integer reversible.

1. Reversible Karhunen-Loeve Transform - RKLT
2. Pairwise Orthogonal Transform - POT
3. ISO-Range Pairwise Orthogonal Transform - IRPOT
4. Integer Wavelet Cohen–Daubechies–Feauveau 5/3 - CDF 5/3
5. Integer Wavelet Cohen–Daubechies–Feauveau 9/7 - CDF 9/7

The list below lists all of the codecs used in the lossless compression experiment. A description of how the codecs work can be found in section 2, and which implementations were used can be found in 3.1.5

1. CCSDS-123
2. CCSDS-121
3. JPEG 2000
4. JPEG-LS
5. JPEG XL

3.1.2 Lossy Experiments

The purpose of the lossy compression experiment is to compare different methods of removing irrelevant information from the images. The two methods explored were using JPEG XL and JPEG 2000 to code the hyperspectral scenes, and the other method is truncating the transform coefficients of the Karhunen-Loeve transformed representation. The paper [37] was the first to conduct similar experiments and was the inspiration for these experiments. The two following sections explain how the experiments were conducted.

JPEG2000 vs JPEGXL

Figure 3.2 and figure 3.3 show how the compression experiment for lossy JPEG 2000 and lossy JPEG XL, respectively, was conducted. Since the goal is to compare the two codecs, the experiments are designed to be as similar as possible. The experiments only differ by one step.

First, all the samples in the images are divided by 16 and rounded to the nearest integer. This step reduces the dynamic range of the images. The next step is to transform the images using the Karhunen-Loeve transform along the spectral axis. Then a constant is added to all of the transform coefficients so that all of the coefficients are non-negative. For JPEG XL, the images are then flattened to a two-dimensional image before being coded by the JPEG XL coder, while for JPEG 2000, the cube of transform coefficients is coded directly. The images are coded with a range of different target bit rates. Then all of the images are decoded using their respective codecs, and the inverse transforms are applied. Lastly, the PSNR and SSIM are calculated between the reconstructions and the original images.

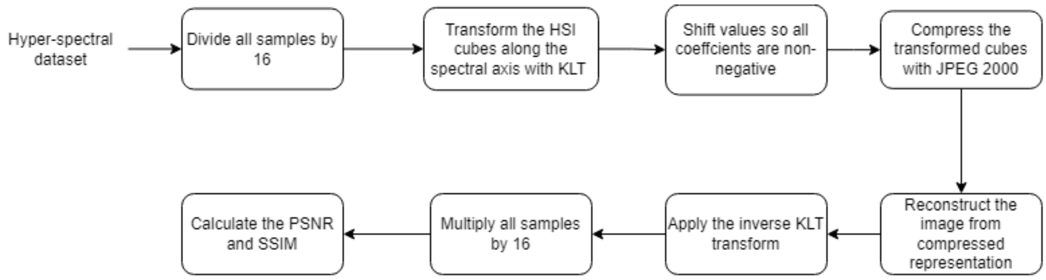
Hyper-spectral dataset
Divide all samples by 16
Transform the HSI cubes along the spectral axis with KLT
Shift values so all coeffcients are non-negative
Compress the transformed cubes with JPEG 2000
Reconstruct the image from compressed representation
Apply the inverse KLT transform
Multiply all samples by 16
Calculate the PSNR and SSIM

Fig. 3.2: Diagram showing how the lossy compression experiment for JPEG 2000 was conducted.

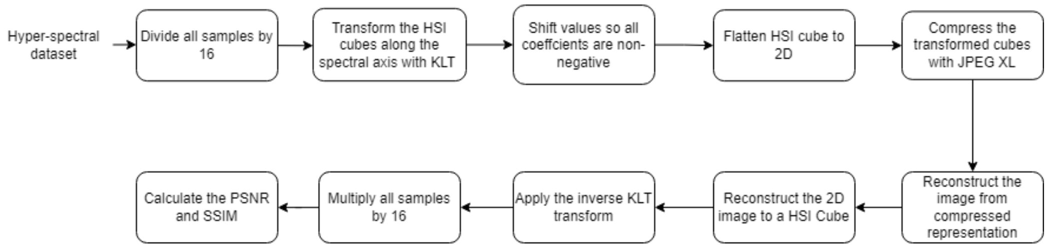
Hyper-spectral dataset
Divide all samples by 16
Transform the HSI cubes along the spectral axis with KLT
Shift values so all coeffcients are non-negative
Flatten HSI cube to 2D
Compress the transformed cubes with JPEG XL
Reconstruct the image from compressed representation
Reconstruct the 2D image to a HSI Cube
Apply the inverse KLT transform
Multiply all samples by 16
Calculate the PSNR and SSIM

Fig. 3.3: Diagram showing how the lossy compression experiment for JPEG XL was conducted.

Dimension reduction

The dimension reduction experiments compress the image by removing the dimensions of the Karhunen-Loeve transformed representation with the least energy. This

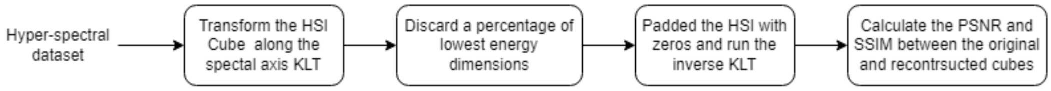
Hyper-spectral dataset
Transform the HSI Cube along the spectal axis KLT
Discard a percentage of lowest energy dimensions
Padded the HSI with zeros and run the inverse KLT
Calculate the PSNR and SSIM between the original and recontrsucted cubes

Fig. 3.4: A diagram showcasing the steps in the dimension reduction experiment.

should give the theoretically lowest possible mean square error as long as the sources are jointly Gaussian distributed [16].

Figure 3.4 shows the steps of dimension reduction experiments. First, the lossy Karhunen-Loeve transform is applied along the spectral axis. Then a percentage of the dimensions with the least energy a removed. The percentages from $10 - 90$ with an increment of 10 were used as the dimension reduction percentage. This truncated set of KLT coefficients would be the analog of the compressed representation in other lossy compression experiments. The truncated coefficient cube is padded back to the original size with zeros to decompress the image, and the inverse KLT is applied. Lastly, the SSIM and PSNR are calculated between the original and reconstructed images.

3.1.3 GICI's Experiments Notebook

To conduct the experiments, the experiments notebook was developed by the Group on Interactive Coding of Images, or GICI for short. GICI belongs to the Department of Information and Communications at the Universitat Autonoma de Barcelona. The notebook has wrappers for the different codecs, which allows for easier use and gives a framework for handling all of the image files. All of the code is open source which allows developers to adapt the software to their needs. We added features to the notebook to allow for transforms before the images are compressed, and the corresponding inverse transforms after decompression.

We thank the Group of Interactive Coding of Images for allowing others to use and modify their code. In addition, we thank them for supplying good documentation and user manuals with their programs.

3.1.4 Transforms

This section highlights the implementations of the transforms used in the compression experiments and how to evaluate their effectiveness in removing redundancy in the images.

Implementations

The Karhunen-Loeve transform used is the spectral transform software published by GICI [17]. The implementation can do both the Reversible integer-to-integer approximation or the lossy versions of the KLT. In addition, the covariance subsampling is a user-defined option for the transform. The implementation is coded in Java, which gives an acceptable performance and platform independence. The transform is 1-dimensional and therefore transforms one spectral pixel at a time.

GICI also publishes the Pairwise Orthogonal transform used. The software implements the algorithm described in [8]. The software is written C++ and source code is available for download on [17]. For the ISO-range Pairwise Orthogonal transform the aforementioned POT transform was modified to the algorithm described in [5]. The main modification is adding a shift matrix when applying the transforms.

The wavelet transforms were implemented in Python, with lifting scheme implementation [43]. The difference between the CDF 5/3 wavelet and the CDF 9/7 wavelet is the level filtering and the coefficients. The implementation is compact enough that code shown in listing 16.

```
def iwt53(c):
    s = c[0::2]
    d = c[1::2]
    l = len(s)

    a = d[0:l-1] - np.floor(0.5*(s[0:l-1]+s[1:l]))
    b = d[l-1] - s[l-1]
    d = np.concatenate((a, b), axis=None)

    a = s[0] + np.floor(0.5*d[0] + 0.5)
    b = s[1:l] + np.floor(0.25*(d[1:l] + d[0:l-1]) + 0.5)
    s = np.concatenate((a, b), axis=None)

    return s, d
```

Listing 3.1: CDF 5/3 integer wavelet transform implemented with lifing scheme in Python.

Entropy evaluation

To evaluate the ability of the transforms to decorrelate the spectral images independently from the encoding, it is possible to compare the entropy of the image before and after we apply the transforms. The same method was used in [27]. The probability density function of the image is estimated by calculating the histograms of the images and then dividing the frequency of the values by the total amount of samples. The probability density function of the image is estimated by calculating the histograms of the images and then using the equation 2.8. The histogram is calculated for the whole cube, not for each spectral band. This essentially is the same entropy as the encoder sees from its perspective if all of the samples are encoded by the same entropy encoder [42]. Finally, employ the equation 2.8 to calculate the entropy of the image. We can use the amount the transforms reduce the entropy as an indication of how well the transforms would work in a compression scheme. However, there is no guarantee that all reductions in entropy will create equal reductions in bits per sample in the compressed image.

3.1.5 Codecs

This section highlights the software behind the codecs used for the compression experiments.

CCSDS-123 and CCSDS-121

For CCSDS-123 and CCSDS-121 the Centre National d'études spatiales CNES, which is the French national center for space studies, publishes the implementations used. The center is the largest space agency in France. They do not publish the source code, but only the program binaries for Linux. The implementations are up to date with the standards of both CCSDS-123.0-b-2 and CCSDS-121-0-b-3. The standards allow for near-lossless compression but since the codecs were only used for lossless compression, these features are not used.

JPEG LS

The JPEG-LS implementation used was the reference implementation given in libjpeg reference software. The implementation is open source and with published with the GNU public license. The codec can only encode 2-dimensional images, so the hyperspectral cube is flattened before being coded by JPEG-LS. In addition, the standard is only for images with 8-bit or 16-bit unsigned samples, which has been solved for signed transform coefficients by shifting all the coefficients by a constant of at least the absolute value of the minimum value of the set of transform coeffcients.

JPEG 2000

The Kakadu software was used for compressing images to the JPEG 2000 standard. The software is proprietary but available for free for non-commercial purposes. The software is owned by Kakadu Software Pty Ltd and developed by Professor David Taubman. The codec can code images with an arbitrary amount of components and images with unsigned 8-bit and 16-bit samples. The specification claims that codec can code signed values and 32-bit, but from experimenting with the codec, we found that the performance was not satisfactory for these images.

JPEG XL

The JPEG XL reference software was used as the JPEG XL codec for the experiments. For the lossless experiments, the tools from the release of version 0.8.1 was used and for the lossy experiments, the tools from the release of version 0.6.1 was used. The older release was used for lossy experiments because it had the option of setting a target bit rate for the compression. All of the default settings for compression were used. We experimented with using a higher effort setting, but the results showed that a higher effort setting than 7 would not yield any compression performance while being significantly slower. The JPEG XL tools do not encode raw images; therefore, the images were flattened and coded as a PNG before being coded by JPEG XL tools.

3.2 Challenges

Some challenges were met when designing and running the experiments. This section outlines these challenges and describes how they were mitigated.

3.2.1 Signed Transform Coefficients

All of the transforms create transform coefficients that are signed. Which causes problems for several of the codecs. This issue was solved by shifting all of the coefficients by a constant value. The shifting of coefficients can be done in different ways. One is by shifting all of the values by 2^b, where b is the number of bits the signed data type uses, and then using a unsigned data type of the same size. This will work regardless of the value of the samples. Another way is to just shift the signed transform coefficients by the absolute value of the minimum. Both of the methods were tested without much difference being found.

3.2.2 Dynamic Range Expansion

The Karhunen-Loeve transform and the Pairwise Orthogonal transform compact the energy so much that the ranges of the values increase. For images with a high dynamic range, the samples might need bigger data types that the codecs don't support. This challenge was mitigated by reducing the dynamic range by dividing the samples by a constant and rounding the samples. This removes information, which makes the application inherently lossy.

3.2.3 Codecs Lacking Support For Multi-Component Images

The codecs made for RGB images often don't have any support for multi-component images. This is mitigated by flattening the 3D cubes into flat 2d images by appending the bands vertically. This makes it impossible for the codecs to decorrelate the images spectrally further than transforms already have.

3.3 Datasets

This section describes the data sets that are used in the experiments. One dataset of HYPSO images is used because the experiments aim to explore better transform methods for possible use in the HYPSO-1 satellite and possible future satellites. A reference dataset of images was taken from test data for the CCSDS-123 standard to check how well the results generalize to other datasets. The exact names of the images can be found in appendix 7.1.

3.3.1 HYPSO

A handful of HYPSO images were chosen for the dataset. In total 8 scenes were chosen randomly from captures of Mjøsa, Lake Balaton, Vancouver and Tampa, Florida. Figure 3.5 shows the images in the dataset. The images all have a width of 956, a height of 684 and 120 bands. The sensor captures 1080 bands with a bit depth of 12, but a binning factor of 9 is used for all the captures in the data set resulting in

120 bands and a bit depth of 16. The HYPSO-1 satellite has a push-broom imaging sensor with a spectral range of $400 - 800$ nanometers [23]. The images used were not calibrated in any way. The size of the dataset was chosen by trial and error. We experienced that having more images in the dataset had little effect on the results but took longer to process. Therefore, eight images are a suitable trade-off between accuracy and efficiency. The relatively small dataset allows us to quickly tweak the transforms and get reliable results.

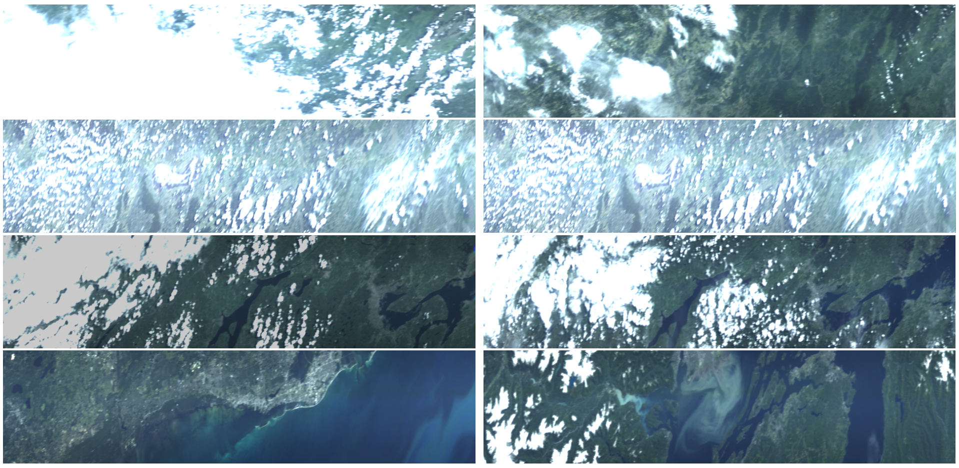

Fig. 3.5: The eight hyperspectral images from the HYPSO dataset.

3.3.2 CCSDS-123 Test Dataset

Table 3.1 shows the width, height, number of bands, and the dynamic range in bits of the six images in the reference dataset. The images are from 5 five different sensors and have a big variation in size and number of bands. The five sensors are Aviris [22], Hyperion [14], CRISM [33], CASI [2], and SFSI [34].

Tab. 3.1: Table with the width, height, number of bands and the dynamic range for the images in the CCSDS-123 test dataset

Scene	**Width**	**Height**	**Bands**	**Dynamic Range [bits]**
Aviris (Hawaii)	614	512	224	10
Aviris (Maine)	680	512	224	12
CASI (t0477f06)	406	1225	72	12
CRISM (msp0001081c)	64	2700	74	12
Hyperion (Cuprite)	256	1024	242	12
SFSI (Mantar)	496	140	240	11

Results 4

Four different experiments were conducted, entropy evaluation of the spectral decorrelation transforms, lossless compression after transforming the spectral pixels, lossy compression after applying the Karhunen-Loeve transform along the spectral axis, and finally, removing dimensions of the Karhunen-Loeve transform with the least energy. The results from these experiments are presented in this section. Most of the results are presented in tables because they allow for a more efficient presentation of the data points and make it easier to compare small differences. All of the data in the tables are presented in plots in appendix 7.2, and when convenient and we want to examine the trend of the data points, the plots are presented in this section.

4.1 Transforms

The transforms listed in section 3.1.1 are the ones evaluated in the lossless compression experiment and the ones used in the entropy evaluation. The theory is summarised in 2.2, and the implementations are stated in section 3.1.4.

4.1.1 Entropy

Table 4.1 shows the average entropy reduction of the RKLT, POT, IRPOT, CDF 5/3 and CDF 9/7. The datasets described in 3.3 are the ones on which the transforms are evaluated. The first column is missing the average entropy reduction for the RKLT and POT because these transforms expand the dynamic range to such a degree that they require 32-bit data types, which many codecs don't support and hurt the performance of others. Therefore the transforms are excluded because the entropy reduction would not indicate any possible performance gain for the lossless compression. The middle column shows the entropy reduction of the transforms on the HYPSO dataset with the dynamic range reduced. The dynamic ranges are reduced by dividing all the samples by 16, which irreversibly removes information. The transforms perform more or less as expected. The RKLT is expected to perform the best, from theory and similar experiments in the literature, and it achieves the highest entropy reduction [27], [39]. Followed by POT, which is an approximation of the KLT. The IRPOT performs a bit worse than the POT, as expected since the IRPOT is a slight modification of the POT designed not to expand the dynamic range of the images [5]. As expected, the wavelet transforms reduce the entropy significantly less than the KLT approximations since they are also significantly faster to apply.

Tab. 4.1: Table of the entropy reductions of the transforms produce on the three given datasets.

	Average entropy reduction for each of the three datasets		
Transform	**Hypso average**	**HYPSO - Reduced DR average**	**CCSDS-123 test dataset**
RKLT	x	4.355896136	4.133300601
POT	x	3.325321337	3.467811644
IRPOT	2.880440349	3.141195418	3.192334133
CDF 5/3	0.938810871	0.96008589	0.472487301
CDF 9/7	1.346026245	1.472122451	1.054725981

Table 4.2 shows the entropy reduction of the transforms on the dataset of HYPSO images with reduced dynamic range in comparison with the original dataset. This leads to the entropy reduction that occurs when reducing the dynamic range of the images is included in the results. The first row, labeled original, shows the average entropy reduction of reducing the dynamic range of the images since the value is the average entropy difference between the original images and the images with reduced dynamic range. The other entropy reductions in this table seem to be the sum of the entropy reduction from the transforms and reducing the dynamic.

Tab. 4.2: Average entropy reductions for the HYPSO dataset with reduced dynamic range with the original HYPSO datasets used as the reference. The result is that the entropy reduction of reducing the dynamic range is included.

Transforms	Average
Original	3.485875262
RKLT	8.014532739
POT	6.65173
IRPOT	6.24873
CDF53	4.35087
CDF97	4.83934

4.1.2 Execution Time

The execution time of transforms is shown in table 4.3. The times shown are the averages for the transforms when applied to the images in the HYPSO dataset. The HYPSO dataset was preferred instead of the CCSDS-123 dataset because the images in the HYPSO dataset all have the same size giving a better indication of how the execution time changes when being applied to an image of the same size. The results are somewhat hard to interpret because the transform implementations affect the execution times significantly. We would expect that if the implementation platform were the same for all of the transforms, we would get the order RKLT, POT/IRPOT, IRPOT/POT, CDF 9/7 and CDF 5/3 from slowest to fastest since that is what the complexity theory suggests. POT and IRPOT are interchangeable because of the similarity in the algorithm and implementation. The RKLT is indeed the slowest transform, but not by the big margin to the wavelets as theory suggests. The wavelets are implemented in Python, and the speed is therefore quite slow. The POT and IRPOT are the fastest transform partly because of the efficient C++ implementations.

The standard deviations of the execution times for most of the transforms are small and can be explained by other processes on the computer interfering with the measurement of the execution time. Except for the RKLT, which has a standard deviation of 14.33 seconds. We get a standard deviation of 5.92 if we remove one outlier image. Which still is the highest standard deviation. The experiment was run on an 8th-generation Intel i5.

Tab. 4.3: Table of the averages and standard deviations of the execution times of the transforms.

Transform	RKLT [s]	POT [s]	IRPOT [s]	CDF 5/3 [s]	CDF 9/7 [s]
Average	47.25	17.08	16.87	20.53	34.36
Standard Deviation	14.88	2.55	2.08	0.71	0.78

4.2 Lossless compression

The results of the experiment are presented in 3.1.1. The compression ratio and bits per pixel per band, BPPPB, are used to evaluate the lossless compression. All of the transforms listed in 3.1.4 and the codecs 3.1.5 are combined to achieve the results presented. For JPEG 2000, different wavelets were used for spatial decorrelation. For the codec labeled Kakadu lossless, a 5/3 reversible integer wavelet was used, and for the codec labeled Kakadu lossless 9/7, a 9/7 reversible integer wavelet was used. A Haar wavelet was used for the codec labeled Kakadu lossless Haar. While the codec labeled Kakadu MCT lossless uses a reversible integer 5/3 wavelet to decorrelate both spatial and spectrally.

4.2.1 No Transform Experiment

Table 4.4 presents the compression ratio and bits per pixel per band, or BPPPB, of the different datasets compressed with different codecs when no spectral decorrelation transform is applied. On the HYPSO data set, all of the transforms codecs perform more or less equally, except for the CCSDS-121, which performs notably worse. It is also worth noting that JPEG XL and JPEG-LS both slightly outperform the CCSDS-123 standard. However, the CCSDS-123 is the clear winner on the CCSDS-123 test dataset. On the HYPSO dataset with reduced dynamic range the bits per pixel per band are about half of what they are the normal HYPSO data set.

Tab. 4.4: Table showing the compression ratio and the bits per pixel per band for the different codecs and datasets when no transform is applied.

	HYPSO		HYPSO Reduced DR		CCSDS-123 Test Dataset	
Codec	CR	BPPPB	CR	BPPPB	CR	BPPPB
CCSDS 121.0-B-3 J64	1.51	10.70	1.81	6.83	1.83	6.49
CCSDS 123.0-B-2 Hybrid	1.98	8.14	3.01	4.09	3.02	4.06
JPEG XL	2.05	7.95	3.04	4.06	2.60	4.62
JPEG-LS	2.05	8.0	2.81	4.42	2.41	4.94
Kakadu MCT lossless	1.96	8.34	2.74	4.54	2.30	5.16
Kakadu lossless	1.96	8.34	2.74	4.54	2.29	5.16
Kakadu lossless 9/7-M	1.94	8.39	2.75	4.51	2.24	5.26
Kakadu lossless Haar	1.84	8.86	2.43	5.13	2.22	5.33

4.2.2 RKLT

The compression ratios and bits per pixel per band for the RKLT lossless compression experiment are presented in table 4.5. The most notable part about the results is that the RKLT only improves the bits per pixel per band metric slightly for the codecs in the experiment, or not at all. Except for the CCSDS-121, which has marked improvement and achieves comparable performance to the other codecs. In addition, the different JPEG 2000 Kakadu codecs improve from about 5 bpppc for the CCSDS-123 dataset to around 4 BPPPB, a 20 % improvement. These results are worse than expected since the RKLT is the most computationally expensive transform, and the entropy reduction indicated that it was the best at removing redundancy.

It is worth noting that the compression ratio metric is somewhat misleading for this experiment. Compression ratio calculation uses the dynamic range in bits as the before size of the image. Since the RKLT expands the dynamic range, the compression ratios are higher in this experiment, even for the same BPPPB.

Tab. 4.5: Table showing the compression ratio and the bits per pixel per band for the different codecs and datasets when RKLT along the spectral axis.

	HYPSO Reduced DR		CCSDS-123 Test Dataset	
Codec	CR	BPPPB	CR	BPPPB
CCSDS 121.0-B-3 J64	3.25	4.64	3.00	4.98
CCSDS 123.0-B-2 Hybrid	3.54	4.27	3.48	4.12
JPEG XL	3.47	4.36	3.34	4.46
JPEG-LS	3.61	4.18	3.53	4.04
Kakadu MCT lossless	3.67	4.12	3.48	4.13
Kakadu lossless	3.67	4.12	3.48	4.13
Kakadu lossless 9/7-M	3.54	4.26	3.31	4.31
Kakadu lossless Haar	3.59	4.21	3.51	4.10

4.2.3 POT

Table 4.6 shows the results of using the POT as the spectral decorrelation transform in the lossless compression experiment. The results show that average performance is slightly worse for all of the codecs on the CCSDS-123 test dataset and slightly better for JPEG-LS and JPEG 2000 Kakadu codecs for the HYPSO dataset, with the dynamic range reduced.

The same issue with the compression ratios also occurs with the results for the POT as the RKLT. Therefore the bit per pixel per band metric is used to compare the results to the other transforms.

4.2.4 IRPOT

The IRPOT was also used as a spectral decorrelation transform in the lossless compression experiment and the results are presented in the table 4.7. The most interesting results to compare to these results are the results from the POT lossless

Tab. 4.6: Table showing the compression ratio and the bits per pixel per band for the different codecs and datasets when the POT is applied.

	HYPSO Reduced DR		CCSDS-123 Test Dataset	
Codec	CR	BPPPB	CR	BPPPB
CCSDS 121.0-B-3 J64	3.18	4.79	2.87	5.26
CCSDS 123.0-B-2 Hybrid	3.52	4.29	3.39	4.25
JPEG XL	3.52	4.34	3.13	4.70
JPEG-LS	3.86	3.94	3.50	4.07
Kakadu MCT lossless	3.86	3.95	3.38	4.23
Kakadu lossless	3.86	3.95	3.38	4.23
Kakadu lossless 9/7-M	3.74	4.08	3.24	4.39
Kakadu lossless Haar	3.70	4.12	3.38	4.23

compression experiment since the algorithms are essentially the same except the IRPOT expands the dynamic range less [5]. The results are very similar across the board, just slightly worse performance for the IRPOT. Compared to results from the no-transform experiment the IRPOT increases the performance slightly for the JPEG 2000 codecs and more considerably for the JPEG-LS codec.

Tab. 4.7: Table showing the compression ratio and the bits per pixel per band for the different codecs and datasets when the IRPOT is applied.

	HYPSO		HYPSO Reduced DR		CCSDS-123 Test Dataset	
Codec	CR	BPPPB	CR	BPPPB	CR	BPPPB
CCSDS 121.0-B-3 J64	1.85	8.71	2.55	4.89	2.29	5.43
CCSDS 123.0-B-2 Hybrid	1.99	8.06	2.97	4.16	2.86	4.22
JPEG XL	2.00	8.14	2.86	4.37	2.51	4.85
JPEG-LS	2.16	7.49	3.12	3.97	2.82	4.22
Kakadu MCT lossless	2.08	7.78	3.13	3.98	2.73	4.37
Kakadu lossless	2.08	7.78	3.13	3.98	2.73	4.37
Kakadu lossless 9/7-M	2.04	7.96	3.04	4.10	2.63	4.53
Kakadu lossless Haar	2.04	7.95	3.00	4.15	2.71	4.40

4.2.5 CDF 5/3

Table 4.8 shows the results from using the CDF 5/3 wavelet transform as a spectral decorrelator in a lossless compression experiment. The results show that performance compared to the CDF 9/7 wavelet is similar, but worse. Compared to the results when no transform is applied, the transform improves the results for JPEG 2000 and JPEG-LS, but hurts the performance for JPEG XL. For the CCSDS-123, the results are more or less unchanged.

Tab. 4.8: Table showing the compression ratio and the bits per pixel per band for the different codecs and datasets when CDF 5/3 wavelet transform is applied.

	HYPSO		HYPSO Reduced DR		CCSDS-123 Test Dataset	
Codec	CR	BPPPB	CR	BPPPB	CR	BPPPB
CCSDS 121.0-B-3 J64	1.72	9.43	2.25	5.56	2.23	5.90
CCSDS 123.0-B-2 Hybrid	1.95	8.31	3.05	4.06	3.22	4.07
JPEG XL	1.78	9.16	2.45	5.18	2.31	5.62
JPEG-LS	2.16	7.57	3.08	4.05	2.87	4.45
Kakadu MCT lossless	2.05	7.93	3.05	4.11	2.75	4.62
Kakadu lossless	2.05	7.93	3.05	4.11	2.75	4.62
Kakadu lossless 9/7-M	2.01	8.09	2.99	4.17	2.67	4.76
Kakadu lossless Haar	1.99	8.18	2.83	4.42	2.70	4.70

Tab. 4.9: Table showing the compression ratio and the bits per pixel per band for the different codecs and datasets when the CDF 9/7 wavelet transform is applied

	HYPSO		HYPSO Reduced DR		CCSDS-123 Test Data set	
Codec	CR	BPPPB	CR	BPPPB	CR	BPPPB
CCSDS 121.0-B-3 J64	1.75	9.00	2.33	5.16	2.27	5.47
CCSDS 123.0-B-2 Hybrid	2.08	7.52	3.31	3.60	3.42	3.62
JPEG XL	1.81	8.77	2.57	4.75	2.37	5.17
JPEG-LS	2.20	7.20	3.27	3.68	2.96	4.04
Kakadu MCT lossless	2.12	7.47	3.25	3.70	2.83	4.20
Kakadu lossless	2.12	7.47	3.25	3.70	2.83	4.20
Kakadu lossless 9/7-M	2.08	7.58	3.19	3.76	2.73	4.34
Kakadu lossless Haar	2.03	7.78	2.99	4.03	2.77	4.29

4.2.6 CDF 9/7

Table 4.9 shows the results of the CDF 9/7 wavelet transform experiment. Notably, it beats the no-transform case for all of the codecs. For the CCSDS-123, the wavelet transform reduces the BPPPB from 8.14 to 7.52 on the HYPSO dataset and from 4.06 to 3.62 on the CCSDS-123 dataset. Which is an improvement of 7.6 % and 10.8 %, respectively. In addition, it achieves the best overall performance on all of the datasets, with JPEG-LS for the HYPSO datasets and the CCSDS-123 for the CCSDS-123 test dataset. The transforms also have very strong performance on the JPEG 2000 Kakadu codecs.

4.3 Lossy compression

This section presents the results from the lossy compression experiments presented in 3.1.2. The section 3.1.4 describes the implementation for the Karhunen-Loeve transform in both of the compression experiments.

4.3.1 JPEG XL vs JPEG 2000

The results from this section are from the experiment described in 3.1.2. In addition, the codec implementations described in 3.1.5 and 3.1.5 are used for respectively JPEG 2000 and JPEG XL. The HYPSO dataset and CCSDS-123 test dataset were used to conduct the experiment. Both PSNR and SSIM were used to measure the performance of the lossy compression. The target bit rates chosen were $, 0.05, 0.25, 0.5, 0.75, 1, 1.5, 2, 2.5$ and 3. They were chosen to cover the entire possible range evenly, with a bit higher concentration close to zero. For the CCSDS-123 dataset, the 3 target bit rate was excluded because, for some images, lossless compression was achieved, which gives an infinite PSNR.

The Karhunen-Loeve transform was applied with a set of different settings. The covariance matrix was subsampled with a subsampling rate $es = 1$, 0.1 and 0.01. In addition, both the reversible and lossy Karhunen-Loeve transform were used. The result of all these combinations of settings is presented in the tables in the section below.

Lossless KLT

The tables 4.10, 4.11,4.12 and 4.13 present the results from this experiment when reversible Karhunen-Loeve transform is used. Table 4.10 and table 4.11 show the PSNR and SSIM, respectively, for the HYPSO dataset. Table 4.12 and 4.13 show the same for the CCSDS-123 test dataset. The JPEG 2000 stands out as the winner for practically all the different settings and bit rates. Especially for the PSNR, the difference is noticeable.

Subsampling has little to no effect on the quality of the reconstruction. Both the PSNR and SSIM metrics are very similar for the three different rates of subsampling. One exception is the SSIM for the bit-rate 0.05 for the HYPSO dataset, where there is a 0.14 difference in SSiM. But the value seems like an outlier because the SSIM is higher than for the bit rate of 3 for JPEG XL.

The SSIMs for the HYPSO dataset are very high for all of the bit rates compared to the CCSDS-123 test data set for similar PSNRs, particularly for the JPEG XL codec.

Tab. 4.10: Table with the average PSNR after reconstructing the images in the HYPSO dataset compressed with JPEG 2000 or JPEG XL. The images are transformed with the reversible KLT, with different subsampling (es) rates, before being compressed by the codecs.

	Target bit rates								
Codec (es)	0.05	0.25	0.5	0.75	1	1.5	2	2.5	3
Kakadu (0.01)	45.50	54.38	57.39	58.85	59.78	61.29	62.48	63.55	64.37
JPEG XL (0.01)	25.28	44.80	46.66	48.53	51.54	52.68	55.78	57.86	62.06
Kakadu (0.1)	45.54	54.49	57.47	58.88	59.82	61.40	62.64	63.79	64.65
JPEG XL (0.1)	25.02	45.20	46.25	48.54	51.02	52.31	55.93	57.72	62.35
Kakadu (1)	45.58	54.38	57.41	58.83	59.77	61.36	62.77	63.81	64.74
JPEG XL (1)	24.92	44.33	46.18	49.89	50.79	52.09	55.54	57.80	62.14

Tab. 4.11: Table with the average SSIM after reconstructing the images in the HYPSO dataset compressed with JPEG 2000 or JPEG XL. The images are transformed with the reversible KLT, with different subsampling rates (es), before being compressed by the codecs.

	Target bit rates								
Codec (es)	0.05	0.25	0.5	0.75	1	1.5	2	2.5	3
Kakadu (0.01)	0.9651	0.9923	0.9949	0.9956	0.9958	0.9962	0.9966	0.9970	0.9973
JPEG XL (0.01)	0.7582	0.9603	0.9675	0.9800	0.9844	0.9848	0.9848	0.9932	0.9961
Kakadu (0.1)	0.9648	0.9923	0.9950	0.9957	0.9960	0.9964	0.9968	0.9971	0.9974
JPEG XL (0.1)	0.7550	0.9590	0.9670	0.9804	0.9843	0.9846	0.9919	0.9935	0.9963
Kakadu (1)	0.9963	0.9923	0.9950	0.9957	0.9960	0.9963	0.9967	0.9971	0.9974
JPEG XL (1)	0.9974	0.9568	0.9661	0.9810	0.9848	0.9847	0.9917	0.9935	0.9963

Tab. 4.12: Table with the average PSNR after reconstructing the images in the CCSDS-123 test dataset compressed with JPEG 2000 or JPEG XL. The images are transformed with the reversible KLT, with different subsampling rates (es), before being compressed by the codecs.

	Target bit rates							
Codec (es)	0.05	0.25	0.5	0.75	1	1.5	2	2.5
Kakadu (0.01)	35.85	45.03	46.17	47.64	48.68	50.19	51.37	52.80
JPEG XL (0.01)	32.39	39.36	44.23	46.47	41.68	43.92	47.19	50.03
Kakadu (0.1)	37.60	45.23	46.44	47.93	49.06	50.45	51.67	53.46
JPEG XL (0.1)	36.71	40.15	42.09	42.83	41.53	45.39	48.83	48.63
Kakadu (1)	37.60	45.23	46.44	47.93	49.06	50.45	51.67	53.46
JPEG XL (1)	36.71	40.15	42.09	42.83	41.53	45.39	48.83	48.63

Tab. 4.13: Table with the average SSIM after reconstructing the images in the CCSDS-123 test dataset compressed with JPEG 2000 or JPEG XL. The images are transformed with the reversible KLT, with different subsampling rates (es), before being compressed by the codecs.

	Target bit rates							
Codec (es)	0.05	0.25	0.5	0.75	1	1.5	2	2.5
Kakadu (0.01)	0.7232	0.9347	0.9430	0.9476	0.9488	0.9523	0.9572	0.9589
JPEG XL (0.01)	0.6905	0.8304	0.8556	0.8812	0.9029	0.9334	0.9510	0.9579
Kakadu (0.1)	0.7233	0.9341	0.9435	0.9479	0.9512	0.9551	0.9600	0.9633
JPEG XL (0.1)	0.6840	0.8431	0.8663	0.8855	0.9059	0.9345	0.9528	0.9627
Kakadu (1)	0.7271	0.9364	0.9494	0.9539	0.9582	0.9620	0.9676	0.9693
JPEG XL (1)	0.6896	0.8437	0.8693	0.8846	0.9068	0.9396	0.9558	0.9637

Lossy KLT

The four tables 4.14, 4.15, 4.16 and 4.17 show the results from this experiment with the lossy irreversible Karhunen-Loeve transform. The tables show the PSNR and SSIM for the HYPSO dataset and for the CCSDS-123 dataset in that order. Again the JPEG 2000 codec is the clear winner for most of the scenarios. The lossy transforms perform better than the lossless version, resulting in a higher-quality reconstruction according to PSNR and SSIM metrics.

The results indicate again that the subsampling does not affect the performance of the lossy compression. This could allow a more low-complexity KLT to be used for lossy compression.

Tab. 4.14: Table with the average PSNR after reconstructing the images in the HYPSO dataset compressed with JPEG 2000 or JPEG XL. The images are transformed with the irreversible KLT, with different subsampling rates (es), before being compressed by the codecs.

	Target bit rates								
Codec (es)	0.05	0.25	0.5	0.75	1	1.5	2	2.5	3
Kakadu (0.01)	45.56	54.51	57.65	59.24	60.33	62.16	64.08	65.84	67.55
JPEG XL (0.01)	25.60	44.52	46.81	48.78	51.53	52.26	56.57	58.46	63.72
Kakadu (0.1)	45.47	54.52	57.73	59.29	60.36	62.28	64.16	65.9976	67.6278
JPEG XL (0.1)	24.90	45.56	46.85	50.00	51.56	51.04	56.61	58.52	63.66
Kakadu (1)	45.59	54.51	57.66	59.23	60.32	62.29	64.12	65.85	67.54
JPEG XL (1)	25.02	46.09	45.52	49.85	51.53	50.80	56.02	58.74	62.89

Tab. 4.15: Table with the average SSIM after reconstructing the images in the HYPSO dataset compressed with JPEG 2000 or JPEG XL. The images are transformed with the irreversible KLT, with different subsampling (es) degrees, before being compressed by the codecs.

	Target bit rates								
Codec (es)	0.05	0.25	0.5	0.75	1	1.5	2	2.5	3
Kakadu (0.01)	0.9656	0.9931	0.9958	0.9965	0.9968	0.9973	0.9977	0.9982	0.9986
JPEG XL (0.01)	0.7591	0.9600	0.9683	0.9808	0.9854	0.9854	0.9926	0.9945	0.9974
Kakadu (0.1)	0.9654	0.9931	0.9958	0.9965	0.9969	0.9973	0.9978	0.9982	0.9987
JPEG XL (0.1)	0.7559	0.9591	0.9678	0.9812	0.9851	0.9851	0.9926	0.9947	0.9975
Kakadu (1)	0.9654	0.9931	0.9958	0.9965	0.9969	0.9973	0.9978	0.9982	0.9987
JPEG XL (1)	0.7571	0.9604	0.9667	0.9808	0.9854	0.9850	0.9927	0.9948	0.9975

Tab. 4.16: Table with the average PSNR after reconstructing the images in the CCSDS-123 test dataset compressed with JPEG 2000 or JPEG XL. The images are transformed with the irreversible KLT, with different subsampling (es) degrees, before being compressed by the codecs.

	Target bit rates							
Codec (es)	0.05	0.25	0.5	0.75	1	1.5	2	2.5
Kakadu (0.01)	35.78	45.79	47.51	49.31	50.74	52.69	54.32	56.05
JPEG XL (0.01)	32.33	39.78	43.74	43.14	42.46	45.22	49.04	53.56
Kakadu (0.1)	37.59	46.02	47.66	49.62	51.24	52.99	54.71	56.45
JPEG XL (0.1)	36.00	40.21	42.46	43.27	42.31	45.59	49.64	54.09
Kakadu (1)	36.67	45.99	47.81	47.81	51.09	52.95	54.60	56.25
JPEG XL (1)	35.89	35.89	42.56	43.31	43.30	45.42	49.41	53.84

Tab. 4.17: Table with the average SSIM after reconstructing the images in the CCSDS-123 test dataset compressed with JPEG 2000 or JPEG XL. The images are transformed with the irreversible KLT, with different subsampling (es) degrees, before being compressed by the codecs.

	Target bit rates							
Codec (es)	0.05	0.25	0.5	0.75	1	1.5	2	2.5
Kakadu (0.01)	0.7226	0.9394	0.9526	0.9587	0.9622	0.9683	0.9766	0.9847
JPEG XL (0.01)	0.6931	0.8345	0.8502	0.8953	0.9097	0.9457	0.9665	0.9783
Kakadu (0.1)	0.7239	0.9397	0.9531	0.9593	0.9626	0.9683	0.9765	0.9847
JPEG XL (0.1)	0.6839	0.8464	0.8498	0.8910	0.9125	0.9467	0.9679	0.9814
Kakadu (1)	0.7274	0.9396	0.9531	0.9592	0.9625	0.9683	0.9765	0.9846
JPEG XL (1)	0.6865	0.8447	0.8524	0.8916	0.9067	0.9473	0.9685	0.9815

4.3.2 Dimension Redcution

This section presents the results from the dimension reduction experiment that is described in section 3.1.2. Figure 4.1 shows the average PSNR and SSIM of the reconstructed images of the HYPSO dataset when the dimension reduction experiment. The x-axis shows the percentage of coefficients removed when truncating the transformed image. The same is done for the CCSDS-123 dataset and presented in figure 4.1.

If we compare these results to the other lossy compression, we find that for the HYPSO dataset, the dimension reduction produces worse quality for the same level of compression by a considerable margin. The margin for the CCSDS-123 test data set is smaller, but the same trend prevails.

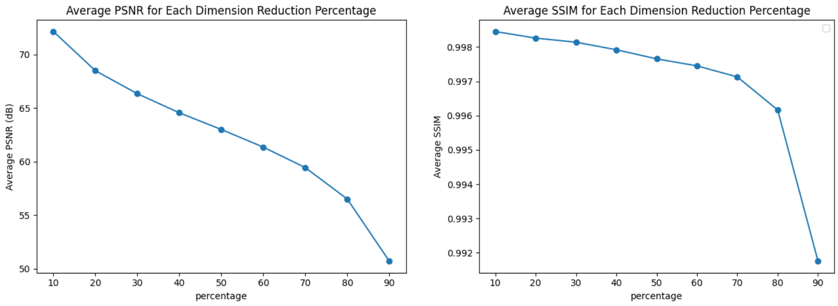

Fig. 4.1: Plots showing the average PSNR and SSIM for dimension reduction experiment for the HYSPO dataset.

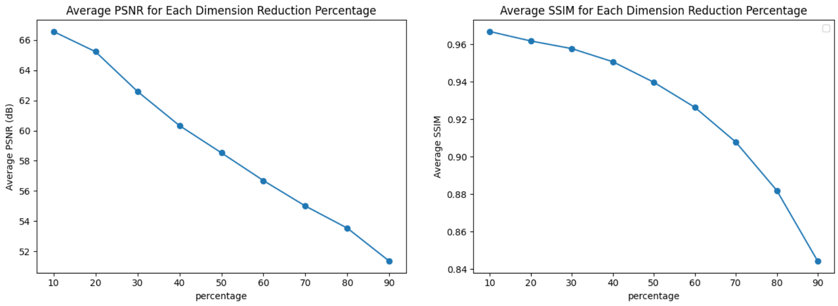

Fig. 4.2: Plots showing the average PSNR and SSIM for dimension reduction experiment for the CCSDS-123 test dataset.

5 Discussion

This section further discusses the results presented in section 4. Improvements and future work on the topic are suggested.

5.1 Lossless experiments

One interesting result we found is that applying a reversible integer CDF 9/7 wavelet transform before applying the CCSDS-123 transform improves the BPPPB achieved by 10 percent on the CCSDS-123 test dataset. Since the transform is fairly easy to compute and can be implemented in hardware [35], it could be a possible improvement to the standard. Further work should include checking if this improvement holds over all of the hyperspectral images in the CCSDS-123 test data. However, for the HYPSO dataset, the improvement was smaller in percentage at 7.68 percent and a 0.62 reduction in BPPPB on average. For the specific case of the HYPSO-1 satellite, it might not be worth it to implement a pre-processing step before employing the CCSDS-123 standard. Since there is likely more improvement to be gained from looking into near-lossless compression methods.

For lossless compression, the reversible Karhunen-Loeve transform did not perform as well as expected. When applying the RKLT as spectral decorrelator before compression the resulting BPPPCs were worse for some of the codecs that are prediction-based, CCSDS-123 and JPEG XL, compared to when no transform was applied. This is most likely because these predictors expect the images to consist of visual characteristics, such as edges and homogeneous areas. The predictor in CCSDS-123 are types of edge-detectors and a transformed image will not have clear edges [48]. However, for JPEG-LS which is also a prediction-based codec the RKLT and other energy-compating transforms did improve the coding performance. It is not clear what causes the JPEG-LS codec to be better at coding transform coefficients than CCSDS-123 when they employ very similar predictors and encoders. For the transform-based JPEG 2000 codecs the RKLT did improve coding performance to a point close to the prediction-based ones, but did not make them better.

In an attempt to make the performance of the RKLT better, it could be interesting to explore encoders and predictors that are designed to work well with the RKLT. Most of the codecs today are not designed with the RKLT in mind since it is often dismissed as too computationally expensive [39]. A way could be to use the information in the transformation matrix to choose a context or model for the encoding or prediction.

In general, ways to improve the coherence between the transforms should be explored. A weakness of the methods in this thesis is that transforms and codecs are combined with not enough emphasis on how they work together.

For the HYPSO dataset, where all images have a dynamic range of 16 bits, the images had an average BPPPB between 7 and 8. For the CCSDS-123 dataset and the HYPSO dataset with reduced dynamic range, the images have an average dynamic range of 12 bits. The average BPPPB was around 3 to 4. In addition, several different methods were used to achieve similar results. This suggests that the images, on average, have around 8 bits of redundancy, or at least the methods, both different predictors and transforms, can only remove 8 bits of redundancy. If the images only contain around 8 bits of redundancy, the BPPPBs would be close to the theoretical limit for lossless compression. This would explain why no methods seem to yield any better performance.

The JPEG XL codec does not code transform coefficients well. Of the different transforms tried, all hindered or had little effect on the performance of the JPEG XL codec. The reason for this could be that since the JPEG XL standard is highly optimized for the compression of visual images, the performance on transform coefficients is worse as a result. In contrast, JPEG 2000 is designed to also support the compression of transform coefficients [45].

The JPEG-LS codec achieves the best compression performance overall with BPPPB of 7.2 in combination with the CDF 9/7 wavelet, on the HYPSO dataset. It is somewhat unexpected that it would perform better or similarly to CCSDS-123 on hyperspectral images because the CCSDS-123 also uses information from the other spectral components for the prediction. A theory might be that when the images are spectrally decorrelated, the information from the other spectral components actually hampers the prediction leading to worse performance. However, the improvement of JPEG-LS over CCSDS-123 is not big enough that it would be unnecessary to change from the CCSDS-123 to JPEG-LS for the HYPSO mission or other satellite missions.

For the HYPSO mission specifically the most logical way to achieve better compression is to employ some near-lossless compression methods. The most natural way would be the near-lossless extension of the CCSDS-123. Another potential way can be found by comparing the BPPPB of the HYPSO dataset and the same dataset with a reduced dynamic range. The dynamic range of the samples is reduced by dividing all of by 16 and rounding the numbers to the closest integer. The samples are then multiplied by 16 when the inverse transforms are applied. The rounding to integers creates a quantization effect since the values are restricted to the nearest multiple of 16. This creates a reconstruction error of the individual samples between 0 and 8 depending on how far the closest multiple of 16 was. If we assume that the errors are distributed uniformly between 0 and 8 we can show that the average mean square error will be the mean of the sum of squares from 0 to 8 which is $\frac{68}{3}$. This mean square error gives a PSNR of 63.2 dB, which is not great for a bit rate of around 4 bits but still not bad for an extremely simple method that requires minimal extra computation. We would assume that the near-lossless extension of the CCSDS-123 would perform better.

5.2 Lossy experiments

The results of the lossy compression experiment clearly show that the JPEG 2000 is the best option for lossy compression paired with the lossy Karhunen-Loeve transform. The JPEG XL is again probably hurt by the fact that is hyper-optimized for the visual data, which hurts its performance on the transform coefficients. One of the main issues with using JPEG XL for transform coefficients is that it employs the Butteraugli distance, described in section 2.1.3, to set the level of acceptable distortion. In contrast, the JPEG 2000 codecs allow us to set the bit rate directly. In addition, JPEG 2000 is designed to work with multi-component transforms. When using the Butteraugli distance metric, which is created to optimize for the human visual system and not for transform coefficients, we found it hard to get the codec to hit the desired bit rates. In general, the Butteraugli algorithm correctly assumed that there was little visual information in the transformed images and discarded a lot of information. If one wants to try and use JPEG XL to code hyperspectral images in the future, we would suggest modifying the standard to be more efficient for transform coefficients or rather only use the ideas such as variable DCT to achieve a better coding gain [47], [1].

The covariance sub-sampling seems to have little to no effect on the performance of the lossy compression. For most use cases it would be preferred to use a KLT with subsampling employed. The degree of subsampling should be in the range between 0.1 and 0.01 because more than 0.01 does not lower the complexity of the KLT much because operations other than the covariance calculation become the dominating factor in the total execution time of the Karhunen-Loeve transform [37].

The dimension reduction experiment produced worse results compared to JPEG 2000 with lossy compression by quite a margin. A way to make the dimension reduction experiment more competitive would be losslessly compress the remaining transform coefficients or even with loss. That could be further explored as a way to efficiently compress hyperspectral images with loss.

5.3 Downsides with the Karhunen-Loeve Transform

The optimality of the Karhunen-Loeve transform is only guaranteed when working with jointly Gaussian sources. There is no guarantee that this holds for the hyperspectral images. For non-gaussian sources, a non-linear transform might be better suited for decorrelating the samples. This would be beneficial for both lossy and lossless compression of hyperspectral images.

5.4 Future Work

The section presents some ideas and suggestions for future work on the topic of lossy and lossless compression of hyperspectral images.

5.4.1 Neural Compression

Neural compression is an interesting new field in image compression. As the name suggests, the field uses neural networks to compress images. Neural networks allow for the approximation of virtually any function if the right weights are found [3]. A neural network could be used as a predictor in a lossless compression scheme, or a variational auto-encoder could be used as a spectral decorrelator for lossy compression, lossless compression and dimension reduction [49]. A potential downside would be that the auto-encoder might remove anomalies in the images if they are not present in the training data. There has already been some research done for neural compression of hyperspectral images [31], [11].

5.4.2 JPEG XL

Currently, the JPEG XL reference software has not reached a full release yet. Therefore it is not always the easiest software to work with. However, adoption of the standard is growing, and the community around the development of the standard is strong. After the release of the full reference software, it could be beneficial to try and use the reference JPEG XL library to write a program specifically for compression of hyperspectral images.

6 Conclusion

The experiments show that spectral decorrelation effectively improves the performance of compression hyperspectral images. However, for lossless compression, the improvements found were minor. The most significant improvements were a 10.8 percent improvement by spectrally decorrelating the samples with the CDF 9/7 wavelet transform before compressing with the CCSDS-123 standard. The same method achieved a 7.6 percent on the HYPSO dataset. Overall the JPEG-LS combined with the CDF 9/7 wavelet transform gave the best performance for lossless compression. Surprisingly the Reversible Karhunen-Loeve does not provide better compression than the other transforms, even though it is better at decorrelating and compacting the energy. For lossy compression, the experiments show that the combination of JPEG 2000 and the lossy Karhunen-Loeve transform gives the best performance of the methods tested. The other methods were a combination of JPEG XL and Karhunen-Loeve transform and dimension reduction by truncating the Karhunen-Loeve transform coefficients. The JPEG XL standard was likely too optimized for the human visual system to be effective for coding transform coefficients. The lossy compression experiments tested different ways to apply the Karhunen-Loeve transform. They showed the lossy version of the Karhunen-Loeve transform performed better and that the low-complexity version with subsampling performed well and only slightly worse than the full-complexity version.

Bibliography

[1] Jyrki Alakuijala, Robert Obryk, Ostap Stoliarchuk, et al. “Guetzli: Perceptually Guided JPEG Encoder”. In: *CoRR* abs/1703.04421 (2017). arXiv: `1703.04421` (cit. on pp. 11, 20, 47).

[2] SK Babey and CD Anger. “A compact airborne spectrographic imager (CASI)”. In: *Quantitative Remote Sensing: An Economic Tool for the Nineties, Volume 1* 2 (1989), pp. 1028–1031 (cit. on p. 31).

[3] Johannes Ballé, Philip A. Chou, David Minnen, et al. “Nonlinear Transform Coding”. In: *IEEE Journal of Selected Topics in Signal Processing* 15.2 (2021), pp. 339–353 (cit. on p. 48).

[4] Michel Barret, Jean-Louis Gutzwiller, Isidore Paul Akam Bita, and Florio Dalla Vedova. “Lossy Hyperspectral Images Coding with Exogenous Quasi Optimal Transforms”. In: *2009 Data Compression Conference*. 2009, pp. 411–419 (cit. on p. 16).

[5] Ian Blanes, Miguel Hernández-Cabronero, Francesc Aulí-Llinàs, Joan Serra-Sagristà, and Michael W. Marcellin. “Isorange Pairwise Orthogonal Transform”. In: *IEEE Transactions on Geoscience and Remote Sensing* 53.6 (2015), pp. 3361–3372 (cit. on pp. 27, 34, 38).

[6] Ian Blanes, Enrico Magli, and Joan Serra-Sagrista. “A Tutorial on Image Compression for Optical Space Imaging Systems”. In: *IEEE Geoscience and Remote Sensing Magazine* 2.3 (2014), pp. 8–26 (cit. on p. 7).

[7] Ian Blanes, Joan Serra-Sagrista, Michael W. Marcellin, and Joan Bartrina-Rapesta. “Divide-and-Conquer Strategies for Hyperspectral Image Processing: A Review of Their Benefits and Advantages”. In: *IEEE Signal Processing Magazine* 29.3 (2012), pp. 71–81 (cit. on p. 16).

[8] Ian Blanes and Joan Serra-Sagristà. “Pairwise Orthogonal Transform for Spectral Image Coding”. In: *Geoscience and Remote Sensing, IEEE Transactions on* 49 (Apr. 2011), pp. 961–972 (cit. on pp. 16, 27).

[9] A.R. Calderbank, Ingrid Daubechies, Wim Sweldens, and Boon-Lock Yeo. “Wavelet Transforms That Map Integers to Integers”. In: *Applied and Computational Harmonic Analysis* 5.3 (1998), pp. 332–369 (cit. on p. 17).

[10] CCSDS. *CCSDS 121.0-B-3: Lossless Data Compression*. Tech. rep. Consultative Committee for Space Data Systems (CCSDS), 2012 (cit. on p. 18).

[11] *Deep Neural Networks applied to low power onboard image compression*. Zenodo, Sept. 2022 (cit. on p. 48).

[12]R Dony et al. “Karhunen-loeve transform”. In: *The transform and data compression handbook* 1.1-34 (2001), p. 29 (cit. on p. 14).

[13]Michael T. Eismann. *Hyper-spectral Remote Sensing*. SPIE, 2012 (cit. on p. 6).

[14]Mark A Folkman, Jay Pearlman, Lushalan B Liao, and Peter J Jarecke. “EO-1/Hyperion hyperspectral imager design, development, characterization, and calibration”. In: *Hyperspectral Remote Sensing of the Land and Atmosphere* 4151 (2001), pp. 40–51 (cit. on p. 31).

[15]L. Galli and S. Salzo. “Lossless hyperspectral compression using KLT”. In: *IGARSS 2004. 2004 IEEE International Geoscience and Remote Sensing Symposium*. Vol. 1. 2004, p. 316 (cit. on p. 15).

[16]Jan J Gerbrands. “On the relationships between SVD, KLT and PCA”. In: *Pattern Recognition* 14 (1981) (cit. on pp. 13–15, 17, 26).

[17]GICI Group, Universitat Autònoma de Barcelona. *Downloads*. Accessed on: 11 July 2023. 2023 (cit. on p. 27).

[18]Solomon Golomb. “Run-length encodings (corresp.)” In: *IEEE transactions on information theory* 12.3 (1966), pp. 399–401 (cit. on p. 8).

[19]V.K. Goyal. “Theoretical foundations of transform coding”. In: *IEEE Signal Processing Magazine* 18.5 (2001), pp. 9–21 (cit. on pp. 6, 13).

[20]V.K. Goyal and M. Vetterli. “Manipulating rates, complexity and error-resilience with discrete transforms”. In: *Conference Record of Thirty-Second Asilomar Conference on Signals, Systems and Computers (Cat. No.98CH36284)*. Vol. 1. 1998, 457–461 vol.1 (cit. on p. 13).

[21]Vivek K Goyal. “High-rate transform coding: How high is high, and does it matter?” In: *IEEE International Symposium on Information Theory*. Citeseer. 2000, pp. 207–207 (cit. on p. 14).

[22]Robert O Green, Michael L Eastwood, Charles M Sarture, et al. “Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (AVIRIS)”. In: *Remote sensing of environment* 65.3 (1998), pp. 227–248 (cit. on p. 31).

[23]Mariusz E. Grøtte, Roger Birkeland, Evelyn Honoré-Livermore, et al. “Ocean Color Hyperspectral Remote Sensing With High Resolution and Low Latency—The HYPSO-1 CubeSat Mission”. In: *IEEE Transactions on Geoscience and Remote Sensing* 60 (2022), pp. 1–19 (cit. on pp. 2, 31).

[24]P. Hao and Q. Shi. “Reversible integer KLT for progressive-to-lossless compression of multiple component images”. In: *Proceedings 2003 International Conference on Image Processing (Cat. No.03CH37429)*. Vol. 1. 2003, pp. I–633 (cit. on pp. 14, 15).

[25]Christopher Heil. *Metrics, Norms, Inner Products, and Operator theory*. Birkhauser, 2018 (cit. on pp. 12, 13, 15).

[26] Miguel Hernández-Cabronero, Aaron B. Kiely, Matthew Klimesh, et al. "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* 9.4 (2021), pp. 102–119 (cit. on p. 18).

[27] Miguel Hernández-Cabronero, Jordi Portell, Ian Blanes, and Joan Serra-Sagristà. "High-Performance Lossless Compression of Hyperspectral Remote Sensing Scenes Based on Spectral Decorrelation". In: *Remote Sensing* 12.18 (2020) (cit. on pp. 2, 14, 27, 34).

[28] Yanping Huang and Rajesh PN Rao. "Predictive coding". In: *Wiley Interdisciplinary Reviews: Cognitive Science* 2.5 (2011), pp. 580–593 (cit. on p. 18).

[29] Graham Hudson, Alain Léger, Birger Niss, István Sebestyén, and Jørgen Vaaben. "JPEG-1 standard 25 years: past, present, and future reasons for a success". In: *Journal of Electronic Imaging* 27.4 (2018), p. 040901 (cit. on p. 19).

[30] David A Huffman. "A method for the construction of minimum-redundancy codes". In: *Proceedings of the IRE* 40.9 (1952), pp. 1098–1101 (cit. on p. 6).

[31] *Hyperspectral image compression using convolutional neural networks with local spectral transforms and non-uniform sample normalisation*. Zenodo, Sept. 2022 (cit. on p. 48).

[32] Harry Jones. "The recent large reduction in space launch cost". In: 48th International Conference on Environmental Systems. 2018 (cit. on p. 1).

[33] S. Murchie, R. Arvidson, P. Bedini, et al. "Compact Reconnaissance Imaging Spectrometer for Mars (CRISM) on Mars Reconnaissance Orbiter (MRO)". In: *Journal of Geophysical Research: Planets* 112.E5 (2007). eprint: `https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2006JE002682` (cit. on p. 31).

[34] Robert A. Neville, Neil Rowlands, Richard Marois, and Ian Powell. "SFSI: Canada's First Airborne SWIR Imaging Spectrometer". In: *Canadian Journal of Remote Sensing* 21.3 (1995), pp. 328–336. eprint: `https://doi.org/10.1080/07038992.1995.10874626` (cit. on p. 31).

[35] S. Padmavati, Vaibhav Meshram, and Jayadevappa. "A hardware implementation of discrete wavelet transform for compression of a natural image". In: *2017 International Conference on Algorithms, Methodology, Models and Applications in Emerging Technologies (ICAMMAET)*. 2017, pp. 1–5 (cit. on p. 45).

[36] B. Penna, T. Tillo, E. Magli, and G. Olmo. "A new low complexity KLT for lossy hyperspectral data compression". In: *IGARSS 2006. IEEE International Conference on Geoscience and Remote Sensing Symposium, 2006.* IEEE. 2006, pp. 3525–3528 (cit. on p. 15).

[37] Barbara Penna, Tammam Tillo, Enrico Magli, and Gabriella Olmo. "Transform Coding Techniques for Lossy Hyperspectral Data Compression". In: *IEEE Transactions on Geoscience and Remote Sensing* 45.5 (2007), pp. 1408–1421 (cit. on pp. 14, 15, 24, 48).

[38] Shen-En Qian. *Optical satellite signal processing and enhancement*. Jan. 2013, pp. 1–505 (cit. on pp. 9, 13).

[39] Richard E. Woods Rafael C. Gonzalez. *Digital Image Processing*. Pearson Education, Inc., 2010 (cit. on pp. 5–7, 12, 17, 34, 46).

[40] YV Ramana Rao and C Eswaran. “A new algorithm for BTC image bit plane coding”. In: *IEEE Transactions on Communications* 43.6 (1995), pp. 2010–2011 (cit. on p. 6).

[41] Jorma Rissanen and Glen G Langdon. “Arithmetic coding”. In: *IBM Journal of research and development* 23.2 (1979), pp. 149–162 (cit. on p. 6).

[42] Myung-Sin Song. “Entropy Encoding in Wavelet Image Compression”. In: Jan. 2008, pp. 293–311 (cit. on pp. 14, 28).

[43] Wim Sweldens. “Lifting scheme: a new philosophy in biorthogonal wavelet constructions”. In: *Wavelet Applications in Signal and Image Processing III*. Ed. by Andrew F. Laine and Michael A. Unser. Vol. 2569. International Society for Optics and Photonics. SPIE, 1995, pp. 68–79 (cit. on pp. 17, 27).

[44] David Taubman. “High performance scalable image compression with EBCOT”. In: *IEEE Transactions on image processing* 9.7 (2000), pp. 1158–1170 (cit. on p. 20).

[45] David S Taubman and Michael W Marcellin. *Optical satellite signal processing and enhancement*. Jan. 2004 (cit. on pp. 20, 46).

[46] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. “Image quality assessment: from error visibility to structural similarity”. In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612 (cit. on p. 10).

[47] J Wassenberg and J Sneyers. “JPEG XL Whitepaper”. In: *JPEG (ISO/IEC JTC 1/SC 29/WG 1) 87th Meeting N*. Vol. 87021. 2020 (cit. on pp. 20, 47).

[48] Marcelo J Weinberger, Gadiel Seroussi, and Guillermo Sapiro. “The LOCO-I lossless image compression algorithm: Principles and standardization into JPEG-LS”. In: *IEEE Transactions on Image processing* 9.8 (2000), pp. 1309–1324 (cit. on pp. 19, 45).

[49] Yibo Yang, Stephan Mandt, and Lucas Theis. *An Introduction to Neural Data Compression*. 2022. arXiv: `2202.06533 [cs.LG]` (cit. on p. 48).

7 Appendix

7.1 Appendix Section 1

7.1.1 HYPSO Dataset

1. balaton-2022-05-17
2. balaton-2022-05-21
3. balatonEast_2022-04-11
4. balaton_2022-04
5. mjosa_2022-06-30
6. mjosa_2022-07-08
7. tampa_2022-10-07
8. vancouvergrieg_2022_07_29T18_36_5

7.1.2 CCSDS-123 test dataset

1. Aviris (hawaii_f011020t01p03r05)
2. Aviris (maine_f030828t01p00r05)
3. CASI (t0477f06)
4. CRISM (msp0001081c)
5. Hyperion (Cuprite)
6. SFSI (Mantar)

7.2 Appendix Section 2

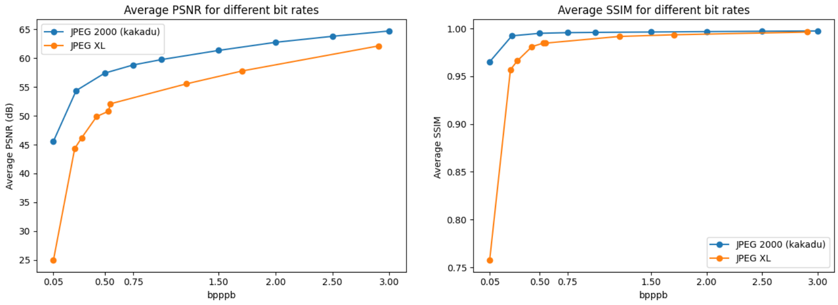

Fig. 7.1: Plots showing the achieved PSNR and SSIM for the target bit rates for JPEG 2000 and JPEG XL, the HYPSO dataset in lossy compression experiment. For this experiment, the lossless reversible Karhunen-Loeve transform was applied without subsampling.

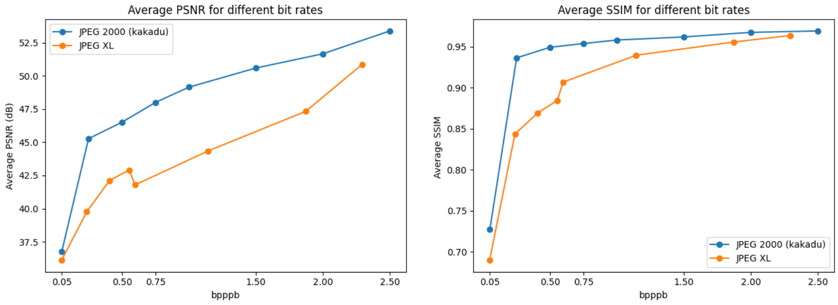

Fig. 7.2: Plots showing the achieved PSNR and SSIM for the target bit rates for JPEG 2000 and JPEG XL, the CCSDS-123 test dataset in lossy compression experiment. For this experiment, the lossless reversible Karhunen-Loeve transform was applied without subsampling.

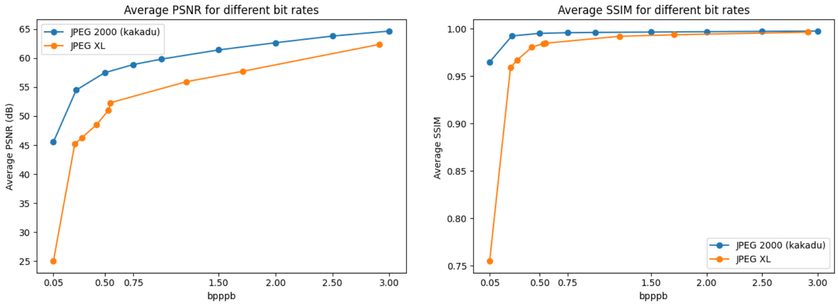

Fig. 7.3: Plots showing the achieved PSNR and SSIM for the target bit rates for JPEG 2000 and JPEG XL, the HYPSO dataset in lossy compression experiment. The lossless reversible Karhunen-Loeve transform was applied with a subsampling rate of 0.1 for this experiment.

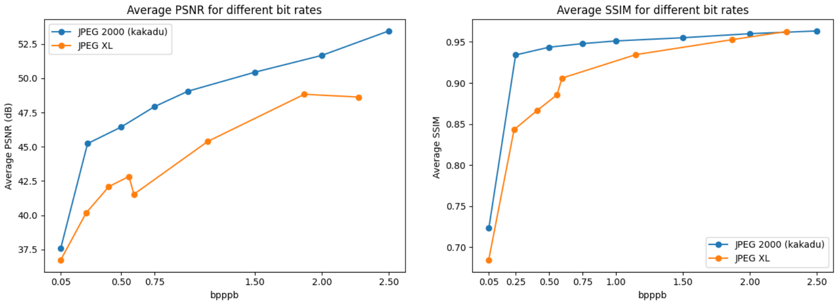

Fig. 7.4: Plots showing the achieved PSNR and SSIM for the target bit rates for JPEG 2000 and JPEG XL, the CCSDS-123 test dataset in lossy compression experiment. The lossless reversible Karhunen-Loeve transform was applied with a subsampling rate of 0.1 for this experiment.

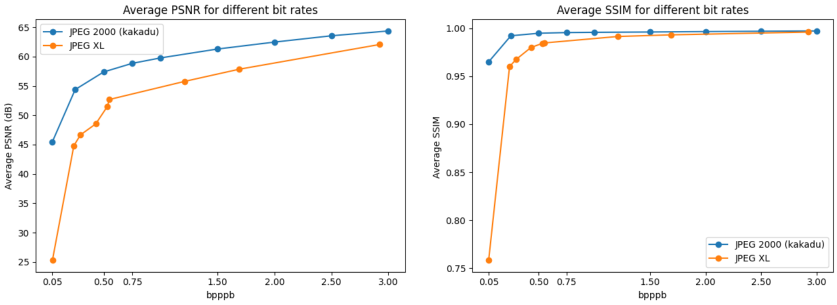

Fig. 7.5: Plots showing the achieved PSNR and SSIM for the target bit rates for JPEG 2000 and JPEG XL, the HYPSO dataset in lossy compression experiment. The lossless reversible Karhunen-Loeve transform was applied with a subsampling rate of 0.01 for this experiment.

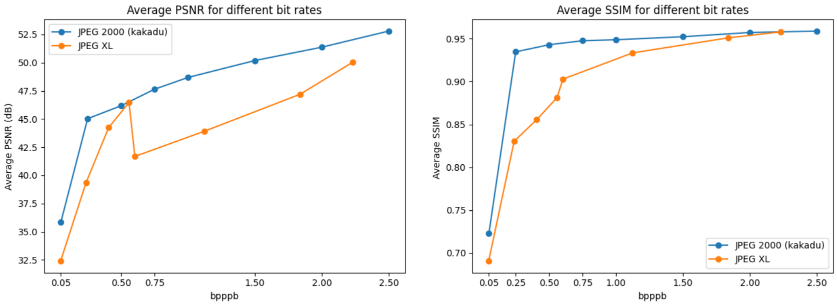

Fig. 7.6: Plots showing the achieved PSNR and SSIM for the target bit rates for JPEG 2000 and JPEG XL, the CCSDS-123 test dataset in lossy compression experiment. The lossless reversible Karhunen-Loeve transform was applied with a subsampling rate of 0.01 for this experiment.

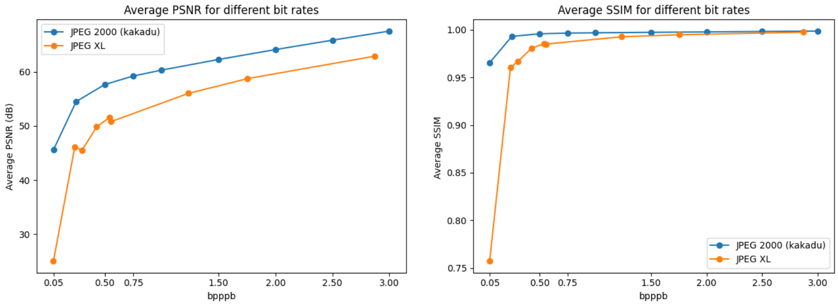

Fig. 7.7: Plots showing the achieved PSNR and SSIM for the target bit rates for JPEG 2000 and JPEG XL, the HYPSO dataset in lossy compression experiment. For this experiment, the lossy Karhunen-Loeve transform was applied without subsampling.

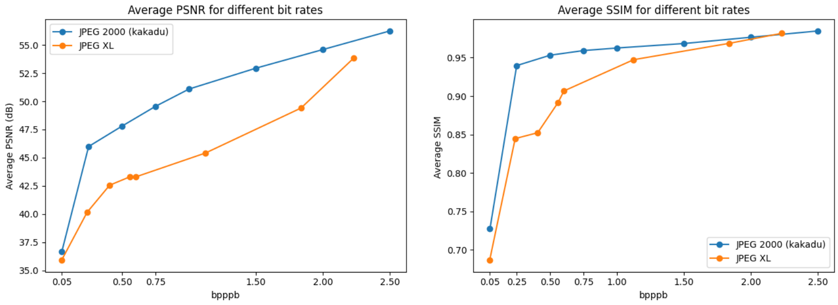

Fig. 7.8: Plots showing the achieved PSNR and SSIM for the target bit rates for JPEG 2000 and JPEG XL, the CCSDS-123 test dataset in lossy compression experiment. For this experiment, the lossy reversible Karhunen-Loeve transform was applied without subsampling.

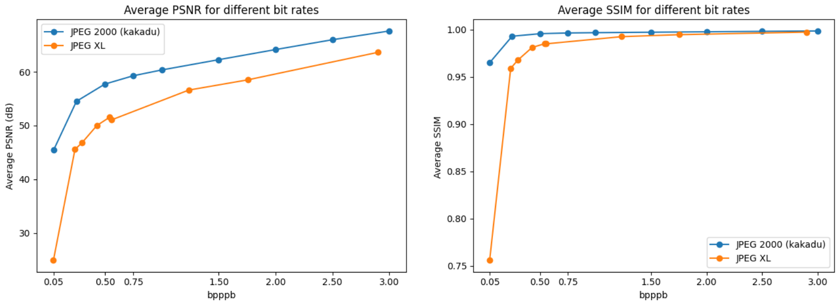

Fig. 7.9: Plots showing the achieved PSNR and SSIM for the target bit rates for JPEG 2000 and JPEG XL, the HYPSO dataset in lossy compression experiment. The lossy reversible Karhunen-Loeve transform was applied with a subsampling rate of 0.1 for this experiment.

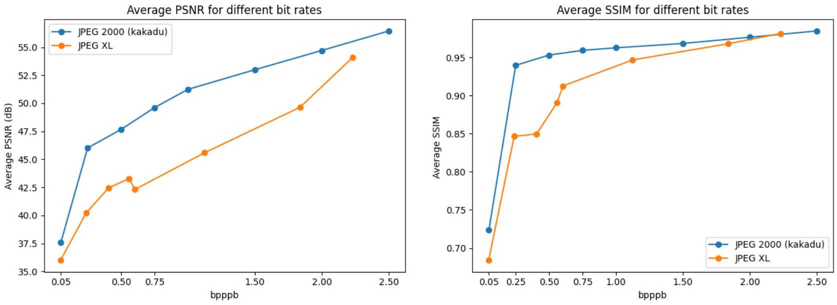

Fig. 7.10: Plots showing the achieved PSNR and SSIM for the target bit rates for JPEG 2000 and JPEG XL, the CCSDS-123 test dataset in lossy compression experiment. The lossy Karhunen-Loeve transform was applied with a subsampling rate of 0.1 for this experiment.

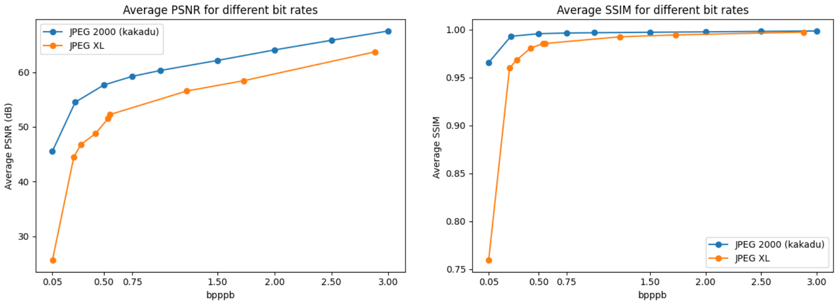

Fig. 7.11: Plots showing the achieved PSNR and SSIM for the target bit rates for JPEG 2000 and JPEG XL, the HYPSO dataset in lossy compression experiment. The lossy reversible Karhunen-Loeve transform was applied with a subsampling rate of 0.01 for this experiment.

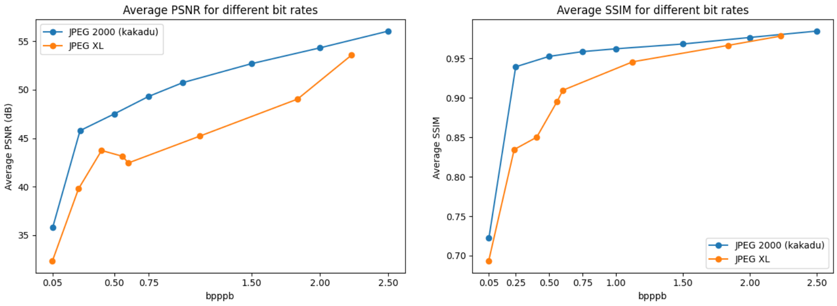

Fig. 7.12: Plots showing the achieved PSNR and SSIM for the target bit rates for JPEG 2000 and JPEG XL, the CCSDS-123 test dataset in lossy compression experiment. The lossy reversible Karhunen-Loeve transform was applied with a subsampling rate of 0.01 for this experiment.

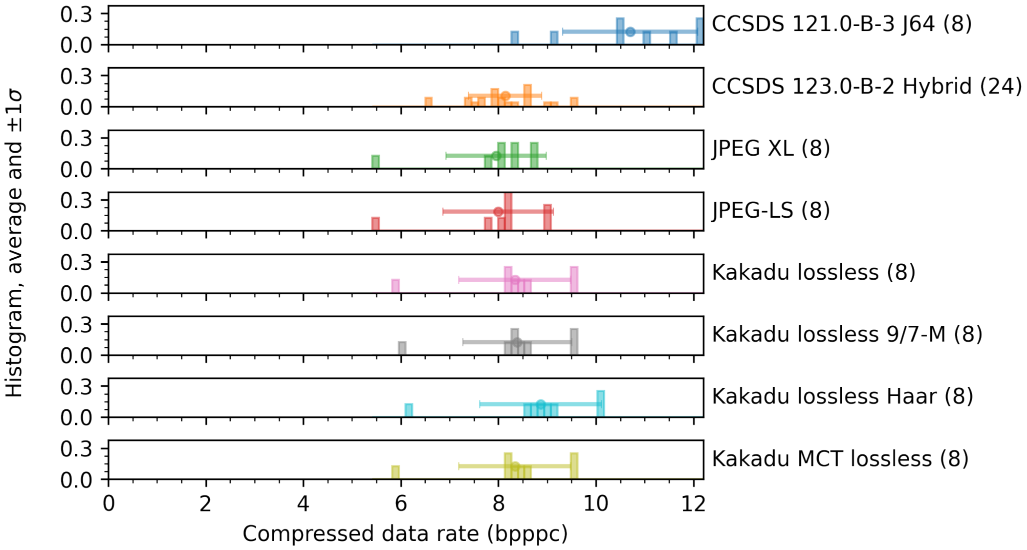

Fig. 7.13: Plot showing the BPPPB the different codecs in lossless compression achieved on the HYPSO dataset. No transform was applied before compressing.

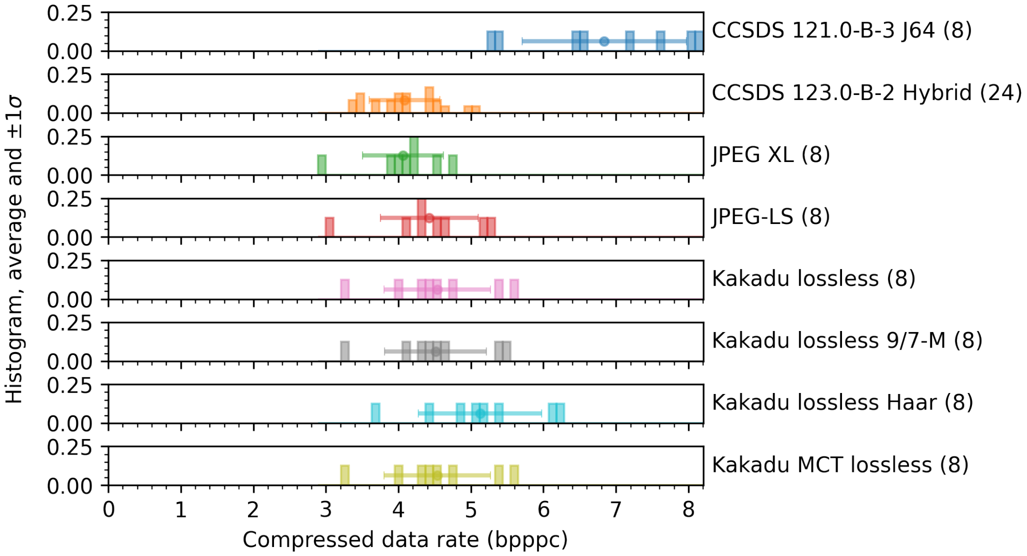

Fig. 7.14: Plot showing the BPPPB the different codecs in lossless compression achieved on the HYPSO dataset with reduced dynamic range. No transform was applied before compressing.

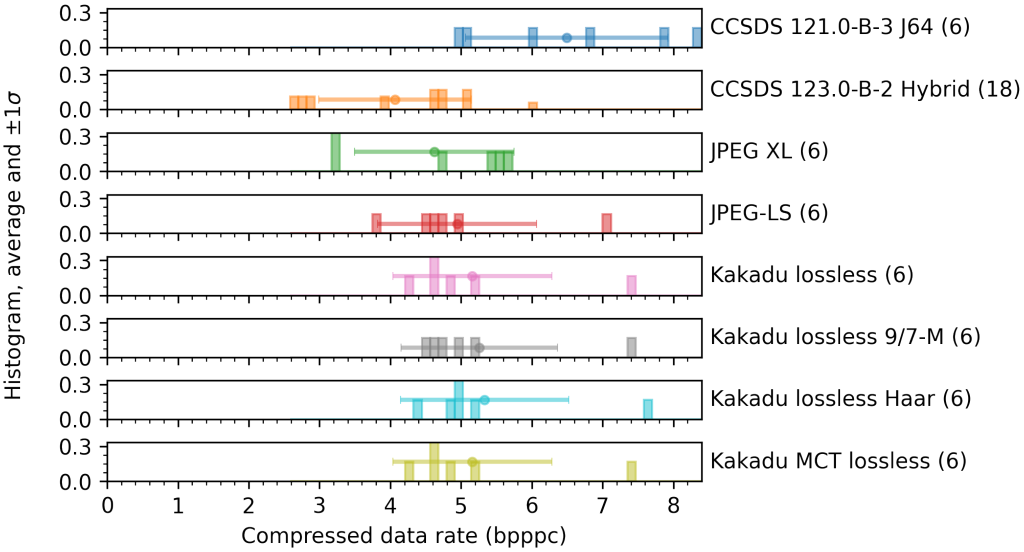

Fig. 7.15: Plot showing the BPPPB the different codecs in lossless compression achieved on the CCSDS-123 test dataset with reduced dynamic range. No transform was applied before compressing.

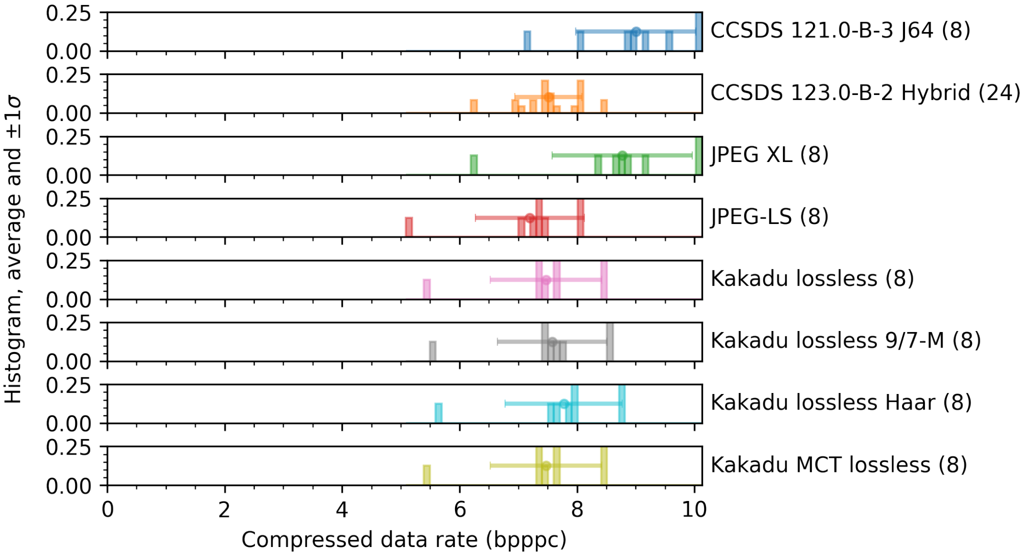

Fig. 7.16: Plot showing the BPPPB the different codecs in lossless compression achieved on the HYPSO dataset. The integer wavelet CDF wavelet transform was applied before compressing.

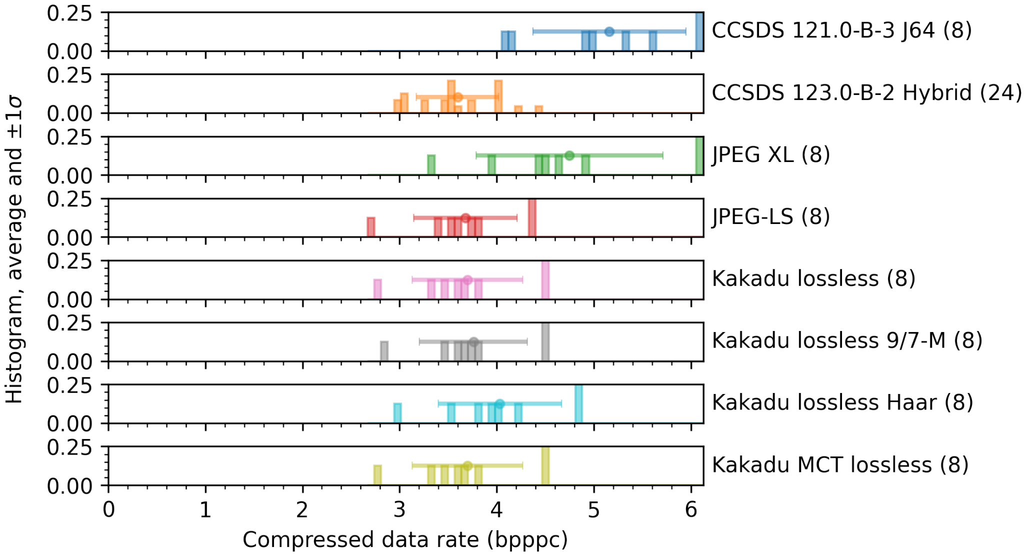

Fig. 7.17: Plot showing the BPPPB the different codecs in lossless compression achieved on the HYPSO dataset with reduced dynamic range. The integer wavelet CDF wavelet transform was applied before compressing.

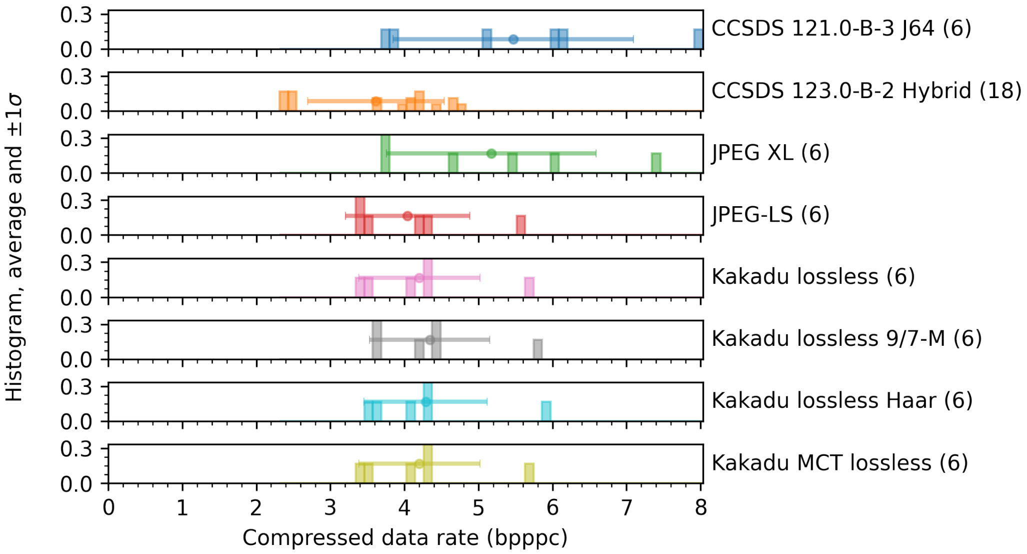

Fig. 7.18: Plot showing the BPPPB the different codecs in lossless compression achieved on the CCSDS-123 test dataset with reduced dynamic range. The integer wavelet CDF wavelet transform was applied before compressing.

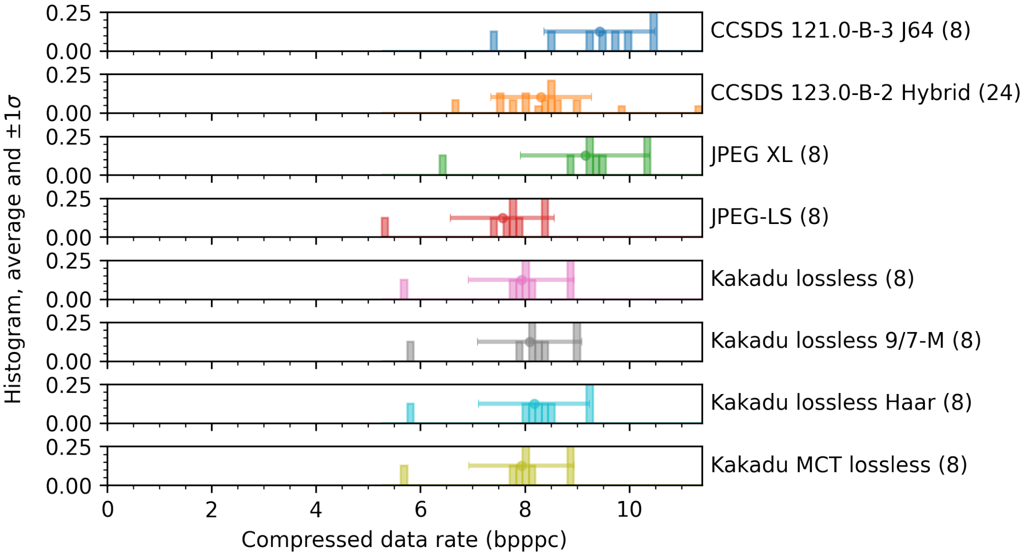

Fig. 7.19: Plot showing the BPPPB the different codecs in lossless compression achieved on the HYPSO dataset. The integer wavelet CDF wavelet transform was applied before compressing.

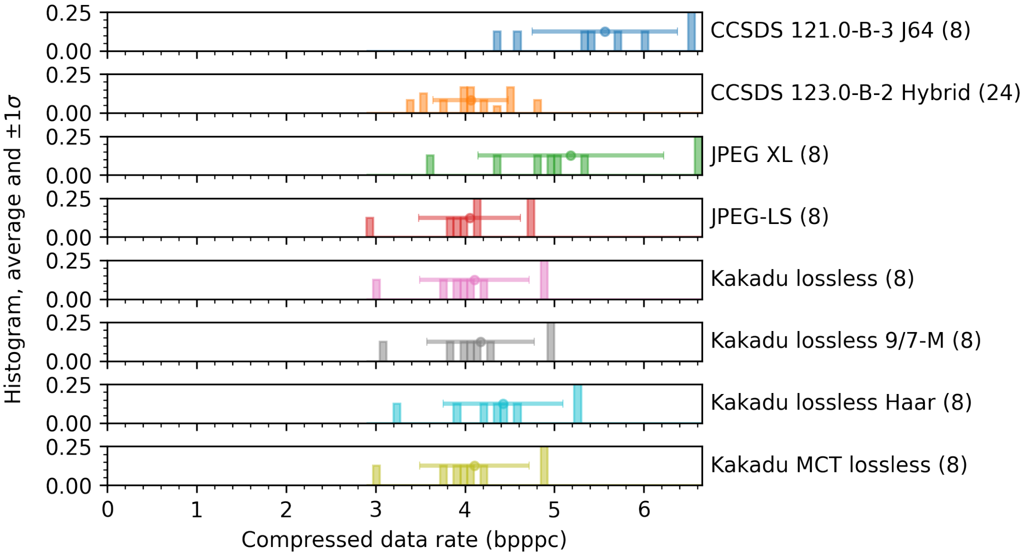

Fig. 7.20: Plot showing the BPPPB the different codecs in lossless compression achieved on the HYPSO dataset with reduced dynamic range. The integer wavelet CDF wavelet transform was applied before compressing.

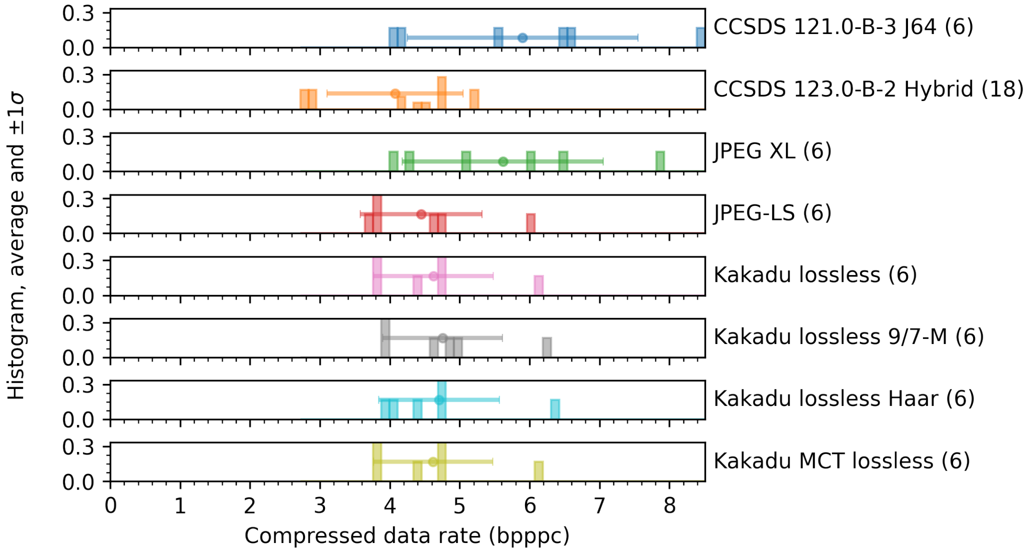

Fig. 7.21: Plot showing the BPPPB the different codecs in lossless compression achieved on the CCSDS-123 test dataset with reduced dynamic range. The integer wavelet CDF wavelet transform was applied before compressing.

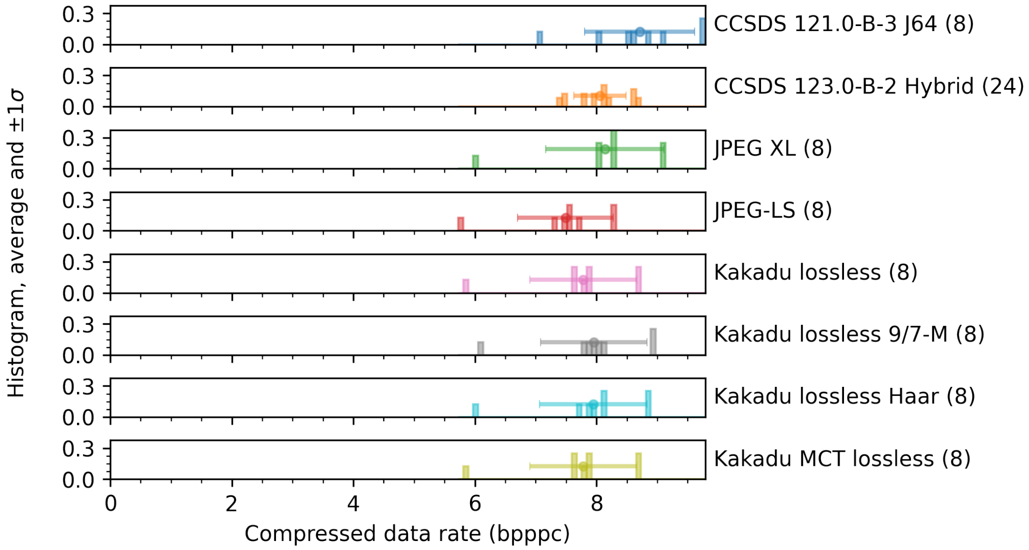

Fig. 7.22: Plot showing the BPPPB the different codecs in lossless compression achieved on the HYPSO dataset. The ISO-range pairwise orthogonal transform was applied before compressing.

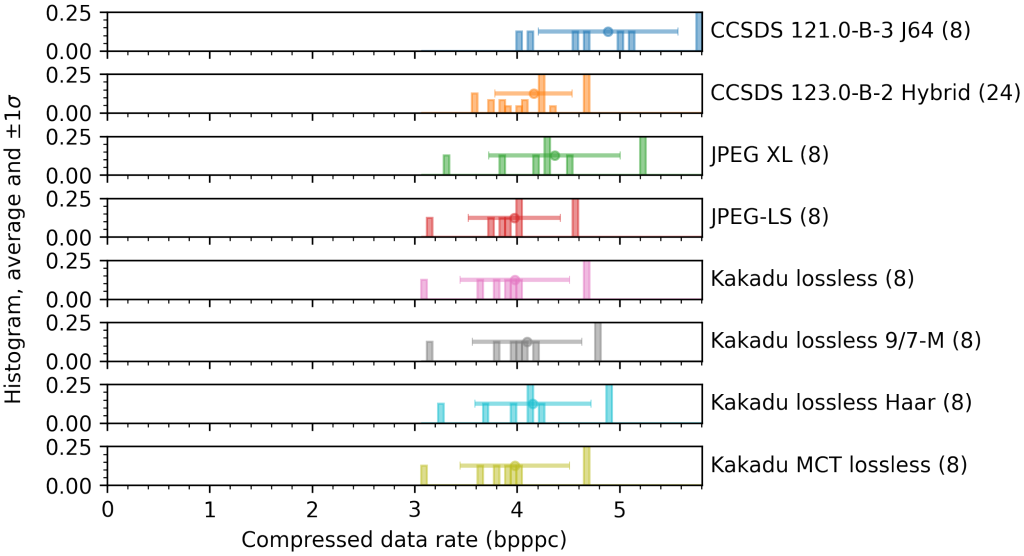

Fig. 7.23: Plot showing the BPPPB the different codecs in lossless compression achieved on the HYPSO dataset with reduced dynamic range. The ISO-range pairwise orthogonal transform was applied before compressing.

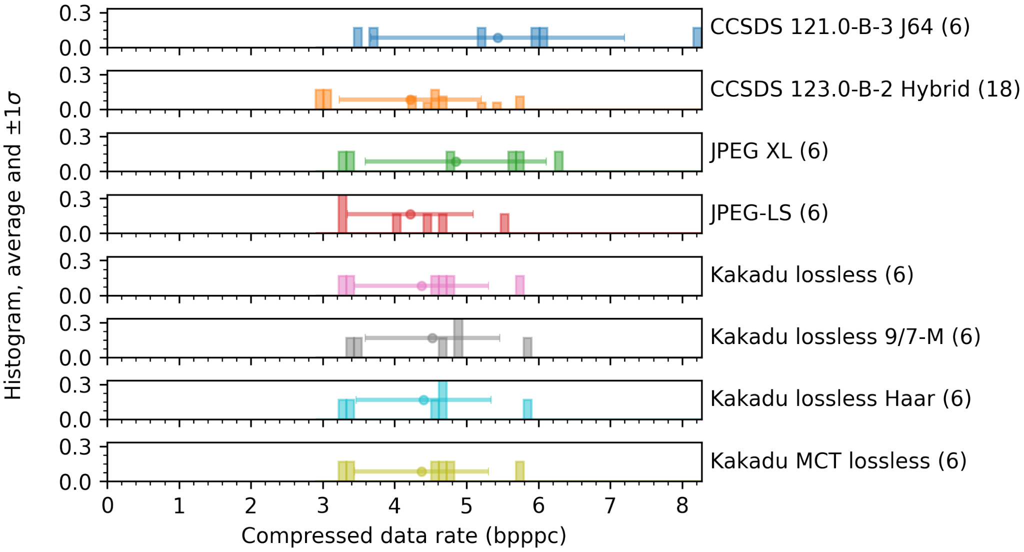

Fig. 7.24: Plot showing the BPPPB the different codecs in lossless compression achieved on the CCSDS-123 test dataset with reduced dynamic range. The ISO-range pairwise orthogonal transform was applied before compressing

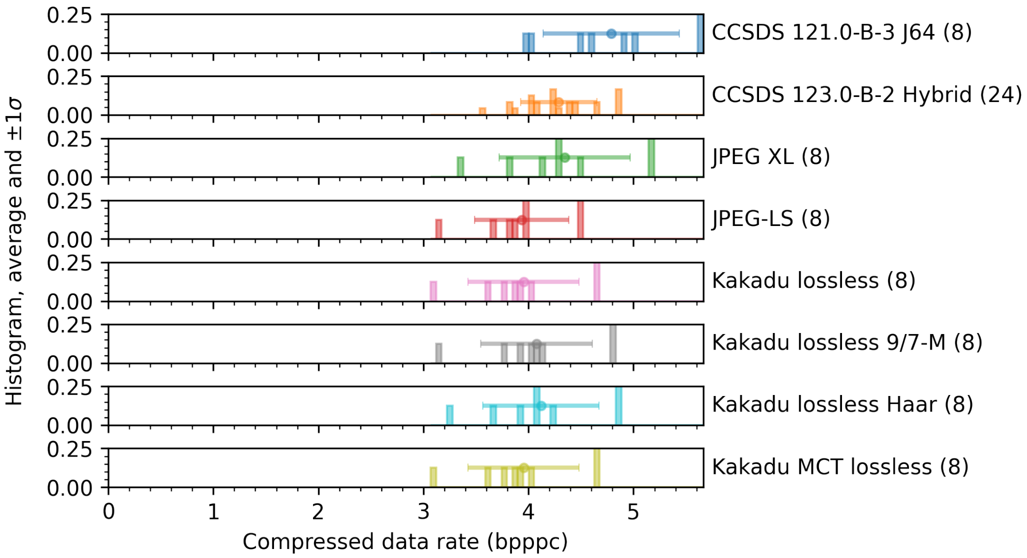

Fig. 7.25: Plot showing the BPPPB the different codecs in lossless compression achieved on the HYPSO dataset with reduced dynamic range. The pairwise orthogonal transform was applied before compressing.

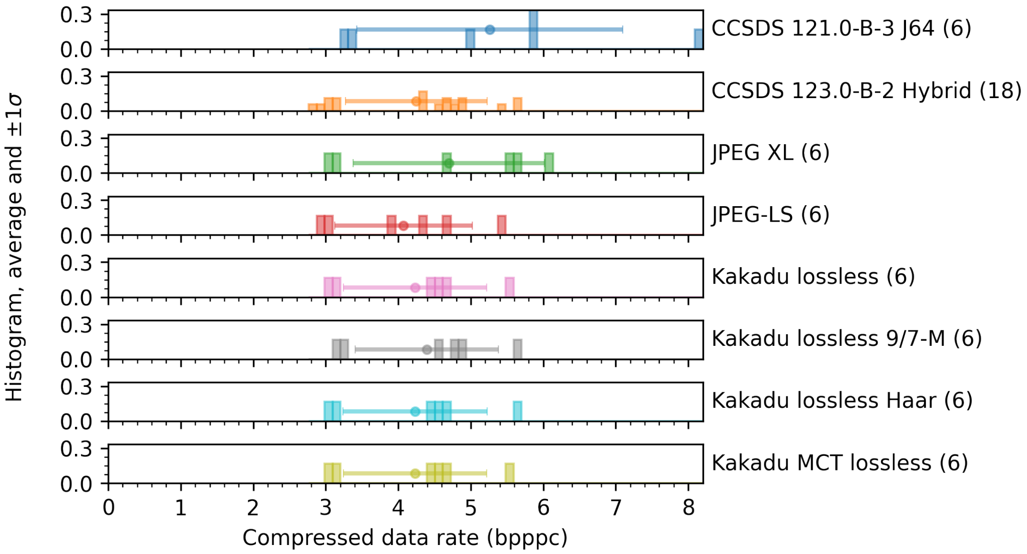

Fig. 7.26: Plot showing the BPPPB the different codecs in lossless compression achieved on the CCSDS-123 test dataset with reduced dynamic range. The pairwise orthogonal transform was applied before compressing.

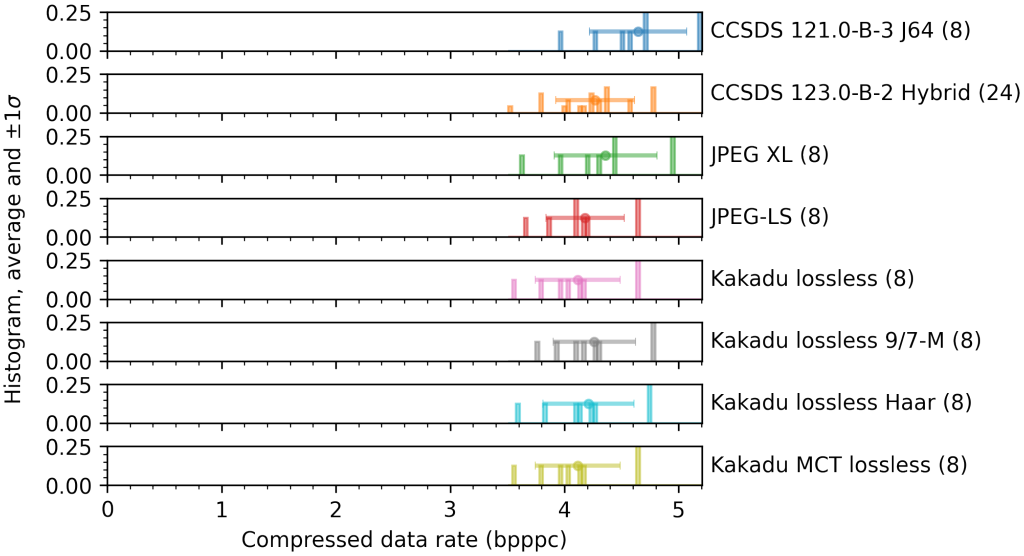

Fig. 7.27: Plot showing the BPPPB the different codecs in lossless compression achieved on the HYPSO dataset with reduced dynamic range. The reversible Karhunen-Loeve transform was applied before compressing.

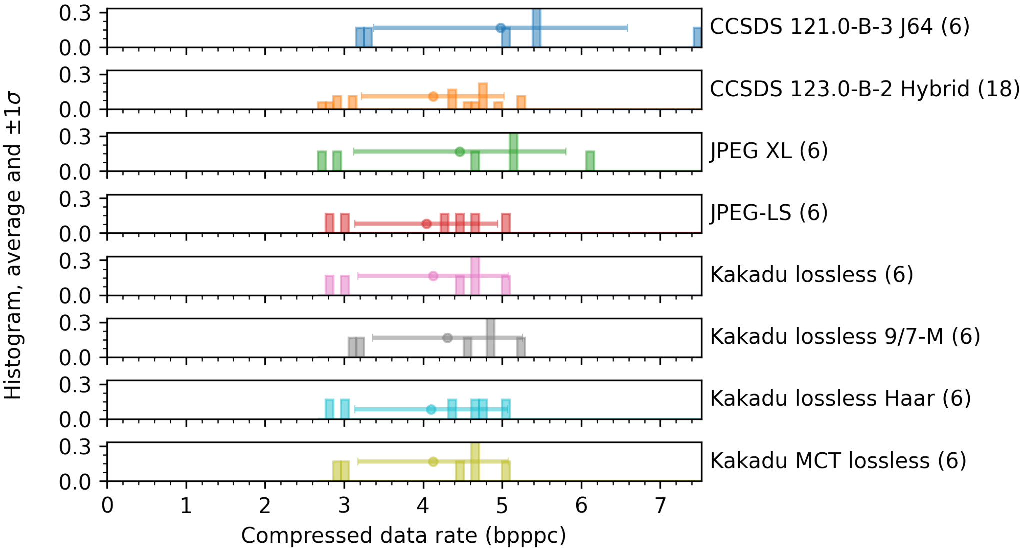

Fig. 7.28: Plot showing the BPPPB the different codecs in lossless compression achieved on the CCSDS-123 test dataset with reduced dynamic range. The reversible Karhunen-Loeve transform was applied before compressing.

NTNU

Norwegian University of
Science and Technology