FAST PIECEWISE LINEAR PREDICTORS FOR LOSSLESS COMPRESSION OF HYPERSPECTRAL IMAGERY

by

Leila Susana Rodríguez del Río

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

ELECTRICAL ENGINEERING

UNIVERSITY OF PUERTO RICO
MAYAGUEZ CAMPUS
2003

Approved by:

Domingo Rodríguez, PhD. Member, Graduate Committee	Date
Miguel Vélez Reyes, PhD. Member, Graduate Committee	Date
Shawn D. Hunt, PhD. President, Graduate Committee	Date
Edgar Acuña Fernández, PhD. Representative of Graduate Studies	Date
José L. Cruz Rivera, PhD. Chairperson of the Department	Date

Abstract

Image compression of hyperspectral is necessary because of the large storage requirements. The main objective of this research is to develop and implement fast and good predictors, for use in lossless compression of hyperspectral images. The algorithms developed in this research are compared in terms of compression performance and computational complexity against the best existent algorithms. These benchmark algorithms are LOCO-I and CALIC-Extended. Lossless compression algorithms are typically divided into two stages, a prediction stage to eliminate redundancy and a coding stage. The predictors can utilize pixels of the same spectral band, adjacent bands or both in order to perform the prediction. Algorithms found in documented research use information of their same band or an adjacent band, but not at the same time. An example of this type of algorithm is algorithm LOCO-I, which only uses the information in the same band in order to predict. However, the CALIC-Extended algorithm alternates between the information in the same band and that of the adjacent one. The algorithms developed during this research use pixels of both bands in order to do the prediction. The best lossless compression predictor implemented in this research, gave better compression than the state of the art along with a low computational complexity.

Resumen

La compresión de imágenes hiperespectrales es necesaria porque estas imágenes requieren un espacio considerable de almacenamiento. Este trabajo tiene como propósito desarrollar e implementar predictores para compresión sin perdida de imágenes hiperespectrales, que sean rápidos y que tengan poca complejidad. Los algoritmos implementados en este trabajo serán comparados en términos de la razón de compresión y complejidad computacional contra los algoritmos de compresión más populares en la literatura. Los algoritmos que utilizaremos como punto de comparación son LOCO-I y CALIC-Extendido. Típicamente los algoritmos de compresión sin pérdida consisten de una etapa de predicción para reducir la redundancia y de una etapa de codificación. Los predictores pueden utilizar tanto píxeles en la misma banda espectral como en la banda previa o en ambas. Los algoritmos presentados en la literatura se basan en predictores donde solo utilizan la información de la misma banda de predicción o utilizan la información de la banda previa, pero nunca utilizan ambas al mismo tiempo. Ejemplo de esto, es el algoritmo LOCO-I donde solo utiliza la información de la misma banda para hacer la predicción. Sin embargo, CALIC-Extended intercambia entre la misma banda y la banda previa. Los algoritmos desarrollados en este trabajo utilizaran píxeles de ambas banda para hacer la predicción. Los mejores algoritmos implementados en este trabajo dieron los mejores resultados de compresión en comparación a los algoritmos mas utilizados en la literatura, y a su vez con poca complejidad computacional.

Dedicatory

To you, God by giving me health and a great deal of wisdom to be able to carry out this work. To you, Noel, for your unconditional support and for always believing in me. To my family, for sharing with me all my achievements. To all my friends of the lab, for all the moments of joy.

“A Dios por darme salud y mucha sabiduría para poder realizar este trabajo. A ti Noel por tu apoyo incondicional y por tus palabras de aliento. A mi familia por compartir junto a mí todos mis logros. A todos mis amigos del laboratorio por los momentos alegres en el lab.”

Acknowledgements

First of all, I want to thank my advisor, Dr. Shawn Hunt for his guidance. Also I want to thank the members of my committee, Dr. Miguel Vélez-Reyes and Dr. Domingo Rodríguez for helping in my research.

I want to thank all the personnel of LARSIP for providing me all the necessary equipment to carry out my work. Also I want to thank TCESS for providing me the financial aid during all my graduate studies.

Thanks Noel, for all those wise counsels and by scolding me occasionally. Thanks Rocío, for taking the time to help me, even though you are busy with your own thesis. Thanks to Emmanuel Arzuaga for helping me a lot. A Special thanks to all my friends of the laboratory for always being there for me. God bless you all!

Work reported herein was funded primarily by NASA URC Program under Grant NCC5-518. This work used facilities supported by Engineering Research Centers Program of the National Science Foundation under award EEC-9986821.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Problem Statement

The main objective of this research is to develop and implement a fast piecewise predictor for lossless compression of Hyperspectral Image data with a lower computational complexity and better efficiency in terms of the compression ratio compared to existing algorithms.

Image compression is a very important problem in the image-processing field since very large images require a large amount of storage space. One example of very large images are those taken with hyperspectral sensors. Hyperspectral images can be defined as images with a high spectral resolution, typically 100 to 300 different wavelengths. The hyperspectral images used in this research are those from an AVIRIS (Airborne Visible and Infrared Imaging) sensor, Hyperion sensor and the hyperspectral camera used in LARSIP (Laboratory for Remote Sensing and Image Processing).

A digital image can be represented as an array of matrixes, in which each element of the matrix is known as a pixel. The image has two spatial dimensions (high and width) and one spectral dimension (See Fig.1). The image obtained from the AVIRIS sensor can have a size of up to approximately 460Mbytes. The AVIRIS sensor consists of 224 spectral bands, where each pixel has 12 bits of precision [1]. Another sensor that will be used in this work is the Hyperion Sensor, which consists of 120 spectral bands, where each pixel has 12 bits of resolution and the images compiled of

This sensor can have a size of approximately 200Mbytes. The hyperspectral camera has a maximum compilation capacity of 120 bands in the visible range, taken at a wavelength of 400nm to 900nm, where each pixel has 12 bits of resolution. The images obtained from the hyperspectral camera have a size of approximately 100Mbytes. As one can observe, the images of either device require a large quantity of storage space due to their large size.

The use of a lossless compression algorithm is necessary in cases where the exact image data must be recovered. The data actually stored in a compression algorithm is the coded difference between the original data and the predicted value and is called prediction error. If the code, the prediction error and the prediction method used are known, the original data can be recovered. Two methods can be used to design this type of algorithm, transforms (i.e. Wavelet, Discrete cosine, etc.) and predictor based. Prediction can be defined as the estimation of a data based on some known information. For the purpose of this research, the prediction-based method will be used in order to implement a lossless compression algorithm.

Different types of lossless compression algorithms based on prediction are well documented in the literature. The algorithms developed in this research are compared in terms of compression performance and computational complexity against the best existent algorithms. The algorithms developed here use pixels in both the same and adjacent spectral bands for prediction, while maintaining low complexity. Examples of the most popular algorithms found in literature are LOCO-I (Low Complexity lossless

Compression for Image) [2] and CALIC-Extended [3]. Compression ratios will be determined either theoretically, using entropy, or by the actual compression achieved.

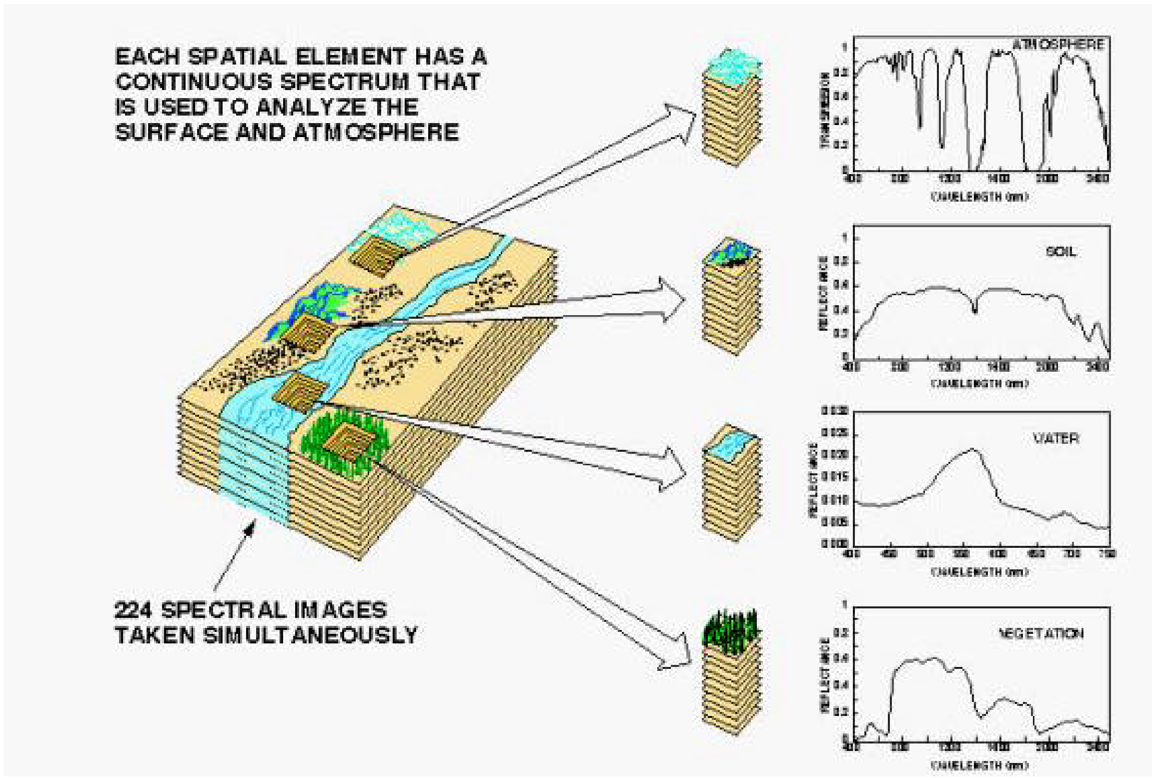

EACH SPATIAL ELEMENT HAS A CONTINUOUS SPECTRUM THAT IS USED TO ANALYZE THE SURFACE AND ATMOSPHERE
224 SPECTRAL IMAGES TAKEN SIMULTANEOUSLY
ATMOSPHERE
SOIL
WATER
VEGETATION
TRANSMISSION
REFLECTANCE
WAVELENGTH (nm)

Figure 1 Digital images are composed of two- dimensional arrays, small squares regions known as pixels and multiple bands.[1]

1.2 Objective

The objectives of the proposed research were:

- To determine theoretical bounds on the compression ratio of different imagery using entropy.
- To design and compare piecewise linear predictors for multidimensional data using different dimensionalities for the prediction.

[1] Figure obtained from http://www.cis.rit.edu/~jaa6739/thesis.htm

- To design and compare piecewise nonlinear predictors for multidimensional data using different dimensionalities.
- To compare multiple levels of context in context coding.
- To compare fixed multi-band prediction with optimal one band prediction.

1.2 Thesis Overview

This work will present lossless compression algorithms for Hyperspectral Image data that contain a low computational complexity as well as a good compression ratio. In order to build fundamental knowledge and have a benchmark for discussion, a review of different lossless compression algorithms existing in literature will proceed. The project will consider their complexity and the prediction methods. New algorithms will be introduced and compared with algorithms that already exist in revised literature. A comparison between the existing algorithms and the proposed ones will be performed considering their computational complexity and their compression ratio. This will be broken up bye chapters as follows:

- This chapter will present the justification, problem statement and the objectives of this work.
- Chapter 2 will present the theory utilized in the development of this research. The following is a brief explanation of the different topics that will be discussed throughout this chapter:

  - Hyperspectral images are hundreds of images that are taken of narrow and contiguous spectral bands. For this reason, in many cases these images require considerable storage space.
  - Entropy is the average information output from the source (i.e. an image), which can be measured in bits per data. This will be measured in bits per pixel.
  - The different algorithms existent in literature will also be discussed. One example is the LOCO-I algorithm that uses the information of three pixels in the same spectral band in order to do the prediction. Another example is the CALIC-Extended algorithm, which alternates between using the same band or the previous one for prediction. A comparison between spectral and spatial data will also be discussed in this chapter.
- Chapter 3 will explain the algorithms proposed in this work. Those algorithms are structured in two main parts: Data Redundancy Reduction and Coding (see Fig. 2). Some of the proposed algorithms are LOCO-3-dimensional and linear 3-D predictor. The first algorithm is similar to the LOCO-I algorithm, but uses two spectral bands for the prediction. The second one is based on an auto-regressive model. The linear predictor uses spectral and spatial data for the prediction. A theoretical explanation, along with a description of each proposed algorithm will be provided.

  In terms of description some of the key points that will be discussed are the prediction method and the data used in order to perform the pixels prediction. A comparison between spectral and spatial data will also be discussed in this chapter.

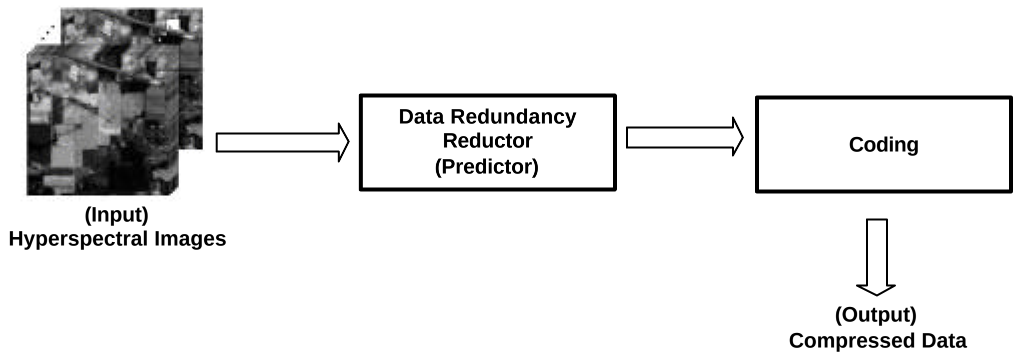

Data Redundancy Reductor (Predictor)
Coding
(Input) Hyperspectral Images
(Output) Compressed Data

Figure 2 Data Compression Block Diagram

- Chapter 4 will present the different experiments done for this work, along with the results of compression ratio using the different prediction algorithms that were mentioned in chapter 2 and 3. We will talk about the different criteria considered in order to select which algorithm's result was the best.
- The last and fifth chapter will present the conclusions and future research that can be conducted in order to improve the results obtained.

Chapter 2: Theoretical Background

2.1 Hyperspectral Image Data

There are diverse instruments used for the collection of image data. These instruments acquire data at different wavelengths arranged**,** spatially**,** pixels (See Fig.1.1), and quantified into discrete levels of brightness. An example of this type of instrument is a hyperspectral sensor. These sensors acquire data in a vast number of narrow and contiguous spectral bands, thus the use of the term hyperspectral. The hyperspectral sensors employed in this work are AVIRIS, Hyperion and a hyperspectral camera used in LARSIP (Laboratory of Applied Remote Sensing and Image Processing).

2.1.1 AVIRIS Sensor

The AVIRIS sensor consists of 224 spectral bands, where each pixel has 12 bits of precision before correction. This sensor has a spectral range of 400nm to 720nm, covering the visible and near infrared regions. The images obtained from the AVIRIS sensor can have a size of up to approximately 460Mbytes. The images used for this research were:

1. North West Indian Pine Test Site in Indiana, consisting of 145 x 145 pixels and 220 spectral bands. See Figure 6.

2. Moffet Field in California, consisting of 148 x 294 pixels and 132 spectral bands. See Figure 14.
3. NASA Kennedy Space Center, consisting of 397 x 268 pixels and 204 spectral bands. See Figure 22.

2.1.2 Hyperspectral Camera

The hyperspectral camera model SOC 700, has a maximum compilation capacity of 120 bands at a visible range, taken at a wavelength of 400nm to 900nm, where each pixel has 12 bits of precision. The image obtained by the Surface Optics Corporation has a size of approximately 100Mbytes. The image used in this work is a parking lot area in San Diego. The image consists of 120 spectral bands and 640 x 640 pixels. See Figure 24.

2.1.3 Hyperion Sensor

The Hyperion sensor consists of 242 spectral bands, where each pixel has 12 bits of precision. The images obtained from a Hyperion sensor can have a size of up to approximately 550Mbytes. The image used for this work consists of 196 spectral bands due to the elimination of noise bands and has been atmospherically corrected. The image is from La Parguera, Puerto Rico, and it has a size of 200Mbytes, since it consists of 3128 x 256 pixels. See Figure 23.

Summary of the specifications of the hyperspectral sensors and camera used for this work:

Table 1 Hyperspectral sensors and camera

Sensor	Bits per píxel per Bands	Spectral bands	Wavelength Range (nm)	Spectral band bandwidth (nm)
AVIRIS	12	224	400-2500	10
Hyperion	12	220	400-2500	10
Camera SOC 700	12	120	400-900	4

2.2 Basic Concepts of Entropy

The lossless compression is archived by the transformation and codification of the data, in a new way using fewer bits to represent the data. In lossless compression the desire is that given a sequence of data to be coded, find the minimum amount of bits to be used in order to code that sequence. Base in this is that we can use the notion of information and probabilistic modeling.

“Assume that a source outputs data from the allowed set according to a certain density function. Every time the source outputs a number, information is transmitted. Information can be viewed as a measure of uncertainty, and is based on how likely or unlikely each value in the allowed range is. Numbers that have a high probability of being output convey less information than numbers that have low probability. In the extreme case where the probability of a number is one, then we know what the next data is before it is output, and it gives us no information”.[11] This is represented in the following definition of information,

$$I_i = \frac{1}{\log_2(p(a_i))} = -\log_2(p(a_i)) \ . \qquad (2.1)$$

Where, it can be said that the information is inversely proportional to the probability. The logarithm base two is used in order to represents the information in bits.

The entropy is the expected value of the information, which can be represented by the formula,

$$H(a) = -\sum_{i=1}^{N} p(a_i)\log_2(p(a_i)) \ . \qquad (2.2)$$

where, this can be use as an estimation of the average amount of bits needed to code the data. This is if each data is coded independently. This will result in an approximation of the lower bound for the bits per data average. The results of bits per data obtained are only an approximation to the real values, since entropy assumes that an optimal code is been use for these results. However, if data is being coded independently and the numbers of bits per data are similar to those values obtained using the first order entropy, it can be said that the code found is good.

“Entropy of any data is a measurement that can be used to determine the theoretical lower bound to which any given data can be compressed without losing any information in order to reconstruct the original data” [5]. Entropy will be used to estimate the information content of the image, in other words, one can estimate the minimum number of bits needed for a sequence. This is very difficult to determine in real cases, however, it is intended to establish a close approximation.

The joint entropy of two data values can be define as,

$$H(a,b) = -\sum_{i=1}^{N}\sum_{j=1}^{N} p(a_i,b_j)\log_2(p(a_i,b_j)) \ . \tag{2.3}$$

It represents a lower bound on the number of bits required to code a sequence of inputs if they are coded two at a time. However, this adds more complexity.

First-order entropy is very useful when each data is coded independently from the others. In case that the data is not independent, one can approximate the number of bits available for use to those of the conditional entropy. Conditional entropy is the average information in any data, given a known data set. Conditional entropy can be defined by

$$H(a/b = b_k) = -\sum_{i=1}^{N} P(a_i/b_k)\log_2(P(a_i/b_k)), \tag{2.4}$$

where $p(a_i /b = b_k)$ is the conditional probability of a_i , given that a set of data b is known.

2.3 Spectral and Spatial Information

Linear predictors are commonly used for lossless compression. The goal is to compress the hyperspectral image efficiently and maintain all the information from the original image. With hyperspectral images, there are two kinds of information to make the prediction, spatial and spectral information.

The hyperspectral images can be modeled as a 3-dimensional matrix composed of two types of coordinates: spatial coordinates and the spectral coordinate. The spatial coordinates represent the intensity (brightness) of the image at a given point. For the purpose of this research, the spatial information is the spatial coordinate contained in the same band of the prediction. One example is Figure 3, where spatial information based on the band of the predicted pixel y is represented in all the pixels that are in the same band (i.e. y_1, y_2, y_3). The spectral coordinate is the wavelength at which the image is acquired. An example of this can be observed in Figure 3, where the pixels in the previous band are the spectral information, because these pixels are in a different wavelength where y is contained. The algorithms implemented in this work can use pixels in the same spectral band (spatial information), in an adjacent band (spectral information) or in both, in order to do the prediction. [17]

2.4 State-of-the-art Algorithms

2.4.1 LOCO-I Algorithm

Prediction

Marcelo J. Weinberger, Gadiel Seroussi and Guillermo Sapiro [2] presented a lossless compression algorithm, called LOCO-I (LOw COmplexity LOssless COmpression). This algorithm is a lossless compression algorithm for continuous-tone images, which combines the simplicity of Huffman coding with the compression potential of context models.

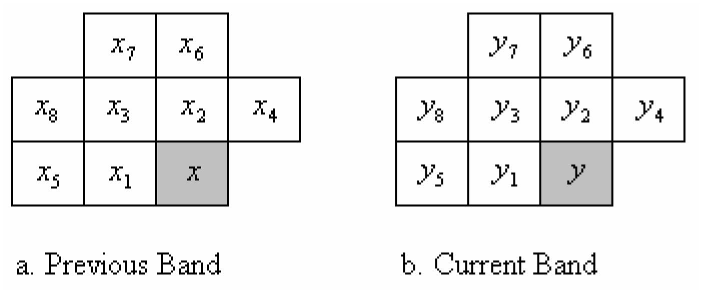


Figure 3 Represents the neighboring pixels used in prediction and for the prediction algorithms.

The LOCO-I algorithm uses the information of three neighboring pixels (the pixels y_1, y_2 and y_3 in fig. 3) in the same spectral band in order to do the prediction, in other words it only uses the spatial information. This algorithm consists of a performance test to detect vertical and horizontal edges. If an edge is not detected, then the predicted value is $y_1 + y_2 - y_3$, as this would be the value of y, if the current pixel belonged to the plane defined by the three neighboring pixels. The LOCO-I predictor is defined as:

$$\hat{y} = \begin{cases} \min(y_1, y_2) & \text{if } y_3 \geq \max(y_1, y_2) \\ \max(y_1, y_2) & \text{if } y_3 \leq \max(y_1, y_2) \\ y_1 + y_2 - y_3 & \text{otherwise} \end{cases} \tag{2.5}$$

As mentioned above, this predictor detects the existence of an edge. For example, if a vertical edge exists then typically $y_1 \leq y_2$, and the predictor (2.5) in many cases chooses the pixel y_1 for prediction. If a horizontal edge is detected the predictor selects the pixel y_2. However, if any edge is detected the predictor uses the plane $y_1 + y_2 - y_3$ for prediction.

Context Modeling and Coding

"The principle of context adaptive coding is to attempt to model the conditional entropy of symbols based on their surrounding neighborhood. This can be seen as a generalized form of prediction where each context determines a complete probability model for the new data (as opposed to only a predicted value, which would be the mean of the conditional distribution). The number of different values for the neighborhood can be quite large, especially for a large alphabet sources, and thus it is normal practice to partition the possible neighborhoods into a smaller set of classes. The appropriate number of classes has to be kept small based on complexity consideration but also to avoid "context dilution" situations."[16]

The total number of parameters in a model depends on the number of free parameters defining the coding distribution in each context and in the number of contexts. The algorithm LOCO-I uses a piecewise-linear predictor with rudimentary edge detecting capability, and is based on a very simple context model, determined by quantified gradients, "thus capturing the level of activity (smoothness, edginess) surrounding a sample, which governs the statistical behavior of the prediction errors".[6] These gradients are defined by the following differences: $g_1 = y_4 - y_3$, $g_2 = y_3 - y_2$, $g_3 = y_2 - y_1$, and $g_4 = y_1 - y_5$. (See fig. 3) By symmetry, g_1, g_2 and g_3 influence the model in the same way and is quantified into up to 9 connected regions by a quantifier k, where the value k yield the shortest code length. These gradients are used to find the values of the boundaries between quantization regions. Those regions are represented by an alphabet of 16-bit per pixel, and they determine the contexts. This is achieved through a single parameter probability distribution per context, efficiently implemented by an adaptive, symbol-wise, Golomb-Rice code. In addition, reduced redundancy is achieved by embedding an alphabet extension in regions. "LOCO-I has achieved the best compromise between compression performance and computational complexity" [7]. [14,15]

2.4.2 CALIC Extended Algorithm

XiaolinWu, and Nasir Memon [3,13] presented a lossless interband compression algorithm, called Extended CALIC (Context-based, Adaptive, Lossless Image Codec).

The original CALIC algorithm only used the inter-band predictor in order to perform the prediction. This predictor only used the spatial information, while Extended CALIC added the spectral information. In other words Extended CALIC added an intra-band predictor. "CALIC algorithm includes a Gradient Adjusted Predictor (GAP), which adapts the prediction according to local gradients and hence gives a more robust performance compared to standard linear predictors. GAP weighs the neighboring pixels of y according to the estimated gradients in the neighborhood. In GAP the gradient of the intensity function at the current pixel y is estimated by computing the equations (2.6)". [15] In other words, CALIC algorithm uses the GAP to perform the prediction. It is based on a context adaptive nonlinear predictor that operates in either binary or continuous-tone mode, depending on the context of the current pixel. In the continuous-tone mode, gradient adjusted prediction takes place and is further improved by error feedback, where prediction errors are modeled under different contexts, resulting in reduced conditional entropies. The coding step may involve either Huffman or arithmetic coding.

The following is a summary of the prediction method employed by Extended CALIC algorithm. First, it computes the vertical and horizontal gradient magnitudes near the current pixel y. (See Fig. 3) These gradients are used to calculate the intraband predictor. However, the interband predictor uses the images as a 3-dimensional system, using the information of the previous band to estimate the current pixel y. This algorithm uses both predictions (inter/intra-band) for estimate the pixel y, which uses the correlation coefficient results to select what kind of prediction is the best to estimate the current pixel. It has been demonstrated that its performance is comparable to that of the universal context modeling method and slightly better than that of the LOCO-I method, but its complexity is higher. [8]

Description of the Inter/Intra Band Prediction of the CALIC Extended Algorithm:

The following procedure describes the inter-band prediction:

```
if $(|x - x_2| - |x - x_1| > T)$  $\{ \hat{y} = \hat{y}_h \}$          (Sharp horizontal edge)
esle if $(|x - x_2| - |x - x_1| < -T)$  $\{ \hat{y} = \hat{y}_v \}$   (Sharp vertical edge)
else $\{ \hat{y} = (\hat{y}_h + \hat{y}_v)/2 \}$
```

Switching to Intra-band Prediction using the following parameters:

The Correlation Coefficient:

$$\wp(X,Y) = \frac{8\sum_{i=1}^{8} x_i y_i - \sum_{i=1}^{8} x_i \sum_{i=1}^{8} y_i}{\sqrt{\left[8\sum_{i=1}^{8} x_i^2 - \left(\sum_{i=1}^{8} x_i\right)^2\right]\left[8\sum_{i=1}^{8} y_i^2 - \left(\sum_{i=1}^{8} y_i\right)^2\right]}}$$

If $\wp(X,Y) > 0.5$ then it is use the proposed inter-band predictor $\hat{y}$; otherwise it is use an intra-band predictor $\hat{y}$.

The intra-band predictor is computed by the following procedure:

```
if $(d_v - d_h > T_1)$   $\{ \hat{y} = \hat{y}_1 \}$                  (Sharp horizontal edge)
esle if $(d_v - d_h < -T_1)$                              (Sharp vertical edge)
else $\{ \hat{y} = (y_1 + y_2)/2 + (y_3 - y_4)/4;$
    if $(d_v - d_h > T_2)$      $\{ \hat{y} = (\hat{y} + y_1)/2 \}$     (Horizontal edge)
esle if $(d_v - d_h > T_3)$   { $\hat{y} = (3\hat{y} + y_1)/4$ }     (Weak horizontal edge)
esle if $(d_v - d_h < -T_2)$  { $\hat{y} = (\hat{y} + y_2)/2$ }      (Vertical edge)
esle if $(d_v - d_h < -T_3)$  { $\hat{y} = (3\hat{y} + y_2)/4$ }}    (Weak vertical edge)
```

Where $T_1 = 80,\ T_2 = 32,\ T_3 = 8$ and the vertical and horizontal gradients are:

$$\begin{aligned} d_h &= |y_5 - y_1| + |y_3 - y_2| + |y_2 - y_4| \\ d_v &= |y_6 - y_2| + |y_3 - y_1| + |y_7 - y_3| \end{aligned} \tag{2.6}$$

2.4.3 FELICS Algorithm

Paul G. Howard and Jeffrey Scott Vitter [4] presented a very fast and very simple method for lossless image compression that gave comparable results to JPEG lossless mode. The method was called Fast Efficient, Lossless Image Compression System (FELICS). It combines both prediction and error-modeling steps by using the two nearest neighbors of a pixel in a raster scan in order to estimate the probability distribution of the pixel intensity. The two neighboring pixels are those located above and to the left of the estimated pixel. For example, the pixels y_1 and y_2 are the neighboring pixels of y in figure 3. Based on a parameter estimation method, the most suitable error model is chosen from a set, and the intensity is encoded using the Rice-Golomb code of the model.

2.5 Preview Comparison Results

This section will discuss results and comparisons of the most used algorithms in scientific literature to this point.

In 1993, Howard and Scout [4] implemented an algorithm called FELICS. This algorithm was compared to the best existing algorithm at that point, which was JPEG lossless mode. Experimentally, JPEG lossless mode gave better compression results than FELICS algorithm, but FELICS had less computational complexity than JPEG, because it runs about five times faster. However, in 1996, Weingberger, Seroussi and Sapiro implemented a new algorithm, LOCO-I, obtaining better results in terms of compression ratio and computational complexity than FELICS and JPEG. LOCO-I was also compared to CALIC algorithm, the latter being the better one in terms of compression, while LOCO-I was the best in terms of computational complexity. There has been a lot of effort put into the development of algorithms that improve compression ratios, but at this point the best in terms of compression ratio is still CALIC whereas LOCO-I remains the best one in terms of complexity.

Wu and Memon [3] implemented a modification of the CALIC algorithm calling it Extended CALIC. This algorithm introduced a new concept in terms of prediction by using the information of the previous band. Chapter 3 will explore a variety of algorithms implemented during this work based on the low computational complexity of LOCO-I and the compression ratio results of Extended CALIC.

2.6 Time Complexity Analysis

"When analyzing the efficiency of an algorithm in terms of time, we do not determine the actual number of CPU cycles, because this depends on the particular computer on which the algorithm is run". [12] Besides, the desire is not to perform the analysis against all the instructions executed, because this depends on the programming language and in the way that the programmer implements the algorithm. It can be said that the running time of an algorithm depends in the amount of time a basic operation is executed. Therefore, the efficiency of the algorithms was analyzed base in the number of time in which some basics operations was executed as a function of the input. In terms of the algorithms implemented in this research, all of them will depend on the size of the image. These images are considered as three-dimensional matrices, where the input is the number of bands, rows and columns of the image.

"In general, a time complexity analysis of an algorithm is the determination of how times the basic operation is done for each value of the input size. Although we do not want to consider the details of how an algorithm is implemented, we will ordinarily assume that the basic operation is implemented as efficiently as possible."[12]

Chapter 3: Developed Algorithms

3.1 Data Compression Background

The main objective of the compression algorithms described in this thesis is to minimize a 3-dimensional image data volume with no loss of information. There are two different ways of doing compression, transform-based methods and prediction methods. Prediction methods can achieve higher compression ratios in a much less expensive way, tend to be much faster than transform-based, and can easily be implemented in hardware [9]. This work will focus on the prediction method in order to implement lossless compression algorithms.

The compression process used can be divided into two steps, a de-correlation stage, where the intention is to produce an independent sequence, and a coding stage. One of the theoretical justifications for this method can be explained by Wold's decomposition theorem (1938) that states the following:

Let $x[n]$ be a wide sense stationary (WSS) process. A process is WSS if its mean is constant and its auto-correlation is a function of (n_1- n_2). See Eq. (3.1) and (3.2).

$$E\{x[n]\} = m[n] = m \ . \qquad (3.1)$$

$$R(n_1 - n_2) = E\{x[n_1]x^*[n_2]\} \ . \qquad (3.2)$$

$x[n]$ can then be written as the sum of two processes, described in the following equation:

$$x[n] = x_r[n] + x_p[n]. \tag{3.3}$$

Where $x_r[n]$ is a regular process, and $x_p[n]$ is a predictable process, and $x_r[n]$ is orthogonal to $x_p[n]$. [10]

A process is regular when the power spectral density has no impulses, where the spectral density of $x_r[n]$ can be represented as $S_r(e^{j})$. If the spectral density were represented in Discrete Time Fourier Transform, this would be the transform of the autocorrelation of $x_r[n]$,

$$R_r[m] = E\{x_r[n]x_r[n-m]\}, \tag{3.4}$$

$$S_r(e^{j}) = \sum_{m=-\infty}^{\infty} R_r[m]e^{-jm}. \tag{3.5}$$

If this is regular process its power spectral density would be a continuous function and the elements of this process are independent. The process $x_p[n]$ is predictive, so

$$x_p[n] = \sum_{k=1}^{\infty} h_k x_p[n-k], \tag{3.6}$$

where h_k are coefficients of a linear filter. In other words $x_p[n]$ can be determined by a linear relation of its previous values. If it is assumed that the data to be compressed is of a WSS process, then one can break the data in the way that the Wolds theorem suggests in equation (3.1). In order to compress an image, the first step is to find the prediction model for $x_p[n]$, which is the predictable section of the signal. When the prediction model is applied to the signal, the result is the non-predictable part or the regular process, $x_r[n]$. (See Fig. 4) Since the elements are independent, this is a white noise process. This regular, non-predictable part is the prediction error, since it is the difference between the original signal and the output of the predictor. The idea is to save the information provided by the prediction error and the prediction model of $x_p[n]$ rather than saving the original signal. In general, the original signal $x[n]$ has higher first order entropy than the prediction error. It will thus take fewer bits to independently code each sample of the prediction error than the original signal.

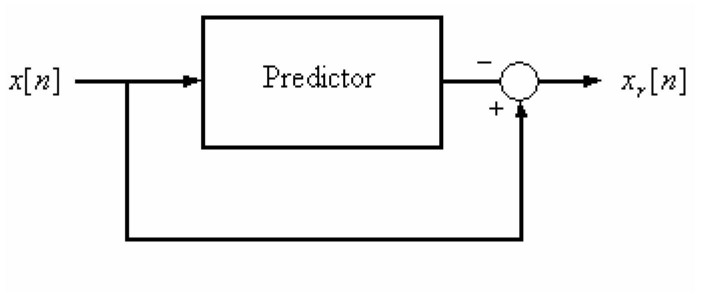


Figure 4 Prediction Diagram

However, in this research will be working with hyperspectral images. In other words the data is 3 dimensional. Where the predictor can be represented as a predictable process $x_p[n]$ in the following equation:

$$x_p[b,m,n] = \sum_{i} \sum_{j} \sum_{k} w(i,j,k) x_p(b-i, m-j, n-k) \quad (3.7)$$
$$(i,j,k) \in \Pi$$

In [18,19] Images are shown to follow on auto-regressive model. This can be written in 3-dimensional as:

$$\hat{x}_p[b,m,n] = \sum_i \sum_{\substack{j \\ (i,j,k)\in\Pi}} \sum_k w(i,j,k)\,x_p(b-i,m-j,n-k) + x_0 + u(b,m,n) \qquad (3.8)$$

Where $w(i,j,k)$ is a 3-D prediction coefficient array, Π is a set of integer pairs to be specified later, x_0 represents a locally constant bias coefficient added to the input of the feedback system, and $u(b,m,n)$ is the 3-D input sequence [18,19]. The exact number of elements to be included in equation 3.9 is determined experimentally in Chapter 4.

3.2 Entropy Coding and Prediction

How can it be determine which predictor is the optimum predictor? This can be determined utilizing entropy. Entropy coding is based on the principle that some of the numbers in an allowed range are used more often than others. "Instead of giving the same length code to each number, the ones that are used more often (more probable and have less information) are given a shorter code than the ones that are used often (less probable and have more information)". [11]

The fundamental question of lossless compression given a sequence that needs to be coded is, what is the minimum number of bits that can be used to code the sequence? This work will use entropy as the average information output from the image measured in bits per data. The first entropy is the expected value of the information, in this case the information given by the output value of the prediction, $x_r[n]$. In other words, in this work this would be the estimated value of *y* since this is the desired value to predict. This entropy can be represented by the following equation,

$$H(x_r(b,m,n)) = -\sum_{i=1}^{B}\sum_{j=1}^{M}\sum_{k=1}^{N} p(x_r(b-i,m-j,n-k))\log_2(p(x_r(b-i,m-j,n-k))) , \qquad (3.9)$$

where *N* represents the number of rows, *M* the number of columns and *k* represents the spectral band where the prediction is being done.

Additionally, based on first order entropy, conditional entropy will be used, which was defined in Chapter 2. This entropy represents the amount of bits per pixels in the prediction error in order to predict pixel *y*, given a known data set. An example is to presume that pixel y_1 is the information known in order to predict pixel *y*. For this reason the conditional entropy can be represented by the following:

$$H(\mathbf{Y}/\mathbf{Y}_1 = y_1) = -\sum_{i=1}^{B}\sum_{j=1}^{N}\sum_{k=1}^{M} P(y/y_1)\log_2(P(y/y_1)) \qquad (3.10)$$

where *N* represents the number of rows, *M* the number of columns and *B* the number of bands. This entropy indicates the amount of bits per pixels that the prediction error would have if pixel y_1 were used to make the prediction of pixel *y*. (See Fig. 3)

3.3 New algorithms for Lossless Compression

3.3.1 Linear-3D predictor

The start point for the development of new predictors was the linear 3-D predictor suggested by Wold's decomposition theorem. Here a linear relation of previous values can determine the predictable part. The prediction model used is as follows:

$$\hat{y} = w_1 x + w_2 y_1 + w_3 y_2 . \tag{3.11}$$

This prediction model utilizes the neighbor pixels y_1 and y_2, that are contained in the same band where the prediction is being done and the pixel x of the previous band. (See Fig. 3) This is an auto-regressive model, as the predictable part depends on the previous pixels. In order to find the weights of the optimum linear filter, the Yule-Walker equations can be utilized. In equation (3.12) one can observe the Yule-Walker equations for our prediction model in matrix form.

$$\begin{bmatrix} R_{dd} & R_{dc} & R_{db} \\ R_{cd} & R_{cc} & R_{cb} \\ R_{bd} & R_{bc} & R_{bb} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} r_{dx} \\ r_{cx} \\ r_{bx} \end{bmatrix} \Rightarrow \mathbf{Rw} = \mathbf{r} . \tag{3.12}$$

The correlations of the system can be represented by equations (3.13) through (3.18).

$$R_{dd} = E\{\mathbf{x}\mathbf{x}^T\} \tag{3.13}$$

$$R_{dc} = E\{\mathbf{x}\mathbf{y}_1^T\} \tag{3.14}$$

$$R_{db} = E\{\mathbf{x}\mathbf{y}_2^T\} \tag{3.15}$$

$$R_{cc} = E\{\mathbf{y}_1\mathbf{y}_1^T\} \tag{3.16}$$

$$R_{cb} = E\{\mathbf{y}_1\mathbf{y}_2^T\} \quad (3.17)$$

$$R_{bb} = E\{\mathbf{y}_2\mathbf{y}_2^T\} \quad (3.18)$$

Vector $\mathbf{r}$ represents the correlation that exists between the pixel that is intended to predict and each one of the pixels that will be used to do the prediction. These correlations are defined in equations (3.19), (3.20) y (3.21).

$$R_{xd} = E\{\mathbf{y}\mathbf{x}^T\} \quad (3.19)$$

$$R_{xc} = E\{\mathbf{y}\mathbf{y}_1^T\} \quad (3.20)$$

$$R_{xb} = E\{\mathbf{y}\mathbf{y}_2^T\} \quad (3.21)$$

Assuming that matrix $\mathbf{R}$ is nonsingular (i.e. the inverse matrix $\mathbf{R}^{-1}$ exists), it is obtained the solution of the equation (3.22) is

$$\mathbf{w} = \mathbf{R}^{-1}\mathbf{r}, \quad (3.22)$$

The predictors' values are:

$$\hat{\mathbf{y}} = \mathbf{w}^{\mathrm{T}}\mathbf{Y}, \quad (3.23)$$

where

$$\mathbf{Y} = [x, y_1, y_2]^T. \quad (3.24)$$

Based on the main objective, which is a low complexity algorithm, the different predictors are developed next that archived this goal. Due to the complexity of the Linear-3D, it was decided to research and develop simpler predictors with the same or better compression ratio.

3.3.2 LCL-3D Predictor

JPEG-LS is the standard for lossless compression using only spatial information. The predictor used in this algorithm is the LOCO-I described in Chapter 2. It is characterized by good performance and low complexity based on a piecewise linear predictor. Based on these characteristics of the predictor algorithm, the studies were focused on finding a 3-D predictor with similar characteristics. In this implementation it is desired to use the spectral and spatial information of an image in the same predictor. Common lossless compression algorithms consist of predictors where the pixels of the same spectral band, in an adjacent spectral band. LOCO-I is an example of an algorithm that uses three pixels of the same spectral band (spatial information only). CALIC-Extended changes between using the same spectral band and the previous band in order to perform the prediction. The algorithms developed in this work use pixels in both the same and adjacent spectral bands for the prediction, while maintaining low complexity. The simplest 3-D predictor was studied first, represented by equation (3.25).

$$\hat{y} = y_1 + (x - x_1)\,. \tag{3.25}$$

This algorithm is based on the selection of pixel y_1 as the ideal predictor for y. As seen in the entropy results in the next chapter, the best pixel to use for predicting y in the same spectral band is y_1. A correction factor is added to this $(x - x_1)$ where it is assumed that the difference between y and y_1 is similar to x and x_1. If this were an ideal predictor, the prediction error (E_p) would be equal to zero,

$$E_p(k) = y - y_1 = 0\,. \tag{3.26}$$

The same behavior happens when this concept is applied to the previous spectral band. There is also a prediction error of zero,

$$E_p(k-1) = x - x_1 = 0\,. \tag{3.27}$$

If the predictor was an ideal predictor both equations would be equal to zero, for this reason equation (3.26) can be equal to equation (3.27), giving the following:

$$y - y_1 = x - x_1\,. \tag{3.28}$$

These equations are equal to each other in order to find a predictor where it can combine both information, the information of the same spectral band and the adjacent band in the same predictor. If equation (3.28) is solve for pixel y it is obtained as a result equation (3.25) giving this a new lossless compression predictor for hyperspectral images.

3.3.3 New 3-D Algorithms based on LOCO-I

LOCO-I algorithm was used as a starting point, since this algorithm combines the results of a good compression ratio and low computational complexity, currently making it the fastest and most efficient algorithm for images of small dimensions (i.e. RGB and Gray Scale Images). It is also worth mentioning that CALIC-Extended is the best algorithm documented in literature in terms of compression, but it has the trade-off of being more complex. The main objective of this work is to find an algorithm more efficient in term of compression ratio and in terms of computational complexity than those algorithms existent in literature. The following algorithms are variations of the LOCO-I algorithm. The variations/modifications consist in adding the spectral information, since this information gives better compression performance.

I. LOCO-2B algorithm

This algorithm consists of using the LOCO-I algorithm in two bands simultaneously.

```
if $(y_3 \geq \max(y_1, y_2)$
        $\hat{y} = \min(y_1, y_2)$
        $\hat{x} = \min(x_1, x_2)$
else if $(y_3 \geq \min(y_1, y_2)$
        $\hat{y} = \max(y_1, y_2)$
        $\hat{x} = \max(x_1, x_2)$
else
        $\hat{y} = y_1 + y_2 - y_3$
        $\hat{x} = x_1 + x_2 - x_3$

  $y_{pre} = \hat{y} + x - \hat{x}$
```

Figure 5 LOCO 2B algorithm description

Figure 5 shows the description of this algorithm. LOCO-I in two bands (LOCO-2B) performs the same analysis to detect the existence of a horizontal or vertical edge that is performed by LOCO-I. If an edge is not found the same procedure as LOCO-I is followed. The same selection procedure of how to do the prediction in LOCO-I is applied to the spectral band where the prediction will take place. After making the selection to predict the value of pixel y, the same is applied to the previous band and an estimated value of $\hat{x}$ is found. For example, if no edge is found in the band where the prediction is desired, the estimated value of y will be computed using the following equation,

$$\hat{y} = y_1 + y_2 - y_3 . \tag{3.29}$$

In the previous band the same selection is applied, obtaining as a result the estimated value of pixel $\hat{x}$ equation (3.30).

$$\hat{x} = x_1 + x_2 - x_3 . \tag{3.30}$$

The same will occur for other cases. After obtaining both estimated values $\hat{x}$ and $\hat{y}$ the prediction error would be calculated for both cases. It is presumed that the prediction error will be the same for both cases, resulting on the following equation:

$$y_{pre} = \hat{y} + x - \hat{x} \ . \tag{3.31}$$

Where y_{pre} will be the estimated value of the pixel that is intended to predict, y.

II. LOCO-3D algorithm

The algorithm LOCO-3D is also base in the algorithm LOCO-I, which is describe in equation (3.32)

$$\hat{y} = \begin{cases} \min(y_1, y_2) & \qquad y_3 \geq \max(y_1, y_2) \\ \max(y_1, y_2) & \quad if \quad y_3 \geq \min(y_1, y_2) \\ y_1 - x + x_1 & \qquad otherwise \end{cases} \tag{3.32}$$

Based in the compression results obtained by the predictor LOCO-I, the same analysis was done in order to detect vertical or horizontal edges. If an edge is not detected the predictor uses the following equation, $y_1 + x - x_1$ in order to predict pixel *y*. As it can be observed this equation is the same equation used in predictor LCL-3D. By using this equation the spectral information is added to the predictor.

Assuming that $y_2 \le y_1$, then predictor (3.32) in most of the cases selects y_2 when a vertical edge is detected. If a horizontal edge is detected y_1 is selected in most of the cases. If no edge is detected the predictor selects $y_1 + x - x_1$ for the prediction of pixel *y*.

III. LOCO-SI algorithm

The next algorithm based on LOCO-I, called the LOCO-SI is presented below.

$$\hat{y} = \begin{cases} y_1 - x_1 + x & & |x_1 - y_1| < T \text{ and } |x_2 - y_2| < T \\ \min(y_1, y_2) & \textit{if} & X_{k,i-1,j-1} \ge \max(y_1, y_2) \\ \max(y_1, y_2) & & X_{k,i-1,j-1} \ge \min(y_1, y_2) \\ y_1 + y_2 - y_3 & & \textit{otherwise} \end{cases} \tag{3.33}$$

The first step that is to verify how similar the previous band (Band *k*-1) is to the current spectral band (Band *k*). In order to determine the similarity between these two bands, a threshold, *T* is established. This threshold will indicate how similar the pixels of the previous band are to those in the current band. The absolute value is found of the following differences $x_1 - y_1$ and $x_2 - y_2$ in order to determine using the established threshold how similar these spectral bands are (See Eq. (3.34)). If the bands are similar, the value of pixel y is estimated using the LCL-3D predictor, described above. The threshold, T used for this predictor was 350.

$$|x_1 - y_1| < 350 \text{ and } |x_2 - y_2| < 350 \tag{3.34}$$

If the condition similarity is not detected, the second step of the predictor is used. This step consists in finding the existence of a horizontal or vertical edge. If this were not found, equation (3.29) would be applied to the system in order to estimate the value of pixel y.

Chapter 4: Experimental Results

4.1 Introduction

The hyperspectral images used for this research have between 12 and 16 bits of resolution per pixel. The objective of this research is to represent the information in these images with fewer bits in order to reduce their storage space. The LOCO-I and CALIC-Extended algorithms are the state of the art in lossless compression and so were used as reference, and used to determine which algorithm resulted in the best lossless compression performance. The comparison of the different algorithms was based on compression performance and computational complexity. All the proposed algorithms are based on the prediction of pixel y using the neighbor pixels and the pixels of the previous band. In the following sections the results of the entropy analysis, the results of compression for each image using algorithms developed in this research, as well as the algorithm, which was the best for each image in terms of compression performance are presented. In the last section, the algorithm with the lowest computational complexity based on the execution time is determined.

4.2 Entropy Analysis of Images

In this section is presented the first set of experiments of this research is presented. These experiment were ran to determine which pixel combination provides the best entropy results. When used for prediction and the first order entropy of the images. This is determined by using conditional entropy. All compression algorithms use a combination of pixels for prediction. The best set of pixels to be used on the data, and so will be determined and used for all algorithms. This experiment's objective was to verify which type of information was going to provide the best prediction results. In other words, the objective was to determine which prediction provided the best results: prediction only using spatial information, prediction only using spectral information or the one using both spectral and spatial information.

Experiment 1:

The objective of this experiment was to determine which pixel combination provides the best compression results. In order to do so, the conditional entropy was used, since this method provides the entropy of error using the optimum predictor. For this experiment the Indiana image with 12 bits per pixel was used. Due to the fact that these images have to be represented with 16 bits in order to be process, therefore the maximum value of a pixel can be 65536. In order to apply the conditional entropy to two pixels, given a pixel of value 65536, is necessary approximately 4Gbytes of memory to perform this process. Given that this memory capacity is not available in the facilities, the conditional entropy for only one pixel was done. (See Table 2)

Table 2 Conditional entropy of y conditioned on different pixel combination.

Combination Pixels	**Entropy Value**
y_3	4.314
y_1	3.957
y_2	4.265
x	3.926

As it can be seen in Table 2, the best conditional entropy result was obtained using pixel x. This indicates this that for this type of hyperspectral image the best compression results are found when using information from the previous band. However, using pixel y_1 also gave good entropy results.

Experiment 2:

A second experiment was run, due to the issue of finding the conditional entropy. The experiment consisted in reducing the resolution of the Indiana image to 8 bits, making the image smaller and easier to manage. That the 8 bit and 12 bit images had the same behavior with respect to entropy was verified first. Once this was done, the 8-bit image was used to calculate the conditional entropies.

Table 3 Conditional entropy of *y* conditioned on different pixel combination.

Pixels Combination	Entropy Value
y_3	3.028379
y_2	2.723524
y_1	2.509909
x	1.418566
y_2, y_1	1.775573
y_2, x	1.192973
y_1, x	1.185436
y_2, y_3	2.196057
y_3, y_1	1.981593
y_3, x	1.192791

Another objective of the experiment was to compare the behavior presented in Table 2 with the results in Table 3. As it can be seen in both tables the best entropy results is found when using pixel x to perform the prediction. Therefore, it can be presumed that the behavior for the entropy values will be the same when using more than one pixel to perform the prediction.

Table 3, shows the different entropy results with the different combinations of pixels. The pixels from the same spectral band and the adjacent band of pixel *y* were used. As before, when using one pixel *x* is best. For a combination of two pixels, *x* and y_1 are best. But as it can be observed in the table, the best entropy results were found when using pixel *x* to perform the prediction. However, the best pixel combination was pixel *x* and y_1, since the lowest entropy value is found when using this pixel combination, for the case that can be observe in Table 3.

Experiment 3:

The issue of using conditional entropy is that it has a high computational complexity. This is due to the fact that a large amount of memory is needed in order to process the conditional probabilities, when they are conditioned to three pixels. For this reason it was determined using the combination of two pixels in order to predict *y*. In order to address this issue a linear predictor was used for different combination of pixels, and the first order entropy of the prediction error was calculated. This was then compared to the results obtained using conditional entropy.

This linear predictor consists of the following equations in order to determine all the possible combinations. Equation (4.1) represents the predictor used for this stage of the research. Where *n* is the number of pixels used to do the prediction w_k, represents the weight that is given to each pixel, x_k represents the pixel to be used and w_o is an offset that is added to the equation.

$$\hat{y} = \sum_{k=1}^{n} w_k x_k + w_o \tag{4.1}$$

The entropies shown on Table 4 represent the number of bits per pixel after prediction. The lower this numbers**,** the better the compression. As one can see, using the adjacent spectral band (pixel ***x***) again leads to the best compression results. These results were compared with the ones obtained using conditional entropies and it can be observed that they have the same behavior. When more information is added, better compression results can be obtained, but at the expense of higher computational complexity since more data needs to be processed. Another interesting observation is captured by this table is the fact that if the amount of spectral information increases, a better compression performance is archived.

Table 4 Entropy values vs. the pixels used for prediction

Pixels Combination	**Entropy Value**
x_1	7.692
x	6.069
y_1	7.596
y_2	7.894
y_3	8.128
$x\,x_1$	6.068
y_1, x_1	7.604
y_1, x	6.021
y_2, x_1	7.504
y_2, x	6.044
y_2, y_1	7.446
y_3, x_1	7.656
y_3, x	6.051

y_1, y_3	7.586
y_2, y_3	7.887
y_1, x, x_1	5.860
y_2, x_1, x	6.046
y_2, y_1, x_1	7.451
y_2, y_1, x	6.008
y_3, x_1, x	6.053
y_3, y_1, x	6.019
y_3, y_1, x_1	7.589
y_3, y_2, y_1	7.300
y_3, y_2, x	6.043
y_3, y_2, x_1	7.376

If the results in experiment are observed, it can be notice that the behavior of the entropy values is the same to the results obtained in Table 4. As the other cases already mentioned, it can be observed that when using pixel x, better entropy results are found. However, it can be concluded that the best entropy results are found when using the pixel combination x, x_1, and y_1.

Experiment 4:

Once all the entropy values for the different pixel combination were found, the fourth experiment was started. This consisted in determine the entropy values for each hyperspectral image using different predictors. Once these results were found, they will be compared with the results of the first experiment, since experiment 1 will be use as a reference point to determine the quality of the predictor. Having in mind that the results of Table 2 are the values of bits per pixel needed to code the data given an optimum predictor. Therefore, with the results of Table 2, it will be determine which predictor is closer to the optimum predictor. Also determine which predictor of Table 5 obtains the lowest entropy value for each of the images used in this work.

Table 5 Entropy results using different predictors (bits/pixels)

Image	Original Entropy	LOCO-I	LOCO-2B	LOCO-3D	LOCO-SI	LCL-3D	CALIC	Linear-3D
Indiana	8.860	7.092	6.4	6.787	6.536	6.221	6.508	6.16
Kennedy	8.480	7.514	6.182	7.156	6.397	6.17	6.719	5.634
Moffet	8.962	6.944	6.486	6.791	6.452	6.403	6.791	6.386
Parguera	9.409	8.493	7.874	8.253	7.792	7.823	7.777	7.757

Table 6 Compression ratio results based on entropy of Table 5

Image	LOCO-I	LOCO-2B	LOCO-3D	LOCO-SI	LCL-3D	CALIC	Linear-3D
Indiana	1.692	1.875	1.768	1.836	1.929	1.844	1.948
Kennedy	1.597	1.941	1.677	1.876	1.945	1.786	2.13
Moffet	1.728	1.85	1.767	1.86	1.874	1.767	1.879
Parguera	1.413	1.524	1.454	1.54	1.534	1.543	1.547

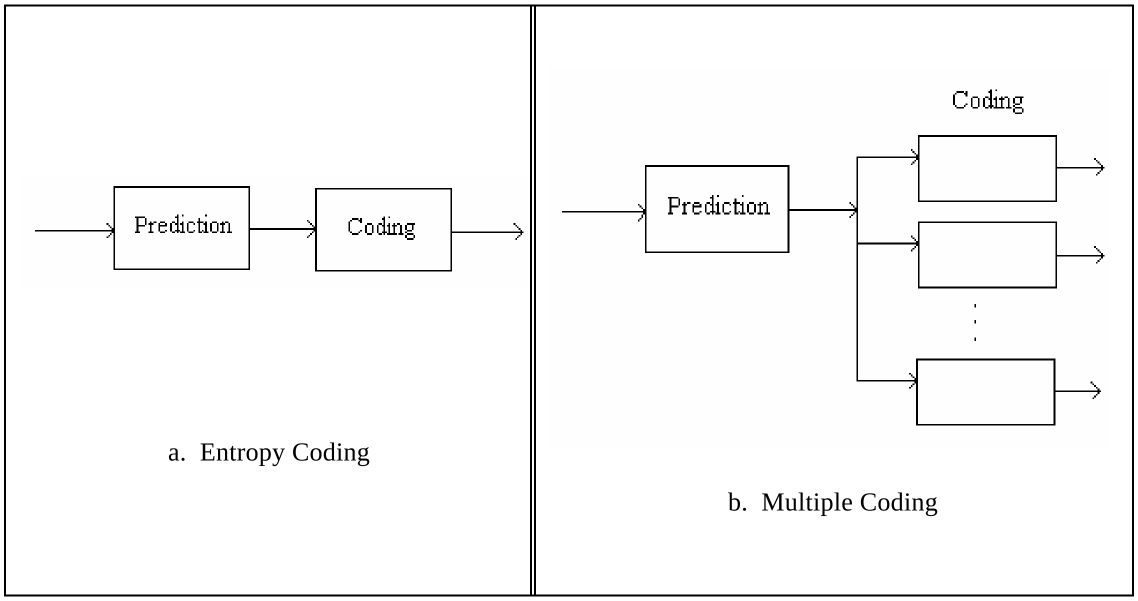
In Table 6 it can be observe the compression ratio results when applying the different predictors to the hyperspectral images. These compression ratios were obtained using the value of the first order entropy for each predictor. In other words, this table represents the bits per pixel ratio of the original image (i.e. 12 bits per pixel) divided by the entropy value obtained for each case. These results are base in the existence of only one code to do the compression, which gives a lower compression ratio. For this reason it was decided to use a different coding method in order to improve the compression ratios. The results of applying this coding method are mentioned above.

4.3 Predictor Comparison

In this section, the compression results obtained for each of the images using each of the algorithms described in Chapter 3 will be presented and compared to LOCO-I and CALIC- Extended. Compression results indicate the amount of bits per pixel that were used by each band of the image. The following sections present the results obtained for each of the images used in this research. Also, the compression ratios that were obtained by using the predictors described in the previous chapter will be presented.

In order to compress data two stages are needed. The first stage is defined prediction and the second coding. This coding stage is based on multiple codes (See Figure 6). This method has been widely adopted, since it improves the compression results. Using one code is optimum when the data to be predicted comes from a source with one distribution. This is not the case for images, so having more than one code leads to better error coding results.

The results of Table 6 are the optimal compression results using one code. Since the implemented algorithms use multiple codes, the results in Table 7 are better than the optimal in Table 6.



a. Entropy Coding

b. Multiple Coding

Figure 6 Coding Model

4.3.1 Summary Results

Table 7 Compression ratio results using different hyperspectral images.

Image	LOCO-I	LOCO-2B	LOCO-3D	LOCO-SI	LCL-3D	CALIC	Linear-3D	Average
Indiana	2.074	2.818	2.744	2.731	2.908	2.159	2.970	**2.629**
Kennedy	2.695	2.902	2.838	2.881	2.947	2.463	2.783	**2.787**
Moffet	1.908	2.982	2.603	2.947	2.981	2.301	2.990	**2.673**
Parguera	2.631	3.228	3.147	3.211	3.247	3.209	3.132	**3.115**
ParkLot	1.281	1.318	1.319	1.335	1.328	1.405	1.346	**1.333**
Average	**2.118**	**2.650**	**2.530**	**2.621**	**2.682**	**2.307**	**2.644**	

Table 7 is a summary of all the compression ratio results for the different compression algorithms and hyperspectral images used in this research. The predictors that provided the best results in terms of compression ratio average were the LCL-3D and LOCO-2B predictors. As it can be seen in this table, the best compression ratio was obtained with the image of La Parguera coming from the hyperion sensor. This is due to the lack of edges in this image, seen as small changes between pixels *y* and the pixels that are going to be use for the prediction of pixel *y* are very similar, therefore, better prediction results can be obtained and also better compression results. The LCL-3D predictor is a good predictor for this type of image, giving the best results with the Kennedy and La Parguera images. However the best results for the Indiana and Moffet image were obtained using the Linear-3D predictor, since these images have a large amount of color intensity change. The average compression ratios for all the images of the AVIRIS sensor were very similar to each other. In general, the three best compression results were obtained with the following predictors: LCL-3D, LOCO-2B and Linear-3D.

4.3.2 Northwest Indiana Pine Test Site Image Results

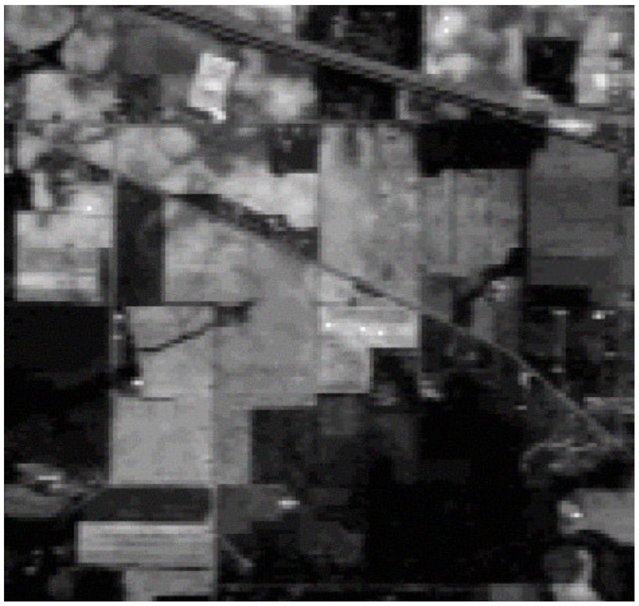


Figure 7 North West Indian Pine Test Site Image

In this section, it will be presented the results obtained by applying the algorithms mentioned in the previous chapter to the Northwest Indian Pine Test Site image in Indiana from an AVIRIS sensor. Each pixel has a twelve bits precision.

Table 8 Compression results (bits/pixel) using the Indiana image (220x145x145).

Image	**LOCO-I**	**LOCO-2B**	**LOCO-3D**	**LOCO-SI**	**LCL-3D**	**CALIC**	**Linear-3D**
Indiana	5.786	4.258	4.373	4.395	4.127	5.558	4.041

Table 9 Compression ratio results using the Indiana image (220x145x145), 12 bits/pixel.

Image	**LOCO-I**	**LOCO-2B**	**LOCO-3D**	**LOCO-SI**	**LCL-3D**	**CALIC**	**Linear-3D**
Indiana	2.074	2.818	2.744	2.730	2.907	2.159	2.970

Table 9 shows the compression ratio results obtained using the different predictors. The best result in terms of compression ratio was obtained using the Linear-3D predictor. In other words, this algorithm is the one that has the most compression. The LCL-3D predictor however, gave us good results in terms of compression, being the second best. Nevertheless, when compared to predictor Linear-3D, predictor LCL-3D has a lower computational complexity. Therefore, as a result of its good compression performance as well as its low complexity, LCL-3D was the best predictor for the Indiana image. Table 8 shows the results of bits per pixel that are obtained when using the different predictors. These results show the same behavior as the results of the compression ratio.

When implementing compression algorithms the desire is to have a prediction error as small as possible. For this reason an analysis of the compression error obtained in the 55th band will be performed for this image. These images represent the magnitude of the prediction error. These magnitudes are represented colors. As an example, the darkest color represents the highest prediction error, while the lightest color represents the smallest prediction error.

Figure 8 represents the prediction error image when using the algorithm LOCO-I. As it can be seen for this predictor it was obtained relatively high prediction errors. Base in the prediction method employed by LOCO-I, it can be concluded that this results can that the method of detecting edges is not the most efficient. These results can be compared to the results of the original image (see Fig. 7). The problem of this predictor is the type of window that uses to perform the prediction. Since this window has a square shape, in other words it only uses three neighbor pixels for pixel y. If a non-square object is found, this will create an issue in the detection. For this reason, the edges analysis is not that efficient for these particular cases. An example of this can be seen when trying to predict the roads in the image. It can be seen that the prediction error for this portion of the image is considerably high; this is caused by the diagonal way that the roads were captured. When trying to move thought the image, there is not the necessary information to detect the roads. In other words, the amount of pixels for the roads is not enough when applying this window making the detection of the object or the edges a very difficult task for the predictor. The same results are obtained when using predictor LOCO-3D. This is because both algorithms LOCO-I and LOCO-3D use the same analysis to detect edges. However, in Figure 9, it can be observed that prediction errors are lower than the results obtained when using predictor LOCO-I. This is because this algorithm uses a different equation than the one used in LOCO-I when no edge is detected. By using equation (3.24) instead to the equation proposed by

LOCO-I gives better compression results. It can be concluded that adding spectral information to the prediction gives better compression results.

In Figure 10 can be observe the prediction error image when using predictor CALIC- Extended to perform the compression. It can be said that the compression errors for this algorithms are better that those of algorithms LOCO-I and LOCO-3D. The reason behind these results is that CALIC- Extended uses more pixels to perform the prediction than LOCO-I and LOCO-3D and also CALIC- Extended uses the information of the adjacent band.

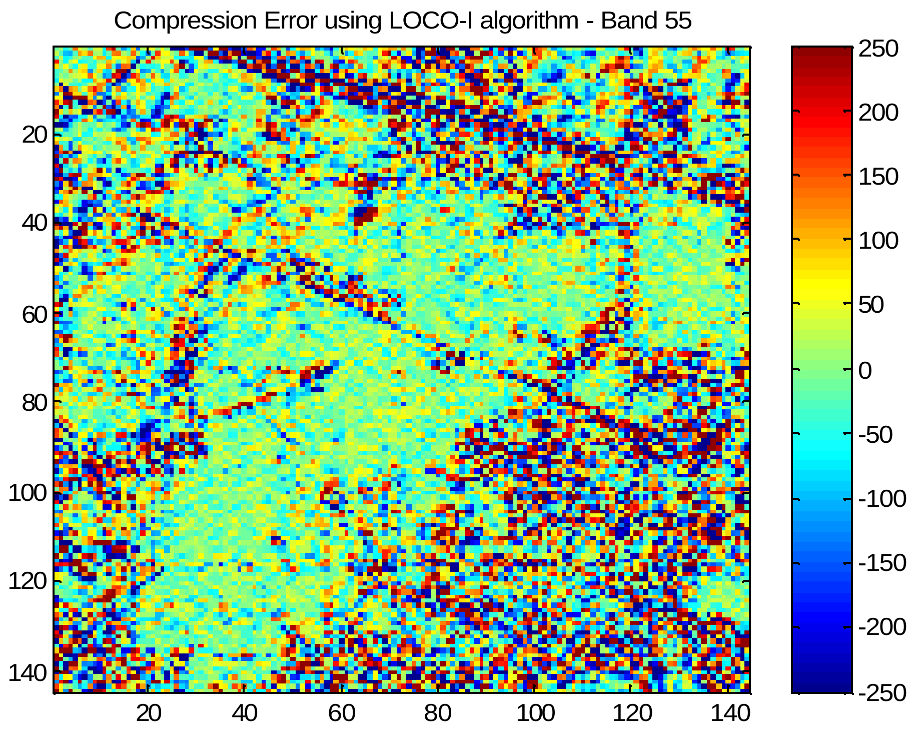


Figure 8 Prediction Error using LOCO-I algorithm

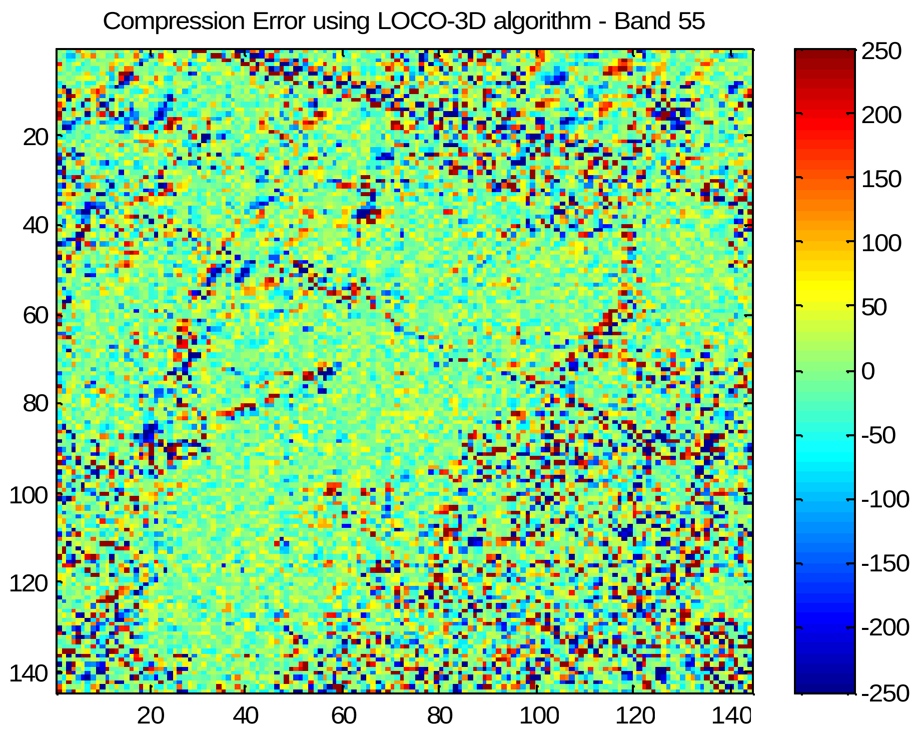


Figure 9 Prediction Error using LOCO-3D algorithm

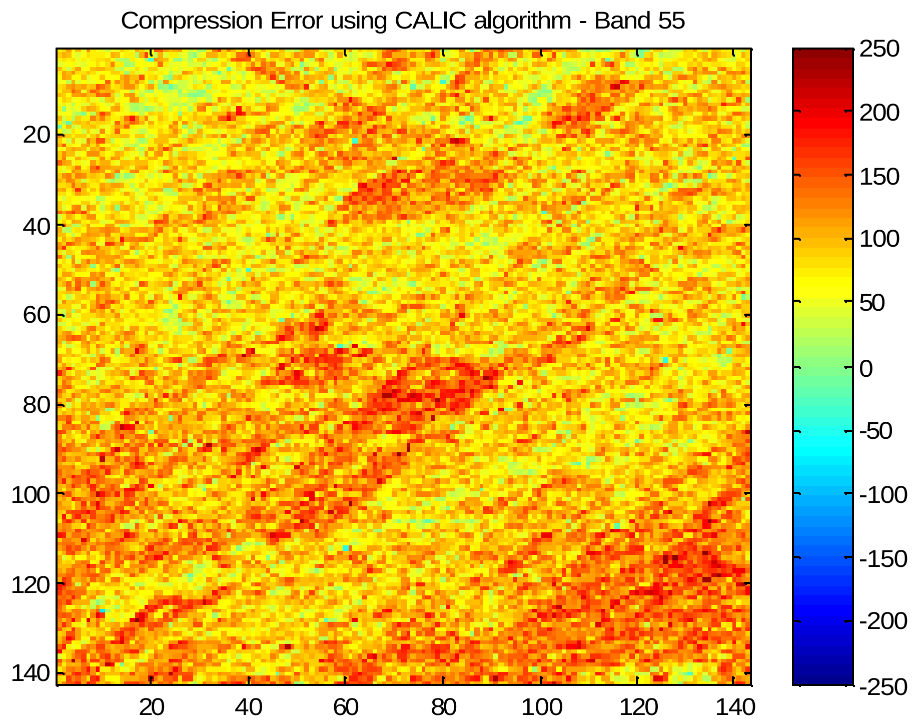


Figure 10 Prediction Error using CALIC- Extended algorithm

The following figures were the best results obtained. What these two predictors have in common is that both used the spectral and spatial information simultaneously, giving the best results in terms of compression performance, since the prediction errors are considerably small. Although, it can be observed that the prediction error for the predictors LOCO-SI and LOCO-2B gave good results, even though they are base in the same method of detecting edges (See Fig. 11 and 12). Opposite to the algorithms previously mentioned algorithm LOCO-2B use the spectral and spatial information simultaneously. This is only an improvement step of the compression. The last step and the most important step are using equation (3.30). This equation is the one that allows the prediction error to be considerably small. If this equation were omitted, the results would be similar to the results obtained when using predictor LOCO-I. In algorithm LOCO-SI the first step is to determine the similarity of the pixels between the previous band and the band where the prediction is taking place. If a similarity is found equation (3.33) is used and if no similarity is found the same analysis as LOCO-I is performed.

Figures 13 and 14 represent the prediction error when using predictor LCL-3D and Linear-3D respectively. The compression results obtained by both predictors were considerably small. It can be concluded that the best compression results are obtained when spectral and spatial information are used simultaneously.

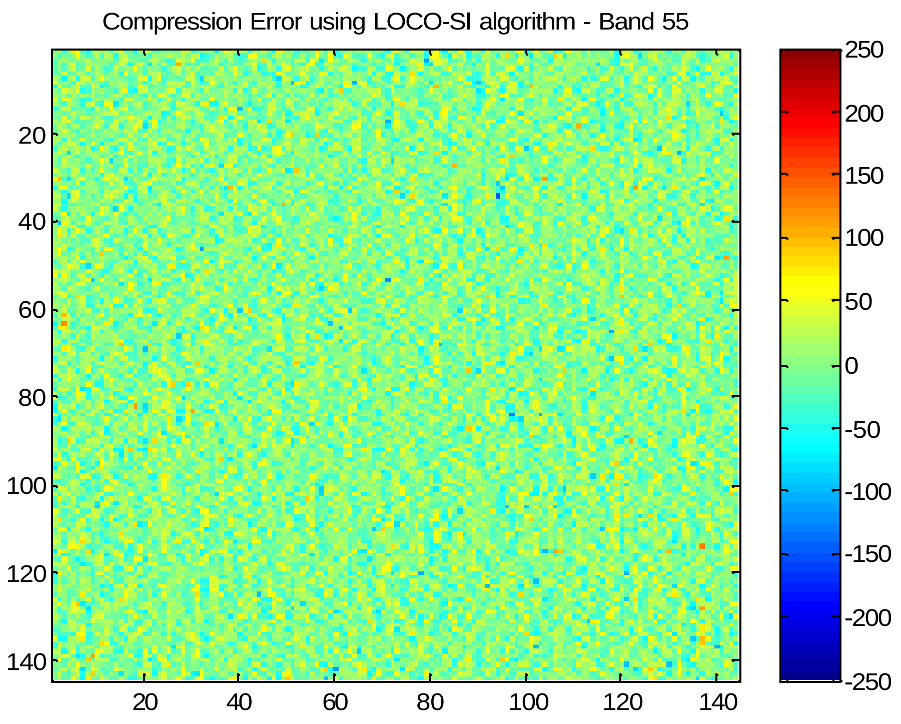


Figure 11 Prediction Error using LOCO-SI algorithm

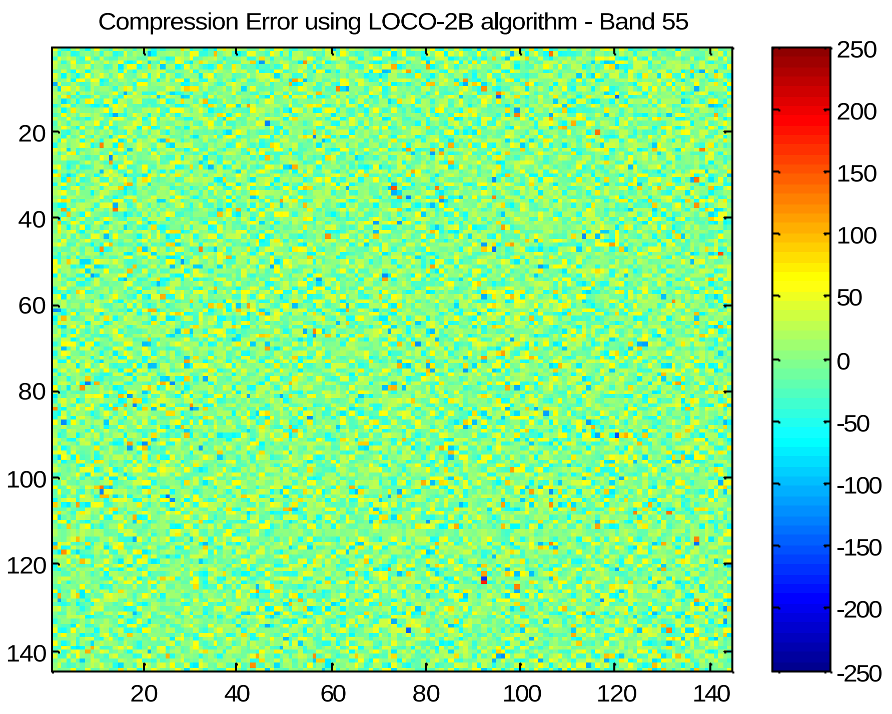


Figure 12 Prediction Error using LOCO-2B algorithm

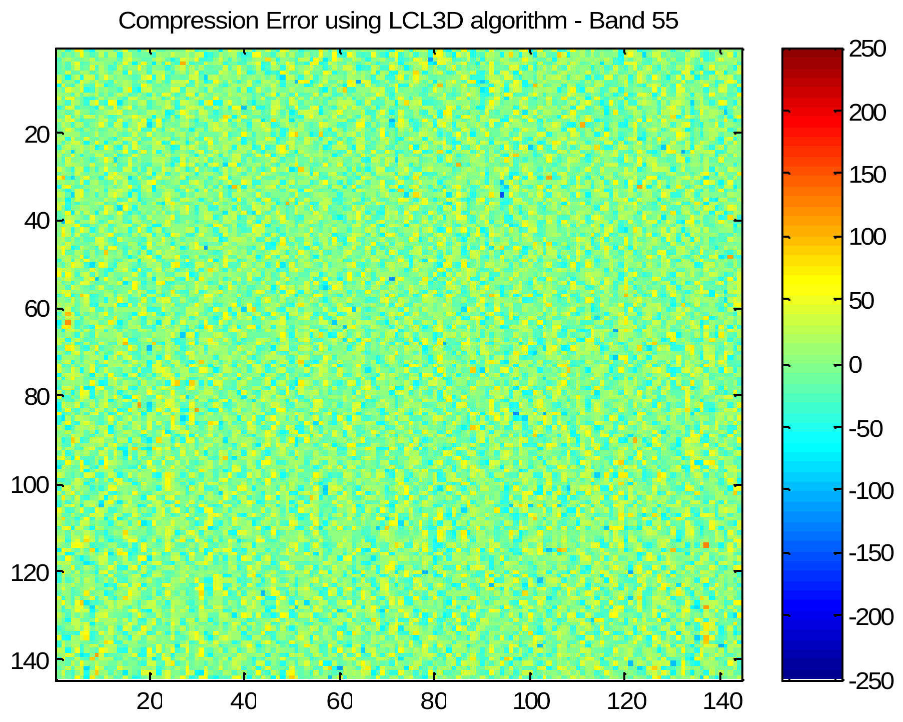


Figure 13 Prediction Error using LCL-3D predictor

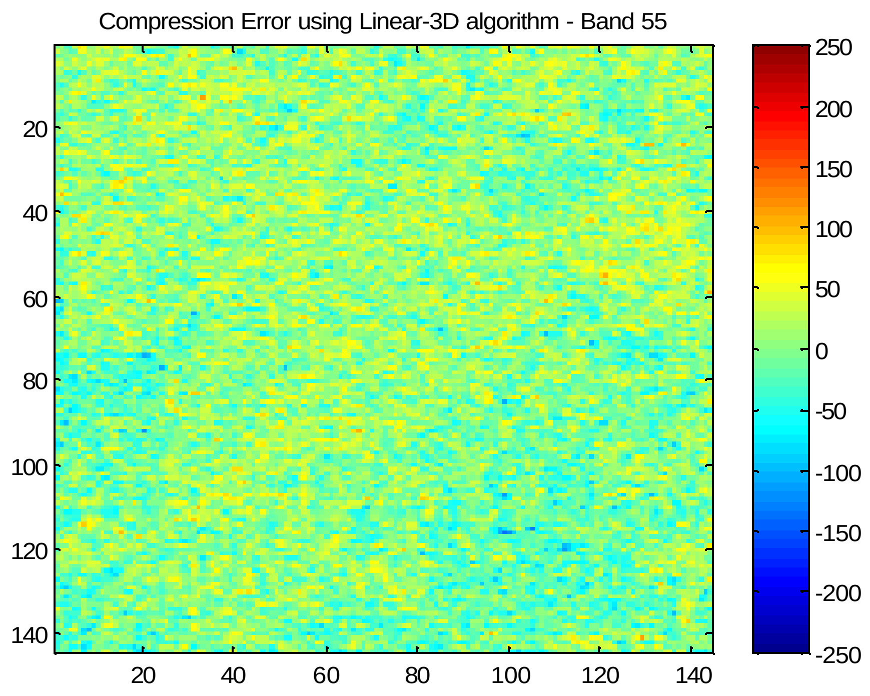


Figure 14 Prediction Error using Linear 3D predictor

4.3.3 Moffet Field Image Results

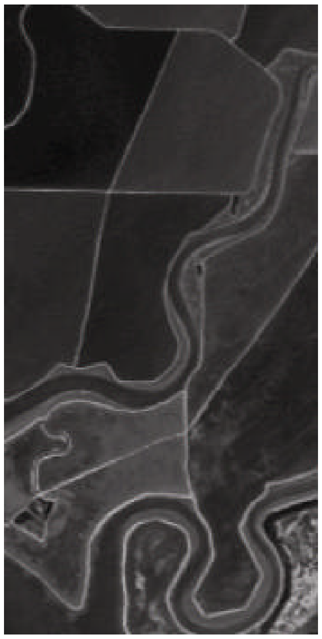


Figure 15 Moffet Field Image

In this section it will be presented the results that were obtained when applying the different algorithms previously mentioned to Moffet Field in California from an AVIRIS sensor. (See Fig. 15)

Table 10 Compression results (bits/pixel) using the Moffet image (132x294x148), 12 bits/pixel

Image	**LOCO-I**	**LOCO-2B**	**LOCO-3D**	**LOCO-SI**	**LCL-3D**	**CALIC**	**Linear-3D**
Moffet	4.453	4.136	4.228	4.165	4.072	4.872	4.312

Table 11 Compression ratio results using the Moffet image (132x294x148)

Image	**LOCO-I**	**LOCO-2B**	**LOCO-3D**	**LOCO-SI**	**LCL-3D**	**CALIC**	**Linear-3D**
Moffet	2.694	2.901	2.838	2.881	2.947	2.463	2.783

Table 10 shows the compression results obtained using the multiple predictors previously mentioned. It can be seen that the average bit rate of the proposed algorithms is better than those of CALIC- Extended and LOCO-I. However, the predictor that gave us the best average of bits per pixel for the Moffet image was the LCL-3D predictor. The same behavior was seen in Table 11, which provides the different compression ratios when using the different algorithms in order to predict pixel *y*.

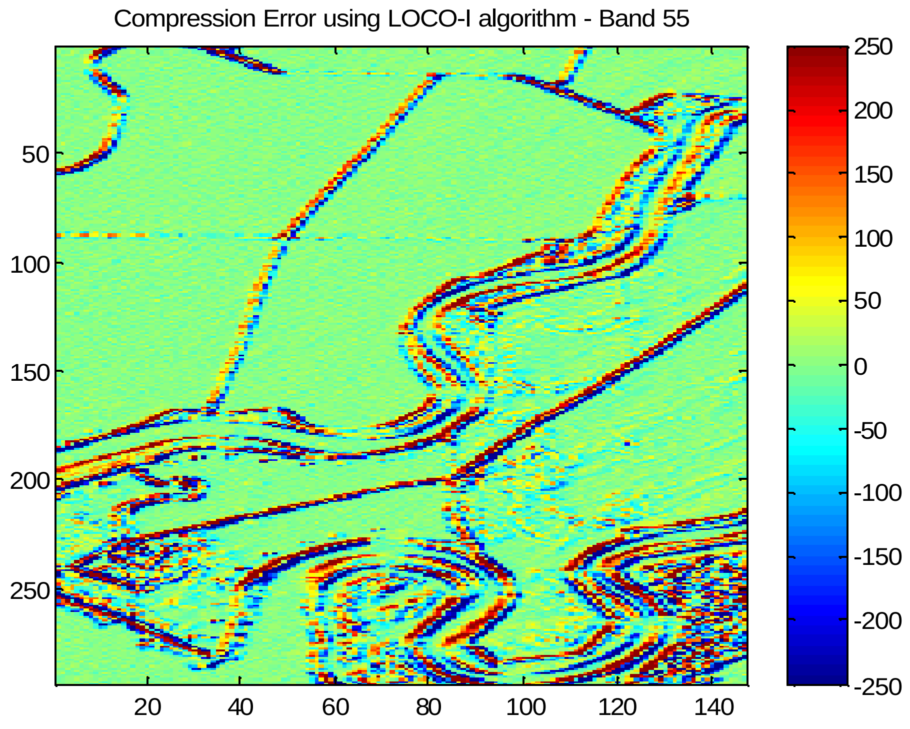


Figure 16 Prediction Error using LOCO-I algorithm

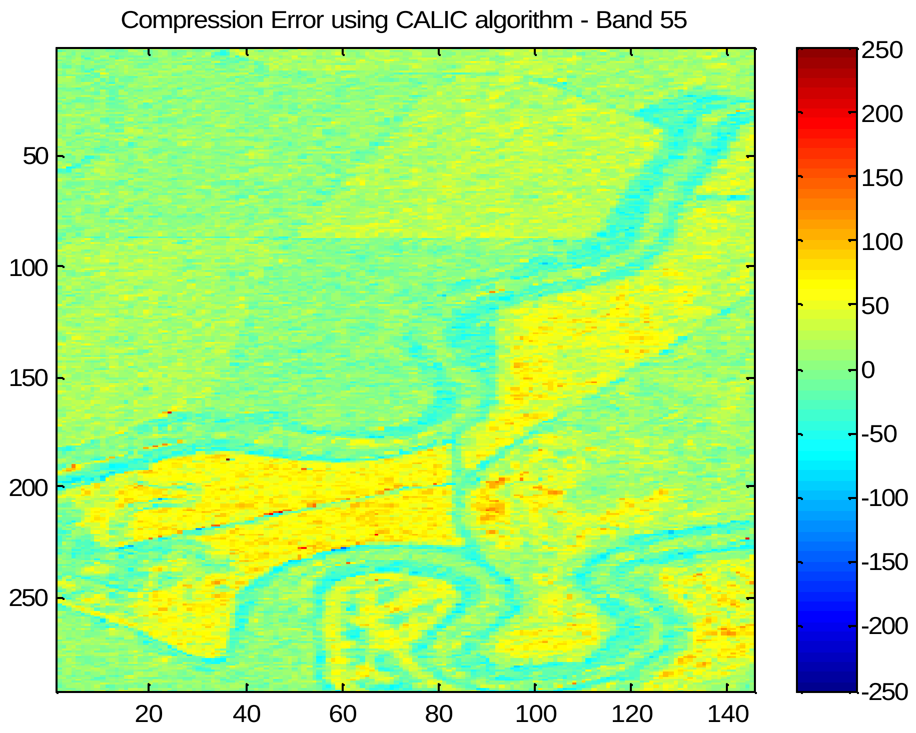


Figure 17 Prediction Error using CALIC- Extended algorithm

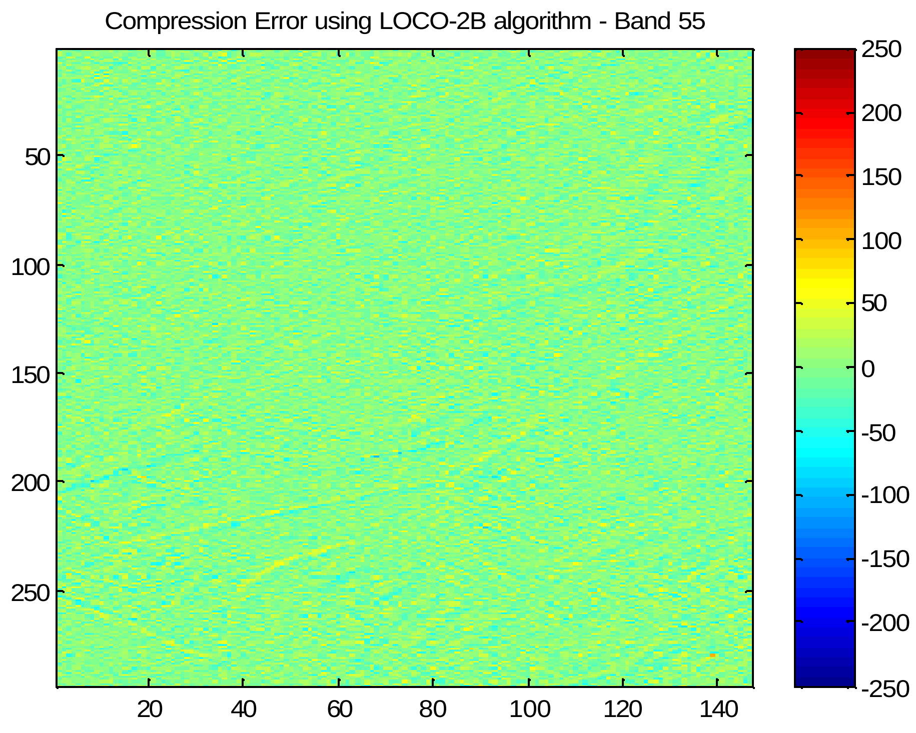


Figure 18 Prediction Error using LOCO-2B algorithm

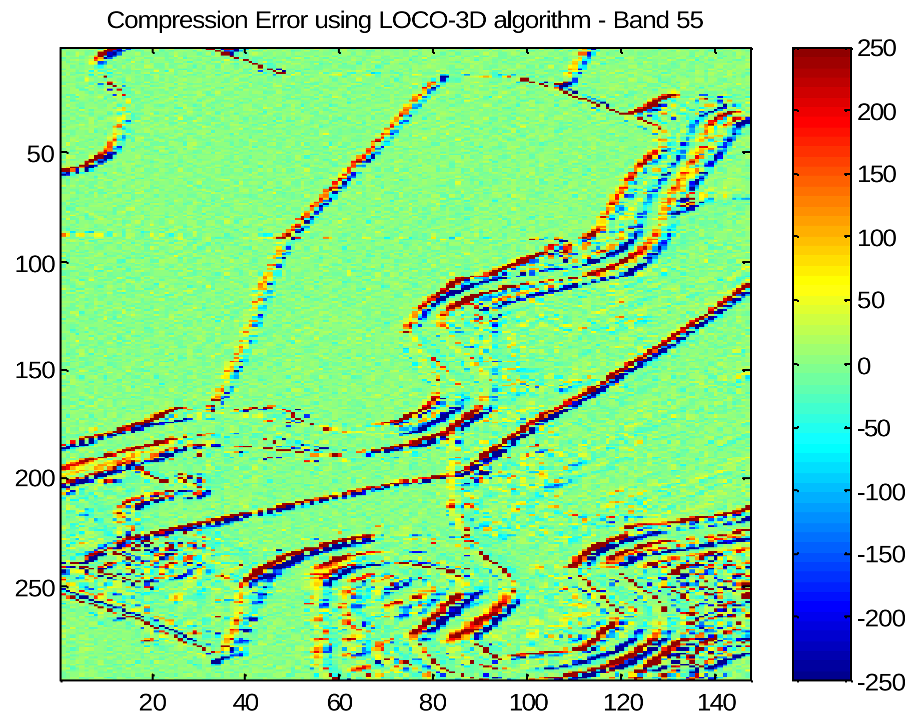


Figure 19 Prediction Error using LOCO-3D algorithm

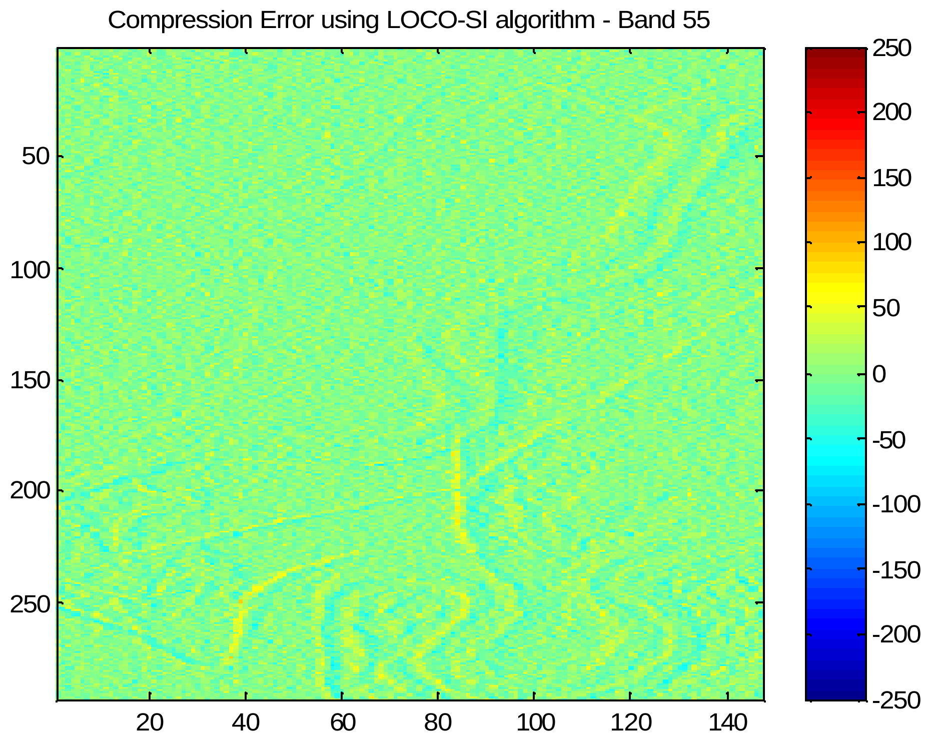


Figure 20 Prediction Error using LOCO-SI algorithm

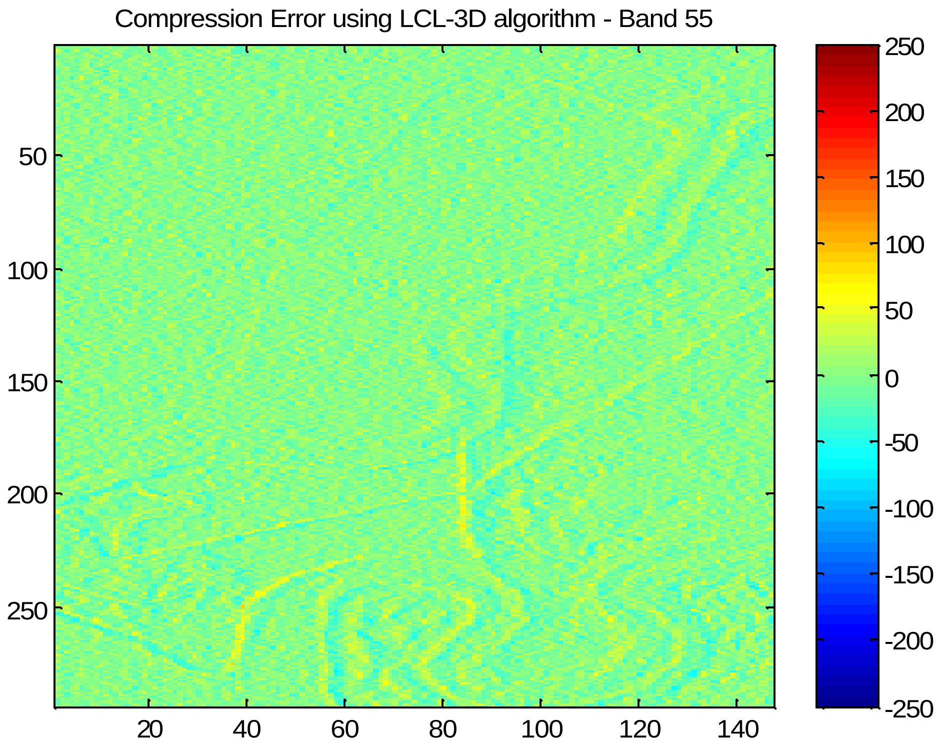


Figure 21 Prediction Error using LCL-3D predictor

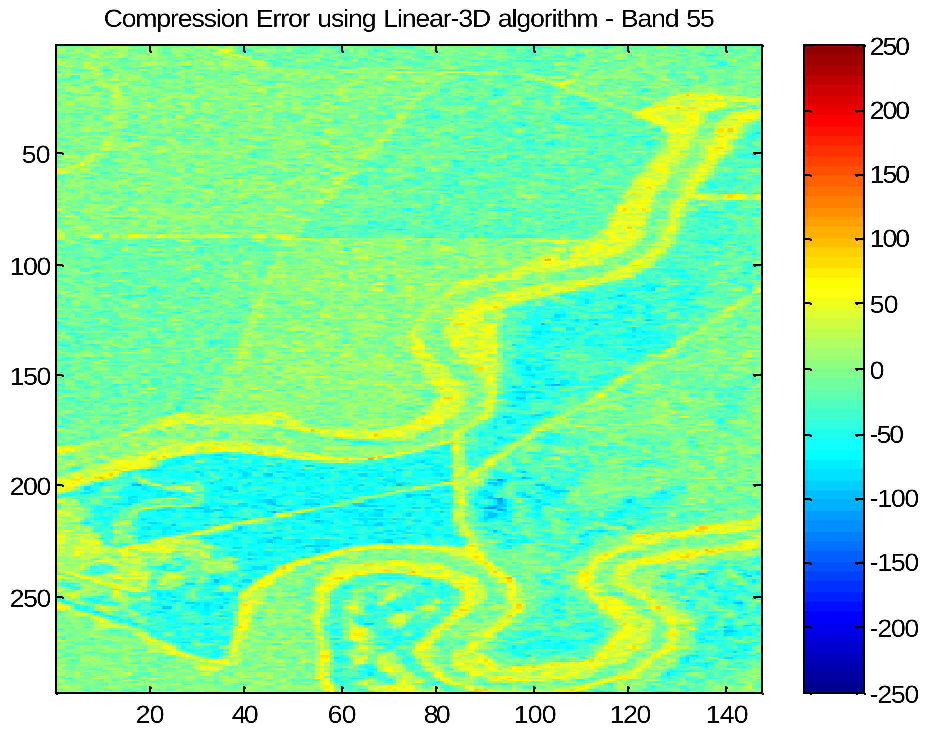


Figure 22 Prediction Error using Linear 3D predictor

From figure 16 to figure 22 it will be presented the prediction error of the 55th band obtained from the different compression algorithms implemented during this research. As a result of the observation of the prediction error images analyzed it can be said that predictor LOCO-2B was the one that showed the lowest prediction error. It can be noticed that predictors LOCO-3D and LOCO-I gave the highest prediction error in comparison to the other predictors, since the prediction error for all other algorithms were in the range of –50 and 50, this range being a considerably small prediction error. The same behavior can be observed in the prediction error plots obtained for the Indiana image.

4.3.4 NASA Kennedy Space Image Results

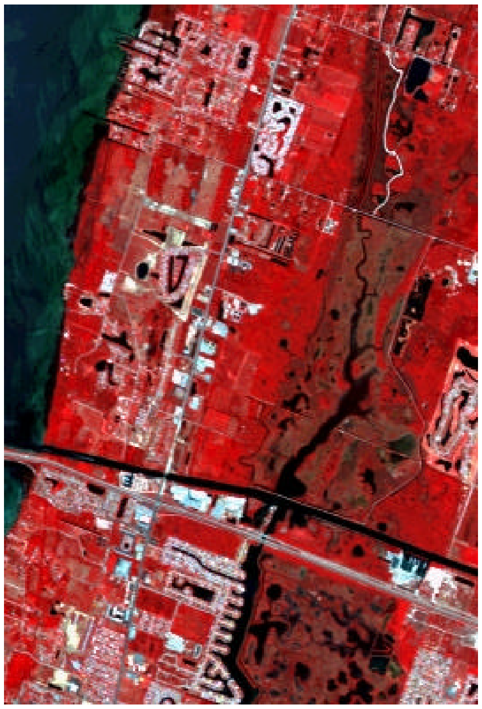


Figure 23 NASA Kennedy Space Center Image

This section presents the results obtained when using the image of the NASA Kennedy Space Center. (See Fig. 23). It will be presented the compression results for each band of this image.

Table 12 Compression results (bits/pixel) using Kennedy image (195x397x268), 12 bits/pixel

Image	**LOCO-I**	**LOCO-2B**	**LOCO-3D**	**LOCO-SI**	**LCL-3D**	**CALIC**	**Linear-3D**
Kennedy	6.291	4.024	4.610	4.072	4.026	5.215	4.013

Table 13 Compression ratio results using Kennedy image (195x397x268)

Image	**LOCO-I**	**LOCO-2B**	**LOCO-3D**	**LOCO-SI**	**LCL-3D**	**CALIC**	**Linear-3D**
Kennedy	1.907	2.982	2.603	2.947	2.980	2.301	2.990

It can be observed in Table 12 that the three predictors that resulted with the lowest amount of bits per pixel were LOCO-2B, LCL-3D and the Linear-3D predictor. In addition, it can be observed that the results of these three algorithms are very similar among them. In the case that one wants to select the best predictor among these in terms of compression performance for this image, one can choose Linear-3D as the best predictor for the Kennedy image. In Table 13 it can be observed that the compression ratio obtained with the different predictors implemented in this research were smaller in comparison with the results obtained by algorithms LOCO-I and CALIC- Extended. Also, it can be said that the results were close to each other.

It can be noticed that for both the Indiana and Kennedy images good compression performance was obtained using predictor Linear-3D. Nevertheless, for the Moffet Field image, the best results were obtained using predictor LCL-3D. However, all the compression results obtained by the algorithms developed during this research had a better performance in comparison to those algorithms already present in literature.

4.3.5 La Parguera Image Results

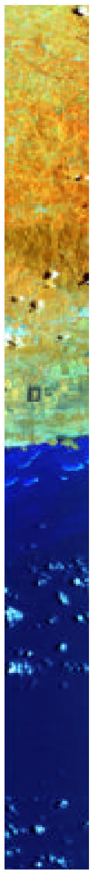


Figure 24 Image of the Area of La Parguera, PR

This image is from La Parguera, in Puerto Rico and it was captured by the Hyperion sensor at a precision of 16 bits per pixel. (See Fig. 24). As it was done on the preceding images, it was also applied the previously mentioned compression algorithms to this particular image. Compression results obtained with the algorithms proposed are for the most part very similar. For this reason, it is very difficult to select or discriminate on which algorithm revealed the best performance.

Table 14 Compression results (bits/pixel) using La Parguera image (169x3128x256), 16bits/pixel

Image	LOCO-I	LOCO-2B	LOCO-3D	LOCO-SI	LCL-3D	CALIC	Linear-3D
Parguera	6.082	4.958	5.085	4.983	4.928	4.986	5.110

Table 15 Compression ratio results using La Parguera image (169x3128x256)

Image	LOCO-I	LOCO-2B	LOCO-3D	LOCO-SI	LCL-3D	CALIC	Linear-3D
Parguera	2.630	3.227	3.146	3.211	3.246	3.209	3.131

Basing our selection on the results on Table 14 it can be determine that predictor LCL-3D resulted in the lowest average of bits per pixel in comparison to the rest of the compression algorithms. In conclusion, for this image the best predictor in terms of compression performance was LCL-3D. As the cases previously presented, the algorithms proposed by this research performed best. The compression ratios obtained by the different predictors are very similar. Observing Table 15 it results very difficult to discriminate between one predictor versus the other. However, it can be observed that the compression ratio obtained by using algorithm LOCO-I was the highest, being this algorithm the one that provide the lowest compression performance.

4.3.6 Parking Lot Image Results

Figure 25 Image of a Parking Lot Area in San Diego

The Parking Lot image was captured by Hyperspectral camera, model SOC 700. The image was captured at a twelve bits precision. This image is composed of 120 bands, 640 rows and 640 columns. (See Fig. 25) In this section, it will be presented the same analysis that was performed on previous images in order to obtain the best compression algorithm for compression performance in this particular image.

Table 16 Compression results (bits/pixel) using the Parking Lot image (120x640x640)

Image	**LOCO-I**	**LOCO-2B**	**LOCO-3D**	**LOCO-SI**	**LCL-3D**	**CALIC**	**Linear-3D**
Parking	9.370	9.107	9.095	8.991	9.039	8.538	8.917

Table 17 Compression ratio results using the Parking Lot image (120x640x640), 12bits/pixel

Image	**LOCO-I**	**LOCO-2B**	**LOCO-3D**	**LOCO-SI**	**LCL-3D**	**CALIC**	**Linear-3D**
Parking	1.280	1.317	1.319	1.334	1.327	1.405	1.345

The best compression result for this particular image was obtained using the CALIC- Extended algorithm. Contrary to the results of previous images, one can observe that, for this particular image, the algorithms proposed in this research did not provide us with the best results in comparison to the well-known and widely used CALIC- Extended algorithm. However, it is worth mentioning that all the images analyzed by the algorithms proposed in this research performed better than those in which the LOCO-I algorithm is employed. This same behavior can be observed in Table 17, since CALIC- Extended was also the predictor with the best compression ratio results. However, in this table shows that the compression ratio is quite small. This is due to the different color and the large amount of different objects that the image has.

4.4 Time Complexity Analysis

The main objective of this work is to find a predictor that gives good compression results, but at the same time with a lower computational complexity in comparison to those algorithms already existent in the literature. In order to determine the complexity of a predictor the time that a predictor takes to perform a prediction will be determine. This is known as a time complexity analysis. Currently, the most widely used compression algorithm is LOCO-I, due to its low computational complexity. Additionally, the time complexity analysis of the algorithms present in literature was computed, in order to have a performance reference. The time complexity analysis of these algorithms was found by measuring the execution time that each predictor took to perform the prediction for each of the images. The relation of bands, rows and columns were eliminated from all the images, obtaining the real execution time of the algorithm regardless the size of the image. For all the algorithms it was found the following time complexity:

$$T(BxMxN) = k(BxMxN) \tag{4.2}$$

where B represents the number of bands, M the number of rows, N the number of columns and k represents the execution time. Algorithms with the type of time complexity shown by equation (4.2) are known as linear-time algorithms, since their time complexities are linear in terms of the input, in this particular case $BxMxN$. All these algorithms are very efficient, and so have a fast execution time.

Since all the algorithms submitted to analysis have the same time complexity, it is very difficult to indicate which algorithm is best in terms of the order obtained. For this reason, the execution time of each algorithm in order to predict the pixels will be used. Table 18 shows the different execution times for each of the algorithms implemented in this research.

Table 18 Execution time for the different algorithms used in this research.

	LOCO-I	**LOCO-2B**	**LOCO-3D**	**LOCO-SI**	**LCL-3D**	**CALIC**	**Linear-3D**
k (ms)	62.931	82.517	69.653	86.633	24.977	679.476	172.991

Based on the results obtained in Table 18, the algorithm with the least execution time is the LCL-3D predictor, followed by LOCO-I, which is the most widely used algorithm.

4.5 Order Analysis

Another method to determine the complexity of an algorithm is the order. To determine the order of the algorithms, the procedure was to determine how many operations needed to be executed in order to perform the prediction. The following table shows the order of the different predictors that were used for this work.

Table 19 Predictors' Order

	LCL-3D	**LOCO-I**	**LOCO-SI**	**LOCO-3D**	**LOCO 2B**	**CALIC**	**Linear-3D**
Order	2BMN	4BMN	8BMN	4BMN	10BMN	104BMN	23BMN +300B

It can be seen in Table 19 that all the predictors have a linear order. This can be described in the following equation:

$$O(B\times M\times N) = k_1(B\times M\times N) + k_2, \tag{4.3}$$

where all the algorithms depend in the number of bands (B), number of rows (M) and number of columns (N). The operations to perform the prediction are represented with the variable k_1, and k_2 represents the operations that do not dependent on the input (B x M x N).

This table however shows that the LCL-3D predictor uses fewer operations to perform the prediction. Other predictors that have a small number of operations are LOCO-I and LOCO-3D, both taking the same amount of operations to do the prediction.

Chapter 5: Conclusions and Future Works

5.1 Compression Performance

In this research it was proposed several lossless compression algorithms for hyperspectral images. These algorithms were compared to the best lossless compression algorithms existent in literature so far. When comparing lossless image coding algorithms, it is needed to consider the compression performance as well as computational complexity. Table 7 shows compression ratio results obtained using the different predictors. Those algorithms proposed for this research achieved the best compression ratio results.

The number of bits per pixel that is obtained when using predictor LCL-3D is the lowest in comparison to the other algorithms implemented in this research. CALIC-Extended algorithm is used as point of comparison between my research and traditionally used algorithms, since this algorithm has been used by the literature for several years as basic reference due to its compression performance. Predictor LCL-3D is the algorithm with the best compression performance for hyperspectral images, since its results are better when compared to the results obtained by the CALIC- Extended algorithm.

5.2 Computational Complexity

When an algorithm is being developed and implemented is very important to consider its computational complexity. In order to determine how complex an algorithm is, it was used as a point of comparison algorithm LOCO-I. LOCO-I is known as the best algorithm in literature in terms of its computational complexity. Based on the results of Table 18 it can be determine that predictor LCL-3D has a better execution time than algorithm LOCO-I. These results show that the computational complexity of LCL-3D is lower than that of the better known and widely used algorithm LOCO-I. Also, the LCL-3D algorithm has a smaller order in comparison to the algorithms existent in the literature. (See Table 19) However, the LOCO-3D algorithm performs the same amount of operations than LOCO-I. In general, the algorithms developed during this work are simpler and they have low computational complexity.

5.3 Conclusions

During this research it was proposed new lossless compression algorithms for hyperspectral images. These new algorithms gave considerably good results in comparison to the compression results obtained by both LOCO-I and CALIC- Extended algorithms. This tells that, when spectral and spatial information is used simultaneously, a better prediction can be achieved since more information is provided; in this way one can obtain better compression results. Predictor LCL-3D is the best predictor presented in this research due to its good compression results and low complexity in comparison to all the algorithms implemented during this research. However, this predictor is not the most efficient for images with large variability (large amount of objects). The best predictor for this type of images is the Linear-3D, but this has a higher computational complexity. Another predictor that shows to have good results in term of compression regardless the type of image is the predictor LOCO-2B. In general, it can be concluded that all the predictors developed during this work give better compression results in comparison to those algorithms existent in the literature.

5.4 Future Works

The following future works can be performed in order to improve the results:

- Compression results obtained by predictor Linear-3D are very good in comparison to those algorithms in literature. The main problem with this algorithm is its high computational complexity. In order to improve this issue one can consider the evaluation of new methods for selecting rules. Two examples of these methods can be Fuzzy Logic and Neural Networks.
- The compression algorithms developed in this research use the same coding method used by algorithm LOCO-I. The results obtained can be improved by implementing a new method of coding, which can include spectral information, since spectral information improves the compression performance of a predictor.

References

[1] John A. Richards and Xiuping Jia. “Remote Sensing Digital Image Analysis”. Springer-Verlag Berlin Heidelberg New York, Third Edition, 1999.

[2] M. J. Weinberger, G. Seroussi and G. Sapiro. “LOCO-I: A Low Complexity, Context-Based, Lossless Image Compression Algorithm” Hewlett-Packard Laboratories, *Proceeding of the IEEE Data Compression Conference*, Snowbird, Utah, March-April 1996.

[3] X. Wu and N. Memon, “Context-Based Lossless Interband Compression-Extending CALIC”. *IEEE Transactions on Image Processing*, Vol. 9, No. 6, June 2000, pp. 994-1001.

[4] Howard, P.G.; Vitter, J.S., “Fast and Efficient Lossless Image Compression”, *Data Compression Conference*, 1993. DCC '93. , 30 March - 2 April 1993 pp. 351 -360.

[5] N. Tavakoli, “Short Communication Entropy and Image Compression”, *Journal of Visual Communication and Image Representation*, Vol. 4, No. 3, September 1993, pp.271-278.

[6] M. J. Weinberger, G. Seroussi and G. Sapiro. “The LOCO-I Lossless Image Compression Algorithm: Principles and Standardization into JPEG-LS”. IEEE Transactions on Image Processing, Vol. 9, No. 8, August 2000, pp. 1309-1324.

[7] G. Deng, L. W. Cahill and Hua Ye, “Lossless Image Compression Using Adaptive Predictor Combination, Symbol Mapping and Context Filtering”, *Image Processing, 1999. ICIP 99. Proceedings. 1999 International Conference*, Vol. 4 , 24-28 Oct. 1999, pp. 63 -67.

[8] A. Savakis. “Evaluation of Algorithms for Lossless Compression of Continuous-tone Images”, *Journal of Electronic Imaging*, January 2002, Vol. 11(1), pp.75-86.

[9] M. Sonka, V. Hlavac and R. Boyle. “Image Processing Analysis, and Machine Vision”. *PWS Publishing*, 1998, pp.621-645.

[11] S. D. Hunt. “White Paper on Image Processing Research”. June 1998 http://www.ece.uprm.edu/~hunt/research/whitepap.htm

[12] R. E. Neapolitan and K. Naimipour. “Foundation of Algorithms Using C++ pseudocode”. Jones and Bartlett Publishers, Susbry, Massachussetts, Second Edition, 1998.

[13] X. Wu and N. Memon. “CALIC-A Context Based Adaptive Lossless Image Codec”. *IEEE Acoustics, Speech, and Signal Processing*, 1996

[14] G. Deng, H. Ye and L. W. Cahill. “Adaptive Predictor Combination and Its Applications in Lossless Image Coding”. *IEEE Advances in Medical Signal and Information Processing*, 2000.

[15] N. Memon, Vishal Sippy and Xiaolin Wu. “A Comparison of Prediction Schemes Proposed for a New Lossless Image Compression Standard”. *IEEE Circuits and Systems*, 1996.

[16] W. Jiang and A. Ortega. “Forward/backward adaptive context selection with applications to motion vector field encoding”. *IEEE Image Processing, International Conference on* ,Oct 1997, Vol 2.

[17] B. Brower, A. Lan and J. M. McCabe, “Hyperspectral Lossless Compression”, *Part of the SPIE Conference on Imaging Spectrometry V, Denver, Colorado*, July 1999, Vol. 3753, pp. 247-257.

[18] Maragos, P.; Schafer, R.; Mersereau, R., “Two-dimensional linear prediction and its application to adaptive predictive coding of images”, *Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on* , Vol. 32 Issue: 6 , Dec 1984, Page(s): 1213 –1229.

[19] J. Makhoul, “Linear Prediction: A Tutorial Review”, *Proceedings of the IEEE*, Vol. 63, No. 4, April 1975, pp. 561-580.

Apendice: Implemented Algorithms

A.1. Linear-3D predictor

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>


#define total_col   31900 /*The number of columns in AVHRR raw data*/
#define ncols       145   /*The number of columns in each spectral band*/
#define nbands      220   /*The number of spectral bands*/
#define max_rows    145   /*The maximum number of rows to be processed*/
#define max_value   9620 /*Image values are integers from 0 to max_value-1*/
#define pred_ord    3     /*The order of the linear predictor*/
#define data_bits   14    /*Number of bits in image values*/
#define BITS 16   //Define el tamano de la data 8 o 16 bits
#define R 4
#define T 1
#define R1 1
#define R2 2
#define R3 3
#define R4 4
#define R5 5
#define R6 6
#define R7 7
#define R8 8
#define R9 9
#define T1 10
#define T2 11
#define T3 12
#define  TotalContext ((2*T + 1)*(2*R + 1)*(2*R + 1)*(2*R + 1))
#define square(a)   ((a)*(a))
#define N   (ncols*max_rows) /*The number of pixels per bands*/
#define M   20736
#define m 3 //Number of columns of! the matrix
#define n 3 //Number of rows of the matrix


/*This program reads in raw AVIRIS data and calculates 1st order
entropies of error using 2 bands for prediction.
*/
// function prototypes ////////////////////////////////////
void multiplyMatrix(double a1[][n],double a2[][m],double a3[][m],int filas,int columnas,int
numNcomun);
void GIdentity();
void ClearMatrix(double A[][m]);
void StoreG(int counter);
void MatrixTranspose(double A[][n],double C[][n]);
void QRalgorithm(double Mat[][m],int r1, int c1);
void PrintMatrix(double A[][n], int index1,int index2);

long          min,file_length;
unsigned long  count;
int    prob[2*max_value],X;
short  im_data[nbands][max_rows][ncols],im_dataB[M],im_dataC[M];
short  im_dataX[M],im_dataD[M];
int    c,npixels,nrows,err,max_err,min_err,Xnew,minE,maxE,A,B,C,D,E,g1,g2,g3,g4;
double  Ru0,Ru1,Ru2,Ru3,Ru4,Ru5,Ru6,Ru7,Ru8,Rud[pred_ord][1],detRuu,invRuu[n][m];
int counter = 0,i,j;
double H,H_cond,prob1;
char   outfile[40];
double dist,Co,Se;
double Ruu[n][m],G[n][n],Rx[n][m],RuuNew[n][m],GStore[n][n][n],GTransStore[n][n][n];
double Gt D[n][n],Q[n][n],Qt[n][n],Qr[n][n],Qd[n][1],Weigth[n];
int  Gcounter;
long   min,file_length;
unsigned long  count;
int counter1 = 0, counter;
int ContextTable[TotalContext][5];
unsigned long compare = 0;   // used to count the # of comparisons made
unsigned long swap    = 0;   // used to count the # of swaps made
float Coding(int,int);
void GenContextTable();
FILE   *ifp1,*ofp1,*ifp2;

void main()
{
        int  k1,k2,k3,k4,k5,k6,k7,k8,k9;
        float TotalCounter = 0;
        //int FreqTable[TotalContext];
        int TempContext;
        int nThreshold = 256;
        int k;
        int index;

        printf("Enter name for input file: ");
        gets(outfile);
        if((ifp1=fopen(outfile,"rb"))==NULL)
        {
                printf("fopen1 failed\n");
                exit(0);
        }


        printf("Enter name for output file1: ");
        gets(outfile);
        if((ofp1=fopen(outfile,"w"))==NULL)
        {
                printf("fopen2 failed\n");
                exit(0);
}

        /* Determine the number of rows in the AVIRIS data file*/

        c=1;
        nrows=0;
        fseek(ifp1,0,2);
        file_length=ftell(ifp1);
        file_length/=2;
        fseek(ifp1,0,0);
        nrows=file_length/total_col;

        printf("nrows=%d\n",nrows);

        if(nrows == 0)
        {
                printf("File too small to process"); exit(0);
        }
        if(nrows > max_rows)
        {
                nrows=max_rows;
        }
        npixels=nrows*ncols;
        //printf("nrows=%d npixels=%d\n",nrows,npixels);


        /* Read in raw data and put into 3D matrix */

        for(count=0;count<nrows;++count)
        {
                for(k2=0;k2<ncols;++k2)
                {
                        for(k1=0;k1<nbands;++k1)
                        {
                                c=fread(&im_data[k1][count][k2],2,1,ifp1);
                        }
                }
        }

        fprintf(ofp1,"Bandas\tEntropia\tConteo\n");
        GenContextTable(); /*Generacion tabla de contexto y tabla de frecuencia*/

        for (k7=1;k7<nbands;++k7)
        //k7 = 55;
        {
                printf("Bands %d\r",k7);

                H=0.0;
                for(k3=0;k3<2*max_value;++k3)
                {
                        prob[k3]=0;
                }
```

```
//printf("Band=%d\n",k7);
j=0;

for (k3=1;k3<max_rows;++k3)
{
	for (k5=1;k5<ncols;++k5)
	{
		im_dataB[j] = im_data[k7][k3-1][k5];
		im_dataC[j] = im_data[k7][k3][k5-1];
		im_dataD[j] = im_data[k7-1][k3][k5];
		im_dataX[j] = im_data[k7][k3][k5];
		j++;
	}
}

/*for (i=0;i<9;i++)
{
	printf("X[%d] = %d\n",i,im_dataX[i]);
	printf("D[%d] = %d\n",i,im_dataD[i]);
	printf("C[%d] = %d\n",i,im_dataC[i]);
	printf("B[%d] = %d\n",i,im_dataB[i]);
}*/

//printf("j=%d\n",j);
//getch();
Ru0=0.0;
Ru1=0.0;
Ru2=0.0;
Ru3=0.0;
Ru4=0.0;
Ru5=0.0;
Ru6=0.0;
Ru7=0.0;
Ru8=0.0;

for (i=0;i<M;++i)
{
	Ru0 = Ru0 + im_dataD[i]*im_dataD[i];
	Ru1 = Ru1 + im_dataD[i]*im_dataC[i];
	Ru2 = Ru2 + im_dataD[i]*im_dataB[i];
	Ru3 = Ru3 + im_dataC[i]*im_dataC[i];
	Ru4 = Ru4 + im_dataB[i]*im_dataB[i];
	Ru5 = Ru5 + im_dataB[i]*im_dataC[i];
	Ru6 = Ru6 + im_dataD[i]*im_dataX[i];
	Ru7 = Ru7 + im_dataC[i]*im_dataX[i];
	Ru8 = Ru8 + im_dataB[i]*im_dataX[i];

}

Ru0 = Ru0/M;
Ru1 = Ru1/M;
Ru2 = Ru2/M;
```

```
		Ru3 = Ru3/M;
		Ru4 = Ru4/M;
		Ru5 = Ru5/M;
		Ru6 = Ru6/M;
		Ru7 = Ru7/M;
		Ru8 = Ru8/M;


		/*if ((R1>R2) && (R1>R3))
		{
			++y;
		}*/


/*		printf("R0[%d] = %lf\n",k7,R0);
		printf("R1[%d] = %lf\n",k7,R1);
		printf("R2[%d] = %lf\n",k7,R2);
		printf("R3[%d] = %lf\n",k7,R3);
		printf("R4[%d] = %lf\n",k7,R4);
		printf("R5[%d] = %lf\n",k7,R5);
		printf("R6[%d] = %lf\n",k7,R6);
		printf("R7[%d] = %lf\n",k7,R7);
		printf("R8[%d] = %lf\n",k7,R8);

		printf("\n");
		getch();*/

		Ruu[0][0]=Ru0;
		Ruu[0][1]=Ru1;
		Ruu[0][2]=Ru2;
		Ruu[1][0]=Ru1;
		Ruu[1][1]=Ru3;
		Ruu[1][2]=Ru5;
		Ruu[2][0]=Ru2;
		Ruu[2][1]=Ru5;
		Ruu[2][2]=Ru4;

		Rud[0][0]=Ru6;
		Rud[1][0]=Ru7;
		Rud[2][0]=Ru8;


		for (k8=0;k8<n;k8++)
		{
			Qd[k8][0] = 0;
			Weigth[k8] = 0;
		}

		ClearMatrix(Q);
		ClearMatrix(Qt);
		ClearMatrix(Qr);

		QRalgorithm(Ruu,n,m);
```

```
        /*printf("Qt = ");
        PrintMatrix(Qt,m,n);
        printf("RuuNew = ");
        PrintMatrix(RuuNew,m,n);*/


        for (k4=0;k4<n;k4++)
        {
                for (k6=0;k6<1;k6++)
                {
                        for (k9=0;k9<m;k9++)
                        {
                                Qd[k4][k6] += Qt[k4][k9]*Rud[k9][k6];
                        }
                }
        }


                        Weigth[2] = (Qd[2][0]/RuuNew[2][2]);
                        Weigth[1] = ((Qd[1][0] -
(RuuNew[1][2]*Weigth[2]))/RuuNew[1][1]);
                        Weigth[0] = ((Qd[0][0] - (RuuNew[0][1]*Weigth[1]) -
(RuuNew[0][2]*Weigth[2]))/RuuNew[0][0]);



                        for (k3=1;k3<nrows; ++k3)
                        {
                                for (k5=1; k5<ncols; ++k5)
                                {
                        A = im_data[k7][k3-1][k4-1];
                        B = im_data[k7][k3-1][k4];
                        C = im_data[k7][k3][k4-1];
                        D =     im_data[k7][k3-1][k4 + 1];
                        E =     im_data[k7][k3][k4-2];
                        g1 = D - B;
                        g2 = B - A;
                        g3 = A - C;
                        g4 = C - E;

                                        X = ((Weigth[0]*(double)im_data[k7-1][k3][k5]) +
(Weigth[1]*(double)im_data[k7][k3][k5-1]) + (Weigth[2]*(double)im_data[k7][k3-1][k5]));
                                        err = (int)(im_data[k7][k3][k5] - X);



                        ClearMatrix(Qt);

                        Ru0=0.0;
                        Ru1=0.0;
                        Ru2=0.0;
                        Ru3=0.0;
```

```
                    Ru4=0.0;
                    Ru5=0.0;
                    Ru6=0.0;
                    Ru7=0.0;
                    Ru8=0.0;

                    if(err < (-1*(pow(2,BITS)/2)) || err > ((pow(2,BITS)/2)-1))
                    {
                            counter1++;
                    }
////////////////////G1///////////////////////////
                    if(g1 == 0)
                    {
                            TempContext = R1 * 9;
                    }
                    else if(g1 == 1 || g1 == 2)
                    {
                            TempContext = R2 * 9;
                    }
                    else if(g1 == -1 || g1 == -2)
                    {
                            TempContext = R3 * 9;
                    }
                    else if(g1 >= 3 && g1 <= 6)
                    {
                            TempContext = R4 * 9;
                    }
                    else if(g1 <= -3 && g1 >= -6)
                    {
                            TempContext = R5 * 9;
                    }
                    else if(g1 >= 7 && g1 <= 14)
                    {
                            TempContext = R6 * 9;
                    }
                    else if(g1 <= -7 && g1 >= -14)
                    {
                            TempContext = R7 * 9;
                    }
                    else if(g1 >= 15)
                    {
                            TempContext = R8 * 9;
                    }
                    else if(g1 <= -15)
                    {
                            TempContext = R9 * 9;
                    }
////////////////////G2///////////////////////////
                    if(g2 == 0)
                    {
                            TempContext += R1 * 81;
                    }
                    else if(g2 == 1 || g2 == 2)
```

```
                                        {
                                                TempContext += R2 * 81;
                                        }
                                        else if(g2 == -1 || g2 == -2)
                                        {
                                                TempContext += R3 * 81;
                                        }
                                        else if(g2 >= 3 && g2 <= 6)
                                        {
                                                TempContext += R4 * 81;
                                        }
                                        else if(g2 <= -3 && g2 >= -6)
                                        {
                                                TempContext += R5 * 81;
                                        }
                                        else if(g2 >= 7 && g2 <= 14)
                                        {
                                                TempContext += R6 * 81;
                                        }
                                        else if(g2 <= -7 && g2 >= -14)
                                        {
                                                TempContext += R7 * 81;
                                        }
                                        else if(g2 >= 15)
                                        {
                                                TempContext += R8 * 81;
                                        }
                                        else if(g2 <= -15)
                                        {
                                                TempContext += R9 * 81;
                                        }
////////////////////G3//////////////////////////
                                        if(g3 == 0)
                                        {
                                                TempContext += R1 * 729;
                                        }
                                        else if(g3 == 1 || g3 == 2)
                                        {
                                                TempContext += R2 * 729;
                                        }
                                        else if(g3 == -1 || g3 == -2)
                                        {
                                                TempContext += R3 * 729;
                                        }
                                        else if(g3 >= 3 && g3 <= 6)
                                        {
                                                TempContext += R4 * 729;
                                        }
                                        else if(g3 <= -3 && g3 >= -6)
                                        {
                                                TempContext += R5 * 729;
                                        }
                                        else if(g3 >= 7 && g3 <= 14)
                                        {
                                                TempContext += R6 * 729;
                                        }
                                        else if(g3 <= -7 && g3 >= -14)
                                        {
                                                TempContext += R7 * 729;
                                        }
                                        else if(g3 >= 15)
                                        {
                                                TempContext += R8 * 729;
                                        }
                                        else if(g2 <= -15)
                                        {
                                                TempContext += R9 * 729;
                                        }
////////////////////g4////////////////////////////
                                        if(abs(g4) < 5)
                                        {
                                                TempContext +=  T1 * 3;
                                        }
                                        else if(g4 >= 5)
                                        {
                                                TempContext +=  T2 * 3;
                                        }
                                        else if(g4 <= -5)
                                        {
                                                TempContext +=  T3 * 3;
                                        }
                                        for(counter = 0; counter < TotalContext;counter++)
                                        {
                                                if(ContextTable[counter][0] == TempContext)
                                                {
                                                        ++ContextTable[counter][1];
                                                        ++ContextTable[counter][2];//Represents N
                                                        ContextTable[counter][3] += abs(err); //Represents
A
                                                        if(ContextTable[counter][2] >= nThreshold)
                                                        {
                                                                ContextTable[counter][2] = 0;
                                                                ContextTable[counter][3] = 0;
                                                        }
                                                        else
                                                        {
                                                                for(k = 0;(ContextTable[counter][2]<<k) <
ContextTable[counter][3];k++);
                                                                ContextTable[counter][4] +=
Coding(err,k);
                                                        }
                                                        break;
                                                }
                                        }

                                        err+=max_value;
```

```
					//fprintf(ofp1,"%d\n",err);
					++prob[err];
					}
				}
				//printf("counter[%d]= %d\n", k1,	counter);
				//getchar();
				//H=0.0;
				for(k3=0;k3<2*max_value;++k3)
				{
					if(prob[k3]!=0)
					{
						prob1=(double)prob[k3]/(double)npixels;
						H+=-prob1*log(prob1);
					}
				}
				H/=log(2.0);
				for(index = 0; index < TotalContext;index++)
				{
					TotalCounter += ContextTable[index][4];
					ContextTable[index][4] = 0;
				}
				fprintf(ofp1,"%d\t%f\n",k7,TotalCounter/(ncols*max_rows));
			//	printf("%d\t%f\t%f\n",k1,H,TotalCounter/(ncols*max_rows));
				TotalCounter = 0;
				//getch();

			}
	fclose(ifp1);
	fclose(ofp1);
	printf("DONE\n");
}


/***************************************************************/
float Coding(int err, int K)
{
	double conteoTotal = 0;	//Guarda el conteo total de la imagen
	int TempValue;
	int Bin[BITS];
	int PosValue;
	int remainder;
	int number;
	int index = BITS - 1;  //variable para los conteos y moverse por los arreglos
	int counterPerCell = 0;
	double TotalCounter = 0;
	int t,x,value,counter;

	//int K = log10(BITS)/log10(2); //Buscando la K
	x = pow(2,(BITS-(K+1)));	//Valor que se define para cambiar de Binario a decimal
	value = 0;
	index = BITS - 1;	//Volviendo el index a estado original para proxima iteracion
	remainder = 0;	//Remainder = 0 para proxima iteracion de esta forma sabemos que
remainder no tiene un numero no deseado
```

```
	for( counter = 0; counter < BITS; counter++) // Se llena el arreglo Bin de 0 para evitar numeros
no deseados
	{
		Bin[counter] = 0;
	}
	if(err < 0)
	{
		TempValue = err * -1; // Convirtiendolo a positivo
		PosValue = (2 * TempValue) - 1; //Aplicando regla de 2n-1
		number = PosValue;//Asignando valor de PosValue a number para conversion a binario
	}
	else if(err >= 0)
	{
		number = 2*err;
	}
	do
	{
		remainder = number%2; //Esta parte aplica la conversion de Dec a Binario
		Bin[index] = remainder;//de la forma de division
		number = number>>1;	//Dando un shift a la derecha para reducir el numero por una
base de 2
		index--;		//El index se disminuye con la idea de llenar el arreglo de Bin de size a
0
	}while(number > 1);	//Seguir aplicando conversion mientras el numero sea mayor que 1
	if(number <= 1)//Si el numero es 1 o 0
	{
		Bin[index] = number; //Quiere decir que terminamos la conversion y el valor del
numero se convierte en el
	}
	// MSB del arreglo
	for(t=0;t<=(BITS-(K+1));t++) //Conversion de Binario a decimal de la parte de arreglo
eliminando los espacios de K
	{
		if(Bin[t] == 1)							//Continua la conversion
			value+=x;		//Value es el que guarda el valor decimal
		x/=2;	//Divide X entre dos para bajar a la proxima potencia de dos
	}
	counterPerCell = value + K;
	return counterPerCell;//Guarda el valor de cada celda de la matriz ya sumando K
}//Hasta aqui es la misma funcionalidad que el if solo que para valores positivos


void GenContextTable()
{
	int
index,tempContext,Tcounter,G1Counter,G2Counter,G3Counter,FirstContext,Taccumulator,G1accumulat
or;
	int G2accumulator,G3accuulator;
	G1Counter = G2Counter = G3Counter = -1;
	Tcounter = Taccumulator = G1accumulator = G2accumulator = G3accuulator  = 0;
	FirstContext = R1*9 + R1*81 + R1*729 + T1*3;
	for(index = 0; index < TotalContext;index++)
	{
		Tcounter = index%3;
		if(Tcounter == 0)
		{
			G3Counter++;
			Tcounter = 0;
			if(G3Counter%9 == 0)
			{
				G2Counter++;
				G3Counter = 0;
				if(G2Counter%9 == 0)
				{
					G1Counter++;
					G2Counter = 0;
				}
			}
		}
		tempContext= FirstContext + 3*Tcounter + 9*G1Counter + 81*G2Counter +
729*G3Counter;
		ContextTable[index][0] = tempContext;
		ContextTable[index][1] = 0;
		ContextTable[index][2] = 0;
		ContextTable[index][3] = 0;
	}
}

void QRalgorithm(double Mat[][m],int r1, int c1)
{
	int d1,d2,d3,d4,y1,y2,y3;
	Gcounter = 0;

	GIdentity();

/// Inicializando a cero Q y Qtranspuesta
	ClearMatrix(Q);
	ClearMatrix(Qt);
	ClearMatrix(Qr);

/// Inicializando la matriz triangular
	ClearMatrix(Rx);

	for ( d3=0;d3<r1;++d3)
	{
		for ( d4=0;d4<c1;++d4)
		{
			RuuNew[d3][d4] = Mat[d3][d4];
		}
	}


	for (d1=0;d1<c1-1; ++d1)
```

```
{
        for (d2= r1-1;d2>=d1+1; --d2)
        {
                dist = pow(RuuNew[d1][d1],2) + pow(RuuNew[d2][d1],2);
                Co = RuuNew[d1][d1]/(sqrt(dist));
                //printf("Co = %lf\n",Co);
                Se = RuuNew[d2][d1]/(sqrt(dist));
                //printf("Se = %lf\n",Se);
                G[d1][d1] = Co;
                G[d2][d2] = Co;
                G[d1][d2] = Se;
                G[d2][d1] = -Se;

                StoreG(Gcounter);
                MatrixTranspose(GTransStore[Gcounter],GStore[Gcounter]);

                Gcounter++;

                //PrintMatrix(RuuNew,m,n);
                //PrintMatrix(G,m,n);

                multiplyMatrix(G,RuuNew,Rx,r1,c1,r1);


                //PrintMatrix(R,m,n);

                GIdentity();

                for ( d3=0;d3<r1;++d3)
                {
                        for ( d4=0;d4<c1;++d4)
                        {
                                RuuNew[d3][d4] = Rx[d3][d4];
                                //printf("  %lf  ",RuuNew[k3][k4]);
                                //getch();
                        }
                        // printf("\n");
                }
                // printf("\n");


                ClearMatrix(Rx);
        }

}
//PrintMatrix(RuuNew,m,n);

ClearMatrix(GtD);

multiplyMatrix(GTransStore[0],GTransStore[1],GtD,r1,r1,r1);
multiplyMatrix(GtD,GTransStore[2],Q,r1,r1,r1);
//PrintMatrix(Q,m,n);
```

```
        MatrixTranspose(Qt,Q);
        multiplyMatrix(Qt,Q,Qr,r1,r1,r1);
        //printf("Band =  %d\n",k7);
        //PrintMatrix(Qt,m,n);
        //PrintMatrix(Qr,m,n);
        ClearMatrix(Qr);

        ClearMatrix(GtD);
        //ClearMatrix(Qt);

        return;
}

void multiplyMatrix(double A[][n],double B[][m],double D[][m],int filas,int columnas,int numNcomun)
{
        int y1,y2,y3;
        for (y1=0;y1<filas;y1++)
        {
                for (y2=0;y2<columnas;y2++)
                {
                        for (y3=0;y3<numNcomun;y3++)
                        {
                                D[y1][y2] += A[y1][y3]*B[y3][y2];
                        }
                        //printf("D[%d][%d] = %lf\n",y1,y2,D[y1][y 2]);
                        //getch();
                }
        }

}

void GIdentity()
{
        int d3,d4;

        for ( d3=0;d3<n;++d3)
        {
                for ( d4=0;d4<n;++d4)
                {
                        if (d3 == d4)
                        {
                                G[d3][d4] = 1;
                        }
                        else
                        {
                                G[d3][d4] = 0;
                        }
                }
        }
}

void ClearMatrix(double A[][m])
```

```
{
	int d3,d4;

	for ( d3=0;d3<n;++d3)
	{
		for ( d4=0;d4<m;++d4)
		{
			A[d3][d4] = 0;
		}
	}
}

void StoreG(int counter)
{
	int index, index1;

	for(index = 0; index < n; ++index)
	{
		for(index1 = 0; index1 < n; ++index1)
		{
			GStore[counter][index][index1] = G[index][index1];
		}
	}
}

void MatrixTranspose(double A[][n],double B[][n])
{
	int index, index1, index2;

//	printf("Esta son las G originales: \n");

	for(index1 = 0; index1 < n; ++index1)
	{
		for(index2 = 0; index2 < n; ++index2)
		{
			A[index1][index2] = B[index2][index1];
			//printf("%lf  ",GStore[index][index1][index2]);
		}

	}

}


void PrintMatrix(double A[][n], int index1, int index2)
{
	for(index1 = 0; index1 < n; ++index1)
	{
		for(index2 = 0; index2 < n; ++index2)
		{
			printf("%lf  ",A[index1][index2]);
		}
		printf("\n");
```

```
        }
        printf("\n");
        }
```

A.2. LCL-3D predictor

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<time.h>

#define total_col   31900 /*The number of columns in AVHRR raw data*/
#define ncols       145   /*The number of columns in each spectral band*/
#define nbands      220   /*The number of spectral bands*/
#define max_rows    145   /*The maximum number of rows to be
processed*/
//#define max_value   256 /*Image values are integers from 0 to max_value-1*/
#define max_value   9620 /*Image values are integers from 0 to max_value-1*/
#define pred_ord    3     /*The order of the linear predictor*/
#define data_bits   14    /*Number of bits in image values*/
#define BITS 16   //Define el tamano de la data 8 o 16 bits
#define R 4
#define T 1
#define R1 1
#define R2 2
#define R3 3
#define R4 4
#define R5 5
#define R6 6
#define R7 7
#define R8 8
#define R9 9
#define T1 10
#define T2 11
#define T3 12
#define  TotalContext ((2*T + 1)*(2*R + 1)*(2*R + 1)*(2*R + 1))


/*
This program reads in raw AVIRIS data and calculates 1st order
entropies of error using 2 bands for prediction.
*/


long            min,file_length;
unsigned long  count;

int    prob[2*max_value];
short  im_data[nbands][max_rows][ncols],error[21025];
int    c,npixels,nrows,err,max_err,min_err,A,B,C,C0,D0,max,min,X,j,D,E,g1,g2,g3,g4;
int counter1 = 0, counter;
```

```
double H,H_cond,prob1;
char   outfile[40];
float  W[pred_ord+1][pred_ord+1],Wp[pred_ord+1][pred_ord+1];
int ContextTable[TotalContext][5];
float  w[pred_ord];
FILE   *ifp1,*ofp1,*ifp2;
unsigned long compare = 0;   // used to count the # of comparisons made
unsigned long swap    = 0;   // used to count the # of swaps made
void   rotin(),reinitw(),block_process(),weightch();
float Coding(int,int);
void GenContextTable();

void main()
{
        int  k1,k2,k3,k4;
        float TotalCounter = 0;
        //int FreqTable[TotalContext];
        int TempContext;
        int nThreshold = 256;
        int k;
        int index;

        printf("Enter name for input file: ");
        gets(outfile);
        if((ifp1=fopen(outfile,"rb"))==NULL)
        {
                printf("fopen1 failed\n");
                exit(0);
        }


        printf("Enter name for output file1: ");
        gets(outfile);
        if((ofp1=fopen(outfile,"w"))==NULL)
        {
                printf("fopen2 failed\n");
                exit(0);
        }


        /* Determine the number of rows in the AVIRIS data file*/

        c=1;
        nrows=0;
        fseek(ifp1,0,2);
        file_length=ftell(ifp1);
        file_length/=2;
        fseek(ifp1,0,0);
        nrows=file_length/total_col;

        printf("nrows=%d\n",nrows);

        if(nrows == 0)
```

```
          {
                    printf("File too small to process"); exit(0);
          }
          if(nrows > max_rows)
          {
                    nrows=max_rows;
          }
          npixels=nrows*ncols;
          //printf("nrows=%d npixels=%d\n",nrows,npixels);


          /* Read in raw data and put into 3D matrix */

          for(count=0;count<nrows;++count)
          {
                    for(k2=0;k2<ncols;++k2)
                    {
                              for(k1=0;k1<nbands;++k1)
                              {
                                        c=fread(&im_data[k1][count][k2],2,1,ifp1);
                              }
                    }
          }



fprintf(ofp1,"Bandas\tEntropia\tConteo\n");
GenContextTable(); /*Generacion tabla de contexto y tabla de frecuencia*/
          /* Calculate the entropy of the error using one band for prediction */

for(k1=1;k1<nbands;++k1)
          {

          printf("Bands %d\r",k1);
                    //k1 = 200;
                    H=0.0;
                    for(k3=0;k3<2*max_value;++k3)
                    {
                              prob[k3]=0;
                    }
                    j = 0;
                    for(k3=1;k3<nrows;++k3)
                    {
                              for(k4=2;k4<ncols-1;++k4)
                              {
                                        A = im_data[k1][k3-1][k4-1];
                                        B = im_data[k1][k3-1][k4];
                                        C = im_data[k1][k3][k4-1];
                                        D =       im_data[k1][k3-1][k4 + 1];
                                        E =       im_data[k1][k3][k4-2];
                                        g1 = D - B;
                                        g2 = B - A;
                                        g3 = A - C;
                                        g4 = C - E;

                                        C0=im_data[k1-1][k3][k4-1];
                                        D0 = im_data[k1-1][k3][k4];


                                        X = D0 - C0 + C;

                                        err = im_data[k1][k3][k4]-X;
                                        //printf("error = %d\n", err);
                                        //getchar();
                                        if(err < (-1*(pow(2,BITS)/2)) || err > ((pow(2,BITS)/2)-1))
                                        {
                                                counter1++;
                                        }
//////////////////////G1////////////////////////////
                                        if(g1 == 0)
                                        {
                                                TempContext = R1 * 9;
                                        }
                                        else if(g1 == 1 || g1 == 2)
                                        {
                                                TempContext = R2 * 9;
                                        }
                                        else if(g1 == -1 || g1 == -2)
                                        {
                                                TempContext = R3 * 9;
                                        }
                                        else if(g1 >= 3 && g1 <= 6)
                                        {
                                                TempContext = R4 * 9;
                                        }
                                        else if(g1 <= -3 && g1 >= -6)
                                        {
                                                TempContext = R5 * 9;
                                        }
                                        else if(g1 >= 7 && g1 <= 14)
                                        {
                                                TempContext = R6 * 9;
                                        }
                                        else if(g1 <= -7 && g1 >= -14)
                                        {
                                                TempContext = R7 * 9;
                                        }
                                        else if(g1 >= 15)
                                        {
                                                TempContext = R8 * 9;
                                        }
                                        else if(g1 <= -15)
                                        {
                                                TempContext = R9 * 9;
                                        }
//////////////////////G2////////////////////////////
```

```
                                        if(g2 == 0)
                                        {
                                                TempContext += R1 * 81;
                                        }
                                        else if(g2 == 1 || g2 == 2)
                                        {
                                                TempContext += R2 * 81;
                                        }
                                        else if(g2 == -1 || g2 == -2)
                                        {
                                                TempContext += R3 * 81;
                                        }
                                        else if(g2 >= 3 && g2 <= 6)
                                        {
                                                TempContext += R4 * 81;
                                        }
                                        else if(g2 <= -3 && g2 >= -6)
                                        {
                                                TempContext += R5 * 81;
                                        }
                                        else if(g2 >= 7 && g2 <= 14)
                                        {
                                                TempContext += R6 * 81;
                                        }
                                        else if(g2 <= -7 && g2 >= -14)
                                        {
                                                TempContext += R7 * 81;
                                        }
                                        else if(g2 >= 15)
                                        {
                                                TempContext += R8 * 81;
                                        }
                                        else if(g2 <= -15)
                                        {
                                                TempContext += R9 * 81;
                                        }
////////////////////G3///////////////////////////
                                        if(g3 == 0)
                                        {
                                                TempContext += R1 * 729;
                                        }
                                        else if(g3 == 1 || g3 == 2)
                                        {
                                                TempContext += R2 * 729;
                                        }
                                        else if(g3 == -1 || g3 == -2)
                                        {
                                                TempContext += R3 * 729;
                                        }
                                        else if(g3 >= 3 && g3 <= 6)
                                        {
                                                TempContext += R4 * 729;
                                        }
```

```
                                else if(g3 <= -3 && g3 >= -6)
                                {
                                        TempContext += R5 * 729;
                                }
                                else if(g3 >= 7 && g3 <= 14)
                                {
                                        TempContext += R6 * 729;
                                }
                                else if(g3 <= -7 && g3 >= -14)
                                {
                                        TempContext += R7 * 729;
                                }
                                else if(g3 >= 15)
                                {
                                        TempContext += R8 * 729;
                                }
                                else if(g2 <= -15)
                                {
                                        TempContext += R9 * 729;
                                }
////////////////////g4////////////////////////////
                                if(abs(g4) < 5)
                                {
                                        TempContext +=  T1 * 3;
                                }
                                else if(g4 >= 5)
                                {
                                        TempContext +=  T2 * 3;
                                }
                                else if(g4 <= -5)
                                {
                                        TempContext +=  T3 * 3;
                                }
                                for(counter = 0; counter < TotalContext;counter++)
                                {
                                        if(ContextTable[counter][0] == TempContext)
                                        {
                                                ++ContextTable[counter][1];
                                                ++ContextTable[counter][2];//Represents N
                                                ContextTable[counter][3] += abs(err); //Represents
A
                                                if(ContextTable[counter][2] >= nThreshold)
                                                {
                                                        ContextTable[counter][2] = 0;
                                                        ContextTable[counter][3] = 0;
                                                }
                                                else
                                                {
                                                        for(k = 0;(ContextTable[counter][2]<<k) <
ContextTable[counter][3];k++);
                                                        ContextTable[counter][4] +=
Coding(err,k);
                                                }
                                        break;
                                    }
                                }
                                //err+=256;
                                err+=9620;
                                //fprintf(ofp1,"%d\n",err);
                                ++prob[err];
                                }
                        }
                        //printf("counter[%d]= %d\n", k1,    counter);
                        //getchar();
                        H=0.0;
                        for(k3=0;k3<2*max_value;++k3)
                        {
                                if(prob[k3]!=0)
                                {
                                        prob1=(double)prob[k3]/(double)npixels;
                                        H+=-prob1*log(prob1);
                                }
                        }
                        H/=log(2.0);
                        for(index = 0; index < TotalContext;index++)
                        {
                                TotalCounter += ContextTable[index][4];
                                ContextTable[index][4] = 0;
                        }
                        fprintf(ofp1,"%d\t%f\t%f\n",k1,H,TotalCounter/(ncols*max_rows));
                        //printf("%d\t%f\t%f\n",k1,H,TotalCounter/(ncols*max_rows));
                        TotalCounter = 0;
                        //getch();

                }
        fclose(ifp1);
        fclose(ofp1);
        printf("DONE\n");
}
/*************************/
//      printf("Conteo Total =  %f  Bits\n",TotalCounter);
//      printf("Conteo Total =  %f  Bytes\n",TotalCounter/8);
//      return 0;

/*************************************************************/
float Coding(int err, int K)
{
        double conteoTotal = 0;     //Guarda el conteo total de la imagen
        int TempValue;
        int Bin[BITS];
        int PosValue;
        int remainder;
        int number;
        int index = BITS - 1;  //variable para los conteos y moverse por los arreglos
        int counterPerCell = 0;
        double TotalCounter = 0;
```

```
	int i,x,value,counter;

	//int K = log10(BITS)/log10(2); //Buscando la K
	x = pow(2,(BITS-(K+1)));	//Valor que se define para cambiar de Binario a decimal
	value = 0;
	index = BITS - 1;	//Volviendo el index a estado original para proxima iteracion
	remainder = 0;		//Remainder = 0 para proxima iteracion de esta forma sabemos que
remainder no tiene un numero no deseado

	for( counter = 0; counter < BITS; counter++) // Se llena el arreglo Bin de 0 para evitar numeros
no deseados
	{
		Bin[counter] = 0;
	}
	if(err < 0)
	{
		TempValue = err * -1; // Convirtiendolo a positivo
		PosValue = (2 * TempValue) - 1; //Aplicando regla de 2n-1
		number = PosValue;//Asignando valor de PosValue a number para conversion a binario
	}
	else if(err >= 0)
	{
		number = 2*err;
	}
	do
	{
		remainder = number%2; //Esta parte aplica la conversion de Dec a Binario
		Bin[index] = remainder;//de la forma de division
		number = number>>1;	//Dando un shift a la derecha para reducir el numero por una
base de 2
		index--;		//El index se disminuye con la idea de llenar el arreglo de Bin de size a
0
	}while(number > 1);	//Seguir aplicando conversion mientras el numero sea mayor que 1
	if(number <= 1)//Si el numero es 1 o 0
	{
		Bin[index] = number; //Quiere decir que terminamos la conversion y el valor del
numero se convierte en el
	}
	// MSB del arreglo
	for(i=0;i<=(BITS-(K+1));i++) //Conversion de Binario a decimal de la parte de arreglo
eliminando los espacios de K
	{
		if(Bin[i] == 1)						//Continua la conversion
			value+=x;		//Value es el que guarda el valor decimal
		x/=2;	//Divide X entre dos para bajar a la proxima potencia de dos
	}
	counterPerCell = value + K;
	return counterPerCell;//Guarda el valor de cada celda de la matriz ya sumando K
}//Hasta aqui es la misma funcionalidad que el if solo que para valores positivos


void GenContextTable()
{
	int
index,tempContext,Tcounter,G1Counter,G2Counter,G3Counter,FirstContext,Taccumulator,G1accumulat
or;
	int G2accumulator,G3accuulator;
	G1Counter = G2Counter = G3Counter = -1;
	Tcounter = Taccumulator = G1accumulator = G2accumulator = G3accuulator  = 0;


	FirstContext = R1*9 + R1*81 + R1*729 + T1*3;
	for(index = 0; index < TotalContext;index++)
	{
		Tcounter = index%3;
		if(Tcounter == 0)
		{
			G3Counter++;
			Tcounter = 0;
			if(G3Counter%9 == 0)
			{
				G2Counter++;
				G3Counter = 0;
				if(G2Counter%9 == 0)
				{
					G1Counter++;
					G2Counter = 0;
				}
			}
		}
		tempContext= FirstContext + 3*Tcounter + 9*G1Counter + 81*G2Counter +
729*G3Counter;
		ContextTable[index][0] = tempContext;
		ContextTable[index][1] = 0;
		ContextTable[index][2] = 0;
		ContextTable[index][3] = 0;
	}
}
```

A.3. LOCO-2B predictor

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<time.h>

#define total_col   31900 /*The number of columns in AVHRR raw data*/
#define ncols       145   /*The number of columns in each spectral band*/
#define nbands      220   /*The number of spectral bands*/
#define max_rows    145   /*The maximum number of rows to be
processed*/
//#define max_value   256 /*Image values are integers from 0 to max_value-1*/
#define max_value   9620 /*Image values are integers from 0 to max_value-1*/
```

```
#define pred_ord    3     /*The order of the linear predictor*/
#define data_bits   14    /*Number of bits in image values*/
#define BITS 16   //Define el tamano de la data 8 o 16 bits
#define R 4
#define T 1
#define R1 1
#define R2 2
#define R3 3
#define R4 4
#define R5 5
#define R6 6
#define R7 7
#define R8 8
#define R9 9
#define T1 10
#define T2 11
#define T3 12
#define  TotalContext ((2*T + 1)*(2*R + 1)*(2*R + 1)*(2*R + 1))


/*
This program reads in raw AVIRIS data and calculates 1st order
entropies of error using 2 bands for prediction.
*/


long           min,file_length;
unsigned long  count;

int    prob[2*max_value];
short  im_data[nbands][max_rows][ncols],error[21025];
int    c,npixels,nrows,err,max_err,min_err,max,min,X,j,D,E,g1,g2,g3,g4;
int    A0,A1,B0,B1,C0,C1,D0,max1,min1,max0,min0,X1,X0;
int counter1 = 0, counter;
double H,H_cond,prob1;
char   outfile[40];
int ContextTable[TotalContext][5];
FILE   *ifp1,*ofp1,*ifp2;
unsigned long compare = 0;   // used to count the # of comparisons made
unsigned long swap    = 0;   // used to count the # of swaps made
void   rotin(),reinitw(),block_process(),weightch();
float Coding(int,int);
void GenContextTable();

void main()
{
        int  k1,k2,k3,k4;
        float TotalCounter = 0;
        //int FreqTable[TotalContext];
        int TempContext;
        int nThreshold = 256;
        int k;
        int index;
```

```
printf("Enter name for input file: ");
gets(outfile);
if((ifp1=fopen(outfile,"rb"))==NULL)
{
        printf("fopen1 failed\n");
        exit(0);
}


printf("Enter name for output file1: ");
gets(outfile);
if((ofp1=fopen(outfile,"w"))==NULL)
{
        printf("fopen2 failed\n");
        exit(0);
}


/* Determine the number of rows in the AVIRIS data file*/

c=1;
nrows=0;
fseek(ifp1,0,2);
file_length=ftell(ifp1);
file_length/=2;
fseek(ifp1,0,0);
nrows=file_length/total_col;

printf("nrows=%d\n",nrows);

if(nrows == 0)
{
        printf("File too small to process"); exit(0);
}
if(nrows > max_rows)
{
        nrows=max_rows;
}
npixels=nrows*ncols;
//printf("nrows=%d npixels=%d\n",nrows,npixels);


/* Read in raw data and put into 3D matrix */

for(count=0;count<nrows;++count)
{
        for(k2=0;k2<ncols;++k2)
        {
                for(k1=0;k1<nbands;++k1)
                {
                        c=fread(&im_data[k1][count][k2],2,1,ifp1);
                }
                }
        }



fprintf(ofp1,"Bandas\tEntropia\tConteo\n");
GenContextTable(); /*Generacion tabla de contexto y tabla de frecuencia*/
        /* Calculate the entropy of the error using one band for prediction */

for(k1=1;k1<nbands;++k1)
        {

        printf("Bands %d\r",k1);
                //k1 = 200;
                H=0.0;
                for(k3=0;k3<2*max_value;++k3)
                {
                        prob[k3]=0;
                }
                j = 0;
                for(k3=1;k3<nrows;++k3)
                {
                        for(k4=2;k4<ncols-1;++k4)
                        {
                                A1 = im_data[k1][k3-1][k4-1];
                                B1 = im_data[k1][k3-1][k4];
                                C1 = im_data[k1][k3][k4-1];
                                D =     im_data[k1][k3-1][k4 + 1];
                                E =     im_data[k1][k3][k4-2];
                                g1 = D - B1;
                                g2 = B1 - A1;
                                g3 = A1 - C1;
                                g4 = C1 - E;


                                A0=im_data[k1-1][k3-1][k4-1];
                                B0=im_data[k1-1][k3-1][k4];
                                C0=im_data[k1-1][k3][k4-1];
                                D0=im_data[k1-1][k3][k4];
                                /**********Maximo**********/
                                if(B1>C1)
                                {
                                        max1 = B1;
                                        min1 = C1;
                                }
                                else
                                {
                                        max1 = C1;
                                        min1 = B1;
                                }
                                if(B0>C0)
                                {
                                        max0 = B0;
                                        min0 = C0;
                                }
                                else
                                {
                                        max0 = C0;
                                        min0 = B0;
                                }

                                /**********LOCO Algorithm**********/
                                if(A1>=max1)
                                {
                                        X1 = min1;
                                        X0 = min0;
                                }
                                else if(A1<=min1)
                                {
                                        X1 = max1;
                                        X0 = max0;
                                }
                                else
                                {
                                        X1 = B1+C1-A1;
                                        X0 = B0+C0-A0;
                                }


                                X = (int)(X1 + D0 - X0);
                                err = im_data[k1][k3][k4]-X;
                                //printf("error = %d\n", err);
                                //getchar();
                                if(err < (-1*(pow(2,BITS)/2)) || err > ((pow(2,BITS)/2)-1))
                                {
                                        counter1++;
                                }
//////////////////////G1/////////////////////////////
                                if(g1 == 0)
                                {
                                        TempContext = R1 * 9;
                                }
                                else if(g1 == 1 || g1 == 2)
                                {
                                        TempContext = R2 * 9;
                                }
                                else if(g1 == -1 || g1 == -2)
                                {
                                        TempContext = R3 * 9;
                                }
                                else if(g1 >= 3 && g1 <= 6)
                                {
                                        TempContext = R4 * 9;
                                }
                                else if(g1 <= -3 && g1 >= -6)
                                {
```

```
                TempContext = R5 * 9;
        }
        else if(g1 >= 7 && g1 <= 14)
        {
                TempContext = R6 * 9;
        }
        else if(g1 <= -7 && g1 >= -14)
        {
                TempContext = R7 * 9;
        }
        else if(g1 >= 15)
        {
                TempContext = R8 * 9;
        }
        else if(g1 <= -15)
        {
                TempContext = R9 * 9;
        }
////////////////////G2//////////////////////////
        if(g2 == 0)
        {
                TempContext += R1 * 81;
        }
        else if(g2 == 1 || g2 == 2)
        {
                TempContext += R2 * 81;
        }
        else if(g2 == -1 || g2 == -2)
        {
                TempContext += R3 * 81;
        }
        else if(g2 >= 3 && g2 <= 6)
        {
                TempContext += R4 * 81;
        }
        else if(g2 <= -3 && g2 >= -6)
        {
                TempContext += R5 * 81;
        }
        else if(g2 >= 7 && g2 <= 14)
        {
                TempContext += R6 * 81;
        }
        else if(g2 <= -7 && g2 >= -14)
        {
                TempContext += R7 * 81;
        }
        else if(g2 >= 15)
        {
                TempContext += R8 * 81;
        }
        else if(g2 <= -15)
        {
                TempContext += R9 * 81;
        }
////////////////////G3//////////////////////////
        if(g3 == 0)
        {
                TempContext += R1 * 729;
        }
        else if(g3 == 1 || g3 == 2)
        {
                TempContext += R2 * 729;
        }
        else if(g3 == -1 || g3 == -2)
        {
                TempContext += R3 * 729;
        }
        else if(g3 >= 3 && g3 <= 6)
        {
                TempContext += R4 * 729;
        }
        else if(g3 <= -3 && g3 >= -6)
        {
                TempContext += R5 * 729;
        }
        else if(g3 >= 7 && g3 <= 14)
        {
                TempContext += R6 * 729;
        }
        else if(g3 <= -7 && g3 >= -14)
        {
                TempContext += R7 * 729;
        }
        else if(g3 >= 15)
        {
                TempContext += R8 * 729;
        }
        else if(g2 <= -15)
        {
                TempContext += R9 * 729;
        }
////////////////////g4///////////////////////////
        if(abs(g4) < 5)
        {
                TempContext +=  T1 * 3;
        }
        else if(g4 >= 5)
        {
                TempContext +=  T2 * 3;
        }
        else if(g4 <= -5)
        {
                TempContext +=  T3 * 3;
        }
        for(counter = 0; counter < TotalContext;counter++)
```

```
                        {
                                if(ContextTable[counter][0] == TempContext)
                                {
                                        ++ContextTable[counter][1];
                                        ++ContextTable[counter][2];//Represents N
                                        ContextTable[counter][3] += abs(err); //Represents
A
                                        if(ContextTable[counter][2] >= nThreshold)
                                        {
                                                ContextTable[counter][2] = 0;
                                                ContextTable[counter][3] = 0;
                                        }
                                        else
                                        {
                                                for(k = 0;(ContextTable[counter][2]<<k) <
ContextTable[counter][3];k++);
                                                ContextTable[counter][4] +=
Coding(err,k);
                                        }
                                        break;
                                }
                        }
                        //err+=256;
                        err+=9620;
                        //fprintf(ofp1,"%d\n",err);
                        ++prob[err];
                        }
                }
                //printf("counter[%d]= %d\n", k1,   counter);
                //getchar();
                H=0.0;
                for(k3=0;k3<2*max_value;++k3)
                {
                        if(prob[k3]!=0)
                        {
                                prob1=(double)prob[k3]/(double)npixels;
                                H+=-prob1*log(prob1);
                        }
                }
                H/=log(2.0);
                for(index = 0; index < TotalContext;index++)
                {
                        TotalCounter +=  ContextTable[index][4];
                        ContextTable[index][4] = 0;
                }
                fprintf(ofp1,"%d\t%f\t%f\n",k1,H,TotalCounter/(145*145));
                TotalCounter = 0;
                //getch();

        }
    fclose(ifp1);
    fclose(ofp1);
    printf("DONE\n");
```

```
}
/************************/
//        printf("Conteo Total =  %f  Bits\n",TotalCounter);
//        printf("Conteo Total =  %f  Bytes\n",TotalCounter/8);
//        return 0;

/************************************************************/
float Coding(int err, int K)
{
        double conteoTotal = 0;     //Guarda el conteo total de la imagen
        int TempValue;
        int Bin[BITS];
        int PosValue;
        int remainder;
        int number;
        int index = BITS - 1;  //variable para los conteos y moverse por los arreglos
        int counterPerCell = 0;
        double TotalCounter = 0;
        int i,x,value,counter;

        //int K = log10(BITS)/log10(2); //Buscando la K
        x = pow(2,(BITS-(K+1)));      //Valor que se define para cambiar de Binario a decimal
        value = 0;
        index = BITS - 1;     //Volviendo el index a estado original para proxima iteracion
        remainder = 0;        //Remainder = 0 para proxima iteracion de esta forma sabemos que
remainder no tiene un numero no deseado

        for( counter = 0; counter < BITS; counter++) // Se llena el arreglo Bin de 0 para evitar numeros
no deseados
        {
                Bin[counter] = 0;
        }
        if(err < 0)
        {
                TempValue = err * -1; // Convirtiendolo a positivo
                PosValue = (2 * TempValue) - 1; //Aplicando regla de 2n-1
                number = PosValue;//Asignando valor de PosValue a number para conversion a binario
        }
        else if(err >= 0)
        {
                number = 2*err;
        }
        do
        {
                remainder = number%2; //Esta parte aplica la conversion de Dec a Binario
                Bin[index] = remainder;//de la forma de division
                number = number>>1;    //Dando un shift a la derecha para reducir el numero por una
base de 2
                index--;               //El index se disminuye con la idea de llenar el arreglo de Bin de size a
0
        }while(number > 1);      //Seguir aplicando conversion mientras el numero sea mayor que 1
        if(number <= 1)//Si el numero es 1 o 0
        {
```

```
		Bin[index] = number; //Quiere decir que terminamos la conversion y el valor del 
numero se convierte en el
	}
	// MSB del arreglo
	for(i=0;i<=(BITS-(K+1));i++) //Conversion de Binario a decimal de la parte de arreglo 
eliminando los espacios de K
	{
		if(Bin[i] == 1)					//Continua la conversion
			value+=x;		//Value es el que guarda el valor decimal
		x/=2;	//Divide X entre dos para bajar a la proxima potencia de dos
	}
	counterPerCell = value + K;
	return counterPerCell;//Guarda el valor de cada celda de la matriz ya sumando K
}//Hasta aqui es la misma funcionalidad que el if solo que para valores positivos


void GenContextTable()
{
	int 
index,tempContext,Tcounter,G1Counter,G2Counter,G3Counter,FirstContext,Taccumulator,G1accumulat
or;
	int G2accumulator,G3accuulator;
	G1Counter = G2Counter = G3Counter = -1;
	Tcounter = Taccumulator = G1accumulator = G2accumulator = G3accuulator  = 0;


	FirstContext = R1*9 + R1*81 + R1*729 + T1*3;
	for(index = 0; index < TotalContext;index++)
	{
		Tcounter = index%3;
		if(Tcounter == 0)
		{
			G3Counter++;
			Tcounter = 0;
			if(G3Counter%9 == 0)
			{
				G2Counter++;
				G3Counter = 0;
				if(G2Counter%9 == 0)
				{
					G1Counter++;
					G2Counter = 0;
				}
			}
		}
		tempContext= FirstContext + 3*Tcounter + 9*G1Counter + 81*G2Counter + 
729*G3Counter;
		ContextTable[index][0] = tempContext;
		ContextTable[index][1] = 0;
		ContextTable[index][2] = 0;
		ContextTable[index][3] = 0;
	}
}
```

A.4. LOCO-3D predictor

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<time.h>

#define total_col   31900 /*The number of columns in AVHRR raw data*/
#define ncols       145   /*The number of columns in each spectral band*/
#define nbands      220   /*The number of spectral bands*/
#define max_rows    145   /*The maximum number of rows to be
processed*/
//#define max_value   256 /*Image values are integers from 0 to max_value-1*/
#define max_value   9620 /*Image values are integers from 0 to max_value-1*/
#define pred_ord    3     /*The order of the linear predictor*/
#define data_bits   14    /*Number of bits in image values*/
#define BITS 16   //Define el tamano de la data 8 o 16 bits
#define R 4
#define T 1
#define R1 1
#define R2 2
#define R3 3
#define R4 4
#define R5 5
#define R6 6
#define R7 7
#define R8 8
#define R9 9
#define T1 10
#define T2 11
#define T3 12
#define  TotalContext ((2*T + 1)*(2*R + 1)*(2*R + 1)*(2*R + 1))


/*
This program reads in raw AVIRIS data and calculates 1st order
entropies of error using 2 bands for prediction.
*/


long           min,file_length;
unsigned long  count;

int    prob[2*max_value];
```

```
short  im_data[nbands][max_rows][ncols],error[21025];
int    c,npixels,nrows,err,max_err,min_err,A,B,C,B0,C0,D0,max,min,X,j,D,E,g1,g2,g3,g4;
int counter1 = 0, counter;
double H,H_cond,prob1;
char   outfile[40];
float  W[pred_ord+1][pred_ord+1],Wp[pred_ord+1][pred_ord+1];
int ContextTable[TotalContext][5];
float  w[pred_ord];
FILE   *ifp1,*ofp1,*ifp2;
unsigned long compare = 0;   // used to count the # of comparisons made
unsigned long swap    = 0;   // used to count the # of swaps made
void   rotin(),reinitw(),block_process(),weightch();
float Coding(int,int);
void GenContextTable();

void main()
{
        int  k1,k2,k3,k4;
        float TotalCounter = 0;
        //int FreqTable[TotalContext];
        int TempContext;
        int nThreshold = 256;
        int k;
        int index;

        printf("Enter name for input file: ");
        gets(outfile);
        if((ifp1=fopen(outfile,"rb"))==NULL)
        {
                printf("fopen1 failed\n");
                exit(0);
        }


        printf("Enter name for output file1: ");
        gets(outfile);
        if((ofp1=fopen(outfile,"w"))==NULL)
        {
                printf("fopen2 failed\n");
                exit(0);
        }


        /* Determine the number of rows in the AVIRIS data file*/

        c=1;
        nrows=0;
        fseek(ifp1,0,2);
        file_length=ftell(ifp1);
        file_length/=2;
        fseek(ifp1,0,0);
        nrows=file_length/total_col;
```

```
          printf("nrows=%d\n",nrows);

          if(nrows == 0)
          {
                    printf("File too small to process"); exit(0);
          }
          if(nrows > max_rows)
          {
                    nrows=max_rows;
          }
          npixels=nrows*ncols;
          //printf("nrows=%d npixels=%d\n",nrows,npixels);


          /* Read in raw data and put into 3D matrix */

          for(count=0;count<nrows;++count)
          {
                    for(k2=0;k2<ncols;++k2)
                    {
                              for(k1=0;k1<nbands;++k1)
                              {
                                        c=fread(&im_data[k1][count][k2],2,1,ifp1);
                              }
                    }
          }



fprintf(ofp1,"Bandas\tEntropia\tConteo\n");
GenContextTable(); /*Generacion tabla de contexto y tabla de frecuencia*/
          /* Calculate the entropy of the error using one band for prediction */

for(k1=1;k1<nbands;++k1)
          {

          printf("Bands %d\r",k1);
                    //k1 = 200;
                    H=0.0;
                    for(k3=0;k3<2*max_value;++k3)
                    {
                              prob[k3]=0;
                    }
                    j = 0;
                    for(k3=1;k3<nrows;++k3)
                    {
                              for(k4=2;k4<ncols-1;++k4)
                              {
                                        A = im_data[k1][k3-1][k4-1];
                                        B = im_data[k1][k3-1][k4];
                                        C = im_data[k1][k3][k4-1];
                                        D =       im_data[k1][k3-1][k4 + 1];
                                        E =       im_data[k1][k3][k4-2];
```

```
                g1 = D - B;
                g2 = B - A;
                g3 = A - C;
                g4 = C - E;


        B0=im_data[k1-1][k3-1][k4];

        C0=im_data[k1-1][k3][k4-1];
        D0 = im_data[k1-1][k3][k4];

        /**********Maximo**********/
        if(B>C)
        {
                max = B;
                min = C;
        }
        else
        {
                max = C;
                min = B;
        }

        /**********LOCO 3D algorithm*************/
        if(A>=max)
        {
                X = min;
        }
        else if(A<=min)
        {
                X = max;
        }
        else
        {
                X = C - C0 + D0;

        }
        err = im_data[k1][k3][k4]-X;
                //printf("error = %d\n", err);
                //getchar();
                if(err < (-1*(pow(2,BITS)/2)) || err > ((pow(2,BITS)/2)-1))
                {
                        counter1++;
                }
////////////////////G1//////////////////////////
                if(g1 == 0)
                {
                        TempContext = R1 * 9;
                }
                else if(g1 == 1 || g1 == 2)
                {
                        TempContext = R2 * 9;
                }
```

```
                                        else if(g1 == -1 || g1 == -2)
                                        {
                                                TempContext = R3 * 9;
                                        }
                                        else if(g1 >= 3 && g1 <= 6)
                                        {
                                                TempContext = R4 * 9;
                                        }
                                        else if(g1 <= -3 && g1 >= -6)
                                        {
                                                TempContext = R5 * 9;
                                        }
                                        else if(g1 >= 7 && g1 <= 14)
                                        {
                                                TempContext = R6 * 9;
                                        }
                                        else if(g1 <= -7 && g1 >= -14)
                                        {
                                                TempContext = R7 * 9;
                                        }
                                        else if(g1 >= 15)
                                        {
                                                TempContext = R8 * 9;
                                        }
                                        else if(g1 <= -15)
                                        {
                                                TempContext = R9 * 9;
                                        }
////////////////////G2///////////////////////////
                                        if(g2 == 0)
                                        {
                                                TempContext += R1 * 81;
                                        }
                                        else if(g2 == 1 || g2 == 2)
                                        {
                                                TempContext += R2 * 81;
                                        }
                                        else if(g2 == -1 || g2 == -2)
                                        {
                                                TempContext += R3 * 81;
                                        }
                                        else if(g2 >= 3 && g2 <= 6)
                                        {
                                                TempContext += R4 * 81;
                                        }
                                        else if(g2 <= -3 && g2 >= -6)
                                        {
                                                TempContext += R5 * 81;
                                        }
                                        else if(g2 >= 7 && g2 <= 14)
                                        {
                                                TempContext += R6 * 81;
                                        }
                                        else if(g2 <= -7 && g2 >= -14)
                                        {
                                                TempContext += R7 * 81;
                                        }
                                        else if(g2 >= 15)
                                        {
                                                TempContext += R8 * 81;
                                        }
                                        else if(g2 <= -15)
                                        {
                                                TempContext += R9 * 81;
                                        }
/////////////////////G3///////////////////////////
                                        if(g3 == 0)
                                        {
                                                TempContext += R1 * 729;
                                        }
                                        else if(g3 == 1 || g3 == 2)
                                        {
                                                TempContext += R2 * 729;
                                        }
                                        else if(g3 == -1 || g3 == -2)
                                        {
                                                TempContext += R3 * 729;
                                        }
                                        else if(g3 >= 3 && g3 <= 6)
                                        {
                                                TempContext += R4 * 729;
                                        }
                                        else if(g3 <= -3 && g3 >= -6)
                                        {
                                                TempContext += R5 * 729;
                                        }
                                        else if(g3 >= 7 && g3 <= 14)
                                        {
                                                TempContext += R6 * 729;
                                        }
                                        else if(g3 <= -7 && g3 >= -14)
                                        {
                                                TempContext += R7 * 729;
                                        }
                                        else if(g3 >= 15)
                                        {
                                                TempContext += R8 * 729;
                                        }
                                        else if(g2 <= -15)
                                        {
                                                TempContext += R9 * 729;
                                        }
////////////////////g4////////////////////////////
                                        if(abs(g4) < 5)
                                        {
                                                TempContext +=  T1 * 3;
```

```
                }
                else if(g4 >= 5)
                {
                    TempContext +=  T2 * 3;
                }
                else if(g4 <= -5)
                {
                    TempContext +=  T3 * 3;
                }
                for(counter = 0; counter < TotalContext;counter++)
                {
                    if(ContextTable[counter][0] == TempContext)
                    {
                        ++ContextTable[counter][1];
                        ++ContextTable[counter][2];//Represents N
                        ContextTable[counter][3] += abs(err); //Represents
A
                        if(ContextTable[counter][2] >= nThreshold)
                        {
                            ContextTable[counter][2] = 0;
                            ContextTable[counter][3] = 0;
                        }
                        else
                        {
                            for(k = 0;(ContextTable[counter][2]<<k) <
ContextTable[counter][3];k++);
                            ContextTable[counter][4] +=
Coding(err,k);
                        }
                        break;
                    }
                }
                //err+=256;
                err+=9620;
                //fprintf(ofp1,"%d\n",err);
                ++prob[err];
                }
            }
            //printf("counter[%d]= %d\n", k1,   counter);
            //getchar();
            H=0.0;
            for(k3=0;k3<2*max_value;++k3)
            {
                if(prob[k3]!=0)
                {
                    prob1=(double)prob[k3]/(double)npixels;
                    H+=-prob1*log(prob1);
                }
            }
            H/=log(2.0);
            for(index = 0; index < TotalContext;index++)
            {
                TotalCounter +=  ContextTable[index][4];
```

```
                        ContextTable[index][4] = 0;
                    }
                    fprintf(ofp1,"%d\t%f\t%f\n",k1,H,TotalCounter/(145*145));
                    TotalCounter = 0;
                    //getch();

                }
        fclose(ifp1);
        fclose(ofp1);
        printf("DONE\n");
}
/*************************/
//      printf("Conteo Total =  %f  Bits\n",TotalCounter);
//      printf("Conteo Total =  %f  Bytes\n",TotalCounter/8);
//      return 0;

/************************************************************/
float Coding(int err, int K)
{
        double conteoTotal = 0;     //Guarda el conteo total de la imagen
        int TempValue;
        int Bin[BITS];
        int PosValue;
        int remainder;
        int number;
        int index = BITS - 1;  //variable para los conteos y moverse por los arreglos
        int counterPerCell = 0;
        double TotalCounter = 0;
        int i,x,value,counter;

        //int K = log10(BITS)/log10(2); //Buscando la K
        x = pow(2,(BITS-(K+1)));      //Valor que se define para cambiar de Binario a decimal
        value = 0;
        index = BITS - 1;     //Volviendo el index a estado original para proxima iteracion
        remainder = 0;        //Remainder = 0 para proxima iteracion de esta forma sabemos que
remainder no tiene un numero no deseado

        for( counter = 0; counter < BITS; counter++) // Se llena el arreglo Bin de 0 para evitar numeros
no deseados
        {
                Bin[counter] = 0;
        }
        if(err < 0)
        {
                TempValue = err * -1; // Convirtiendolo a positivo
                PosValue = (2 * TempValue) - 1; //Aplicando regla de 2n-1
                number = PosValue;//Asignando valor de PosVa lue a number para conversion a binario
        }
        else if(err >= 0)
        {
                number = 2*err;
        }
        do
        {
                remainder = number%2; //Esta parte aplica la conversion de Dec a Binario
                Bin[index] = remainder;//de la forma de division
                number = number>>1;    //Dando un shift a la derecha para reducir el numero por una
base de 2
                index--;               //El index se disminuye con la idea de llenar el arreglo de Bin de size a
0
        }while(number > 1);      //Seguir aplicando conversion mientras el numero sea mayor que 1
        if(number <= 1)//Si el numero es 1 o 0
        {
                Bin[index] = number; //Quiere decir que terminamos la conversion y el valor del
numero se convierte en el
        }
        // MSB del arreglo
        for(i=0;i<=(BITS-(K+1));i++) //Conversion de Binario a decimal de la parte de arreglo
eliminando los espacios de K
        {
                if(Bin[i] == 1)                                        //Continua la conversion
                        value+=x;              //Value es el que guarda el valor decimal
                x/=2;      //Divide X entre dos para bajar a la proxima potencia de dos
        }
        counterPerCell = value + K;
        return counterPerCell;//Guarda el valor de cada celda de la matriz ya sumando K
}//Hasta aqui es la misma funcionalidad que el if solo que para valores positivos


void GenContextTable()
{
        int
index,tempContext,Tcounter,G1Counter,G2Counter,G3Counter,FirstContext,Taccumulator,G1accumulat
or;
        int G2accumulator,G3accuulator;
        G1Counter = G2Counter = G3Counter = -1;
        Tcounter = Taccumulator = G1accumulator = G2accumulator = G3accuulator  = 0;


        FirstContext = R1*9 + R1*81 + R1*729 + T1*3;
        for(index = 0; index < TotalContext;index++)
        {
                Tcounter = index%3;
                if(Tcounter == 0)
                {
                        G3Counter++;
                        Tcounter = 0;
                        if(G3Counter%9 == 0)
                        {
                                G2Counter++;
                                G3Counter = 0;
                                if(G2Counter%9 == 0)
                                {
                                        G1Counter++;
                                        G2Counter = 0;
                                }
                        }
                }
                tempContext= FirstContext + 3*Tcounter + 9*G1Counter + 81*G2Counter +
729*G3Counter;
                ContextTable[index][0] = tempContext;
                ContextTable[index][1] = 0;
                ContextTable[index][2] = 0;
                ContextTable[index][3] = 0;
        }
}
```

A.5. LOCO-SI predictor

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<time.h>

#define total_col   31900 /*The number of columns in AVHRR raw data*/
#define ncols       145   /*The number of columns in each spectral band*/
#define nbands      220   /*The number of spectral bands*/
#define max_rows    145   /*The maximum number of rows to be
processed*/
//#define max_value   256 /*Image values are integers from 0 to max_value-1*/
#define max_value   9620 /*Image values are integers from 0 to max_value-1*/
#define pred_ord    3     /*The order of the linear predictor*/
#define data_bits   14    /*Number of bits in image values*/
#define BITS 16   //Define el tamano de la data 8 o 16 bits
#define R 4
#define T 1
#define R1 1
#define R2 2
#define R3 3
#define R4 4
#define R5 5
#define R6 6
#define R7 7
#define R8 8
#define R9 9
#define T1 10
#define T2 11
#define T3 12
#define  TotalContext ((2*T + 1)*(2*R + 1)*(2*R + 1)*(2*R + 1))


/*
This program reads in raw AVIRIS data and calculates 1st order
entropies of error using 2 bands for prediction.
*/
```

```
long          min,file_length;
unsigned long  count;

int    prob[2*max_value];
short  im_data[nbands][max_rows][ncols],error[21025];
int    c,npixels,nrows,err,max_err,min_err,A,B,C,A0,B0,C0,D0,max,min,X,j,D,E,g1,g2,g3,g4;
int counter1 = 0, counter;
double H,H_cond,prob1;
char   outfile[40];
float  W[pred_ord+1][pred_ord+1],Wp[pred_ord+1][pred_ord+1];
int ContextTable[TotalContext][5];
float  w[pred_ord];
FILE   *ifp1,*ofp1,*ifp2;
unsigned long compare = 0;   // used to count the # of comparisons made
unsigned long swap    = 0;   // used to count the # of swaps made
void   rotin(),reinitw(),block_process(),weightch();
float Coding(int,int);
void GenContextTable();

void main()
{
        int  k1,k2,k3,k4;
        float TotalCounter = 0;
        //int FreqTable[TotalContext];
        int TempContext;
        int nThreshold = 256;
        int k;
        int index;

        printf("Enter name for input file: ");
        gets(outfile);
        if((ifp1=fopen(outfile,"rb"))==NULL)
        {
                printf("fopen1 failed\n");
                exit(0);
        }


        printf("Enter name fo r output file1: ");
        gets(outfile);
        if((ofp1=fopen(outfile,"w"))==NULL)
        {
                printf("fopen2 failed\n");
                exit(0);
        }


        /* Determine the number of rows in the AVIRIS data file*/

        c=1;
        nrows=0;
        fseek(ifp1,0,2);
        file_length=ftell(ifp1);
        file_length/=2;
        fseek(ifp1,0,0);
        nrows=file_length/total_col;

        printf("nrows=%d\n",nrows);

        if(nrows == 0)
        {
                printf("File too small to process"); exit(0);
        }
        if(nrows > max_rows)
        {
                nrows=max_rows;
        }
        npixels=nrows*ncols;
        //printf("nrows=%d npixels=%d\n",nrows,npixels);


        /* Read in raw data and put into 3D matrix */

        for(count=0;count<nrows;++count)
        {
                for(k2=0;k2<ncols;++k2)
                {
                        for(k1=0;k1<nbands;++k1)
                        {
                                c=fread(&im_data[k1][count][k2],2,1,ifp1);
                        }
                }
        }



fprintf(ofp1,"Bandas\tEntropia\tConteo\n");
GenContextTable(); /*Generacion tabla de contexto y tabla de frecuencia*/
        /* Calculate the entropy of the error using one band for prediction */

for(k1=1;k1<nbands;++k1)
        {

        printf("Bands %d\r",k1);
                //k1 = 200;
                H=0.0;
                for(k3=0;k3<2* max_value;++k3)
                {
                        prob[k3]=0;
                }
                j = 0;
                for(k3=1;k3<nrows;++k3)
                {
                        for(k4=2;k4<ncols -1;++k4)
                        {
                                A = im_data[k1][k3-1][k4-1];
```

```
		B = im_data[k1][k3-1][k4];
		C = im_data[k1][k3][k4-1];
		D =	im_data[k1][k3-1][k4 + 1];
		E =	im_data[k1][k3][k4-2];
		g1 = D - B;
		g2 = B - A;
		g3 = A - C;
		g4 = C - E;



	A0=im_data[k1-1][k3-1][k4-1];

	B0=im_data[k1-1][k3-1][k4];

	C0=im_data[k1-1][k3][k4-1];
	D0 = im_data[k1-1][k3][k4];

	/**********Maximo**********/
	if(B>C)
	{
		max = B;
		min = C;
	}
	else
	{
		max = C;
		min = B;
	}

	/**********LOCO algorithm*************/
	if ((abs(C0 - C) < 350) && (abs(B0 - B) < 350))
	{
		X = C - C0 + D0;
	}

	else if(A>=max)
	{
		X = min;
	}
	else if(A<=min)
	{
		X = max;
	}
	else
	{
		X = B + C - A;

	}
	err = im_data[k1][k3][k4]-X;
		//printf("error = %d\n", err);
		//getchar();
		if(err < (-1*(pow(2,BITS)/2)) || err > ((pow(2,BITS)/2)-1))
```

```
                {
                        counter1++;
                }
//////////////////////G1////////////////////////////
                if(g1 == 0)
                {
                        TempContext = R1 * 9;
                }
                else if(g1 == 1 || g1 == 2)
                {
                        TempContext = R2 * 9;
                }
                else if(g1 == -1 || g1 == -2)
                {
                        TempContext = R3 * 9;
                }
                else if(g1 >= 3 && g1 <= 6)
                {
                        TempContext = R4 * 9;
                }
                else if(g1 <= -3 && g1 >= -6)
                {
                        TempContext = R5 * 9;
                }
                else if(g1 >= 7 && g1 <= 14)
                {
                        TempContext = R6 * 9;
                }
                else if(g1 <= -7 && g1>= -14)
                {
                        TempContext = R7 * 9;
                }
                else if(g1 >= 15)
                {
                        TempContext = R8 * 9;
                }
                else if(g1 <= -15)
                {
                        TempContext = R9 * 9;
                }
//////////////////////G2////////////////////////////
                if(g2 == 0)
                {
                        TempContext += R1 * 81;
                }
                else if(g2 == 1 || g2 == 2)
                {
                        TempContext += R2 * 81;
                }
                else if(g2 == -1 || g2 == -2)
                {
                        TempContext += R3 * 81;
                }
```

```
                    else if(g2 >= 3 && g2 <= 6)
                    {
                            TempContext += R4 * 81;
                    }
                    else if(g2 <= -3 && g2 >= -6)
                    {
                            TempContext += R5 * 81;
                    }
                    else if(g2 >= 7 && g2 <= 14)
                    {
                            TempContext += R6 * 81;
                    }
                    else if(g2 <= -7 && g2 >= -14)
                    {
                            TempContext += R7 * 81;
                    }
                    else if(g2 >= 15)
                    {
                            TempContext += R8 * 81;
                    }
                    else if(g2 <= -15)
                    {
                            TempContext += R9 * 81;
                    }
////////////////////G3///////////////////////////
                    if(g3 == 0)
                    {
                            TempContext += R1 * 729;
                    }
                    else if(g3 == 1 || g3 == 2)
                    {
                            TempContext += R2 * 729;
                    }
                    else if(g3 == -1 || g3 == -2)
                    {
                            TempContext += R3 * 729;
                    }
                    else if(g3 >= 3 && g3 <= 6)
                    {
                            TempContext += R4 * 729;
                    }
                    else if(g3 <= -3 && g3 >= -6)
                    {
                            TempContext += R5 * 729;
                    }
                    else if(g3 >= 7 && g3 <= 14)
                    {
                            TempContext += R6 * 729;
                    }
                    else if(g3 <= -7 && g3 >= -14)
                    {
                            TempContext += R7 * 729;
                    }
```

```
                                        else if(g3 >= 15)
                                        {
                                                TempContext += R8 * 729;
                                        }
                                        else if(g2 <= -15)
                                        {
                                                TempContext += R9 * 729;
                                        }
////////////////////g4/////////////////////////////
                                        if(abs(g4) < 5)
                                        {
                                                TempContext +=  T1 * 3;
                                        }
                                        else if(g4 >= 5)
                                        {
                                                TempContext +=  T2 * 3;
                                        }
                                        else if(g4 <= -5)
                                        {
                                                TempContext +=  T3 * 3;
                                        }
                                        for(counter = 0; counter < TotalContext;counter++)
                                        {
                                                if(ContextTable[counter][0] == TempContext)
                                                {
                                                        ++ContextTable[counter][1];
                                                        ++ContextTable[counter][2];//Represents N
                                                        ContextTable[counter][3] += abs(err); //Represents
A
                                                        if(ContextTable[counter][2] >= nThreshold)
                                                        {
                                                                ContextTable[counter][2] = 0;
                                                                ContextTable[counter][3] = 0;
                                                        }
                                                        else
                                                        {
                                                                for(k = 0;(ContextTable[counter][2]<<k) <
ContextTable[counter][3];k++);
                                                                ContextTable[counter][4] +=
Coding(err,k);
                                                        }
                                                        break;
                                                }
                                        }
                                        //err+=256;
                                        err+=9620;
                                        //fprintf(ofp1,"%d\n",err);
                                        ++prob[err];
                                        }
                                }
                                //printf("counter[%d]= %d\n", k1,    counter);
                                //getchar();
                                H=0.0;
```

```
                        for(k3=0;k3<2*max_value;++k3)
                        {
                                if(prob[k3]!=0)
                                {
                                        prob1=(double)prob[k3]/(double)npixels;
                                        H+=-prob1*log(prob1);
                                }
                        }
                        H/=log(2.0);
                        for(index = 0; index < TotalContext;index++)
                        {
                                TotalCounter +=  ContextTable[index][4];
                                ContextTable[index][4] = 0;
                        }
                        fprintf(ofp1,"%d\t%f\t%f\n",k1,H,TotalCounter/(145*145));
                        TotalCounter = 0;
                        //getch();

                }
        fclose(ifp1);
        fclose(ofp1);
        printf("DONE\n");
}
/*************************/
//      printf("Conteo Total =  %f  Bits\n",TotalCounter);
//      printf("Conteo Total =  %f  Bytes\n",TotalCounter/8);
//      return 0;

/***********************************************************/
float Coding(int err, int K)
{
        double conteoTotal = 0;     //Guarda el conteo total de la imagen
        int TempValue;
        int Bin[BITS];
        int PosValue;
        int remainder;
        int number;
        int index = BITS - 1;  //variable para los conteos y moverse por los arreglos
        int counterPerCell = 0;
        double TotalCounter = 0;
        int i,x,value,counter;

        //int K = log10(BITS)/log10(2); //Buscando la K
        x = pow(2,(BITS-(K+1)));      //Valor que se define para cambiar de Binario a decimal
        value = 0;
        index = BITS - 1;     //Volviendo el index a estado original para proxima iteracion
        remainder = 0;        //Remainder = 0 para proxima iteracion de esta forma sabemos que
remainder no tiene un numero no deseado

        for( counter = 0; counter < BITS; counter++) // Se llena el arreglo Bin de 0 para evitar numeros
no deseados
        {
                Bin[counter] = 0;
```

```
          }
          if(err < 0)
          {
                    TempValue = err * -1; // Convirtiendolo a positivo
                    PosValue = (2 * TempValue) - 1; //Aplicando regla de 2n-1
                    number = PosValue;//Asignando valor de PosValue a number para conversion a binario
          }
          else if(err >= 0)
          {
                    number = 2*err;
          }
          do
          {
                    remainder = number%2; //Esta parte aplica la conversion de Dec a Binario
                    Bin[index] = remainder;//de la forma de division
                    number = number>>1;    //Dando un shift a la derecha para reducir el numero por una
base de 2
                    index--;               //El index se disminuye con la idea de llenar el arreglo de Bin de size a
0
          }while(number > 1);      //Seguir aplicando conversion mientras el numero sea mayor que 1
          if(number <= 1)//Si el numero es 1 o 0
          {
                    Bin[index] = number; //Quiere decir que terminamos la conversion y el valor del
numero se convierte en el
          }
          // MSB del arreglo
          for(i=0;i<=(BITS-(K+1));i++) //Conversion de Binario a decimal de la parte de arreglo
eliminando los espacios de K
          {
                    if(Bin[i] == 1)                                                     //Continua la conversion
                              value+=x;              //Value es el que guarda el valor decimal
                    x/=2;      //Divide X entre dos para bajar a la proxima potencia de dos
          }
          counterPerCell = value + K;
          return counterPerCell;//Guarda el valor de cada celda de la matriz ya sumando K
}//Hasta aqui es la misma funcionalidad que el if solo que para valores positivos


void GenContextTable()
{
          int
index,tempContext,Tcounter,G1Counter,G2Counter,G3Counter,FirstContext,Taccumulator,G1accumulat
or;
          int G2accumulator,G3accuulator;
          G1Counter = G2Counter = G3Counter = -1;
          Tcounter = Taccumulator = G1accumulator = G2accumulator = G3accuulator  = 0;


          FirstContext = R1*9 + R1*81 + R1*729 + T1*3;
          for(index = 0; index < TotalContext;index++)
          {
                    Tcounter = index%3;
                    if(Tcounter == 0)
                    {
                              G3Counter++;
                              Tcounter = 0;
                              if(G3Counter%9 == 0)
                              {
                                        G2Counter++;
                                        G3Counter = 0;
                                        if(G2Counter%9 == 0)
                                        {
                                                  G1Counter++;
                                                  G2Counter = 0;
                                        }
                              }
                    }
                    tempContext= FirstContext + 3*Tcounter + 9*G1Counter + 81*G2Counter +
729*G3Counter;
                    ContextTable[index][0] = tempContext;
                    ContextTable[index][1] = 0;
                    ContextTable[index][2] = 0;
                    ContextTable[index][3] = 0;
          }
}
```