# Algorithms to Optimize Battery Consumption in Precision Livestock Farming.
# Algoritmos de compresión para optimizar el consumo de batería en la ganadería de precisión

## ABSTRACT

The 34% world supply of food proteins comes from livestock [1] and the need to supplement it, makes that the number of animals rearing increases day by day. Nowadays, this process is not effective due to the farmers not having the correct tools and devices to minimize their energy consumption.In line, the objective of this project is to design an algorithm that helps to compress and decompress images to optimize the energy that is required for classifying and obtaining the information of the animals. The algorithms that we implemented to achieve the objective previously mentioned were the lossy image compression with Fast Fourier Transform and lossless image compression with Huffman Coding, they were the ones that gave us the best results in terms of complexity execution time, the least possible loss of information and with a good compression ratio.

El 34% del suministro mundial de proteínas alimentarias proviene de la ganadería [1] y la necesidad de complementarla, hace que el número de crías de animales aumente día a día. En la actualidad, este proceso no es efectivo debido a que los ganaderos no cuentan con las herramientas y dispositivos adecuados para minimizar su consumo de energía.En línea, el objetivo de este proyecto es diseñar un algoritmo que ayude a comprimir y descomprimir imágenes para optimizar la energía que se requiere para clasificar y obtener la información de los animales. Los algoritmos que implementamos para lograr el objetivo antes mencionado fueron la compresión de imágenes con pérdida con la Transformada Rápida de Fourier y la compresión de imágenes sin pérdida con la Codificación Huffman, fueron los que nos dieron los mejores resultados en cuanto a complejidad tiempo de ejecución, la menor pérdida de información posible y con un buen ratio de compresión.

## Author Keywords

Compression algorithms, machine learning, deep learning, precision livestock farming, animal health.

Algoritmos de compresión, machine learning, deep learning, ganadería de precisión, sanidad animal.

## 1 Introduction

Livestock farming has played a very important economic and socio-cultural role throughout history. A few years ago, a multidisciplinary science called the Precision Livestock Farming (PLF) came up with the purpose of getting better livestock activity using information and communication technology.

Currently PLF has some important challenges like data digitization, reducing data dimensions and some other that are important to apply the system correctly in the farms. During the years, some experts like Debauche had concluded that the best way to assume those challenges is with an algorithm that helps to compress the data [2] to determine some characteristics of the animal like its health, behavior, and other.

### 1.1 Problem

We are facing a problem that is designing an algorithm to compress and decompress images to reduce the energy consumption in the context of precision livestock farming. Moreover, the compressed images will be evaluated through a classification algorithm that determines the animal health. The objective is to get the more accurate compression and try to have an error rate less than 5%.

According to D.Berckmans´s article a major problem of the livestock sector is and will be the continuous monitoring of animal health within the big groups of animals[3]. And it is because as decades have passed the number of animals has increased but the number of farmers has decreased. For the above reasons and more is important to develop an efficient solution that makes farmers able to have more information about the health and other conditions of the animals.

### 1.2 Solution

In this work, we used a convolutional neural network to classify animal health, in cattle, in the context of precision livestock farming (PLF). A common problem in PLF is that networking infrastructure is very limited, thus data compression is required.

Basically, in the lossy-data compression we implemented image scaling using two different algorithms. Firstly, we choose nearest neighbor interpolation, this method simply determines the “nearest” neighbouring pixel and assumes the intensity value of it. The main reason we decided to implement it, is because this algorithm has an appropriate complexity to work with a huge amount of data (the time complexity for the worst case is O(nxm)) and also because compared with other algorithms like bilinear interpolation, you don’t lose much information when you compress the image. Figure 1 shows the image obtained after applied Nearest neighbor interpolation.

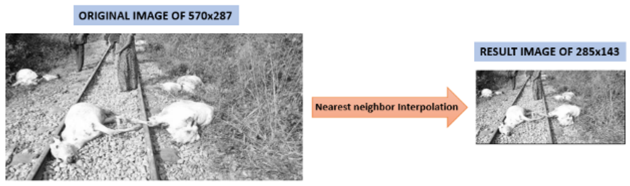


Figure 1 Image Scaling using Nearest neighbor interpolation

Secondly, we used Fast Fourier Transformation (FFT), which is commonly use to modify an image between the spatial and frequency domain. This algorithm transforms our image matrix into a matrix of coefficients which are frequencies, and if we only keep the higher of them it will reduce the size of the matrix. For example, we can only keep 5% of those coefficients and the loss of information would be really small. Also, this algorithm has an appropriate time complexity, it is O(m*n*log(n*m)) for the worst case. Figure 2 shows the process the image go through applying Fast Fourier Transform.

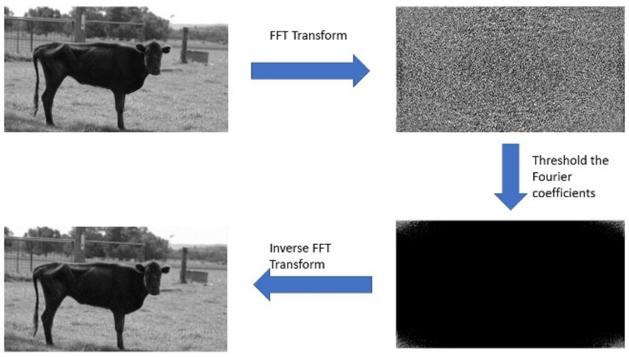


Figure 2 Image Scaling using FFT.

Lastly, we tested other method that is Singular Value Decomposition (SVD), this algorithm creates 3 variables from the image matrix and if you multiply these variables, it will be equal to the original matrix: Knowing this we can choose fewer elements from those 3 variables and create an approximation that can keep a lot of information with a reduction of almost 80% of its size. All this with a time complexity of O (max $(m, n)^2$). Figure 3 shows the process the image go through applying Singular Value Decomposition.

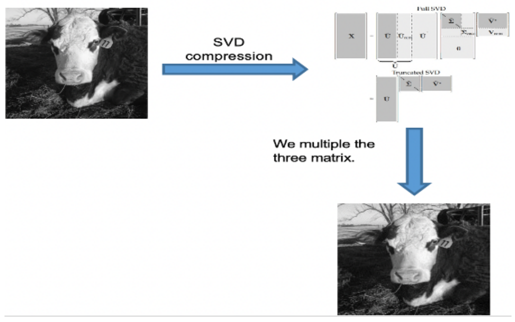


Figure 3 Compression image using SVD .

The algorithm with lossy image-compression that we chose to solve the problem is the Fast Fourier Transform (FFT), because this method has an appropriate time complexity to compute large amounts of data. Also, it has a great compression/lost information correlation of compressing.

On the other hand, in lossless-data compression we implement algorithms like LZ77, LZW and Huffman. The LZ77 algorithm is explained a few sections below hold its data in a structure called slicing window; the larger the sliding window, the longer it takes to encode the information. LZW is an improved implementation of LZ78 algorithm, it compresses a file into a smaller one using a dictionary-based lookup algorithm, it takes each input sequence of bits and creates an entry in the dictionary for that particular bit pattern. And lastly, the Huffman coding, also explained a few sections below.

Finally, the algorithm with lossless-compression that we choose to solve the problem is The Huffman algorithm, because this method has an appropriate complexity and also the implementation accomplished all the requirements the best way possible. This implementation was the most optimal compared to the others, so we decided to implement the Fast Fourier Transform merged with Huffman coding because it would give us the best result in terms of the amount of data compressed and the execution time.

## 2 RELATED WORK

In what follows, we explain four related works on the domain of animal-health classification and image compression in the context of PLF.

### 2.1 An Animal Welfare Platform for Extensive Livestock Production Systems

Currently, agriculture policy reforms in the USA have favored livestock production showing respect to animal welfare. In order, this project was based on giving a solution for tracking and monitoring animal activity and behavior in livestock farms, to get the respective indicators about animal well-being. They designed an automated system with a type of wireless sensor, in this case, a collar device, to monitor the animal‘s well-being according to their movement, speed, and geolocation information, with low implementation cost. Figure 4 illustrated the collar device that were used in that project to monitor the animals.

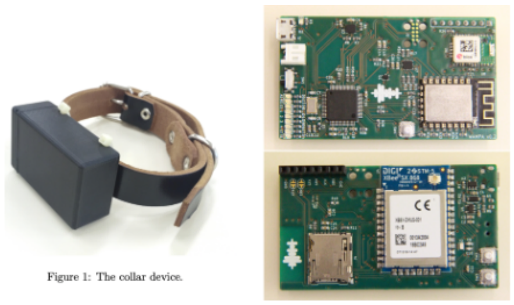


Figure 4 Collar device to monitor the animals

The system was able to perform offline and real-time data processing for pattern recognition through Deep Neural Network pattern recognition algorithms. Finally, cloud computing processed both data and Deep Learning model storage, and the significant data were presented to the farmers in mobile devices.[4]

### 2.2 Visual Localisation and Individual Identification of Holstein Friesian Cattle via Deep Learning

In this work they prove using computer vision pipelines full of architectures of neural networks that the identification of individual Holstein Friesian can occur automatically and non-intrusively. The deep network that they use to address the problem of detecting and locating Holstein through images is the R-CNN adaptation of the VGG CNN M 1024 network published as part of several other network architecture proposals. And to detect them through videos, they use the standard Kernelized Correlation Filter (KCF) tracking algorithm. The results they obtained were really good to prove what they want, however there were some mistakes and identifications that were not well-done, some false positives, some identifications were less than what they expected and so on [5].

### 2.3 Cloud services integration for farm animals´ behavior studies based on smartphones as an activity sensor

The problem that they analyzed was the optimization that the sensors need. First because of the power consumption, second the storage and the processing of large amounts of data and third the matching of data with complementary data. They describe a new infrastructure which brings benefits in storage, real-time processing and abilities for large scale data storage and analytics that allows to collect, store, treat and share information between scientists. In the case of the compression, they performed it in two ways: First by eliminating redundancies and replacement of redundant data by a time interval during which the value remains constant was applied to preserve data integrity. And second by truncating data to 3, 4 and 5 decimal digits [6].

### 2.4 A systematic literature review on the use of machine learning in precision livestock farming

The project aims to reduce livestock environmental impact and identify the most appropriate process in livestock. With the recent concept of Precision Livestock Farming (PLF), which focuses on making decisions based on quantitative data obtained in real-time, they want to improve the farming process and propose technological solutions in agriculture and livestock production systems. To get and study the data, this system uses data analysis, machine learning, control systems, and ICT.[7] Figure 5 indicate the process they used in the investigation to ger data and study the animals behavior.

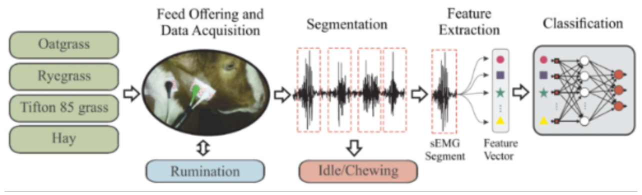


Figure 5 Process to get data and study animals.

## 3 MATERIALS AND METHODS

In this section, we explain how the data was collected and processed and, after different image-compression algorithm alternatives to solve, improve animal-health classification.

### 3.1 Data Collection and Processing

We collected data from Google Images and Bing Images divided into two groups: healthy cattle and sick cattle. For healthy cattle, the search string was “cow”. For sick cattle, the search string was “cow + sick”. In the next step, both groups of images were transformed into grayscale using Python OpenCV and they were transformed into Comma Separated Values (CSV) files. It was found out that the datasets were balanced. The dataset was divided into 70% for training and 30% for testing. Datasets are available at https://github.com/mauriciotoro/ST0245Eafit/tree/master /proyecto/datasets Finally, using the training data set, we trained a convolutional neural network for binary image-classification using Google Teachable Machine available at https://teachablemachine. withgoogle.com/train/image.

### 3.2 Lossy Image-compression alternatives

In what follows, we present different algorithms used to compress images.

#### 3.2.1 Fractal compression

Fractal compression is a lossy compression method for digital images, based on fractals. This method uses an algorithm to convert these parts into fractal codes which are used to recreate the encoded image. [8] Figure 6 shows fractal compression method.

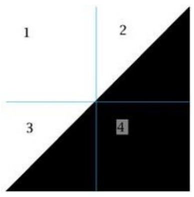


Figure 6 Fractal compression method.

#### 3.2.2 Seam Carving

Seam carving is an algorithm for content-aware image resizing. It establishes several seams in an image and automatically removes seams to reduce image size or inserts seams to extend it. How does it work? First, it receives an image. After that, it calculates the weight/density/energy of each pixel, this is done by an algorithm called gradient magnitude. From the energy, make a list of seams. Seams are ranked by energy, with low energy seams being of least importance to the content of the image. It removes low-energy seams as needed and finally, it returns the image.This process will be illustrated in figure 7[9]

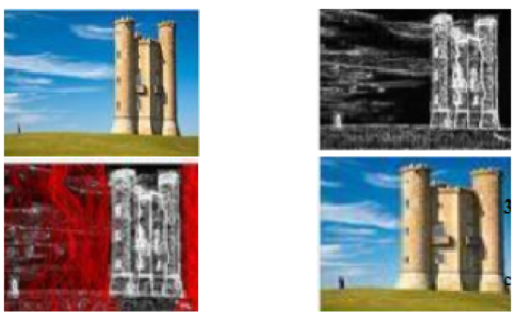


Figure 7 Seam Carving Process

#### 3.2.3 Discrete cosine transform

The Discrete Cosine Transform (DTC) is very important to the process of image/video compression because it has a very strong energy compaction [10]. It uses many arithmetic operations for that cause it is very important to make the implementation of the DTC efficient in the real-time image transformation. Figure 8 indicates the process to compress using Discrete Cosine Transform

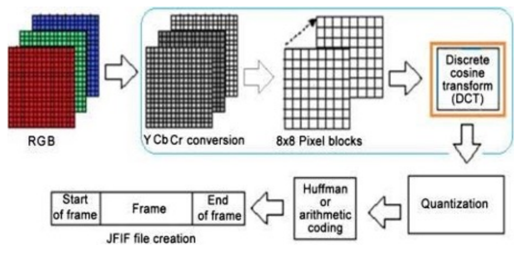


Figure 8 Discrete Cosine Transform

Generally, to use the DTC´s algorithm, you need to get the image file information, then you divide it in a block of 8x8 matrix and the you apply the discrete cosine (a sum of cosine functions oscillating at different frequencies) [11][12]

### 3.2.4 Image Scaling

Image scaling refers to resizing or resampling a digital image. Depending on the images, they can be scaled using geometric transformations or create a new one with a higher or lower number of pixels [13]. There exist three common scaling algorithms: 1. Nearest Neighbor Scaling. 2.Bilinear and 3. Bicubic Interpolation [14]. The differents algorithms of image scaling will be illustrated in figure 9

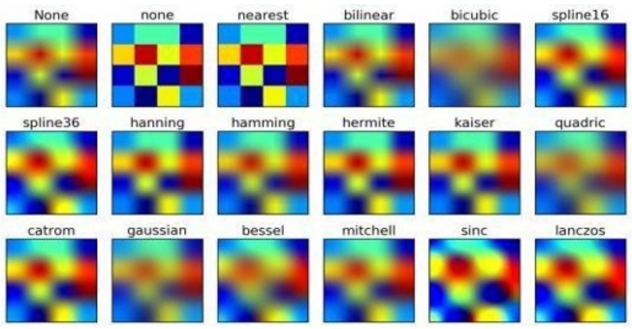


Figure 9 Common scaling algorithm.

## 3.3 Lossless Image-compression alternatives

In what follows, we present different algorithms used to compress images.

### 3.3.1 Huffman coding

The Huffman coding is an algorithm to compress data, basically for compressing files. The idea of the algorithm is to assign variable-length codes to input characters considering which are used and unused to turn out the most optimal code lengths [15]. There are mainly two major parts in Huffman Coding: 1. Build a Huffman Tree from input characters. 2. Traverse a Huffman Tree and assign codes to characters [16]. The Huffman Tree will be shown in figure 10

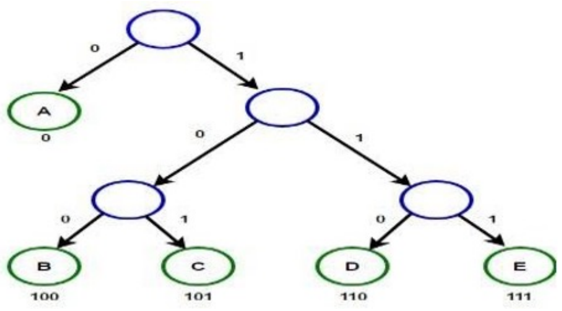


Figure 10 Huffman coding process.

### 3.3.2 LZ77

The LZ77 algorithm is used to analyze the input data and find out how to reduce the size by replacing redundant information with metadata(XML-formatted data that defines the characteristics of an update) . Steps for the LZ77 algorithm: 1. Set the coding position to the beginning of the input stream. 2. Find the longest match in the window for the lookahead buffer. 3. If a match is found, output the pointer P. Move the coding position (and the window) L bytes forward. 4. If a match is not found, output a null pointer and the first byte in the lookahead buffer. Move the coding position (and the window) one byte forward. 5. If the lookahead buffer is not empty, return to step 2 [17].Figure 11 illustrate the process to compress using LZ77 algorithm

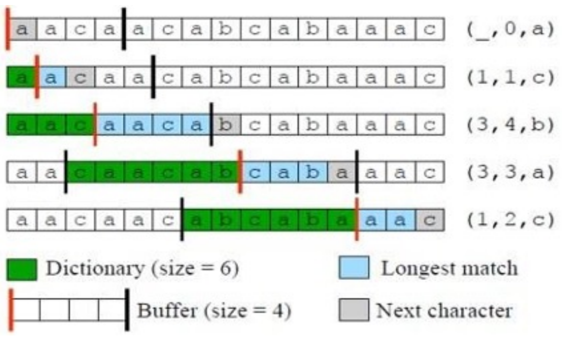
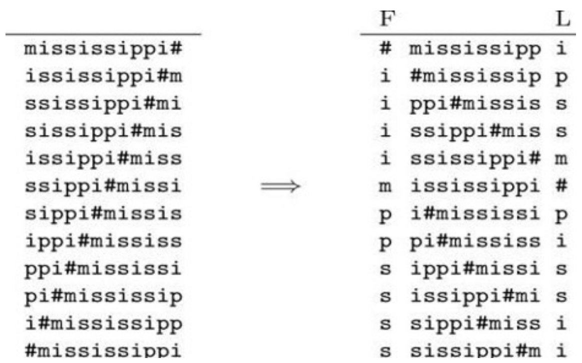


Figure 11 LZ77 algorithm.

### 3.3.3 Burrows-Wheeler transformation

The Burrows-Wheeler transformation rearranges a character string into similar character strings. This is useful for compression and decoding using algorithms, as the movetofront transform technique and run-length encoding make it easy to compress a string that has repeated character runs. The potential utility of the BWT of large amounts of shortread data (”reads”) has not been fully studied. The BWT basically serves as a dictionary of lossless reads. For example, in reading genomes, unlike the results of heuristic mapping and lossy reads that are conventionally obtained, the use of bwt is usually much more efficient. In the future, this is expected to lead to the development of sensitive methods for analyzing short-read data.[18] Figure 12 indicates how the Burrows-Wheeler transformation functions.

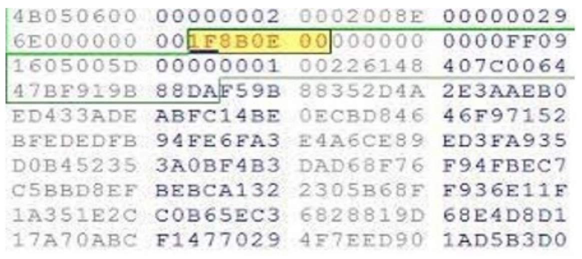


```
                      F            L
mississippi#          # mississipp i
ississippi#m          i #mississip p
ssissippi#mi          i ppi#missis s
sissippi#mis          i ssippi#mis s
issippi#miss          i ssissippi# m
ssippi#missi    ⟹    m ississippi #
sippi#missis          p i#mississi p
ippi#mississ          p pi#mississ i
ppi#mississi          s ippi#missi s
pi#mississip          s issippi#mi s
i#mississipp          s sippi#miss i
#mississippi          s sissippi#m i
```

Figure 12 Burrows-Wheeler transformation.

### 3.3.4 LZMA

In LZMA compression, the compressed sequence is a sequence of bits, encoded by an adaptive binary range encoder. The stream is divided into packets, each packet describing a single byte or LZ77 sequence with its length and distance encoded either implicitly or explicitly within each one. In addition, each part of each packet is modeled with independent contexts, so its probability predictions are correlated to each bit and this in turn is correlated with the values of the related bits of the same field) in previous packets of the same type.[19] In figure 13 the LZMA algorithm compression will be shown.



```
4B050600 00000002 0002008E 00000029
6E000000 001F8B0E 00000000 0000FF09
1605005D 00000001 00226148 407C0064
47BF919B 88DAF59B 88352D4A 2E3AAEB0
ED433ADE ABFC14BE 0ECBD846 46F97152
BFEDEDFB 94FE6FA3 E4A6CE89 ED3FA935
D0B45235 3A0BF4B3 DAD68F76 F94FBEC7
C5BBD8EF BEBCA132 2305B68F F936E11F
1A351E2C C0B65EC3 6828819D 68E4D8D1
17A70ABC F1477029 4F7EED90 1AD5B3D0
```

Figure 13 LZMA algorithm.

## 4 ALGORITHM DESIGN AND IMPLEMENTATION

In what follows, we explain the data structures and the algorithms used in this work. The implementations of the data structures and algorithms are available at Github[28]

## 4.1 Algorithms

In this work, we propose a compression algorithm which is a combination of a lossy image-compression algorithm and a lossless image-compression algorithm. We also explain how decompression for the proposed algorithm works.

We have implemented Fast Fourier transform merged with Huffman coding. Firstly, we applied the Fast Fourier transform and it returns a csv file, which is the input to apply the compression with huffman coding. Due to when we applied the huffman compression it gives us a binary file, we have to apply the huffman decompression. Finally to get the size of the original imagen, we applied the Fast Fourier Transform. In the following section the two algorithms selected are explained deeper.

## 4.2 Data Structures

In the Huffman coding the input is an array of unique characters, in this case an array with the unique pixels of the image with their frequencies of occurrences and the output is the Huffman tree. The Huffman tree is a binary tree generated for the exact frequencies of each pixel (in this case) [26]. It is really important because it allows you to assign the codes that will represent each unique pixel in the compression and decompression. These codes are made up of ones and zeros and that's because when building the tree, for every left edge we will assign number zero and for every right edge number one. Finally, you designate each code by traversing the tree from the root.

1. Create a leaf node for each unique pixel and build a min heap of all leaf nodes.
2. Extract two nodes with the minimum frequency from the min heap.
3. Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted as its right child. Add this node to the min heap.
4. Repeat steps 2 and 3 until the heap contains only one done. The remaining node is the root node and the tree is complete.

Another important data structures in the Huffman coding are the dictionaries and the priority queues. Remember that dictionaries are used to store a group of objects using a set of keys where each key has a single associate value, in this case we used a dictionary to store the frequency values of each pixel. And the priority queues, which are an abstract data structure where each element has a 'priority' associated with it, the elements with high priority are served first than the ones with low priority; in this case, we will store every frequency value of the dictionary in a priority queue[27]. Figure 14 illustrates a Huffman Tree.

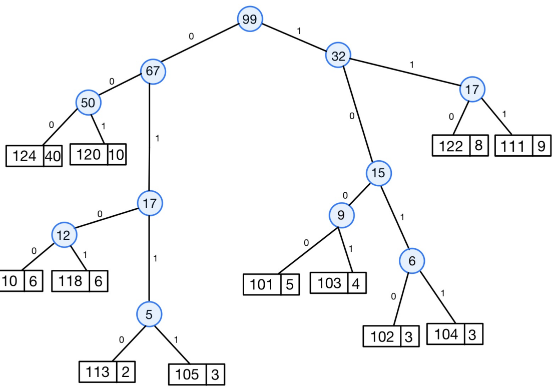


Figure 14 Huffman tree generated from a set of pixels of an image.

## 4.3 Algorithms

In this work, we propose a compression algorithm which is a combination of a lossy image-compression algorithm and a lossless image-compression algorithm. We also explain how decompression for the proposed algorithm works.

### 4.3.1 Lossy image-compression algorithm

**Fast Fourier Transform (FFT)**
The FFT is an optimized way to implement the Discrete Fourier Transform, in which the image´s data (the domain of pixels) is transformed into a set of frequencies in terms

of sines and cosines. The above process generates a coefficient matrix where the greater coefficients represent the highest frequencies, generally we keep around the 5% to 1% of the coefficients of the matrix with greater frequency because they are the ones that contribute the most in the generation of the new data matrix that will be the pixels of the compressed image. [20] Figure 15 shows a the sums in terms of frequencies which is a part of the process in the Fast Fourier Transform compression.

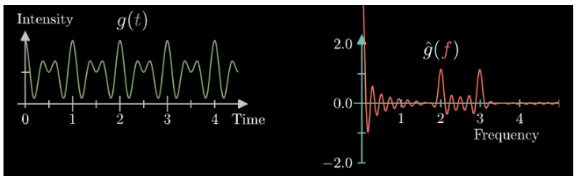


Figure 15 Function in terms of frequency.

What we do with the minor coefficients is that we change them to 0 because they are insignificant in the processing data, so we don´t take it into account. Finally, we use a sparse matrix, implemented with a linked list of linked lists, in other words a graph, which skips the weight of the zeros in the file and just stores the most important data, and that makes that image's weight less and the compression effectively. Next to that, we load the image and make the decompression. The decompression for this algorithm requires inversing the DFT matrix. To decompress, we are given a value representation of our polynomial and we want to find the coefficient representation, which means we have to multiply the value representation by the inverse of the DFT matrix.

### 4.3.2 Lossless image-compression algorithm

**Huffman Coding**
Huffman process is a lossless compression algorithm that works by analyzing the frequencies that certain symbols appear in a message. Those symbols that appear more often will be encoded as a shorter-bit string while symbols that aren't used as much will be encoded as longer strings.

Basically, we created a dictionary where we store the frequencies of each data and we created a priority queue with a minheap. Later, we implement a binary tree where the nodes store the dictionary, and sequentially it merges the pairs of nodes which sum the less frequencies, creating an intermediate node which stores that sum. Finally, it is assigned the respective code for each symbol, each code is the sequence from the root until each character following the labels of the edge.Figure 16 shows the result of the encoding process using Huffman algorithm.

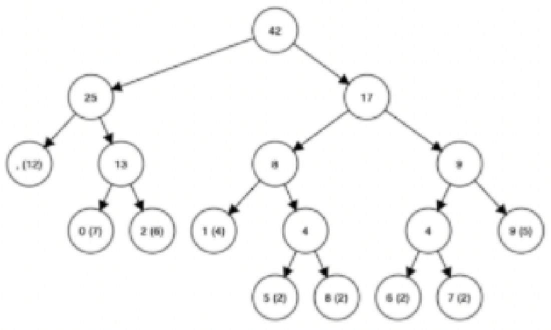


Figure 16 Huffman Encoding.

After creating the Huffman tree and saving it in a binary file, we have completed the compress phase. Then, to do the decompression we have to read the binary file and replace the valid Huffman Code bits with the character values, save the decoded data into a csv file and with that we must have the same file as we had before the compression.

## 4.4 Complexity analysis of the algorithms

In the worst case possible for the Huffman algorithm the time complexity is O(n*m*log(n*m)) where n is the total number of rows and m is the total number of columns of a picture. This is because when we create the huffman tree and we read it, we have to compute all the values that are O(n*m) and we compute each one of those values for compression or search them in the tree for decompression as O(log(n*m)).

Besides, the time complexity for Fast Fourier Transform is O(m*n*log(n*m)) for the worst case, because it divides the signal into two smaller signals (even and odds), compute the DFT of the two smaller signals and join them to get the DFT of the larger signal. In particular, the algorithm implemented with Fast Fourier Transform and Huffman coding will have a time complexity of O(m*n*log(n*m)) where n is the total number of rows and m is the total number of columns of a picture.

Let´s p = m*n, so the time complexity of Huffman Algorithm will be as follows. Table 1 shows the results of the time complexity for the worst case in the image compression and decompression.

| Algorithm | Time Complexity |
|---|---|
| Compression | O(p log(p)) |
| Decompression | O(p log(p)) |

**Table 1: Time Complexity of the image-compression and image-decompression algorithms. (p is the number of pixels of the image).**

The time complexity for the algorithm implemented with Fast Fourier Transform and Huffman coding will O(m*n) because with those algorithms we are creating a new matrix that takes a certain length and width, so with that n is the total number of rows and m is the total number of columns of a picture. Let´s k = m*n, so the time complexity of Huffman Algorithm will be as follows.Table 2 indicates the memory complexity for the worst case in the image compression and decompression.

| Algorithm | Memory Complexity |
|---|---|
| Compression | O(k) |
| Decompression | O(k) |

**Table 2: Memory Complexity of the image-compression and image-decompression algorithms. (Where k is the number of different pixels in the matrix)**

## 4.5 Design criteria of the algorithm

We decided to compress the image using FFT (Fast Fourier transform) and Huffman Algorithm because compared with others it was the most complete implementation, which satisfies all the requirements that we were asked for. Since theoretically the LZ77 has a better time complexity than the Huffman coding, we try to implement it. However, when we were coding it we realized that it doesn't exist any method to determine the window size and taking into account that it has a close relation with the compression ratio an the time consumption, it was really difficult to determine which was the optimal window size to compress an specific data because it change depending on the data size.

Also, we implemented LZW to compare with the other ones, this algorithm had a good compression ratio and it was efficient. However, it was not proficient to compress csv files, only as png and jpg images, which was a critical point due to that format already have a compression and in the case of jpg some loss of information, so it was not the optimal way to compress because it makes the images bigger or smaller based on their current size.

In fact, merging the lossy compression (FFT) and the lossless compression (Huffman coding) we get an appropriate time complexity and a better compression ratio of 4:1. Despite this, the memory consumption was increased when we combined those algorithms.

## 5 RESULTS

### 5.1 Execution Times

In what follows we explain the relation of the average execution time and average file size of the images in the data set, in Table 3 and Table 4.

| | Average execution time (s) | Average file size(MB) |
|---|---|---|
| Compression | 6.77 s | 15.37 MB |
| Decompression | 10.99 s | 15.37 MB |

**Table 3: Execution time of Fast Fourier Transform merged with Huffman coding and decoding both algorithms for different images in the data set.**

| | Average execution time (s) | Average file size(MB) |
|---|---|---|
| Compression | 53.78 s | 1.3 MB |
| Decompression | 67 s | 1.3 MB |

**Table 4: Execution time of Fast Fourier Transform merged with Huffman coding and decoding both algorithms for different images in the data set.**

### 5.2 Memory consumption

We present memory consumption of the compression and decompression algorithms in Table 5 and Table 6.

### 5.3 Compression ratio

We present the average compression ratio of the compression algorithm in Table 7.

| | Average memory consumption(s) | Average file size(MB) |
|---|---|---|
| Compression | 60.29 MiB | 0.49 MiB |
| Decompression | 75.37 MiB | 0.49 MiB |

**Table 5: Average Memory consumption of all the images in the data set for both compression and decompression implemented with Huffman Coding.**

| | Average memory consumption(s) | Average file size(MB) |
|---|---|---|
| Compression | 1283.31 MiB | 0.24 MiB |
| Decompression | 1304.35 MiB | 0.24 MiB |

**Table 6: Average Memory consumption of all the images in the data set for both compression and decompression implemented with FFT and Huffman Coding**

| | Healthy Cattle | Sick Cattle |
|---|---|---|
| Average compression ratio for Huffman coding. | 5 : 2 | 5 : 2 |
| Average compression ratio for FFT merged with Huffman coding | 4 : 1 | 4 :1 |

**Table 7: Rounded Average Compression Ratio of all the images of Healthy Cattle and Sick Cattle**

## 6 DISCUSSION RESULTS

When we look at the results obtained by the two types of compression we can see different types of results. For the lossy compression we have a high compression capacity in comparison with the lossless; also with a lower execution time. The FFT implemented with a sparse matrix shows a really good compression where the rate is 32:1 and even with this high rate the quality of the image is good, the loss of information is not high, so a lot of the details can be seen after the decompression. However, this algorithm has its own disadvantages like the huge memory consumption and the noise it can create in the image with the compression rate that make the classification algorithm less precise when it comes to sorting the images.

On the other side we have the lossless algorithm, Huffman coding, which gives us a high quality and precision of the image and this method has a efficient complexity in terms of time and memory. However, this algorithm has a smaller average compression ratio compared with the Fast Fourier Transform implemented with a sparse matrix, it is 5:2

Thus, for the final implementation where we compress with FFT and Huffman we realized that it gives us a better compression ratio, increasing from 5:2 to 4:1 and it still has an appropriate time complexity taking into account that we are using two compression algorithms. Nevertheless, the memory complexity was affected, because with this implementation it requires much more memory to execute all the operations of the code.

## 6.1 Future Work

We would like to improve in our algorithms to implement the image scaling compression and merge its compression with the LZ77 encoding to reduce the time consumption and improve the compression ratio. The objective of implementing that algorithm is to compare its functionality with the implementation of Fast Fourier Transform with Huffman Code and Fast Fourier Transform with sparse matrices to verify which one has better results.

Also some ideas of using wavelet compression came out, but for this we have to do better research on the subject but this is a good option to consider for the future.

## 7 Acknowledgments

We would like to express our gratitude to Kevin A. Sossa Chavarria who helped and supported us by explaining some things like how to read the csv files, how some of the compression algorithms work and things about complexity. Also, we want to express our thanks to our friend Kevin López who supported us with the coding process, correcting our mistakes, teaching us new techniques and how to optimize each code. Finally, thanks to Isabel Piedrahita Velez that answered all the doubts regarding to the work.

## 8 References

[1] From Las Naciones Unidas Para La Alimentación y la Agricultura: http://www.fao.org/animalproduction/es/

[2] Toro M. Marín,S. Proyecto Final ED1-Algoritmos de compresión para optimizar el consumo de batería en la ganadería de precisión.

[3] D.Berckmans. General Introduction to precision livestock farming. January 2017. From academic oup:https://watermark.silverchair.com/6.pdf?token=AQECAHi208BE49Ooan9kkhW_Ercy

[4] Doulgerakis, V., Kalyvas, D. Bocaj, E. Giannousis, C., Feidakis, M. Laliotis, G. P. Patrikakis, C. Bizelis, I. , 2019. An animal welfare platform for extensive livestock production systems. In: CEUR Workshop Proceedings, vol. 2492.

[5] Andrew, W., Greatwood, C., Burghardt, T., 2017. Visual localisation and individual identification of holstein friesian cattle via deep learning. In: 2017 IEEE International Conference on Computer Vision Workshops (ICCVW), pp. 2850–2859.

[6] Debauche, O., Mahmoudi, S., Andriamandroso, A.L.H., Manneback, P., Bindelle, J., Lebeau, F., 2019. Cloud services integration for farm animals' behavior studies based on smartphones as activity sensors. J. Ambient Intell. Humanized Comput. 10 (12), 4651–4662

[7] Rodrigo Garcia, Jose Aguilar, Mauricio Toro, Angel Pinto, Paul Rodriguez. A systematic literature review on the use of machine learning in precision livestock farming. Computers and Electronics in Agriculture Volume 179, December 2020.

[8] Wikipedia contributors. (2021, 18 enero). Fractal compression. Wikipedia. https://en.wikipedia.org/wiki/Fractal$_{c}ompression$

[9] Wikipedia contributors. (2020, 7 noviembre).Seam carving.Wikipedia.https://en.wikipedia.org/wiki/Seam$_{c}arving$

[10] Xiuhua, J., Caiming, Z., Jiaye, W., Kai, W. 2009. A Fast 2D*8x8 Discrete Cosine Transform Algorithm for Image Coding. Science in China Series F Information Sciences 52(2):215-225 11. Kumar, A. 2021. Discrete Cosine Transform (Algorithm and Program). From:https://www.geeksforgeeks.org/discretecosine-transform-algorithm-program/

[11] Wikipedia Contributors. Discrete Cosine Transform, 2020. From: https://en.wikipedia.org/wiki/Discrete$_{c}osine_{t}ransform$

[12] Wikipedia Contributors. Image Scaling, 2021. From: https://en.wikipedia.org/wiki/Image$_{s}caling14$.

[13] Tabora, V. JPEG Image Scaling Algorithms, 2019. From:https://medium.com/hd-pro/jpeg-imagescaling-algorithms- 913987c9d588: :text=JPEG%20Scaling%20Techniques,nearest

[14] Huffman coding compression algorithm. From: https://www.techiedelight.com/huffman-coding. 16. Huffman coding. 2021. From: https://www.geeksforgeeks.org/huffman-codinggreedy-algo-3

[15] LZ77 Compression algorithm, March, 2020. From: https://docs.microsoft.com/enus/openspecs/windows protocols/mswusp/fb98aa28-5cd7-407f-8869-a6cef1ff1ccb.

[16] Kimura, K., Koike, A. (2015, 9 diciembre). Analysis of genomic rearrangements by using the Burrows-Wheeler transform of short-read data.BMC Bioinformatics. https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-16-S18-S5

[17] lzma — Compression using the LZMA algorithm — Python 3.9.1 documentation. (2021, enero). Compression using the LZMA algorithm. https://docs.python.org/3/library/lzma.html

[18] Steven L. Brunton,J. Nathan Kutz(2019)Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control.

[19] Steve Brunton. (2020, 31 marzo). The Fast Fourier Transform (FFT). YouTube. https://www.youtube.com/watch?v=E8HeD-MUrjYt=116s

[20] Reducible. (2020, 14 noviembre). The Fast Fourier Transform (FFT): Most Ingenious Algorithm Ever? YouTube. https://www.youtube.com/watch?v=h7apO7q16V0t=789s

[21] Steve Brunton. (2020b, abril 4). The Fast Fourier Transform Algorithm. YouTube. https://www.youtube.com/watch?v=toj$_{I}oCQE-4t=175s$

[22] Steve Brunton. (2020c, junio 8). Image Compression and the FFT (Examples in Python). YouTube. https://www.youtube.com/watch?v=uB3v6n8t2dQ

[23] Creative. (2019, 10 agosto). Fast Fourier Transform (FFT) Algorithm. ALLSIGNALPROCESSING.COM. https://allsignalprocessing.com/lessons/fast-fourier-transform-fft-algorithm/

[24] Steve Brunton. (2020c, junio 8). Image Compression and the FFT (Examples in Python). YouTube.

[25] Creative. (2019, 10 agosto). Fast Fourier Transform (FFT) Algorithm. ALLSIGNALPROCESSING.COM. https://allsignalprocessing.com/lessons/fast-fourier-transform-fft-algorithm/

[26] Geek for Geeks (2021, May 23). Huffman Coding. https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/

[27] Wikipedia Contributors (2021, May 5). Priority queue. https://en.wikipedia.org/wiki/Priority$_{q}ueue$

[28] https://github.com/imontoyah/ST0245-002/tree/master/proyecto

## 8.1 Images´ References

[29] Doulgerakis, V., Kalyvas, D. Bocaj, E. Giannousis, C., Feidakis, M. Laliotis, G. P. Patrikakis, C. , Bizelis, I. , 2019. An animal welfare platform for extensive livestock production systems [Fotography]. In: CEUR Workshop Proceedings, vol. 2492.

[30] Rodrigo Garcia, Jose Aguilar, Mauricio Toro, Angel Pinto, Paul Rodriguez. A systematic literature review on the use of machine learning in precision livestock farming [Fotography]. Computers and Electronics in Agriculture Volume 179, December 2020.

[31] Wikipedia contributors. (2021, 18 enero). Fractal compression [Photography]. Wikipedia. https://en.wikipedia.org/wiki/Fractal$_compression$

[32] Wikipedia contributors. (2020, 7 noviembre).Seam carving [Photography] .Wikipedia.https://en.wikipedia.org/wiki/Seam$_carving.$

[33] Spie Digital Library (2014, september). Discrete Cosine Transformation [Fotography]. https://www.spiedigitallibrary.org/journals/journal -of-electronic-imaging/volume-23/issue6/061110/Adaptive-discrete-cosine-transformbased-image-compression-method-ona/10.1117/1.JEI.23.6.061110.short?SSO=1tab=A rticleLink

[34] Super User (2016, August). Image Scaling Algorithm [Photography]. https://superuser.com/questions/375718/whichresize-algorithm-to-choose-for-videos

[35] Techie Delight. Huffman Coding [Fotography]. https://www.techiedelight.com/huffman-coding/.

[36] Shi, P., Li, B., Thinke, B., Ding, L. (2018, January). LZ77 encoding example [Photography]. https://www.researchgate.net/figure/An-exampleof-LZ77-encoding$_{fig}4_{3}$22296027$Ferragina, P.$(2005, $july).Burrowswheelertransform[Photography].https :$ $//www.researchgate.net/figure/Example - ofBurrows - Wheeler - transform - for - the - string - Tmississippi - The - matrix - on - the_{fig}5_{2}$20430619.

[37][38] Waste, G. (2014, 8 october). LZMA compression method [Fotography]. LZMA compression method in GZIP files. https://forum.xentax.com/viewtopic.php?f=21t= 12065.

[39] Illustration 18 Huffman Tree https://people.ok.ubc.ca/ylucet/DS/Huffman.html