COMPRESSION AND DECOMPRESSION OF IMAGES FOR PRECISION FARMING

Alejandro Torres Muñoz
Universidad Eafit
Colombia
eatorresm@eafit.edu.co

Mateo Muñoz Cadavid
Universidad Eafit
Colombia
mmunozc4@eafit.edu.co

Simón Marín
Universidad Eafit
Colombia
smaring1@eafit.edu.co

Mauricio Toro
Universidad Eafit
Colombia
mtorobe@eafit.edu.co

ABSTRACT

This report seeks to solve the compression and decompression of images in order to optimize energy consumption in the context of precision farming, and in this way, that data management in traditional farming is optimal to achieve a good analysis of animal health production.

There are several similar problems:

• **"An Animal Welfare Platform for Extensive Livestock Production Systems",** which seeks automated control of data to achieve optimal animal welfare for agricultural production. A connection system is implemented between a collar that is responsible for reading animal behavior and the cloud where the analyzed data is stored.

• **"Visual localization and individual identification of Holstein Friesian cattle via deep learning",** algorithm that, through computer vision pipes, uses deep neural architectures, which are suitable for automated detection of Holstein Friesian cattle, as well as individual identification in relevant agricultural settings.

• **"Cloud services integration for farm animals' behavior studies based on smartphones as activity sensors",** which expresses how Smartphones are suitable for researching animal behavior because they contain essential components such as accelerometers that measure inertial acceleration, a gyroscope that measures angular rotation and a magnometer that improves the accuracy of gyroscopic measurements. The article implements a "lambda cloud architecture" that is used to archive and process high-frequency data

Keywords

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

1. INTRODUCTION

Traditional livestock is focused on animal products for human consumption (meat, milk, …) which provide a large percentage of the protein in the human diet. Data on animal husbandry and farms in general are found in notebooks or in systems that are not very viable to handle such information. In himself, decisions about data in livestock are based on the experience of the producer.

Technology has provided great support to livestock production, thus achieving that such data on animal behavior is automated and processed daily. Recently a term of Compression Livestock (GdP) has emerged, which implements information and communication technologies and therefore allows a correct handling of all production and animal health data.

1.1. Problem

It seeks to create an algorithm that achieves compression and decompression of images in order to automate all the data that guarantees a good management of precision livestock (GdP). This is achieved based on various types of algorithms investigated. The system must achieve the best possible compression.

1.2 Solution

In this project, 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.

We chose image scaling because it can satisfy in a practical way the objectives planned and proposed in the project.

It is an efficient algorithm for the compression and decompression of images, since it allows us to transform the size of an image and compared to the process time that others take compared to its construction and operation, it is easier to implement.

Along with nearest neighbor interpolation, one of its advantages is its simplicity and its common use in real-time image rendering.

1.3 Article structure

In what follows, in Section 2, we present related work to the problem. Later, in Section 3, we present the data sets and methods used in this research. In Section 4, we present the algorithm design. After, in Section 5, we present the results. Finally, in Section 6, we discuss the results, and we propose some future work directions.

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.

3.1 Cloud services integration for farm animals' behavior studies based on smartphones as activity sensors.

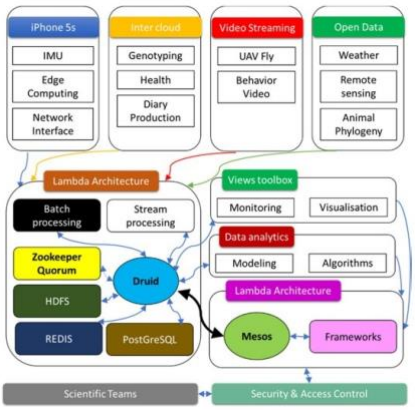
The use of sensors is being used on a large scale in livestock to achieve better management of animal behavior, since they provide essential information on animal health and status. For this, 2 main components are required:

1) The location obtained by radio conference triangulation or by the global positioning system (GPS).
2) The low-frequency component of behavior. For the large data handling that this requires, an architecture of the lambda cloud is proposed, using Smartphones as a basis, since they can be very valuable instruments for researching animal behavior, since they do not require much hardware development, they contain a large amount sensor and are equipped with high performance Inertial Measurement Units (IMU) and absolute positioning systems.

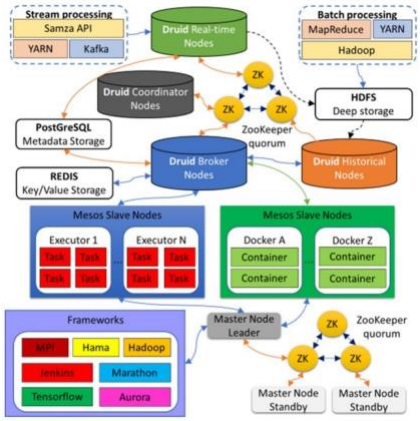
The researchers of this system aim to implement technology in precision farming (PLF). An open-source observer-based classification algorithm is suggested in iPhone IMU for the implementation of essential data and information for the identification of behavior, this with high precision and at the same time with sensors that must be optimized to reduce power consumption.

As mentioned before, Smartphones are suitable for the correct management of data in livestock with IMU units, which contain an accelerometer that measures inertial acceleration, a gyroscope that measures angular rotation and a magnometer that improves the precision of measurements. gyroscopic. The IMUs of the iPhone 4s / 5s were used as they measure various raw signals. Their precision was evaluated by incorporating them into the animals and thus calculating the variations of the IMU measurements. The battery of the device was evaluated achieving a frequency of 100Hz being the maximum supported by the device. All data recovered by phones must be processed and stored in the cloud. Because of this, several options were suggested:

1) **Hadoop:** A dedicated system to store large amounts of data but does not offer a guarantee of speed.
2) **Apache Spark:** Processes large amounts of data with low latency. You need an external storage.
3) **Apache Storm, Apache Spark Streaming and Apache Samza:** Low latency models, processes data in real time presenting errors in data precision.
4) **Druid:** Fault-tolerant, real-time data warehouse.
5) **Apache Mesos and Apache Zoopkeper:** They store data with great tolerance to faults, being mesos a great solution to implement because it allows hosting applications for multiple use cases.



The lambda cloud architecture is designed to handle large amounts of data, it adapts to a wide range of use cases, being capable of handling all types of data. Data is separated into video streams processed by streaming processing and time / event related data.



Yang F, Tschetter E, Léauté X, Ray N, Merlino G, Ganguli D (2014) A real-time analytical data store. SIGMOD'14, June 22–27, 2014, Snowbird, UT, USA. ACM 978-1-4503-2376-5$414/06.

3.2 Visual Localization and Individual Identification of Holstein Friesian Cattle via Deep Learning

The problem of automated biometric identification of cattle has been well studied over time. The approaches can be divided into three categories:

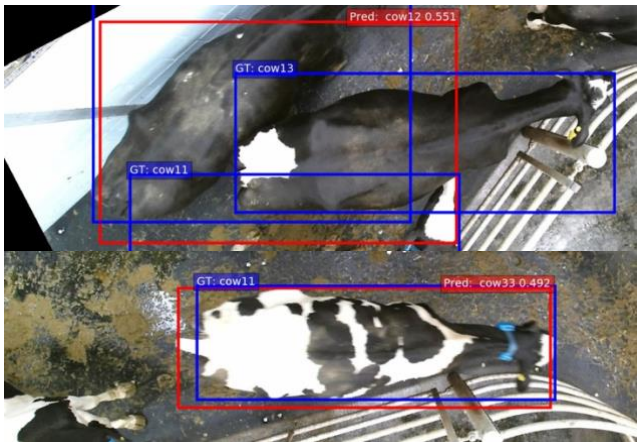
1. Those who use cattle bovine patterns.
2. Rarer systems that employ retinal, facial or body scans
3. Those that take advantage of the characteristics of the coat pattern.

An algorithm is proposed using computer vision pipelines, powered by standard architectures using deep neural networks. Computer vision pipelines using deep neural architectures are suitable for performing automated detection of Holstein Friesian cattle, as well as individual identification in relevant agricultural settings. In this way, robust identification of individual Holstein Friesian cattle can occur automatically and non-intrusively.

Using standard networks, end-to-end identification can be performed on top-down still images acquired with still cameras, presenting a video processing pipeline made up of standard components to efficiently process dynamic herd footage, filmed by unmanned aerial vehicles (UAV).

Detection and location of Friesian cattle can be robustly performed with 99.3% accuracy on this data. The individual identification taking advantage of the uniqueness of the fur in 940 RGB stills taken after milking in the barn (89 individuals, precision = 86.1%), also evaluating the identification through a video processing pipeline at 46,430 frames originating from 34 clips (approximately 20 s in length each) of UAV images taken during grazing (23 individuals, accuracy = 98.1%). These tests suggest that, particularly when videotaping small herds in clear environments, a markerless Friesian cattle identification application is not only feasible using standard deep learning components but appears robust enough to aid marking methods. existing.

(a) Predicted and ground truth (b) Predicted and ground truth IDs = {3, 10} respectively. IDs = {11, 14} respectively.

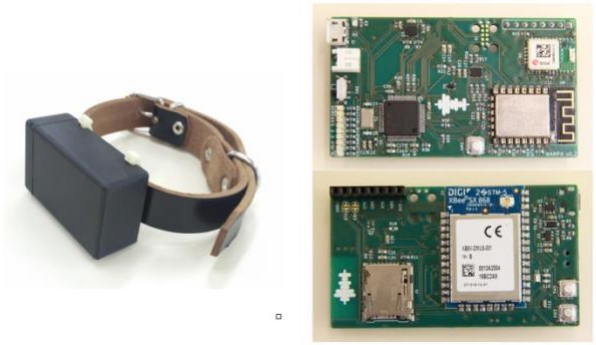


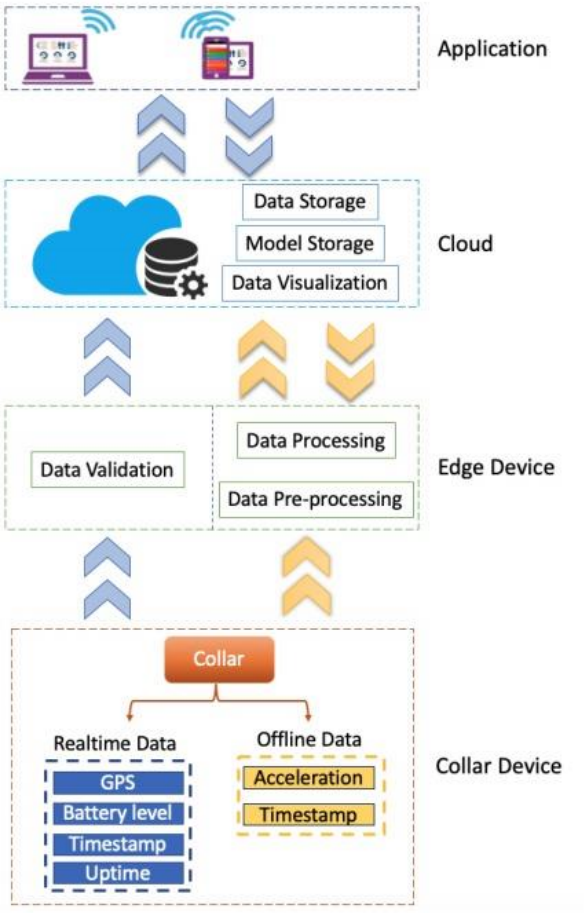
Single Frame Individual Identification

3.3 An Animal Welfare Platform for Extensive Livestock Production Systems

Previously, agricultural production focused more on the needs of the customer, but as time went by, consumers demanded stricter animal welfare standards, which led to demands to produce with greater respect for animals. This went so far that the EU Common Policy clarifies that farmers must achieve high welfare standards in order to receive a payment.

Technologies were implemented to automate the monitoring of animal behavior. Using a TI's Sensor Tag (temperature, humidity and pressure sensor), an intelligent animal production management system is proposed. This system is implemented by means of a collar, capable of collecting all the animal's data (blood pressure, heart rate, hormone levels, feeding changes, among others), such data is taken by an Edge-device which is responsible for managing, processing and providing analysis of the data. Consequently, the data is taken to the cloud, composed of Java SprinBoot. The mobile app is responsible for reporting on the location and welfare of the animal.





Min Chen, Wei Li, Giancarlo Fortino, Yixue Hao, Long Hu, and Iztok Humar. A dynamic service migration mechanism in edge cognitive computing. ACM Transactions on Internet Technology, 19(2):30:1–30:15, April 2019.

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

A new term has been implemented in livestock farming, which is determined as precision livestock farming (PLF). It is based on adding information and communication technologies to improve the crop and livestock process, thus increasing production and animal welfare, while reducing investment costs. *To implement it, the models of agricultural companies, interoperable standards in livestock systems, dairy energy and animal health were analyzed by means of systematic literature review (SLR).

Machine Learning (ML) and control systems are implemented to analyze the quantitative data obtained in real time. The main objectives of PLF are to identify appropriate feeding, reduce environmental impact, manage farming processes, ensure food safety, improve animal health and crop efficiency.

Several systems and algorithms were presented, one of the most significant was a combination between the offline K-nears neighbors (KNN) and an online learning algorithm, this algorithm classifies relevant sheep behavior. A variation of KNN was to classify livestock behavior between active and inactive, it was called hierarchical ML which using Random Forest (RF), Support vector machine (SVM) and Deep Belief Networks (DBN) classifiers achieved much better than conventional KNN.

To obtain more information on behavior, 3 ethograms were used: grazing, lying, standing, walking, active and inactive behavior and body posture. Behavioral characteristics were evaluated using 4 ML algorithms; Classification and regression trees, SVM, linear discriminant analysis and quadratic discriminant analysis.

Makinde, A., Islam, M.M., Scott, S.D., 2019. Opportunities for ACI in PLF: Applying animal- And user-centred design to precision livestock farming. In: ACM International Conference Proceeding Series. Association for Computing Machinery.

Yang, Q., Liu, Y., Chen, T., Tong, Y., 2019. Federated machine learning: Concept and applications. ACM Trans. Intell. Syst. Technol. 10 (2).

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/ST0245-Eafit/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.

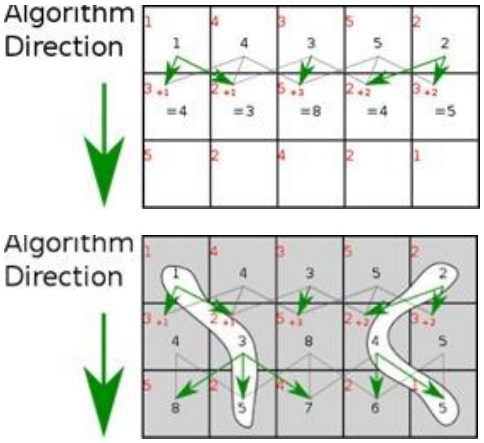
3.2 Lossy Image-compression alternatives

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

3.2.1 Seam Carving

It is an algorithm for image resizing, which is done through seams (Minor Paths) in an image and automatically removes the seams to reduce the image size or insert seams to extract them. The seam carving is able to define areas in which the image pixels cannot be modified. Its goal is to solve the projection of the image without distortion on media of different sizes, using document standards that support dynamic changes in the page and text, but not in the image, such as HTML. The seams can be vertical; it is a path of pixels connected from top to bottom in an image with one pixel in each row, or horizontal; similar to vertical, but from left to right. The problem with this algorithm is based on the need for user-supplied information to reduce errors, sometimes the algorithm can create a higher energy seam by eliminating a lower energy seam.

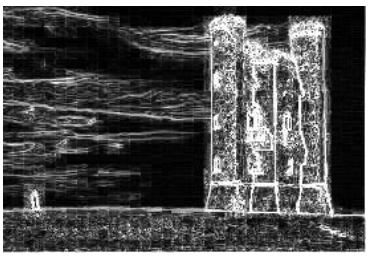
The seam carving has a dynamic programming, which is a method for storing the results of complex calculations, by means of which the seams can be calculated. The lowest energy vertical seam is calculated for each pixel in a row by calculating the energy of the current pixel plus the energy of one of the pixels above it.



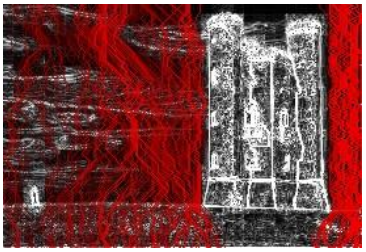
Resizing process:

1. Start with an image.

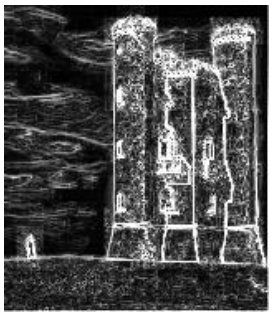
2. Calculate the weight / density / energy of each pixel. This can be done by several algorithms: gradient magnitude, entropy, visual saliency, gaze motion.



3. From the energy, make a list of seams. The seams are sorted by energy, with low energy seams being of less importance to the image content. The seams can be calculated using the dynamic programming method shown below.



4. Remove low energy seams as necessary.



5. Final image.

3.2.2 Image scaling

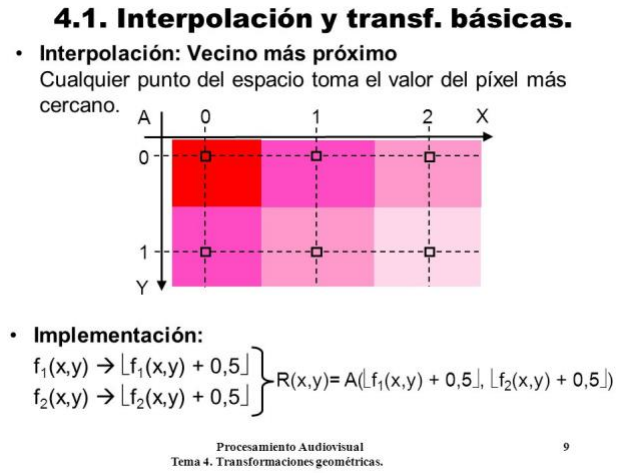
When scaling a vector graphic image, the graphic primitives that make up the image can be scaled using geometric transformations, without loss of image quality. When scaling a bitmap graphics image, a new image must be generated with a higher or lower number of pixels. In the case of decreasing the number of pixels (scaling down), this generally results in a loss of visible quality. From a digital signal processing point of view, raster graphics scaling is a two-dimensional example of sample rate conversion, converting a discrete signal to one sample rate (in this case, the local sample rate) to another.

(a) (b) (c) (d)

Here are four algorithms for resizing an image:

- **Nearest neighbor interpolation**

Each pixel is replaced with the closest pixel in the output to increase the scale, this means that multiple pixels of the same color will be present. This can preserve sharp details in pixel art, but also introduce unevenness in previously smooth images. A common implementation is to always round to zero. Rounding in this way produces fewer artifacts and is faster to calculate.



- **Box sampling**

It consists of considering the target pixel as a box in the original image and sampling all the pixels within the box. This ensures that all input pixels contribute to the output. The biggest weakness of this algorithm is that it is difficult to optimize.

- **Edge-directed interpolation**

Edge-directed interpolation algorithms aim to preserve edges in the image after scaling, unlike other algorithms, which can introduce ladder artifacts.

- **Fourier transform methods**

Simple interpolation based on the Fourier transform fills the frequency domain with zero components (a soft focus based on windows would reduce the timbre). In addition to the good preservation (or recovery) of the details, the ringing and circular bleeding of the content stands out from the left edge to the right (and vice versa).

3.2.3 Discrete cosine transformation

The DCT has a good compaction capacity since it obtains most of the information in a few coefficients. This transformation is independent of the data, it does not vary with the data it receives. It manages to encode the coefficients using quantization values. The DCT can be

implemented in image compression due to its high correlation of elements.

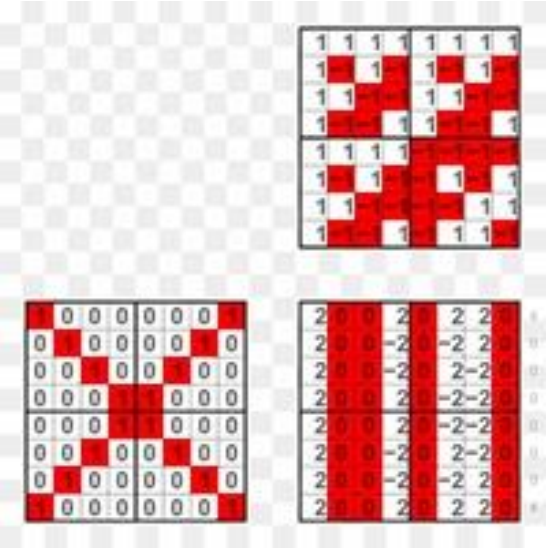
The function takes a set of discrete input values, generating another set from it, with a matrix of I, J variables that vary in each call, thus achieving a fast execution. In general, the blocks are 8x8 / 16x16, these blocks are traversed, achieving a structural decomposition, which can be used adaptively by selecting the coefficients of the transform block by block. The algorithm receives an image composed of pixels, takes the value of these pixels and transforms them to coefficients, due to the correlation of some coefficients these are truncated or omitted in order to obtain a compressed image. There is an extra implementation of this algorithm, which consists in predicting the value of a pixel according to the previous tables (if the value of a pixel can be predicted, this pixel should not be encoded).

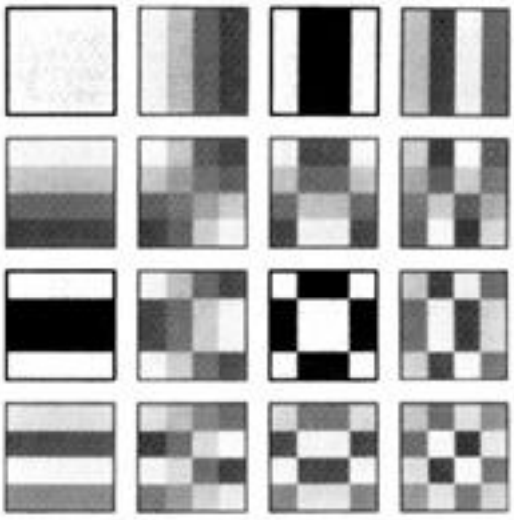
DCT:

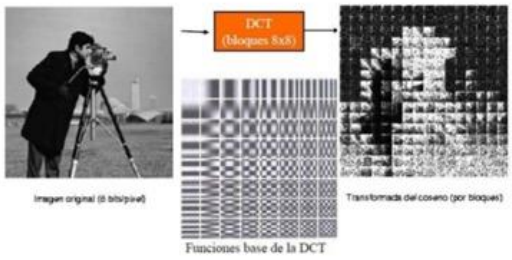
1. Sampling spaces of short duration are taken.

2. Each sample space is considered as half of an even and periodic signal. In this respect there are four 4 strategies that can be used to convert the sample space into an even and periodic signal. The Figure 2 illustrates these strategies.

3. A modified DCT is applied according to the strategy used to convert the sample space into an even and periodic signal. Even and periodic signal.







Funciones base de la DCT

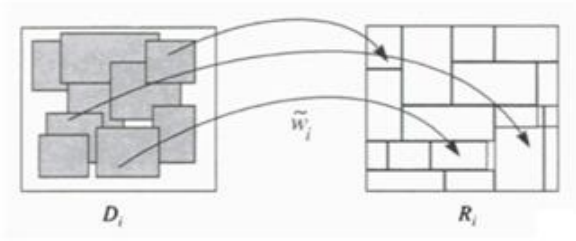
3.2.4 Fractal Compression

1) We split f into a series of range cells {Ri}. These cells must completely cover f and must not overlap.

2) We cover f with a set of cells domain {Di}. Usually there are many domain cells, and they overlap. The greater the number of Di the better the final result, but the compression will take longer.

3) For each Ri, we search in the set of {Di}the domain cell and the transformation Wi that best cover it. The Wi consists of a rotation, a reduction and a brightness and contrast adjustment. We say that a cell Di covers a cell Ri if the dRMS between them is less than a given tolerance.

4) If the fit was not good enough according to the tolerance, we divide the Ri cell into four subcells and repeat the previous step for each subcell.



3.3 Lossless Image-compression alternatives

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

3.3.1 Huffman coding

It is an algorithm used for understanding data. It consists of creating a binary tree in which the leaf nodes are labeled with the characters, along with their frequencies, and each pair of nodes that less frequently add up are consecutively joined, creating a new intermediate node labeled with said sum. This action is carried out until there are no leaf nodes left to join any superior node, and the binary tree has been formed.

Subsequently, the edges that join each of the nodes are labeled with zeros and ones (right and left child, respectively). The resulting code for each character is the reading, following the branch, from the root to each character (or vice versa) of each of the edge labels.

3.3.2 LZ77 and LZ78

LZ77 and LZ78 are the two lossless data compression algorithms. Both are, in theory, dictionary coders. LZ77 maintains a sliding window during compression, it encodes and decodes from a sliding window over the characters, decompression must always start at the beginning of the input. Decompressing the LZ78 could allow random access to the input if the entire dictionary was known beforehand. However, in practice, the dictionary is created during encoding and decoding by creating a new phrase each time a token is issued.

The LZ77 algorithms achieve compression by replacing repeated occurrences of data with references to a single copy of that data previously existing in the uncompressed data stream. A match is encoded by a pair of numbers called a length-distance pair, which is equivalent to the statement "each of the following characters in length is equal to the characters exactly at a distance behind it in the uncompressed stream." (The distance is sometimes called the offset.)

The LZ78 algorithms achieve compression by replacing repeated occurrences of data with references to a dictionary that is built based on the input data stream. Each dictionary entry is in the format dictionary [...] = {index, character}, where index is the index of a previous dictionary entry and the character is added to the string represented by dictionary [index].

LZ77 Example: Decompression

```
Decoding stored sequence
(0,0)     t    t
(0,0)     i    ti
(0,0)     p    tip
(1,1)     _    tipp_
(5,1)     a    tipp_ta
(4,3)     i    tipp_tap_ti
(9,9)     p    tipp_tap_tipp_tap_tip
(1,1)     e    tipp_tap_tipp_tap_tippe
(6,6)     _    tipp_tap_tipp_tap_tippe_tippe_
(21,7)    p    tipp_tap_tipp_tap_tippe_tippe_tipp_tap
Compressed file size: 78.95%
```

3.3.3 Burrows–Wheeler transform

The Burrows–Wheeler transform is an algorithm used to prepare data for use with data compression techniques such as bzip2.

Rearranges a character string into similar character strings. This is useful for compression, as it tends to be easy to compress a string that has repeating character runs using techniques such as move-to-front transform and run-length encoding.

The transform is done by sorting all the circular shifts of a text in lexicographic order and by extracting the last column and the index of the original string in the set of sorted permutations of S.

Given an input string S = ^BANANA| (step 1 in the table below), rotate it N times (step 2), where N = 8 is the length of the S string considering also the symbol ^ representing the start of the string and the red | character representing the

'EOF' pointer; these rotations, or circular shifts, are then sorted lexicographically (step 3). The output of the encoding phase is the last column L = BNN^AA|A after step 3, and the index (0-based) I of the row containing the original string S, in this case I = 6.

Transformation				
1. Input	2. All rotations	3. Sort into lexical order	4. Take the last column	5. Output
^BANANA|	^BANANA| |^BANANA A|^BANAN NA|^BANA ANA|^BAN NANA|^BA ANANA|^B BANANA|^	ANANA|^B ANA|^BAN A|^BANAN BANANA|^ NANA|^BA NA|^BANA ^BANANA| |^BANANA	ANANA|^B ANA|^BAN A|^BANAN BANANA|^ NANA|^BA NA|^BANA ^BANANA| |^BANANA	BNN^AA|A

URL FOR WATCHING VIDEO ON YOUTUBE:

https://www.youtube.com/watch?v=4n7NPk5lwbI

3.3.4 LZSS

Decoding Strings:

Step1. Initialize the dictionary to a known value.

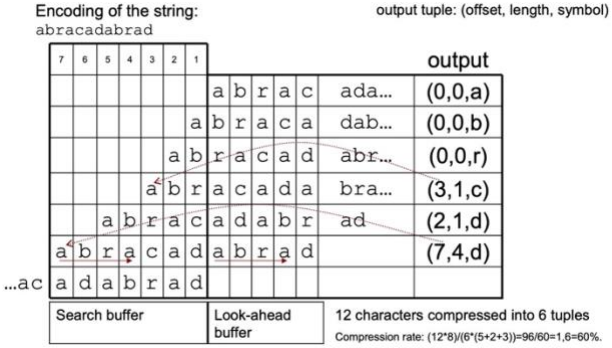
Step 2. Read the encoded/not encoded flag.

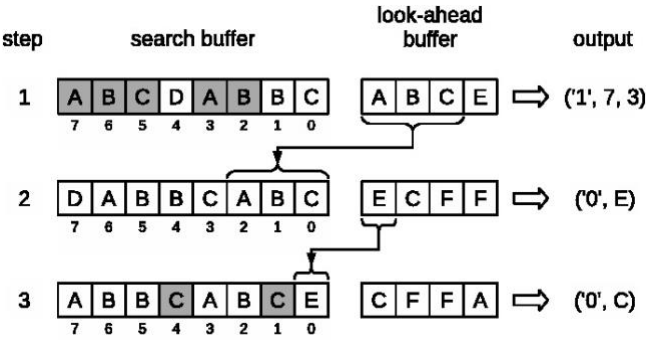
Step 3. If the flag indicates an encoded string:

> **A**: Read the encoded length and offset, then copy the specified number of symbols from the dictionary to the decoded output.
>
> **B:** Otherwise, read the next character and write it to the decoded output.

Step 4. Shift a copy of the symbols written to the decoded output into the dictionary.

Step 5. Repeat from Step 2, until all the entire input has been decoded.





URL FOR WATCHING VIDEO ON YOUTUBE:

https://www.youtube.com/watch?v=hEy2bvWO6gk

4. ALGORITHM DESIGN AND IMPLEMENTATION

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

4.1 Data Structures

In this project we use 2D matrices as data structures, which are tables of values, stored in rows and columns, which allow us to store a set of data of the same type. The matrix is defined with a single name and each element of it is referenced by means of two subscripts.

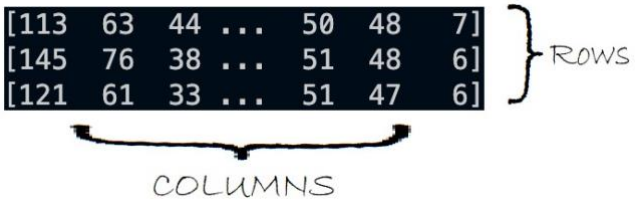


Figure 1: 2D matrix generated by scaling images in Python.

In this example we store integer values. All elements of the array must be of the same type (int, float, String etc.). The rows and columns start to be numbered from zero, similar to vectors.

4.2 Algorithms

In this project, 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.

Image scaling is the resizing of a digital image. When scaling a vector graphic image, the graphic primitives that make up the image can be scaled using geometric transformations, without loss of image quality. When scaling a bitmap graphics image, a new image must be generated with a higher or lower number of pixels. In the case of decreasing the number of pixels (scaling down), this generally results in a loss of visible quality.

Together with the nearest neighbor interpolation, we increase or decrease the size of the image by replacing each pixel with the closest one in the output, this by means of 2D matrices which make up the image (Height and width).

COMPRESSION:

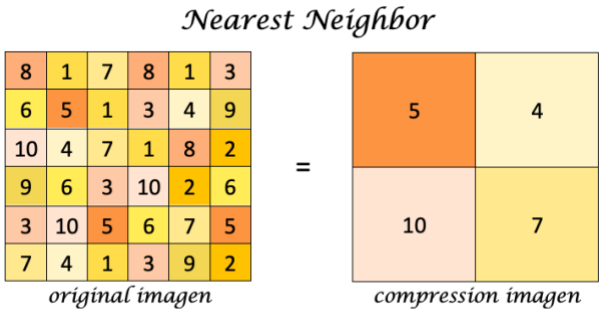


Figure 1: Compression generated from Nearest Neighbor.

The BFS algorithm was used for decompression.

BFS is a traversing algorithm where you should start traversing from a selected node (source or starting node) and traverse the graph layerwise thus exploring the neighbor nodes (nodes which are directly connected to source node). You must then move towards the next-level neighbor nodes.

As the name BFS suggests, you are required to traverse the graph breadthwise as follows:

- First move horizontally and visit all the nodes of the current layer
- Move to the next layer

DECOMPRESSION:

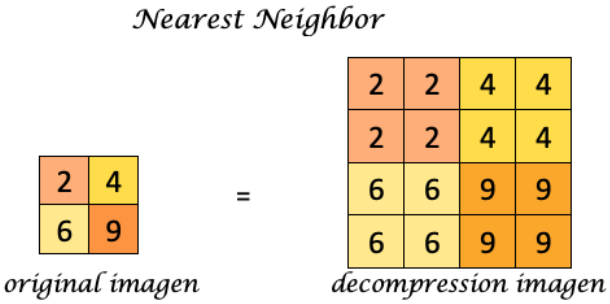


Figure 2: Decompression generated from Nearest Neighbor.

4.2.1 Lossy image-compression algorithm

To achieve image compression, we consider applying the image scaling algorithm together with the nearest neighbor interpolation, for this we use the PIL library, which is used in Python to add, open, manipulate and save different image files. Through it we use the Image module to load images from files and create new images. The process consists of loading the image to later convert it to grayscale, we also consider a factor variable to define the level of decompression that is desired, finally we save the compressed image as a new file.

COMPRESSION:

Figure 3: Compression generated using Image Scaling with Nearest Neighbor.

DECOMPRESSION:

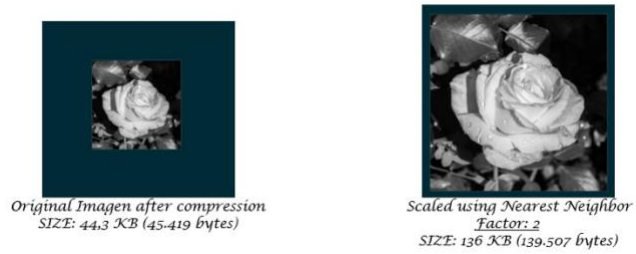
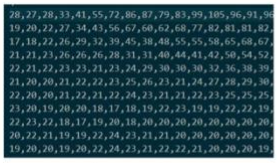


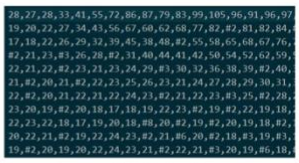
Figure 4: Decompression generated using Image Scaling with Nearest Neighbor and BFS.

4.2.2 Lossless image-compression algorithm

For lossless understanding we implement Run Length encoding. This checks if there is a series of repeated contiguous numbers, replacing with an identifier '#' that will validate the number of repetitions of the value and will separate it, differentiating the number of repetitions from the repeated one. We verify by means of a counter that the repetitions are greater than one so in such case, do not add a 1 before them. These values are later copied into a list of lists, generating a CSV. Finally for decompression we will only have to iterate over them.



after compression



Using Run Length

4.3 Complexity analysis of the algorithms

Thanks to the algorithms traversing the image matrix only once, we obtain a time consumption O (N * M), where N and M are the length of the matrix (Columns and Rows).

Algorithm	Time Complexity
Image Scaling Compression	O(N*M)
Image Scaling Decompression	O(N*M)
Run Length Compression	O(N*M)
Run Length Decompression	O(N*M)

Table 1: Temporal complexity of image compression and decompression algorithms.

Algorithms consume memory similar to the linearity that compose them, therefore in time complexity as in memory complexity, we will have O (N * M), where N and M are the length of the matrix (Columns and Rows).

Algorithm	Memory Complexity
Image Scaling Compression	O(N*M)
Image Scaling Decompression	O(N*M)
Run Length Compression	O(N*M)
Run Length Decompression	O(N*M)

Table 2: Memory Complexity of the image-compression and image-decompression algorithms.

4.4 Design criteria of the algorithm

Thanks to an arduous search during the semester, in front of the possible algorithms that we could implement in the project, we finally opted for the compression and decompression of images with loss of image scaling due to its efficiency and since compared to the process time taken by others and their proper construction and operation, it is easier to implement. In addition, it allows us to adapt the size to suit the user and finally we will obtain a file that is easier to process. For the lossless algorithm we use the Run Length because it is an algorithm that does not use many resources, and its ease of compression and decompression in most computers.

5. RESULTS

5.2 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 6.

	Average execution time (s)	*Average file size (MB)*
Image Scaling Compression	0,5 s	0,227 MB
Image Scaling Decompression	12,2 s	0,227 MB
Run Length Compression	4,9 s	0,227 MB
Run Length Decompression	0,9 s	0,227 MB

Table 6: Execution time of the Image Scaling and Run Length algorithms for different images in the data set.

5.3 Memory consumption

We present memory consumption of the compression and decompression algorithms in Table 7.

	Average memory consumption (MB)	*Average file size (MB)*
Image Scaling Compression	6,7 MB	0,227 MB

Image Scaling Decompression	10,9 MB	0,227 MB
Run Length Compression	40,3 MB	0,227 MB
Run Length Decompression	6,3 MB	0,227 MB

Table 7: Average Memory consumption of all the images in the data set for both compression and decompression.

5.3 Compression ratio

We present the average compression ratio of the compression algorithm in Table 8.

	Healthy Cattle	***Sick Cattle***
Average compression ratio	11:0,1	11:0,1

Table 8: Rounded Average Compression Ratio of all the images of Healthy Cattle and Sick Cattle.

6. DISCUSSION OF THE RESULTS

We consider that the results are appropriate due to the relationship between the time it takes to process each image and the memory consumption, emphasizing that it is an algorithm that consumes few resources and therefore computationally economical. It is a good relationship because when applying the algorithm, the files significantly decrease their size giving a satisfactory understanding.
The results would vary according to the size reduction factor in each image; however, it is an algorithm that will fulfill its promise in different conditions and will complement the classification of animal health in the context of PLF.

6.1 Future work

In the future we consider the use of the discrete cosine transformation for the lossy algorithm, due to its greater optimization in the different compression and decompression rates of images. This will keep the most accurate points. For the lossless algorithm we conceptualize on the LZ77 and LZ78 which, unlike the Run Length, will not look for immediately continuous repetitions, this will provide us with greater efficiency.

ACKNOWLEDGEMENTS

N/A

REFERENCES

LZSS - Wikipedia, la enciclopedia libre

LZSS: 2021. *https://es.m.wikipedia.org/wiki/LZSS*. Accessed: 2021- 02- 09.

LZSS (LZ77) Discussion and Implementation (dipperstein.com)

Dipperstein, M. LZSS (LZ77) Discussion and Implementation: 2015. *http://michael.dipperstein.com/lzss/*. Accessed: 2021- 02- 09.

https://www.youtube.com/watch?v=hEy2bvWO6gk&feature=emb_title

Ivannikov, G.19. LZSS encoding and decoding with examples.:
2019. *https://www.youtube.com/watch?v=hEy2bvWO6gk&feature=emb_title*. Accessed: 2021- 02- 09.

https://michaeldipperstein.github.io/lzss.html

Dipperstein, M.LZSS (LZ77) Discussion and Implementation:
2018. *https://michaeldipperstein.github.io/lzss.html*.
Accessed: 2021- 02- 09.

https://es.wikipedia.org/wiki/Compresi%C3%B3n_fractal

Compresión fractal:
2019. *https://es.wikipedia.org/wiki/Compresi%C3%B3n_fractal*. Accessed: 2021- 02- 08.

Compresion_Fractal.pdf (unam.mx)

Perez-Becker, S. 2021. *Compresion Fractal de Imagenes*. Facultad de Ciencias, Universidad Nacional Autonoma de Mexico. Accessed: 2021- 02- 08.

Compresión Fractal de imágenes | Computación Científica (wordpress.com)

Compresión Fractal de imágenes:
2009. *https://scicomp.wordpress.com/2009/11/19/22/*.
Accessed: 2021- 02- 08.

https://sites.google.com/site/ticslearn/Transformada_discreta_del_Coseno

Transformada discreta del Coseno - Curso de algoritmia:
2015. *https://sites.google.com/site/ticslearn/Transformada_discreta_del_Coseno*. Accessed: 2021- 02- 07.

https://maixx.files.wordpress.com/2012/10/cap12_dct_v01_01_03.pdf

Ibarra Carrillo, M. 2012. *Capítulo 02. Transformada Coseno Discreta*. Accessed: 2021- 02- 07

https://signalsprocessingup.files.wordpress.com/2011/12/dct.pdf

TRANSFORMADA DISCRETA DEL COSENO (DCT). 2021. *Signalsprocessingup.files.wordpress.com*. https://signalsprocessingup.files.wordpress.com/2011/12/dct.pdf. Accessed: 2021- 02- 07

Image Compressing using Discrete Cosine Transform in Matlab- Part 1 - YouTube

Agarwal, R. 2015. Image Compressing using Discrete Cosine Transform in Matlab- Part 1. *Youtube*. https://www.youtube.com/watch?v=UU0tLHsMaOA. Accessed: 2021- 02- 07

Image Scaling - Wikipedia, the free encyclopedia

Image Scaling - Wikipedia, la enciclopedia libre: 2021. *https://en.wikipedia.org/wiki/Image_scaling* Accessed: 2021- 02- 07.

Huffman Coding - Wikipedia, the free encyclopedia: 2021. *https://en.wikipedia.org/wiki/Huffman_coding* Accessed: 2021- 02- 07.

LZ77 and LZ78 – Wikipedia, the free encyclopedia: 2021. *https://en.wikipedia.org/wiki/LZ77_and_LZ78* Accessed: 2021- 02- 07.

LZ77 and LZ78 - Slideshare: 2018. *https://www.slideshare.net/MustafaGokce/lz77-and-lz78-compression-algorithms* Accessed: 2021- 02- 11.

Burrows Wheeler transform – Wikipedia, the free encyclopedia:
2021. *https://en.wikipedia.org/wiki/Burrows%E2%80%93Wheeler_transform* Accessed: 2021- 02- 07.

Burrows Wheeler Transform - YouTube: 2014. *https://www.youtube.com/watch?v=4n7NPk5lwbI* Accessed: 2021- 02- 11.

Yang F, Tschetter E, Léauté X, Ray N, Merlino G, Ganguli D (2014) A real-time analytical data store. SIGMOD'14, June 22–27, 2014, Snowbird, UT, USA. ACM 978-1-4503-2376-5$414/06.

Min Chen, Wei Li, Giancarlo Fortino, Yixue Hao, Long Hu, and Iztok Humar. A dynamic service migration mechanism in edge cognitive computing. ACM Transactions on Internet Technology, 19(2):30:1–30:15, April 2019.

Makinde, A., Islam, M.M., Scott, S.D., 2019. Opportunities for ACI in PLF: Applying animal- And user-centred design to precision livestock farming. In: ACM International Conference Proceeding Series. Association for Computing Machinery.

Yang, Q., Liu, Y., Chen, T., Tong, Y., 2019. Federated machine learning: Concept and applications. ACM Trans. Intell. Syst. Technol. 10 (2).