**Image compression algorithms for the livestock industry**

|  |  |  |  |
| --- | --- | --- | --- |
| Esteban Bernal Correa  Universidad Eafit  Colombia  ebernalc@eafit.edu.co | Juan Manuel Garzon  Universidad Eafit  Colombia jmgarzonv@eafit.edu.co | Simón Marín Universidad Eafit Colombia smaring1@eafit.edu.co | Mauricio Toro  Universidad Eafit  Colombia  mtorobe@eafit.edu.co |

**Summary**

In this project we are trying to tackle is finding an alternate solution to uploading the farmers files to the algorithm with low computing power, as the availability of internet and advanced computing on farms is very low. This problem is important as it could help to automate the processes of identifying unhealthy cattle with a machine, which to this day, had never been done correctly due to the poor technology accessibility. Furthermore, this will help digitize and automatize information that was previously stored in unconventional ways, such as printed photographs on notebooks.

## **Keywords**

|  |
| --- |
| Compression algorithms, machine learning,  deep learning, precision husbandry, animal health. |

**1.Introduction**

Since technology revolutionized the world and all of its industries, there have been certain sectors of the economy that couldn't keep up. As cattle raising is mostly located on rural areas, it has been harder for them to have the top-notch technology at their advantage. In this specific situation, we are facing cattle raisers with low storage capacity and long upload times. As these farmers are mostly using images to document their cattle, compressing these images will not only allow for more images to be stored, but will also shorten the upload time to the main algorithm.

**1.1** Problem

The problem we’ll be solving is the creation of an algorithm that has the ability to compress and decompress images (on the quickest time possible), via hash tables and other methods. To measure the success of this compression / decompression, the images must go through a AI algorithm with less than 5% margin of error.

**1.2 Solution**

In this work, we use a convolutional neural network to classify animal health, in livestock, in the context of precision agriculture (GdP). A common problem in the GoP is that the network infrastructure is very limited, so data compression is required.

The algorithm that we are going to use is called the nearest neighbor, we chose this algorithm because it is related to games and it caught our attention to understand how it works

**2. RELATED JOBS**

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

**2.1** Hash tables

Hash tables are data structures that are used to store large amounts of data that later need to be searched for and inserted into a very efficient file system.

A hash table stores a set of "(key, value)" pairs. The key is unique for each element of the table and is the data used to find a certain value.

The implementation of a hash table is based on the following elements:

• A reasonably sized table to store the pairs (key, value).

• A “hash” function that receives the key and returns an index to access a position in the table

• A procedure to deal with cases where the above function returns the same index for two different keys.

**2.2** Tree B +

It is a tree-like data structure that represents a set of ordered data and that also allows the efficient implementation and elimination of the elements found in the set.

All information is saved on the sheets. Internal nodes only contain keys and pointers. All the leaves are at the same level, which corresponds to the lowest. The leaf nodes are linked together as a linked list to primarily allow range retrieval by searching sequences.

**2.3** Red-Black Tree

The red-black tree consists of a binary search tree, which means that it groups the values ​​into two groups, on the left the smallest and on the right the largest, but the difference between this and other binary trees is that it is It incorporates an extra bit of storage per node, which allows determining if a node is red or black according to the properties of this structure. With the help of these properties and the color of each node, an order of the data is determined that changes as new information is added, so that the tree is balanced and shortens the path when looking for certain data.

## 

## **2.4** Cattle monitoring via collar

The main concern is the wellbeing of the animal we are eating, given that the consumption of “unhealthy meat” can often cause diseases. In this case, the solution was to monitor the cows via a collar. This was built with an accelerometer and a gyroscope for activity recognition, also with the capacity of geolocation. This meant that with this single collar (which gave all the information in real time), most of the cows' behaviors could be monitored, such as: the cows movement, eating routines and even the proximity to other cows. So, through deep learning, the algorithm warns the user when a cow if showing unfamiliar behaviors (not much food consumption, little to no movement and isolation from other cows) and also gives them the exact location.  This is all given thought an application installed on the user's mobile device or pc, where they can also access all of the data they want, whenever they want.

## **3. Materials and methods**

In this section, we explain how the data was collected and processed, and then different alternative image compression algorithms to improve the classification of animal health.

**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 sets of images were grayscale using Python OpenCV and transformed into comma-separated value (CSV) files. The data sets were balanced.

## The data set was divided into 70% for training and 30% for testing. The data sets are available at

## https://github.com/mauriciotoro/ST0245-Eafit/tree/master/project/datasets.

## Finally, using the training dataset, we trained a convolutional neural network for binary image classification using Google's Teachable Machine available at <https://teachablemachine.withgoogle.com/train/image>.

**3.2 Lossy data compression algorithms**

**3.2.1** Fractal compression

Is a lossy compression method for digital images, based on fractals. The method is best suited for natural images and textures, a property of which parts of an image often resemble other parts of the same image. Fractal algorithms make these parts into mathematical data called "fractal codes" which are used to recreate the encoded image

**3.2.2** Imaging Escalade

Consists of dividing a digital image into several groups of pixels called segments. Segmentation is a pixel classification process that assigns a category to each pixel in the analyzed image. This general problem is divided into specialized problems, leading for example to:

• segmentation by color

• segmentation by textures

• superpixel

• semantic segmentation

**3.2.3** Wavelet transformation discreet

decomposes an image into a set of small successively orthonormal images. On the other hand, while the gray-level histogram of the original image can be of any shape, those reflections of the wavelet transforms are only unimodal and symmetric near zero.

This simplifies the statistical analysis of the signal properties.

**3.2.4** Discrete cosine transformation

An image is split where each pixel or point of the image is formed by an intensity of red, another of green, and another of blue, so the first thing the algorithm does is transform that way of representing the image, to another in the We They have two color channels and one brightness channel, then for each component of the image it is divided into small blocks of 8 × 8 pixels, which are processed almost independently, significantly reducing the computation time. This results in the typical grid pattern, which becomes visible in images saved with high compression.

**3.3 Lossless data compression algorithms**

**3.3.1** Burrows-Wheeler transform

The Burrows–Wheeler transformation (BTW) main purpose is to rearrange a strings characters to form patters of the same character, this way, it's easier to compress a file with repeated characters. BTW is special because this transformation does not change any character, which means it doesn’t have to store any additional data. Furthermore, the transformation is reversible, and it only needs the position of the first original character to give you back the original string.

So how does it work? The algorithm starts by permutating the string into every combination possible, without changing the adjacent characters (without taking into account the first and last character). When all of the permutations are completed, they are sorted into alphabetical order, and the last character of each permutation is taken to form the new string. The wort case performance for this algorithm is O(n).

**3.3.2** LZ77

Referred to by many as the “king of compressors”, and deriving a whole family of compressors, LZ77 is a compressor developed by Jacob Ziv and Abraham Lempel in the 70s. The algorithm works by splitting the string into 2 segments: The left side called the search buffer (SB), which contains the symbols we’ve already seen and already process. The right side of the string is called the lookahead buffer (LA), which contains the symbols we haven't seen. The encoder will read a symbol from the LA and will try to search for a match in the SB, if it's found, the encoder will read more symbols in the LA and search backwards in the SB until it finds the longest match. When the longest match is found, a token will be generated. This token is made up of 3 parts: an offset (distance from LA symbols to SB match), the length (number of symbols matched) and the next symbol in the LA. Once this is done, the string will be splitted again, just only after the symbols we just analyzed.

**3.3.3** Huffman coding

The compression algorithm developed by David A. Huffman, is the basic foundation to what we know today as modern-day text compression algorithms. For the algorithm to work, it first needs to make a list of how frequent each character is, and from there, you will begin to build what's called a Huffman tree. To do this correctly, you'll start by takin the least 2 frequent characters and grouping them up (in some sort of branch) and summing their frequencies. This new branch with the sum of the frequencies will go up the list according to the result of the sum. Repeat this process until you have the whole Huffman tree.

After you’ve done this, to convert the text into 1’s and 0’s, all you have to do is go to the top of the tree and each time you take a left on the tree you write a 0, each time you take a right you write a 1. This is able to compress text ass the more common characters are based on the top of the tree, meaning it takes less bits to store the most repeated characters. The only fault this system has, is the fact that it must store the Huffman tree, making it inefficient for short text compressions.

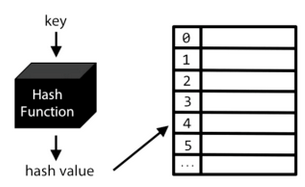
**3.3.4 LZS**

LZS compression utilizes an LZ77 type of calculation mixed with some of Huffman's coding principles. It utilizes the last 2 KB of uncompressed information as a sliding-window for word reference. The way LZS compression works, it searches for matches between the information to be compacted and the last 2 KB of data. In the event that it finds a match, it encodes an offset/length reference to the word reference. On the off chance that no match is discovered, the next byte is encoded as a "literal" byte. The compacted information stream closes with an end-marker.

**4. Data structures**

The data structure that we are going to use is the hash tables, which relate keys to value since access to data is usually very fast.

The way this works is that a hash value is first calculated from the data of a record. The hash values ​​of all the records in a database are stored in the hash table. Through another mathematical operation, the location of said information in the Data Structure is calculated from the hash value. If the user then enters a term in the search field, this term is also hashed.

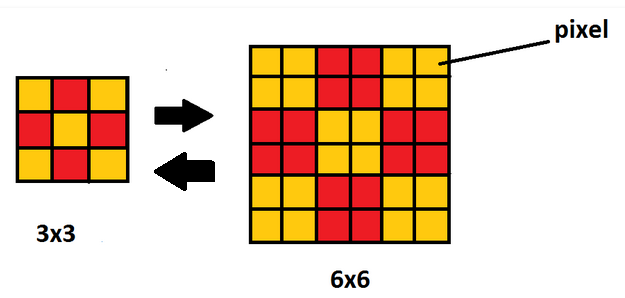


**4.2 Algorithm**

**4.2.1 Lossy image compression algorithm**

The algorithm to compress the image receives two parameters which are: the image to compress (which is in an array) and how much it will be compressed (through an int called rateOfDownscale). The new array will be created first, the dimension of which is calculated by dividing the original size by the value stored as rateOfDownscale. Then the algorithm runs through the given matrix and generates an average of a square matrix of the size of the saved value (it should be noted that no matrix is ​​generated so as not to waste space) this average is saved in the first position of our new matrix, that is, it will average a square of pixels to convert it to one. This is repeated until the entire given matrix has been traversed by these square matrices and all the pixels have been compressed.

For the decompression of the image, this algorithm will receive two similar parameters: the image to be compressed (which is in a matrix), and how much this image will be enlarged (by means of an int called rateOfUpscale). This algorithm will also start by creating a new matrix, this dimension is calculated by multiplying the original size and the rateOfUpscale variable. By having our new matrix, simply to fill it we will go through the old matrix and for each position that we go through it generates a square matrix of the size of the saved value (it should be clarified that no matrix is ​​generated so as not to waste space) and it will be placed in the respective position, that is, this will convert one pixel into several pixels that generate a "square". This is repeated until the entire given matrix has been traversed, and all the pixels have been decompressed.



**4.3 Analysis of the complexity of the algorithms.**

|  |  |
| --- | --- |
| **RLE** | **The complexity of time** |
| **Compression** | O(N^2) |
| **Decompression** | O(N^2) |

|  |  |
| --- | --- |
| **RLE** | **Memory complexity** |
| **Compression** | O(N\*M ) |
| **Decompression** | O(M\*N) |

**4.4 Algorithm design criteria**

The RLE algorithm was designed in this way because when we realized that many pixels were repeated and together with the other 2 algorithms developed and compared, the RLE is more efficient in memory consumption and time for images in which it is very common to find equal pixel lines

**5.1 Compression rate**

|  |  |  |
| --- | --- | --- |
|  | Healthy cattle | Sick cattle |
| Average compression rate | 20:12 | 20:12 |

## **5.2 Execution times |**

## In the following, we explain the relationship between the average execution time and the average size of the images of the complete data set.

|  |  |  |
| --- | --- | --- |
|  | ***Average execution time (s)*** | ***Average file size (MB)*** |
| *Compression* | 0.54 s | 0.145 MB |
| *Decompression* | 0.54 s | 0.145 MB |

**5.3 Final conclusion**

The algorithm that is most efficient to compress images in which it is very common to find lines of equal pixels is the RLE since it is very efficient in memory and time consumption compared to the other algorithms developed.

**6 Future work**

Develop algorithms more efficiently in relation to memory consumption and time consumption and achieve that an image can be compressed first with an algorithm with loss and then without loss

# **REFERENCES**

1. Stalin, F. Árbol rojo negro la evolución de los árboles binarios de búsqueda. Retrieved August 10, 2017, from Computer science in esmeraldas. <http://computerscienceesmeraldas.blogspot.com.co/2016/03/>

2. Universidad Carlos III de Madrid. Tablas hash. Retrieved August 12, 2017, from Departamento de ingeniería telemática. http://www.it.uc3m.es/abel/as/MMC/M2/HashTable\_es.html.

3. Wikipedia. Árbol-B. Retrieved August 11, 2017. https://es.wikipedia.org/wiki/Árbol-B.

4. Wikipedia. Árbol B+. Retrieved August 11, 2017. https://es.wikipedia.org/wiki/Árbol\_B%2B.

5. Solis, S. Árboles B. Retrieved August 12, 2017, from youtube https://

6. <https://github.com/mauriciotoro/ST0245-Eafit/blob/master/proyecto/problemas-relacionados/An%20Animal%20Welfare%20Platform%20for%20Extensive%20LivestockProduction%20Systems.pdf>

7.http://www.scielo.org.mx/scielo.php?script=sci\_arttext&pid=S1405-77432006000300005

8.https://es.wikipedia.org/wiki/Compresi%C3%B3n\_fractal

9. https://www.youtube.com/watch?v=y3xSuPDvpOE

10.https://compilandoconocimiento.com/2017/01/19/tablas-hash/