COMPRESIÓN DE IMÁGENES PARA ALGORITMO DE DIAGNÓSTICO DE GANADO

Humberto Carbonó	Valentina Ortega	Julian Caro	Simón Marín	Mauricio Toro
Universidad Eafit	Universidad Eafit	Universidad Eafit	Universidad Eafit	Universidad Eafit
Colombia	Colombia	Colombia	Colombia	Colombia
hacarbonop@eafit.edu.co	vortegav@eafit.edu.co	ajcaror@eafit.edu.co	smaring1@eafit.edu.co	mtorobe@eafit.edu.co

RESUMEN

Durante años la industria ganadera se ha visto afectada por las distintas enfermedades que contrae el ganado trayendo como consecuencia la muerte, y con esta, grandes pérdidas para este sector, o, en caso de desconocerse la condición de salud, la comercialización de carne, causando así posibles enfermedades a sus consumidores. Generalmente estas enfermedades no se pueden identificar con facilidad y es complicado para los ganaderos acceder a un diagnóstico de salud de sus animales. Es importante resolver esta problemática para que los ganaderos puedan llevar un control continuo de la salud del ganado y garantizar así el correcto desarrollo del sector y la salubridad pública.

Para la solución utilizamos 2 algoritmos: el vecino más cercano y LZ77. El vecino más cercano es un algoritmo de compresión de imágenes con pérdidas que selecciona un pixel y elimina los que están arriba, abajo y a los lados. Su tasa de compresión es de 2:1 ya que elimina la mitad de las filas y la mitad de las columnas. El LZ77 es un algoritmo de compresión sin pérdidas que tiene una ventana con un tamaño elegido dependiendo de qué tanto queramos comprimir la imagen (entre mayor sea el tamaño de la ventana, más se comprime). Se puede decir que ambos algoritmos fueron muy efectivos y se usarían dependiendo de si la persona está dispuesta o no a perder calidad de la imagen y el tiempo que requiera. Lo anterior se debe a que, aunque el consumo de memoria de ambos algoritmos es casi igual, el tiempo de ejecución no lo es. El tiempo promedio del algoritmo del vecino es de 2,16s, mientras que el tiempo promedio del LZ77 es de 10,66z y esto con un tamaño de ventana de solo 13 y si quisiéramos comprimir más llevaría mucho más tiempo.

Palabras clave

Algoritmos de compresión, aprendizaje de máquina, aprendizaje profundo, ganadería de precisión, salud animal.

1. INTRODUCCIÓN

La ganadería se ha visto frecuentemente afectada por las enfermedades que acechan al ganado y que, usualmente, no se pueden detectar a simple vista. Lo anterior genera pérdidas a nivel económico y puede representar un riesgo a la salud pública.

Se ha desarrollado un algoritmo que por medio de una imagen del ganado logra analizar y dar a conocer el estado de salud del animal. Sin embargo, esta solución trae consigo otro problema: poner este programa en marcha acarrea un gran consumo de datos por el tamaño de la imagen, lo cual dificulta la utilización de este, ya que en la zonas ganaderas existe poca cobertura para la ejecución de este proceso.

Por lo anterior se ha pensado en desarrollar un algoritmo que sea útil para compresión de la imágenes de tal manera que este sea capaz de leerlas de manera exitosa, con el mínimo de recursos y de manera eficaz.

1.1. Problema

El algoritmo que se pensó en un principio para el diagnóstico de la salud del ganado es muy útil para los ganaderos ya que les permite llevar un control de salud constante y la identificación oportuna de las enfermedades para que sea posible tratarlas a tiempo.

Sin embargo, resulta ser un programa muy difícil de ejecutar en un escenario real, ya que en las zonas ganaderas generalmente no hay una buena cobertura que permita el buen funcionamiento del algoritmo de clasificación debido a que este demanda un gran consumo de datos por el tamaño de la imagen a procesar.

Necesitamos pensar ahora en un algoritmo que nos permita comprimir al máximo la imagen de tal manera que esta pueda ser leída correctamente y el programa no consuma tantos datos a la hora de ser ejecutado.

1.2 Solución

En este trabajo, utilizamos una red neuronal convolucional para clasificar la salud animal, en el ganado vacuno, en el contexto de la ganadería de precisión (GdP). Un problema común en la GdP es que la infraestructura de la red es muy limitada, por lo que se requiere la comprensión de los datos.

Para la solución a esta problemática implementamos el algoritmo de nearest, que consiste en seleccionar cuatro(4) megapixels y comprimirlos en uno solo. Escogimos este método ya que a la hora de ejecutarlo tarda aproximadamente 0,5 segundos en finalizar y reduce el tamaño de la imagen a la mitad, lo que lo hace muy eficiente. Adicionalmente, adecuamos el algoritmo para que sin importar el tamaño de la imagen, reduzca la misma a 480mp y así, todas las imágenes serán lo menos pesadas

posibles y será posible analizarlas con una red no muy buena

1.3 Estructura del artículo

En lo que sigue, en la Sección 2, presentamos trabajos relacionados con el problema. Más adelante, en la sección 3, presentamos los conjuntos de datos y los métodos utilizados en esta investigación. En la Sección 4, presentamos el diseño del algoritmo. Después, en la Sección 5, presentamos los resultados. Finalmente, en la Sección 6, discutimos los resultados y proponemos algunas direcciones de trabajo futuras.

2. TRABAJOS RELACIONADOS

En lo que sigue, explicamos cuatro trabajos relacionados. en el dominio de la clasificación de la salud animal y la compresión de datos. en el contexto del PLF.

2.1 Monitoreo automatizado de la salud animal

IoT based Animal Health Monitoring with Naive Bayes Classification

En este trabajo el problema a solucionar fue la salud del cuerpo humano. “nuestra vida depende del funcionamiento de nuestra salud, ya que es lo más importante para nuestro cuerpo”. Se plantea que una de las razones por la que nuestra salud no está bien son las enfermedades en el ganado como fiebre, cetosis, etc .
Para lo anterior se propone un sistema de monitoreo de salud animal (AHM) que ayuda a los agricultores de forma eficaz y asequible. El sistema es automatizado, libre de interferencia humana y el usuario lo puede controlar desde una ubicación remota. En caso de alguna anomalía en la salud del animal recibirá una alerta a través de su teléfono móvil. El sistema consta de nodeMCU, microcontrolador de temperatura corporal animal, humedad, frecuencia cardíaca y sensores de celda de carga. Para análisis y clasificación de datos se implementa el algoritmo Naive Bayes. [4]

2.2 Técnicas para la correcta compresión de imágenes

En este artículo se brinda información acerca de las técnicas y formatos más utilizados para la compresión de imágenes lo cual nos ayuda a darle una solución al problema de la utilización de datos al momento de la ejecución del programa. [2]

2.3 Sistema de monitoreo con sensores de las condiciones de salud del ganado

Se presenta un sistema muy completo que permite la evaluación de las condiciones de salud del ganado por medio de distintos sensores encargados de verificar varias partes del animal y dar un diagnóstico completo.

Este sistema está basado en la tecnología de comunicación inalámbrica de bajo consumo, Zigbee. Esto es de gran relevancia para el problema que estamos tratando de solucionar.[1]

2.4 Esquema híbrido DWT-DCT para compresión de imágenes médicas

En este artículo se propone como foco la importancia de la imagenología médica en la atención médica contemporánea, ya que esta sirve para un diagnóstico primario y como guía para procedimientos quirúrgicos y terapéuticos. El problema que hay en esto es que la cantidad de datos generados por los dispositivos de imagen actuales son muy grandes y están en constante aumento, por lo que se requiere formas eficientes para comprimir y codificar esta información. Lo que propone el artículo es un esquema híbrido para la compresión de imágenes médicas de diferentes modalidades de imagen. Como los detalles de DWT generalmente tienen una media cero y una pequeña variación, la compresión basada en DCT se aplica para lograr una compresión más alta mientras se conserva la información de diagnóstico importante. [3]

El algoritmo utilizado en esta ocasión fue el siguiente:

classify_block (): clasifica los bloques y establece el límite del área de distorsión permitida

Entrada: imagen secundaria de tamaño $N \times N$

Salida: límite del área de distorsión (γ)

Paso 1: Calcule el umbral (Th) para la clasificación de bloques.

Paso 2: Decide la clase de los bloques (puros o complicados)

si la varianza del bloque (v)> Th

γ = pequeño_valor

demás

γ = valor_grande

Fin

Paso 3: Devuelve γ.

3. MATERIALES Y MÉTODOS

En esta sección, explicamos cómo se recogieron y procesaron los datos y, después, diferentes alternativas de algoritmos de compresión de imágenes para mejorar la clasificación de la salud animal.

3.1 Recopilación y procesamiento de datos

Recogimos datos de *Google Images* y *Bing Images* divididos en dos grupos: ganado sano y ganado enfermo.

Para el ganado sano, la cadena de búsqueda era "cow". Para el ganado enfermo, la cadena de búsqueda era "cow + sick".

En el siguiente paso, ambos grupos de imágenes fueron transformadas a escala de grises usando Python OpenCV y fueron transformadas en archivos de valores separados por comas (en inglés, CSV). Los conjuntos de datos estaban equilibrados.

El conjunto de datos se dividió en un 70% para entrenamiento y un 30% para pruebas. Los conjuntos de datos están disponibles en https://github.com/mauriciotoro/ST0245-Eafit/tree/master/proyecto/datasets .

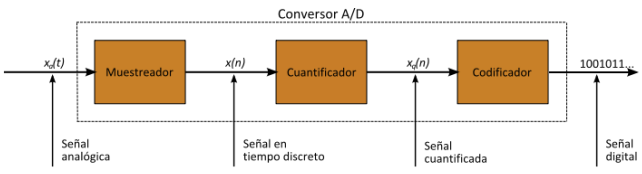
Por último, utilizando el conjunto de datos de entrenamiento, entrenamos una red neuronal convolucional para la clasificación binaria de imágenes utilizando *Teachable Machine* de Google disponible en https://teachablemachine.withgoogle.com/train/image.

En lo que sigue, presentamos diferentes algoritmos usados para comprimir imágenes con pérdida.

3.2.1 Codificación por transformación

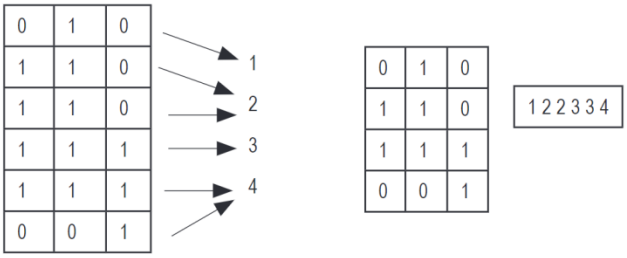
Este algoritmo permite crear una imagen con un número de datos reducido. Generalmente se utiliza la transformada discreta de coseno ya que empaqueta la mayor parte de la información en el menor número de coeficientes.

En la codificación por transformación, el conocimiento de la aplicación se utiliza para elegir la información a descartar para, de esa forma, disminuir su ancho de banda.[2]



3.2.2 Vector de cuantización

Este algoritmo divide la imagen en bloques de tamaño fijo y los almacena en una tabla. Luego el algoritmo elimina los bloques repetidos de la tabla quedándose únicamente con vectores diferentes de la imagen original.[2]



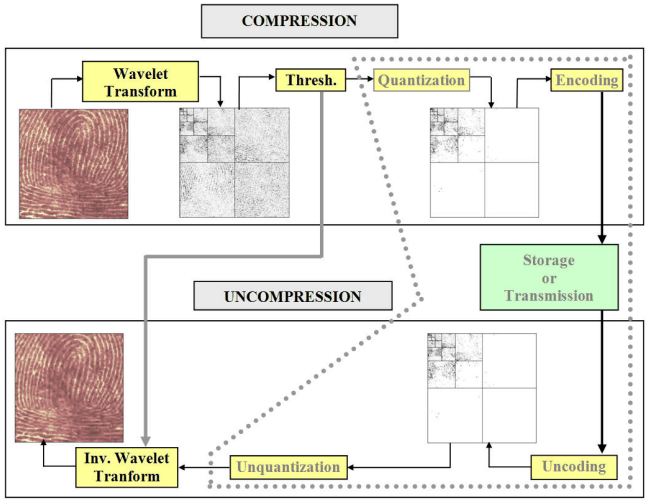
3.2.3 Compresión fractal

La compresión fractal consiste en representar diversas imágenes, que presentan una repetición de su estructura a diferentes escalas, con una función que permite definir la forma en la cual se crea la imagen. Esto hace posible que se pueda transmitir únicamente el coeficiente que identifica a la función hallada. [2]

3.2.4 Compresión wavelet

Este algoritmo es una variación de la transformada de cosenos discreta que utiliza wavelets en lugar del algoritmo basado en bloques.

El algoritmo crea tantos coeficientes como píxeles haya en la imagen y luego se cuantifican. Por último se le aplica codificación sin pérdida a los datos cuantificados y da como resultado la imagen comprimida.

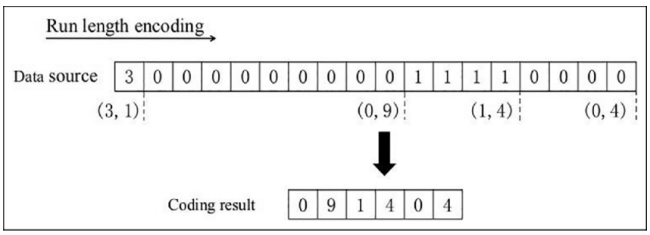


3.3 Alternativas de compresión de imágenes sin pérdida

En lo que sigue, presentamos diferentes algoritmos usados para comprimir imágenes sin pérdida.

3.3.1 Run-length encoding (RLE)

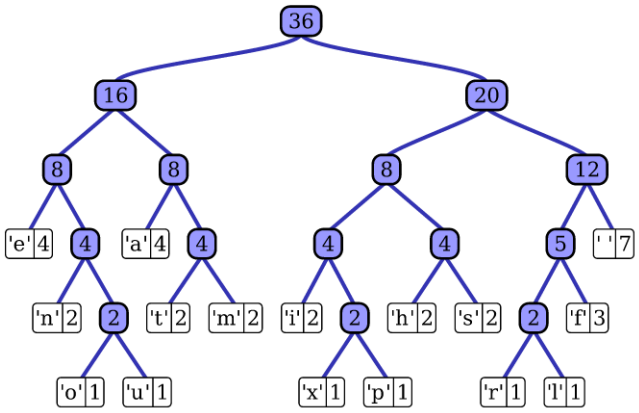
Es un método de compresión muy simple, y es muy útil en imágenes en las cuales se repiten algunas secuencias de caracteres. Este algoritmo consiste en almacenar el número de caracteres repetidos con el carácter a su lado.[2]



3.3.2 Codificación de Huffman

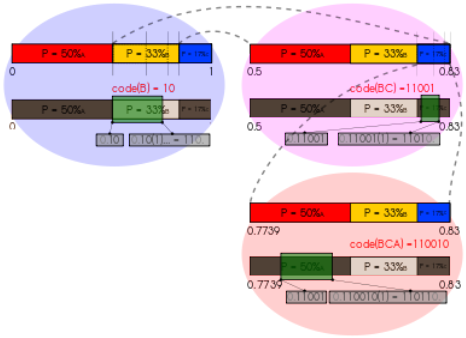
Es una técnica que consiste en asignarle código de bits más cortos a los datos que mayor frecuencia de aparición tienen y códigos más largos a los que aparecen con menos regularidad.

Este algoritmo consiste en la creación de un árbol binario desde abajo hacia arriba.[2]



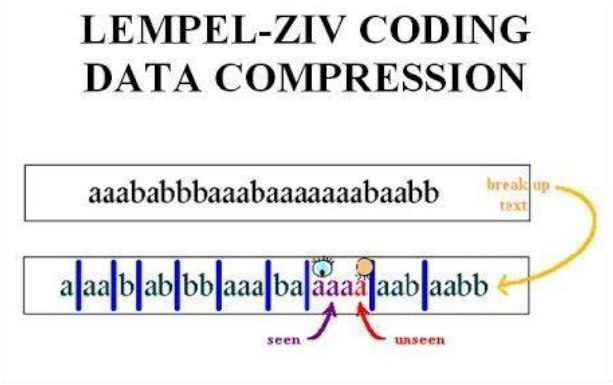
3.3.3 Codificación aritmética

El algoritmo codifica una secuencia de símbolos representándolos de manera binaria. Cuando se convierte una secuencia de caracteres a codificación aritmética, se almacenan los caracteres más usados con menos bits y los que aparecen con menos frecuencia se almacenan con más bits, de esta manera se obtiene un número total de bits menor. La codificación aritmética codifica el mensaje entero a un sólo número real entre 0 y 1. [2]



3.3.4 Lempel-Ziv

Este algoritmo busca secuencias repetidas dentro de los datos, y cada vez que encuentra una de ellas la reemplaza por un puntero a la zona en la que comienza la primera secuencia, más la longitud que se debe tomar a partir de esa posición. En caso de que no haya repeticiones, se emite la secuencia como un literal.[2]



4. DISEÑO E IMPLEMENTACIÓN DE LOS ALGORITMOS

En lo que sigue, explicamos las estructuras de datos y los algoritmos utilizados en este trabajo. Las implementaciones de las estructuras de datos y los algoritmos están disponibles en Github[1].

4.1 Estructuras de datos

En este caso utilizamos las matrices de la librería numpy, las cuales devuelven objetos similares a una matriz o a una cadena de datos. Las matrices en numpy conservan su naturalidad en 2D, pero puede tener más operaciones que hace que sea más util a la hora de programar.

4.2 Algoritmos

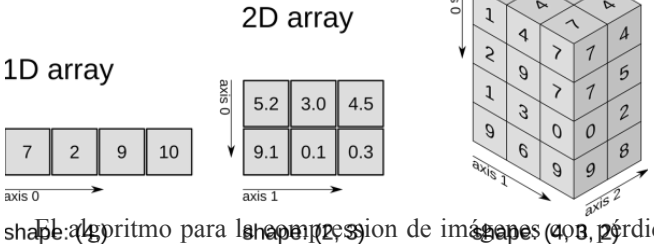
En este trabajo, proponemos un algoritmo de compresión que es una combinación de un algoritmo de compresión de imágenes con pérdidas y un algoritmo de compresión de imágenes sin pérdidas. También explicamos cómo funciona la descompresión para el algoritmo propuesto.

4.2.1 Algoritmo de compresión de imágenes con pérdida

1	4	5	7	3	7	8	9
5	6	8	9	1	2	4	3
6	6	7	9	3	7	1	4
8	7	4	4	5	9	1	4
7	7	4	6	5	1	8	8
9	1	4	7	6	4	5	6

[1]http://ww

1	5	3	8
6	7	3	1
7	4	5	8



El algoritmo para la compresion de imágenes con pérdida escogido fu el algoritmo de K-Nearest. Este algoritmo consiste en reducer el número de Megapíxeles de una imagen, y así su peso, por medio de matrices. Para implementar este algoritmo se debe poner la imagen en una matriz y cada megapixel de esta será estará en una posición diferente. Lo que se hace es que se escoge la primera posición de la matriz y sus vecinos más cercanos, es decir, el de la derecha, el de la izquierda y el que está diagonal a su derecha hacia abajo se “eliminan”. El número seleccionado inicialmente se agrega a una nueva matriz. La matriz se recorre saltando de a 2 posiciones y se aplica el mismo procedimiento en todo el proceso. Finalmente, la matriz resultante será la imagen comprimida a la mitad

4.2.2 Algoritmo de compresión de imágenes sin pérdida

El algoritmo sin pérdidas que se utilizó para la compresión sin pérdidas el algoritmo LZ77 el cual analiza una cadena de caracteres y agrupa los caracteres repetidos en una tripleta la cual contiene como primer elemento hace cuántas posiciones del historial encontró la repetición, el segundo elemento indica cuántos caracteres apareó y el tercero indica qué caracter se incluye en el historial.

El algoritmo fué implementado en python con algunas estructuras de datos como matrices y arreglos dinámicos, buscando darle la mejor eficiencia al código intentamos utilizar las estructuras más adecuadas para mejorar la complejidad como listas enlazadas con complejidad al añadir de O(1), pero fué complicado trabajar el algoritmo con estas estructuras de datos porque imposibilita algunas funciones necesarias.

4.3 Análisis de la complejidad de los algoritmos

Algoritmo	La complejidad del tiempo
Compresión	O(N* M)
Descompresión	O(N*M)

***Tabla* 2:** Complejidad temporal de los algoritmos de compresión y descompresión de imágenes. En el algoritmo N y M representan el número de filas y columna respectivamente.

Algoritmo	Complejidad de la memoria
Compresión	O(N*M)
Descompresión	O(M*N)

***Tabla* 3:** En este caso, N y M representan el número de filas y columnas respectivamente.

4.4 Criterios de diseño del algoritmo

El algoritmo fue diseñado de esa manera debido a que es bastante eficiente, dependiendo eso sí, de diversos factores como lo son el tamaño de la sliding window, el tamaño que le establecemos para el búfer de avance, etc, a los cuales les dimos unos tamaños considerablemente pequeños, de manera que pudiera ser eficiente. Funciona de la misma forma tanto comprimiendo como descomprimiendo, sin embargo, la descompresión acarrea menos tiempo que la misma compresión e incluso que otros algoritmos como lo son el algoritmo de Huffman a la hora de la descompresión.

La decisión de haber elegido este algoritmo de compresión sin pérdidas fue de igual forma por su diseño en sí, el cual es un modelo de codificación basado en diccionario, es decir, se encarga de buscar las coincidencias entre el texto a comprimir y un conjunto de caracteres en una estructura llamada diccionario, el cual es un "historial" en el que vamos añadiendo todos los caracteres vistos (Por esto mismo este es uno de los factores a tener en cuenta dentro de la eficiencia, el tamaño de este diccionario, el cual entre mayor sea, mayor será la compresión pero también será mayor el consumo de memoria).

5. RESULTADOS

5.2 Tiempos de ejecución

En lo que sigue explicamos la relación entre el tiempo promedio de ejecución y el tamaño promedio de las imágenes del conjunto de datos completo, en la Tabla 6.

Calcular el tiempo de ejecución de cada imagen en Github. Informar del tiempo medio de ejecución vs. el tamaño medio del archivo.

	Tiempo promedio de ejecución (s)	***Tamaño promedio del archivo (MB)***
Compresión	23.19s	0.715 MB
Descompresión	0.08s	0.461 MB

Tabla 6: Tiempo de ejecución de los algoritmos *(Por favor, escriba el nombre de los algoritmos, por ejemplo, tallado de costuras y LZ77)* para diferentes imágenes en el conjunto de datos.

5.3 Consumo de memoria

Presentamos el consumo de memoria de los algoritmos de compresión y descompresión en la Tabla 7.

	Consumo promedio de memoria (MB)	***Tamaño promedio del archivo (MB)***
Compresión	84.6 MB	0.715 MB
Descompresión	88.5 MB	0.416 MB

Tabla 7: Consumo promedio de memoria de todas las imágenes del conjunto de datos, tanto para la compresión como para la descompresión.

5.3 Tasa de compresión

Presentamos los resultados de la tasa de compresión del algoritmo en la Tabla 8.

Con un tamaño de ventana de 40 y una imagen de 0.715MB cada una tuvo la misma tasa de descompresión.

	Ganado sano	***Ganado enfermo***
Tasa de compresión promedio	1.72:1	1.72:1

Tabla 8: Promedio redondeado de la tasa de compresión de todas las imágenes de ganado sano y ganado enfermo.

REFERENCIAS

1. A. Kumar, G. Hancke. A Zigbee-Based Animal Health Monitoring System. *IEEE Sensors Journal Volume 15*. 2014.
2. N. La Serna, Mg. Luzmila, C. Durán. Compresión de imágenes: Fundamentos, técnicas y formatos. *Revista de ingeniería de sistemas e informática Vol. 6.* Universidad Nacional Mayor de San Marcos. 2009.
3. S. Singh, V. Kumar y HK Verma Esquema híbrido DWT-DCT para la compresión de imágenes médicas, *Journal of Medical Engineering & Technology*, (2007) 109-122.

4. Tejaswinee A. Shinde, Dr. Jayashree R. Prasad IoT based Animal Health Monitoring with Naive Bayes Classification. *Procedia International Journal on Emerging Trends in Technology (IJETT)*. 2017.