ESTRUCTURA DE DATOS PARA LA CLASIFICACIÓN Y OPTIMIZACIÓN APLICADA A LA GANADERÍA DE PRECISIÓN

María Antonia Velásquez
Universidad EAFIT
Colombia
mavelasqur@eafit.edu.co

Diego Alexander Múnera
Universidad EAFIT
Colombia
damunerat@eafit.edu.co

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

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

RESUMEN

Desarrollar un método óptimo mediante el cual pueda llevarse a cabo el control de calidad y análisis del proceso ganadero de manera precisa y automatizada realizando un estudio de la apariencia del ganado que convertido en fragmentos de texto más detallados, es decir, sometido a descompresión permita la clasificación de este con el más mínimo margen de error

Esta alternativa permitirá un estudio más acertado a la hora de evaluar el proceso ganadero en las granjas para conocer el ganado en buen y mal estado lo cual es indispensable tratándose del consumo humano de este y de crear tal alternativa tomando en cuenta el detalle más mínimo puede acertar en el proceso más que el ojo humano para además optimizar tiempo, materia prima y errores.
Para resolver esta problemática, hemos implementado el algoritmo de Huffman el cual nos permitió realizar el proceso de compresión a bajo costo computacional, de manera simple y efectiva, pues el peso original de cada imagen a comprimir inició siendo de +500kb y con la compresión terminó siendo de un peso de 3kb. El proceso de la compresión de todos los archivos demora en su totalidad un tiempo de 20 minutos, con una tasa de compresión de y una exactitud de reconstrucción casi del 100%

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

1. INTRODUCCIÓN

La actividad ganadera es una de las más básicas e importantes
en el desarrollo de la vida diaria pues de allí provienen productos como leche y carne para el consumo humano.

Debido a esto, es de vital importancia cuidar el estado del ganado durante su desarrollo y proceso de producción que
terminará en el mercado.

Y es por esto que, la implementación de tecnologías inteligentes, la inteligencia artificial y el lenguaje de máquina nos brindaría un estudio minucioso del estado del ganado durante todo su proceso de producción permitiendo además que este proceso dure menos tiempo y tenga menos fallas en su análisis.

1.1. Problema

Comprimir las imágenes de ganado para clasificar la salud animal en el contexto de la ganadería de precisión permitiría que el ganado en mal estado pueda quedar fuera de consideración para la producción y distribución del producto final con un mayor índice de acertación y optimización del tiempo. Con esto a su vez, se evitarían enfermedades en los humanos debido a la carne en mal estado que producirían vacas enfermas que no fueron clasificadas correctamente.

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 compresión de los datos.

Expliquen, brevemente, su solución al problema *(En este semestre, la solución es una implementación de algoritmos de compresión. ¿Qué algoritmos han elegido? ¿Por qué?)*

Para la compresión de imágenes sin perdida, escogimos el algoritmo de Huffman, nuestra elección se debe a que este algoritmo nos permite comprimir nuestro archivo .csv que, en efecto, representa la imagen correspondiente según el

peso de sus caracteres para así tener un costo computacional menor. Es simple y eficiente al momento de descomprimir y retornar de nuevo la imagen.

1.3 Estructura del artículo

En lo que sigue, en la Sección 2, presentamos trabajos relacional con el problema. Más adelante, en la Ssección 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.

Explique cuatro (4) artículos relacionados con el problema descrito en la sección 1.1. Puede encontrar los problemas relacionados en las revistas científicas, en lo posible, en inglés. Considere Google Scholar para su búsqueda. *(En este semestre, el trabajo relacionado es la investigación sobre la clasificación de la salud animal y la compresión de datos, en el contexto de la GdP).*

3.1 Escriba un título para el primer problema

Deberían mencionar el problema que resolvieron, el algoritmo que usaron, la métrica que obtuvieron y la cita de ACM.

3.2 Escriba un título para el segundo problema

Deberían mencionar el problema que resolvieron, el algoritmo que usaron, la métrica que obtuvieron y la cita de ACM.

3.3 Escriba un título para el tercer problema

Deberían mencionar el problema que resolvieron, el algoritmo que usaron, la métrica que obtuvieron y la cita de ACM.

3.4 Escriba un título para el cuarto problema

Deberían mencionar el problema que resolvieron, el algoritmo que usaron, la métrica que obtuvieron y la cita de ACM.

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.

3.2 Alternativas de compresión de imágenes con pérdida

A continuación, serán presentados diferentes algoritmos existentes usados para comprimir imágenes con perdida de algunos parametros para reducir la memoria utilizada.

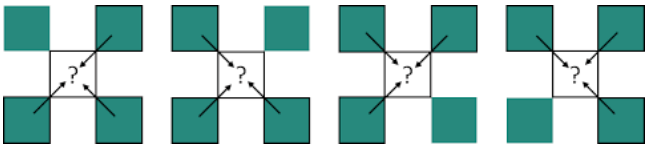
3.2.1 Nearest-neighbor interpolation

Este algoritmo está enfocado en reemplazar cada pixel con el que este más cercano en la salida para aumentar la escala de la imagen. A partir de esto, se introducen irregularidades en las imagenes que fuesen suaves antes de la codificación. La complejidad del algoritmo está enfocada en que el vecino mas cercano no se refiere como tal a este matematicamente sino, un redondeo, haciendo más facil de calcular el tamaño.

3.2.2 Bilinear and bicubic algorithm

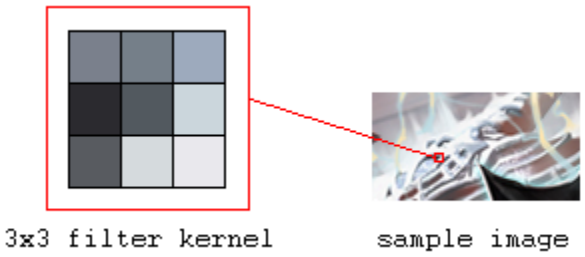
Este algoritmo parte de su funcionalidad interpolando los valores de color en cada pixel, introduciendo una transición continua en la salida. Este algoritmo logra reducir los bordes nitidos de la imagen. La complejidad de este algoritmo surge a partir de sus dos formas de ejecutarse, bilineal y

bicubicamente; una genera la imagen con bordes pixelados y la otra con bordes lisos pero con un costo computacional alto.



3.2.3 Box sampling

Este algoritmo consiste en tomar el pixel destino como una 'caja' en la imagen original y luego mapear todos los pixeles dentro de la caja. Esto con el fin de que al convertir de nuevo la imagen, todos los pixels contribuyan a la salida. La complejidad de ese algoritmo radicó en eliminar los errores o las debilidades de algoritmos como 'Sinc and Lanczos sampling' con su baja optimización de pixeles.



3.2.4 Vectorization

Este algoritmo extrae las representaciones visuals de la resolucion del grafico a comprimir como vectores independientes y luego, esta version se representa como una imagen rasterizada con una resolución optima.

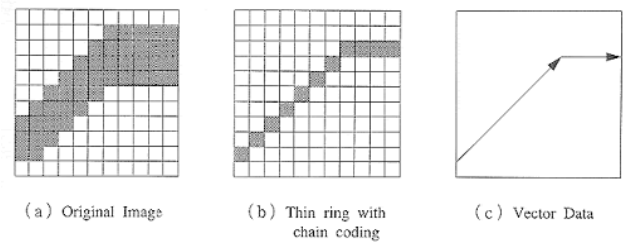


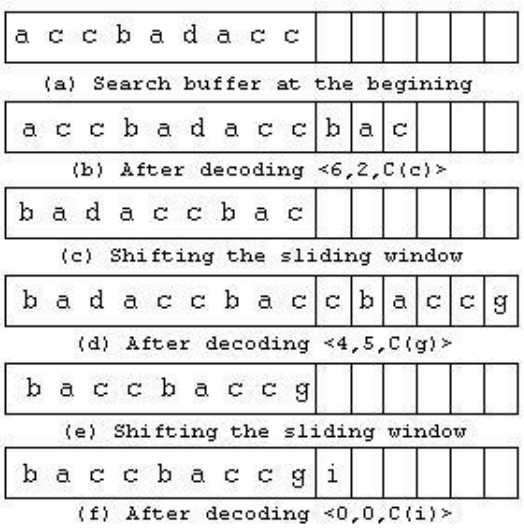
Figure 3.8 A Simple Vectorization Algorithm

3.3 Alternativas de compresión de imágenes sin pérdida

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

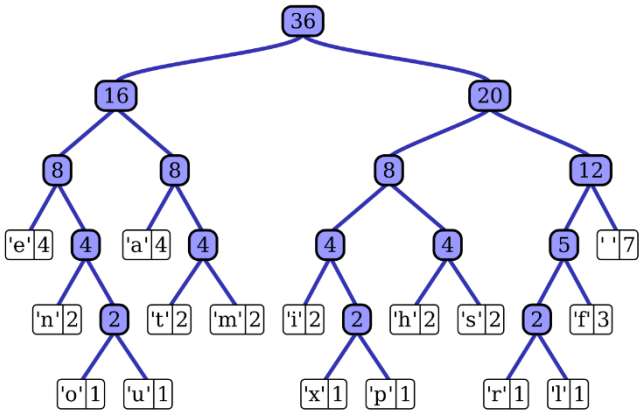
3.3.1 LZ77_and_LZ78

Este algoritmo logra la compresion de imagenes reemplazando las ocurrencias repetidas de datos con referencias en memoria a una sola copia de esos datos existentes anteriormente en el flujo de datos sin comprimir. La complejidad de este algoritmo radica en que para encontrar las coincidencias se codifican dos números que, están llamados longitude-distancia y estos denominan el desplazamiento en el buffer del flujo de datos sin comprimir.



3.3.2 Huffman Compression Algorithm

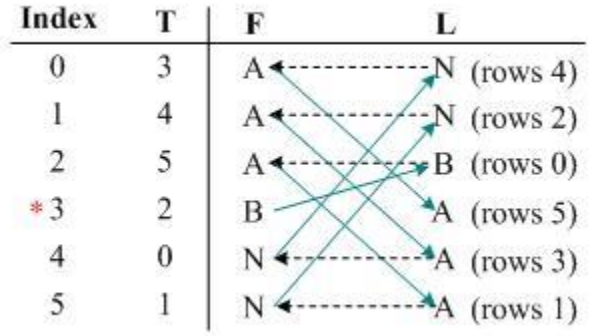
Este algoritmo funciona creando un arbol binario de nodos en los que se puede almacenar un vector regular, cuyo tamaño depende del numero de símbolos n. Cada nodo puede ser un nodo hoja o un nodo interno. En cada uno de estos aparece inicialmente su simbolo y su frecuencia de aparición. Así que, las letras se ordenan aumentando la frecuencia y las menos frecuentes en cada paso se combinan y reinsertan construyendo un arbol parcial.



3.3.3 Burrows Wheeler transform

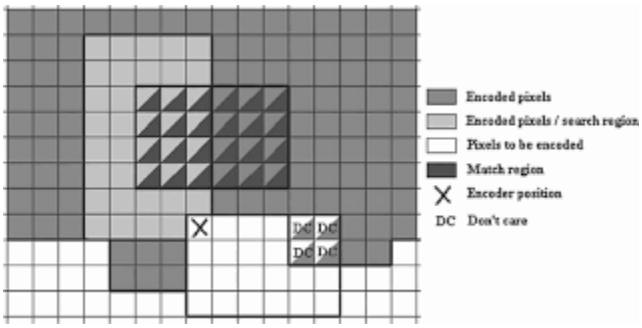
Este algoritmo trabaja con el concepto de compresion por clasificacion de bloques, éste reorganiza una cadena de

caracteres en series de estos mismos similares. Este tipo de algoritmo es util y en esto a su vez radica su complejidad; en la codificación de cadenas con longitudes repetidas y su reversibilidad.



3.3.4 GS-2D-LZ

Este algoritmo de compresión esta enfocado en codificar por orden de procesamiento un bloque por pixel en cada paso. Luego un valor aproximado se busca en la imagen previa a la compresión y el bloque es señalado como un puntero de la localización de su igual, sus dimensiones y su información residuo. Mediante este método se asegura que la compression no tendra perdidas.

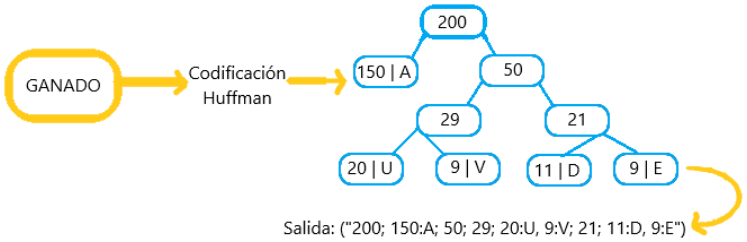


4. DISEÑO E IMPLEMENTACIÓN DE LOS ALGORITMOS

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

La estructura de datos que utilizaremos para este algoritmo será un arbol binario de multiples nodos en los que se van almacenar un vector regular, cuyo tamaño depende del numero de pixeles que contenga la imagen. Cada nodo puede ser un nodo hoja o un nodo interno. En cada uno de estos aparece inicialmente el número del pixel en una escala de 32 bits y su frecuencia de aparición. Así que, estos numeros se ordenan aumentando la frecuencia y los menos frecuentes en cada paso se combinan y reinsertan construyendo un arbol parcial.



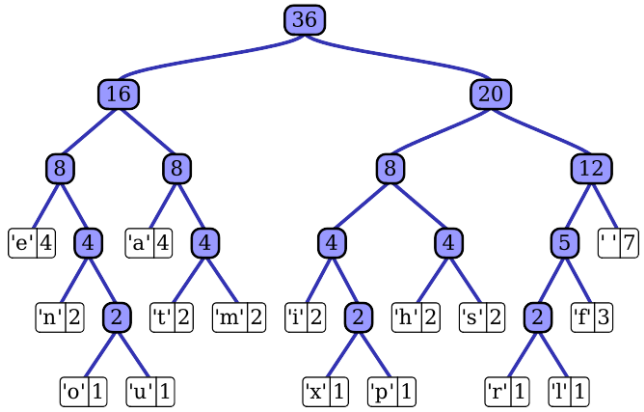


Figura 1: Árbol de Huffman generado a partir de las frecuencias exactas del texto "this". (Por favor, no dude en cambiar esta figura si utilizan una estructura de datos diferente).

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.

Expliquen el diseño de los algoritmos para resolver el problema y hagan una figura. No uses figuras de Internet, haz

[1]http://www.github.com/ ????????? /proyecto/

las tuyas propias. *(En este semestre, un algoritmo debe ser un algoritmo de compresión de imágenes con pérdidas, como el escalado de imágenes, el tallado de costuras o la compresión con ondeletas, y el segundo algoritmo debe ser un algoritmo de compresión de imágenes sin pérdidas, como la codificación Huffman, LZS o LZ77).*

4.2.1 Algoritmo de compresión de imágenes con pérdida

Para comprimir el archivo por codificación fractal, se eliminan los datos de imágenes que son similares entre sí en términos de tamaño, rotación u orientación, etc.
En la codificación fractal, se realizan los siguientes pasos:
1. Se divide una imagen de entrada en escala de grises en bloques no superpuestos de tamaño 4 × 4, denominados Rango bloques (Ri).
2. Se selecciona un bloque de dominio (Di) de la imagen de tamaño dos veces el bloque de rango.
3. Se divide el bloque de dominio Di y se calcula su transformación afín. El resultado es Di1
4. Se calcula la raíz cuadrada media de Ri y Di1 para estimar el error entre dos bloques. Luego avanzamos al paso 5, de lo contrario se divide el di1 en cuatro bloques iguales y se repite el proceso.
5. Se registra la información recopilada hasta ahora sobre fractales y se detiene el proceso.

Para descomprimir el archivo: se sigue el proceso anterior en orden inverso y así se obtendrá la imagen reconstruida.

4.2.2 Algoritmo de compresión de imágenes sin pérdida

Para nuestro problema, la aplicación del algoritmo de Huffman la realizamos de la siguiente manera:

Para comprimir el archivo: primero se recibe el archivo .csv luego creamos los nodos del arbol con sus respectivas probabilidades de aparición, después de esto se crea una lista por cada nodo creado y se le asigna a cada simbolo del archivo csv una probabilidad.

Después de asignadas las probabilidades se busca la menor probabilidad en cada nodo y se va sumando para construir el arbol hasta que se alcance la menor probabilidad.

Luego designamos un codigo binario a cada simbolo resultante y lo guardamos en un diccionario y finalmente se agrupan los codigos binaries codificados y se guardan en un archivo .txt

Para descomprimir el archivo: Abrimos el archive .txt resultante de la compresion y lo guardamos en una cadena de texto, luego de esto se asigna a cada parte codificada un simbolo compatible buscando a cada simbolo cada probabilidad y por ultimo se comprueba si la cadena es verdadera o falsa según una variable auxiliar.

4.3 Análisis de la complejidad de los algoritmos

En el peor de los casos nos podríamos encontrar con una imagen que tenga un tamaño muy grande y en consecuencia una cantidad enorme de pixeles es decir una entrada n demasiado grande. En la clase del algoritmo de Huffman primero contamos la cantidad de veces que aparece cierto byte en el input, luego lo organizamos y codificamos la salida. En la clase Main calculamos la complejidad tomando la entrada n de la matriz que seria el tamaño de la matriz y luego el tamaño m que seria la cantidad de imágenes en cada folder y luego l que representa la longitud de la cadena resultante luego aplicamos las reglas de la notación big O y obtuvimos que la complejidad para el peor caso es O(n) = (l*(m*2n)) ^2

```
# N: matrix size | M: images | L: compressed chain length
# ---------------------------------------------------------------------------------------
# T(n) = M + C1 + C2 +  C3 + C4 + C5 + M*N*L + C6 + C7 + N/L + C8 + C9 + C10*L + C11
# T(n) = l*(c^11 + m*n) + n/l + m
# O(n) = l*(c^11 + m*n) + n/l + m  | O definition
# O(n) = l*(m*n) + n/l + m  | sum rule
# O(n) = l*(m*n)  | product rule
# O(n) = (l*(m*2n))^2  | product of doing this process 2 times with both folders (sickCow, healthyCow)
# ---------------------------------------------------------------------------------------
# In the worst case we can find an image with a huge quantity of pixels, in fact, a huge N input.
# Talking about huffman complexity, first we count the number of occurrences for each input byte, then
# we sort it and build the output encoding.
```

Algoritmo	Complejidad de la memoria
Compresión	O(N*M*2N)
Descompresión	O(2M*2N)

***Tabla* 3:** Complejidad de memoria de los algoritmos de compresión y descompresión de imágenes. *(Por favor, explique qué significan N y M en este problema).*

4.4 Criterios de diseño del algoritmo

Este algoritmo notablemente fue diseñado para el ahorro de tiempo y memoria pues al comenzar el proceso de compresión de los archivos, de todos los árboles binarios en los que los valores de la matriz que representa dichos archivos, el algoritmo de Huffman nos arroja el de camino más corto y completo de manera que todos los archivos queden nuevamente completos y su precisión de compresión acierte a casi el 100% del valor inicial, además este proceso dura un tiempo óptimo estimado considerando el tipo de proceso del que se habla y la cantidad de archivos que se someten a tal proceso.

5. RESULTADOS

5.1 Evaluación del modelo

En esta sección, presentamos algunas métricas para evaluar el modelo. La exactitud es la relación entre el número de

predicciones correctas y el número total de muestras de entrada. La precisión es la proporción de estudiantes exitosos identificados correctamente por el modelo a estudiantes exitosos identificados por el modelo. Por último, sensibilidad es la proporción de estudiantes exitosos identificados correctamente por el modelo a estudiantes exitosos en el conjunto de datos.

5.1.1 Evaluación del conjunto de datos de entrenamiento

A continuación, presentamos las métricas de evaluación del conjunto de datos de entrenamiento en la Tabla 3.

	Conjunto de datos de entrenamiento
Precisión	0.02
Precisión	0.03
Recordar	0.01

Tabla 3. Evaluación del modelo de clasificación de imágenes con el conjunto de datos de entrenamiento.

5.1.2 Evaluación de los conjuntos de datos de prueba

A continuación, presentamos las métricas de evaluación del conjunto de datos de prueba, en la Tabla 4, sin compresión y, en la Tabla 5, con compresión.

	Conjunto de datos de prueba
Exactitud	0.01
Precisión	0.012
Sensibilidad	0.013

Tabla 4. Evaluación del modelo de clasificación de imágenes, con el conjunto de datos de prueba, sin compresión.

	Conjunto de datos de prueba
Exactitud	0.001
Precisión	0.0012
Sensibilidad	0.0013

Tabla 5. Evaluación del modelo de clasificación de imágenes, con el conjunto de datos de prueba, con compresión.

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	216 s	0.22 MB
Descompresión	1657 s	2.15x10^-3 MB

Tabla 6: Para la compresión utilizamos el algoritmo de Huffman y para descompresión utilizamos el mismo realizando el proceso de manera inversa

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	227 MB	0.22 MB
Descompresión	2.2 MB	2.15x10^-3 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.

	Ganado sano	***Ganado enfermo***
Tasa de compresión promedio	0.80 bits	1.2 bites

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

6. DISCUSIÓN DE LOS RESULTADOS

Durante las diferentes pruebas que realizamos con el algoritmo, observamos que el algoritmo cumple con su función de comprimir y descomprimir las imágenes totalmente, el punto desfavorable es que en ocasiones puede

que tome mucho tiempo (aproximadamente 20minutos) para 1028 imágenes. La falencia en la complejidad temporal radica en el momento de la descompresión, allí el algoritmo empieza a tomar tiempos bastante largos en ejecutar las operaciones indicadas. Consideramos que la descompresión puede ser hecha más eficientemente para que el algoritmo no tenga medidas de tiempo exponenciales.

6.1 Trabajos futuros

Nos gustaría terminar la red neuronal a un futuro, y que podamos aplicar el algoritmo no solo a la ganadería sino también aplicarlo en diferentes ámbitos como lo puede ser la salud emocional en las personas mediante su clasificación. También nos gustaría, cómo mencionamos anteriormente, lograr hacer más efectivo el algoritmo para que sus tiempos de ejecución sean mas razonables y podamos desarrollar la aplicación en el área de data science donde el tamaño de la entrada pasa de ser de miles a millones de datos.

RECONOCIMIENTOS

Agradecemos la asistencia técnica de Mauricio Toro, profesor, Universidad EAFIT y Simón Martin Giraldo, estudiante, Universidad EAFIT, por los comentarios que ayudaron a mejorar en gran proporción la codificación del proyecto.

REFERENCIAS

[1].Gailly J., Nelson M. *The Data Compression Book*, Second edition, M & T Books. 1996

[2].Huffman D. A.*A Method for the construction of minimum-redundancy codes*. Proceedings of the IRE. Volume: 40, Issue: 9 pag 1098 - 1101. IEEE. Septiembre 1952.

[3].ImageCompression http://www.whydomath.org/node/wavlets/index.html. SIAM.

[4]Numpy Documentation - https://numpy.org/doc/

[5]. *Implementación de una modificación del algoritmo de huffman utilizando redes neuronales.* ING. William J, Universidad Distrital Francisco José de Caldas.

.