OPTIMIZACION DEL PROCESO DE GANADERIA MEDIANTE EL USO DE ESTRUCTURAS DE DATOS

Juan Andrés Gómez
Universidad Eafit
Colombia
jagomezd@eafit.edu.co

Laura Ortiz
Universidad Eafit
Colombia
lortizc3@eafit.edu.co

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

RESUMEN

El problema a tratar es el cómo optimizar el proceso de la ganadería mediante las tecnologías de la información diseñando algoritmos basados en compresión de imágenes y tablas de hash potenciando el consumo de energía en ámbitos de la ganadería de precisión, la cual es importante, ya que esta misma le da un mejor enfoque a la ganadería, y al mismo tiempo al cuidado bovino.

Una problemática relacionada con el tópico de estudio es la aplicación de satélites de teledetección en ámbito agrícola puesto que en estas mismas se usan tecnología de la información las cuales permiten un estudio optimo de la superficie por la cual se puede deducir que tipo de fertilizante necesita cada sector del cultivo.

1. INTRODUCCIÓN

Una de las principales necesidades del ser humano siempre ha sido la alimentación, lo cual ha llevado ya en el pasar del tiempo de la raza humana a la masificación de este tipo de necesidades por el aumento poblacional. Esto ha llevado a que en el ultimo milenio aumente de una manera exponencial la necesidad de buenas fuentes de alimento. Esto ha direccionado a industrias tales como son las ganaderas a un crecimiento de la mano con la demanda de sus productos. Debido a lo tedioso que ha sido mantener un modelo el cual parte de lo tradicional, se ha generado nuevas tecnologías las cuales permiten una gran mejora de los diferentes estudios los cuales proveen una manutención optima del sector agropecuario.

Este mismo fenómeno ya descrito ha generado un nuevo movimiento de las tecnologías la cual se llama “Ganadería de precisión” (GdP), esto ha llevado al desarrollo de tecnologías similares las cuales ayudan a gran parte del sector agropecuario tal y como puede ser la aplicación de satélites de teledetección.

1.1. Problema

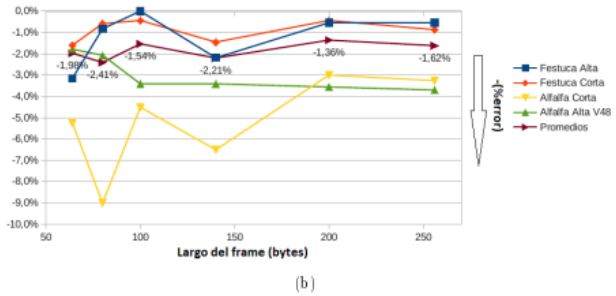
El problema al cual nos enfrentamos es crear mediante una estructura de datos un sistema eficiente el cual permita identificar el estado de salud animal mediante imagen de madera eficiente en cuanto a memoria y tiempo. Esto significaría un avance gigante en la ganadería de precisión la cual podría impulsarse mas gracias a estos sistemas

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 Almacenamiento de sonidos para ganadería de precisión

Captar los sonidos producidos por los animales durante su alimentación, sin interferir en su comportamiento normal y sin intervención del operador, se eligió usar algoritmos de compresión con perdida ya que eran mas útiles que los algoritmos sin perdida y se termino usando el algoritmo CELT. A continuación la figura #1.



(b)

Figura #1: Margen de error de compresión del algoritmo CELT

2.2 Control biológico de la garrapata mediante imagen

Contribuir con la escuela de veterinaria con el Desarrollo de un software con procesamiento de imágenes que reconozca la rapidez y eficacia de extracto de Neem como garrapaticida, sin la necesidad de la presencia de los doctores, se termino binarizando las imágenes para una mejor eficacia en el sistema

2.3 Cloud services integration for farm animals’ behavior studies based on smartphones as activity sensors

Se plantea el uso de smartphones como sensores de actividad para la ganadería debido a que cada vez se ha venido implementando el uso de sensores en la industria, se proponen dos algoritmos de compresión complementarios: Un algoritmo que elimine información redundante y otro que trunca la información a 3 ,4 y 5 decimales. A continuación la figura #2.

Individual data	Compression rate [%] on individual data				Mean compression rate [%] by group			
	6 decimal	5 decimal	4 decimal	3 decimal	6 decimal	5 decimal	4 decimal	3 decimal
Acceleration in the X axis	46.02	46.02	83.56	96.16	4.26	4.26	4.26	4.27
Acceleration in the Y axis	55.81	55.81	86.93	97.64				
Acceleration in the Z axis	61.76	61.76	86.93	92.77				
Euler angles of the device (Roll)	30.38	61.28	90.44	98.49	24.13	24.13	24.13	24.13
Euler angles of the device (Pitch)	27.10	46.62	86.10	97.84				
Euler angles of the device (Yaw)	24.64	28.93	57.16	93.75				
Attitude quaternion in the X axis	27.85	51.73	91.22	98.97	24.13	24.13	24.13	24.47
Attitude quaternion in the Y axis	27.77	51.34	90.77	98.99				
Attitude quaternion in the Z axis	26.06	39.83	81.40	98.01				
Attitude quaternion in the W axis	25.94	39.54	81.10	98.01				
Rotation matrix (element 11)	25.66	37.49	80.69	98.01	24.13	24.13	24.13	24.15
Rotation matrix (element 12)	25.73	37.94	80.90	98.01				
Rotation matrix (element 13)	27.83	50.20	87.98	98.57				
Rotation matrix (element 21)	26.12	40.43	81.29	98.01				
Rotation matrix (element 22)	25.89	39.10	81.00	98.01				
Rotation matrix (element 23)	30.52	61.69	90.68	98.60				
Rotation matrix (element 31)	26.09	40.98	84.59	98.26				
Rotation matrix (element 32)	26.11	40.98	84.72	98.31				
Rotation matrix (element 33)	31.43	66.12	94.94	99.37				
3D Gravitational acceleration (X)	27.79	50.20	87.98	98.57	24.13	24.13	24.23	32.02
3D Gravitational acceleration (Y)	30.49	40.43	90.68	98.60				
3D Gravitational acceleration (Z)	31.40	39.10	94.94	99.37				
3D User acceleration (X)	31.29	62.49	90.49	98.10	24.13	24.13	24.13	24.25
3D User acceleration (Y)	29.62	58.60	90.09	98.19				
3D User acceleration (Z)	30.90	62.41	91.73	98.56				
Rotation rate in the X axis	25.70	62.02	76.10	91.65	24.13	24.13	24.13	24.13
Rotation rate in the Y axis	25.62	36.76	73.97	89.75				
Rotation rate in the Z axis	25.53	35.90	69.75	84.94				
Magnetic Heading	68.40	68.43	68.63	70.87	0.01	0.01	0.01	0.01
True Heading	68.25	68.28	68.51	70.78				
Heading Accuracy	100.00	100.00	100.00	100.00				
Magnetometer data in the X axis	79.31	82.22	94.79	97.86	60.82	60.83	60.84	60.86
Magnetometer data in the Z axis	98.22	98.22	98.22	98.22				
Magnetometer data in the Z axis	72.40	72.40	75.67	88.68				
Latitude	99.93	99.99	100.00	100.00	99.85	99.94	99.99	99.99
Longitude	99.92	99.98	100.00	100.00				
Position Accuracy	100.00	100.00	100.00	100.00				
Course	99.96	99.96	99.96	99.96	99.75	99.75	99.75	99.75
Speed	99.85	99.85	99.85	99.87				
Altitude	99.99	99.99	99.99	99.99	99.99	99.99	99.99	99.99

Figura #2:Efectividad de compresión

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

En lo que sigue, presentamos diferentes algoritmos usados para comprimir imágenes con pérdida. (En este semestre, ejemplos de tales algoritmos son el tallado de costuras, el escalado de imágenes, la transformación de coseno discreto, la compresión con ondeletas y la compresión fractal).

3.2.1 Liquid Rescaling

Es un algoritmo para el re-tallado de imágenes desarrollado por Shai Avidan y Ariel Shamir, que consiste en establecer una serie de costuras o caminos de menor importancia, una vez aplicado se pueden agregar o eliminar lo que amplia o reduce el tamaño de la imagen respectivamente.

Permite definir de manera manual las áreas en las que no se desea que se modifiquen los pixeles y presenta la habilidad de remover objetos completos de las fotografías.

Figura #3: 1er paso: Imagen Original

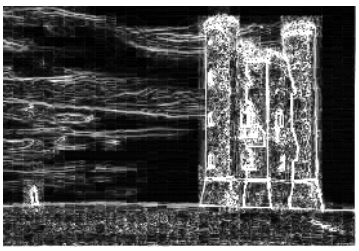


Figura #4: 2do paso: Se calcula el peso, densidad y energía de cada pixel

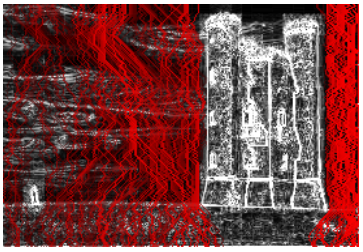


Figura #5: 3er paso: Se realiza una lista de costuras en base a la energía

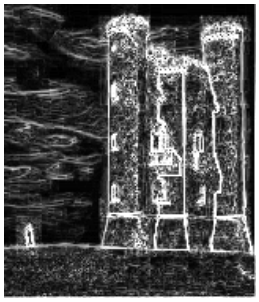


Figura #6: 4to paso: Se eliminan todas las costuras de baja energía

Figura #7: 5to paso: Imagen Final

3.2.2 Image Scaling

Se refiere al ajuste de tamaño de una imagen digital, que en el medio de tecnología de videos se conoce como "upscaling"; al momento de escalar un vector gráfico, los elementos primitivos que componen la imagen se pueden modificar usando transformaciones geométricas sin pérdida de la calidad de imagen, cuando se escalan gráficos ráster se debe generar otra imagen con un mayor o menor número de pixeles, en caso de que decrezcan (scaling down) resulta con una reducción de la calidad de imagen.

Figura #8: Ejemplo de compresión Image Scaling

3.2.3 Compresión JPEG

Este algoritmo, pierde información y esto se refleja en una disminución de la calidad de la imagen; pero tal pérdida no es visible para el ojo humano. El funcionamiento del algoritmo se explica a continuación: se inicia con una imagen en donde cada píxel tiene 3 canales de color (rojo, azul y verde) y lo primero que hace el algoritmo es transformar estos 3 canales en 2 de color y uno de brillo; luego, a los canales de color les baja la resolución y allí es donde se generan las perdidas ya que elimina o modifica alguna información, ya que el ojo humano no nota estas modificaciones del color, pero si lo notase en el brillo. Luego divide cada canal en cuadros de 8x8 pixeles y a cada cuadro por separado le aplica la transformada discreta bidimensional del coseno y cuantificación perceptual que lo que realiza es suavizar las variaciones bruscas de brillo y color, y ya por último se aplica un algoritmo de compresión sin perdida el cual se explica líneas adelante, el algoritmo de Huffman.

Figura #9: Ejemplo de compresión por JPEG

3.2.4 Compresión Fractal

La palabra fractal es término reciente propuesto en el año de 1975, el cual se define como un patrón geométrico que se repite en varias escalas. Por tanto, como su nombre lo indica, la compresión fractal es un método de compresión basado en fractales, este es el más apropiado para imágenes o texturas naturales, porque ciertas partes de la imagen se parecen a otras. Como se deben buscar similitudes propias de la imagen, este método puede llegar a ser bastante costoso, aunque, por otro lado, la decodificación de la imagen se hace de una manera bastante rápida. Por tanto, la eficiencia del algoritmo va a depender mucho en la complejidad de la imagen y la profundidad del color. Una de las principales características de la compresión fractal, es que las imágenes se vuelven independientes de la resolución (escalado fractal), y esto hace posible, que la imagen se pueda pasar una mejor resolución (interpolación fractal).

Figura #10: Ejemplo de compresión Fractal

3.3 Alternativas de compresión de imágenes sin pérdida

En lo que sigue, presentamos diferentes algoritmos usados para comprimir imágenes sin pérdida. (En este semestre, ejemplos de tales algoritmos son la transformada de Borrows y Wheeler, LZ77, LZ78, la codificación Huffman y LZS).

3.3.1 Codificación de Huffman

Es un procedimiento que permite asignar a los diferentes símbolos a comprimir, un código binario. Este algoritmo crea un árbol de nodos; primero se debe establecer un orden prioritario, el más importante es el símbolo que aparece con menor frecuencia en la cadena, luego, se eliminan los dos símbolos más prioritarios, construyendo así un nuevo "padre" que es el resultado de la suma de las frecuencias eliminadas y se ubica nuevamente en la cola de prioridad, iterativamente se realiza este proceso hasta que sólo quede un elemento, así queda construido el árbol. Luego, para asignar el código binario se contarán los pasos efectuados para llegar a cada símbolo del árbol (movimiento a la izquierda=0, a la derecha=1), obteniendo el valor de cada uno y por último reemplazándolos en la cadena.

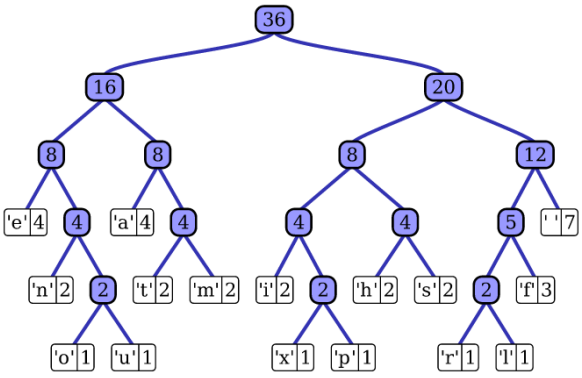


Figura #11: Grafico de Huffman

3.3.2 LZ78

El algoritmo LZ78 usa una estructura de datos, específicamente trie, basada en diccionario para comprimir los datos. Este nació para mejorar el rendimiento respecto al LZ77, es por así decirlo, una segunda versión mejorada. Respecto al funcionamiento para la compresión, se lee una cadena carácter por carácter y verificamos si el nodo actual (Cada nodo está marcado con el índice del diccionario) tiene algún borde de salida que contenga al carácter leído; en caso positivo configuramos el nodo actual como el nodo encontrado y repetimos el proceso, en caso negativo, se crea un borde de salida con el carácter leído se crea un nodo nuevo, el cual pasa a ser el nodo raíz e igualmente se repite el proceso

Example 1: LZ78 Compression Step 6

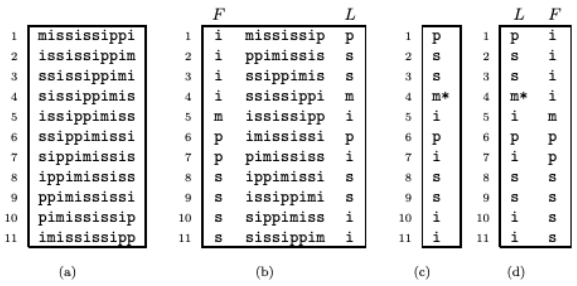
ABBCBCABABCAABCAAB **POS = 10**
C = A
P = BCA

STEP	POS	OUTPUT	DICTIONARY
1	1	(0,A)	A
2	2	(0,B)	B
3	3	(2,C)	BC
4	5	(3,A)	BCA
5	8	(2,A)	BA
6	10	(4,A)	BCAA
12			

Figura #12: Tabla explicando el funcionamiento del algoritmo LZ78

3.3.3 Burrows Transform

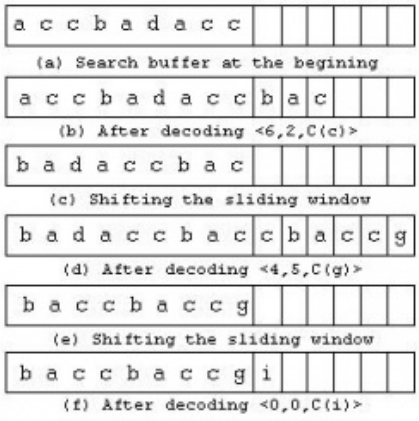
La transformación de Burrows-Wheeler reorganiza una cadena de caracteres en series similares. Es un algoritmo que prepara los datos para su posterior uso con técnicas de compresión, al ingresar una cadena de caracteres, la transformación conmuta su orden. El algoritmo toma la cadena de entrada y realiza todas sus rotaciones hasta que el carácter inicial vuelva a quedar en su misma posición, luego con los resultados de todas estas rotaciones los ordena por orden alfabético (los símbolos especiales como *,|,") se toman como últimos y ya una vez organizados, se toma el último carácter década cadena y así se obtiene la salida esperada.



Figura#13: Codificación del algoritmo de Burrows

3.3.4 LZ77

Es un compresor basado en algoritmo sin pérdida, este algoritmo es un tipo de codificador diccionario en el cual existen los literales, banderas y palabras claves; se empieza a recorrer la cadena y si se encuentra con un literal lo deja totalmente igual, si encuentra una bandera especifica si lo que sigue es un literal o un comprimido y si lo anterior es un comprimido (que es una especie de palabra clave) se lleva a una posición en un diccionario que arroja que bytes continúan. Por ejemplo, en una imagen las esquinas pueden ser iguales y al comprimir tales esquinas lo que hacemos es que se tenga que guardar una sola vez y por así decirlo evitar las repeticiones.



Figura#14: Ejemplo de codificación del algoritmo LZ77

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.

4.1 Estructuras de datos

4.1.2 Algoritmo de compresión de imágenes con perdida

Para almacenar los datos usamos una lista enlazada, y dentro de ella en cada posición tenemos un diccionario que contiene el nombre del archivo y una matriz que contiene los valores de los pixeles, utilizamos el diccionario ya que nos permitía conservar los nombres originales del archivo luego de la compresión y descompresión. Para el efecto del algoritmo usamos sólo la matriz de pixeles.

65	34	2	128
64	92	14	23
18	38	64	24
115	2	45	37

Figura #15: Matriz de 4x4, la cual contiene los pixeles de una imagen

Después de la compresión la matriz se reducirá y luego la descompresión hará que vuelva a su tamaño original

4.1.2 Algoritmo de compresión de imágenes sin perdida

Para la compresión y descompresión de imágenes sin perdida igualmente que para la compresión de imágenes con perdida usamos una lista enlazada para almacenar los archivos del dataset y en cada posición de la lista almacenamos un diccionario que contiene el nombre del archivo (esto para poder conservar el mismo nombre del archivo luego de la compresión y descompresión) y la matriz de píxeles; para el algoritmo usamos un arreglo tanto para la compresión como para la descompresión; en el caso de la compresión el arreglo tendrá en cada posición una tupla y en la descompresión un arreglo de números, en este caso píxeles de la imagen. En la siguiente figura se ilustra mejor lo explicado anteriormente:

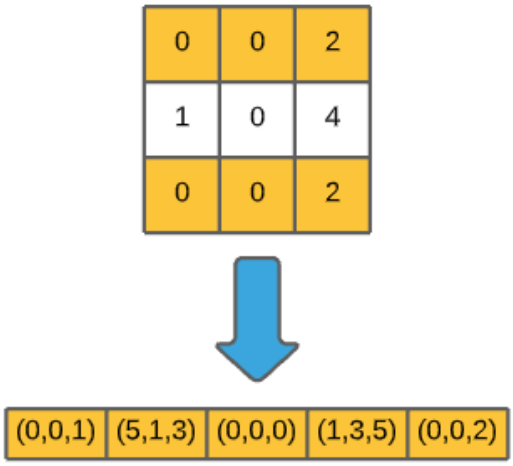


Figura #16: Matriz de 3x3, la cual contiene los pixeles de una imagen y un arreglo de tuplas que contiene el resultado de la compresión

Como se puede observar en la imagen, la primera matriz es la original del archivo, es decir, aquella que contiene los píxeles y el arreglo bajo la flecha es el resultado de la compresión de dicha matriz, como se puede ver es un arreglo de tuplas y luego de la descompresión la matriz quedara igual a la original

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

Para la compresión y descompresión de las imágenes con pérdidas usamos el escalado por medio del vecino más cercano. Este método es muy sencillo, lo único que se hace es dividir la matriz de pixeles de la imagen en submatrices de igual tamaño y escogemos cualquier pixel dentro de cada submatriz, pero siempre el de la misma posición, es decir si en la primera submatriz escogemos el pixel de la mitad en todas las submatrices también deberemos de seleccionar el de la mitad

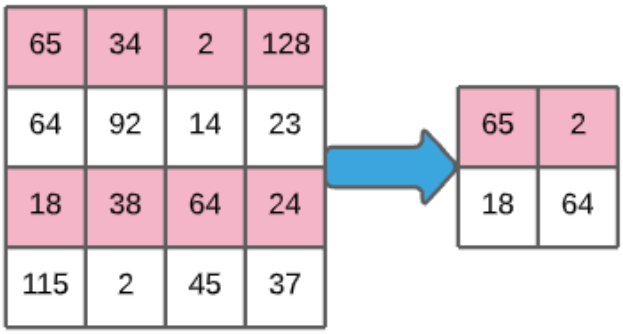
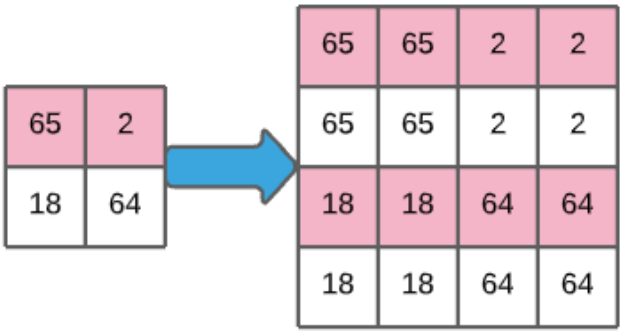


Figura #17: Compresión mediante el algoritmo elegido

Para la descompresión lo que hacemos es duplicar el valor que obtuvimos de la compresión a lo largo del tamaño de la submatriz definido y así obtendremos nuevamente la cantidad de píxeles originales.



Figura#18: Descompresión mediante el algoritmo elegido

4.2.1 Algoritmo de compresión de imágenes con pérdida

Para la compresión y descompresión de imágenes sin perdida usamos el algoritmo LZ77. Por un lado, para la compresión se debe convertir la matriz de pixeles en un arreglo, esto se puede hacer a través de la concatenación; luego de tener la matriz vista como un arreglo se define un tamaño de ventana y de buffer, este tiene que tener un buen balance, es decir, si se pone un buffer muy pequeño la matriz no se comprimirá nada o muy poco y, por otro lado, un buffer grande significa mayor tiempo de ejecución. Luego de tener esto, el compresor creara tuplas de la forma (S, Q, V) en donde S es la cantidad de espacios que se mueve hacia atrás para encontrar una repetición, Q es la cantidad de espacios que toma para la repetición y V es el valor que continua; este método consiste básicamente en encontrar repeticiones y almacenarlas en formas de tuplas que contienen toda la información necesaria (sin perdida).

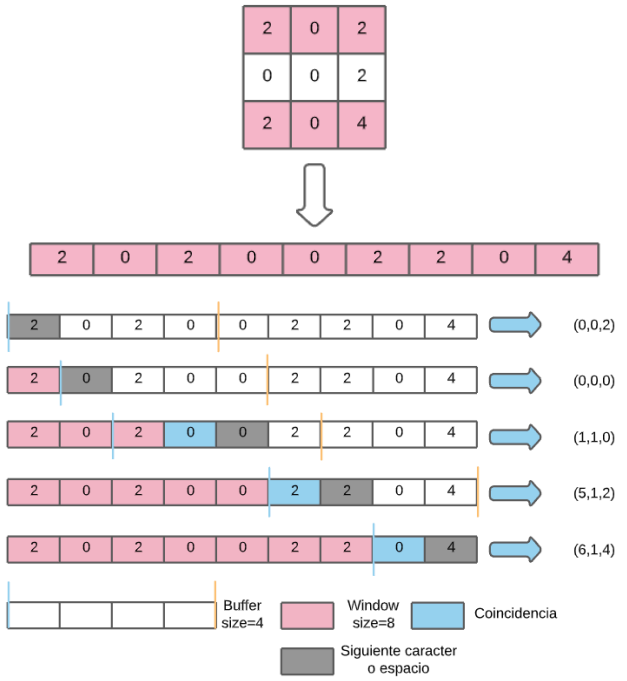


Figura #19: Compresión usando LZ77

Por otro lado, para la descompresión se toman estas tuplas que se obtienen en la compresión y se seguirán las indicaciones contenidas en la información que brindan estas tuplas; es decir, S = cuantos espacios me devuelvo hacia atrás, Q = cuantos espacios tomo, V = que carácter sigue, entonces si se tiene un arreglo que contiene por ejemplo "[4,0]" y se encuentra una tupla (2,1,4) esto quiere decir que se devuelve dos posiciones atrás, en este caso al 4, toma 1 espacio que es el 4 y el espacio que sigue contiene un 4, entonces esto arrojara como resultado "[4,0,4,4]"; finalmente este arreglo se debe convertir nuevamente en matriz. En la imagen a continuación se ilustra con mayor claridad.

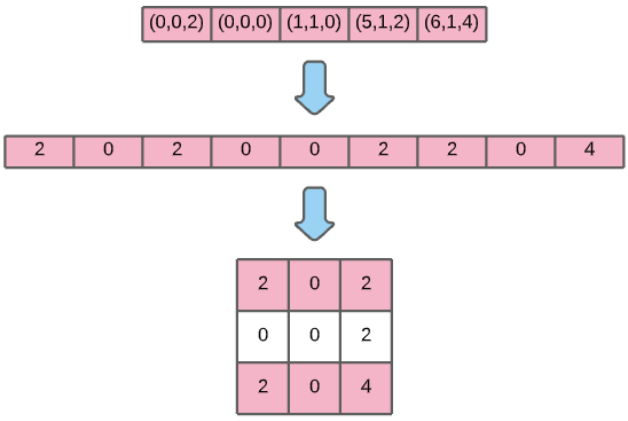


Figura #20: Descompresión usando LZ77

4.3. Análisis de la complejidad de los algoritmos

Para lograr un buen balance entre tiempo y consumo de memoria, teniendo en cuenta que las imágenes suelen ser de tamaños muy grandes, aplicamos primeramente el algoritmo del vecino mas cercano y luego el LZ77 para así trabajar con imágenes mas pequeñas, por ende, la imagen quedaría con un poco de perdida. A continuación, explicamos la complejidad de cada algoritmo y la total.

Algoritmo vecino mas cercano	Complejidad tiempo
Compresion	O(n*m)
Descompresion	O(n*m)

Tabla #1: Complejidad temporal de los algoritmos de compresión y descompresión de imágenes por medio del vecino mas cercano. En donde N es la cantidad de filas y M la cantidad de columnas de la matriz luego de la compresión o descompresión

Algoritmo LZ77	Complejidad tiempo
Compresion	O(n^2*m^2)
Descompresion	O(n^2*m^2)

Tabla #2: Complejidad temporal de los algoritmos de compresión y descompresión de imágenes por medio del LZ77. En donde N es la cantidad de filas y M la cantidad de columnas de la matriz que contiene los pixeles de la imagen

Algoritmo Final	Complejidad tiempo
Compresion	O(n^2*m^2)
Descompresion	O(n^2*m^2)

Tabla 3: Complejidad temporal de los algoritmos de compresión y descompresión de imágenes por medio del vecino mas cercano y LZ77. En donde N es la cantidad de filas y M la cantidad de columnas de la matriz que contiene los pixeles de la imagen

Luego de tener las complejidades temporales de todos los algoritmos, realizamos el mismo procedimiento para la complejidad en memoria.

Algoritmo vecino mas cercano	Complejidad memoria
Compresion	O(n*m)
Descompresion	O(n*m)

Tabla 4: Complejidad de memoria de los algoritmos de compresión y descompresión de imágenes por medio del vecino mas cercano, en donde N es la cantidad de filas y M la cantidad de columnas de la matriz luego de la compresión o descompresión

Algoritmo LZ77	Complejidad memoria
Compresion	O(n*m)
Descompresion	O(n*m)

Tabla 5: Complejidad de memoria de los algoritmos de compresión y descompresión de imágenes por medio del LZ77. En donde N es la cantidad de filas y M la cantidad de columnas de la matriz que contiene los pixeles de la imagen

Algoritmo Final	Complejidad tiempo
Compresion	O(n*m)
Descompresion	O(n*m)

Tabla 6: Complejidad de memoria de los algoritmos de compresión y descompresión de imágenes por medio del vecino mas cercano y LZ77. En donde N es la cantidad de filas y M la cantidad de columnas de la matriz que contiene los pixeles de la imagen

4.4. Criterios de diseño del algoritmo

Para el diseño del algoritmo nos basamos tanto en su complejidad en tiempo como en memoria, es decir, su eficiencia; como también en los fines para los cuales será usado el algoritmo, en este caso para un modelo de inteligencia artificial en el cual se transmiten las imágenes y este arroja si el ganado según la imagen esta enfermo o sano.

En el caso del algoritmo de compresión de imágenes con perdida se eligió el escalado mediante el vecino mas cercano, ya que en primer lugar teniendo en cuenta el uso posterior de la imagen el re-tallado como remueve objetos de la imagen, no era viable; por otro lado, la compresión fractal puede resultar muy costosa en algunas ocasiones, entonces por estas razones optamos por el escalado de imágenes y por ultimo, la interpolación bolinear y el vecino mas cercano poseen la misma eficiencia en tiempo y memoria, pero, mientras que la interpolación bolinear deja la imagen borrosa, el vecino mas cercano la deja tipo pixel art; por lo que consideramos que el modelo puede identificar mas fácilmente el patrón en una imagen tipo pixel art que conserva su forma a una imagen que en ocasiones podría ser demasiado borrosa. Por otro lado, escogimos el algoritmo LZ77 ya que, por ejemplo comparándolo con el LZ78, este tiene unas mejoras respecto al consumo de memoria pero tarda mas tiempo, y teniendo en cuenta que la prioridad era el tiempo descartamos este algoritmo; por otro lado el algoritmo de Huffman tiene una complejidad en tiempo de O(N*M*log(N*M)) tanto para compresión como para descompresión, por lo que pudimos notar que aunque la compresión se hace mas rápida respecto al LZ77, la descompresión es mas lenta y además este algoritmo usa mucha mas memoria, aunque sus tasas de compresión suelen ser muy similares; por ultimo el algoritmo de BurrowsWheeler notamos que tiene una complejidad en tiempo de O(N*M*P) en donde N son las filas, M las columnas de la matriz y P la repetición mas larga en toda la cadena ingresada, por lo que en ocasiones su complejidad puede ser mayor al LZ77 aunque su descompresión y complejidad en memoria O(N*M) si es igual al LZ77. Finalmente, nos pareció que el LZ77 era un buen algoritmo y sus resultados fueron bastante óptimos. Y por ultimo, aplicamos una combinación del algoritmo del vecino mas cercano y LZ77 ya que, teniendo en cuenta que el conjunto de datos tenía imágenes muy grandes, esto iba a causar que la compresión tardara mucho mas tiempo, entonces optamos por un buen balance reduciendo la imagen mediante el vecino mas cercano para luego aplicarle el LZ77 y así causar que el algoritmo se ejecute en un menor tiempo, aunque tendrá un poco de perdida pero consideramos que no afectara mucho la precisión del modelo. Por otro lado, para los modelos se suelen usar imágenes muy pequeñas, por lo que se podría realizar solo con el LZ77. Y por último al experimentar con los diferentes tamaños de ventana y buffer en el algoritmo LZ77, notamos que, a menor tamaño, el algoritmo se ejecuta mas rápido, pero comprimiendo a una menor tasa y viceversa si el tamaño de estos es mas grande; por lo que realizamos el algoritmo con un buen tamaño de buffer y ventana para que corra a una tasa medianamente rápida y a la vez pueda encontrar repeticiones entre los pixeles de la imagen.

5. Resultados

5.2 Tiempos de ejecución

A continuación, explicamos la relación entre el tiempo promedio de ejecución en segundos y el tamaño promedio en KB de las imágenes del conjunto de datos completo, en la Tabla 8.

	Tiempo promedio	Tamaño promedio
Compresion	~30	~284
Descompresion	~0.4	~284

Tabla 7: Tiempo de ejecución de los algoritmos LZ77 y escalamiento mediante el vecino mas cercano para diferentes imágenes en el conjunto de datos, tomamos 10 imágenes del conjunto de datos, tomamos su tiempo de ejecución y los promediamos; estos tiempos se tomaron con un buffer igual a 15 y una ventana igual a 3

	Tiempo promedio ventana20 buffer25	Tamaño promedio ventana20 buffer10
Compresion	~91	~28
Descompresion	~0.5	~0.5

Tabla 8: Tiempo de ejecución de los algoritmos LZ77 y escalamiento mediante el vecino mas cercano para diferentes imágenes con tamaños de buffer y ventana diferentes a la tabla anterior

De igual manera, teniendo en cuenta que los tiempos de ejecución varían considerablemente según el tamaño de la ventana, tomamos el promedio de los tiempos para las mismas imágenes de la tabla anterior con otros tamaños de buffer y ventana.

5.3 Consumo de memoria

Presentamos el consumo promedio de memoria en MB de los algoritmos de compresión y descompresión en KB en la Tabla 10.

	Consumo promedio memoria	Tamaño promedio archivo
Compresion	~2	~250
Descompresion	~1.6	~250

Tabla 9: Consumo de memoria de los algoritmos LZ77 y escalamiento mediante el vecino mas cercano para diferentes imágenes en el conjunto de dato

6. Discusión de resultados

Consideramos que si es apropiado el consumo de memoria y tiempo considerando los tamaños de las imágenes; aunque también pudimos notar que la descompresión se ejecuta muy rápidamente, mientras que la compresión tarda mucho mas tiempo en el algoritmo LZ77, pero esto se puede corroborar observando la complejidad en tiempo según la notación O. La relación de compresión es adecuada ya que como aplicamos el vecino más cercano y se pierden algunos datos, no podríamos reducir este aun mas ya que se podrían generar grandes perdidas y por ´ultimo consideramos que la compresión no debería afectar significativamente la exactitud del modelo.

6.1. Trabajos futuros

Lograr un mayor aprendizaje a la IA y una mejora al algoritmo buscando una mayor optimización de los procesos de compresión y descompresión del algoritmo, una mayor destinación de recursos a alimentar la IA y lograr una disminución en los posibles errores en cuanto al producto final

7. Reconocimientos

Esta investigación fue parcialmente apoyada por el Gobierno Nacional mediante la Generación E - Componente Excelencia 2020.

REFERENCIAS

1. Jose Chelloti O. 2014. Desarrollo e implementación de un dispositivo de adquisición y almacenamiento de sonidos para ganadería de precisión. (November 2014). Retrieved August 16, 2021 from http://sedici.unlp.edu.ar/handle/10915/42007

2. Zenteno Aguilar, Cristian del Carmen. 2012. Control biológico de la garrapata en el sector ganadero con procesamiento de imágenes. (January 2012). Retrieved August 16, 2021 from http://repositoriodigital.tuxtla.tecnm.mx/xmlui/handle/123456789/171

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

4. Explicación detallada del algoritmo de interpolación bilineal de imágenes Python realiza el algoritmo de interpolación bilineal – programador clicl.(s.f.) Programador clic. Recuperado 11 de abril de 2021, de https://programmerclick.com/article/ 9212974781

5. Moreno,R.Algoritmos básicos para la compresión sin perdidas. 2010. ahs

6. Programmer click. Explicación detallada del algoritmo de interpolación bilineal de imágenes toro