National College of Ireland

Optimizing Data Storage through Neural Network Based Adaptive Compression

MSc Research Project
MSc in Cloud Computing

Carlos Alberto Noyola Sanchez
Student ID: X22232991

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……. Carlos Alberto Noyola Sanchez ……

Student ID: …… X22232991 ……..……

Programme: …… MSc in Cloud Computing …… **Year:** 2024..

Module: …… MSc Research Project ……..……

Supervisor: …… Vikas Sahni ……..……

Submission Due Date: …… 12th Aug 2024 ……..……

Project Title: Optimizing Data Storage through Neural Network Based Adaptive Compression

Word Count: ……6889…… **Page Count**……22……..……..

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: …… Carlos Alberto Noyola Sanchez ……

Date: …… 10th Aug 2024 ……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Optimizing Data Storage through Neural Network Based Adaptive Compression

Carlos Alberto Noyola Sanchez
X22232991

Abstract

Nowadays digital information is growing exponentially, with large amounts of information being generated and stored every second. This exponential growth presents significant challenges for data storage infrastructure. This is increasing the cost for storing data which impacts negatively on all kinds of businesses. Traditional compression algorithms are effective for specific data types but often fail when they are applied to mixed data types. Each file type—audio, video, text, or executable—presents unique byte-level patterns, which can be helpful in finding the best compression method.

This research addressed the problem of inefficient compression methods when an individual method is applied to mixed data types. This can lead to inefficient data storage and increase the costs. While data is growing, businesses need to find an efficient way to store data without sacrificing performance and quality.

To tackle this problem, a CNN based solution is proposed to identify and analyse patterns to byte level in files. By converting file data into grayscale images, a CNN can be trained to detect patterns that helps to identify the right compression method that offers the highest compression ratio from a predefined list of categories.

Results show that the CNN was able to infer the compression mechanism for different files by analysing patterns in the grayscale images, generated from the mixed data files. The experiments run showed no noticeable differences when the dimensions of the images were changed or when more layers were used. A 96% of accuracy was obtained in overall for the experiments performed.

1 Introduction

The amount of stored data has been increasing rapidly around the world. It is projected that in 2025, the global datasphere will expand to 163 zettabytes and will have increased again by a further ten times by 2026 (Reinsel, Gantz and Rydning, 2017). However, data storage cannot keep up with the growing pace of the evolution of technology due to Moore's Law (Chien and Karamcheti, 2013). Despite the advances in technology, the actual storage solutions are not enough for the large volume of information that is generated daily. This is why a better way to store the information is needed.

A promising solution for this challenge of efficient data storage is the lossless data compression, a field that has been evolving since David Huffman presented his lossless

compression algorithm (Moffat, 2020). Unlike lossy compression methods that sacrifice some data in order to reduce storage space, lossless compression maintains the integrity of the original data. This ensures that no information is lost during the compression process. This characteristic is highly important in scenarios where data precision is important, such as medical imaging, archival storage and scientific research.

Throughout the years, many lossless compression techniques have been developed, and each one uses different strategies to achieve efficient data storage. Algorithms like Run-Length Encoding (RLE) (Emon *et al*., 2023), Lempel-Ziv (LZ78) (Welch, 1984), DEFLATE (Deutsch, 1996) and so on, use different approaches to identify and utilize patterns in the files in order to reduce the storage requirements. The efficiency of those algorithms will depend on the patterns of each individual file. In this matter, the RLE compression method can sometimes outperform Huffman encoding, even when the files share same format.

Due to the nature of patterns in the files, machine learning (ML) models present a good alternative to improve data compression methods. ML models have shown the capability to identify complex patterns and do predictions based on that information. By using those capabilities ML models can be the key to develop new algorithms of adaptative compression that identify dynamically patterns in the files, maximizing the compression ratio and reducing the storage needed for the file.

1.1 Time and cost

One of the main issues in managing this amount of data is the inefficiency of traditional storage expansion methods, which have not kept pace with the data growth. This is because each type of data file—whether audio, video, text, or executable— presents unique patterns at the byte level that make some compression methods more effective than others. Identifying and using these patterns can lead to more efficient storage solutions. Those patterns can be used to predict the best way to compress files. Convolutional Neural Networks (CNNs), known for their good pattern recognition capabilities, have demonstrated potential in multiple domains, one being malware detection files which are converted into images for analysis (Akash *et al*., 2023). This capability suggests a novel approach: using CNNs to determine the optimal compression method for multiple files based on their byte-level patterns using grayscale image representation.

Traditional compression algorithms can be good for certain types of data but can be inefficient for many other types of data, giving different results, even when it is the same data type (Hidayat *et al*., 2023). As a result, a compression technique that performs well for audio files may not be good for text documents, and even when it is the same file type, some algorithms can achieve better compression ratios than generic methods. The most common compression algorithms such as Run-Length Encoding (RLE), Huffman coding, and the Lempel-Ziv family (LZ77, LZ78, and LZMA) have many limitations in their way to adapt to various data types. Because of this, there is a need for a solution that can dynamically analyse a file's byte sequence and determine the most effective compression technique.

In terms of data storage infrastructure, the financial cost of inefficient data storage is significant. As the data volumes and file sizes increase, the data storage costs will increase as well. An efficient compression method could reduce these costs significantly, which would help the economy of businesses and organizations in terms of storage solutions. Furthermore, the Quality of Service (QoS) would improve with an efficient data storage, which in turn would reduce the time needed to retrieve the data and network latency. This improvement can help organizations to get faster to the data and get a better application performance. On the other hand, inefficient storage can lead to much higher costs, slow down the data retrieval, and increase latency. This can negatively impact the user experience and the overall performance of to the final users.

The efficient storage management also has benefits for the environment. By reducing the energy required to transfer large files and optimizing disk space usage, the carbon footprint would be minimized, and this will lead us to more sustainable data management practices. The traditional compression methods have been well-studied, but they often rely on heuristic or manual selection processes that may not always be the best way to compress all file types. With the introduction of modern hardware, especially the high-end GPUs and ASICs like Tensor Processing Units (TPUs), it is easier to train and deploy neural networks for complex tasks like image recognition, text generation and big data analysis. In this research, it is believed that CNN could also be useful to infer data compression algorithms.

1.2 Research question

Can a convolutional neural network (CNN) infer the compression method that offers the best compression ratio by analysing patterns in files from graphical representation in binary of mixed data types.

This research presents an innovative approach to cloud data storage optimization that uses Convolutional Neural Networks. By analysing the grayscale representation of files at the byte level and identifying patterns on those images, CNNs can infer the most suitable compression technique for each file. The proposed solution includes training a CNN with a grayscale image with tags to classify. The CNN will select the best compression method more effectively than the traditional heuristic approaches. This approach not only focuses on compression efficiency but also will adapt to the specific characteristics of each file, providing a good solution that would increase the storage efficiency and reduce costs.

The use of traditional lossless compression techniques and Machine Learning capabilities could offer a solution for the efficient data storage challenge. This research will explore how machine learning models can be used to improve the way data is stored, by identifying complex patterns in files, and finding the compression method that offers the best compression ratio in a world where data information is growing exponentially.

2 Related Work

File compression is a critical requirement to save storage space as the size of the files are growing exponentially every day, but the storage grows at Moore's Law pace. Although the field of data compression is making great advances every day due to the need to efficiently transmit and store the vast amounts of digital information that is produced worldwide, there are still some areas that can be improved. This research will review the contributions and the most up-to-date advances in file compression mechanisms. This section will review the heuristic compression mechanisms, the use of neural networks for pattern recognition as compression mechanisms based on machine learning. Moreover, this section will explore the uses cases of adaptative compression mechanisms that optimize the storage dynamically based on features and characteristics of the file to be compressed.

2.1 Heuristic file compression mechanisms for files.

Sarkar, Sarkar and Banerjee (2016) present a novel Huffman approach to reduce the size of large files, which is needed for multiple applications. The method proposed aims to solve challenges like limited storage size and the data transfer by compressing the data in a lossless way. Implemented using MATLAB, the algorithm compresses the files by taking the numeric groups and converting their ASCII values, which gives a new level of security. The outcomes suggest that the algorithm can achieve a higher level of compression ratio which makes it a viable solution for compression and decompression for both offline and online data. Similarly, a comparison of methods are studied in Aich *et al*. (2019) with Huffman encoding which look for a way to store text data efficiently. The compression algorithms are Huffman encoding, Binary encoding and One-Hot encoding, in which Huffman encoding shows that reducing the storage needed requires just 0.1% of memory when it is compared with On-Hot encoding for a sample of 8000 words. The reduction is achieved by coding a variable based on the word occurrence, assigning short codes to the most frequent words. However, the study showed that the Huffman encoding requires a high computational power along with a large volume of RAM, especially with large datasets. Those kinds of challenges have to be taken into account when a solution that is computationally efficient and has a good storage optimization is provided

Mukherjee *et al*. (2023) show SCOPe a framework designed to optimize the cost of storing files in the cloud and at the same time ensures that performance and latency requirements are met. The framework includes a module called OPTASSIGN that optimizes the compression based on patterns and COMPREDICT that predicts what the compression performance would be. When the historical data is analysed, SCOPe can come up with strategies for the different types of data and files, this way SCOPe can achieve high compression ratios. It concludes that 50% to 83% of cost savings could be achieved on large datasets. This approach shares similarities with the Ares framework proposed by Devarajan, Kougkas and Sun (2019) which is another modular and adaptative framework. Ares can handle the massive amounts of data produced by modern applications, which require different compression techniques that vary

from the format and file type. The framework classifies the files and selects the most appropriate compression method. The results show that Ares outperforms heuristic compression methods which means that Ares can achieve 2-6x better performance with little overhead and provide an infrastructure that can change according to the application needs.

Another framework is proposed by Dong *et al*. (2021), who addressed the challenges of compress large volumes of data in real time to reduce the storage overhead and avoid the network saturation. This compression algorithm called NOCAQF is based in quadratic functions and provides a high rate of compression and a low computational complexity, making it suitable for dynamic industrial environments. The results shows that NOCAQF achieves a compression rate of 87% while using 32% of CPU processing power. NOCAQF outperforms most of the linear and non-linear algorithms, but the algorithm dependency of quadratic functions can lead to unpredictable results with patterns that are really complex or unpredictable. Those shortcomings along with the delay when it is compared with linear algorithms can be a limitation in real-time applications that require a low latency of processing. This method is parallel to the methods proposed by Kato, Yamagiwa and Wada (2023), a way to compress the data by dividing the data stream and assigning a different task in a multicore processor. The technique has three strategies to handle the different fragments: fully in-order, hybrid, and fully out-out-order. The out-of-order method shows better results in terms of performance, speed and throughput. Despites those improvements, this method introduced many complex tasks such as management of the chunks during the compression, bigger sizes due to metadata that is included in the chunks and a possible overhead with thread management. Additionally, the efficiency of the method depends on the hardware capabilities and the characteristics of the data stream.

Lastly, Bhardwaj and Garg (2023) explores lossless ZIP compression which shows that more data can be stored in QR codes without losing information. This is achieved by using DEFLATE and INFLATE methods from the ZXing library[1]. The study showed an increase of data stored in the QR code, that goes from 7% up to 73% of compression. In contrast, the compression and decompression of the data comes with an increase of computational load, that could be a limitation for real-time applications. Moreover, the ZIP compression can be considered effective, but it may not be the best for all types of QR codes and also, would not be the most effective when the data varies, taking into account Audio, Video, Images and Document files.

2.2 Detection of patterns in CNN

The use of CNN to detect malware is well studied, those methods transform the binary data into visual representations as images and heatmaps. Mavric and Yeo (2018) developed a tool to visualize binary data of the PDF as 2D heatmaps that help to identify embedded JavaScript or shellcode. However, it may not work against sophisticated obfuscation techniques.

[1] https://zxing.github.io/zxing/index.html

Similarly, Xiao and Yang (2019) proposed a method to detect malware in android by converting the bytecode into RGB images, this way the CNN will be able to extract features and characteristics and detect malware patterns. While effective, it faces challenges in processing efficiency as the bytecode files include benign segments that do not contribute in the malware detection. Cui *et al*. (2018) and Kalash *et al*. (2018) took the concept of convert the malware code into grayscale images. Those images were used by the CNN to classify and detect malicious patterns. Both research papers achieved high accuracy rates, but also noticed challenges like the fixed size of the image and the limitations of the model's performance due to data imbalance and quality.

Other research has focused to improve malware detection my using more complex scenarios and different file types. Vinayakumar *et al*. (2019) presented ScaleMalNet, a framework that combine CNN and LSTM to detect and classify malware, using binary image representations. This method is effective to detect obfuscated malware, but it requires large datasets and significant computational resources, which may not be feasible in all environments. Also imbalanced datasets may lead to bias in the model and affect the precision of the model. Wang *et al*. (2017) applied CNNs to detect malware by converting the network traffic into images. However, this method focused on spatial features rather than temporal features. The future work highlighted the use of balanced data and integrate features like temporal data to increase the detection accuracy.

A way to detect malware in files by converting the binary files into grayscale images was proposed in Jonnala *et al*. (2023), and then those images were used to train a CNN and detect if the file presented malware or not. When the neural network is fed with the grayscale images, it was able to identify complex patterns which improved the malware detection compared to signature-based methods. This method can classify the malware into 25 categories with high precision, which provides an effective way to detect malware in real-time. However, the model presents a high computational requirement to process the images and train the neural network. Also, the performance depends on the quality and diversity of the training data which can be difficult to generalize to the new and unseen malwares without updates and retraining. Similarly, in Kalyan *et al*. (2022) with malware detection, transforming the binary executables into grayscale images and classify them if the files contains malware or not. Both researchers reported high accuracy in malware detection, which indicates the effectiveness of CNNs to find malware in images. The difference lays in the way the images are pre-processed; the first of them converts the binary data into hexadecimal and then into images, while the second one uses PIL library in python to do the conversion binary-to-image directly into 64x64 pixel images. While both methods show the potential of CNN to detect malware based on grayscale images, they also need high computational requirements and a large dataset to train the models.

Some research explored new methods to detect malware other than binary-to-image representations. Jeong, Mswahili and Kang (2023) proposed a method to detect malware focusing on non-executable files like Microsoft Office documents. This method adds stream-level results to improve file-level predictions, increasing the detection performance of malware. But it relies on aggregation functions that may not capture complex patterns across byte

streams. Hwang *et al*. (2020) introduced a model called D-PACK to detect anomalies in the network traffics by using binary representations. By transforming the packet bytes into binary representations and the train the CNN, D-PACK achieved a high precision but limitations in the model flexibility and obfuscation techniques. Future work suggests that D-PACK could benefit from more sophisticated aggregation techniques, dynamic parameters adjustments and additional models to improve the malware detection.

2.3 Compression methods using machine learning

Berezkin *et al*. (2022) explored the autoencoders with feed-forward neural networks. This method transforms the binary vectors into BCH codewords that are later utilized to train the autoencoder; this way the bits transmitted are reduced. The study shows that a feed-forward neural autoencoder can efficiently compress the data and at the same time maintain high accuracy when it is reconstructed, making it ideal for communication systems that require a real-time data stream. But one of the biggest limitations is due to the complexity of the error-correcting algorithm and the need for extensive data training. Even when the algorithm improves the data compression mechanism, the algorithm can also incorporate computational overhead and increase the latency, and this, over time, can evolve into a problem.

Goyal *et al*. (2018) used RNN with combined arithmetic codification to compress sequential data, and this included text and a genomic dataset. In the research, DeepZip is used to estimate the probability to predict the next symbol in the data stream, then uses the arithmetic coding to compress the information. This approach outperforms heuristic compression methods like Gzip and achieves a great compression for synthetic datasets. However, the framework presents a computational overhead and the training process for large datasets becomes complex. Also, there is an additional difficulty for the decoding process because of the dependency on the previous decoded symbols.

Then, Song *et al*. (2019) used a CNN model to classify 16 types of compression algorithms for data bitstream. The proposed CNN uses a 5-layer model with a Spatial Pyramid Pooling SPP layer to handle the variable inputs (He *et al*., 2015). This way, the data can be transformed into a fixed-length vector for further classification. The study showed that the CNN model outperforms the heuristics compression methods like Support Vector Machines SVM in terms of precision classification, achieving 98.54% against the 63.84% of the SVM model. Nevertheless, a drawback of the solution is a limited model to treat different text files, which requires further refinement. Another is that the complexity of the model presents a computational overhead which can lead to low performance in real-time applications. A third is that extensive training data is needed to ensure accuracy across the different compression methods.

3 Research Methodology

Adaptative compression classifiers analyses the patterns found in grayscale images and then determine what is the best compression method for a specific file. This technique can be used

to reduce the space needed to store files by finding the compression mechanism that offers the highest compression ratio. In this study, a CNN-based algorithm is used to recognize patterns in grayscale images created from files in order to predict the best compression algorithm.

3.1 Process workflow

Figure 1 presents the workflow to process the data gathered. Different files have been collected from different projects on Kaggle. Those files are free audio files[2], PDFs[3] and images[4] that have been uploaded for multiple varied uses. In this case, as real information is needed, this information is used as a dataset. Then the more than 72k files will be compressed in different compression algorithms manually. This way, the best compression algorithm will be determined, and each file will be labelled according to the compression algorithm that offers the best compression ratio. Once the original files and the compression algorithm offering the best compression ratio for each file are identified, each file will be transformed into a grayscale image representation.

Once the grayscale image representation of the file and the label for each file are obtained, the CNN will be trained with the grayscale images. The convolutional layers will extract features and patterns from the input images that highlight different aspects of the file. Finally, when the model is fully trained, the accuracy of the model will be evaluated, testing the model's capacity to find the compression algorithm that offers the best compression ratio.

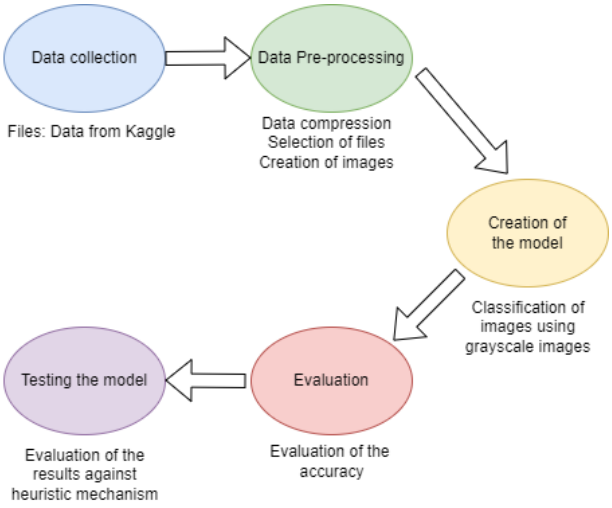

Figure 1: CNN compression methodology workflow

[2] https://www.kaggle.com/datasets/ttahara/birdsong-resampled-train-audio-00
https://www.kaggle.com/datasets/ttahara/birdsong-resampled-train-audio-01
https://www.kaggle.com/datasets/ttahara/birdsong-resampled-train-audio-02
https://www.kaggle.com/datasets/ttahara/birdsong-resampled-train-audio-03
https://www.kaggle.com/datasets/ttahara/birdsong-resampled-train-audio-04

[3] https://www.kaggle.com/datasets/paultimothymooney/cvpr-2019-papers

[4] https://www.kaggle.com/datasets/miljan/stanford-dogs-dataset-traintest/data

4 Design Specification

Google Colab: Colab is a convenient choice for scalability that offers access to GPU and TPU that can handle demanding computational tasks when there are not more options (AWS Sage maker or physical PC with GPU). This cloud-based environment allows to train and run large models without the need of expensive hardware. Despite the limitations like session time limits, shared resources and the need for optimization, Colab allows users to scale tests without the need to acquire expensive hardware.

Convolutional neural network: This is a deep learning model that works well for tasks that need to recognize patterns in images. This kind of deep learning architecture is very effective in multiple domains like object detection, image classification, and natural language processing when the data is previously pre-processed. In this case, the model will be used to find patterns in the images and predict the compression technique that offers the best compression ratio.

Also, a dataset of grayscale images generated from different files downloaded from Kaggle will be used. The output will be a trained model that will be stored for future inferences. The data will be split into training and validation datasets. In the convolutional neural network, various layers will be set up, combining Conv2D, MaxPooling2D, Flatten, and Dense layers.

Algorithm 1 Image Classification with CNN

Input: CSV file with image names and labels: *csv_file*
Input: Directory containing images: *image_dir*
Output: Trained CNN model: *image_classification_model*
Load and Preprocess Data:
Read CSV file and extract image paths and labels.
Encode and convert labels to categorical format.
Split data into training and validation sets.
Load and preprocess images (resize and normalize).
Build and Compile Model:
Initialize Sequential model.
Add Conv2D, MaxPooling2D, Flatten, and Dense layers.
Compile model with 'adam' optimizer and 'categorical_crossentropy' loss.
Train and Save Model:
Train model with training data.
Save model as *image_classification_model*.

Figure 2: Training CNN model

The model is compiled using the Adam optimizer and the categorical cross entropy using TensorFlow. After the model is trained and stored, the results will be tested with the training dataset, and the accuracy of the results will be checked.

5 Implementation

The research project uses the python language in Google Colab. This is an online platform that helps us to write and run Python code using the browser, by using virtual machines in the

background. This configuration offers multiple advantages in an interactive interface. Those are some of the advantages.

1. Free access to GPU depending on the load.
2. Collaborative environment
3. Support of ASIC to run Machine learning model faster.
4. There is no need for previous configuration

The environment helps us to implement Convolutional neural models efficiently by using the Nvidia accelerators to train models faster. Some python libraries have been used to train the models as they provide the functions to perform the data pre-processing, splitting and compilation of the model before the training. 3 different datasets have been used that gather documents, audio files and images from public datasets.

The dataset that is used in CNN compression classification is a combination of multiple open-source datasets, converted into grayscale images to feed the ML model. The files were loaded into Google drive for persistency, then the folders were iterated in order to get all the files names and then converted into images. The conversion process is detailed in the following algorithm.

Algorithm 2 Convert binary files to images

```
set input_directory
set output_directory
for each file in input_dir and subdirectories do
    open input_file in binary read mode
    read binary data as pixel values
    calculate image dimensions
    create and resize image 48px/120px
    save image to output_image
end for
```

Figure 3: Conversion of files into grayscale images

Those files were compressed using different compression algorithms to get the best compression algorithm for each file. Then later a CSV file was created which contained the name of the original file and the compression algorithm that offered the highest compression ratio. Once the CSV file containing the filenames and compression ratios was prepared, the original files were converted into grayscale binary images. The data was then split into training and testing datasets, with 80% of the dataset used for training and 20% for testing.

The process was done in Python using the Python Imaging Library (PIL); a system that can create an image by using a binary data stream from the file. The width and height of the image is calculated based on the size of the data stream. The image that results from this process is stored in PNG format. Now that the labels for the images have been created in the CSV file and the images have been prepared to feed the CNN, the model can be trained. As part of the

preparation for training, the categories—specifically the compression algorithm from the CSV file—must be converted into numerical values and then into one-hot vectors that represent the corresponding value of each category.

The next step is to load and pre-process the images before using them in the CNN model. The TensorFlow library was used for this purpose. The image is selected from the CSV file, as it contains the full name of the image, and is loaded in Python, converted into a NumPy array, and finally normalized. To make the values suitable for the CNN, the values must first be normalized. Before the model is compiled, the array must be reshaped into four dimensions, as neural networks (especially convolutional neural networks) expect a 4D input consisting of batch size, height, width, and channels. In this case, since grayscale images are being used, the number of channels is 1. To make the array compatible with the CNN, the addition of the extra dimension is necessary. Then the compilation of the CNN model is shown in the Figure 4 where the model is compiled using 3 Conv2D layers, 3 MaxPooling layers, a flattened layer that transforms the 3D output into a 1D vector. Finally, two dense layers, the last having the same number of neurons as the number of classes.

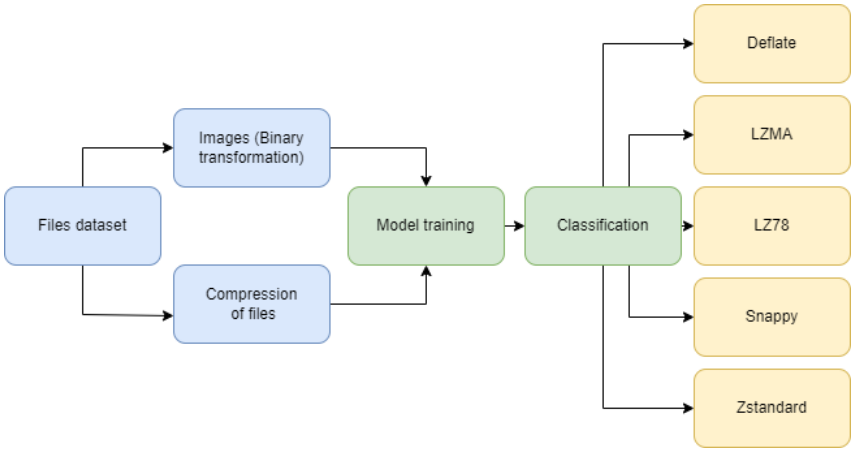

Figure 4: CNN compression prediction model

6 Evaluation

In this section, the different experiments conducted in this paper are discussed. Two models were trained with different grayscale image resolutions. The images were rescaled to 48x48 pixels and 120x120 pixels, and they were used in 3-layer and 4-layer CNNs.

6.1 Experiment / 3-layered CNN & Image resolution 48 pixels

In the first experiment, the 3-layered model was evaluated using 48px grayscale images and 30 epoch. This first approach was to have a balance between complexity and computational efficiency. A small image size was selected in order to reduce the computational load and get faster training times. This model was suitable for simple image recognition, where the objective was to capture essential features to avoid overfitting. The training accuracy for this model is

96.6%, meaning that the model performed well. A detailed analysis of the model is provided in the confusion matrix.

	Precision	Recall	F1-Score	Support
Deflate (0)	0.97	0.95	0.96	296
LZ78 (1)	0.15	0.01	0.03	279
LZMA (2)	0.93	0.99	0.96	4192
Snappy (3)	0.99	0.99	0.99	1788
Zstandard (4)	0.99	0.99	0.99	6318
Accuracy			0.97	12873
Macro avg.	0.81	0.79	0.79	12873
Weighted avg.	0.95	0.97	0.96	12873

Table 1: Classification report 3L-CNN 48px

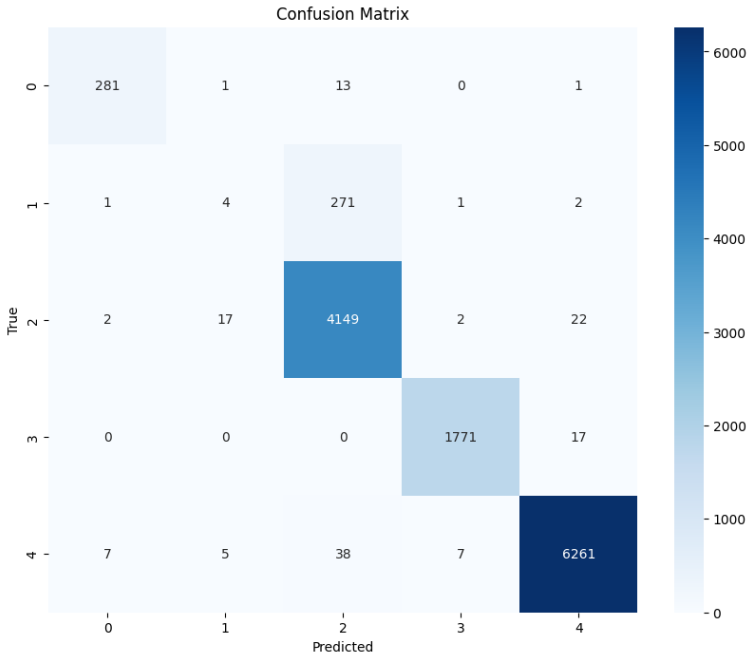

Figure 5: Confusion matrix 3L-CNN 48px

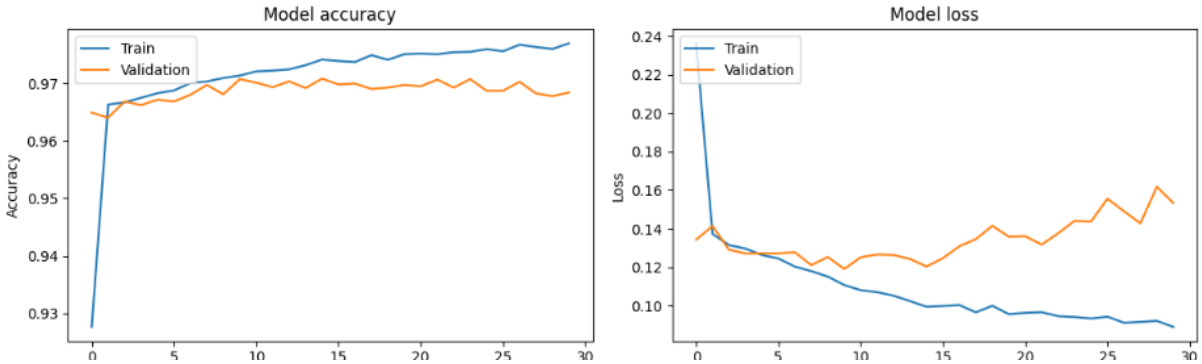

Figure 6: Model accuracy & lost 3L-CNN 48px

6.2 Experiment / 4-layered CNN & Image resolution 48 pixels

The second experiment was done with a 4-layered CNN with 48x48 grayscale images and 30 epoch. The objective of adding an extra layer was to capture more complex patterns and characteristics from the images that a simple model could have not captured. The increase of the layer's model can provide a deeper feature extraction, resulting in an improvement of the model's ability to differentiate between subtle variation in the dataset. But the difference between the 3-layered and 4-layered CNN was almost negligible.

	Precision	Recall	F1-Score	Support
Deflate (0)	0.98	0.94	0.96	296
LZ78 (1)	0.21	0.08	0.11	279
LZMA (2)	0.93	0.98	0.96	4192
Snappy (3)	1.00	0.99	0.99	1788
Zstandard (4)	0.99	0.99	0.99	6318
Accuracy			0.97	12873
Macro avg.	0.82	0.80	0.80	12873
Weighted avg.	0.96	0.97	0.96	12873

Table 2: Classification report 4L-CNN 48px

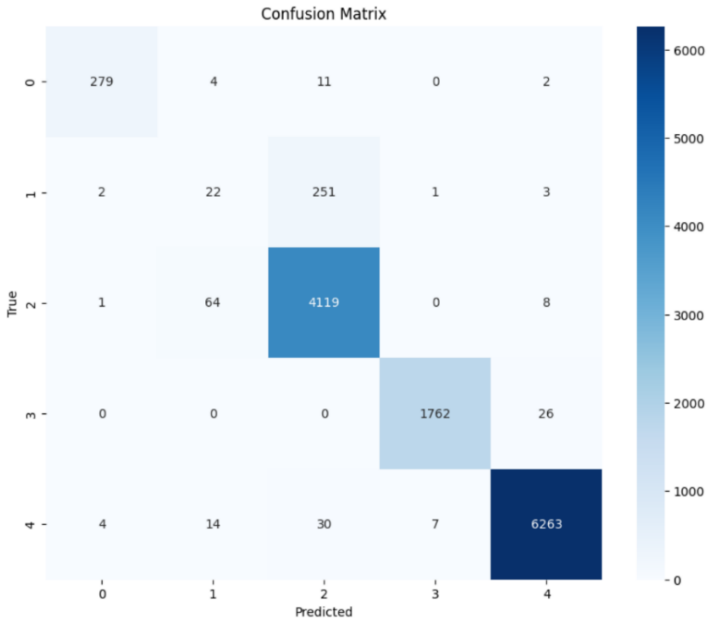

Figure 7: Confusion matrix 4L-CNN 48px

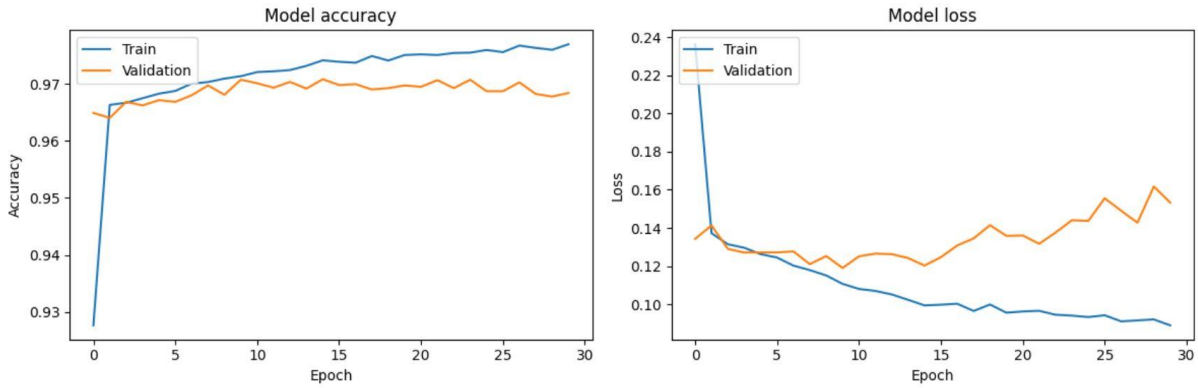

Figure 8: Model accuracy & lost 4L-CNN 48px

6.3 Experiment / 3-layered CNN & Image resolution 120 pixels

The third experiment, the model was repeated, but with images in higher resolution. Images in higher resolution were chosen - 120 pixels – to provide more detailed information that allowed the model to learn from characteristics and patterns that are not evident in low resolution images. The 3-layered architecture maintained the balance between complexity and training time considering the computational requirements for bigger images. The results showed that a higher resolution has a higher loss in the validation dataset with 34.81%. The confusion matrix shows that there is not a noticeable difference between 48px and 120px using the same CNN model.

	Precision	Recall	F1-Score	Support
Deflate (0)	0.97	0.94	0.95	296
LZ78 (1)	0.09	0.02	0.03	279
LZMA (2)	0.93	0.99	0.96	4192
Snappy (3)	0.99	1.00	1.00	1788
Zstandard (4)	1.00	0.99	1.00	6318
Accuracy			0.97	12873
Macro avg.	0.79	0.79	0.79	12873
Weighted avg.	0.95	0.97	0.96	12873

Table 3: Classification report 3L-CNN 120px

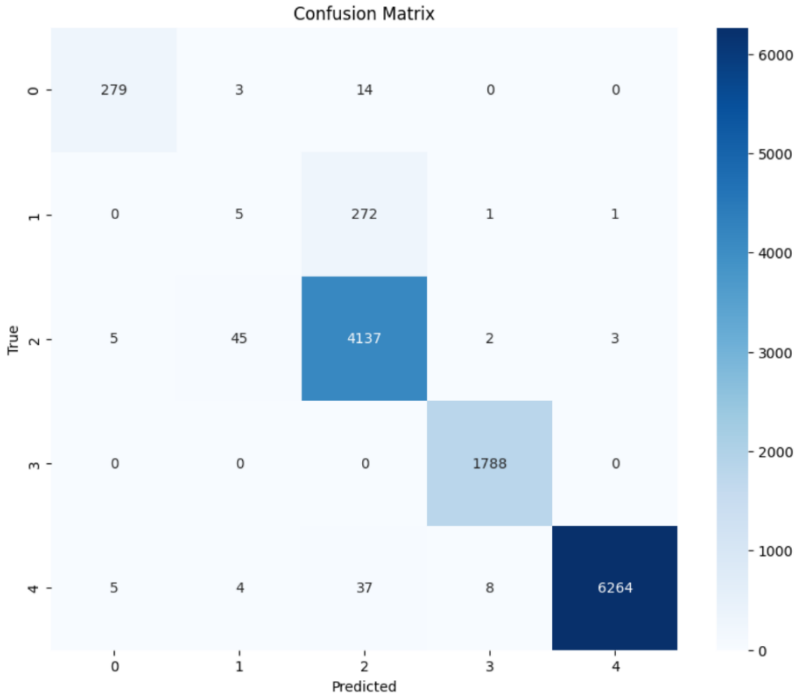

Figure 9: Confusion matrix 3L-CNN 120px

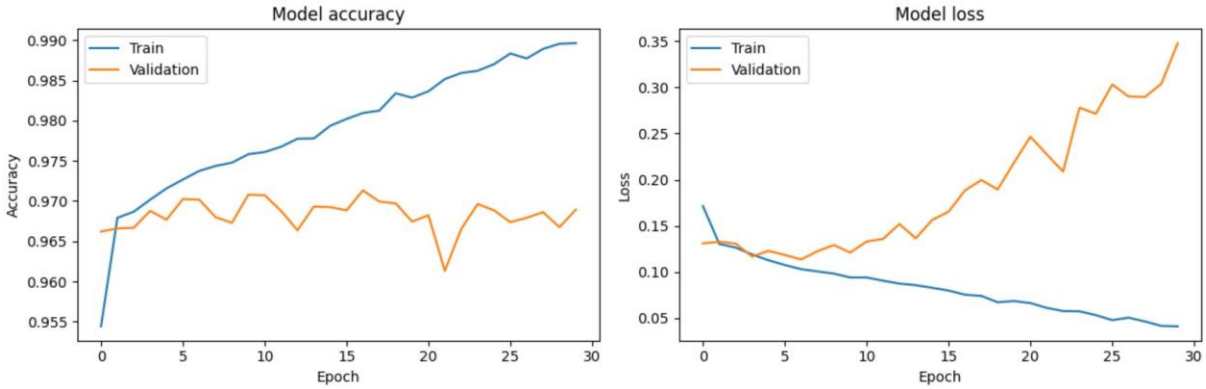

Figure 10: Model accuracy & lost 3L-CNN 120px

6.4 Experiment / 4-layered CNN & Image resolution 120 pixels

The fourth experiment involved 4-layered CNN with 120px images. This configuration aimed to combine the advantages of higher resolution images and a more complex CNN model. This configuration performed better than 3-layered 120px images with 34% test lost, but not as good as 3-layered 48px with 15.33% test lost and 4-layered 48px 13.68% test lost.

	Precision	Recall	F1-Score	Support
Deflate (0)	0.98	0.95	0.96	296
LZ78 (1)	0.14	0.04	0.06	279
LZMA (2)	0.93	0.98	0.95	4192
Snappy (3)	0.99	1.00	1.00	1788
Zstandard (4)	1.00	0.99	0.99	6318
Accuracy			0.97	12873
Macro avg.	0.81	0.79	0.79	12873
Weighted avg.	0.96	0.97	0.96	12873

Table 4: Classification report 4L-CNN 120px

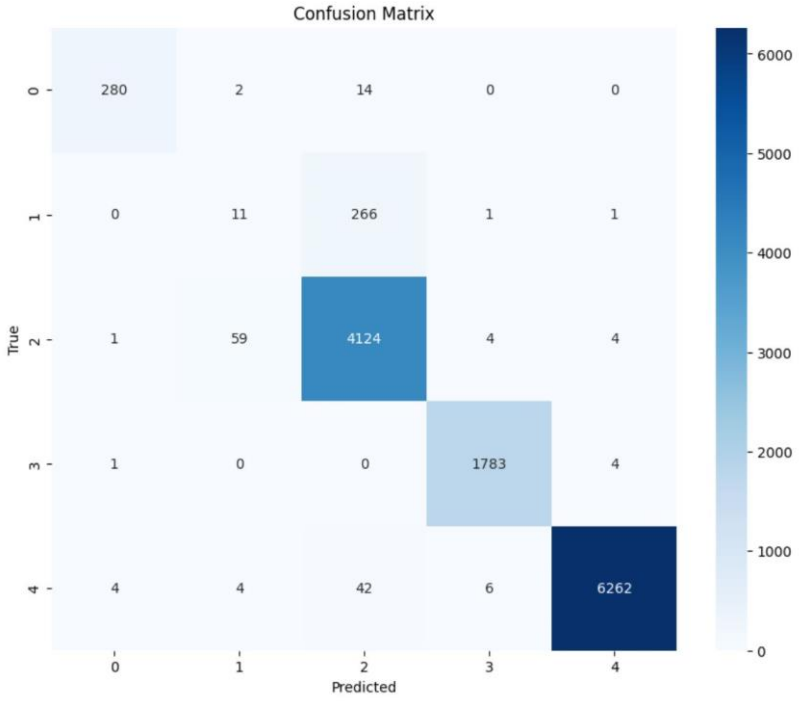

Figure 11: Confusion matrix 4L-CNN 120px

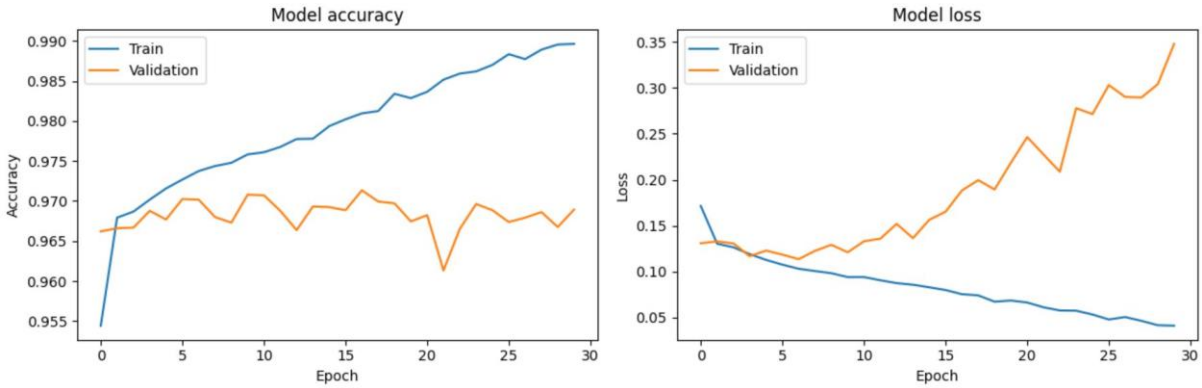

Figure 12: Model accuracy & lost 4L-CNN 120px

6.5 Discussion

To predict the compression method that offers the best compression ratio, a CNN model was implemented. Using 3-layered and 4-layered CNN models and different image sizes - 48px and 120px respectively- with 5 categories – DEFLATE, LZMA, LZ78, Snappy and Zstandard- it was proven that inferring the compression method was possible. Despite using higher image resolutions, the increase in accuracy was insignificant. Furthermore, the increase of layers in the model showed a negligible reduction in the accuracy as well.

A 3 and 4 layered CNN model was implemented to predict the best compression method for each file, from .txt, .pdf, .wav to .jpg. The prediction of DEFLATE, LZMA, LZ78, Snappy and Zstandard compression methods was assessed using CNN models. The accuracy when using 48x48 pixel images was 96.83% for the 3-layered CNN model and 96.67% for the 4-layered CNN model. When using 120x120 pixel images, the accuracy was 96.89% for the 3-layered CNN model and 96.79% for the 4-layered CNN model. The categorization of different compression methods helps to save time and storage space by choosing the right compression method based on the patterns found in the file.

7 Conclusion and Future Work

Ways to optimize the data storage have been widely studied, as the data that is generated everyday can surpass the data storage that most organizations have. This research aimed to explore the different ways to optimize storage by finding the compression method that offers the best compression ratio based on the characteristics of individual files. The experiments conducted proved that the CNN model is able to predict compression methods based on graphical representations of a file as grayscale binary images but are also able to infer the compression mechanisms that offer the best compression ratio. The CNN model was able to infer compression methods for a file from different categories - DEFLATE, LZMA, LZ78, Snappy and Zstandard - showing that CNN models are capable of identifying patterns in the graphical representation of images. Researchers have been using multiple machine learning algorithms to improve the accuracy of predictions for the efficient storage of files.

This study used open-source datasets that do not vary too much. One of them is a collection of pictures of dogs that may have similar features, and as such could bias the model. Another of the datasets is a compilation of audios of bird songs that could also bias the model. Future work may employ more variety in the dataset to prevent bias in the model when it comes to learning from patterns in the files.

References

Aich, A., Krishna, A., Akhilesh, V. and Hegde, C. (2019) ‘Encoding web-based data for efficient storage in machine learning applications’, in *2019 Fifteenth International Conference on Information Processing (ICINPRO)*. Bengaluru, India, 20-22 December 2019, pp. 1–6. doi: 10.1109/ICInPro47689.2019.9092264

Akash, A. R., Ahn, B., Jenkins, A., Khot, A., Silva, L., Tavares-Vengas, H. and Kim, T. (2023) 'Quantum convolutional neural network-based online malware file detection for smart grid devices', in *2023 IEEE Design Methodologies Conference (DMC)*. Miami, FL, USA, 24-26 September 2023, pp. 1–5. doi: 10.1109/DMC58182.2023.10412597

Berezkin, A., Slepnev, A., Kirichek, R., Kukunin, D. and Matveev, D. (2022) 'Data compression methods based on neural networks', in *5th International Conference on Future Networks & Distributed Systems, ICFNDS 2021*. Dubai, United Arab Emirates, 15-16 December 2021, pp. 511–515. doi: 10.1145/3508072.3508177

Bhardwaj, C. and Garg, H. (2023) 'An approach for enhancing data storage capacity in quick response code using zip compression technique', in *2023 International Conference on Artificial Intelligence and Smart Communication (AISC)*. Greater Noida, India, 27-29 January 2023, pp. 520–524. doi: 10.1109/AISC56616.2023.10085559

Chien, A. A. and Karamcheti, V. (2013) 'Moore's Law: The first ending and a new beginning', *Computer*, pp. 48–53. doi: 10.1109/MC.2013.431

Cui, Z., Xue, F., Cai, X., Cao, Y., Wang, G. and Chen, J. (2018) 'Detection of malicious code variants based on deep learning', *IEEE Transactions on Industrial Informatics*, 14(7), pp. 3187–3196. doi: 10.1109/TII.2018.2822680

Deutsch, P. (1996) *RFC1951: DEFLATE compressed data format specification version 1.3*. Available at: https://doi.org/10.17487/RFC1951 [Accessed 7 August 2024].

Devarajan, H., Kougkas, A. and Sun, X.-H. (2019) 'An intelligent, adaptive, and flexible data compression framework', in *19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. Larnaca, Cyprus, 14-17 May 2019, pp. 82–91. doi: 10.1109/CCGRID.2019.00019

Dong, W., Guan, J., Yang, L., Yu, B. and Li, W. (2021) 'NOCAQF: An efficient nonlinear online data compression algorithm', in *2021 International Conference on Intelligent Computing, Automation and Systems (ICICAS)*. Chongqing, China, 29-31 December 2021, pp. 20–24. doi: 10.1109/ICICAS53977.2021.00012

Emon, A. D., Roy, P. C., Singh, L. D. and Saha, A. K. (2023) 'Combining Haar transform and row-column directional RLE in image compression', in *2023 Eighth International Conference on Informatics and Computing (ICIC)*. Manado, Indonesia, 8-9 December 2023, pp. 1–6. doi: 10.1109/ICIC60109.2023.10382031

Goyal, M., Tatwawadi, K., Chandak, S. and Ochoa, I. (2018) *DeepZip: Lossless data compression using recurrent neural networks*. Available at: https://doi.org/10.48550/arXiv.1811.08162 [Accessed 7 August 2024].

He, K., Zhang, X., Ren, S. and Sun, J. (2015) 'Spatial pyramid pooling in deep convolutional networks for visual recognition', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(9), pp. 1904–1916. doi: 10.1109/TPAMI.2015.2389824

Hidayat, T., Nurcholis, M. T., Nugroho, A. and Santoso, J. D. (2023) 'Critical understanding performance of Huffman and Lempel-Zip to pattern audio data 16-bit', in *2023 6th*

International Conference of Computer and Informatics Engineering (IC2IE). Lombok, Indonesia, 14-15 September 2013, pp. 7–12. doi: 10.1109/IC2IE60547.2023.10331457

Hwang, R.-H., Peng, M.-C., Huang, C.-W., Lin, P.-C. and Nguyen, V.-L. (2020) 'An unsupervised deep learning model for early network traffic anomaly detection', *IEEE Access*, 8, pp. 30387–30399. doi: 10.1109/ACCESS.2020.2973023

Jeong, Y.-S., Mswahili, M. E. and Kang, A. R. (2023) 'File-level malware detection using byte streams', *Scientific Reports*, 13, 8925. doi: 10.1038/s41598-023-36088-2

Jonnala, Y. D., Mahajan, V. S., Menon, D., Kothakapu, S. R. and Chandamollu, S. R. (2023) 'Malware detection using binary visualization and neural networks', *E3S Web of Conferences*, 391, 01107. doi: 10.1051/e3sconf/202339101107

Kalash, M., Rochan, M., Mohammed, N., Bruce, N. D. B., Wang, Y. and Iqbal, F. (2018) 'Malware classification with deep convolutional neural networks', in *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. Paris, France, 26-28 February 2018, pp. 1–5. doi: 10.1109/NTMS.2018.8328749

Kalyan, E. V. P., Adarsh, A. P., Reddy, S. S. L. and Renjith, P. (2022) 'Detection of malware using CNN', in *2022 Second International Conference on Computer Science, Engineering and Applications (ICCSEA)*. Gunupur, India, 8 September 2022, pp. 1–6. doi: 10.1109/ICCSEA54677.2022.9936225

Kato, T., Yamagiwa, S. and Wada, K. (2023) 'Toward parallelization technique for stream-based lossless data compression', in *2023 IEEE International Conference on Big Data (BigData)*. Sorrento, Italy, 15-18 December 2023, pp. 2667–2672. doi: 10.1109/BigData59044.2023.10386184

Mavric, S. H. T. and Yeo, C. K. (2018) 'Online binary visualization for PDF documents', in *2018 International Symposium on Consumer Technologies (ISCT)*. St Petersburg, Russia, 11-12 May 2018 pp. 18–21. doi: 10.1109/ISCE.2018.8408906

Moffat, A. (2020) 'Huffman coding', *ACM Computing Surveys*, 52(4), pp. 1–35. doi: 10.1145/3342555

Mukherjee, K., Shah, R., Saini, S., Singh, K., Kesarwani, H., Barnwal, K. and Chauhan, A. (2023) 'Towards optimizing storage costs on the cloud', in *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. Anaheim, CA, USA, 3-7 April 2023, pp. 2919–2932. doi: 10.1109/ICDE55515.2023.00223

Reinsel, D., Gantz, J. and Rydning, J. (2017) *Data age 2025: The evolution of data to life-critical.* Available at: https://www.seagate.com/files/www-content/our-story/trends/files/Seagate-WP-DataAge2025-March-2017.pdf [Accessed 7 August 2024].

Sarkar, S. J., Sarkar, N. K. and Banerjee, A. (2016) 'A novel Huffman coding based approach to reduce the size of large data array', in *2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT)*. Nagercoil, India, 18-19 March 2016, pp. 1–5. doi: 10.1109/ICCPCT.2016.7530355

Song, H., Kwon, B., Lee, S. and Lee, S. (2019) ‘Dictionary based compression type classification using a CNN architecture’, in *2019 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*. Lanzhou, China, 18-21 November 2019, pp. 245–248. doi: 10.1109/APSIPAASC47483.2019.9023258

Vinayakumar, R., Alazab, M., Soman, K. P., Poornachandran, P. and Venkatraman, S. (2019) ‘Robust intelligent malware detection using deep learning’, *IEEE Access*, 7, pp. 46717–46738. doi: 10.1109/ACCESS.2019.2906934

Wang, W., Zhu, M., Zeng, X., Ye, X. and Sheng, Y. (2017) ‘Malware traffic classification using convolutional neural network for representation learning’ in *2017 International Conference on Information Networking (ICOIN)*. Da Nang, Vietnam, 11-13 January 2017, pp. 712–717. doi: 10.1109/ICOIN.2017.7899588

Welch, T. A. (1984) ‘A technique for high-performance data compression’, *Computer*, 17, pp. 8–19. doi: 10.1109/MC.1984.1659158

Xiao, X. and Yang, S. (2019) ‘An image-inspired and CNN-based Android malware detection approach’, in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. San Diego, CA, USA, 11-15 November 2019, pp. 1259–1261. doi: 10.1109/ASE.2019.00155