# Take Me One More: Efficient Clustering Compression using Inter-Frame Encoding

Ignacio Brasca

February 19, 2024

## Abstract

*This paper introduces a novel intraframe data encoding compression algorithm aimed at significantly reducing storage requirements for quick-capture devices, such as CCTV cameras and smartphones. By leveraging Huffman coding alongside Discrete Cosine Transform (DCT) techniques, the proposed algorithm offers an efficient solution for minimizing data file sizes. Unlike traditional image compression methods that treat each frame independently, our approach utilizes inter-frame encoding to exploit temporal redundancies between consecutive frames, achieving higher compression ratios while maintaining image quality. The algorithm's workflow, implementation details, and its application in dynamic cluster formation and memory optimization are discussed, alongside case studies demonstrating its effectiveness in various real-world scenarios.*

**Keywords:** *intraframe data encoding, compression algorithm, Huffman coding, Discrete Cosine Transform (DCT), inter-frame encoding, dynamic cluster formation, memory optimization, data compression*

## 1 Introduction

In recent times, there has been a surge in the need for efficient data compression techniques due to the rapid expansion of quick-capture devices like CCTV cameras, smartphones, and other portable storage-constrained devices. In response to this challenge, there's formulated in this essay a potential concept for an intraframe data encoding compression algorithm that allows reduction of storage requirements. This proposed algorithm leverages the power of Huffman coding and Discrete Cosine Transform (DCT) techniques, which are highly effective in minimizing the size of data files.

Using Huffman coding as a form of entropy encoding technique that replaces a sequence of symbols with shorter codes based on their probability distribution we can save storage in benefit of redundancy. In contrast, DCT is a mathematical process that decomposes an image into a series of frequency components and helps reduce the redundancy as well, in pictures as well as video frames (pictures in a time-series). Combining these two powerful techniques, the proposed algorithm can offer significant storage savings for quick-capture devices.

The method here proposed allow us to understand not only one picture as a whole but an ecosystem of similar pictures based on a timestamp technique. Enabling us the categorization based a on threshold generated after a difference threshold computed using the inception frame. This technique has been in used in the past for video compression, but not for single images. This essay will provide an overview of the inter-frame encoding algorithm, its workflow, implementation details, and case studies to demonstrate its effectiveness.

## 2 Background

Traditional image compression techniques, such as JPEG and PNG, rely on compressing individual frames independently. While these methods are effective for compressing individual images, they may not be optimal for sequences of similar images when those are being captured by real-time devices or smart phones. inter-frame encoding help us to overpass this limitation by exploiting the temporal redundancy between consecutive frames in a sequence. By storing only the differences between frames, inter-frame encoding can achieve higher compression ratios while preserving image quality.

## 2.1 Formal Definitions

1. **Inception Frame** ($I$)**:** The first frame in a sequence of images.
2. **Beam** ($beam_{n,m}$)**:** A data point in a frame that can be modified using a set of operations.
3. **Difference Threshold** ($N$)**:** The maximum difference allowed between consecutive frames to be considered similar.
4. **Cluster Frames** ($K$)**:** A group of similar images within the difference threshold.

## 2.2 Sequence Compression Against a Set of Frames

To introduce the underlying techniques, we can define a set of frames $K$ as a set of frames that are part of a sequence of images. Ideally, a set of images computed with this technique will use a limitation of $N$ frames, where $N$ is the amount of frames we want to compute the difference against in a timeframe $t$.

For a set of frames as

$$K = \{X_1, X_2, X_3\}$$

As soon as we cluster them down, we can define a set of expressions we can perform at the frame level as

$$X_k = X_n \circ X_m$$

whereas we can also combine these to produce a new $X_k$ frame based on the statement generated after the application of expressions.

Operations $\circ$ can be computed at runtime, where a new piece of information appears in the set of $K$ frames, and this will be included in the set of new data points $K$.

A data point part of $K$ will be called a *beam* and will be part of a set of $beams_{n,m}$ where $n, m$ are the dimensions restrictions of the matrix containing all the beams available in the source of information presented as a frame in $K$.

## 2.3 Restrictions

There are a few restrictions we need to take into account when computing the difference between frames in a sequence of images:

1. We need to compute always the same amount of information in a finite combination of RGB values; this means each frame should contain at least $\max(X_n)$ pixels, where $X_n$ is the frame with the highest amount of color depth (information per pixel).
2. $len(K)$ should always be higher than 1 where $K$ is the set of frames we want to compute the difference against.
3. $X_n \circ X_m$ should always output a valid matrix as a result.

Operations available at the *beam* level are placed uttermost in the front of any pixel transformation technique available, always using the inter-frame perspective of the information account.

In case multiple beams $x_{i,j}$ would be intertwined, we can define a net set of operations to establish what or which part of the frames we want to modify during which amount of time.

## 2.4 Inception frame

Currently, compression against a set of frames is being computed against a linear relationship where $K$ frames are only compressed after $N$ times during a $t$ span. Presented here is the utilization of a global difference matrix called $K_{global}$ where we track against a set of values used on the same matrix as it was a memory registry.

Starting from frame $n$, we can compute and utilize the same amount of $beams_{n,m}$ already stored from the initial inception frame.

From an *inception* frame, we should always compute the difference against the same amount of frames, where $n$ is the frame we want to compute the difference against and then evaluate if the difference threshold is small enough to be considered part of the same cluster.

Operations $\circ$ can be computed at runtime, against a set of $beams_{n,m}$ where $n, m$ are the dimensions restrictions of the matrix containing all the beams available in the source of information presented as a frame in $K$. This information is beneficial in order to calculate not only the difference between frames but also to compute inception frame to obtain the same set of frames in a different time span $t$.

### 2.5 Reverting frames

In order to revert an operation $\circ$, we can simply apply the inverse operation $\circ^{-1}$ to the frame $X_k$ and obtain the original frame $X_n$ or $X_m$. This operation is only possible if the original frame $X_n$ or $X_m$ is available in the set of frames $K$.

## 3 Architecture

The proposed two-step architecture for processing frames in a cluster of pictures is designed to efficiently analyze a series of images while minimizing redundancy and memory usage. The first part involves dynamically defining image clusters, where each cluster consists of images with similar content up to a specified threshold. Images exceeding the threshold are considered distinct and initiate a new cluster. The second part focuses on optimizing memory and efficiently matching differences between frames to avoid processing redundant information.

The two steps in the proposed architecture can be named as follows:

1. Dynamic Cluster Formation (DCF)
2. Memory Optimization and Inter-frame Matching (MOIM)

### 3.1 Dynamic Cluster Formation

The technique here described (Dynamic Cluster Formation) or from now on: DCF, is an step that continuously checks for changes in the input stream of images based on a initial timestamp. When a difference greater than a defined threshold is detected, the current cluster is considered complete, and a new cluster begins. This step ensures that similar images are grouped together, allowing for efficient inter-frame encoding.

The DCF step can be implemented using a variety of techniques, such as threshold-based clustering, k-means clustering, or hierarchical clustering. The choice of clustering technique depends on the specific requirements of the application and the nature of the input data. For images, we recommend the usage of a threshold-based clustering technique, where the difference between consecutive frames is compared against a user-defined threshold during a timeframe $t$.

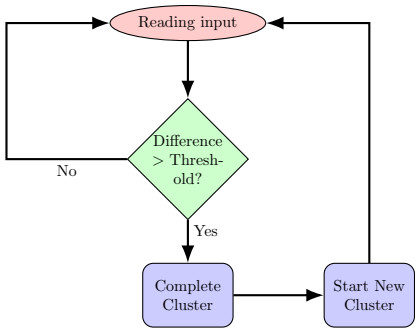


Figure 1: Dynamic Cluster Formation (DCF) Step Diagram

The DCF step is essential for efficiently grouping similar images into clusters, which can then be processed using the technique described in the next section.

### 3.2 Memory Optimization and Inter-frame Matching

The *MOIM* step aims to minimize redundant processing and memory usage. It matches differences between frames and avoids reprocessing frames with similar information. This optimization step is crucial for achieving high compression ratios and efficient processing. (View Figure 2)

Proposed here is the utilization of a global difference matrix called $K_{global}$, where we track against a set of values used on the same matrix as if it were a memory registry. This matrix will be used to compute the difference between frames and then evaluate if the difference threshold is small enough to be considered part of the same cluster.

Huffman coding and Discrete Cosine Transform (DCT) techniques can be applied to the differences between frames to achieve high compression ratios. This step also involves managing memory efficiently to minimize the computational overhead of processing differences (see Annex A for more information). In collaboration with the first step of this pipeline, *MOIM* ensures that the encoding algorithm can achieve high compression ratios while maintaining image quality.

# 4 Clustering Encoding

Applying the inter-frame encoding algorithm works by computing the difference between consecutive frames inside a cluster in a time sequence $t$. Let $K_t$ represent the $t$-th frame in the sequence, and $I_{t-1}$ represent the previous frame. The difference between $I_t$ and $I_{t-1}$ is computed as follows:

$$\Delta I_t = I_t - I_{t-1}$$

The difference $\Delta I_t$ is then stored using techinques expressed in the previous section along with any additional information required for reconstruction.

The inter-frame encoding algorithm can be applied to a wide range of applications, including video compression, medical imaging, and remote sensing. By exploiting temporal redundancy between consecutive frames, inter-frame encoding can achieve high compression ratios while maintaining image quality. As you can see in Figure 3 this process is repeated for the entire sequence of $K$ frames.

# 5 Performance Metrics

Performance rely on the reduction of redundancy and the minimization of memory usage. The following metrics can be used to evaluate the performance of the encoding algorithm:

1. Compression Ratio: The ratio of the original file size to the compressed file size.
2. Memory Usage: The amount of memory required to store the compressed metadata and reconstructed frames.
3. Processing Time: The time required to encode and decode the differences.
4. Image Quality: The visual quality of the reconstructed frames compared to the original frames.
5. Number of Clusters: The number of clusters formed during the dynamic cluster formation step.

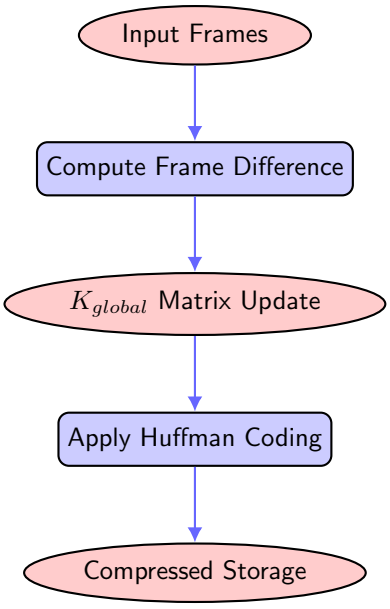


Figure 2: Memory Optimization and Inter-frame Matching (MOIM) Step Diagram

# 6 Proposed Workflow

The workflow for inter-frame encoding involves several steps:

1. Read base image ($I_0$) from disk starting from the cluster defined in DCF.
2. Iterate over the sequence of images until the end of the cluster.
3. Compute the difference between each consecutive frame using DCF.
4. Store the differences in a compressed metadata format using MOIM.

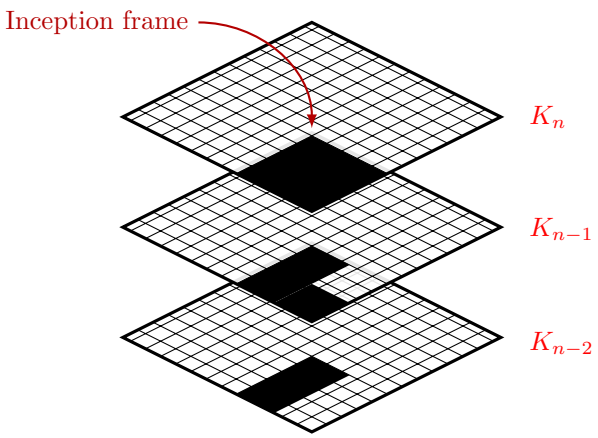


Figure 3: Inter-frame Encoding Diagram

5. Repeat the process for the entire sequence of images inside the storage utilized.

The following diagram illustrates the proposed workflow for inter-frame encoding:

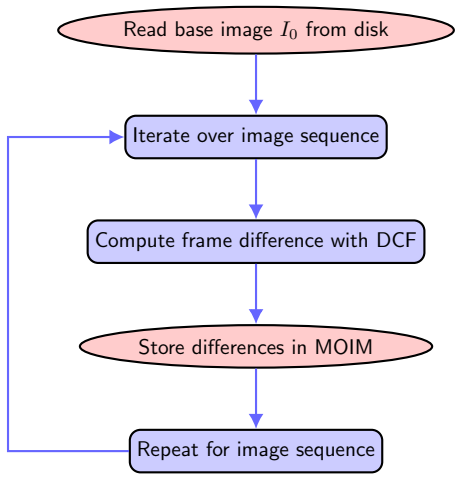


Figure 4: Proposed Workflow for Inter-frame Encoding

# 7 Implementation Details

Implementing inter-frame encoding requires careful consideration of several factors, including file formats, compression techniques, and computational complexity. Various file formats, such as JSON or binary formats, can be used to store image data and metadata efficiently. Additionally, compression techniques, such as run-length encoding or delta encoding, can further reduce storage requirements. It is essential to balance compression ratios with computational overhead to achieve optimal performance.

# 8 Case Studies

inter-frame encoding has been widely used in video compression standards such as MPEG and H.264. These standards leverage inter-frame encoding to achieve significant reductions in file size while maintaining high-quality video playback. Additionally, inter-frame encoding has applications in medical imaging, surveillance systems, and remote sensing, where storage efficiency is critical.

# 9 Conclusion

In conclusion, inter-frame encoding offers a powerful solution for compressing sequences of similar images efficiently. By exploiting temporal redundancy between consecutive frames, inter-frame encoding can achieve high compression ratios without sacrificing image quality. The algorithm's versatility and effectiveness make it an essential tool in various applications, from multimedia compression to medical imaging.

# 10 References


1. Z. Wang, D. Chanda, S. Simon, "Memory Efficient Lossless Compression of Image Sequences with JPEG-LS and Temporal Prediction."
2. J.-D. Lee, S.-Y. Wan, C.-M. Ma, R.-F. Wu, "Compressing Sets of Similar Images Using Hybrid Compression Model."
3. S. Ait-Aoudia, A. Gabis, A. Naimi, "Compressing Sets of Similar Images."
4. T. Wiegand, G. J. Sullivan, G. Bjøntegaard, A. Luthra, "Overview of the H.264/AVC Video Coding Standard," IEEE Transactions on Circuits and Systems for Video Technology, vol. 13, no. 7, July 2003.
5. T. Koga, Y. Iijima, K. Iinuma, T. Ishiguro, "Statistical Performance Analysis of an Interframe Encoder for Broadcast Television Signals," 1981.
6. H. Ban Choi, A. A. Trofimov, "Fast Test Zone Search Algorithm for Interframe Encoding," 2017.
7. B. Girod, A. Aaron, S. Rane, D. Rebollo-Monedero, "Distributed Video Coding," 2005.
8. E. Belyaev, "An Efficient Compressive Sensed Video Codec with Inter-Frame Decoding and Low-Complexity Intra-Frame Encoding," 2023.
9. D. Huffman, "A Method for the Construction of Minimum-Redundancy Codes," Proceedings of the IRE, vol. 40, no. 9, pp. 1098–1101, 1952.
10. J. Van Leeuwen, "On the construction of Huffman trees," in Proceedings of the 3rd Colloquium on Automata, Languages, and Programming (ICALP), pp. 382–410, 1976.

# A Annex A: Utilizing Huffman Coding in MOIM for Efficient Data Compression

## A.1 Introduction

Huffman coding, a popular method for lossless data compression, can significantly enhance the MOIM step by reducing the amount of data required to represent frame differences. This section explains the process and provides a practical example of its application.

## A.2 Huffman Coding in MOIM

The MOIM step, crucial for minimizing redundant data and optimizing memory usage, can benefit from Huffman coding by encoding the frame differences more compactly. Huffman coding achieves this by assigning variable-length codes to input characters, with shorter codes for more frequent characters.

### A.2.1 Example

Consider a simplified scenario where we have computed the differences between consecutive frames, resulting in a series of values. Given the frame differences:

4 1 2 4 4 2 3 1 4

Frequency of each difference value:

| Value | Frequency |
|---|---|
| 1 | 2 |
| 2 | 2 |
| 3 | 1 |
| 4 | 4 |

Applying Huffman coding to these differences:

| Value | Huffman Code |
|---|---|
| 4 | 0 |
| 1 | 10 |
| 2 | 110 |
| 3 | 111 |

Thus, the encoded sequence using Huffman codes would be:

0 10 110 0 0 110 111 10 0

This encoded sequence is significantly shorter than the original representation, demonstrating how Huffman coding can compress data efficiently.

## A.3 Conclusion

Incorporating Huffman coding into the MOIM step allows for a significant reduction in data size by efficiently encoding frame differences. This process not only saves storage space but also accelerates data processing and retrieval, making it a valuable technique for optimizing inter-frame matching and memory usage in video and image compression systems.