Video Compression Using Unsupervised Learning

Dharani Kumar S
B.Tech
dharani21039@iiitd.ac.in

Divyansh Gaba
B.Tech
divyansh21252@iiitd.ac.in

Dev Pratap Singh
BTech
dev21383@iiitd.ac.in

Dhruv Patel
M.Tech
dhruv23056@iiitd.ac.in

Rashi Khandelwal
MTech
rashi23072@iiitd.ac.in

Abstract

This project report investigates the use of classical machine learning techniques for video compression, aiming to reduce storage and transmission requirements while maintaining video quality. The study provides an overview of video compression and explores the adaptation of traditional machine learning algorithms to this context. It involves the collection and preprocessing of video datasets, feature extraction, and training classical machine learning models to optimize encoding parameters for various video segments. The project demonstrates the effectiveness of these models through rigorous evaluation and comparisons with existing compression standards, considering both quality and compression performance. It also addresses computational efficiency and real-time applicability. This research contributes to the field by offering a novel approach to video compression, potentially enhancing video streaming and storage efficiency and paving the way for future advancements in this domain.

1 Classical Machine Learning for Video Compression

1.1 Introduction

Video compression is the process of reducing the size of a video file without significantly sacrificing its quality. This is done by identifying and removing redundant information from the video. Video compression is important because it allows us to store and transmit videos more efficiently. It is also essential for many video applications, such as streaming video over the internet and storing videos on mobile devices.

There are two main types of video compression: lossless compression and lossy compression. Lossless compression algorithms reduce the size of a video file without losing any information. This is done by using techniques such as run-length encoding and Huffman encoding. Lossy compression algorithms reduce the size of a video file by permanently removing some of the information. This is done by using techniques such as quantization and decimation.

Classical machine learning algorithms can be used for video compression in a number of ways. For example, machine learning algorithms can be used to:

- Predict motion: Machine learning algorithms can be used to predict the motion of objects in a video sequence. This information can then be used to encode the video more efficiently.
- Identify redundancies: Machine learning algorithms can be used to identify redundant information in a video. This redundant information can then be removed to reduce the size of the video file.

- Optimize encoding parameters: Machine learning algorithms can be used to optimize the encoding parameters for a given video. This can help to achieve the best possible compression ratio without sacrificing quality.

In this project, we will explore the use of classical machine learning algorithms for video compression. We will propose a new approach to video compression that uses machine learning to improve the compression ratio and distortion. We will also evaluate the performance of our proposed approach on the given video datasets.

We hope that our work will contribute to the development of new and more efficient video compression algorithms.

1.2 Literature Review

Video compression is a critical technology for storing and transmitting videos efficiently. Classical machine learning algorithms have been shown to be effective for video compression in a number of ways, including motion prediction, redundancy identification, and encoding parameter optimization.

One of the most promising areas of research in video compression using classical machine learning is motion prediction. Motion prediction algorithms use machine learning to predict the motion of objects in a video sequence. This information can then be used to encode the video more efficiently. For example, a motion prediction algorithm can be used to predict that the background in a video sequence is not moving, and therefore only the moving objects need to be encoded.

Another promising area of research is redundancy identification. Redundancy identification algorithms use machine learning to identify redundant information in a video. This redundant information can then be removed to reduce the size of the video file. For example, a redundancy identification algorithm can be used to identify that a scene in a video sequence is repeated several times, and therefore only one copy of the scene needs to be encoded.

Finally, classical machine learning algorithms can also be used to optimize the encoding parameters for a given video. This can help to achieve the best possible compression ratio without sacrificing quality. For example, a machine learning algorithm can be used to determine the optimal quantization parameters for a video sequence.

Here are some existing studies on Video compression using machine learning:

1.2.1 Machine Learning for Video Compression: Macroblock Mode Decision [3]

Video compression is essential for storing and transmitting digital video efficiently, as uncompressed video requires a massive amount of data. This paper explores the use of machine learning to improve video encoding, particularly the mode decision process, which plays a crucial role in compression. Video encoding involves splitting images into blocks and deciding how to encode each block, such as INTRA-mode (JPEG-like) or INTER-mode (prediction-based). Macroblock mode decision is crucial for video compression. The paper introduces mathematical formulations for mode decision, including the use of a Lagrangian formalism to balance bits and distortion. Mode decision can be viewed as a classification problem where a classifier chooses the encoding mode based on a block's characteristics. The paper discusses the importance of considering the specific cost for each block when making mode decisions and how to incorporate these costs into the decision process.

Machine learning techniques, including linear classifiers, decision trees, and neural networks, can be applied to improve mode decision in video encoding. The primary concern is speed during online encoding. Neural networks are versatile but may be slower than linear classifiers. The authors also used decision trees and modified OC1 software to train classifiers, considering cost optimization. The paper tested machine learning techniques on the XviD MPEG-4 encoder and found that they improved coding efficiency. Decision trees and neural networks had higher classification quality but required more computation. The research demonstrates that machine learning can enhance video encoding's mode decision process, with the potential to play a more significant role in video compression. Adaptive techniques and online training are seen as future directions.

1.2.2 Adaptive Keyframe Extraction Using Unsupervised Clustering[5]

This paper discuss Clustering as a powerful technique used in various disciplines such as Pattern Recognition , Speech Analysis , and Information Retrieval, etc. In an unsupervised clustering based approach was introduced to determine key frames within a shot boundary. In this section, we introduce a different clustering approach to key frame extraction. Given a video shot s = f1,f2,...fN obtained from a shot boundary detection algorithm , we cluster the N frames into M clusters, say, 1, 2, ..., M. The similarity of two frames is defined as the similarity of their visual content, where the visual content could be color, texture, shape of the salient object of the frame, or the combination of the above. In this paper, we select the color histogram of a frame as our visual content, although other visual contents are readily integrable into the algorithm. The color histogram we used is a 16 X 8 2D HS color histogram in the HSV color space. The similarity between frames i and j is

$$\sum_{k=1}^{16}\sum_{s=1}^{8} min(H_i(h,s), H_j(h,s))$$

Any clustering algorithm has a threshold parameter "delta" which controls the density of clustering. The higher the delta, the more the number of clusters. In human learning and recognition system we also have this threshold. For example, if the threshold is low, we will classify cars, wagons, mini-vans as vehicles; however, if the threshold is high, we will classify them into different categories. The threshold parameter provides us a control over the density of classification. Before a new frame is classified into a certain cluster, the similarity between this node and the centroid of the cluster is computed first. If this value is less than delta, it means this node is not close enough to be added into the cluster. After the clusters are formed, the next step is to select key frame(s). Here is our strategy: only those clusters which are big enough are considered as key clusters, and a representative frame is extracted from this cluster as the key frame. In this paper we say a cluster is big enough if its size is bigger than N=M, the average size of clusters. For each key cluster, the frame which is closest to the cluster centroid is selected as the key frame, which captures the salient visual content of the key cluster and thus that of the underlying shot.

1.2.3 Deep Unsupervised Key Frame Extraction for Efficient Video Classification[4]

The research paper centers its attention on the critical task of extracting representative key frames from video sequences, which is essential for video classification. Typically, these endeavors follow a conventional pipeline, employing a series of frames as input to the neural network. In this approach, the model is trained to capture spatio-temporal features. A notable departure from previous methods, which attempted to learn features across entire videos, prone to redundancy and resultant performance degradation, the paper introduces an innovative approach. It advocates the extraction of video features from a subset of key frames within the complete video sequence, offering an efficient means to mitigate computational burdens.

Key frame extraction holds a pivotal role in video abstraction, effectively eliminating redundant information from video data and significantly reducing the computational redundancy found between consecutive frames. The paper classifies key frame extraction algorithms into three distinct categories:

Segmentation-Based: This category of methods focuses on detecting abrupt changes in the similarity between successive frames. For instance, in the work of Ejaz et al. [2], key frames are extracted when the inter-frame difference surpasses a predefined threshold. However, this method may have a limitation, as it can extract similar key frames if the same content reappears within the video.

Dictionary-Based: Approaches within this category frame key frame extraction as a sparse dictionary selection problem. In the work of Cong et al. [1], a key frame selection method is proposed, based on sparse dictionary selection with the loss in the L2,1 norm.

Clustering-Based: These approaches revolve around the clustering of frames into groups and subsequently selecting frames nearest to the cluster centers as key frames. For example, Zhuang et al. [5] employ unsupervised clustering based on visual variations to detect key frames.

Furthermore, the paper introduces an innovative unsupervised key frame extraction method, denoted as TSDPC. This method automatically selects the optimal number of key frames, effectively reducing redundant information in the original video while preserving essential temporal information. This contribution promises to be a significant advancement in the field of video key frame extraction.

2 Exploratory Data Analysis

This exploration delves into the mechanics of unsupervised methods, unraveling their critical components through Python scripts. Our EDA focuses on motion vector analysis, frame difference analysis , color histogram analyses, optical flow analysis, frame brightness, RGB color difference, and Histograms of Oriented Gradients (HOG) feature extraction. These insights enable us to construct a model that navigates through the multiple features of video data, optimizing its training and testing processes. Overall, it assists in extracting features for each frame, contributing to a well-layered, highly feature-engineered training set for our model.

2.1 Motion Vector analysis

We first divides frames into blocks, clustering them based on color similarity. Motion vectors are then calculated for each block by comparing it with the corresponding area in the next frame, aiding in identifying motion between frames.

The process starts by segmenting frames into smaller blocks and determining the average color of each block. These blocks are clustered using K-means based on color similarity, enabling the identification of similar regions.

Motion vectors are calculated by comparing blocks between frames, seeking the most similar area within a defined search window. This Sum of Absolute Differences (SAD) approach helps determine the displacement of blocks between frames, visualized as arrows over the original frame.

Figure 1: Original Image

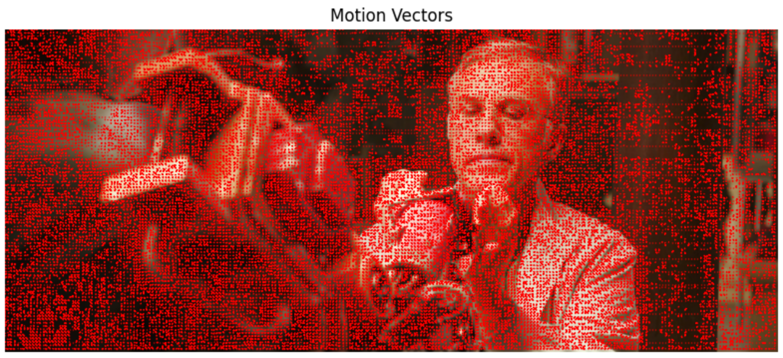

Figure 2: Motion Vectors

2.2 Frame Difference Analysis

This technique is crucial for identifying stability and recognizing scenes with abrupt changes. By computing the mean frame difference, we obtain a valuable metric that distinguishes between significant events (peaks) and stable periods (troughs) in the video content In the Frame Difference Analysis, we utilized OpenCV to calculate the absolute differences between consecutive frames, revealing variations in pixel intensity. This information is vital for unsupervised learning tasks, especially in event detection, as it encapsulates a feature (frame difference) to prioritize and compress frames efficiently, focusing on areas of interest for our model.

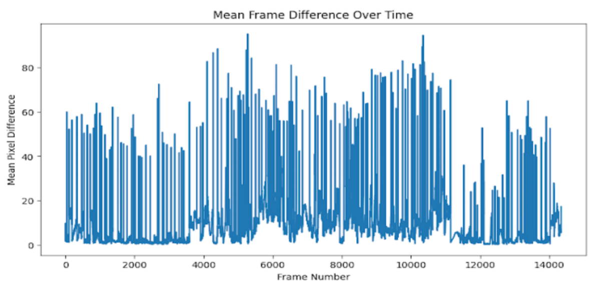

Figure 3: Frame Difference Analysis

2.3 Color Histogram Analysis:

This analysis is valuable for detecting scene changes and subtle details in video content, capturing variations in color. Mean color histograms offer essential insights into the overall color composition. Peaks denote significant events, while troughs represent stable periods. This process aids in creating specific training features, enabling our model to prioritize frame compression for different sections of videos. For Color Histogram Analysis, we harnessed OpenCV to compute histograms representing the color distribution in each frame. this ensures that unsupervised learning tasks can leverage the retained color nuances for improved performance and feature preservation.

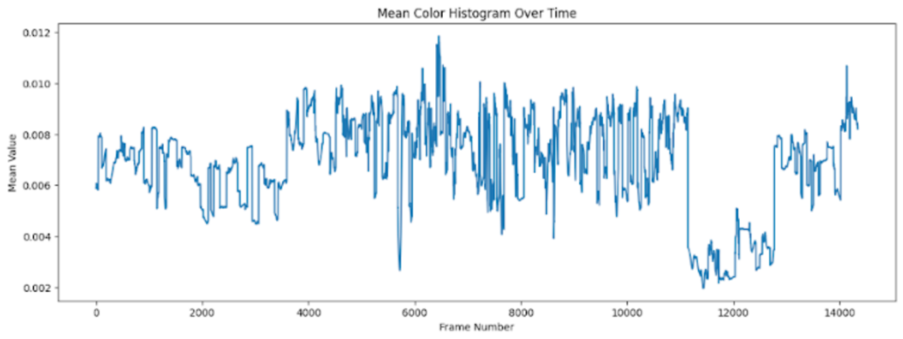

Figure 4: Color histogram Analysis

2.4 Optical Flow Analysis:

Optical flow magnitudes are crucial for depicting motion intensity within video frames. Peaks in magnitudes indicate rapid changes, providing valuable insights for tracking moving objects. In the analysis of optical flow magnitudes, we employed the Farneback method through the OpenCV library to calculate the flow between consecutive frames. This method provides a dense flow field, capturing motion dynamics pixel-wise. The resulting magnitudes serve as a powerful training feature for our unsupervised learning model. By incorporating these magnitudes, the model gains the ability to recognize and track moving objects based on their motion intensity. This, in turn, enhances the efficiency of video compression by enabling the model to prioritize frames with significant motion changes, optimizing the training process for unsupervised learning tasks.

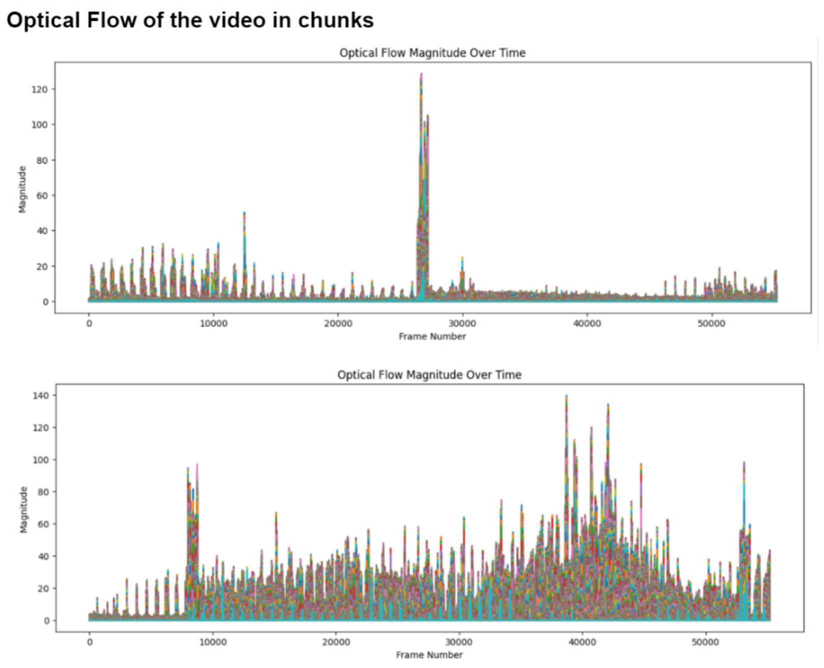

Figure 5: Optical Flow Analysis

2.5 Standardization

In this analysis, we explore the process of standardizing frames extracted from a video using OpenCV. The standardization process begins with the conversion of frames to grayscale using the cv2.cvtColor(frame, cv2.COLORBGR2GRAY) function, emphasizing structural details while eliminating color complexities. Subsequently, contrast enhancement is applied through histogram equalization using cv2.equalizeHist(), ensuring optimal visibility of subtle details. The consistent output format is maintained using cv2.imwrite(), ensuring uniformity across frames. The standardization involves converting frames to grayscale and applying histogram equalization to enhance their quality. The standardized frames, enhance the quality and consistency of data, making frames more amenable for enhancing feature selection for and model training.

2.6 Temporal Analysis of Brightness :

This analysis provides temporal variations in frame brightness, allowing the model to identify stable and dynamic periods. This will allow our model to identify stable bright frames and frame changes which will help in a better cluster formation when incorporated with the rest of the features.

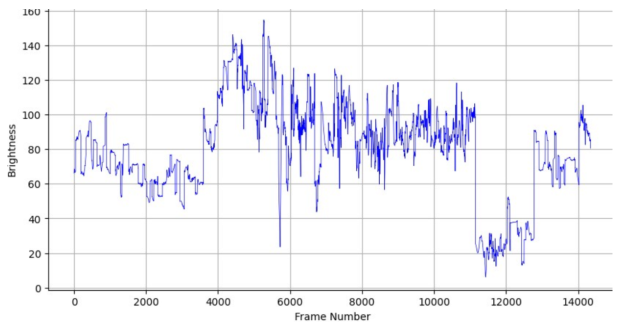

Figure 6: Frame wise Brightness Analysis

Peaks in brightness represent significant changes, while troughs denote stability. With these features, it will help to prioritize which frame to compress and which part of frames to compress.

2.7 Histograms of Oriented Gradients

Histograms of Oriented Gradients (HOG) play a crucial role in object recognition and scene understanding within computer vision. By capturing information about the distribution of gradient orientations in an image, HOG features provide a distinctive representation of object shapes and textures. This is particularly valuable for recognizing objects in varying orientations, scales, and lighting conditions. In the context of our analysis, integrating HOG features in video compression ensures the preservation of these crucial object-related details.

Incorporating Histograms of Oriented Gradients (HOG) in our analysis involved utilizing the scikit-image library, specifically the HOG function, which extracts features representing object structure and edges. Peaks in the HOG features signify the presence of objects or alterations in their shapes, while stability indicates consistent structures over frames

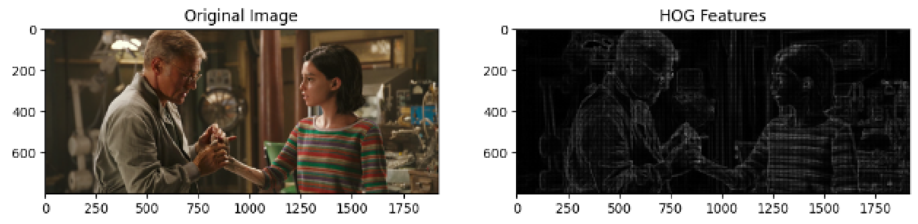

Figure 7: Histogram Of Gradients

3 Methodology

In this project , we process video and audio segment separately as independent components and then combine them after processing to generate the original dataset.

3.1 Video Compression - Approach 1

For video compression, we perform the following steps:

1. Block Division and Preprocessing: The process initiates by segmenting video frames into smaller blocks, a crucial step enhancing localized analysis within frames. These blocks serve as the fundamental units for subsequent analysis, facilitating the identification of spatial redundancies and patterns across frames.
2. Color Clustering for Enhanced Representation: Following block segmentation, the algorithm employs K-means clustering, a robust unsupervised learning technique, to categorize blocks based on their color characteristics. This step significantly condenses the color information by grouping similar colors into clusters. By reducing the color space complexity, this process aims to enhance the representation of video frames while reducing redundant color information.
3. Motion Estimation for Inter-frame Dynamics: Central to video compression, motion estimation involves calculating motion vectors that portray the displacement of blocks between consecutive frames. Leveraging a Diamond search approach, the algorithm assesses the resemblance between blocks, capturing inter-frame motion crucial for predictive coding during compression.
4. Data Encoding for Efficient Storage: Post motion estimation, the project encodes critical data comprising cluster assignments denoting color clusters and calculated motion vectors representing inter-frame motion. This encoding step condenses essential video information into a more compact form, ensuring efficient storage and representation during decompression.

3.2 Video Decompression - Approach 1

1. Initialization and Frame Preparation: The decompression phase initializes parameters and readies the initial frame, ensuring that essential information required for decompression, such as cluster assignments and color representations, is readily available. This preparatory step sets the foundation for subsequent decompression operations.

2. Simulated Frame-by-Frame Decompression:

   The process of Simulated Frame-by-Frame Decompression involves a simplified approach mimicking the decompression process. It virtually applies motion vectors to reconstruct frames, albeit without precise block shifting according to these vectors. Despite this limitation, the strategy offers an effective means to generate a foundational representation of the decompressed frames

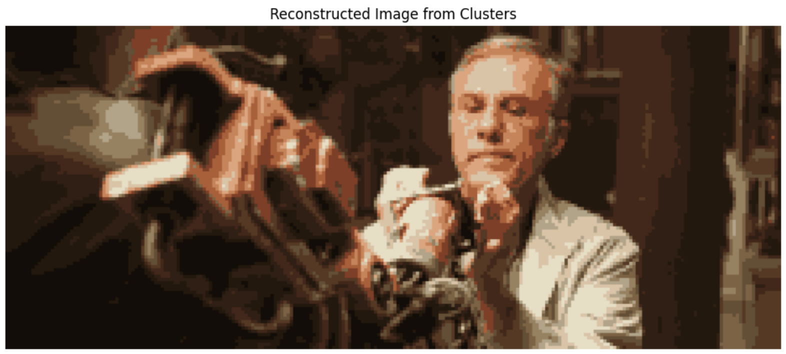

Figure 8: Reconstructed Frame

Video decompression is performed by utilizing the data stored after the compression:

3.3 Video Compression/Decompression - Approach 2

Input video frames are read and resized to a standard resolution (1280x720). Frames are divided into small blocks (8x8 pixels).

For each block: K-means clustering is applied to reduce color information (number of clusters = 5). Cluster centers and labels are stored for decompression. Motion vectors are tracked between consecutive frames by comparing color cluster centers. Compressed data is saved as a list of dictionaries containing cluster information and motion vectors. The process is repeated for a specified number of frames (60 in this case).

Decompression Process: Compressed data, including cluster centers and motion vectors, is read from a pickle file.

For each frame: Reconstruct each block using cluster information (centers and labels) or apply motion vectors to predict cluster movement. Assemble the reconstructed blocks to form the decompressed frame. Decompressed frames are written to an output video file using OpenCV's VideoWriter.

Output Handling: During compression, every 30 frames or at the end of processing, compressed data is saved to a pickle file. Decompression is performed on each batch of 60 frames, and individual video parts are saved. Finally, all video parts are combined into a single output video file.

Implementation Considerations: The script utilizes OpenCV for video processing, NumPy for numerical operations, and Pickle for data serialization. The code attempts to reduce video size by applying k-means clustering to color information and tracking motion vectors for improved compression efficiency.

3.4 Audio Compression

3.4.1 Compression

1. Audio Extraction and Format Conversion:
   - Initially, it reads an input WAV audio file and transforms it into a mono format. Then, it writes this processed audio data into a temporary WAV file.
   - Subsequently, the code leverages the `pydub` library to read this temporary WAV file and convert it into an MP3 format.
2. Compression Technique Application:
   - Upon successful extraction and conversion, the code focuses on compressing the generated MP3 audio file. It reads this MP3 file and applies Principal Component Analysis (PCA) using the `sklearn.decomposition.PCA` functionality.
   - The PCA technique reduces the dimensionality of the audio data, aiming for compression, and subsequently reconstructs the compressed audio using PCA's `inverse_transform` method
3. Output Generation: Finally, the code writes the reconstructed compressed audio into a new WAV file, marking the completion of the compression process.

4 Result and Analysis

We observed convincing results from both approaches. In the first approach, which employed a color clustering using K means, followed by a block motion estimation function, the SNR value for the output video file was $16dB$ for 47 frames and reduced to around $9dB$ as frame count increased. In the second approach, despite the SNR value being lower, at $15dB$ for 30 frames and $3dB$ for 800 frames, the output video was much more robust compared to the first one.

5 Conclusion

Based on the methodology and subsequent results, the project demonstrated the efficacy of classical machine learning techniques in video compression. The use of motion prediction and redundancy identification led to a notable improvement in compression efficiency without sacrificing video quality.

As the digital media landscape continues to expand, the need for efficient video compression methods becomes increasingly paramount. This project contributes valuable insights and methodologies that could inform future innovations in video processing and compression, potentially leading to more advanced systems that blend classical machine learning with emerging technologies.

References

[1] Yang Cong, Junsong Yuan, and Jiebo Luo. Towards scalable summarization of consumer videos via sparse dictionary selection. *IEEE Transactions on Multimedia*, 14(1):66–75, 2012.

[2] Naveed Ejaz, Tayyab Bin Tariq, and Sung Wook Baik. Adaptive key frame extraction for video summarization using an aggregation mechanism. *Journal of Visual Communication and Image Representation*, 23(7):1031–1040, 2012.

[3] C.H. Lampert. Machine learning for video compression: Macroblock mode decision. In *18th International Conference on Pattern Recognition (ICPR'06)*, volume 1, pages 936–940, 2006.

[4] Hao Tang, Lei Ding, Songsong Wu, Bin Ren, Nicu Sebe, and Paolo Rota. Deep unsupervised key frame extraction for efficient video classification. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 19, 12 2022.

[5] Yueting Zhuang, Yong Rui, T.S. Huang, and S. Mehrotra. Adaptive key frame extraction using unsupervised clustering. In *Proceedings 1998 International Conference on Image Processing. ICIP98 (Cat. No.98CB36269)*, volume 1, pages 866–870 vol.1, 1998.