A Project Report

EXPLORING VIDEO FILE FORMAT FOR FORENSIC ANALYSIS

Submitted in partial fulfilment of the
Requirements for the award of the Degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE & ENGINEERING

By

SVN RAMAKANTH

1602-19-733-118

Under the guidance of

Dr. K. Srinivas

Associate Professor

Department of Computer Science & Engineering

Vasavi College of Engineering (Autonomous)

(Affiliated to Osmania University)

Ibrahimbagh, Hyderabad-31

2023

Vasavi College of Engineering (Autonomous)

(Affiliated to Osmania University)

Hyderabad-500 031

Department of Computer Science & Engineering

DECLARATION BY THE CANDIDATE

I, **SVN RAMAKANTH** bearing hall ticket number, **1602-19-733-118**, hereby declare that the project report entitled "**EXPLORING VIDEO FILE FORMAT FOR FORENSIC ANALYSIS**" under the guidance of **Dr.K. Srinivas**, ASSOCIATE PROFESSOR, Department of Computer Science & Engineering, VCE, Hyderabad, is submitted in partial fulfilment of the requirement for the award of the degree of **Bachelor of Engineering in Computer Science & Engineering**.

This is a record of bonafide work carried out by me and the results embodied in this project report have not been submitted to any other university or institute for the award of any other degree or diploma.

SVN RAMAKANTH

1602-19-733-118

Vasavi College of Engineering (Autonomous)

(Affiliated to Osmania University)

Hyderabad-500 031

Department of Computer Science & Engineering

BONAFIDE CERTIFICATE

This is to certify that the project entitled **"EXPLORING VIDEO FILE FORMAT FOR FORENSIC ANALYSIS"** is being submitted by **SVN Ramakanth**, bearing 1602-19-733-118**,** in partial fulfilment of the requirements for the award of the degree of Bachelor of Engineering in Computer Science & Engineering is a record of bonafide work carried out by him/her under my guidance.

Dr.K. Srinivas,
Associate Professor,
Internal Guide

Dr. T.Adilakshmi,
Professor & HOD,
Dept. of CSE

ACKNOWLEDGEMENT

I would like to express my gratitude to all the members of the department who gave me the opportunity to work on this project. I owe my profound gratitude to my project internal guide Dr. K. Srinivas, Associate Professor, Department of Computer Science and Engineering, who helped me, by giving suggestions and encouragement and helped me complete the project.

I am also grateful to our Head of the Department, Dr. T. Adilakshmi, Principal, Dr. S. V. Ramana and the management of the institute, for providing this opportunity.

SVN Ramakanth

1602-19-733-118

ABSTRACT

Video file carving is a technique used in digital forensics to recover deleted or damaged video files from a storage device like a disk in the absence of file table information. It involves analyzing the raw data on the storage device to locate fragments or whole blocks of data that may belong to a video file. The carved data is then reconstructed and extracted into a playable video file. It can be a useful technique in digital forensics, data recovery, and criminal investigations, as it can help to recover important evidence that might otherwise be lost. However, it can also be a time-consuming and complex process that requires specialized tools and expertise to perform accurately. The process of video file carving requires specialized software tools and a deep understanding of file system structures and data recovery techniques. Our project is an attempt to build a forensic tool that aids in video file carving.

An atom is a basic unit of data that represents a specific type of information or metadata about the content of the file. The current project automates the process of Atom Classification considering MP4 Video File format. As atom Classification lays the foundation for any forensic activity, given a video file as input, we will classify the video file into different atoms and their corresponding attributes, which helps us in understanding the nature of video. In video compression, a keyframe (also known as an I-frame or intra-frame) is a type of frame that contains a full image, independent of any other frames in the video sequence. This project also focuses on extracting key frames using various methods and fragment reconstruction based on structural evidence.

Table of Contents

List of Figures

1. INTRODUCTION

1.1 Overview

File carving is the process of recovering files that have been deleted or lost due to data corruption or other issues without using file table information. When a file is stored on a device, it is broken down into smaller units of data called atoms. Atom classification is an important concept in file carving as it helps to identify the different types of data that may be present on a storage device. These atoms can then be classified based on their type, whether they are ftyp type atoms which tells us about file type or mvhd atoms which tells us about movie header etc.

By classifying atoms, file carving tools can more effectively search for and extract specific types of data. For example, if a user is looking to recover a deleted image file, a file carving tool can scan the storage device for atoms that are classified as JPEG image data, and then extract those atoms to recover the image. Atom classification is also useful for identifying file fragments. When a file is fragmented, its atoms may be scattered across different areas of the storage device. By classifying the atoms, file carving tools can identify which atoms belong to the same file and reassemble them to recover the complete file.

The below atoms can be classified by our tool:

1. ftyp
2. free
3. mdat
4. moov
5. mvhd
6. trak
7. tkhd
8. mdia
9. mdhd
10. hdlr
11. minf
12. smhd
13. vmhd
14. dinf
15. dref
16. url

17. stsd
18. stss
19. stts
20. stsc
21. stsz
22. stco
23. ctts
24. sdtp
25. stps
26. cslg
27. udta
28. sgpd
29. sbgp
30. iods
31. edts
32. elst
33. stbl

Stage-2 deals with identification and classification of frames. In video, a frame is a single still image that is displayed on the screen for a specific period of time before being replaced by the next frame. Each frame in a video contains information about the colour and brightness values of each pixel in the image, as well as any other metadata that might be associated with the frame, such as the timecode or frame number. If we have enough information, we can identify and extract frames from the hex content we have.

Key frame identification is an important aspect of file carving because it can improve the accuracy and efficiency of the file recovery process. Key frames are frames in a video file that represent significant changes in the content, it captures the scene changes or camera movements. By identifying these key frames, file carving tools can more easily and accurately recover video files, as they can use the key frames to identify the boundaries of the file and extract the video data more efficiently. Without the key frame identification, file carving tools may need to scan through the entire storage device and examine every frame of the video file to recover it, which can be a time-consuming and resource-intensive process. By using key frame identification, file carving tools can more quickly and accurately recover video files, as they can focus their efforts on the relevant sections of the storage device and extract the necessary data more efficiently.

By accurately identifying the boundaries of the video file, file carving tools can ensure that the recovered file is complete and free from errors or corruption.

Stage-3 deals with the redundant attributes present in the atoms classified and tracks them down with their corresponding values. If there is a corruption or missing values of the redundant attributes, then the most probable choice value (max count) is considered to be the final value and is assigned to the attributes in all atoms where the redundancy is found and a new corruption-free fragment is generated. It will also consider the structure-based evidence in order to come up with certain conclusions related to the considered partial file fragment.

Below ideas are used for partial file carving:

1. Attribute Redundancy
2. Integrity Checker
3. Structural Evidences

1.2 Motivation

In many cases, users may not have a backup of their data and need to recover it using file carving techniques. For example, if a user accidentally deletes an important file from their hard drive, they may use a file carving tool to recover the deleted file. Similarly, if a storage device becomes corrupted, file carving techniques may be used to recover as much data as possible.

File carving techniques are also commonly used in forensic investigations to recover data that may have been intentionally deleted or hidden. For example, in a criminal investigation, a file carving tool may be used to recover deleted emails or documents that could provide important evidence.

In general, the motivation for using file carving techniques is to recover lost or deleted data that is important or valuable to the user. It is a useful technique for recovering data that cannot be retrieved through other methods, such as backups or system restore points.

1.3 Problem Definition

The purpose of the project **"**File Carving Tool" is to automate the process of Atom Classification considering MP4 Video File format. Atom Classification lays the foundation for any forensic activity. When a file is fragmented, its atoms may be scattered across different areas of the storage device. By classifying the atoms, we can identify which atoms belong to the same file and reassemble them to recover the complete file. This project also focuses on extracting key frames using various methods like AMDFrameDetector, BDFrameDetector etc and fragment reconstruction based on structural evidence.

1.4 Objective

The objective of this project is to lay a foundation for recovery of deleted video files in the case of meta data corruption or no knowledge on file system. Given a video file, an in-depth atom classification is done to extract the meta data information like creation time of the video, modification time, time scale, duration, predefines etc which helps us in understanding the nature of video file content. Right after the atom classification, we can deep dive into the frame data to identify the key frames and try to stitch the split key frames without any loss of data. Finally, using structural evidence, we can able to make meaningful conclusions on the nature of the given partial file fragment which takes us to the advanced stage of extending the partial fragments found on disks to make a video playable.

1.5 Scope

The scope of file carving is quite broad, as it can be used to recover a wide variety of file types from different types of storage devices. File carving can be used on hard drives, solid-state drives, USB drives, memory cards, and other types of storage devices.

Some of the file types that can be recovered using file carving techniques include images, videos, audio files, documents, and archives. The ability to recover such a wide range of file types makes file carving a valuable tool for data recovery, forensic investigations, and other applications.

The scope of file carving is constantly expanding as new technologies and file formats are developed. As a result, file carving tools must be updated to support these new formats and to remain effective in recovering lost or deleted data.

Overall, the scope of file carving is quite broad and it is a valuable tool for data recovery, forensic investigations, and other applications where the recovery of lost or deleted data is needed.

2. LITERATURE SURVEY

Overview of file carving: File carving has its roots in computer forensics, which emerged in the 1980s as a means of investigating digital crimes. The focus of computer forensics was on recovering data from storage media, such as hard drives. However, as the use of digital devices became more widespread, the need for more advanced techniques to recover data from fragmented, corrupted, or deleted files grew.

The concept of file carving, also known as data carving, emerged in the early 2000s as a means of recovering data from incomplete or damaged files. The technique involves scanning a storage device for header and footer signatures that identify the file type and using these signatures to extract data from the file. There are unique signatures available for every file type. This approach allows data to be recovered even when the file system meta data information, such as the file name or location, is missing/corrupted.

In the years that followed, file carving techniques evolved to become more sophisticated and capable of recovering a wider range of file types, including images, videos, audio files, and other binary data. Today, file carving is widely used in forensic investigations, data recovery, and other areas where data needs to be extracted from damaged or incomplete files.

Advancements in technology, including the growth of cloud storage and mobile devices, have also contributed to the evolution of file carving. As data becomes more distributed and fragmented across multiple devices and storage locations, the need for more advanced file carving techniques to recover data from these sources has increased.

Overall, the history and evolution of file carving reflect the ongoing need for new and more advanced techniques to recover data from digital devices and storage media. As technology continues to evolve, it is likely that file carving techniques will continue to evolve and become even more sophisticated and effective in the years to come.

File carving techniques: There are several file carving techniques that are commonly used in digital forensics and data recovery, including:

Header/footer carving: In this technique we search for the known file header and footer signatures to locate and extract files from the raw data on a storage device. These signatures are unique to the file types available. They mark the beginning and end of a file, respectively. This technique is particularly effective for recovering files that have been deleted or lost due to file system corruption.

Signature-based carving: This technique involves searching for known file signatures, which are unique byte patterns that identify a specific file type. Signature-based carving can be used to identify and extract files even if the file headers and footers are missing or corrupted.

Content-based carving: This technique involves searching for patterns or sequences of data that are characteristic of a particular file type. For example, image files often have recognizable patterns in their pixel data, while text files may have identifiable patterns in their character encoding. Content-based carving can be used to recover files that have been partially overwritten or fragmented.

File system-based carving: This technique involves searching for files based on their location within the file system. This technique is particularly useful when recovering files from damaged file systems or when attempting to recover specific files from a large volume of data.

Metadata carving: This technique involves extracting file metadata, such as creation date, modification date, and file size, to identify and recover files. This technique can be used to recover files that have been deleted or lost due to file system corruption or other issues.

Each of these file carving techniques has its strengths and weaknesses, and the choice of technique depends on the specific situation and the type of data that needs to be recovered. A combination of techniques may be used to increase the chances of successfully recovering data from damaged or incomplete files.

File carving tools: There are several file carving tools available that are commonly used in digital forensics and data recovery, including:

Defraser: Defraser is a free and open-source digital forensic tool that can be used to detect and recover video files from a wide range of digital storage media. It is commonly used by forensic investigators and law enforcement agencies to recover video evidence from digital cameras, smartphones, and other devices.

Scalpel: Scalpel is a file carving tool that uses a configuration file to define the file types and headers and footers to search for. It supports a wide range of file types and can be configured to search for files based on their content or location within the file system.

PhotoRec: PhotoRec is a free, open-source file carving tool that is designed to recover lost files from digital media. It supports a wide range of file types, including images, videos, and documents, and can be used to recover files from damaged or deleted partitions.

Keyframe identification: There are several existing key frame identification techniques used in file carving, including:

Scene change-based identification: In this technique, the key frames are identified based upon how a scene is getting transitioned from frame 1 to the next frame. Motion based changes, pixel level changes etc are considered.

Frame count-based identification: In this technique, the key frames are identified based upon the frame count. Example: If frame count is fixed to a value 8, then every 8th frame will be considered as a key frame.

Motion-based identification: : In this technique, the key frames are identified based upon their motion characteristics. For example, key frames could be identified based on their high motion content or the presence of specific types of motion.

Object-based identification: : In this technique, the key frames are identified based upon the presence of specific objects or patterns within the video file. For example, key frames could be identified based on the presence of identifiable objects.

Challenges and limitations: Some of the key challenges and limitations in file carving include:

Fragmentation: Fragmentation occurs when a file is split into multiple fragments and scattered across different parts of the storage media. File carving tools may not be able to recover the entire file if some fragments are missing or damaged.

File system dependencies: File carving relies on the ability to identify and extract files based on their content, rather than their metadata. However, some file systems may overwrite or modify file content, making it difficult or impossible to recover the original file.

File type identification: File carving tools must be able to accurately identify the file type in order to recover the file. However, some file types may have non-standard headers or may be encrypted, making it difficult to identify them.

Time and resource constraints: File carving can be a time-consuming and resource-intensive process, particularly for large or complex files. It may also require specialized hardware or software resources, such as high-end processors or large amounts of RAM.

Legal and ethical considerations: File carving may involve accessing sensitive or confidential data, and may be subject to legal and ethical considerations, such as data privacy laws and regulations.

3. Design of File Carving Tool

3.1 Proposed System

The proposed system works to serve the following purpose:

- Automatically the atom classification is done given a video file. (Stage1)
- Frames are extracted and key frame identification is done. (Stage 2)
- Given a partial fragment belonging to a video sample, it will be able to recover the corrupted parts or missing parts and generate the new partial fragment. It decides upon the nature of the file based upon the structural evidence that are present in the fragment. (Stage 3)

Stage-1 (Atom Classification)

Atom classification is an important task in file carving because it helps to identify the type of file that is being carved. Files are made up of a series of atoms, which are the smallest units of data in a file. By analysing the content of these atoms, it is possible to classify the file type.

The classification of file types is important in file carving because it helps to determine how the data should be reconstructed. Different file types have different structures and characteristics, and the process of reconstructing a file can be different for each type. For example, an image file may have a different structure than a text file and reconstructing each type of file requires different techniques.

In Stage-1, a video file/partial video file fragment is taken as an input from the user. Atom Classification task is applied onto the input fragment. We use atom dictionaries and atom classifiers to classify the group of bytes of the video file into a specific atom. There are about 20 atom classifiers which purely work using pointers pointing to a hex file. The attribute's size and type related information is taken from the atom dictionaries which are predefined in a standard reference. We keep a track of redundant attributes (attributes which occur in two or more atoms) in order to use the most probable value corresponding to the redundant attribute as the final value during corruption or missing value state. Integrity checker is used to find the next possible atom's type and size (acts

like a lookahead). Few support functions are used for the sake of conversions (little endian format, big endian format, hex to ascii conversions etc).

Stage-2 (Key Frame Identification)

Key frame identification is an important aspect of file carving because it can significantly improve the accuracy and efficiency of the file recovery process. Key frames are frames in a video file that represent significant changes in the content, such as scene changes or camera movements. By identifying these key frames, file carving tools can more easily and accurately recover video files, as they can use the key frames to identify the boundaries of the file and extract the video data more efficiently.

Without key frame identification, file carving tools may need to scan through the entire storage device and examine every frame of the video file to recover it, which can be a time-consuming and resource-intensive process. By using key frame identification, file carving tools can more quickly and accurately recover video files, as they can focus their efforts on the relevant sections of the storage device and extract the necessary data more efficiently**.**

In Stage-2, as a part of key frame identification, we used 7 methods:

- **AMDFrameDetector** (absolute mean difference between consecutive frames is calculated and if the difference is greater than predefined threshold, that is considered as a key frame)
- **BDFrameDetector** (blob detection method uses key frame detection parameters and their fixed thresholds to identify key frames)
- **BGSubFrameDetector** (extracts frames from a video file based on background subtraction and saves only those frames where significant changes in foreground objects are detected)
- **CustomFrameDetector** (extracts keyframes based on user's custom frame input)
- **FIFrameDetector** (based on saving every n^{th} frame (key_frame_interval))

- **OptFlowFrameDetector** (Horn-Schunck method Implimentation)
- **RMSFrameDetector** (based on rms value between consecutive frames)

Stage-2 also focuses on certain important tasks such as identifying the file type based on content or metadata or signatures and video file frame splitting and stitching task. Overall, it focuses on frame level classification of a video file.

Stage-3 (Recovery of Corrupted Fragments)

Stage-3 focuses on the redundant attributes present in the atoms classified and tracks them down with their corresponding values. If there is a corruption or missing values of the redundant attributes, then the most probable choice value (max count) is considered to be the final value and is assigned to the attributes in all atoms where the redundancy is found and a new corruption-free fragment is generated. It will also consider the structure-based evidence in order to come up with certain conclusions related to the considered partial file fragment.

Considered evidence are:

- Certain file types, including video files, have specific header information or "magic numbers" that can be used to identify the file type. For example, the magic number for a MPEG-4 file is 0x000001BA.
  (Converting 0x000001BA to ASCII results in a non-printable character, as it falls outside the printable ASCII range (0x20 to 0x7E). 0x000001BA is actually a hexadecimal value commonly found in the header of MPEG-2 video files, indicating the start of a Program Association Table (PAT).
- By analysing the entropy or distribution of bytes in a file segment, it may be possible to identify if the file segment is a video file.
- The presence of certain patterns or sequences in the file segment may suggest that it is a video file. For example, MPEG-4 video files often contain repeating sequences of data.
- Looking for evidence of video-specific artifacts, such as pixelation or blocking.

- Analysis of the file's binary structure may reveal the presence of video-related data, such as motion vectors, quantization tables, etc.
- If the code segment(hex) when converted to ASCII contains any of the atom names in it, then it might be evidence for the video file being present.
- Collected file fragments, if any of the possible video formats are in ftyp then it might be evidence for the video file being present. (File header= signature = ftyp and a suitable value to it by maintaining a list of all the possible values)
- Attribute-based evidence can be collected from selective atoms.For example: we can collect Creation_time, Modification_time, Duration (To be checked), Version, Size_of_Video (Can be found by seeking through atoms).
- Find file extensions like (sample.*mp4*,*mpeg*) this may suggest a video file
- Metadata associated with the file can indicate the presence of video-related information, such as video dimensions, framerate, etc.

3.2 Software Requirements

VS Code: Visual Studio Code is “a free editor that helps the programmer write code, helps in debugging and corrects the code. Visual Studio Code has some very unique features: supports multiple programming languages, Cross platform support etc.

Hex Editor: A hex editor is a type of computer program that allows users to view and edit binary files. Binary files are files that contain machine-readable code or data, and they are often used for low-level system tasks, such as device drivers or firmware. Unlike text editors, which are designed to work with human-readable text, hex editors allow users to view and edit the raw binary data of a file. In our project, we used Hex editor (UWP) and Hex editor (NEO).

DiffChecker: Diffchecker is an online tool that allows users to compare and highlight the differences between two pieces of text or files. It is commonly used for comparing code files, documents, and website content. Diffchecker can be used to compare two files or two text inputs by pasting them into the tool or uploading them. When two inputs are compared, Diffchecker highlights the differences between them and provides a visual representation of the changes. The tool uses color-coding to highlight additions

and deletions, making it easy to see what has changed between the two inputs. Users can also choose to view the differences in a side-by-side view or in a unified view, where the changes are displayed in one file with added and deleted lines marked.

Defraser: Defraser is a free and open-source digital forensic tool that can be used to detect and recover video files from a wide range of digital storage media. It is commonly used by forensic investigators and law enforcement agencies to recover video evidence from digital cameras, smartphones, and other devices.It works by scanning the storage media for video files and then analysing them to detect any corruption or damage. It can detect a variety of video file formats, including AVI, MPEG, and WMV. The tool also provides a number of options for refining the search and recovering specific types of video files, such as those recorded by a particular camera model or in a particular format.

3.3 Hardware Requirements

The hardware requirements for file carving tasks can vary depending on the specific software tools and techniques being used, as well as the size and complexity of the storage media being analyzed. Generally, however, file carving tasks can be resource-intensive and require a reasonably powerful computer with the following minimum hardware specifications:

Processor: A multi-core CPU, such as an Intel Core i5 or i7, or an AMD Ryzen processor, is recommended for faster processing of data.

RAM: At least 8GB of RAM is recommended, as file carving tasks can require significant amounts of memory.

Storage: Sufficient storage space is required to hold the recovered files, which can be quite large in size. Ideally, an external hard drive or SSD with a fast transfer rate is recommended to store the recovered files.

Graphics Card: A dedicated graphics card is not necessarily required, but it can be useful for certain file carving tasks, such as video or image recovery, where visualization of the recovered files is important.

Some examples of graphics cards that could be suitable for file carving include:

NVIDIA GeForce RTX 3090: This high-end graphics card has 10,496 CUDA cores, 24 GB of memory, and a clock speed of 1.70 GHz.

AMD Radeon RX 6900 XT: This graphics card has 5,120 Stream processors, 16 GB of memory, and a clock speed of 2.25 GHz.

4. IMPLEMENTATION

4.1 Sample Code

Imported Modules

Binascii:

The binascii module in Python provides a set of functions to convert binary data to ASCII-encoded strings and vice versa. It contains various methods to perform common binary-to-text and text-to-binary conversions, such as hexadecimal, Base64, and ASCII. hexlify(data): This function takes binary data as input and returns its hexadecimal representation as a string.

CV2:

The cv2 module in Python is a computer vision library that provides a variety of functions and tools for image and video processing.

Numpy:

The numpy module in Python is a fundamental package for scientific computing that provides support for large, multi-dimensional arrays and matrices, along with a vast library of mathematical functions to operate on these arrays.

Magic:

The magic module in Python is used to determine the file type of a given file using magic numbers, which are unique signatures or patterns in the first few bytes of a file.

Pymp4parse:

The pymp4parse module is a Python library for parsing MP4 files. MP4 is a digital multimedia format that is used for storing video and audio data, as well as other types of multimedia content such as subtitles, images, and metadata. The format is widely used for streaming video over the internet, as well as for digital downloads.

The pymp4parse module provides a way to parse MP4 files and extract metadata and data streams from them. The module supports both ISO Base Media File Format (ISO/IEC 14496-12) and the QuickTime File Format (QTFF). It can be used to extract information such as the format of the video and audio streams, the duration of the video, the frame rate, and the bit rate.

hachoir.parser:

The hachoir.parser module in Python is a part of the Hachoir project, which is a collection of Python libraries for binary file formats parsing. The hachoir.parser module provides a framework for parsing various file formats, such as archives, executable files, multimedia files, and many others.

hachoir.metadata:

The hachoir.metadata module in Python is another part of the Hachoir project, which is a collection of Python libraries for binary file formats parsing. The hachoir.metadata module provides a way to extract metadata information from various file formats using parsers from the hachoir.parser module.

Ffmpeg:

ffmpeg is a popular and powerful command-line tool for handling multimedia files such as videos, audios, and images. It can be used to convert, resize, and filter multimedia files, among other tasks.

In addition to the command-line interface, ffmpeg also provides libraries for developers to integrate multimedia processing into their applications. One of the most popular libraries is the pyffmpeg library, which is a Python wrapper around the ffmpeg command-line tool.

Stage-1

AtomDictionaries.py

It consists of the structure of all the atoms considered in this project in the form of python dictionaries. Keys are the attributes of the atoms with values being ‘NONE’ in the beginning but later assigned during classification.

```
ftyp = {
    'Size':None, # 4-bytes
    'Type':'ftyp', # 4-bytes
    'Major_Brand':None, # 4-bytes
    'Minor_Version' : None, # 4-bytes
```

```
    'Compatible_Brands' : [], # Variable size
}

free = {
   'Size':None, # 4-bytes
    'Type':'free', # 4-bytes
}
mdat = {
   'Size':None, # 4-bytes
    'Type':'mdat' # 4-bytes
}
moov = {
   'Size':None, # 4-bytes
    'Type':'moov' # 4-bytes
}
mvhd = {
    'Size':None, # 4-bytes
    'Type':'mvhd',# 4-bytes
    'Version':None,  # 1-byte
    'Flags':None, # 3-bytes
    'Creation_Time':None, # 4-bytes
    'Modification_Time':None, # 4-bytes
    'Time_Scale':None,# 4-bytes
    'Duration':None,# 4-bytes
    'Preferred_Rate':None, # 4-bytes
    'Preferred_Volume':None, # 2-bytes
    'Reserved':None, # 10-bytes
    'Matrix':None, # 36-bytes
    'Preview_Time':None, # 4-bytes
    'Preview_Duration':None, # 4-bytes
    'Poster_Time':None, # 4-bytes
    'Selection_Time':None, # 4-bytes
    'Selection_Duration':None, # 4-bytes
    'Current_Time':None, # 4-bytes
```

```
    'Next_Track_ID':None # 4-bytes,
}
```

Classifiers.py

It consists of the individual classifier function corresponding to each atom. The classifier function parses the hex data and identify values to the attributes present in an atom.

```
from SupportFunctions import bigEnd, hex_to_ascii, dprint, convert_hex_to_datetime
from AtomDictionaries import ftyp, stbl,free,edts, mdat, moov, mvhd, trak, tkhd,
mdia, mdhd, hdlr, minf, smhd, vmhd, dinf, dref, url, stbl, stsd, stts, stss, stsc, stsz,
stco, ctts, sdtp, stps, cslg, udta, sgpd, sbgp, elst, iods
Creation_Time={}
Modification_Time={}
Matrix={}
def ftyp_classifier(pointer,data,inclusion):
    tempo = ""
    pointerr=pointer
    if inclusion==True:
        if(pointerr+8<=len(data)):
            for i in range(pointerr, pointerr+8):
                tempo += data[i]
            ftyp['Size'] = bigEnd(tempo)
            tempo = ""
            pointerr=pointerr+8
        else:
            (pointerr,ftyp)
        if(pointerr+8<=len(data)):
            ftyp['Type']='ftyp'
            pointerr=pointerr+8
        else:
            (pointerr,ftyp)

    if(pointerr+8<=len(data)):
```

```
        for i in range(pointerr, pointerr+8):
            tempo += data[i]
        ftyp['Major_Brand'] = hex_to_ascii(tempo)
        tempo = ""
        pointerr=pointerr+8
    else:
        (pointerr,ftyp)
    if(pointerr+8<=len(data)):
        for i in range(pointerr, pointerr+8):
            tempo += data[i]
        ftyp['Minor_Version'] = (tempo)
        tempo = ""
        pointerr=pointerr+8
    if inclusion==True:
        count = (ftyp['Size']*2-32)/4
        count = int(count/2)
        c_brand = []
        for j in range(0, count):
            c_brand.append(hex_to_ascii(data[pointerr:pointerr+8]))
            pointerr += 8
        ftyp['Compatible_Brands'] += c_brand
        return (pointerr,ftyp)
    else:
        c_brand=[]
        while(pointerr+8<=len(data)):
          c_brand.append(hex_to_ascii(data[pointerr:pointerr+8]))
          pointerr += 8
        ftyp['Compatible_Brands'] += c_brand
        return (pointerr,ftyp)
```

HexData.py

This file consists of code which takes a video file or partial fragment as input and gives us the hex form of it

```
import binascii
with open('VideoSamples/Sample-2-HR.mp4','rb') as f:
  data  = str(binascii.hexlify(f.read()))[2:-1]
```

AtomClassification.py

This is a main function which considers a video file/partial video file fragment as input and classifies it into substituent atoms.

```
from IntegrityChecker import integrity_checker
from SupportFunctions import bigEnd,
dprint,test,hex_to_ascii,convert_string_to_hex,convert_hex_to_datetime
from HexData import data
import os
from AtomDictionaries import ftyp, free, mdat, moov, mvhd, trak, tkhd, mdia, mdhd,
hdlr, minf, smhd, vmhd, dinf, dref, url, stbl, stsd, stts, stss, stsc, stsz, stco, ctts, sdtp,
stps, cslg, udta,edts, iods,sgpd,sbgp,elst

from UpdatedClassifiers import ftyp_classifier,iods_classifier, elst_classifier,
udta_classifier, free_classifier, mdat_classifier, moov_classifier, mvhd_classifier,
trak_classifier, tkhd_classifier, mdia_classifier, mdhd_classifier, hdlr_classifier,
minf_classifier, vmhd_classifier, smhd_classifier, dinf_classifier, dref_classifier,
sample_atom_classifier,elst_classifier, iods_classifier

d={'ftyp':ftyp, 'free':free, 'mdat':mdat, 'moov':moov, 'mvhd':mvhd, 'trak':trak,
'tkhd':tkhd, 'mdia':mdia, 'mdhd':mdhd, 'hdlr':hdlr, 'minf':minf, 'smhd':smhd,
'vmhd':vmhd, 'dinf':dinf, 'dref':dref, 'url':url, 'stbl':stbl, 'stsd':stsd, 'stts':stts, 'stss':stss,
'stsc':stsc, 'stsz':stsz, 'stco':stco, 'ctts':ctts, 'sdtp':sdtp, 'stps':stps, 'cslg':cslg, 'udta':udta,
'sgpd':sgpd, 'sbgp' : sbgp, 'elst':elst, 'iods':iods}

x = {'ftyp': '(File type)', 'free': 'Free type', 'mdat': 'Media Data', 'moov': 'Movie type',
'mvhd': 'Movie Header type', 'trak': 'Track type', 'tkhd': 'Track Header type ', 'mdia':
```

```
'Media type', 'mdhd': 'Media Header', 'hdlr': 'Handler Description', 'minf': 'Media Information', 'smhd': 'Sound Media Header', 'vmhd': 'Video Media Header', 'dinf': 'Data information',
    'dref': 'Data Reference', 'url': 'URL type', 'stbl': 'Sample Atom type', 'stsd': 'Sample Atom type', 'stts': 'Sample Atom type', 'stss': 'Sample Atom type', 'stsc': 'Sample Atom type', 'stsz': 'Sample Atom type', 'stco': 'Sample Atom type', 'ctts': 'Sample Atom type', 'sdtp': 'Sample Atom type', 'stps': 'Sample Atom type', 'cslg': 'Sample Atom type', 'udta': 'User Data','sgpd':'sgpd', 'sbgp' : 'sbgp', 'elst':'elst', 'iods':'iods'}

l=[]
missing_atoms={}
track_counter=1
thisfile=False
import datetime

def WriteToAFile(name="",d={},booll=False,listt=[],missing_atoms={}):
    global thisfile
    global track_counter
    if thisfile is False:
        myf = open('Stage-1\OutputSamples/output.txt', 'w')
        myf.write('Execution time:\n')
        myf.write(str(datetime.datetime.now()))
    else:
        myf = open('Stage-1\OutputSamples/output.txt', 'a')
    if booll==False:
        if name=='udta':
            myf.write("\n"+"\nNext Up We Have User Data Atoms:\n")
        if track_counter==2 and name=='trak':
            myf.write("\n"+"\nNext Up We Have Audio Track Atoms: \n")
        if track_counter==1 and name=='trak':
            myf.write("\n"+"\nNext Up We Have Video Track Atoms: \n")
        if name == 'trak':
            track_counter+=1
```

```
        myf.write("\n"+"\nClassified Atom is : "+str(name)+"----------------------------
\n")
        myf.write('Size : '+str(d['Size'])+"\n")
        for attribute, value in d.items():
            try:
                if len(str(value)) > 1000:
                    myf.write(str(attribute)+" : "+str(value[1050])+"\n")
                else:
                    myf.write(str(attribute)+" : "+str(value)+"\n")
            except:
                pass
    counter=1
    if booll==True:
        myf.write("\n"+"\nList Of Atoms : ----------------------------\n")
        for item in listt:
            myf.write(str(counter)+". "+str(item)+"\n")
            counter+=1
        myf.write("\n"+"\nList Of Missing Atoms : ----------------------------\n")
        if len(missing_atoms)==0:
            myf.write("None")
        for k,v in missing_atoms.items():
            myf.write(str(k)+":"+str(v)+"\n")
    myf.close()

def MainAtomClassification():
    global thisfile
    try:
        # file_size=ClassifyBlock()
        # file_size = os.path.getsize(r'Stage-
3\\RedundancyPartialFileSamples\\CorruptedPartialFile1.txt')
        file_size = os.path.getsize(r'VideoSamples\Sample-2-HR.mp4')
        hex_pointer = 0
        origin_pointer = 0
        while (hex_pointer < (file_size)):
```

```
        cur_size, cur_atom = integrity_checker(hex_pointer,data)
        l.append(cur_atom)
        if (cur_atom == "moov" or cur_atom == "trak" or cur_atom == "mdia" or
cur_atom == "minf" or cur_atom == "dinf"):
            origin_pointer,g = eval(
                cur_atom+'_classifier')(hex_pointer,data,True)
            hex_pointer += 16
            print(cur_atom.upper(
            )+"("+x[cur_atom]+")"+" Atom Details are:-----------------------------------")
            # dprint(d[cur_atom])
            dprint(g)
            WriteToAFile(name=cur_atom,d= d[cur_atom])
            thisfile=True
        elif (cur_atom == "stbl"):
            tempo = ""
            for j in range(hex_pointer, hex_pointer+8):
                tempo += data[j]
            stbl['Size'] = bigEnd(tempo)
            print("\nSample Table (stbl) details:-----------------------------------")
            dprint(stbl)
            WriteToAFile(name=cur_atom,d= d[cur_atom])
            # global thisfile
            thisfile=True
            hex_pointer += 16
        elif (cur_atom == "edts"):
            tempo = ""
            for j in range(hex_pointer, hex_pointer+8):
                tempo += data[j]
            edts['Size'] = bigEnd(tempo)
            print("\n Edit Samples (edts) details:-----------------------------------")
            dprint(edts)
            hex_pointer += 16
            hex_pointer,g = sample_atom_classifier(
                cur_atom, hex_pointer, data,True)
```

```
            print(cur_atom.upper(
            )+"("+x[cur_atom]+")"+" Atom Details are:-----------------------------------")
            # dprint(d[cur_atom])
            dprint(g)
            WriteToAFile(name=cur_atom,d= d[cur_atom])
            thisfile=True
        elif cur_atom == 'udta' or cur_atom == 'elst' or cur_atom == 'iods' :
            hex_pointer,g = eval(
                cur_atom+'_classifier')( hex_pointer, data,True)
            print(cur_atom.upper(
            )+"("+x[cur_atom]+")"+" Atom Details are:-----------------------------------")
            dprint(g)
            WriteToAFile(name=cur_atom,d= d[cur_atom])
            thisfile=True
        elif cur_atom in d.keys():
                hex_pointer,g = eval(cur_atom+'_classifier')(hex_pointer,data,True)
                if cur_atom!='dref':
                    print(cur_atom.upper(
                    )+"("+x[cur_atom]+")"+" Atom Details are:-----------------------------------
----")
                    dprint(g)
                WriteToAFile(name=cur_atom,d= d[cur_atom])
                thisfile=True
        else:
            hex_pointer += cur_size*2
            missing_atoms[cur_atom]=cur_size
    print("File size: ", file_size)
    print("Final hex pointer value: ", hex_pointer//2)
    print("\nAtoms Encountered:")
    print(l)
    WriteToAFile(booll=True,listt=l,missing_atoms=missing_atoms)
    print("\nMissing Atoms:")
    print(missing_atoms)
    print('\nAtom Redundancy:')
```

```
        from UpdatedClassifiers import Creation_Time,Modification_Time,Matrix
        print('Creation Time:')
        print(Creation_Time)
        print('Modification Time:')
        print(Modification_Time)
        print('Matrix:')
        print(Matrix)
        return (Creation_Time,Modification_Time,Matrix)
    except:
        pass
MainAtomClassification()
```

IntegrityChecker.py

This file consists of code which can give us the next possible atom's size and type (acts like a lookahead)

```
import SupportFunctions
def integrity_checker(pointer,data):
    tempo=""
    moov_pointer=pointer
    for i in range(moov_pointer,moov_pointer+8):
        tempo+=data[i]
    temp_size=SupportFunctions.bigEnd(tempo)
    print()
    print("Next atom size is ", temp_size)
    tempo=""
    for i in range(moov_pointer+8,moov_pointer+16):
        tempo+=data[i]
    temp_atom=SupportFunctions.hex_to_ascii(tempo)
    print("Next atom is ",temp_atom,'\n')
    return temp_size, temp_atom
```

Stage-2

Frame Detectors (Key Frame Identification)

AMDFrameDetector.py

```
 import cv2
import os

# Set threshold for absolute difference
threshold = 15500000

# Create output directory to save the frames
output_dir = r'D:\File-Carving\Stage-2\ExtractedFrames\AMD_Frames'
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

# Open input video file
video_path = r"VideoSamples/Sample-1-HR.mp4"
cap = cv2.VideoCapture(video_path)

# Read first frame
ret, prev_frame = cap.read()
prev_gray = cv2.cvtColor(prev_frame, cv2.COLOR_BGR2GRAY)

try:
  # Loop over video frames and detect key frames
  count = 0
  while cap.isOpened():
    # Read next video frame
    ret, frame = cap.read()
    if not ret:
        break
    # Convert to grayscale and calculate absolute difference
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

```
    diff = cv2.absdiff(prev_gray, gray)

    # If difference is greater than threshold, save frame as output image
    if diff.sum() > threshold:
        frame_path = os.path.join(output_dir, f'frame_{count:04d}.png')
        cv2.imwrite(frame_path, frame)
        print('Saved key frame: {}'.format(frame_path))
        count += 1

    # Set current frame as previous frame for next iteration
    prev_gray = gray.copy()

except KeyboardInterrupt :
        print("AMD Frame Extraction Halted!!! \nNumber of Frames Extracted :" ,
count)
# Release input video file
cap.release()
cv2.destroyAllWindows()
```

BGSubFrameDetector.py

```
# Function to perform file carving based on background subtraction
# This code extracts frames from a video file based on background subtraction and
saves only
# those frames where significant changes in foreground objects are detected.
# The foreground objects identified by the background subtraction algorithm are
represented by white pixels in the resulting binary imag
# This binary image can be used to extract the foreground objects from the original
frame, for example by applying a mask.

import cv2
import os

# Create output directory to save the frames
```

```
output_dir = r'D:\File-Carving\Stage-2\ExtractedFrames\BGSub_Frames'
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

Foreground_Obj_Dir = r'D:\File-Carving\Stage-
2\ExtractedFrames\RMS_Frames\ForeGroundObjects'
if not os.path.exists(Foreground_Obj_Dir):
    os.makedirs(Foreground_Obj_Dir)

# Open input video file
video_path = r"VideoSamples/Sample-1-HR.mp4"
cap = cv2.VideoCapture(video_path)

# Define ROI for background subtraction
roi = (500, 200, 800, 500)

# Initialize variables
count = 0
frame_num = 0
try :
    # Loop through video frames and carve out frames based on background
subtraction
    while True:
        # Read next video frame
        ret, frame = cap.read()
        if not ret:
            break

        # Skip frames to reduce overall number of frames processed
        if frame_num % 5 != 0:
            frame_num += 1
            continue

        # Get ROI from the current frame
```

```
        x, y, w, h = roi
        roi_frame = frame[y:y + h, x:x + w]

        # Convert ROI to grayscale
        gray_roi = cv2.cvtColor(roi_frame, cv2.COLOR_BGR2GRAY)

        # Apply Gaussian blur to the ROI
        blur_roi = cv2.GaussianBlur(gray_roi, (5, 5), 0)

        # Define background subtractor
        fgbg = cv2.createBackgroundSubtractorMOG2(history=500,
varThreshold=99999, detectShadows=False)

        # Apply background subtraction to ROI
        fgmask = fgbg.apply(blur_roi)

        # If significant changes in the foreground are detected, save frame
        if cv2.countNonZero(fgmask) > 5000:
            cv2.imwrite(os.path.join(output_dir, "frame{:06d}.png".format(count)),
frame)
            print("Saved: ", "frame{:06d}.png".format(count))
            count += 1
    # Display the foreground objects identified
        cv2.imwrite(os.path.join(Foreground_Obj_Dir,
"Foreground_Object{:06d}.png".format(count)), fgmask)
        print("Foreground Object Saved: ",
"Foreground_Object{:06d}.png".format(count))

        # Update frame number
        frame_num += 1
except KeyboardInterrupt :
        print("AMD Frame Extraction Halted!!! \nNumber of Frames Extracted :" ,
count)
# Release video file
```

```
cap.release()

# Print message to indicate completion
print("background Substraction completed")
```

RMSFrameDetector.py

```
import cv2
import numpy as np
import os

# Set video path
video_path = r"VideoSamples/Sample-1-HR.mp4"
cap = cv2.VideoCapture(video_path)

# set the output directory path to save the frames
output_dir = r'D:\File-Carving\Stage-2\ExtractedFrames\RMS_Frames'

# Create output directory if it doesn't exist
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

# Set threshold value for frame difference
threshold = 9

# Read first frame
ret, previous_frame = cap.read()

# Convert frame to grayscale
# previous_frame = cv2.cvtColor(previous_frame, cv2.COLOR_BGR2GRAY)
previous_frame = cv2.cvtColor(previous_frame, cv2.COLOR_BGR2RGB)

# Set frame counter
frame_counter = 0
```

```
try:
    # Loop through video frames
    while(cap.isOpened()):

        # Read current frame
        ret, current_frame = cap.read()

        # Break loop if end of video
        if not ret:
            break

        # Convert current frame to grayscale
        # current_frame = cv2.cvtColor(current_frame, cv2.COLOR_BGR2GRAY)
        current_frame = cv2.cvtColor(current_frame, cv2.COLOR_BGR2RGB)

        # Compute absolute difference between current frame and previous frame
        diff = cv2.absdiff(current_frame, previous_frame)

        # Compute root mean square difference between current frame and previous
frame
        rms_diff = np.sqrt(np.mean(np.square(diff)))

        # If RMS difference exceeds threshold, save frame to output directory
        if rms_diff > threshold:
            current_frame_rgb = cv2.cvtColor(current_frame, cv2.COLOR_BGR2RGB)
            output_path = os.path.join(output_dir,
'frame_{0:04d}.png'.format(frame_counter))
            cv2.imwrite(output_path, current_frame_rgb)
            frame_counter += 1
            print('Saved key frame: {}'.format(output_path))

        # Set previous frame to current frame for next iteration
        previous_frame = current_frame
```

```
except KeyboardInterrupt:
    print("RMS Frame Extraction Halted!!! \nNumber of Frames Extracted :",
frame_counter)

# Release video capture and destroy windows
cap.release()
cv2.destroyAllWindows()
```

Stage-3

AttributeRedundancy.py

```
import datetime
import sys
sys.path.append(r'C:\\Users\Suddala Kavyasree\Desktop\\File Carving\\Stage-1\\')
import AtomClassification
creation,modif,matrix=AtomClassification.MainAtomClassification()
Final_Creation_Time=''
Final_Modification_Time=''
Final_Matrix=''
# 1.Validation checks (timestamp verification-format,future date,expired date wrt
epoch,non-negative,modif>=creation)
# 2.Attribute value finalization based on frequency of valid values
def FrequencyBasedAttributeRedundancy(original_dict,name):
    value_freq = {}
    for key, value in original_dict.items():
        if value not in value_freq:
            value_freq[value] = 1
        else:
            value_freq[value] += 1
    if name=='Creation_Time':
        d=value_freq.copy()
        for k,v in d.items():
```

```
            if is_timestamp(k) and is_datetime_less_than_current(k) and
is_datetime_greater_than_StandardTime(k) and
is_creationTime_less_than_modificationTime(k,Final_Modification_Time) and
is_valid_timestamp(k):
                pass
            else:
                print("\n"+name+": "+k+" is invalid!!!!\n")
                value_freq.pop(k)
    elif name=='Modification_Time':
        d=value_freq.copy()
        for k,v in d.items():
            if is_timestamp(k) and is_datetime_less_than_current(k) and
is_datetime_greater_than_StandardTime(k) and is_valid_timestamp(k):
                pass
            else:
                print("\n"+name+": "+k+" is invalid!!!!\n")
                value_freq.pop(k)
    most_common_value = max(value_freq, key=value_freq.get)
    return most_common_value
# if all are corrupted values, give the datetime.now()

def atom_seeker(data,pointer,atom):
    cur_pointer=pointer
    tempo = ""
    temp = ""
    while (temp != atom):
        temp = ""
        tempo = ""
        for k in range(cur_pointer, cur_pointer+8):
            tempo += data[k]
        try:
            temp = AtomClassification.hex_to_ascii(tempo)
        except:
            pass
```

```
        cur_pointer += 1
    moov_pointer = cur_pointer-1
    moov_pointer -= 8
    return moov_pointer

def Reconstruction(creation,modif,matrix):
    #Sample-2-HR fragment is considered from ftyp to dinf
    # MVHD corruption is induced
    # Atoms effected are mvhd,tkhd and mdhd for creation time and modification time
    # Atoms effetcted are mvhd and tkhd for matrix
    f2=open('Stage-3\RedundancyPartialFileSamples\FixedPartialFile.txt','w')
    with open('Stage-3\RedundancyPartialFileSamples\CorruptedPartialFile1.txt', "r")
as f1:
        data=str((f1.read()))
        mvhd_position=atom_seeker(data,0,'mvhd')
        tkhd_position=atom_seeker(data,0,'tkhd')
        mdhd_position=atom_seeker(data,0,'mdhd')
        pointer=0
        while(pointer<len(data)):
          if pointer!=mvhd_position and pointer!=tkhd_position and
pointer!=mdhd_position:
              f2.write(data[pointer])
              pointer+=1
          if pointer==mvhd_position:
              f2.write(data[pointer:pointer+24])
              pointer=pointer+24

curr_creation=AtomClassification.convert_hex_to_datetime(data[pointer:pointer+8])

curr_modifi=AtomClassification.convert_hex_to_datetime(data[pointer+8:pointer+16
])
              final_creation=AtomClassification.convert_string_to_hex(creation)
              final_modif=AtomClassification.convert_string_to_hex(modif)
```

```
            if curr_modifi!=curr_creation and curr_modifi!=final_creation and
curr_modifi!=final_modif:
                # if modification time is deleted and other attribute value is read in its
place
                f2.write(final_creation)
                f2.write(final_modif)
                pointer=pointer+8
            else:
                if curr_creation != final_creation or '_' in curr_creation or '?' in
curr_creation:
                    f2.write(final_creation)
                else:
                    f2.write(curr_creation)
                if curr_modifi!=final_modif or '-' in curr_modifi or '?' in curr_modifi:
                    f2.write(final_modif)
                else:
                    f2.write(curr_modifi)
                pointer=pointer+16
            f2.write(data[pointer:pointer+48])
            pointer=pointer+48
            f2.write(matrix)
            pointer=pointer+72
            f2.write(data[pointer:pointer+56])
            pointer=pointer+56
        if pointer==tkhd_position:
            f2.write(data[pointer:pointer+24])
            pointer=pointer+24

curr_creation=AtomClassification.convert_hex_to_datetime(data[pointer:pointer+8])

curr_modifi=AtomClassification.convert_hex_to_datetime(data[pointer+8:pointer+16
])
            final_creation=AtomClassification.convert_string_to_hex(creation)
            final_modif=AtomClassification.convert_string_to_hex(modif)
```

```
            if curr_modifi!=curr_creation and curr_modifi!=final_creation and
curr_modifi!=final_modif:
                # if modification time is deleted and other attribute value is read in its
place
                f2.write(final_creation)
                f2.write(final_modif)
                pointer=pointer+8
            else:
                if curr_creation != final_creation or '_' in curr_creation or '?' in
curr_creation:
                    f2.write(final_creation)
                else:
                    f2.write(curr_creation)
                if curr_modifi!=final_modif or '-' in curr_modifi or '?' in curr_modifi:
                    f2.write(final_modif)
                else:
                    f2.write(curr_modifi)
                pointer=pointer+16
            f2.write(data[pointer:pointer+56])
            pointer=pointer+56
            f2.write(matrix)
            pointer=pointer+72
            f2.write(data[pointer:pointer+16])
            pointer=pointer+16
        if pointer==mdhd_position:
            f2.write(data[pointer:pointer+24])
            pointer=pointer+24

curr_creation=AtomClassification.convert_hex_to_datetime(data[pointer:pointer+8])

curr_modifi=AtomClassification.convert_hex_to_datetime(data[pointer+8:pointer+16
])
            final_creation=AtomClassification.convert_string_to_hex(creation)
            final_modif=AtomClassification.convert_string_to_hex(modif)
```

```
            if curr_modifi!=curr_creation and curr_modifi!=final_creation and
curr_modifi!=final_modif:
                # if modification time is deleted and other attribute value is read in its
place
                f2.write(final_creation)
                f2.write(final_modif)
                pointer=pointer+8
            else:
                if curr_creation != final_creation or '_' in curr_creation or '?' in
curr_creation:
                    f2.write(final_creation)
                else:
                    f2.write(curr_creation)
                if curr_modifi!=final_modif or '-' in curr_modifi or '?' in curr_modifi:
                    f2.write(final_modif)
                else:
                    f2.write(curr_modifi)
                pointer=pointer+16
            f2.write(data[pointer:pointer+24])
            pointer=pointer+24

Final_Modification_Time=FrequencyBasedAttributeRedundancy(modif,'Modification
_Time')
Final_Creation_Time=
FrequencyBasedAttributeRedundancy(creation,'Creation_Time')
Final_Matrix=FrequencyBasedAttributeRedundancy(matrix,'Matrix')
print("\nFinal Values of Redundant Attributes based on Frequency:")
print('\nFinal Creation Time:----------\n')
print(Final_Creation_Time)
print('\nFinal Modification Time:----------\n')
print(Final_Modification_Time)
print('\nFinal Matrix Value:----------\n')
print(Final_Matrix)
print()
```

```
Reconstruction(creation=Final_Creation_Time,modif=Final_Modification_Time,matr
ix=Final_Matrix)
```

StructuralEvidenceCollector.py

```
import ffmpeg
from hachoir.parser import createParser
from hachoir.metadata import extractMetadata
from hachoir.core import config
import numpy as np
import hashlib
import cv2
import magic
import sys
sys.path.append(r'C:\\Users\Suddala Kavyasree\Desktop\\File Carving\\Stage-1\\')

l = ['ftyp','edts', 'free', 'mdat', 'moov', 'mvhd', 'trak', 'tkhd', 'mdia', 'mdhd', 'hdlr', 'minf',
'smhd', 'vmhd', 'dinf', 'dref', 'url',
     'stbl', 'stsd', 'stts', 'stss', 'stsc', 'stsz', 'stco', 'ctts', 'sdtp', 'stps', 'cslg', 'udta', 'sgpd',
'sbgp', 'elst', 'iods']
major_brands_dict = {
    'msnv': 'MobiSaver video format',
    'mss1': 'MSS1 video file format',
    'mss2': 'MSS2 video file format',
    'mts1': 'AVCHD video file format',
    'mts2': 'AVCHD'}

def IsVideoFile_Based_On_AtomPresence(hex_string):
    # Evidence 1 4 11 19
    ascii_string = bytes.fromhex(hex_string).decode('ascii')
    isVideo = False
    for atom in l:
        if atom in ascii_string:
```

```
            isVideo = True
            break
    for ext in major_brands_dict:
        if ext in ascii_string:
            isVideo = True
    return isVideo

    # print(IsVideoFile_Based_On_AtomPresence('48656c6c6f20576f726c64'))

def IsVideoFile_Based_On_Ftyp(hex_string):
    # Evidence 2
    # Checks if it is a video file , after identifying ftyp atom
    ascii_string = bytes.fromhex(hex_string).decode('ascii')
    isVideo = False
    if 'ftyp' in ascii_string:
        ftyp_pos = ascii_string.find('ftyp')
        major_brand = ascii_string[ftyp_pos+4:ftyp_pos+8]
        if major_brand in major_brands_dict.keys():
            isVideo = True
        return isVideo
#
print(IsVideoFile_Based_On_Ftyp('00000020667479706D703432000000006D70343
2'))
def IsVideoFile_Based_On_Attributes(hex_string):
    # Evidence 3
    ll=[]
    ascii_string = bytes.fromhex(hex_string).decode('latin-1')
    for atom in l:
        if atom in ascii_string:
            ll.append(atom)
    for cur_atom in ll:
        atom_pos = ascii_string.find(str(cur_atom))
        return pointerr
import pymp4parse
```

```
def IsVideoFile_Based_On_Checksum(filename):
    # Evidence 10
    # Open the file in binary mode
    with open(filename, 'rb') as f:
        # Calculate the SHA256 hash of the file
        hash_object = hashlib.sha256()
        hash_object.update(f.read())
        file_hash = hash_object.hexdigest()

        # Compare the file hash to known video file hashes
        video_hashes = ['a1b2c3d4e5f6g7h8i9j0', 'k1l2m3n4o5p6q7r8s9t0',
'u1v2w3x4y5z6a7b8c9d0']
        if file_hash in video_hashes:
            return True

    return False

def StructuralEvidenceBasedVideoVerification():

hex_string='00000020667479706D703432000000006D7034326D70343169736F6D6
1766331000033'
    filename=r"C:\\Users\Suddala Kavyasree\Desktop\\File
Carving\\VideoSamples\Sample-2-HR.mp4"

    print('Based_On_AtomPresence:
',IsVideoFile_Based_On_AtomPresence(hex_string),'\n')
    print('Based_On_Ftyp: ',IsVideoFile_Based_On_Ftyp(hex_string),'\n')
    print('Based_On_Attributes: ',IsVideoFile_Based_On_Attributes(hex_string),'\n')
    print('Based_On_MagicSequence:
',IsVideoFile_Based_On_MagicSequence(filename),'\n')
    print('Based_On_Frames: ',IsVideoFile_Based_On_Frames(filename),'\n')
    print('Based_On_BytesDistribution:
',IsVideoFile_Based_On_BytesDistribution(filename),'\n')
```

```
    print('Based_On_Checksum: ',IsVideoFile_Based_On_Checksum(filename),'\n')
    print('Based_On_Dimensions: ',IsVideoFile_Based_On_Dimensions(filename),'\n')
    print('Based_On_CompressionTechnique:
',IsVideoFile_Based_On_CompressionTechnique(filename),'\n')
    print('Based_On_MotionVectors:
',IsVideoFile_Based_On_MotionVectors(filename),'\n')
    print('Based_On_Pixellation: ',IsVideoFile_Based_On_Pixellation(filename),'\n')
    print(IsVideoFile_Based_On_TemporalSequence(filename))

StructuralEvidenceBasedVideoVerification()
```

FragmentRecovery.py

```
atoms = ['ftyp','edts', 'free', 'mdat', 'moov', 'mvhd', 'trak', 'tkhd', 'mdia', 'mdhd', 'hdlr',
'minf', 'smhd', 'vmhd', 'dinf', 'dref', 'url',
     'stbl', 'stsd', 'stts', 'stss', 'stsc', 'stsz', 'stco', 'ctts', 'sdtp', 'stps', 'cslg', 'udta', 'sgpd',
'sbgp', 'elst', 'iods']
major_brand_list=['isom','iso2','avc1','mp41']
import sys
sys.path.append(r'C:\\Users\Suddala Kavyasree\Desktop\\File Carving\\Stage-1\\')
from UpdatedClassifiers import ftyp_classifier,stbl_classifier, edts_classifier,
udta_classifier, free_classifier, mdat_classifier, moov_classifier, mvhd_classifier,
trak_classifier, tkhd_classifier, mdia_classifier, mdhd_classifier, hdlr_classifier,
minf_classifier, vmhd_classifier, smhd_classifier, dinf_classifier, dref_classifier,
sample_atom_classifier, elst_classifier, iods_classifier

import os
directory_path = r"C:\\Users\Suddala Kavyasree\Desktop\\File Carving\\Stage-
3\\FileSamples"
file_list = os.listdir(directory_path)
# print(file_list)

def Recover(hex_string):
```

```
    ascii_string = bytes.fromhex(hex_string).decode('latin-1')
    print('\nAscii String is: '+ascii_string)
    ss=ascii_string
    atoms_present=[]
    partial_atoms_present=[]
    for atom in atoms:
        if atom in ascii_string:
            atoms_present.append(atom)
            ss=ss.replace(atom,'')
    print('\nFull Atoms present in the given fragment are: ',atoms_present)
    for atom in atoms:
        if atom not in atoms_present:
            l=[atom[i:i+2] for i in range(len(atom)-1)]
            for i in l:
                if i in ss:
                    if atom not in partial_atoms_present:
                        partial_atoms_present.append(atom)
    if len(partial_atoms_present)>0:
        print('\nAtoms(Possibility) present in the given fragment are:
',partial_atoms_present)

    if 'ftyp' in ascii_string:
        print('\nFTYP atom found!! Size of ftyp atom is 32 bytes and recovered fragment
is '+('00000020'+hex_string))

    if 'free' in ascii_string:
        print('\nFREE atom found!! Size of free atom is 8 bytes and recovered fragment
is '+('00000008'+hex_string))

    for i in major_brand_list:
        if i in ascii_string and 'ftyp' not in atoms_present and 'ftyp' not in
partial_atoms_present:
            print('\nThere is a possibility of occurence of ftyp as we encountered '+i)
            break
```

```
    found=False
    for atom in partial_atoms_present+atoms_present:
        if atom == ascii_string[-4:]:
            found=True
            print('\nThere is a possibility of occurence of atom '+atom.upper()+' after all atoms in current file fragment\n')
            break
    if found==False:
        for atom in partial_atoms_present+atoms_present:
            l=[atom[i:i+2] for i in range(len(atom)-1)]
            for i in l:
                if i in ascii_string[-4:]:
                    print('\nThere is a possibility of occurence of atom '+atom.upper()+' after all atoms in current file fragment\n')
```

4.2 Results

This is a pilot code which is the base code for interactive console

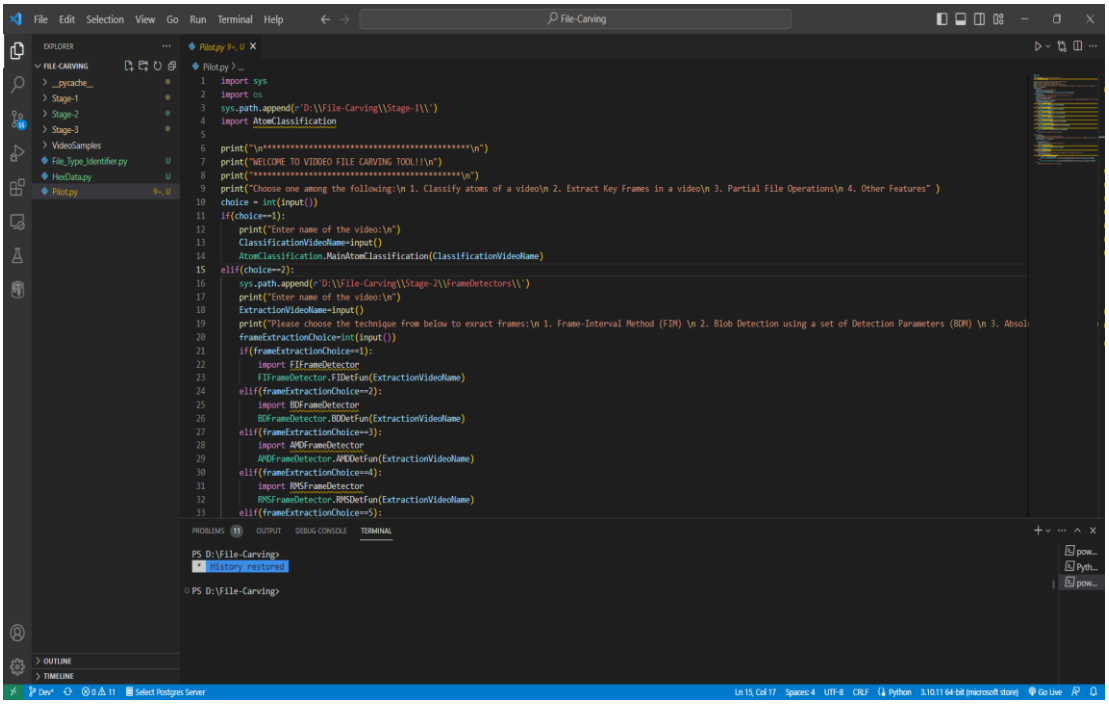


Figure-1 : Overview

The interactive console asks for user input based on the given choices

Figure-2: Initial Interface

The following outputs show the atom classification details. Given a video file/ partial fragment, it classifies the file into various mp4 atoms and the attribute values of atoms are set after parsing the hex data using classifier functions.

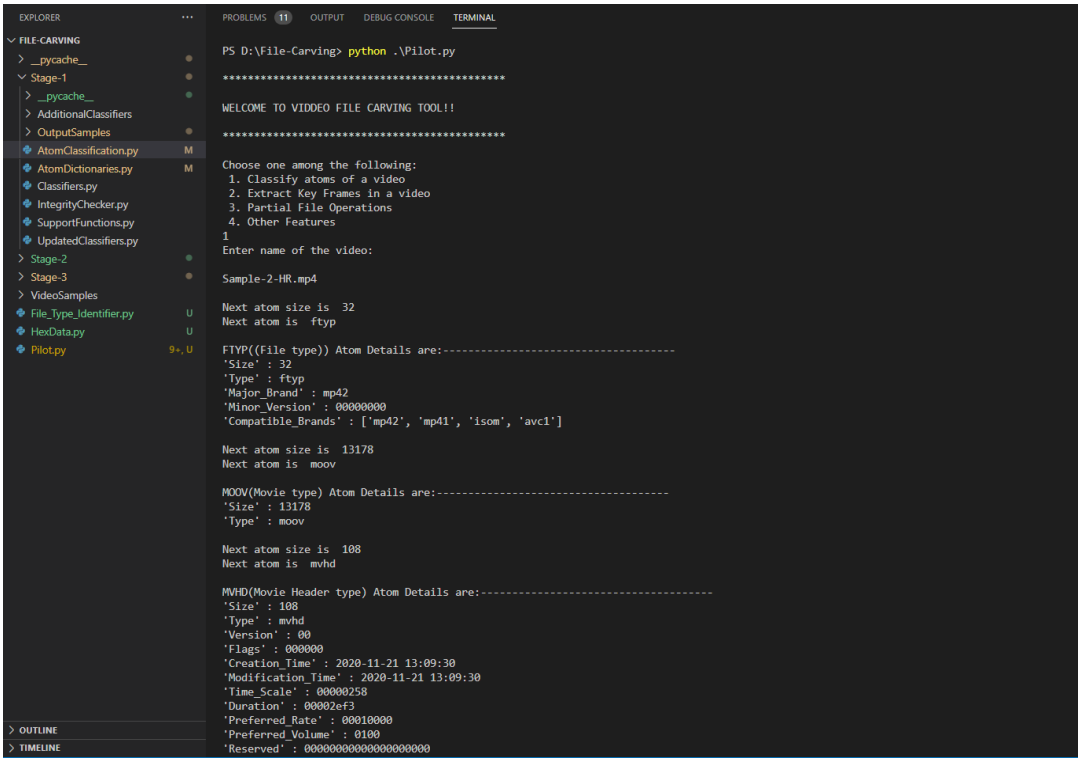


Figure-3: Atom Classification -1

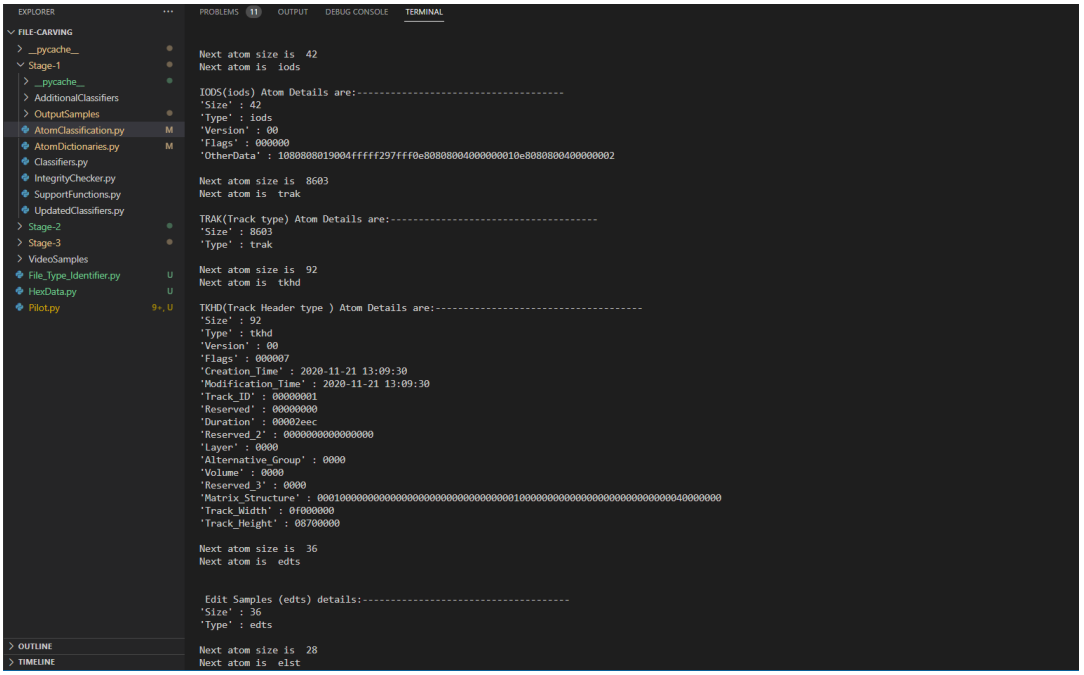


Figure-4: Atom Classification -2

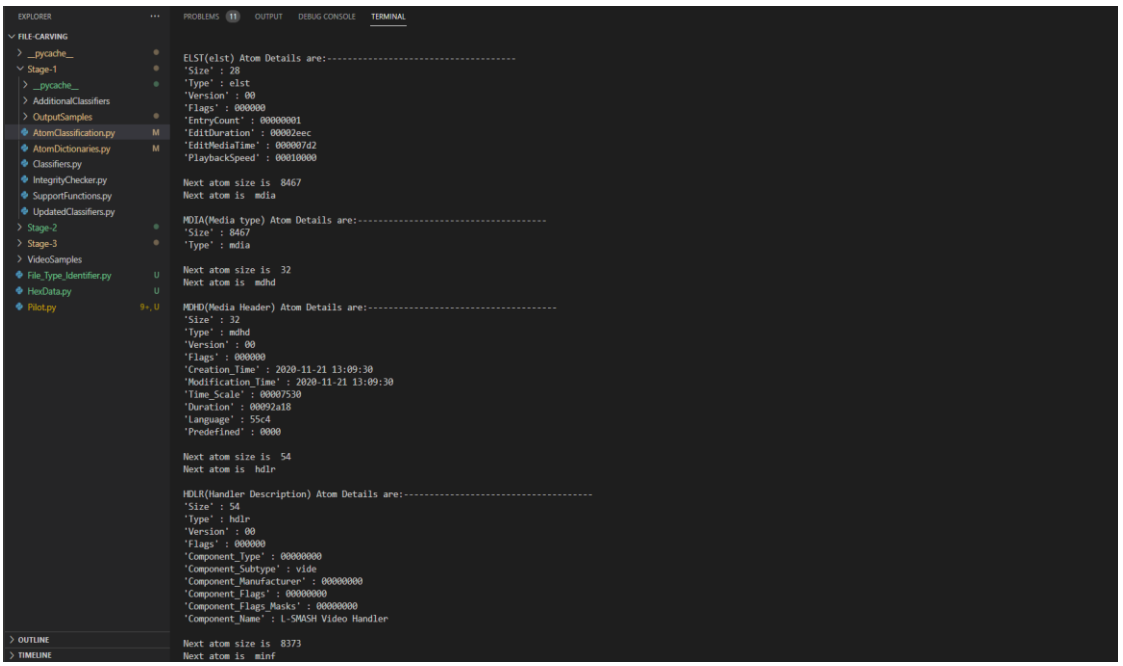


Figure-5: Atom Classification -3

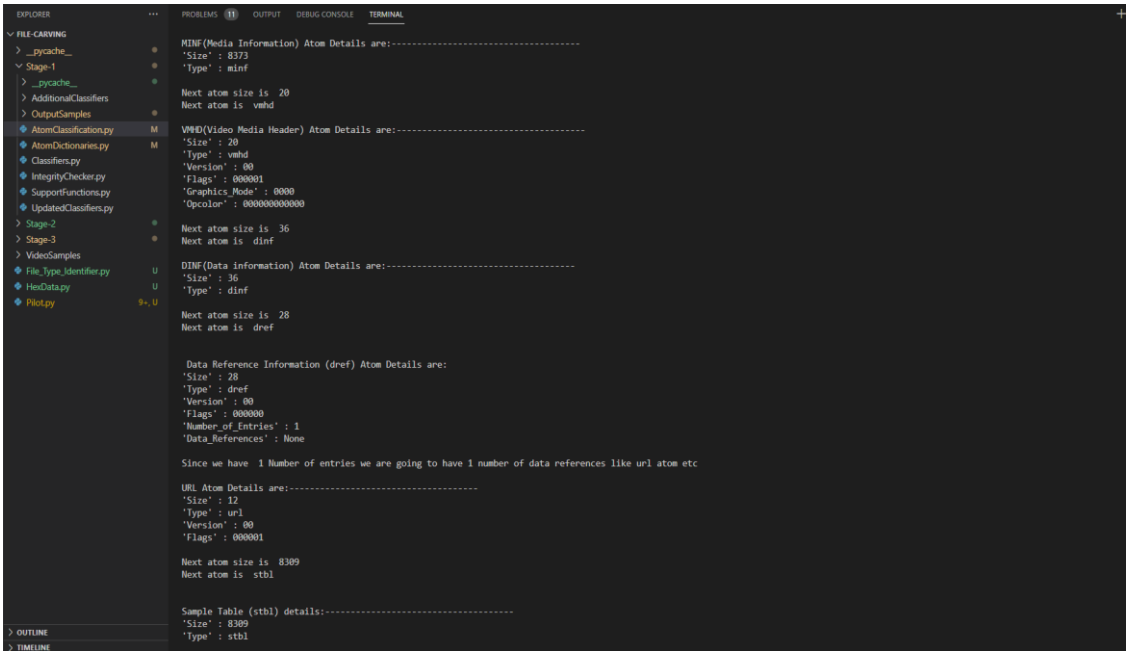


Figure-6: Atom Classification - 4

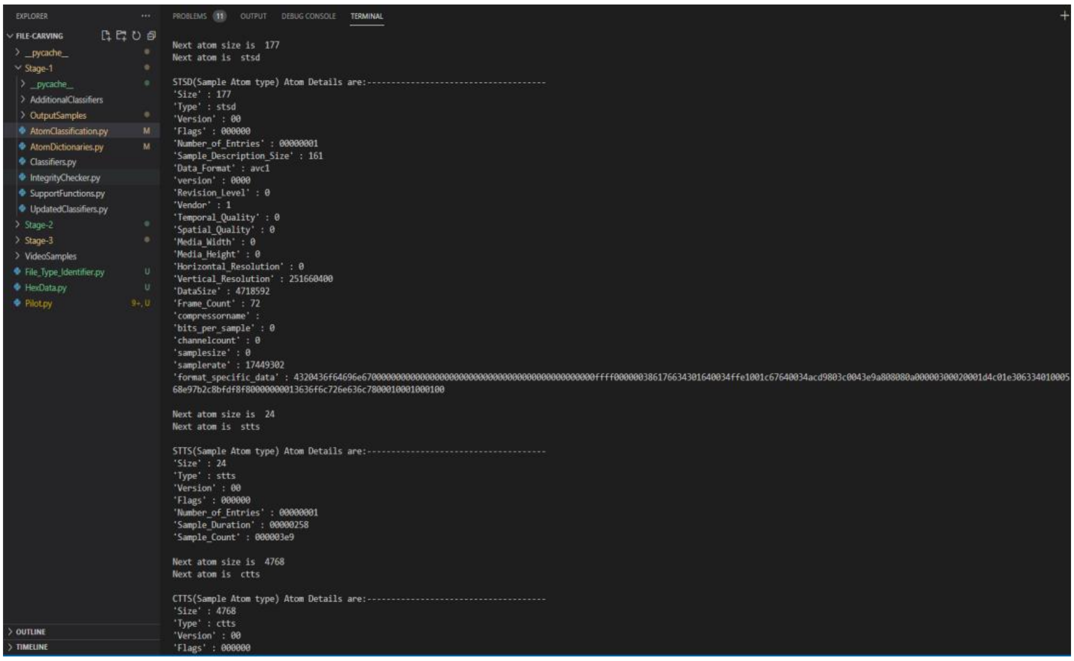


Figure-7: Atom Classification - 5

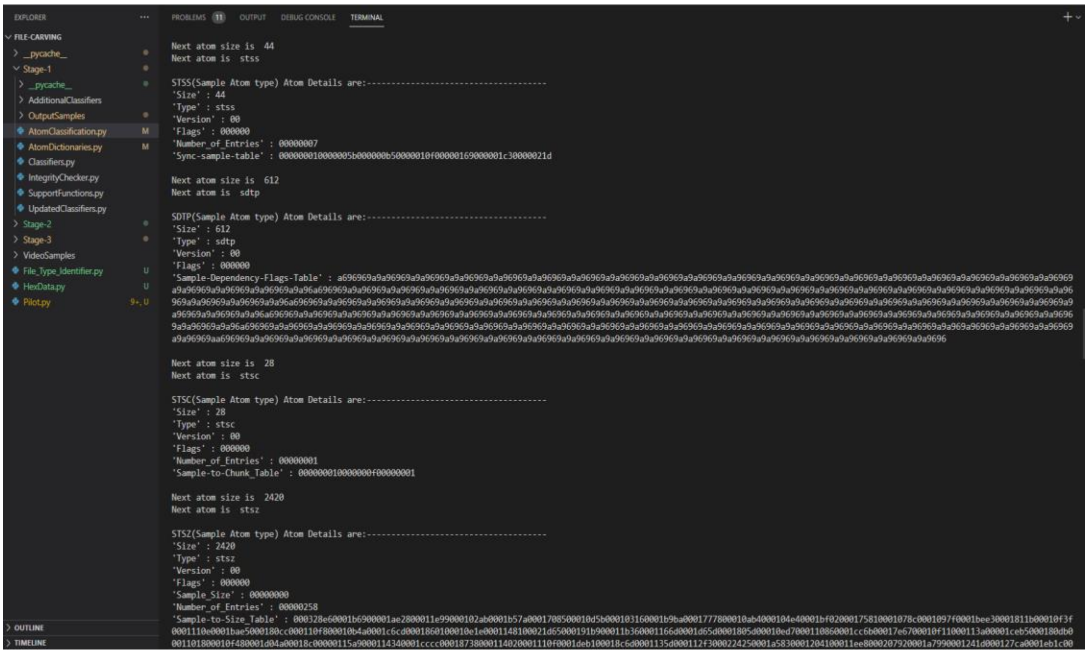


Figure-8: Atom Classification - 6

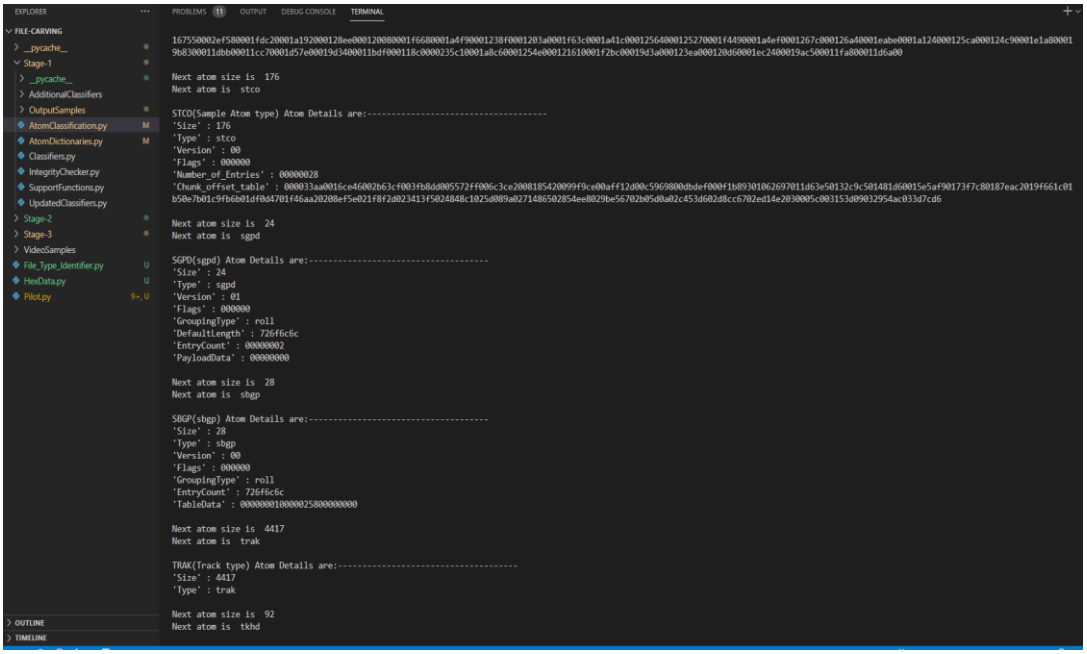


Figure-9: Atom Classification - 7

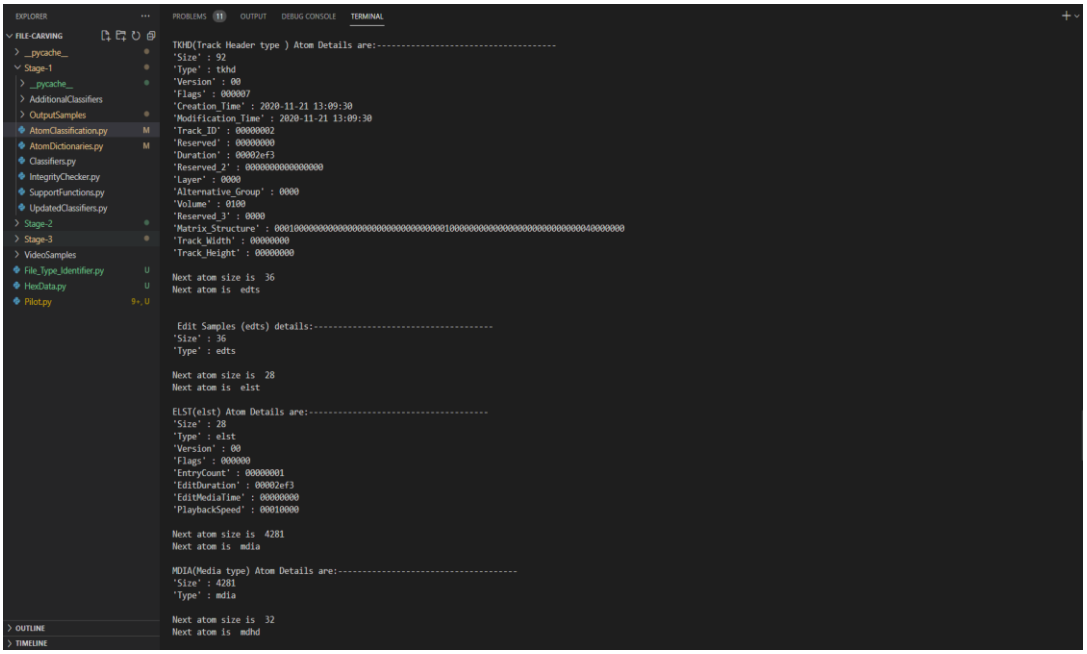


Figure-10: Atom Classification - 8

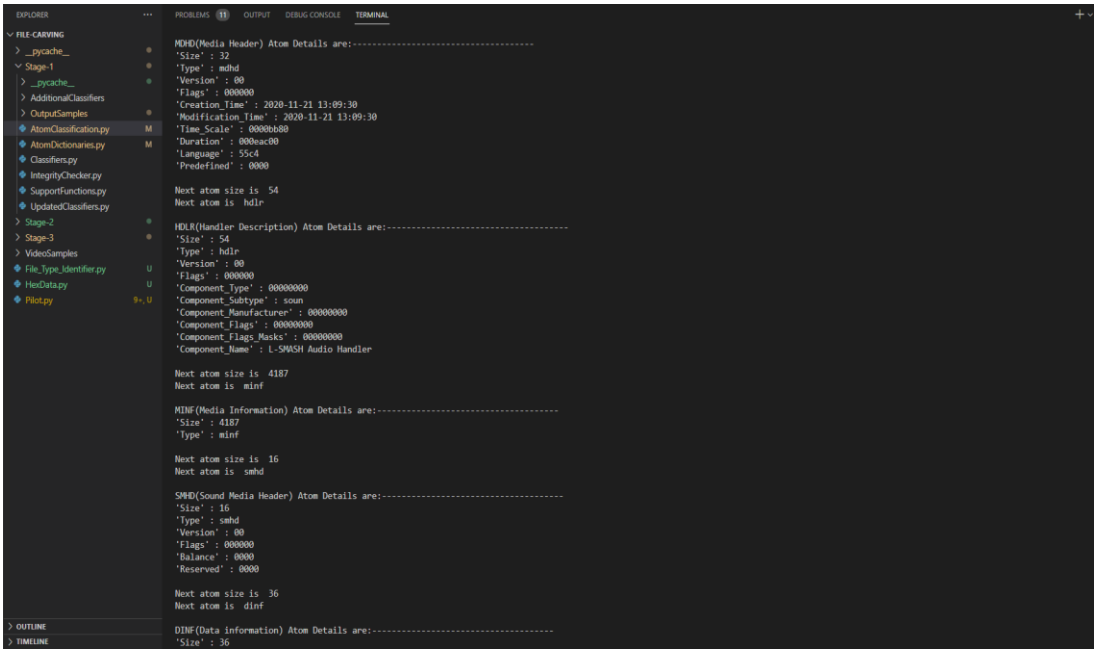


Figure-11: Atom Classification - 9

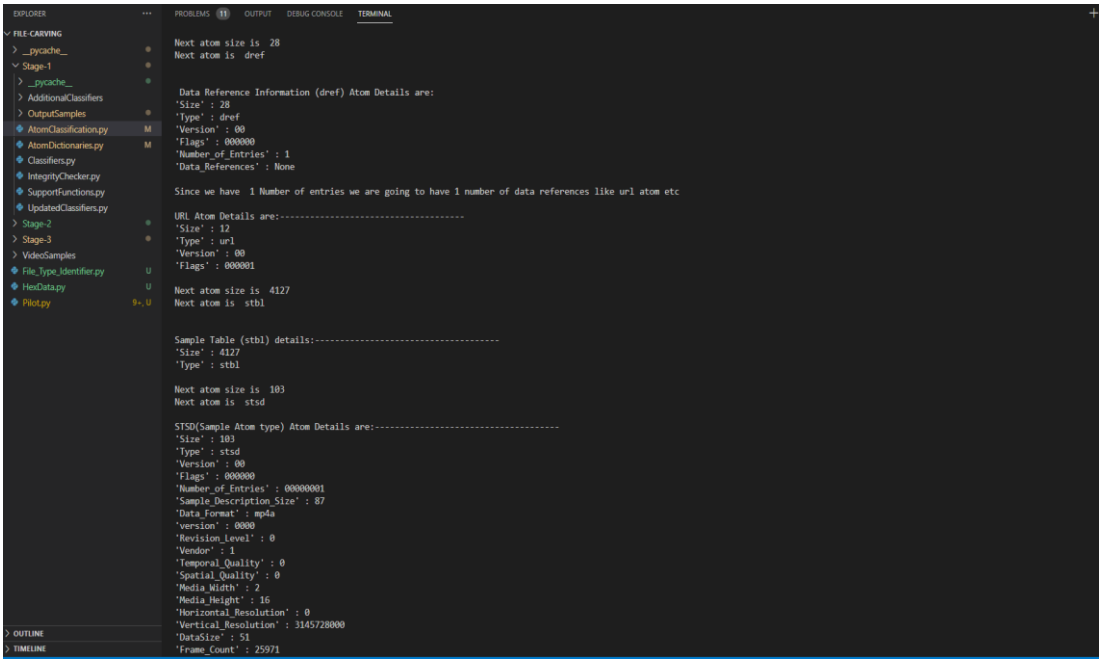


Figure-12: Atom Classification -10

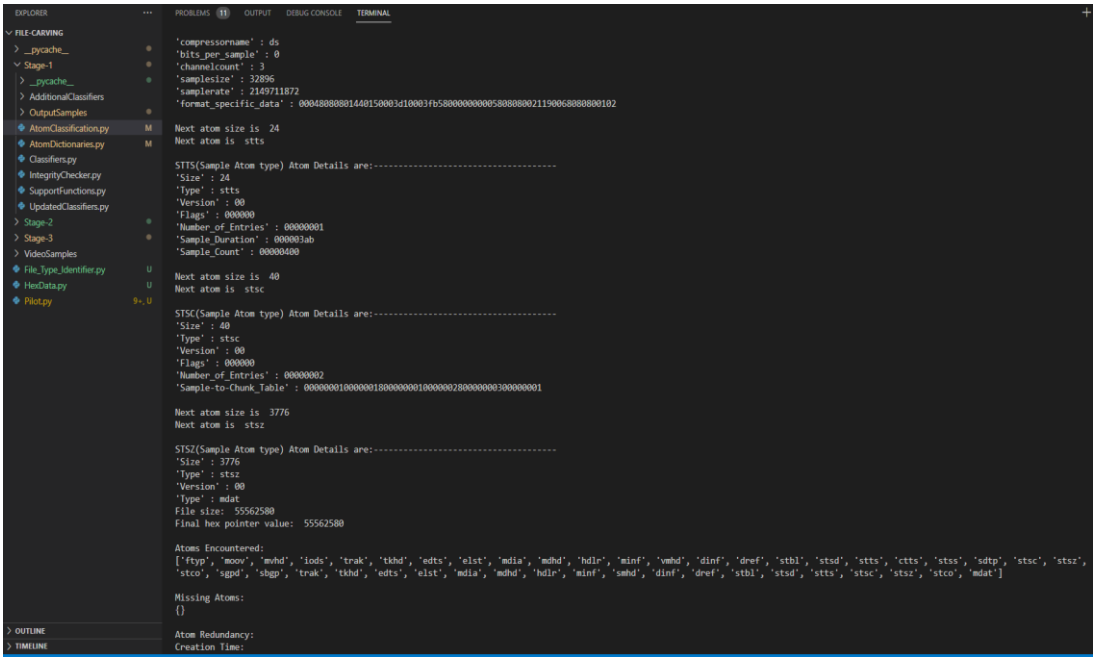


Figure-13: Atom Classification -11

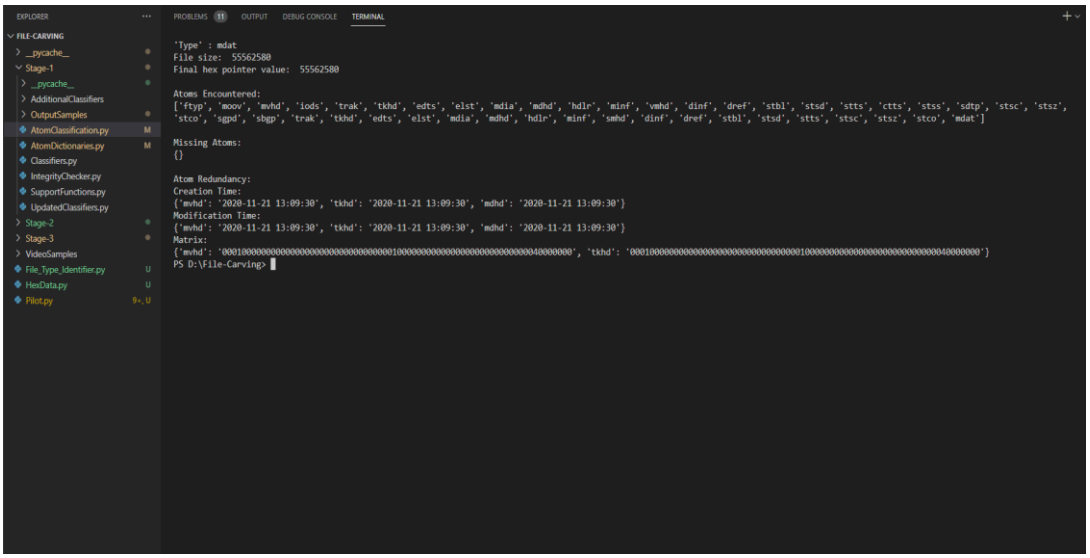


Figure-14: Atom Classification -12

Below outputs show the frame extraction interface. There are 7 methods using which user can extract key frames of a video file.

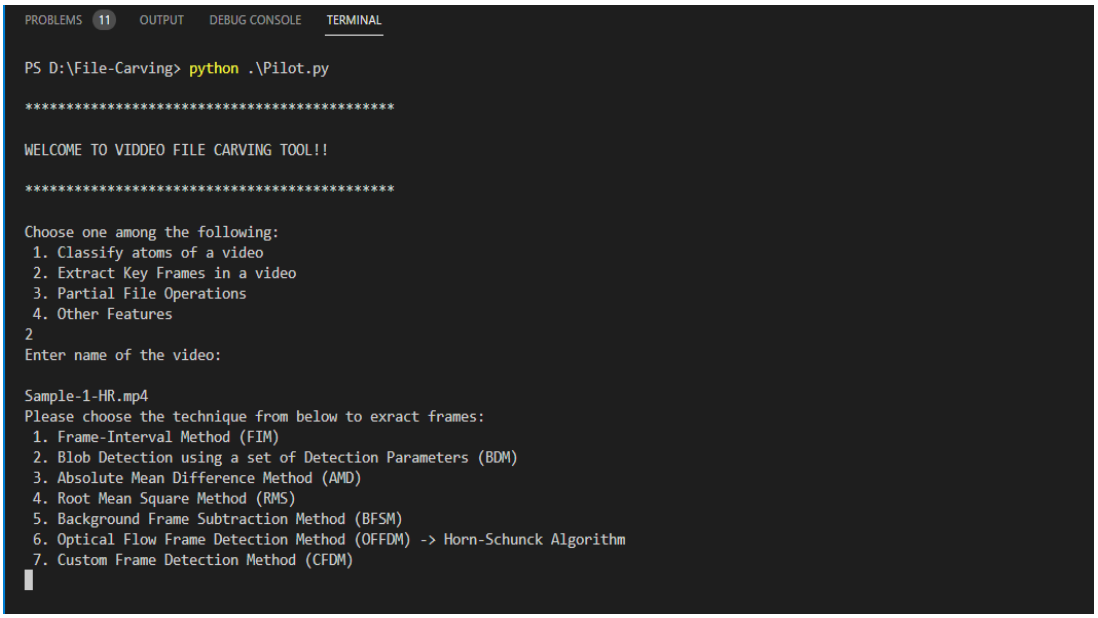


Figure-15: Frame Extraction Interface

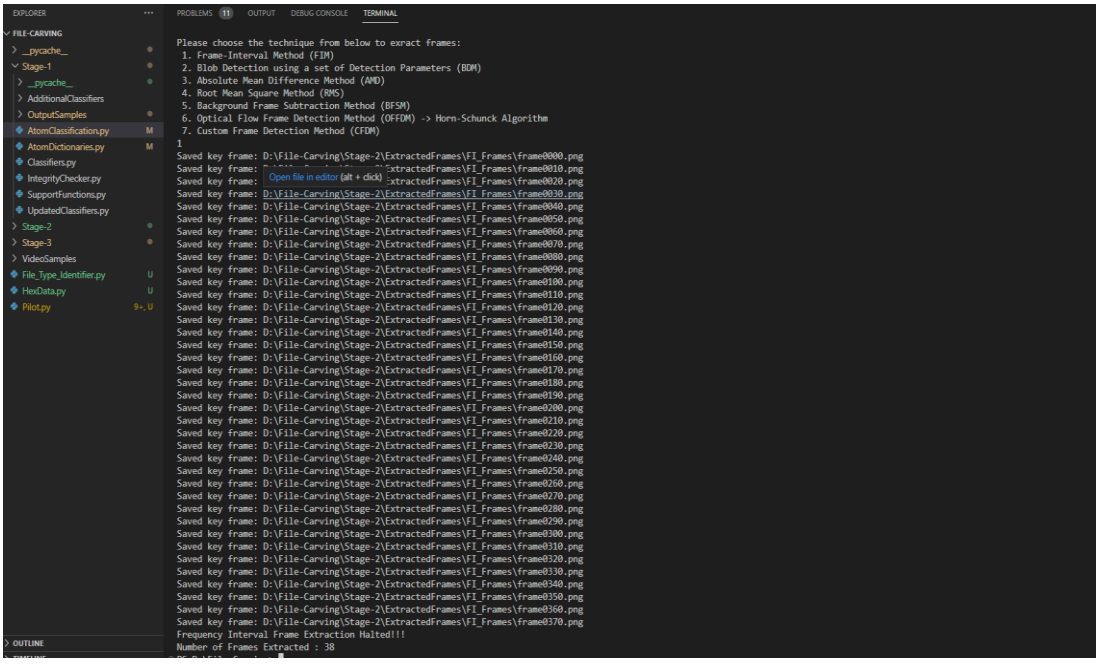


Figure-16: Frame Extraction – Frame Interval Method

The following outputs are of FI Frame extraction method. "Fi" frame extraction method is a technique used in video processing to extract specific frames from a video based on their position within a sequence of frames. "Fi" stands for "frame interval", which refers to the number of frames between the extracted frames. For example, if we want to extract every 10th frame from a video, we can use the "Fi" frame extraction method with a frame interval of 10

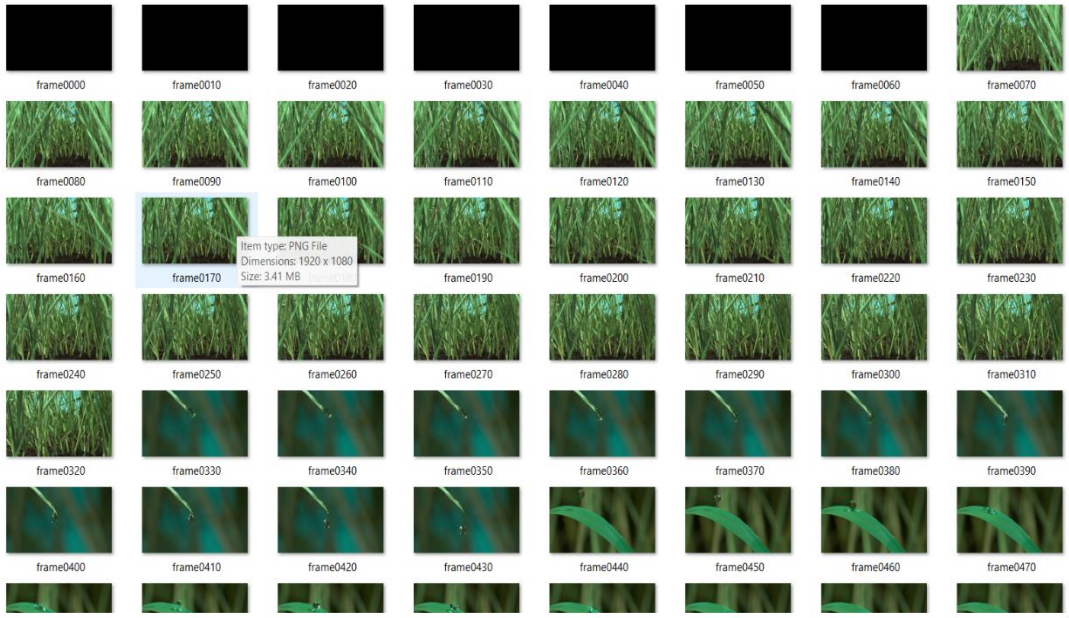


Figure-17: Frame Interval Frame Extraction - 1

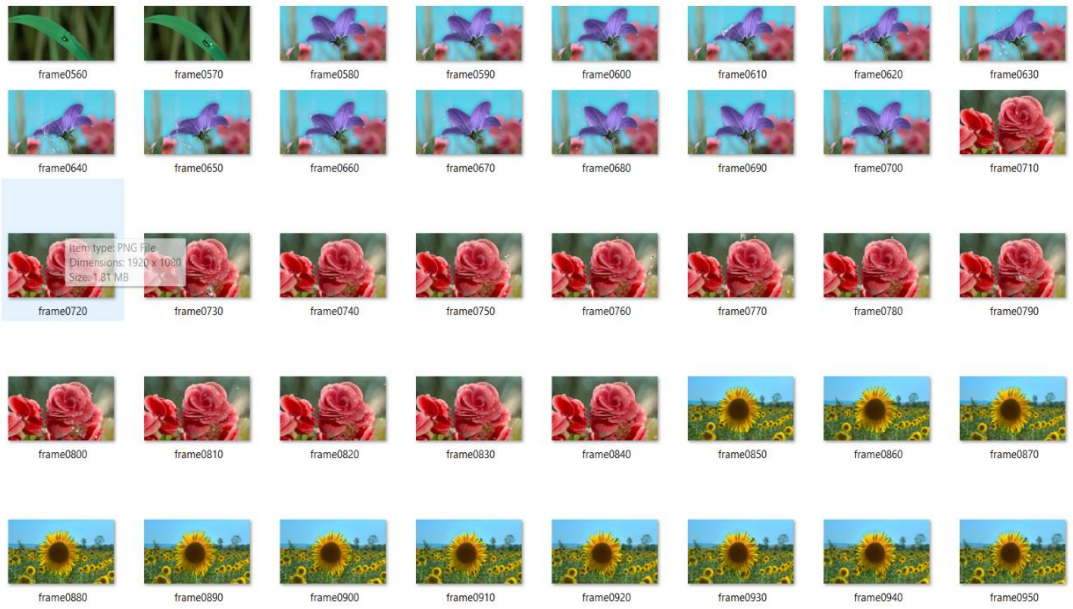


Figure-18: Frame Interval Frame Extraction - 2

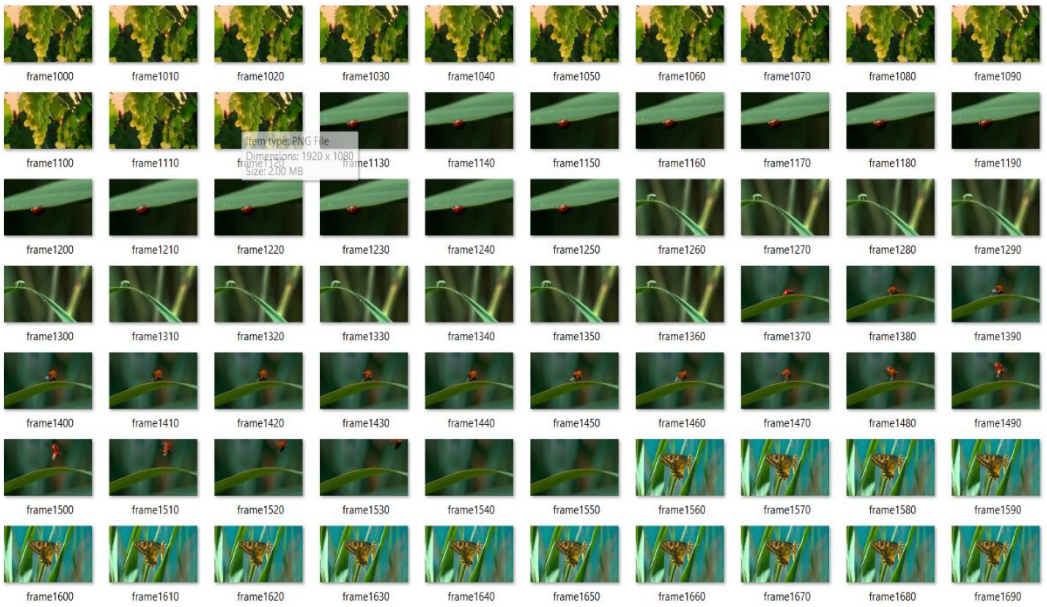


Figure-19: Frame Interval Frame Extraction - 3

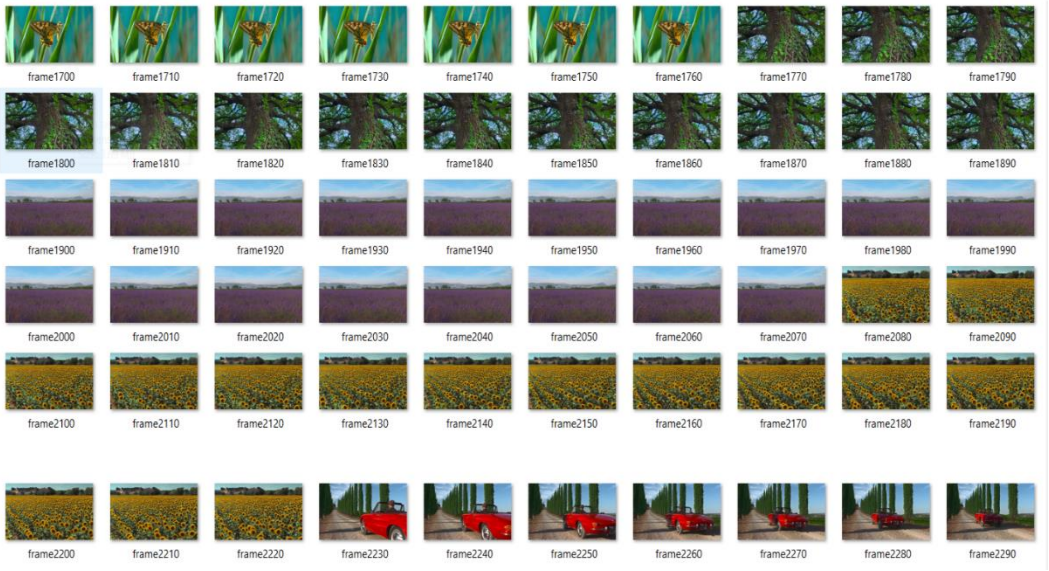


Figure-20: Frame Interval Frame Extraction – 4

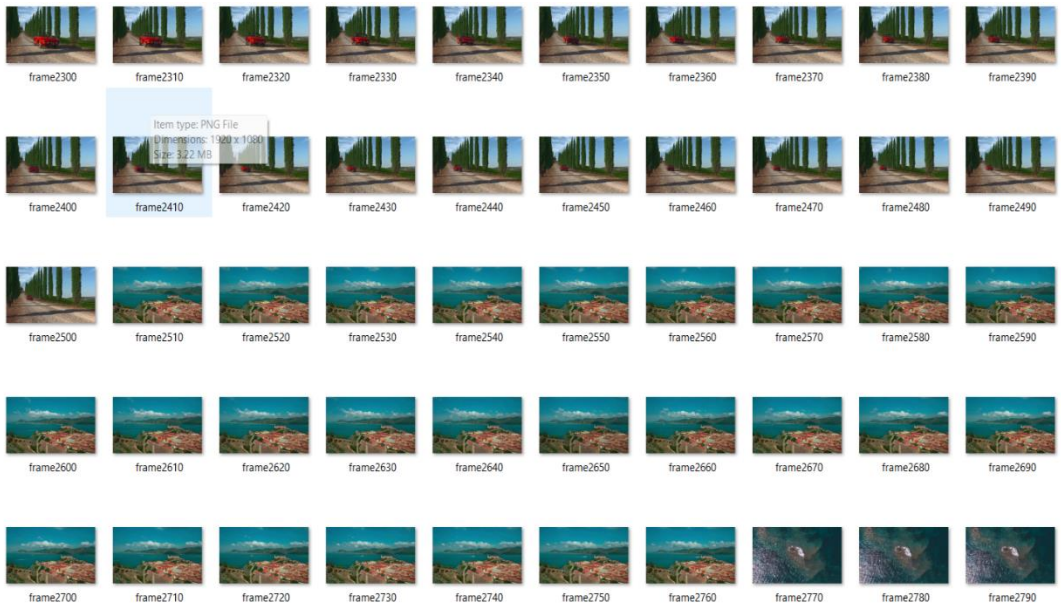


Figure-21: Frame Interval Frame Extraction - 5

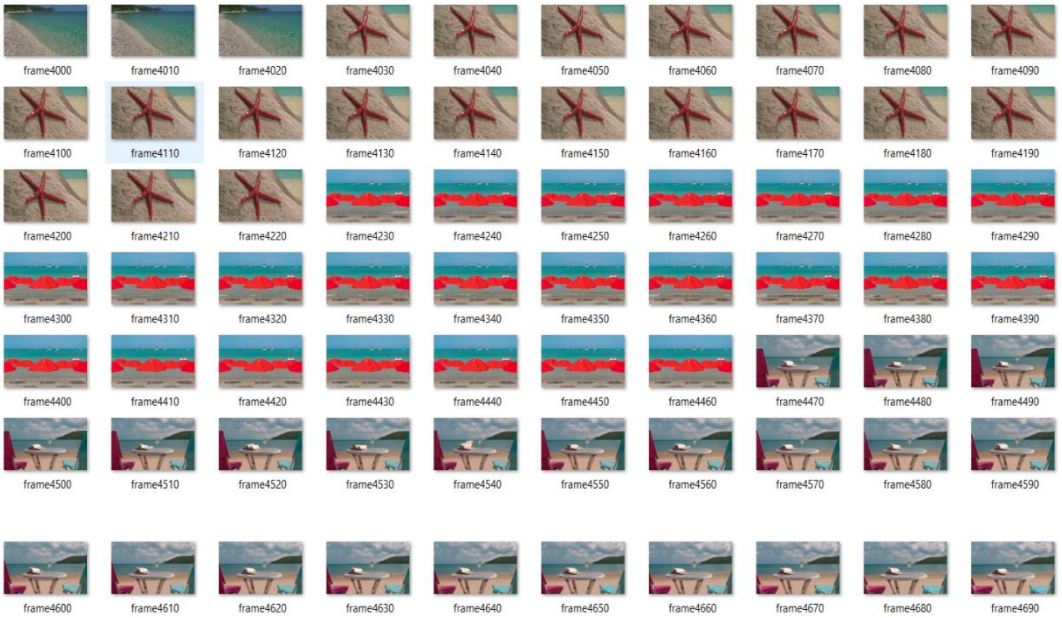


Figure-22: Frame Interval Frame Extraction - 6

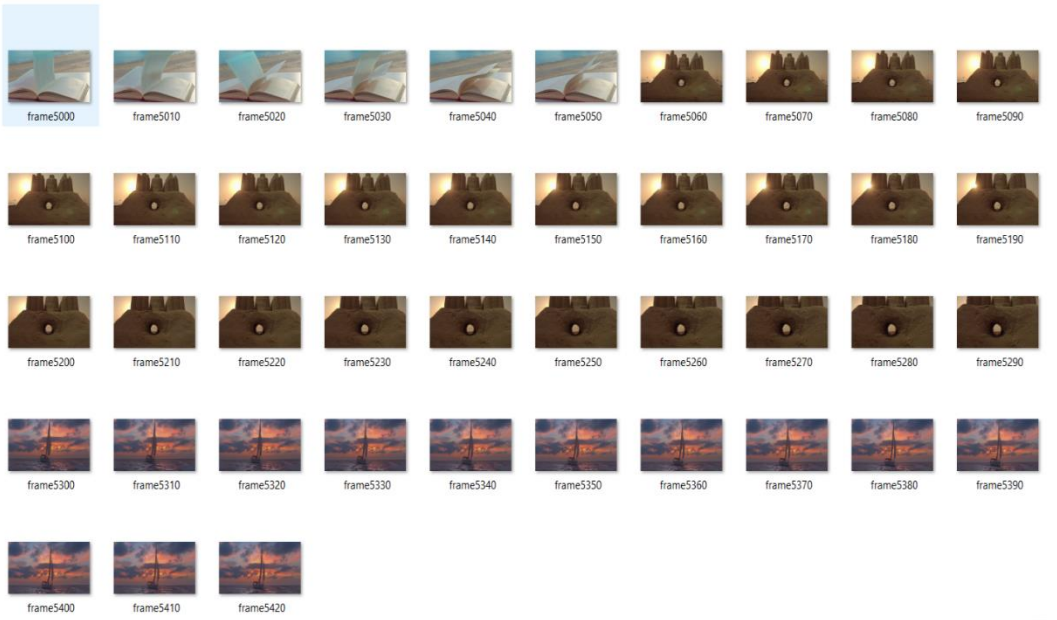


Figure-23: Frame Interval Frame Extraction - 7

The following outputs are of Blob detection method. Blob detection is a technique used in computer vision and image processing to identify regions in an image that differ from the surrounding background in terms of their color, texture, or shape. A blob is defined as a region of an image that has a similar property throughout its area.

```
WELCOME TO VIDDEO FILE CARVING TOOL!!

*****************************************

Choose one among the following:
 1. Classify atoms of a video
 2. Extract Key Frames in a video
 3. Partial File Operations
 4. Other Features
2
Enter name of the video:

Sample-1-HR.mp4
Please choose the technique from below to exract frames:
 1. Frame-Interval Method (FIM)
 2. Blob Detection using a set of Detection Parameters (BDM)
 3. Absolute Mean Difference Method (AMD)
 4. Root Mean Square Method (RMS)
 5. Background Frame Subtraction Method (BFSM)
 6. Optical Flow Frame Detection Method (OFFDM) -> Horn-Schunck Algorithm
 7. Custom Frame Detection Method (CFDM)
2
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\BD_Frames\frame_0000.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\BD_Frames\frame_0001.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\BD_Frames\frame_0002.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\BD_Frames\frame_0003.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\BD_Frames\frame_0004.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\BD_Frames\frame_0005.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\BD_Frames\frame_0006.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\BD_Frames\frame_0007.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\BD_Frames\frame_0008.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\BD_Frames\frame_0009.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\BD_Frames\frame_0010.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\BD_Frames\frame_0011.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\BD_Frames\frame_0012.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\BD_Frames\frame_0013.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\BD_Frames\frame_0014.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\BD_Frames\frame_0015.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\BD_Frames\frame_0016.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\BD_Frames\frame_0017.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\BD_Frames\frame_0018.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\BD_Frames\frame_0019.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\BD_Frames\frame_0020.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\BD_Frames\frame_0021.png
```

Figure-24: Frame Extraction - Blob Detection

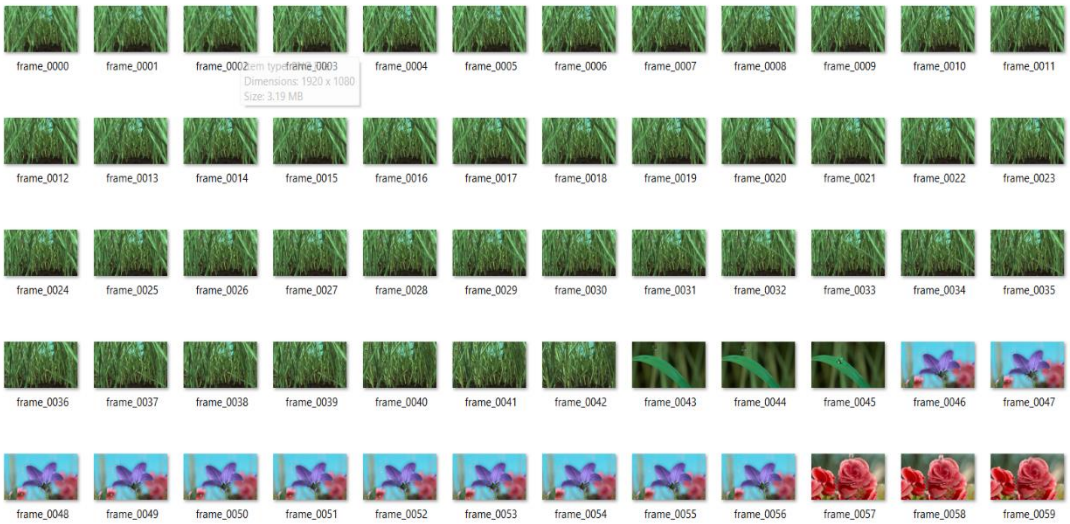


Figure-25: Blob Detection Frame Extraction - 1

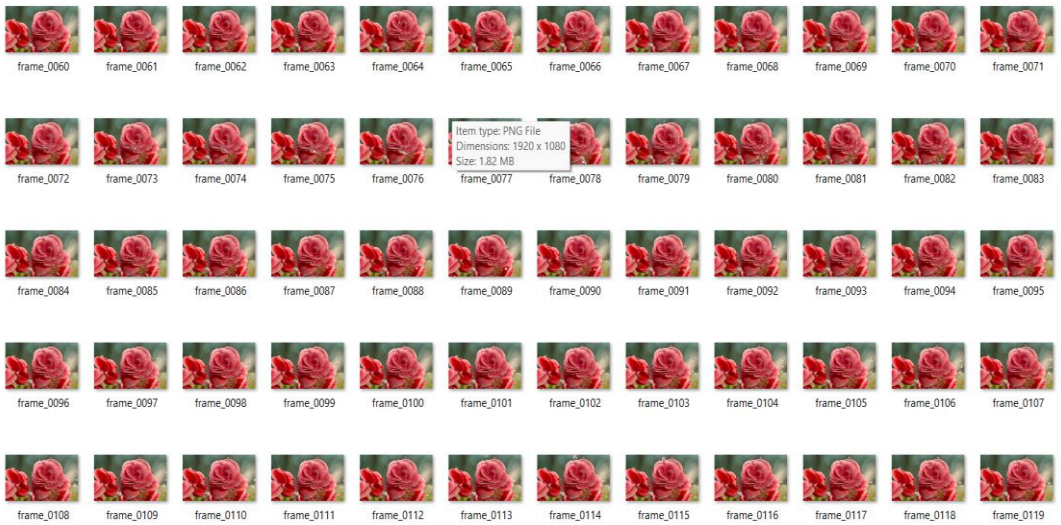


Figure-26: Blob Detection Frame Extraction - 2

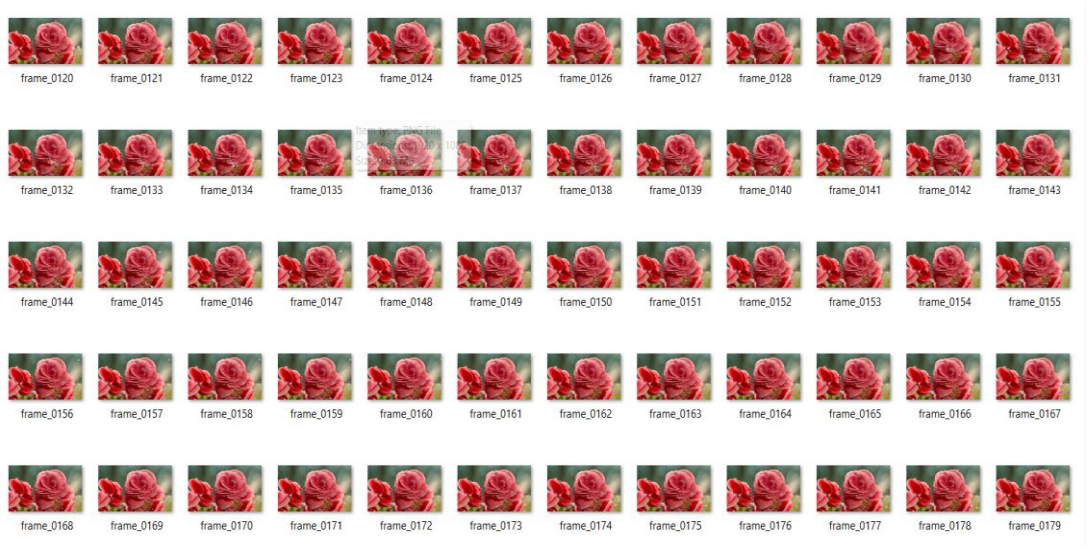


Figure-27: Blob Detection Frame Extraction - 3

Figure-28: Blob Detection Frame Extraction - 4

The following outputs are of AMD method. The AMD method works by computing the absolute difference between each pair of adjacent frames in a video sequence and then taking the maximum difference value for each pixel position across all frame pairs. The resulting values are then averaged to obtain a single value that represents the overall "activity level" of the video sequence.

```
PS D:\File-Carving> python .\Pilot.py

*******************************************

WELCOME TO VIDDEO FILE CARVING TOOL!!

*******************************************

Choose one among the following:
 1. Classify atoms of a video
 2. Extract Key Frames in a video
 3. Partial File Operations
 4. Other Features
2
Enter name of the video:

Sample-1-HR.mp4
Please choose the technique from below to exract frames:
 1. Frame-Interval Method (FIM)
 2. Blob Detection using a set of Detection Parameters (BDM)
 3. Absolute Mean Difference Method (AMD)
 4. Root Mean Square Method (RMS)
 5. Background Frame Subtraction Method (BFSM)
 6. Optical Flow Frame Detection Method (OFFDM) -> Horn-Schunck Algorithm
 7. Custom Frame Detection Method (CFDM)
3
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\AMD_Frames\frame_0000.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\AMD_Frames\frame_0001.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\AMD_Frames\frame_0002.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\AMD_Frames\frame_0003.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\AMD_Frames\frame_0004.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\AMD_Frames\frame_0005.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\AMD_Frames\frame_0006.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\AMD_Frames\frame_0007.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\AMD_Frames\frame_0008.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\AMD_Frames\frame_0009.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\AMD_Frames\frame_0010.png
```

Figure – 29: Frame Extraction – Absolute Mean Difference

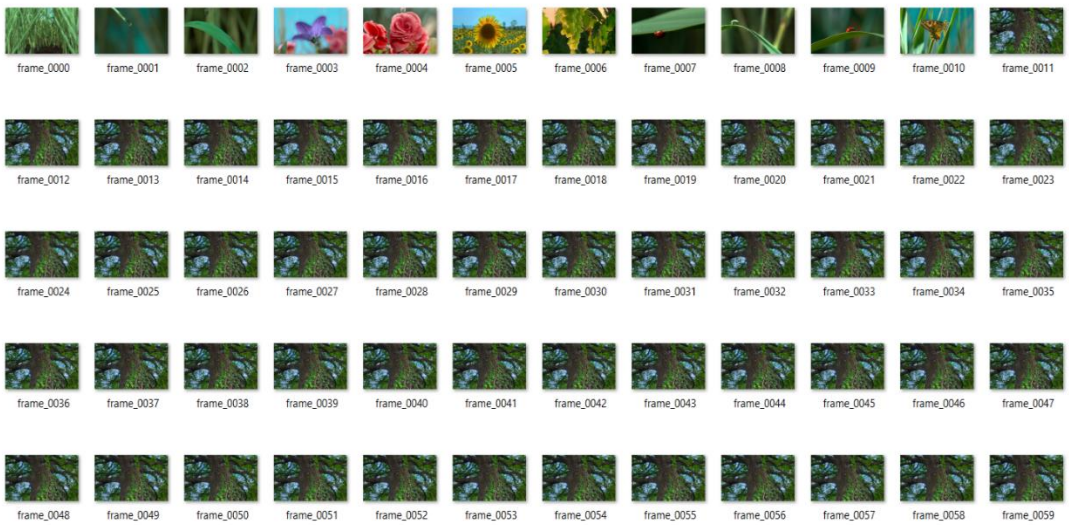


Figure-30: Absolute Mean Difference - 1

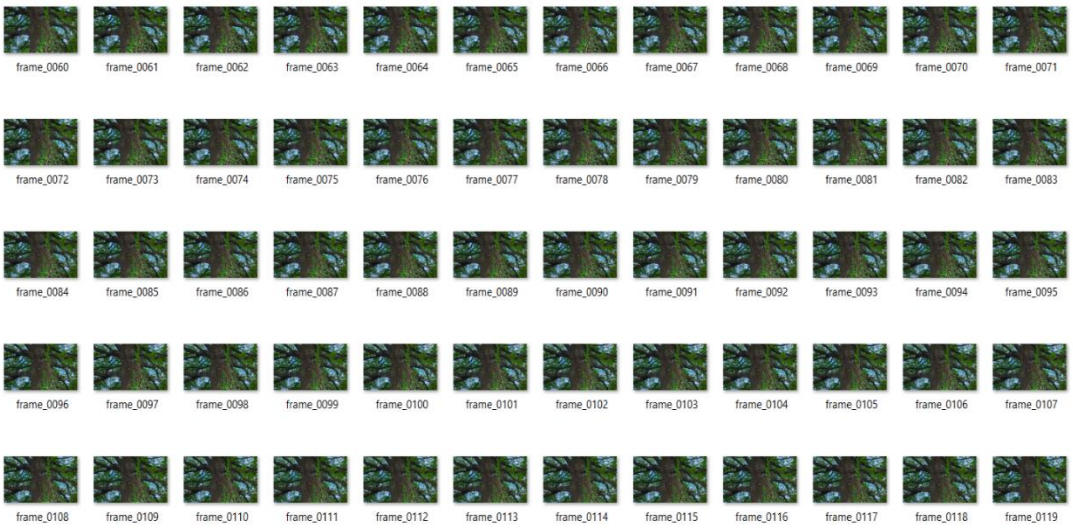


Figure-31: Absolute Mean Difference – 2

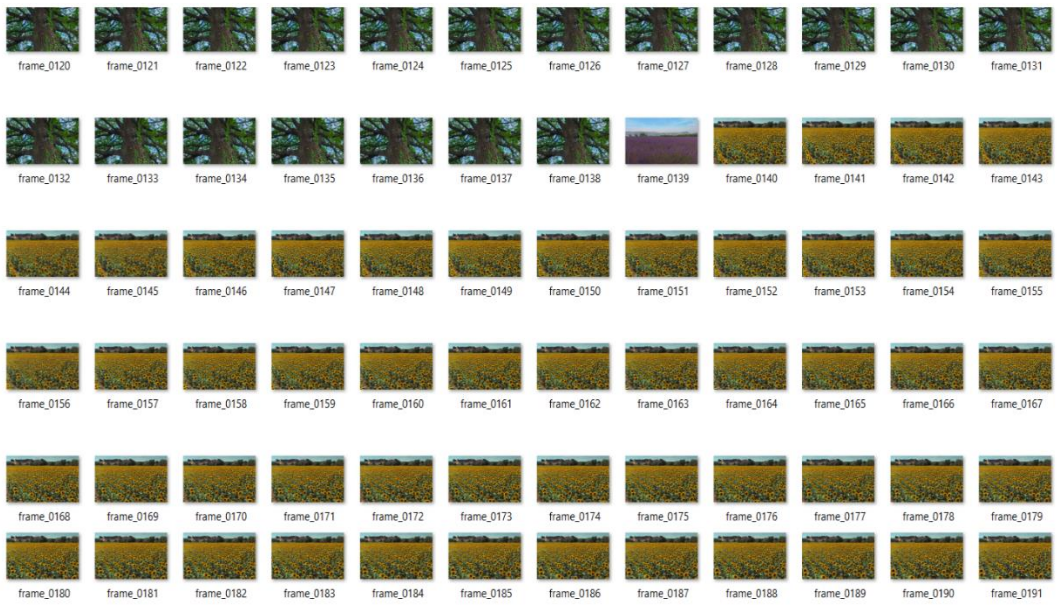


Figure-32: Absolute Mean Difference - 3

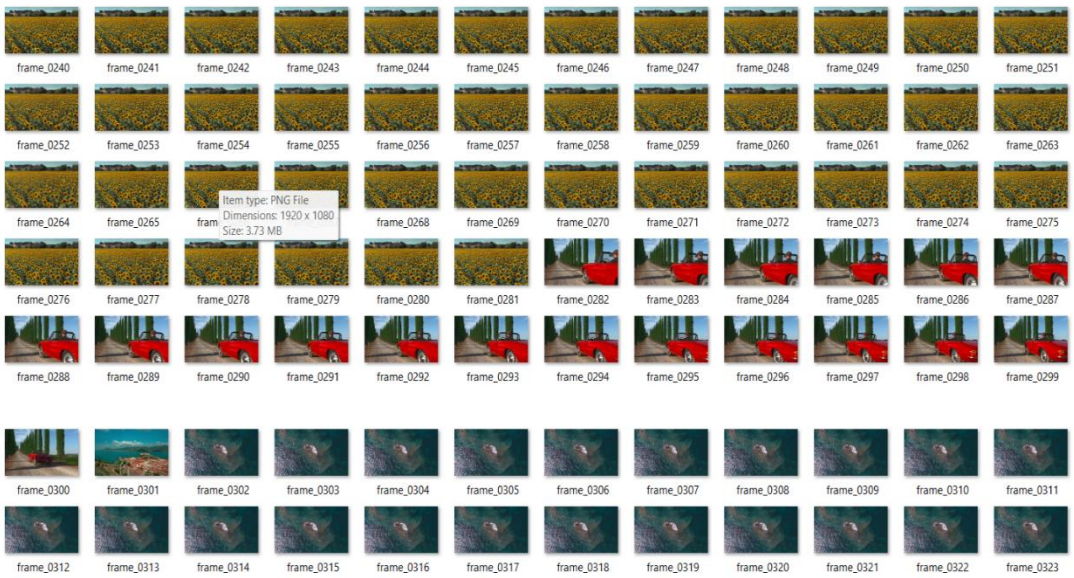


Figure-33: Absolute Mean Difference - 4

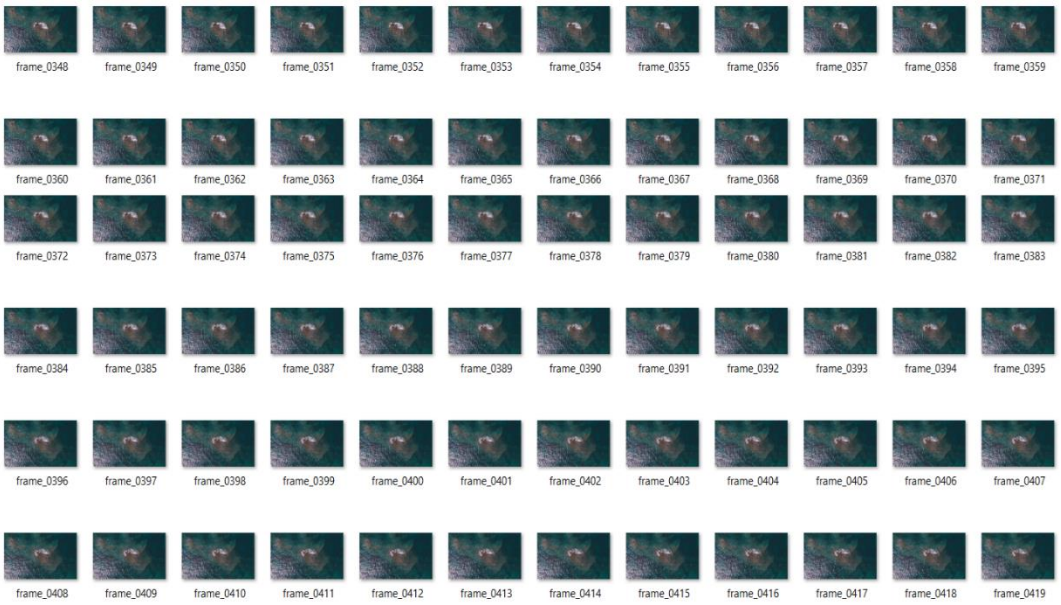


Figure-34: Absolute Mean Difference - 5

The following outputs are of RMS method. The RMS method works by first converting each frame of the video sequence into a grayscale image. Then, the RMS value of each pixel position across all frames is computed by taking the square root of the mean of the squared pixel values. The resulting RMS values are then averaged to obtain a single value that represents the overall "activity level" of the video sequence.

```
PS D:\File-Carving> python .\Pilot.py

*******************************************

WELCOME TO VIDDEO FILE CARVING TOOL!!

*******************************************

Choose one among the following:
 1. Classify atoms of a video
 2. Extract Key Frames in a video
 3. Partial File Operations
 4. Other Features
2
Enter name of the video:

Sample-1-HR.mp4
Please choose the technique from below to exract frames:
 1. Frame-Interval Method (FIM)
 2. Blob Detection using a set of Detection Parameters (BDM)
 3. Absolute Mean Difference Method (AMD)
 4. Root Mean Square Method (RMS)
 5. Background Frame Subtraction Method (BFSM)
 6. Optical Flow Frame Detection Method (OFFDM) -> Horn-Schunck Algorithm
 7. Custom Frame Detection Method (CFDM)
4
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\RMS_Frames\frame_0000.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\RMS_Frames\frame_0001.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\RMS_Frames\frame_0002.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\RMS_Frames\frame_0003.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\RMS_Frames\frame_0004.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\RMS_Frames\frame_0005.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\RMS_Frames\frame_0006.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\RMS_Frames\frame_0007.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\RMS_Frames\frame_0008.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\RMS_Frames\frame_0009.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\RMS_Frames\frame_0010.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\RMS_Frames\frame_0011.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\RMS_Frames\frame_0012.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\RMS_Frames\frame_0013.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\RMS_Frames\frame_0014.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\RMS_Frames\frame_0015.png
```

Figure-35: Frame Extraction - Root Mean Square

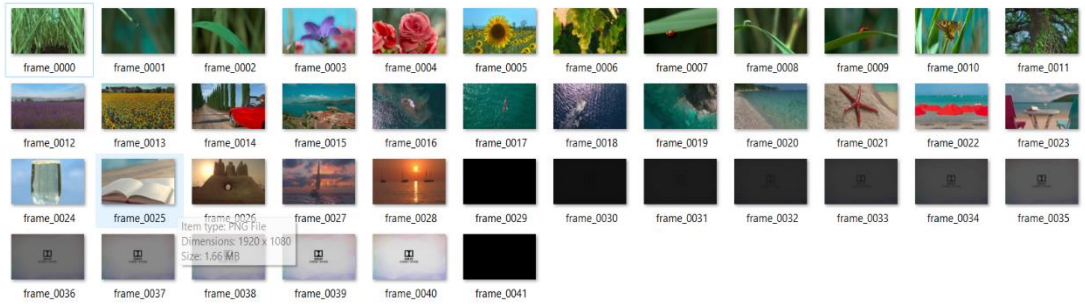


Figure-36: RMS OP

The following outputs are of BGSubtraction method. In the context of file carving, it's possible that "BGSubtraction" refers to a similar technique used to separate foreground data (the file) from the background noise.In this case, the technique would involve analyzing the disk space and looking for patterns or structures that are indicative of file data, such as file headers or magic numbers.

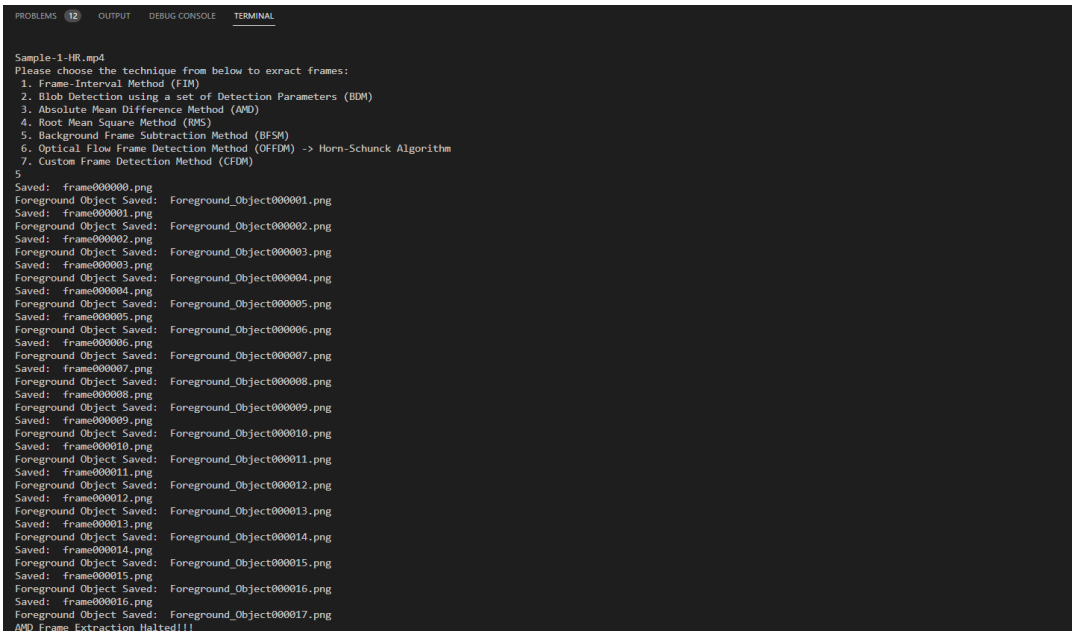


Figure-37: Frame Detection - Background Frame Subtraction

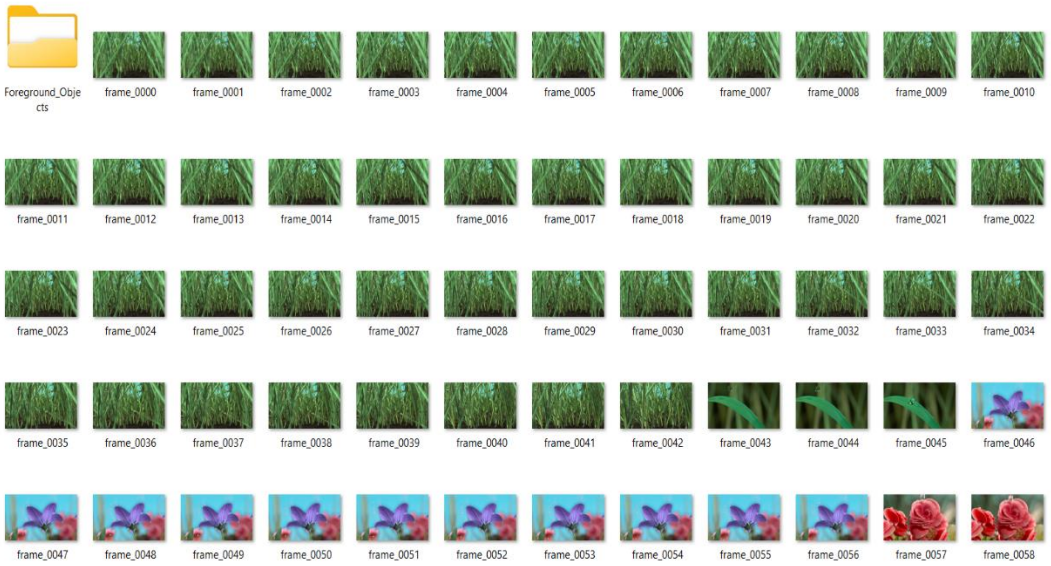


Figure-38: Background Frame Subtraction - 1

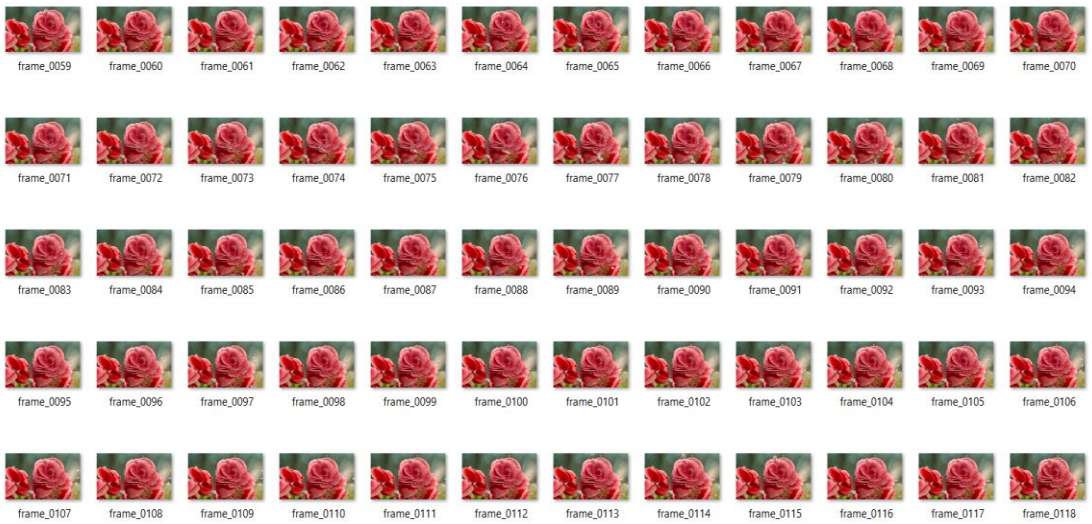


Figure-39: Background Frame Subtraction - 2

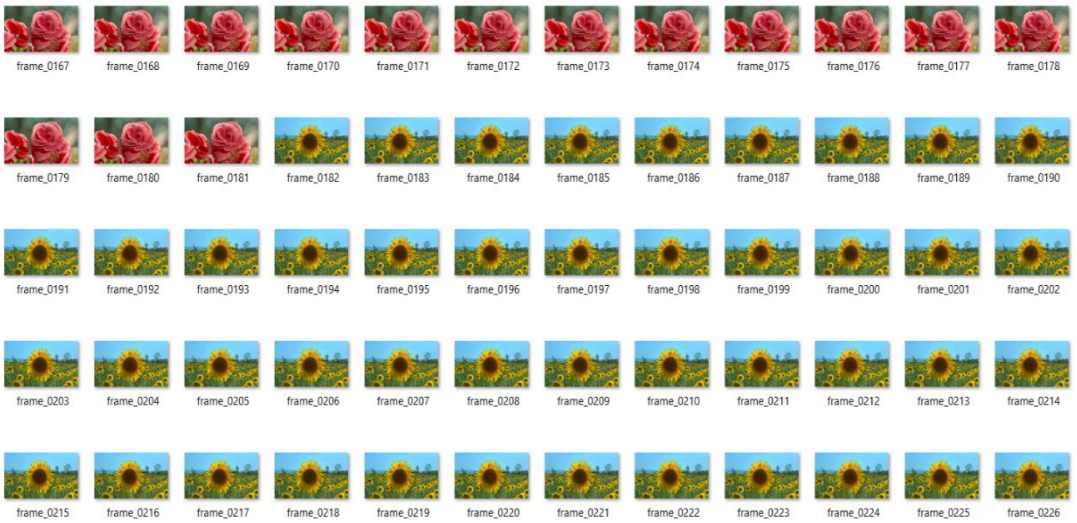


Figure-39: Background Frame Subtraction - 3

Figure-40: Background Frame Subtraction - 4

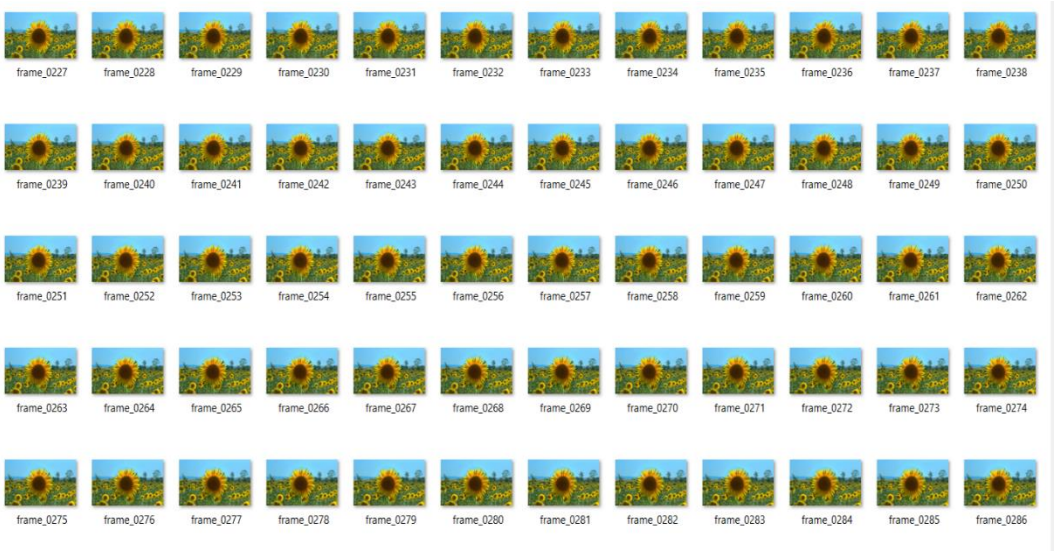
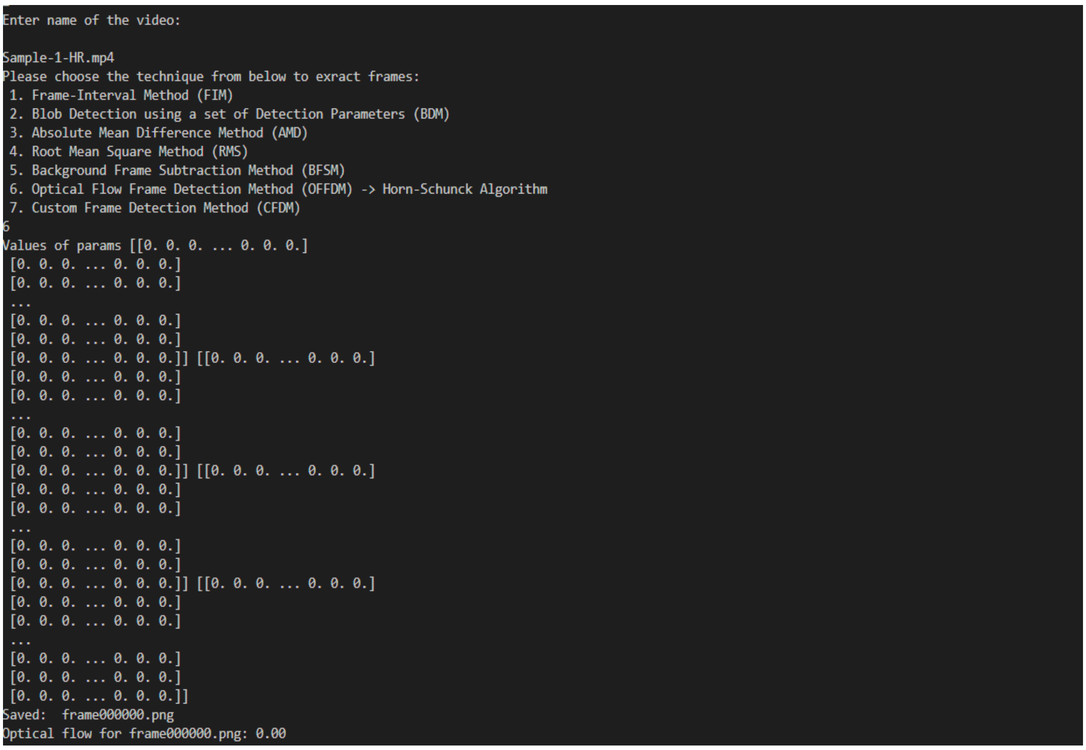


Figure-41: Background Frame Subtraction – 5

Figure-42: Background Frame Subtraction –6

The following outputs are of Optical Flow Detection method. Optical flow is a technique used in computer vision and image processing to estimate the motion of objects in a video sequence. It works by tracking the movement of pixels across frames of the video and analyzing the changes in their position.



```
Enter name of the video:

Sample-1-HR.mp4
Please choose the technique from below to exract frames:
 1. Frame-Interval Method (FIM)
 2. Blob Detection using a set of Detection Parameters (BDM)
 3. Absolute Mean Difference Method (AMD)
 4. Root Mean Square Method (RMS)
 5. Background Frame Subtraction Method (BFSM)
 6. Optical Flow Frame Detection Method (OFFDM) -> Horn-Schunck Algorithm
 7. Custom Frame Detection Method (CFDM)
6
Values of params [[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]] [[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]] [[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]] [[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
Saved:  frame000000.png
Optical flow for frame000000.png: 0.00
```

Figure-43: Optical Flow Detection Interface

Figure-44: Optical Flow Detection Frame Extraction

The following outputs are of Custom Frame Detection method. It typically refers to a technique used in video processing for detecting specific frames or events within a video based on user-defined criteria. This method can be useful when a standard frame extraction method, such as keyframe extraction, does not capture the frames of interest.

```
Enter name of the video:

Sample-1-HR.mp4
Please choose the technique from below to exract frames:
 1. Frame-Interval Method (FIM)
 2. Blob Detection using a set of Detection Parameters (BDM)
 3. Absolute Mean Difference Method (AMD)
 4. Root Mean Square Method (RMS)
 5. Background Frame Subtraction Method (BFSM)
 6. Optical Flow Frame Detection Method (OFFDM) -> Horn-Schunck Algorithm
 7. Custom Frame Detection Method (CFDM)
7
Enter the number of Frames:
170
FPS:  59.94005994005994
Frame_count:  6070
Duration 101.26783333333333
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\Custom_Frames\frame0000.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\Custom_Frames\frame0035.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\Custom_Frames\frame0070.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\Custom_Frames\frame0105.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\Custom_Frames\frame0140.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\Custom_Frames\frame0175.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\Custom_Frames\frame0210.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\Custom_Frames\frame0245.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\Custom_Frames\frame0280.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\Custom_Frames\frame0315.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\Custom_Frames\frame0350.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\Custom_Frames\frame0385.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\Custom_Frames\frame0420.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\Custom_Frames\frame0455.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\Custom_Frames\frame0490.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\Custom_Frames\frame0525.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\Custom_Frames\frame0560.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\Custom_Frames\frame0595.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\Custom_Frames\frame0630.png
Saved key frame: D:\File-Carving\Stage-2\ExtractedFrames\Custom_Frames\frame0665.png
```

Figure-45: Custom Frame Detection – Interface

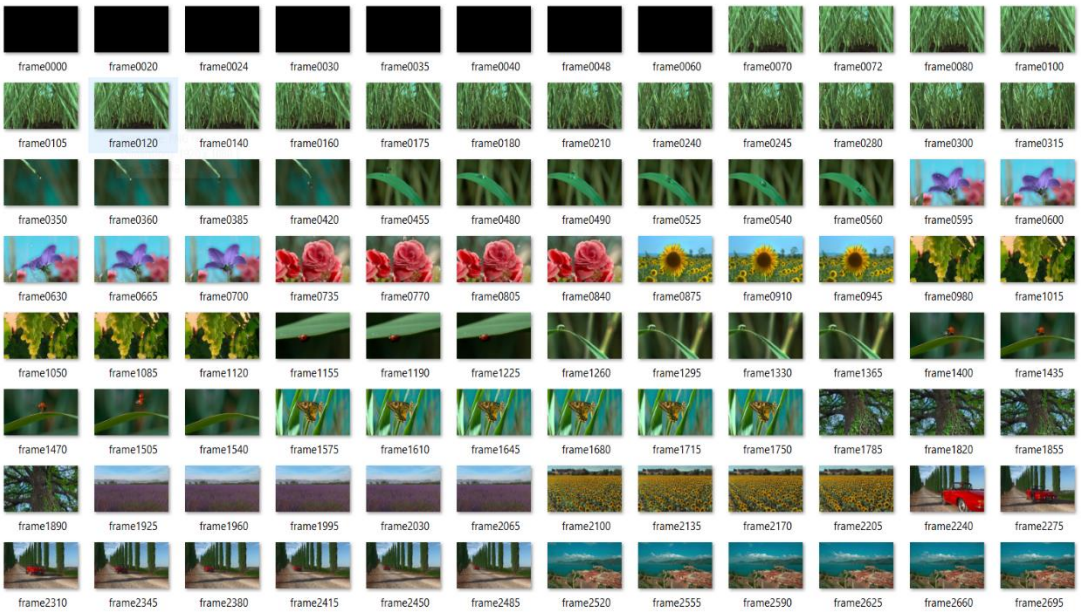


Figure-46: Custom Frame Extraction – 1

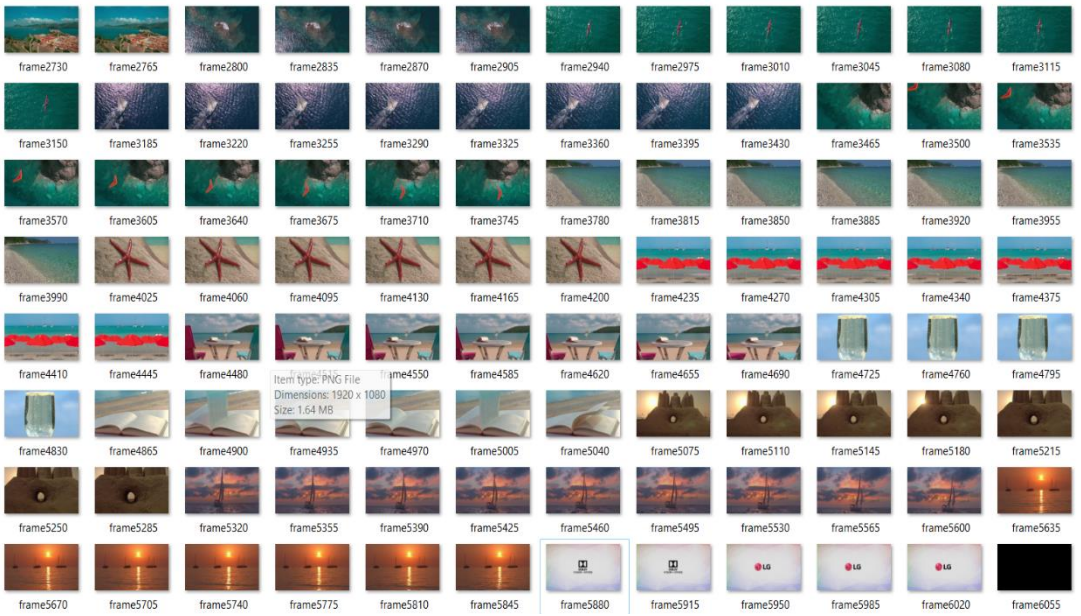


Figure-47: Custom Frame Extraction – 2

The following outputs are related to Attribute Redundancy Feature. Given a corrupted fragment, using redundant attribute values, one can recover the corrupted fragment.(Considering most probable value of corrupted & redundant attribute) .Next, Fragment level carving and structure level carving is done.

```
Choose one among the following:
 1. Classify atoms of a video
 2. Extract Key Frames in a video
 3. Partial File Operations
 4. Other Features
3
Choose one among the following options:
 1.Attribute Level Carving
 2. Fragment Level Carving
 3. Structure Level Carving

1
Enter name of the partial video file:
CorruptedPartialFile1.txt

Next atom size is  32
Next atom is  ftyp

FTYP((File type)) Atom Details are:-----------------------------------
'Size' : 32
'Type' : ftyp
'Major_Brand' : mp42
'Minor_Version' : 00000000
'Compatible_Brands' : ['mp42', 'mp41', 'isom', 'avc1']

Next atom size is  13178
Next atom is  moov

MOOV(Movie type) Atom Details are:-----------------------------------
'Size' : 13178
'Type' : moov

Next atom size is  108
Next atom is  mvhd
```

Figure-48: Attribute Redundancy -1

```
MVHD(Movie Header type) Atom Details are:-----------------------------------
'Size' : 108
'Type' : mvhd
'Version' : 00
'Flags' : 000000
'Creation_Time' : 2020-11-21 13:09:30
'Modification_Time' : 2020-11-21 13:09:30
'Time_Scale' : 00000258
'Duration' : 00002EF3
'Preferred_Rate' : 00010000
'Preferred_Volume' : 0100
'Reserved' : 00000000000000000000
'Matrix' : 00010000000000000000000000000000000000010000000000000000000000000000000040000000
'Preview_Time' : 00000000
'Preview_Duration' : 00000000
'Poster_Time' : 00000000
'Selection_Time' : 00000000
'Selection_Duration' : 00000000
'Current_Time' : 00000000
'Next_Track_ID' : 00000003

Next atom size is  42
Next atom is  iods

IODS(iods) Atom Details are:-----------------------------------
'Size' : 42
'Type' : iods
'Version' : 00
'Flags' : 000000
'OtherData' : 1080808019004FFFFF297FFF0E8080800400000010E8080800400000002

Next atom size is  8603
Next atom is  trak

TRAK(Track type) Atom Details are:-----------------------------------
'Size' : 8603
'Type' : trak

Next atom size is  92
Next atom is  tkhd
```

Figure-49: Attribute Redundancy -2

```
TKHD(Track Header type ) Atom Details are:-----------------------------------
'Size' : 92
'Type' : tkhd
'Version' : 00
'Flags' : 000007
'Creation_Time' : 2020-11-21 13:09:30
'Modification_Time' : 2020-11-21 13:09:30
'Track_ID' : 00000001
'Reserved' : 00000000
'Duration' : 00002EEC
'Reserved_2' : 0000000000000000
'Layer' : 0000
'Alternative_Group' : 0000
'Volume' : 0000
'Reserved_3' : 0000
'Matrix_Structure' : 00010000000000000000000000000000000100000000000000000000000000040000000
'Track_Width' : 0F000000
'Track_Height' : 08700000

Next atom size is  36
Next atom is  edts


 Edit Samples (edts) details:-----------------------------------
'Size' : 36
'Type' : edts

Next atom size is  28
Next atom is  elst

ELST(elst) Atom Details are:-----------------------------------
'Size' : 28
'Type' : elst
'Version' : 00
'Flags' : 000000
'EntryCount' : 00000001
'EditDuration' : 00002EEC
'EditMediaTime' : 000007D2
'PlaybackSpeed' : 00010000
```

Figure-50: Attribute Redundancy -3

```
'Type' : elst
'Version' : 00
'Flags' : 000000
'EntryCount' : 00000001
'EditDuration' : 00002EEC
'EditMediaTime' : 000007D2
'PlaybackSpeed' : 00010000

Next atom size is  8467
Next atom is  mdia

MDIA(Media type) Atom Details are:-----------------------------------
'Size' : 8467
'Type' : mdia

Next atom size is  32
Next atom is  mdhd

MDHD(Media Header) Atom Details are:-----------------------------------
'Size' : 32
'Type' : mdhd
'Version' : 00
'Flags' : 000000
'Creation_Time' : 2020-11-21 13:09:30
'Modification_Time' : 2020-11-21 13:09:30
'Time_Scale' : 00007530
'Duration' : 00092A18
'Language' : 55C4
'Predefined' : 0000
File size:  732
Final hex pointer value:  366
```

Figure-51: Attribute Redundancy - 4

```
Atoms Encountered:
['ftyp', 'moov', 'mvhd', 'iods', 'trak', 'tkhd', 'edts', 'elst', 'mdia', 'mdhd']

Missing Atoms:
{}

Atom Redundancy:
Creation Time:
{'mvhd': '2020-11-21 13:09:30', 'tkhd': '2020-11-21 13:09:30', 'mdhd': '2020-11-21 13:09:30'}
Modification Time:
{'mvhd': '2020-11-21 13:09:30', 'tkhd': '2020-11-21 13:09:30', 'mdhd': '2020-11-21 13:09:30'}
Matrix:
{'mvhd': '00010000000000000000000000000000000100000000000000000000000000040000000', 'tkhd': '00010000000000000000000000000000000100000000000000000000000000040000000'}
{'mvhd': '2020-11-21 13:09:30', 'tkhd': '2020-11-21 13:09:30', 'mdhd': '2020-11-21 13:09:30'} {'mvhd': '2020-11-21 13:09:30', 'tkhd': '2020-11-21 13:09:30', 'mdhd': '2020-11-21 13:09:30
'} {'mvhd': '00010000000000000000000000000000000100000000000000000000000000040000000', 'tkhd': '00010000000000000000000000000000000100000000000000000000000000040000000'}

Final Values of Redundant Attributes based on Frequency:

Final Creation Time:----------------------

2020-11-21 13:09:30

Final Modification Time:----------------------

2020-11-21 13:09:30

Final Matrix Value:----------------------

00010000000000000000000000000000000100000000000000000000000000040000000
```

Figure-52: Attribute Redundancy - 5

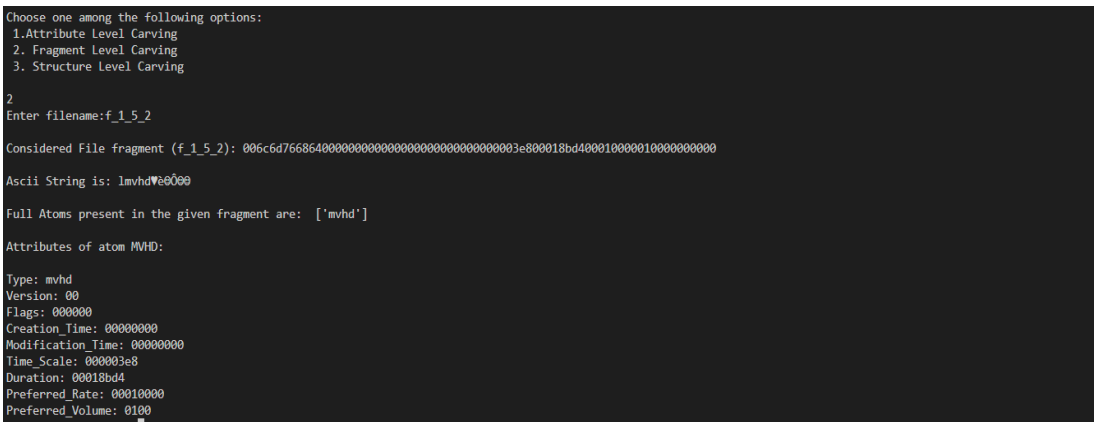


Figure-53: Fragment Level Carving

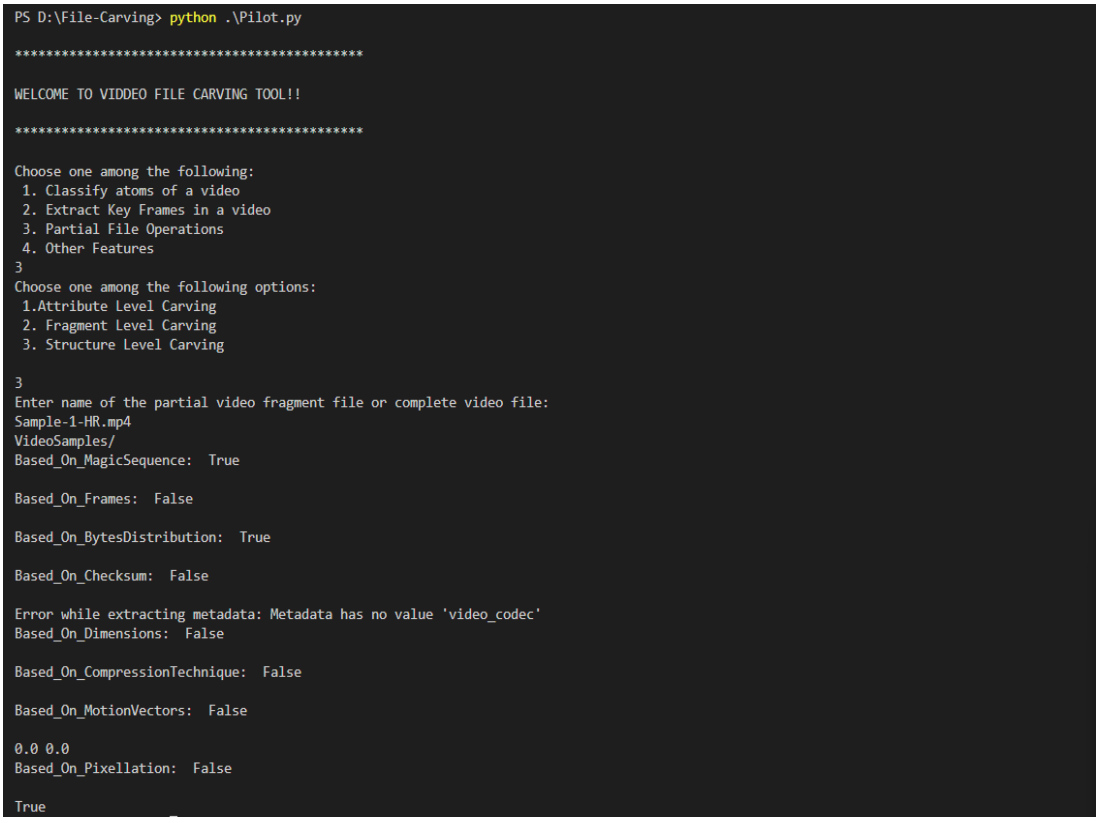


Figure-54: Structure Level Carving

5. CONCLUSION

The success of a file carving project depends on several factors such as the tools used, the expertise of the user, and the quality of the data being carved. It is crucial to have a good understanding of the file system and the file types that are being carved. In conclusion, file carving can be a powerful tool for data recovery and forensic analysis. With the right tools and expertise, it is possible to recover deleted or lost data from various storage media. However, file carving can be a time-consuming process and requires a lot of patience and attention to detail. In this project, given a video file/partial video file fragment, we are able to perform atom classification and frame level classification (by identification of key frames). We are also able to recover the corrupted fragment (corruption at redundant attributes of atoms) and regenerate the corruption-free fragment. This lays the foundation for advanced stages like complete file recovery even after many fragments go missing or corrupted.

6. FUTURE SCOPE

Video file carving is a specialized area of file carving that deals with the recovery of video files from storage media. Some potential areas of future scope for video file carving include:

Improved support for new video codecs: As new video codecs are developed and become more popular, video file carving tools will need to keep up by adding support for these new codecs.

Automated video file reconstruction: Currently, video file carving is a manual process that involves piecing together fragments of video data to reconstruct a complete video file. In the future, it may be possible to automate this process using machine learning algorithms or other techniques.

Real-time video file carving: Real-time video file carving could be useful for applications such as video surveillance, where it is necessary to extract video data in real-time from a storage device.

Cloud-based video file carving: As more data moves to the cloud, video file carving tools will need to evolve to be able to recover video data from cloud storage services.

Integration with video analysis tools: Video file carving tools could be integrated with video analysis tools to provide a more comprehensive solution for forensic analysis and investigations.

These are just a few potential areas of future scope for video file carving. As technology continues to evolve, new challenges and opportunities will arise, and video file carving tools will need to continue to evolve to meet these challenges.

7. REFERENCES

1. Gi-Hyun Na, Kyu-Sun Shim, Ki-Woong Moon, Seong G. Kong, Senior Member, IEEE, Eun-Soo Kim, and Joong Lee,
   Frame-Based Recovery of Corrupted Video Files Using Video Codec Specifications

2. Rainer Poisel, Simon Tjoa, and Paul Tavolato, Insitute of IT Security Research, St. Poelten University of Applied Sciences,
   Advanced File Carving Approaches for Multimedia Files

3. Quick Type File Format (QTFF) -2001 Apple Developers Official Documentation

4. Enes Altinisik and Hüsrev Taha Sencar , Senior Member,
   Automatic Generation of H.264 Parameter Sets to Recover Video File Fragments ,IEEE

5. EnGenious Frank Wang, Software Engineer,
   Introduction to H.264 Video Compression Standard

6. https://developer.apple.com/library/archive/documentation/QuickTime/QTFF/QTFFChap1/qtff1.html#//apple_ref/doc/uid/TP40000939-CH203-BBCGDDDF

7. https://www.file-recovery.com/mp4-signature-format.htm

8. https://www.rfc-editor.org/rfc/rfc6184#page-13

9. https://www.ftyps.com/3gpp.html

10. https://asecuritysite.com/principles/numbers01

11. https://dev.to/alfg/a-quick-dive-into-mp4-57fo

12. https://repairit.wondershare.com/video-repair/m4v-video-repair.html

13. https://github.com/ponchio/untrunc

14. https://xhelmboyx.tripod.com/formats/mp4-layout.txt

15. https://www.vcodex.com/an-overview-of-h264-advanced-video-coding/

16. https://www.e-consystems.com/blog/camera/technology/what-is-h-264-video-encoding-how-does-h-264-codec-work/

17. https://doc-kurento.readthedocs.io/en/latest/knowledge/h264.html

18. https://yasoob.me/posts/understanding-and-writing-jpeg-decoder-in-python/

19. https://sh-tsang.medium.com/reading-h-lstm-hierarchical-lstm-for-fast-h-264-to-hevc-transcoding-fast-codec-prediction-84f300ef70cc

20. https://www.vcodex.com/an-overview-of-h264-advanced-video-coding/

21. https://www.videomaker.com/article/c10/15362-video-formats-explained/

22. https://adydychk.medium.com/what-is-a-video-codec-and-how-does-it-work-6f90bd037fa0

23. https://sanjeev-pandey.medium.com/understanding-the-mpeg-4-moov-atom-pseudo-streaming-in-mp4-93935e1b9e9a

24. https://www.diffchecker.com/EUcjcliq/

25. https://developer.apple.com/library/archive/documentation/QuickTime/QTFF/QTFFAppenG/QTFFAppenG.html