International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 05 | May - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

AUTOMATED VIDEO CONTENT SUMMARIZATION

Sinchana S, Sudarshan S Kakalwar, Vaibhav E S, Veekshith V
Under the guidance of Prof.Belji T Associate Professor
Department of Computer Science and Engineering
Bachelor of Engineering
K. S. School of Engineering and Management, Bengaluru – 560109

ABSTRACT

Automated video content summarization aims to reduce long videos into concise and straightforward summaries. Humans can recognize critical highlights in a video thanks to audio-visual and scenographic cues; however, the majority of methods do not cut it. They lack reliance on multi-modal features, and algorithmic appreciation, or disregard video aesthetics during editing, which compromises the quality of synthesis. In addition, many approaches rely heavily on labeled training data, which is not only tedious to compile, but also devoid of the vast content, and myriad of personal preferences encapsulated within different videos.

To overcome these problems, we propose ADUVS, an unsupervised, aesthetics-based automated video summarization framework. An aesthetics encoder sustains processing of visually important elements to identify the most captivating and salient video segments, including portions that are relevant. We also designed a multi-modal fusion module so different pieces of information can be combined to assess which fragments the summary should contain.

Of note, ADUVS is more economical and simpler to train thanks to its lack of need for annotated data. Through rigorous testing, our approach tested against numerous benchmark techniques and founds ADUVS surpassed them all based on evaluation metrics.

INTRODUCTION

Video summarization enables the condensing of long videos into meaningful clips that convey the main ideas or highlight events without the need to watch the entire footage. It usually entails still summarization in the form of keyframe selection and video skimming which maintains the highlights captured as essential to the video's continuous flow over time. As far as the enhancement of summarization, feature extraction along with clustering, and both old and new deep learning approaches are central to it. These are applicable for ranges of activities starting from content browsing and video retrieval to surveillance as means for efficient managing and comprehension with large volumes of video data. While considering the scope of changes that are bound to arise with technological advancement, artificial intelligence and video technology is poised to revolutionize the changing face of summarization by making is highly intelligent adaptable to context.

The need for effective video summarization is to the constantly increase in video content on the internet, surveillance systems, and personal libraries. Outdated methods like keyframe selection, scene detection, and temporal clustering tend to overlook vital visual details or significant events, which results in uninformative summaries. To offer more accurate summaries, object detection can be integrated.

PROBLEM STATEMENT

As video content expands across social media, security systems, and personal storage, easily gaining access to the important moments has become a hassle. Most conventional summarization techniques lack the ability to make captivating summaries compounded by the fact that they omit crucial highlights. Such difficulties give rise to the need to develop an intelligent method that can recognize fundamental scenes while retrieving the order in context, and also the visual flow within the footage.

In this project we focus on designing a web application to upload videos and retrieve summaries. The user friendly web- based interface outlines a captivating and efficient way to save time when analyzing long videos. Integrating machine learning and deep learning techniques enhances viewing experiences in which important content can be summarized while context is preserved.

GOALS AND OBJECTIVES

3.1 GOALS:

The primary aim of this project is to create an automated video summarization system capable of shortening lengthy videos while retaining essential interactions and events. The system attempts object and activity detection using advanced techniques such as YOLO-based object detection and real-time object tracking. Furthermore, the project intends to improve the summarization process by implementing feature extraction, event analysis, and keyframe selection to create a coherent contextually rich output. This system aids in improving the accessibility of videos, cutting down manual work, and addressing multiple needs like surveillance, content creation, and video browsing.

3.2 OBJECTIVES:

1. Data collection: Gather data and teach the objects for activities like tracking and detection.
2. Object detection: Create software to increase the object detection for recognition and detection in video frames.
3. Object tracking: Trace a detected object through a sequence of frames while applying temporal consistency algorithms.
4. Feature extraction: Study videos of an object’s interactions and movements to extract relevant features and its size, shape, and other details.
5. Keyframe selection and video summary generation: Choose keyframes and make sure they contain the most important objects and events.

REQUIREMENTS

4.1 FRONT END:

4.1.1 Technologies:

- HTML/CSS: To build and design the layout of the page/user interface.
- JavaScript:To provide the required level of interactivity and dynamic activities within the page.

4.1.2 Features:

- Web Interface: A resizable, responsive site that is user-friendly because of the Flask templates and CSS styles.
- User Dashboard: Real-time updates as well as alerts notifications for incidents that have been triggered and detected are shown.
- Content Submission Panel: This allows the users to provide or input data and that can be analyzed as per requirements.

4.2 BACKEND :

4.2.1 Framework:

- Flask: Flask is a framework used with Python that is very simple and is useful in handling all the operations on the server side.

4.2.2 Core Functionalities:

- Video Preprocessing :Use makesense and roboflow for labelling, resize and reshape the object to prepare data for analysis.
- Object Detection:Implement an YOLO model trained on labeled datasets like the COCO dataset from Kaggle.
- Object Tracking : Devise tracking algorithms for the detected objects i.e. SORT and Deep SORT.
- Key frame extraction :Devise an algorithm to select the key frames that are unique from the video or from the live mode of provided queries.
- Summarized output processing : Allow model to selectively fetch unique frames from the video and craft visually compelling compositions for user.

4.3 DATABASE REQUIREMENTS:

4.3.1 Database System:

○ SQL based Databases: User data and the results from the video analysis are stored efficiently on databases like SQLite or MySQL..

4.3.2 Tools:

○ SQLAlchemy:This is regarded as one of the strongest object-relational mappers (ORM).This powerful ORM tool allows the developer to interact with the system without having to write raw SQL commands directly, simply by using python code.

4.3.3 Data Storage:

○ Pre-defined dataset: For training and analysis, the system makes use of the COCO dataset which contains labeled objects and their corresponding metadata.

○ Content Analysis Data: Stored video key frames are classified into common and unique based on their content which aids in more precise and contextually relevant summarization.

○ Summarized output: The system provides users with final video outputs stored in specific folders after the videos have been processed and summarized.

DESIGN

5.1 SYSTEM DESIGN ARCHITECTURE

○ **Backend System**: Responsible for object training, detection, and tracking. It identifies key frames from videos and generates concise video summaries based on detected activity.

○ **Database**: Stores the extracted key frames and their final summarized video output, using data from a pre-trained dataset for reference and comparison.

○ **Frontend System**: Manages the user interface of the web application, displaying outputs and enabling user interaction through a clean and responsive design.

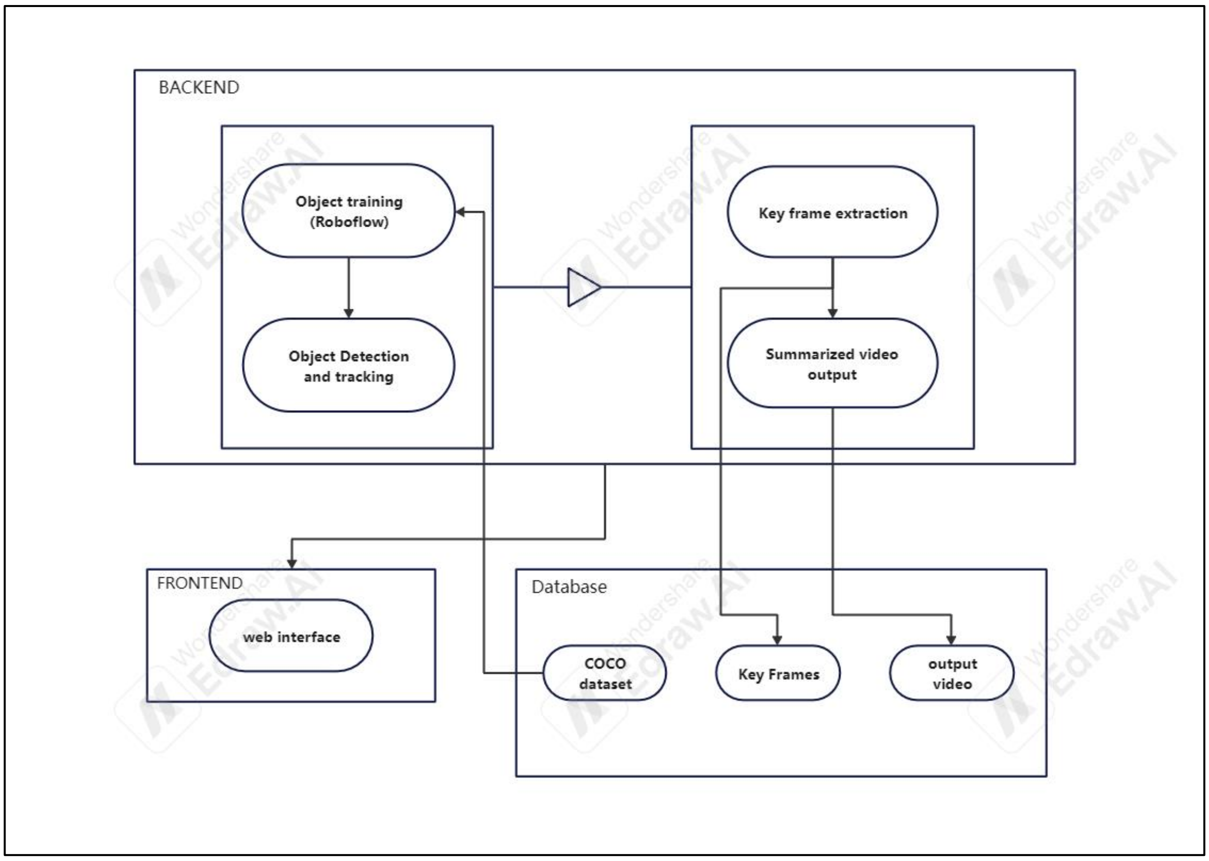

Fig:5.1 System Design Architecture

5.2 USE CASE DIAGRAM :

1. **Actors**:

○ **User**: The individual interacting with the system.

Object Tracking System: A system that manages and processes video data to identify and track objects.
controls: The mechanisms or tools used to operate the Object Tracking System.
Capture Video: The act of recording video content to be analyzed later by the tracking and summarization system
Object Tracking: The system's capability to continuously follow and monitor objects moving through its field of view over time.
tracks objects: Describes the capability of identifying specific items as they move through the camera's view.
Generate Summary: The automated process of producing a brief yet informative overview of a video, highlighting important moments across different timestamps.

2. **Use cases :**

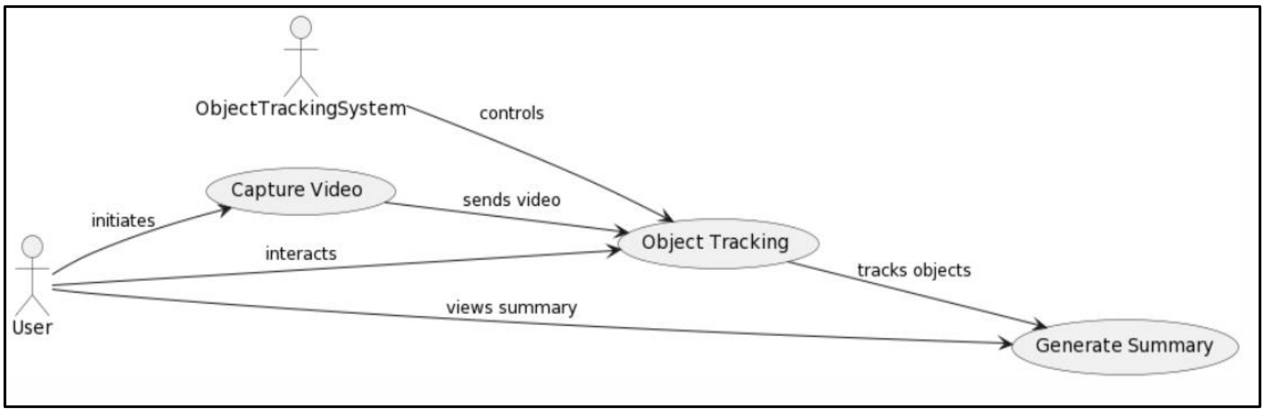

Fig:5.2 Use case diagram

5.3 ACTIVITY FLOW DAIGRAM:

1. **Start**: The system initiates when a video is uploaded, along with a query parameter that defines what the user is interested in tracking or summarizing

2. **Initialization**: The provided query must correspond to an element or class within the pre-trained dataset to ensure accurate processing.

3. **Object tracking**: Uses algorithms like SORT and Deep SORT to determine the motion detection in the frame.

4. **Extract key frames**: If the objected is detected it extracts their regions and generates track ID's for particular frame.

5. **Summarized output**: The unique frames are then compiled into a coherent and visually engaging video summary.

6. **End**: Terminates the process

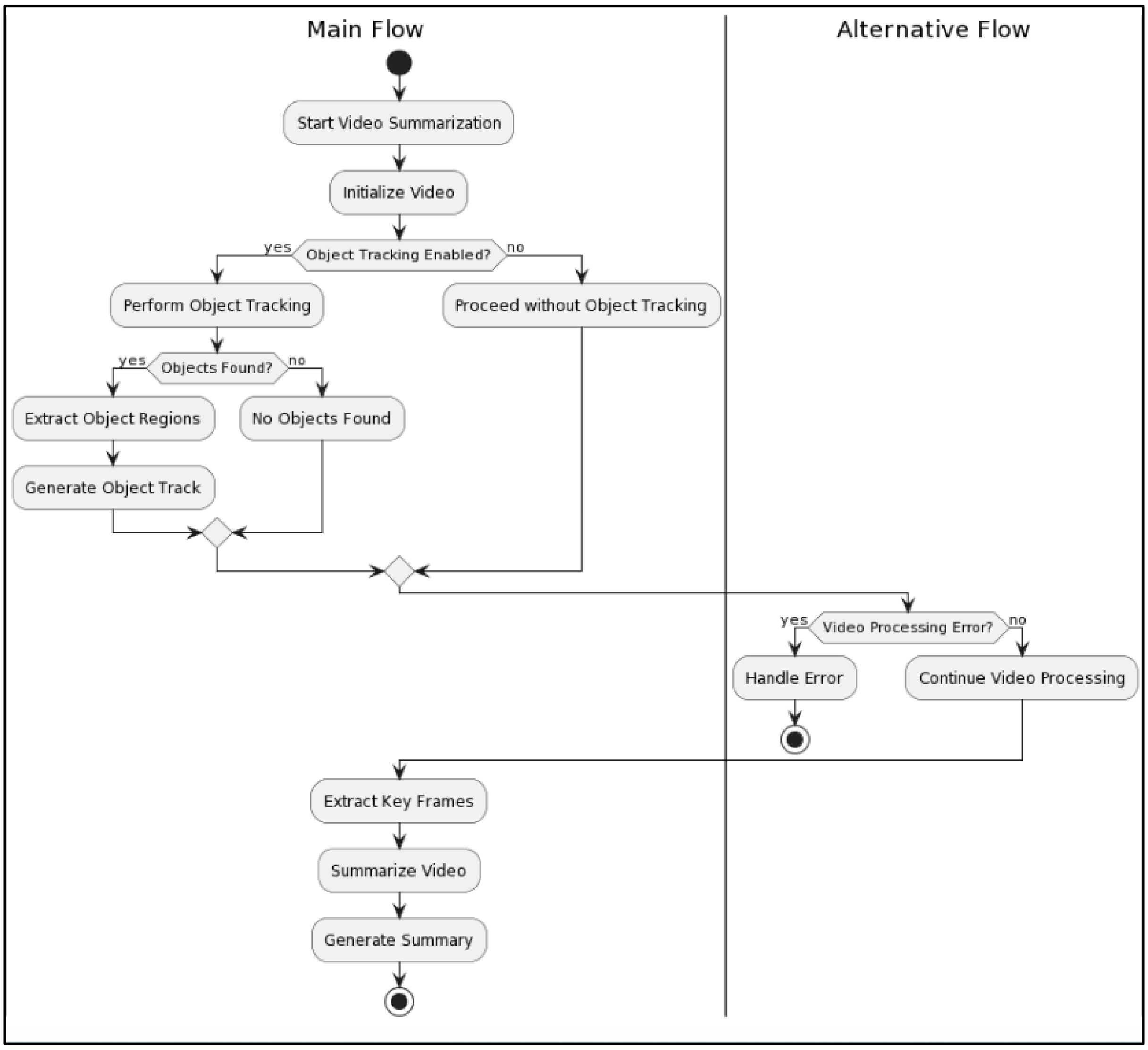

Fig:5.3 Activity flow diagram

IMPLEMENTATION

6.1 Algorithm

1. Preprocessing: Input videos undergo frame extraction alongside possible resizing. To lessen redundancy, optional keyframe extraction might be undertaken at this stage.

2. Object Detection: Frames are analyzed to detect objects using off-the-shelf deep learning models including YOLO and SSD. For each detection, the system produces bounding boxes, object names, and confidence weights.

3. Object Tracking: Upon achieving object detection, tracking algorithms such as Kalman Filters, SORT, or Deep SORT follow the objects across successive frames in the presence of motion and occlusion.

4. Summarization Criteria: The minimum and maximum value metrics are set in the frame to assess importance

and evaluate. These measures may include motion, object interactions, frequency, and time length.

5. Summarization Algorithm: Requirements ensure visual continuity throughout the summarization process, striking behaviors or interactions of objects during the intervals, and smooth transitions around decisive frames towards object parts.

6. Keyframe Selection: A representative set of keyframes that characterize the video content and storyline is selected from all keyframes to represent processed video.

7. Summary Generation: A short logical summary of the video is constructed directly from the selected keyframes by making proper joins, then needs to have fade in, fade out or cross dissolve effect in between for smooth transition.

8. Post-processing: Optional frame fixes are added, which enhance the visual quality, but are not a requirement. All duplicate or triadic frames are taken out.

9. Evaluation: How well the produced summary is done is assessed with the industry standards F-score and precision, perhaps feedback from users or research may also be included.

10. Optimization & Fine-tuning: Improving performance involves changing a few metrics like set thresholds.

11. Integration & Deployment:A fully functioning application or interface is created where a users can access the entire summarization system and works in the real world.

6.2 Code implemented

```
# importing the necessary packages
import numpy as np
import argparse
import imutils
import time
import cv2
import os
# parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-t", "--threshold", type=float, default=0.3,
    help="threshold when applyong non-maxima suppression")
args = vars(ap.parse_args())
# load the COCO class labels our YOLO model was trained on
labelsPath = os.path.sep.join(["yolo-coco/coco.names"])
LABELS = open(labelsPath).read().strip().split("\n")
# initialize a list of colors to represent each possible class label
np.random.seed(42)
```

```
COLORS = np.random.randint(0, 255, size=(len(LABELS), 3), dtype="uint8")
# derive the paths to the YOLO weights and model configuration
weightsPath =os.path.sep.join(["yolo-coco/yolov3.weights"])
configPath = os.path.sep.join(["yolo-coco/yolov3.cfg"])
# load our YOLO object detector trained on COCO dataset (80 classes)
print("[INFO] loading YOLO from disk...")
net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)
ln = net.getLayerNames()
ln = [ln[i- 1] for i in net.getUnconnectedOutLayers()]
vs = cv2.VideoCapture(0)
try:
        prop = cv2.cv.CV_CAP_PROP_FRAME_COU
                 else cv2.CAP_PROP_FRAME_COUNT
        total = int(vs.get(prop))
        print("[INFO] {} total frames in video".format(total))

# an error occurred while trying to determine the total number of frames in the video file
except:
        print("[INFO] could not determine # of frames in video")
        print("[INFO] no approx. completion time can be provided")
        total = -1
# loop over frames from the video file stream
while True:
# construct a blob from the input frame and then perform a forward
# pass of the YOLO object detector, giving us our
blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416,  416),
     swapRB=True, crop=False)
     net.setInput(blob)
     start = time.time()
     layerOutputs = net.forward(ln)
     end = time.time()
# detected bounding boxes, confidences, and class IDs, respectively
     boxes = []
     confidences = []
     classIDs = []
# loop over each of the layer outputs
     for output in layerOutputs
```

```
# loop over each of the detections
            for detection in output:
# extract the class ID and confidence (i.e., probability) of the current object detection
                  scores = detection[5:]
                  classID = np.argmax(scores)
                  confidence = scores[classID]
# probability is greater than the minimum probability
                  if confidence > args["confidence"]:
# scale the bounding box coordinates back relative to
                        box = detection[0:4] * np.array([W, H, W, H])
                        (centerX, centerY, width, height) = box.astype("int")

# use the center (x, y)-coordinates to derive the top
# and and left corner of the bounding box
                        x = int(centerX - (width / 2))
                        y = int(centerY - (height / 2))
# update our list of bounding box coordinates,confidences, and class IDs
                        boxes.append([x, y, int(width), int(height)])
              confidences.append(float(confidence))
                        classIDs.append(classID)
#apply non-maxima suppression to suppress weak, overlapping
# bounding boxes
      idxs = cv2.dnn.NMSBoxes(boxes, confidences, args["confidence"], args["threshold"])
       if len(idxs) > 0:
            count = 0
#loop over the indexes we are keeping
            for i in idxs.flatten():
#extract the bounding box coordinates
                  (x, y) = (boxes[i][0], boxes[i][1])
                  (w, h) = (boxes[i][2], boxes[i][3])
#draw a bounding box rectangle and label on the frame
          text="{}".format(LABELS[classIDs[i]])
          color = [int(c) for c in COLORS[classIDs[i]]]
           cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
                   text = "{}. {}: {:.4f}".format(count, LABELS[classIDs[i]],confidences[i])
cv2.putText(frame, text, (x, y -5),cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
      cv2.imshow('name',frame)
```

```
        if cv2.waitKey(1)
# release the file pointers
print("[INFO] cleaning up...")
#writer.release()
vs.release()
        if len(idxs) > 0:
            print('dfghjkljgchfcghjkj')
# loop over the indexes we are keeping
            for i in idxs.flatten():
# extract the bounding box


# draw a bounding box rectangle and label
                text1="{}".format(LABELS[classIDs[i]])
                print(text1)
        if text1 in query:
                    print('hghghghjg')
                    cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
                    text = "{}: {:.4f}".format(LABELS[classIDs[i]], confidences[i])
                     cv2.putText(frame,text,(x,y-5),  cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
#Check if it a Unique frame? Save it
                    if (((np.sum(np.absolute(frame - prev_frame)) / np.size(frame)) > threshold)):
                        writer.write(frame)
                        prev_frame = frame
                        a += 1
 cv2.imwrite('static/unique/unique'+str(a)+'.jpg', frame)
                    else:
                        prev_frame = frame
                        b += 1
cv2.imwrite('static/common/common'+str(b)+'.jpg', frame)
#show the output frame
        cv2.imshow("Frame", frame)
```

TESTING

Testing is the activity of verifying and validating that a certain system and/or its components were implemented as intended by the design. The system is operated for the purpose of finding discrepancies, errors, defects, or gaps in the requirements that were defined.

Testing Principle

Before coming up with effective test cases, a software engineer should first understand the principle underlying testing:Each test should be linked to some customer requirement.In such a way that the software produced is tested and verified against requirements which confirms that the software aligns and meets expectations.

7.1 TESTING PROCESS

There are various approaches that one can apply in testing software. Two most basic ones are:

1. **Black-BoxTesting**

   This technique is based on Using an application software without having insight into its design or design documentation. The examiner works with the user interface and does not care how the application processes information internally. The only consideration for the tester are the inputs and the outputs, which are provided through the interface.

2. **White-BoxTesting**

This is also referred to as glass-box or open-box testing. It requires an examination of the internal logical structure of the application code. The tester must hold the source code and must know how it functions. White-box testing helps identify logical errors, broken paths, and other internal issues.

7.2 LEVELS OF TESTING

Software testing is carried out at different levels to ensure that each part of the application functions correctly both individually and as a whole. These levels involve various methodologies and are typically structured as follows:

➢ **Functional Testing:**

One of the types of black-box testing functional testing deals with the confirmation of software execution pertaining to the requirement. Functional testing focuses on the behavior of the software, not the structure or workings of the code. Inputs are provided to the software , and the outputs are checked against the expected results and observations during testing..

This is done in fully integrated systems in order to operating, regardless of how they were built, as if from the perspective of the end-user.

- Based on the requirements document, determine what functions and features the application is expected to deliver.

- Create input data that are anticipated to be input into the application and appropriate for input requirements of the application.

- Establish the expected outputs as per the business rules defined in specifications as well as the provided test data.

- Design detailed test scenarios and implement each step, executing them as they would be performed in real-life situations.

- Examine the outputs of the test cases and determine whether the actual results align with expectations.

➢ **Non-functional Testing**

Non-functional testing looks at how well it does. It focuses on things like **speed, security, ease of use, and reliability**. Testing helps make sure the softwar well for users, done at different points during development.

7.3 TEST CASE.

Sl # Test Case : -	UTC-1
Name of Test: -	Image or video capture
Items being tested: -	Input video
Sample Input: -	video Stream
Expected output: -	Should accept input image or video
Actual output: -	Image Captured Successful
Remarks: -	Pass.

Sl # Test Case : -	UTC-2
Name of Test: -	object Detection
Items being tested: -	Labelling
Sample Input: -	Image or video
Expected output: -	Objects detection

Actual output: -	Objects Detected
Remarks: -	Test Passed

Table 7.1 Test Cases

These test cases cover essential functionalities of the Photo Gallery Website, ensuring that the core features work as expected and providing a basis for further testing.

RESULTS AND SNAPSHOTS

The object tracking method, images videos were successfully summarized, made easier to understand, and shorter in time and length. The object of interest was continuously tracked throughout the frames which made it possible to extract moments of key importance. This way, the videos were shortened to only the most useful portions, improving the user's understanding the matter, and also making material retrieval easier.

With the effort, this method was superior to other classical techniques as the computational resources and also processing hours were highly reduced. The more general view, subject videos no longer required lengthy searches for pinpointed delineations because this approach, in addition to detailed relevance, brought time efficiency and versatility in terms of subject adaptability.

8.1 SNAPSHOTS

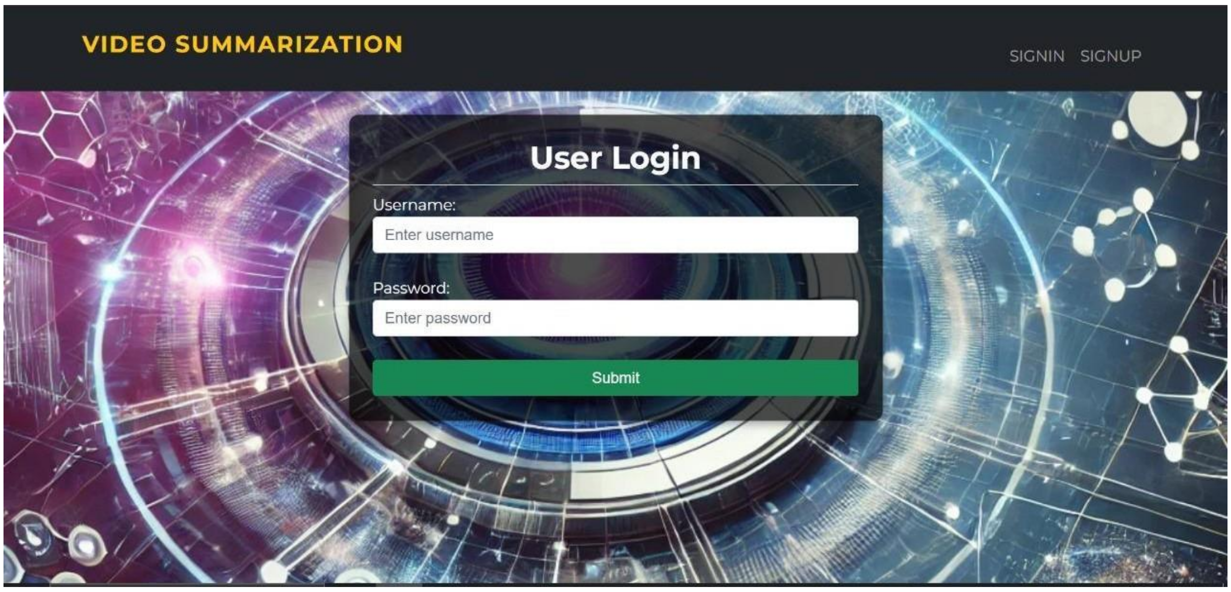

Fig 8.1. Output of Home/ Sign up Page

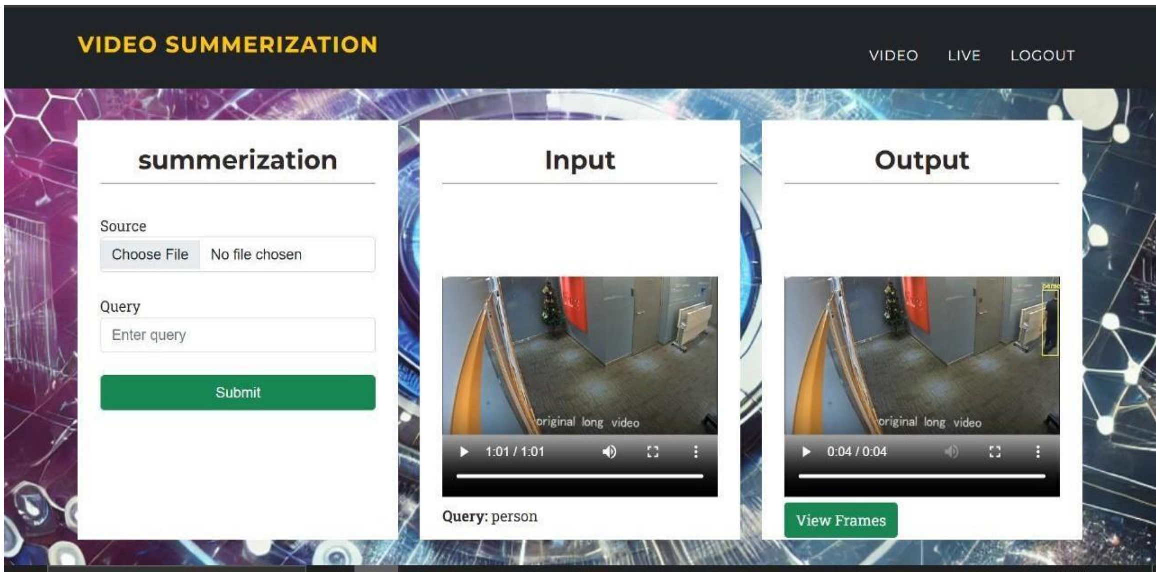

Fig 8.2. Output of User Signup page

Fig 8.3. Output of User Login Page

Fig 8.3 in addition to the straightforward, labeled form at the center of the upload photo page, users can effortlessly add a title, write a description, and select an appropriate category. From their device, users can browse to upload a photo by clicking the "Choose File" button. A thumbnail preview appearing immediately after a photo is selected provides immediate visual feedback.

CONCLUSION

The object tracking approach applied in this particular video summarization project seems to work exceptionally well in reducing long videos to meaningful summaries. Integration of computer vision algorithms allows prompt tracking of major objects in the video, assisting in detecting important moments and capturing them. While ensuring the fundamental content is retained, the video has been significantly reduced in length, increasing ease of understanding and navigation for users, independent of footage viewing.

The ease of highlighting significant scenes and also context and continuity takes center stage as one of the highlights of the method's efficiency. Eased information retrieval for users on top of the simplicity of accessing content translates into enhanced user experience. Furthermore, the summarizations produced were short while providing ample time, and detailed enough to retain meaning.

All in all, the approach supports the user's demand for rapidly available information while the control of the object tracking assists the user for better video content consumption.

REFERENCES

1. **Smith A, Johnson B (2022).** Object-Aware Video Summarization Using Deep Object Detection. Computer Vision and Multimedia Journal, Processing.

2. **Ghulam Mujtaba, Adeel Malik, Eun-Seok Ryu(2022).** A Lightweight Client-Driven Personalized Framework for Video Summarization with 2D CNN.

3. **Weibo Zeng, Xinran Min, Qiuyan Deng, Xingyue Zhao(2023).** Moving Target Tracking Framework Based on a Set and A Topological Space.

4. **Patel K, Lee M (2021).** Object-Centric Video Summarization via Multi-Modal Fusion. ACM Trans. on Multimedia Computing Communications and Applications.

5. **Obada Issa; Tamer Shanableh(2022).** Static Video Summarization Using CNN and HEVC Video Coding Techniques.