RSET
RAJAGIRI SCHOOL OF
ENGINEERING & TECHNOLOGY
(AUTONOMOUS)

Project Report On

CineComic : Automatic video-to-comic generation

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

in

Computer Science and Engineering

By

Raphael Tony (U2003162)
Reuben Dinny (U2003166)
Neha Bimal (U2003147)
Shreya Baburaj (U2003200)

Under the guidance of

Ms. Anu Maria Joykutty

Department of Computer Science and Engineering
Rajagiri School of Engineering & Technology (Autonomous)
(Parent University: APJ Abdul Kalam Technological University)
Rajagiri Valley, Kakkanad, Kochi, 682039
April 2024

CERTIFICATE

This is to certify that the project report entitled ***"CineComic : Automatic video-to-comic generation"*** *is a bonafide record of the work done by* ***Mr. Raphael Tony (U2003162), Mr. Reuben Dinny (U2003166), Ms. Neha Bimal (U2003147), Ms. Shreya Baburaj (U2003200)****, submitted to the Rajagiri School of Engineering & Technology (RSET) (Autonomous) in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology (B. Tech.) in Computer Science and Engineering during the academic year 2023-2024.*

Ms Anu Maria Joykutty
Project Guide
Asst. Professor
Dept. of CSE
RSET

Ms. Anita John
Project Coordinator
Asst. Professor
Dept. of CSE
RSET

Dr. Preetha K.G.
Professor & HoD
Dept. of CSE
RSET

ACKNOWLEDGMENT

We wish to express our sincere gratitude towards **Prof.(Dr). Sreejith P.S.**, Principal of RSET, and **Dr. Preetha K. G.**, Professor, Head of the Department of Computer Science for providing us with the opportunity to undertake our project, "Cinecomic".

We are highly indebted to our project coordinator, **Ms. Anita John**, Assistant Professor, Department of Computer Science and Engineering, for her valuable support.

It is indeed our pleasure and a moment of satisfaction for us to express our sincere gratitude to our project guide, **Ms. Anu Maria Joykutty**, Assistant Professor, Department of Computer Science and Engineering, for her patience and all the priceless advice and wisdom she has shared with us.

Last but not the least, We would like to express our sincere gratitude towards all other teachers and friends for their continuous support and constructive ideas.

Raphael Tony

Reuben Dinny

Neha Bimal

Shreya Baburaj

Abstract

Comics, a widely popular medium for art and information dissemination, is appreciated for its engaging storytelling techniques, including drawing, paneling, and layout. However, traditional comic creation is a time-consuming process that demands professional skills and various resources. This makes it challenging for non-professionals to create their comics. Today, videos and movies are ubiquitous, with a staggering volume of content available on platforms like YouTube. This vast reservoir of videos presents an opportunity to streamline comic content creation and reduce the burden on designers. Therefore, the conversion of videos or movies into comics books is an intriguing and valuable endeavor.

Our project aims to advance the state of automated comic book generation from videos. Given a video input, it performs the following : Extract keyframes, stylize them into comic-style images, create multi-page layouts, and generate emotion-aware speech balloons. The objective is to provide a user-friendly and efficient solution that can produce high-quality comic pages with visually rich layouts and diversified word balloons.

Building upon the foundation of a fully automatic system, our innovations include the automatic generation of subtitle files for videos and an improved keyframe extraction method. By automating the generation of subtitle files, we eliminate the need for user intervention, streamlining the comic creation process. Furthermore, our enhanced keyframe extraction method employs computer vision and deep learning techniques to select keyframes more accurately, resulting in comic-style images of higher quality.

Contents

Acknowledgment **i**

Abstract **ii**

List of Figures **v**

List of Tables **vi**

1 Introduction **1**

1.1 Background 1

1.2 Problem Definition 2

1.3 Scope and Motivation 2

1.4 Objectives 3

1.5 Challenges 3

1.6 Assumptions 4

1.7 Societal/Industrial Relevance 4

1.8 Organization of the Report 5

2 Literature Survey **6**

2.1 Existing System 7

3 Requirements **8**

3.1 Hardware and Software Requirements 8

4 System Architecture **9**

4.1 System Overview 9

4.2 Architecture Design 10

4.3 Module Division 10

4.4 Work Schedule-Gantt Chart 12

5 System Implementation **13**

5.1 Proposed Methodology/Algorithms 13

5.2 User Interface Design 18

5.3 Implementation Strategies 18

5.3.1 Subtitle Generation 19

5.3.2 Keyframe Extraction 21

5.3.3 Panel Layout 24

5.3.4 Region of Interest 28

5.3.5 Speech Bubble Placement 31

5.3.6 Lip Detection 33

5.3.7 Cartoonization 37

5.4 Chapter Conclusion 39

6 Results and Discussions **40**

6.1 Testing 40

6.1.1 Subtitle Generation 41

6.1.2 Keyframe Extraction 42

6.1.3 Page Layout Generation 42

6.1.4 Dialogue Bubble Placement 44

6.2 Quantitative Results 46

6.3 Discussion 46

6.4 Chapter Conclusion 46

7 Conclusion and Future Scope **48**

References **50**

Appendix A: Final Presentation **51**

Appendix B: Vision, Mission, Programme Outcomes and Course Outcomes **74**

Appendix C: CO-PO-PSO Mapping **79**

List of Figures

4.1 Architecture diagram 10
4.2 Gantt chart 12

5.1 Subtitle generation module 13
5.2 Keyframe extraction module 14
5.3 Multi-page layout module 15
5.4 Panel types 16
5.5 Dialogue bubble placement module 16
5.6 Cartoonization module 18

6.1 An SRT file generated using the *Harry Potter* sorting hat scene 41
6.2 Keyframes extracted for the *Harry Potter* sorting hat scene 42
6.3 A scene from *Harry Potter* 42
6.4 A scene from *Dune* 43
6.5 Final output with dialogue bubbles 44
6.6 Final output for *Dune* 45
6.7 Outputs for *The Dark Knight* and *The Chronicles of Narnia* 45

List of Tables

6.1 Execution times for various clips of movies 46

Chapter 1

Introduction

1.1 Background

In today's digital revolution era, the allure of screens and the proliferation of video content have redefined how information is consumed and stories are told. Amidst this transformation, the traditional appeal of books and textual narratives has faced a gradual decline, posing a significant challenge in fostering a love for reading among the younger generation. This shift raises concerns regarding language proficiency, critical thinking, and empathy development, all of which are intricately interwoven with literary immersion. The current trend also emphasizes the struggle in preserving the essence of storytelling in a rapidly evolving digital landscape.

Within this context, existing methods aimed at transposing videos into comic book formats encounter substantial limitations. They often fall short in capturing the emotional depth and immersive storytelling of the original content, hindering their ability to engage and inspire readers. This limitation emphasizes the critical need for an innovative approach that not only preserves the magic of storytelling but also resonates with the contemporary preferences of digitally immersed audiences.

The significance of this project lies in bridging the divide between digital content consumption and the preservation of narrative depth. By reimagining videos as visually stimulating comic narratives, this endeavor seeks to reignite a passion for reading while offering an alternative format that seamlessly integrates with modern consumption patterns. The development of an automated system to transform videos into captivating comics aims to not just adapt but also enhance the storytelling experience, emphasizing a pivotal effort to rejuvenate literary engagement amidst a digitally dominated era.

1.2 Problem Definition

The aim of this project is to address the inefficiencies in current methods of transforming videos into comics, characterized by their inability to capture emotional depth and engaging storytelling effectively. The problem at hand lies in the limitations of existing approaches, which struggle to seamlessly translate the nuanced emotional nuances and immersive storytelling of videos into visually compelling comic narratives.

1.3 Scope and Motivation

The project aims to develop an automated system converting videos into engaging comic narratives, catering to diverse audiences. It offers children and adolescents an engaging learning medium, merging digital content with comics to promote reading and reduce screen time. Educators gain innovative teaching tools, while parents benefit from captivating digital-to-print narratives to encourage reading habits. Enthusiasts of digital and print media experience seamless transitions between mediums, fostering adaptable reading experiences. Ultimately, the project bridges screen time and reading habits, using technology to revive interest in reading and blend digital fluency with traditional literary immersion.

The motivation behind this project is rooted in addressing the evolving dynamics of reading habits amidst the pervasive dominance of screens in today's digital age. With an aim to harmonize screen time with the promotion of reading, the project seeks to transform digital video content into printable comic book formats. This endeavor holds significance in mitigating the effects of prolonged screen exposure, reducing eye strain, and fostering a healthier balance between digital consumption and tangible reading experiences. By tapping into the inherent appeal of comics that perennially intrigues the younger generation, this project endeavors to rekindle an interest in reading while leveraging technology to present narratives in a format that seamlessly transitions from digital to print. Recognizing the cognitive impact and engagement potential of comics, this initiative aspires to offer a bridge between digital fluency and the enduring charm of printed literature, fostering an environment where reading becomes not just a pastime but a vibrant and immersive experience for all.

1.4 Objectives

- Develop an Automated System: Create a sophisticated automated system leveraging cutting-edge technology to seamlessly convert diverse video content into visually captivating comic book formats.
- Capture Emotional Nuances: Enhance the system to accurately capture and transcribe emotional nuances present in the original video content into the comic narrative, ensuring a rich and immersive storytelling experience.
- Optimize Visual Appeal: Implement advanced image processing algorithms and stylization techniques to elevate the visual appeal of the comic narratives, mimicking the aesthetic charm of traditional comics.
- Ensure Adaptability and Diversity: Design the system to be adaptable to various video genres and content styles, enabling it to effectively transform a diverse range of video content into engaging comic narratives.
- Streamline Multi-Page Layouts: Develop a framework for automated multi-page layout creation, optimizing panel arrangements and page compositions to facilitate coherent and visually appealing storytelling across multiple pages.
- Integrate User-Friendly Interface: Incorporate a user-friendly interface for ease of use, allowing users to interact with the system efficiently and customize the comic creation process to suit their preferences.

1.5 Challenges

1. **Performance and Scalability**

   Processing large amounts of video data efficiently and quickly to generate comics could pose performance challenges, especially when dealing with diverse video types and lengths.

2. **Transcribing dialogues with thick accents.**

   Thick accents may be hard to analyse and may not be accurately captured.

3. **Speaker Detection**

   Identifying the speaker accurately in every scene is a challenge.

1.6 Assumptions

1. **Video Quality and Content**

   The input videos are of at least 720p quality, and their content is suitable for conversion into comics. Videos with extremely low resolution or inappropriate content might not be suitable for conversion.

2. **Legal and Copyright Considerations**

   Assume that the videos used for conversion are either in the public domain or that we have the necessary rights and permissions to use and modify the content for this purpose. Copyright issues can be complex, and ensuring that the project operates within legal boundaries is critical.

1.7 Societal/Industrial Relevance

In an era dominated by digital devices, children's affinity for smartphones over traditional books has become a growing concern. Converting videos into comics is an innovative approach to foster love for reading. This strategy not only aligns with the digital preferences of today's youth but also harnesses the power of visual storytelling to seamlessly blend entertainment with education. By bridging the gap between technology and literature, this initiative aims to reignite the joy of reading, encouraging a new generation to explore the world of stories through a visually captivating lens. In the healthcare industry, medical professionals can convert video content into comics for patient education materials. This can help explain medical procedures, treatment plans, and health information in a more understandable format. Converting video content related to emergency response procedures into comics can aid in crisis communication. This format can be more accessible and memorable during high-stress situations.

1.8 Organization of the Report

The organization of the report is structured to provide a comprehensive understanding of the project's development and objectives. The initial segment comprises the acknowledgement, abstract, and a detailed contents page, offering a roadmap for the reader. The introduction section is then divided into several sub-sections, encompassing the background, problem definition, scope and motivation, objectives, social/industrial relevance, and organization of the report. This sets the stage for a thorough exploration of the project's context and significance. The subsequent sections include a literature survey, delving into existing works related to video-to-comic transformation, and an examination of the current system, paving the way for the proposed system's introduction. The design section elucidates the architectural framework, incorporating an architecture diagram and detailed module-wise diagrams for key components such as subtitle generation, keyframe extraction, multi-page layout, dialogue bubble placement and cartoonization. The assumptions and challenges, work breakdown, hardware and software requirements, project milestones and schedule, budget, and risks and challenges sections collectively contribute to a holistic project management perspective. The report concludes with a summarizing section, followed by a comprehensive list of references and appendices containing additional information, including the presentation, vision, mission, program outcomes, course outcomes, and CO-PO-PSO mapping, ensuring transparency and completeness in presenting the project's methodology and outcomes.

Chapter 2

Literature Survey

A few relevant works on creating manga or comics already exist. A semi-automated method for creating black-and-white comic books from an input movie was presented by Ryu et al. [1]. Keyframes are manually taken out of a specific movie. After that, stylization is done using a "comic cut converter" that is based on Mean Shift segmentation and a bilateral filter. Subsequently, they apply stylization to the background by distinguishing the foreground from the background. Lastly, users apply a stylized font and text bubble to the image. However, the main issue is that user intervention affects the entire system, necessitating a lot of manual labour.

A comprehensive method for creating comics was presented by Wang et al. [2]. Despite their claim that their technology is automatic, a pre-prepared script file including the speech content and speaker information was still required. Additionally, the resulting outputs lacked visual excitement due to the usage of predefined balloon shapes and typical grid-based layouts.

A content-aware method for manga generation and generated layouts that maximize the amount of information on a page was described by Jing et al. [3]. However, their methodology is limited to conversational videos—videos featuring character dialogue, like television shows. This implies that it cannot be used for conversation-free videos, such as vlogs. Additionally, the word balloons in their work have a set shape, which makes reading the manga book dull for readers.

Chu et al.'s approach [4] solves multi-page panel allocation explicitly using a labelling optimization to transform an image sequence into a comics-based presentation. However, they employed a set balloon form and only produced standard panel layouts with a restricted range of styles.

Pesko et al. [5] used an advanced keyframe-extraction technique using a deep summarization network. The style transformation was also performed using a custom-trained

CartoonGAN. Although it gave good results, it was not a complete system as panel allocation and speech bubble placement were absent.

2.1 Existing System

The best work we have seen so far is by Yang et al. [6]. A fully automatic system for generating comics was introduced that integrated keyframe extraction, stylization, multipage panel layout, smart balloon generation that was emotion-dependent and balloon placement. However, the authors have pointed out some of the flaws in the system - The keyframes extracted might be redundant and hence the overall extraction process can sometimes be inaccurate. Secondly, the system assumes the existence of a subtitle file for the video and has no provision for generating subtitles.

Chapter 3

Requirements

3.1 Hardware and Software Requirements

- **Processor:** Intel Core i5
- **RAM:** 8GB
- **GPU:** 4GB NVIDIA GeForce GTX 1650
- **Operating System:**
  - Windows: Windows 10 64-bit
  - Linux: Ubuntu
- **IDE:** Visual Studio Code
- **Libraries:** TensorFlow, OpenCV, MixedEmotion, Dlib
- **Frameworks:** Flask, Bootstrap

Chapter 4

System Architecture

4.1 System Overview

The proposed system represents a comprehensive framework designed to seamlessly convert digital video content into captivating comic narratives. Beginning with input processing and analysis, it ingests diverse video formats, extracts audio for speech-to-text conversion, and identifies keyframes using advanced deep learning networks. The system enhances these keyframes through comic-style image processing, mimicking traditional comic aesthetics. Through computed parameters, it arranges keyframes across multiple pages, ensuring cohesive storytelling with optimized panel layouts. Emotion-aware dialogue balloons are generated, positioning them near speakers' mouths for emotive representation. The system compiles these elements into final comic book formats, ensuring adaptability across genres and facilitating smooth transitions between digital and printable media. With an intuitive user interface, it offers creators and readers streamlined control over the conversion process. This proposed system aims to mitigate screen dependency, foster reading interest among youth, and create an engaging bridge between digital and print media, revolutionizing the experience of consuming video content through an innovative fusion of technology and storytelling.

4.2 Architecture Design

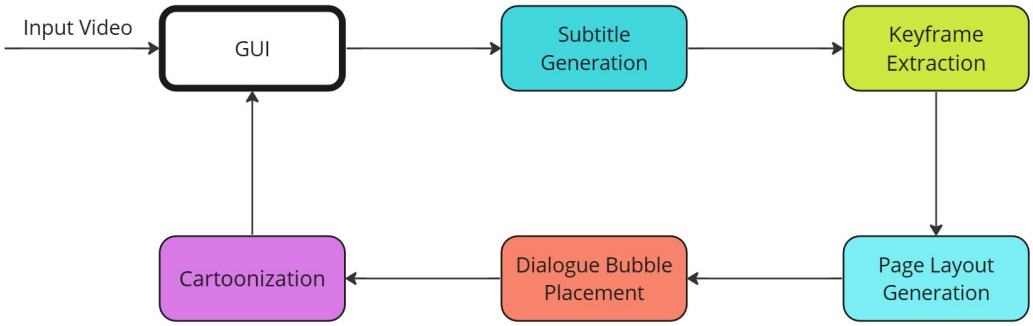

Figure 4.1: Architecture diagram

4.3 Module Division

The video-to-comic conversion system architecture consists of multiple stages to smoothly transform video content into comic book formats:

1. Input Processing:

- Video Input: Various formats of video content are ingested.
- Audio Extraction: Extracts audio from videos for speech-to-text processing.

2.Subtitle Generation

Uses a speech-to-text model to generate subtitles with timestamps.

3. Keyframe Extraction and Analysis:

- Frame Sampling: Samples frames at a set frequency from videos.
- Feature Extraction: Utilizes deep learning models like GoogLeNet v1 to extract features from frames.

- Highlightness Score Calculation: Deep Summarization Network (DSN) computes scores to identify keyframes.
- Dialogue Grouping: Groups frames based on dialogues to select significant keyframes.

4. Panel Layout and Multi-Page Composition:

- Region of interest of a frame is found using Class Activation Mapping Algorithm (CAM).
- CAM is used to produce a heat map for an input image which highlights the regions in the image that the model finds most relevant.
- This is used to map it to a panel type
- Hamming distance is used to select the page template that best matches the type of frames selected
- If there is any change between the type of the page template and frame, re-crop the frames

5. Balloon Generation and Placement:

- Emotion Analysis in Subtitles: Analyzes emotions in subtitles to determine the speech balloon shape.
- Dialogue Bubble Placement: Positions balloons at regions with least ROI near the speakers' mouths using face-detection libraries.

6. Stylization and Visual Enhancement:

- Comic-Style Image Processing: Applies style transfer algorithms to enhance keyframes visually, mimicking traditional comics.

7. Output Generation:

- Comic Book Generation: Compiles stylized frames, dialogue balloons, and layouts into a final comic book format.

- Printable Output: Renders the final comic book format for print media.

This system operates cohesively, integrating models, algorithms, and techniques to convert video content into engaging comic narratives.

4.4 Work Schedule-Gantt Chart

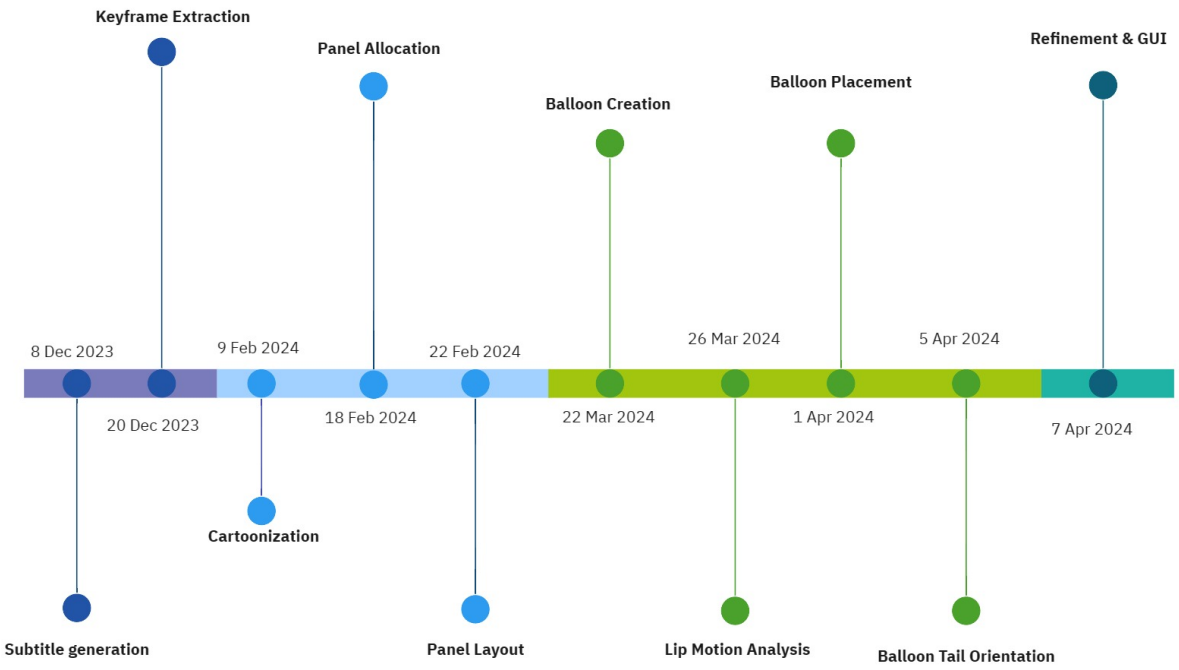

Figure 4.2: Gantt chart

Chapter 5

System Implementation

The entire application was built using Python. Flask was used for the backend server while the User Interface was served using HTML and Bootstrap. Flask is a light-weight framework in Python that aids in web application development. Bootstrap is a CSS framework for importing pre-styled templates.

5.1 Proposed Methodology/Algorithms

Subtitle Generation

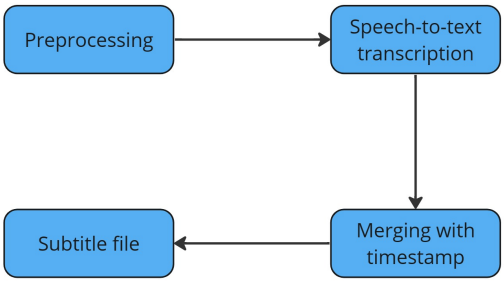

Figure 5.1: Subtitle generation module

Whisper [7] is an open-source speech-to-text model that uses an encoder-decoder transformer. The audio is extracted from the video and passed on to the Whisper model for speech transcription. Based on the original audio file, timestamps are added to the subtitle file following the SRT file format.

Keyframe Extraction

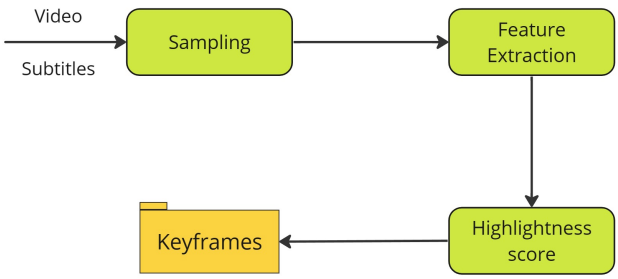

Figure 5.2: Keyframe extraction module

For each subtitle in the subtitles file, a keyframe is extracted. Keyframe extraction is done by sampling two frames per second between the start time and end time of each subtitle in the video. The extracted frames are sent to GoogLeNet v1—which has been pre-trained on ImageNet, and is used to extract features from the frames. The extracted features are passed to the DSN (Deep Summarization Network).

DSN was developed by Zhou *et al.* and it gives a score called highlightness score, to each frame which indicates the probability that the frame is to be selected as a representative frame of the video.

DSN uses an encoder decoder architecture, where the encoder is a CNN(the above mentioned googlenet) and the decoder is a bidirectional recurrent neural network (BiRNN) topped with a fully connected (FC) layer. The BiRNN takes as input the feature vector produced by googlenet and produces the highlightness score. To train the the DSN they used reinforcement learning based approach, and it accounts for both diversity and representativeness of the frame.

The frame with the highest highlightness score is chosen as the keyframe for that subtitle.

Multi-Page Layout

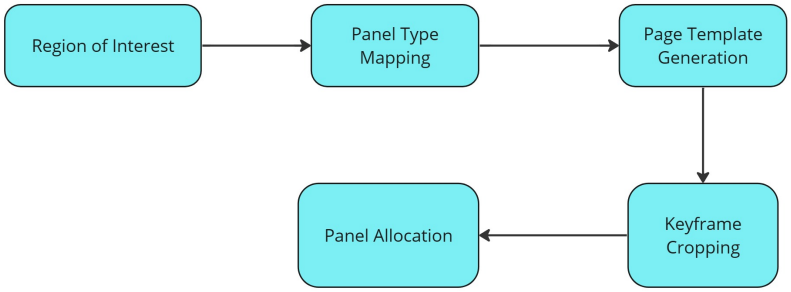

Figure 5.3: Multi-page layout module

The Region of Interest in a frame is captured by using a Class Activation Mapping Algorithm (CAM). CAM is used to produce a heatmap for an input image which highlights the regions in the image that the model finds most relevant. In CAM, an image is processed through a Convolutional Neural Network (CNN) to extract features. These features are then reduced in dimensionality by a global average pooling layer. Finally, a fully connected layer uses these features to generate a heatmap. A bounding box is drawn around the heatmap and the coordinates of the box is passed to the next stage.

4 types of panels were defined and using the coordinates of the bounding box calculated by the CAM algorithm, the most suitable panel type is selected for each frame. Page layouts are defined as the combination of these panel types .After each frame is mapped to a panel type,the input video is converted to a sequence of panel types. The input sequence is segmented with the help of the predefined page templates and hamming distance is used to determine which page template is most similar to the segmented sequence.

We have defined a set of predefined template sequences like 14124114, 312341 which represents a valid page. A valid page is a page where all the panels fit completely in a page. Then we iterate over the page sequence, comparing the hamming distance between the template sequence and the portion of the page sequence. This is done so that the template with the smallest amount of readjustment to input sequence is selected. The template sequence which has the lowest distance is selected and appended to page templates. This process completes until we iterate through the entire page sequence. Finally page templates have a list of sequences where each sequence represents a page.

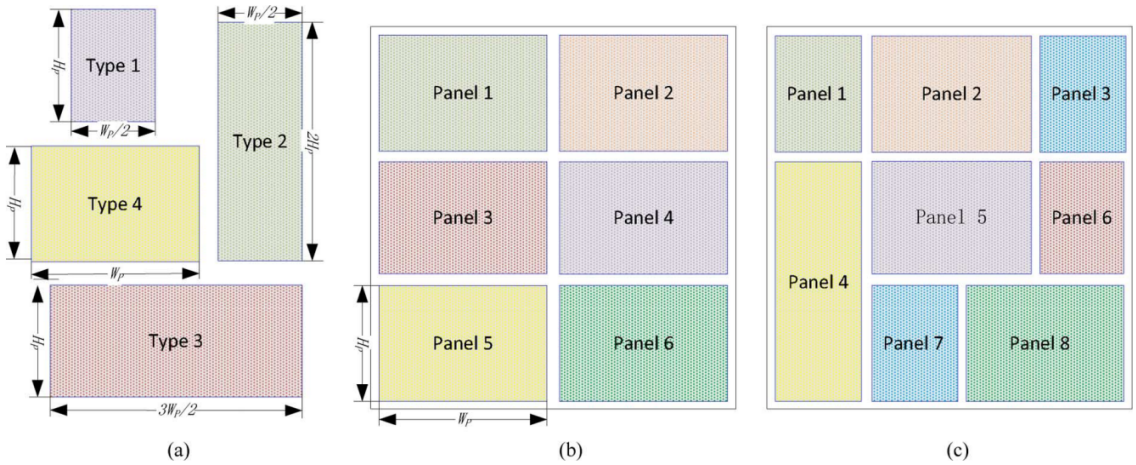

Figure 5.4: Panel types

The keyframes are then cropped with respect to the aspect ratios of the corresponding panel types in the page template sequence. To ensure that most of the ROI region is captured while cropping, we first find the center of the bounding box coordinates obtained by applying CAM. The new coordinates are then obtained such that the aspect ratio of the panel type that the frame is to be cropped to is obtained relative to the center. However, on repositioning the bounding box coordinates, there is a possibility that the box size is greater than the image size or is outside the boundaries of the image. To resolve this, the new bounding box is scaled down if it exceeds the image boundaries. Additionally, if any side of the bounding box extends beyond the image boundary, it is translated inwards. These new coordinates constitute the final crop coordinates, and the image is cropped with respect to these coordinates.

Dialogue Bubble Placement

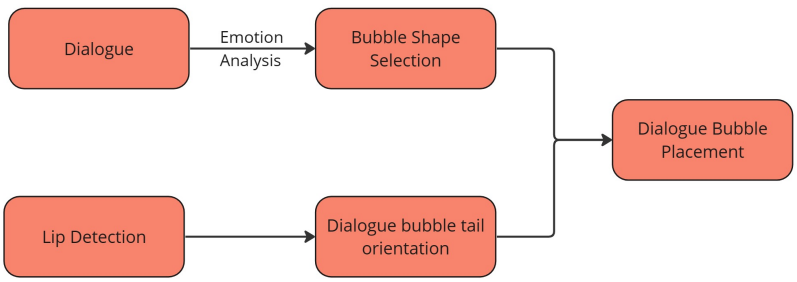

Figure 5.5: Dialogue bubble placement module

In order to choose the shape of the speech bubble based on the emotion of the bubble, we pass the subtitle file to the text emotion recognition model. The model classifies the emotion of the dialogue into 28 different classes of emotions. Based on the emotion, we further classify the bubble shape as either a jagged speech bubble or the regular speech bubble.

The dialogue bubble, along with the subtitle, is now to be positioned within its corresponding panel in the page layout. To ensure that the bubble does not overlap the main components of the image, we need to position it at the point with the least ROI. This point is determined using CAM and is regarded as the center of the bubble. Since, in most cases, the point of least ROI is most likely near the boundary, we translate the point inwards by adding padding to the coordinate, ensuring the bubble remains within the image boundaries. Subsequently, the bubble is placed into the corresponding panel on the page based on the obtained coordinates, and the dialogue is added on top of it.

The lip detection module uses the "Dlib" face-detector Python library to detect mouths via histogram of oriented gradients (HOG). It extracts 68 feature points per face, with points 48th to 59th outlining the mouth. For each frame the following 3 cases are considered:

- If no lips are detected, value of -1 is returned.
- If one lip is detected, the coordinates of the lip is returned.
- If multiple lips are detected, the speaker needs to be identified. In such a case, lip motion analysis is performed in which the mean squared difference of pixel values in the mouth region between consecutive frames. A threshold determines if a character is moving the lips based on the calculated difference. The number of lip movements is counted and a threshold is used to determine if the character is speaking. The coordinates of the detected speaker is then returned.

Tail is initially placed at the center of the bubble. To make the tail point in the direction of the speaker, the angle with which it must tilt is given by the tan inverse of the difference of the x coordinates of the lip and the bubble divided by the difference of the y coordinates of the lip and the bubble. The tail displacement along the x-direction is calculated by 80*cosine component of the angle and along the y-direction is calculated by 80*sine component of the angle.

Cartoonization

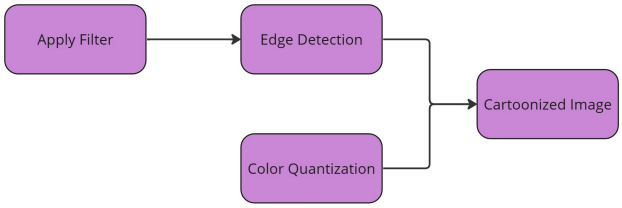

Figure 5.6: Cartoonization module

Cartoonizing each keyframe consists of the following steps. The image is processed using various filters such as Gaussian blur, Median blur and Bilateral filter to reduce the noise in the image.

Then a Laplacian filter is used as an edge detection filter, which identifies the boundaries of objects within the image. The filtered image is converted to greyscale, which simplifies the image by removing color data. Then, a thresholding technique is applied to further simplify the image. This technique turns the image into a binary image, making it purely black and white.

At the same time, the colors in the original image are reduced. This can be done using various techniques, but the goal is to limit the number of colors in the image, which gives it a more cartoon-like appearance. We have used the K-means algorithm to achieve this. Finally, the edge-detected image and the color-reduced image are combined. This results in an image that has bold edges and simplified colors, giving it a cartoon-like appearance.

5.2 User Interface Design

A simple HTML page was styled with Bootstrap. The video can be input in 2 ways:

1. A video file can be directly uploaded
2. A Youtube link to the file can be input

5.3 Implementation Strategies

The entire application was built using Python. The following sections contain code snippets for the various modules.

5.3.1 Subtitle Generation

Subtitle generation using speech-to-text transcription with Whisper and converting it into a SRT format. The file is saved as 'test1.srt'

```
import time
import math
import ffmpeg

from faster_whisper import WhisperModel

input_video = "harry.mp4"
input_video_name = input_video.replace(".mp4", "")

def extract_audio():
    extracted_audio = f"audio-{input_video_name}.wav"
    stream = ffmpeg.input(input_video)
    stream = ffmpeg.output(stream, extracted_audio)
    ffmpeg.run(stream, overwrite_output=True)
    return extracted_audio

def transcribe(audio):
    model = WhisperModel("small")
    segments, info = model.transcribe(audio)
    language = info[0]
    print("Transcription language", info[0])
    segments = list(segments)
    for segment in segments:
        # print(segment)
        print("[%.2fs -> %.2fs] %s" %
              (segment.start, segment.end, segment.text))
    return language, segments
```

```
def format_time(seconds):

    hours = math.floor(seconds / 3600)
    seconds %= 3600
    minutes = math.floor(seconds / 60)
    seconds %= 60
    milliseconds = round((seconds - math.floor(seconds)) * 1000)
    seconds = math.floor(seconds)
    formatted_time = f"{hours:02d}:{minutes:02d}:{seconds:01d},{milliseconds:03d}"

    return formatted_time

def generate_subtitle_file(language, segments):

    subtitle_file = f"sub-{input_video_name}.{language}.srt"
    text = ""
    for index, segment in enumerate(segments):
        segment_start = format_time(segment.start)
        segment_end = format_time(segment.end)
        text += f"{str(index+1)} \n"
        text += f"{segment_start} --> {segment_end} \n"
        text += f"{segment.text} \n"
        text += "\n"

    f = open(subtitle_file, "w")
    f.write(text)
    f.close()

    return subtitle_file

def run():
```

```
    extracted_audio = extract_audio()
    language, segments = transcribe(audio=extracted_audio)
    subtitle_file = generate_subtitle_file(
        language=language,
        segments=segments
    )

run()
```

5.3.2 Keyframe Extraction

Frames were sampled and passed through the DSN to pick out the important keyframes

```
def _get_features(frames, gpu=True, batch_size=1):
    # Load pre-trained GoogLeNet model
    googlenet = torch.hub.load('pytorch/vision:v0.10.0', 'googlenet',
    ↪  weights='GoogLeNet_Weights.DEFAULT')

    # Remove the classification layer (last layer) to obtain features
    googlenet = torch.nn.Sequential(*(list(googlenet.children())[:-1]))

    # Set the model to evaluation mode
    googlenet.eval()

    # Initialize a list to store the features
    features = []

    # Image preprocessing pipeline
    preprocess = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
```

```
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229,
        ↪  0.224, 0.225]),
    ])

    # Iterate through frames
    for frame_path in frames:
        # Load and preprocess the frame
        input_image = Image.open(frame_path)
        input_tensor = preprocess(input_image)
        input_batch = input_tensor.unsqueeze(0)  # Add batch dimension

        # Move the input and model to GPU if available
        if gpu:
            input_batch = input_batch.to('cuda')
            googlenet.to('cuda')

        # Perform feature extraction
        with torch.no_grad():
            output = googlenet(input_batch)

        # Append the features to the list
        features.append(output.squeeze().cpu().numpy())

    # Convert the list of features to a NumPy array
    features = np.array(features)

    return features.astype(np.float32)

# Cell 3
def _get_probs(features, gpu=True, mode=0):
    # model_cache_key = "keyframes_rl_model_cache_" + str(mode)
```

```
    if mode == 1:
        model_path = "backend/keyframes/pretrained_model/model_1.pth.tar"
    else:
        model_path = "backend/keyframes/pretrained_model/model_0.pth.tar"
    model = DSN(in_dim=1024, hid_dim=256, num_layers=1, cell="lstm")
    if gpu:
        checkpoint = torch.load(model_path)
    else:
        checkpoint = torch.load(model_path, map_location='cpu')
    model.load_state_dict(checkpoint)
    if gpu:
        model = nn.DataParallel(model).cuda()
    model.eval()

    seq = torch.from_numpy(features).unsqueeze(0)
    if gpu: seq = seq.cuda()
    probs = model(seq)
    probs = probs.data.cpu().squeeze().numpy()
    return probs



def generate_keyframes(video):
    data=""
    with open("test1.srt") as f:
        data = f.read()

    subs = srt.parse(data)
    torch.cuda.empty_cache()

    for sub in subs:
```

```
        frames = []
        if not os.path.exists(f"frames/sub{sub.index}"):
            os.makedirs(f"frames/sub{sub.index}")
        frames = extract_frames(
            video,
            os.path.join("frames",f"sub{sub.index}"),
            sub.start.total_seconds(),sub.end.total_seconds(),2
        )
        features = _get_features(frames)
        highlight_scores = _get_probs(features)

        try:
            highlight_scores = list(highlight_scores)
            sorted_indices = [i[0] for i in
            ↪  sorted(enumerate(highlight_scores), key=lambda x: x[1])]
            selected_keyframe = sorted_indices[-1]
            frames[selected_keyframe]
            copy_and_rename_file(frames[selected_keyframe],
            ↪  os.path.join("frames","final"),
            ↪  f"frame{sub.index:03}.png")
```

5.3.3 Panel Layout

Mapping input sequence to suitable panel layouts using Hamming distance

```
import os
import random
import copy
from backend.class_def import panel



template_specs = {
```

```
    "1" : {
        "span" : 1,
        "direction": "row"
    },
    "2" : {
        "span" : 2,
        "direction": "row"
    },
    "3" : {
        "span" : 3,
        "direction": "column"
    },
     "4" : {
        "span" : 2,
        "direction": "column"
    }

}

input = '433343333343343333443333443333433333433444433433'



def hammingDist(str1, str2):
    i = 0
    count = 0

    while(i < len(str1)):
        if(str1[i] != str2[i]):
            count += 1
        i += 1
```

```
    return count


templates = ['14124114','312341' , '4432111' , '21411241' ,
             '3241141' , '13411141' , '12411131' ,'1321113', '131423' ,
    '142344' , '234241','2411413','3141214','42111131']

min_length = 6
folder_path = 'frames/final' # Specify the folder path



def get_templates(input):
    page_templates = []
    start = 0

    while(start<len(input)):
        # print(f"start: {start}")
        result = []
        print(input)
        for template in templates:

            temp = input[start:start + len(template)]
            print(f"start: {start} len:{len(template)} temp:{temp}" )
            result.append(hammingDist(temp,template))


        page_templates.append(templates[result.index(min(result))])

        start = start + len(templates[result.index(min(result))])
```

```
    if(len(temp) < min_length):
        if(len(temp) ==1):
          temp="5"
        elif(len(temp) ==2):
          temp="67"
        elif(len(temp) ==3):
          temp="666"
        elif(len(temp) ==4):
          temp="4488"
        elif(len(temp) ==5):
          temp="44446"

        page_templates[len(page_templates)-1] = temp
        # print("****************")

    return page_templates


def panel_create(page_templates):

    panels = []

    images = get_files_in_folder(folder_path)
    print(images)
    count_images = 1

    for page_template in page_templates:
```

```
        if(len(page_template)<min_length): #To handle last page
            panels = last_page(panels,count_images,len(page_template))
            break


        count = 1

        for i in page_template:

            if(template_specs[i]['direction'] == 'row'):
                new = panel(f'frame{count_images:03d}',
                            template_specs[i]['span'] , 1)
            else:
                new = panel(f'frame{count_images:03d}', 1 ,
                            template_specs[i]['span'])
            panels.append(new)
            count = count+1
            count_images+=1



    return(panels)
```

5.3.4 Region of Interest

Region of Interest was found using torchcam.

```
def get_coordinates(img_path):
    # Get your input
    img = read_image(img_path)
    # Preprocess it for your chosen model
```

```
input_tensor = normalize(resize(img, (224, 224)) / 255., [0.485,
↪  0.456, 0.406], [0.229, 0.224, 0.225])

with SmoothGradCAMpp(model) as cam_extractor:
  # Preprocess your data and feed it to the model
  out = model(input_tensor.unsqueeze(0))
  # Retrieve the CAM by passing the class index and the model output
  activation_map = cam_extractor(out.squeeze(0).argmax().item(), out)

cam_map = activation_map[0][0]
arr = np.array(cam_map.cpu())
ten_map = tensor(arr)
cms = cam_map.shape[0]

x_ = img.shape[2] // cms
y_ = img.shape[1] // cms

CAM_data.append({'x_': x_, 'y_': y_, 'ten_map': ten_map})

top,bottom,left,right = -1,-1,-1,-1
threshold = 0.2

# Top
found = False
for i in range(0, ten_map.shape[0]):
    for j in range(0,ten_map.shape[1]):
      if ten_map[i][j] >= threshold:
        top = i
        found = True
        break
    if found:
```

```
      break

#Bottom
found = False
for i in range(ten_map.shape[0]-1, -1, -1):
    for j in range(0,ten_map.shape[1]):
      if ten_map[i][j] >= threshold:
        bottom = i
        found = True
        break
    if found:
      break

#Left
found = False
for j in range(0, ten_map.shape[1]):
    for i in range(0,ten_map.shape[0]):
      if ten_map[i][j] >= threshold:
        left = j
        found = True
        break
    if found:
      break

#Right
found = False
for j in range(ten_map.shape[1]-1, -1, -1):
    for i in range(0,ten_map.shape[0]):
      if ten_map[i][j] >= threshold:
        right = j
        found = True
```

```
            break
        if found:
            break

    top = top * y_
    bottom = bottom * y_
    left = left * x_
    right = right * x_
    left,right,top,bottom

    return left, right, top, bottom
```

5.3.5 Speech Bubble Placement

The bubble was placed in the least region of interest.

```
def get_bubble_position(crop_coord, CAM_data):
    left, right, top, bottom = crop_coord
    x_ = CAM_data['x_']
    y_ = CAM_data['y_']
    ten_map = CAM_data['ten_map']
    print(ten_map)

    new_top = int(top / y_)
    new_bottom = int(bottom / y_)
    new_left = int(left / x_)
    new_right = int(right / x_)
    print(new_top, new_bottom, new_left, new_right)

    # Initialize variables to store the minimum value and its
    ↪ corresponding index
    min_value = float('inf')  # Initialize min value to positive infinity
    min_point = None  # Initialize min point to None
```

```
# Top
found = False
for i in range(new_left, new_right + 1):
    for j in range(new_top, new_bottom + 1):
        if (i < ten_map.shape[0] and j < ten_map.shape[1]) and
        ↪  ten_map[i][j] < min_value:
            min_point = (i, j)
            min_value = ten_map[i][j]

least_roi_x = min_point[0] * x_
least_roi_y = min_point[1] * y_

if least_roi_x < left:
    least_roi_x = left
elif least_roi_x > right:
    least_roi_x = right
if least_roi_y < top:
    least_roi_y = top
elif least_roi_y > bottom:
    least_roi_y = bottom

least_roi_x -= left
least_roi_y -= top
print("Least ROI coords: ", least_roi_x, least_roi_y)

least_roi_x, least_roi_y =
↪  convert_to_css_pixel(least_roi_x,least_roi_y,crop_coord)
print("Least ROI coords after scaling: ", least_roi_x, least_roi_y)
```

```
    least_roi_x, least_roi_y = add_bubble_padding(least_roi_x,
    ↪  least_roi_y, crop_coord)

    return least_roi_x, least_roi_y
```

5.3.6 Lip Detection

Identifying the speaker when multiple lips are detected by analysing consecutive frames for lip movement.

```
def get_multi_speaker_lips(sub,video, keyframe_face_rects):
    start_time = sub.start.total_seconds()
    end_time = sub.end.total_seconds()
    keyframe_path = f"frames/final/frame{sub.index:03}.png"

    vid = cv2.VideoCapture(video)       # Read video
    frames_per_sec = vid.get(cv2.CAP_PROP_FPS)  # Number of frames per
    ↪  second
    # total_frames = int(vid.get(cv2.CAP_PROP_FRAME_COUNT))
    # frames_count = total_frames // frameRate

    # Calculate the frame skip value
    select_index = floor(frames_per_sec / SAMPLE_RATE)  # Select every
    ↪  (skip_rate)'th position frames to get the SAMPLE_RATE number of
    ↪  frames per second
    start_frame = int(start_time * frames_per_sec)
    end_frame = int(end_time * frames_per_sec)

    vid.set(cv2.CAP_PROP_POS_FRAMES, start_frame)
    print("FPS,  select index = ", frames_per_sec, select_index)

    # Initialize frame counter
    current_frame = start_frame
```

```
total_frames_selected = 0

# Parse into frames
frame_buffer = []               # A list to hold frame images
frame_buffer_color = []         # A list to hold original frame
↪ images
while(current_frame<end_frame):
    success, frame = vid.read()                 # Read frame
    if not success:
        break
    if current_frame % select_index == 0:                           #
    ↪ Break if no frame to read left
        gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)   # Convert
        ↪ image into grayscale
        frame_buffer.append(gray)                   # Add image to the
        ↪ frame buffer
        frame_buffer_color.append(frame)
        total_frames_selected += 1
    current_frame += 1
vid.release()

prev_lip_dist = {}      #2D[i][j]
lip_motion_count = {}   #1D[j]
lip_coords = {}         #1D[j]
avg_gap = {}            #2D[i][j]

start_flag = False      #To skip the lip distance calculation for
↪ first frame

for (i, image) in enumerate(frame_buffer):          # Iterate on
↪ frame buffer
```

```
face_rects = face_detector(image,1)              # Detect face
if len(face_rects) < 1:                  # No face detected
    print("No face detected: frame ",i)
    continue
if len(face_rects) >= 1:                   # Too many face
↪  detected

    # Check if area of the first face rectangle is close to
    ↪  keyframe
    if not similar_to_keyframe(face_rects, keyframe_face_rects):
        print("frame not similar: ",i)
        continue

    largest_face = max(face_rects, key=lambda rect: rect.area())
    print("largest face: ", largest_face)

    avg_gap[i] = {}
    prev_lip_dist[i] = {}
    for (j,rect) in enumerate(face_rects):
        if (rect.area() / largest_face.area()) < FACE_AREA:
        ↪  #Consider lip only if face area crosses a
        ↪  threshold(ROI)
            print("Lip skipped: ", j, rect)
            continue

        prev_lip_dist[i][j] = 0
        landmark = landmark_detector(image, rect)   # Detect face
        ↪  landmarks
        # landmark = shape_to_list(landmark)

        part_61 = (landmark.part(61).x,landmark.part(61).y)
```

```
part_67 = (landmark.part(67).x,landmark.part(67).y)
part_62 = (landmark.part(62).x,landmark.part(62).y)
part_66 = (landmark.part(66).x,landmark.part(66).y)
part_63 = (landmark.part(63).x,landmark.part(63).y)
part_65 = (landmark.part(65).x,landmark.part(65).y)
A = dist(part_61, part_67)
B = dist(part_62, part_66)
C = dist(part_63, part_65)

avg_gap[i][j] = (A + B + C) / 3.0

# Store lip coordinate if encountered for first time
if j not in lip_coords:
    lip_coords[j] = part_65

# Loop runs for the first time
if start_flag==False:
    prev_lip_dist[i][j] = avg_gap[i][j]
    start_flag = True
    continue

# Check if lip distance between continous frame is above
↪  threshold, if so increase lip count
print("Difference for frame {0}, lip {1}: {2}".format( i,
↪  j, abs(avg_gap[i][j] - prev_lip_dist[i][j])) )
if abs(avg_gap[i][j] - prev_lip_dist[i][j]) > THETA1:
    lip_motion_count[j] = lip_motion_count.get(j,0) + 1
prev_lip_dist[i][j] = avg_gap[i][j]
```

```
    print("Lip motion count, total_frames_selected = ", lip_motion_count,
    ↪  total_frames_selected)
    # print("max lip count ratio = ", lip_motion_count /
    ↪  (total_frames_selected-1))
    try:
        max_lip_index = max(lip_motion_count, key=lip_motion_count.get)
        # max_value = lip_motion_count[max_lip_index]
        # if max_lip_count / (total_frames_selected-1) > THETA2:
        #     print("speaking")
        if lip_motion_count[max_lip_index] / (total_frames_selected-1) >
        ↪  THETA2:
            return lip_coords[max_lip_index]
        else:
            return (-1,-1)
    except ValueError:
        return (-1,-1)
```

5.3.7 Cartoonization

Cartoonization was performed by Colour Quantization method using K-means .

```
def cartoonize(img_path):
    # Opens an image with cv2
    img = cv2.imread(img_path)

    # Apply some Gaussian blur on the image
    img_gb = cv2.GaussianBlur(img, (7, 7) ,0)
    # Apply some Median blur on the image
    img_mb = cv2.medianBlur(img_gb, 5)
    # Apply a bilateral filer on the image
```

```
img_bf = cv2.bilateralFilter(img_mb, 5, 80, 80)


# Use the laplace filter to detect edges
img_lp_al = cv2.Laplacian(img_bf, cv2.CV_8U, ksize=5)


# Convert the image to greyscale (1D)
img_lp_al_grey = cv2.cvtColor(img_lp_al, cv2.COLOR_BGR2GRAY)

# Remove some additional noise
blur_al = cv2.GaussianBlur(img_lp_al_grey, (5, 5), 0)

# Apply a threshold (Otsu)
_, tresh_al = cv2.threshold(blur_al, 245, 255,cv2.THRESH_BINARY +
↪  cv2.THRESH_OTSU)

# Invert the black and the white
inverted_Bilateral = cv2.subtract(255, tresh_al)

# Reduce the colors of the original image
# div = 64
# img_bins = img // div * div + div // 2

# Reshape the image
img_reshaped = img.reshape((-1,3))
# convert to np.float32
img_reshaped = np.float32(img_reshaped)
# Set the Kmeans criteria
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10,
↪  1.0)
```

```
# Set the amount of K (colors)
K = 16
# Apply Kmeans
_, label, center = cv2.kmeans(img_reshaped, K, None, criteria, 10,
↪  cv2.KMEANS_RANDOM_CENTERS)

# Covert it back to np.int8
center = np.uint8(center)
res = center[label.flatten()]
# Reshape it back to an image
img_Kmeans = res.reshape((img.shape))

# Convert the mask image back to color
inverted_Bilateral = cv2.cvtColor(inverted_Bilateral,
↪  cv2.COLOR_GRAY2RGB)
# Combine the edge image and the binned image
cartoon_Bilateral = cv2.bitwise_and(inverted_Bilateral, img_Kmeans)

# Save the image
cv2.imwrite(img_path, cartoon_Bilateral)
```

5.4 Chapter Conclusion

This chapter outlines the implementation of CineComic, leveraging Python with Flask for backend development and HTML styled with Bootstrap for the user interface. Key functionalities include subtitle generation using the Whisper speech-to-text model, keyframe extraction with GoogLeNet v1 and DSN, multi-page layout with CAM, dialogue bubble placement via lip detection, and cartoonization techniques. The user-friendly interface allows for video input via direct upload or YouTube links. This comprehensive system offers an efficient and enjoyable way to convert videos into visually engaging comic strips.

Chapter 6

Results and Discussions

In this chapter, we delve into the results and discussions of the implemented video-to-comics solution. The following sections provide an overview of the achieved results and in-depth discussions on the outcomes.

6.1 Testing

The sorting hat scene from *Harry Potter* was used to test the solution. The results in the following section are based on this scene primarily. Additionally other examples from movies such as *Dune* and *Chronicles of Narnia* were also included. The intermediate results in each stage can be seen in the results below:

6.1.1 Subtitle Generation

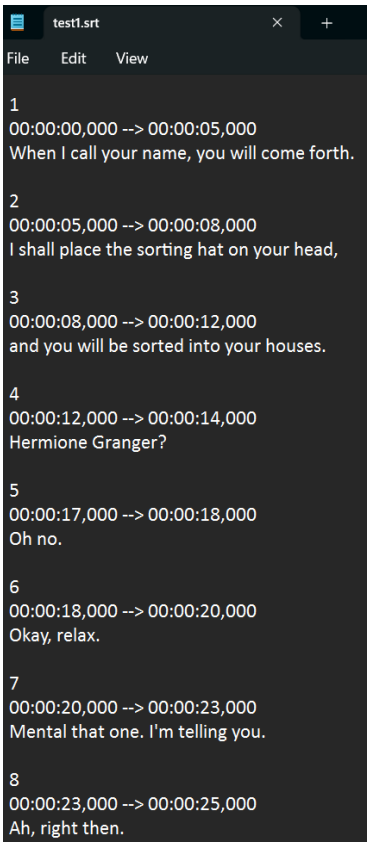

Figure 6.1: An SRT file generated using the *Harry Potter* sorting hat scene

The subtitle file was successfully generated using the Whisper model as shown in Figure 6.1 . The SRT format consists of the subtitle index, timestamps and the subtitle content.

6.1.2 Keyframe Extraction

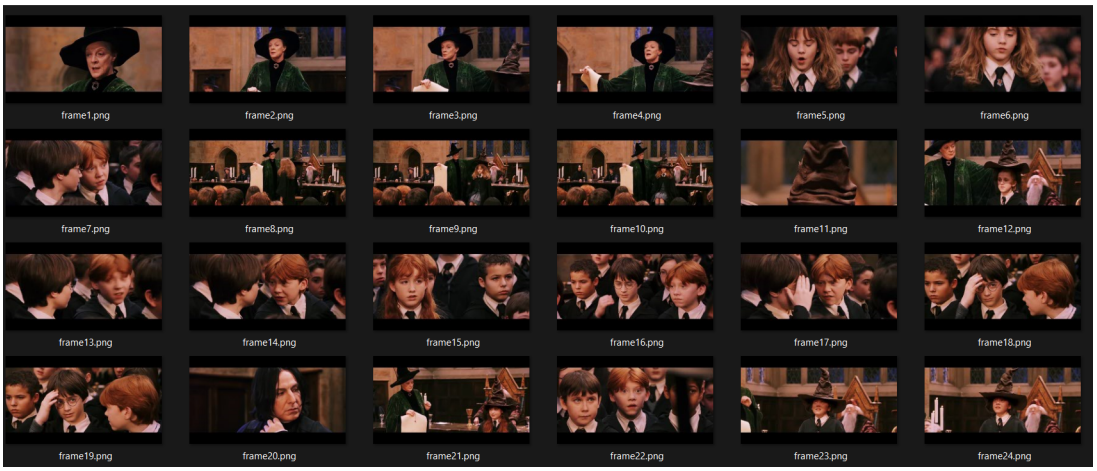

Figure 6.2: Keyframes extracted for the *Harry Potter* sorting hat scene

For every subtitle in the subtitle file, the corresponding keyframes were successfully extracted using the timestamps as shown in 6.2

6.1.3 Page Layout Generation

Figure 6.3: A scene from *Harry Potter*

Figure 6.4: A scene from *Dune*

Figure 6.3 shows the successful generation of the layout for the *Harry Potter* scene. The various panel type shapes can be identified. Figure 6.4 is another example of a scene from the movie *Dune*.

6.1.4 Dialogue Bubble Placement

Figure 6.5: Final output with dialogue bubbles

Figure 6.6: Final output for *Dune*

Figure 6.7: Outputs for *The Dark Knight* and *The Chronicles of Narnia*

The dialogue bubbles were successfully placed in the region of least ROI. We can observe that in some frames, the lips weren't detected and as a result, the tail was omitted from the bubble. Few results are shown in Figures 6.5, 6.6 and 6.7.

6.2 Quantitative Results

The duration and the time for execution to generate the outputs for various movie scenes are recorded in the table below are recorded in Table 6.1

Movie Name	Duration	Number of Frames	Execution Time
Harry Potter	03:31 min	37	06:36 min
Dune	03:46 min	21	05:54 min
The Chronicles of Narnia	01:05 min	22	04:42 min
The Dark Knight	04:10 min	80	11:12 min
Titanic	194 min	2200	6.4 hours

Table 6.1: Execution times for various clips of movies

6.3 Discussion

Few issues can be observed in the outputs. In some frames, the lip detection has not been accurate due to the speaker appearing in angles that are difficult to analyse. This results in the absence of a tail in the speech bubble.
Some dialogues may be too long and may overflow out of the speech bubble. There are some frames in which another person would be speaking but the shot is that of the face of another person. In this case, it would map the dialogue incorrectly to the person in the frame. But overall, we can observe the generated comics are visually appealing and preserve coherence of the story.

6.4 Chapter Conclusion

The results obtained from the testing of the video-to-comics solution demonstrate significant progress and functionality across various stages of the process.

In the subtitle generation phase, the Whisper model successfully transcribed speech to text, generating SRT files with accurate timestamps and content. Keyframe extraction

was efficient, aligning keyframes with subtitles seamlessly, as evidenced by the extracted frames from scenes in *Harry Potter* and *Dune*.

Page layout generation showcased the versatility of the system, with dynamically generated layouts for scenes from *Harry Potter* and *Dune*, featuring distinct panel shapes to enhance visual storytelling.

Dialogue bubble placement was largely successful, although some challenges were encountered with inaccurate lip detection in certain frames, resulting in omitted tails in speech bubbles. Additionally, issues arose with lengthy dialogues overflowing from bubbles and dialogue misalignment in frames with multiple characters.

Quantitative analysis revealed the system's efficiency in processing various movie scenes, with execution times recorded for each clip.

In discussions, it's noted that despite minor issues, the generated comics maintain visual appeal and narrative coherence. Challenges such as accurate lip detection and dialogue mapping in complex scenes warrant further refinement, but overall, the solution presents a promising approach to transforming videos into engaging comic strips.

Moving forward, addressing the identified issues and refining algorithms for improved accuracy will be crucial for enhancing the system's performance and usability, ultimately offering users a more seamless and satisfying experience in converting videos to comics.

Chapter 7

Conclusion and Future Scope

CineComic represents a pioneering effort to bridge the gap between the digital age's screen dominance and the timeless appeal of storytelling through literature. The endeavor has been motivated by the pressing need to rejuvenate interest in reading, especially among digitally immersed individuals, by presenting content in a format that aligns with contemporary technological inclinations.

The literature survey unveiled existing methods and their limitations, emphasizing the need for a more comprehensive and adaptable system. Recognizing this gap, the proposed CineComic system aims to revolutionize video-to-comic conversion by integrating automatic subtitle generation and employing deep neural networks for keyframe extraction. This innovative approach addresses the shortcomings identified in previous works, promising a more robust and versatile solution that seamlessly transforms video content into visually captivating comic narratives.

The proposed system's architecture, detailed in the design section, showcases a cohesive integration of modules, algorithms, and techniques. From input processing and keyframe extraction to stylization, multi-page layout, and balloon generation, each module contributes to a holistic and automated video-to-comic conversion experience. The system's adaptability to diverse video genres and content styles, along with its user-friendly interface, positions it as a potential game-changer in fostering a renewed passion for reading among digital natives.

Despite the ambitious scope and potential impact, the project acknowledges certain assumptions and challenges, such as video quality and legal considerations. These aspects underline the importance of ethical and legal adherence in the development and utilization of the CineComic system.

Several improvements can be made for future work. Different cartoonization effects can be applied to create diverse comic styles. A more optimal method to place panels in

a page can be created to generate truly random panel layouts. The GUI can be improved on to include features such as editing the comic to remove errors such as speech bubble placement, dialogues etc. Translating the dialogues to other languages can widen scope of usage for this application.

References

[1] D.-S. Ryu, S.-H. Park, J.-w. Lee, D.-H. Lee, and H.-G. Cho, "Cinetoon: A semi-automated system for rendering black/white comic books from video streams," in *2008 IEEE 8th International Conference on Computer and Information Technology Workshops*, 2008, pp. 336–341.

[2] M. Wang, R. Hong, X.-T. Yuan, S. Yan, and T.-S. Chua, "Movie2comics: Towards a lively video content presentation," *IEEE Transactions on Multimedia*, vol. 14, no. 3, pp. 858–870, 2012.

[3] G. Jing, Y. Hu, Y. Guo, Y. Yu, and W. Wang, "Content-aware video2comics with manga-style layout," *IEEE Transactions on Multimedia*, vol. 17, pp. 2122–2133, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:206752075

[4] W.-T. Chu, C.-H. Yu, and H.-H. Wang, "Optimized comics-based storytelling for temporal image sequences," *IEEE Transactions on Multimedia*, vol. 17, no. 2, pp. 201–215, 2015.

[5] M. Pundefinedśko, A. Svystun, P. Andruszkiewicz, P. Rokita, T. Trzciński, W. Skarbek, and Y.-D. Zhang, "Comixify: Transform video into comics," *Fundam. Inf.*, vol. 168, no. 2–4, p. 311–333, jan 2019. [Online]. Available: https://doi.org/10.3233/FI-2019-1834

[6] X. Yang, Z. Ma, L. Yu, Y. Cao, B. Yin, X. Wei, Q. Zhang, and R. W. H. Lau, "Automatic comic generation with stylistic multi-page layouts and emotion-driven text balloon generation," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 17, no. 2, may 2021. [Online]. Available: https://doi.org/10.1145/3440053

[7] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," 2022.