SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

B. TECH COMPUTER SCIENCE ENGINEERING

CSE 1014 FOUNDATIONS OF ARTIFICIAL INTELLIGENCE

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Video summarisation with deep reinforcement learning

Submitted by:-

Kunal Makwana 19BAI1011

Prakriti Singh 19BAI1139

Arjun Tandon 19BAI1146

Under the guidance of

Dr. A. Vijayalakshmi

Table of Contents

S. No.	Title	Page No.
1	Abstract	3
2	Introduction	4
3	Methodology Used	6
4	Implementation (Codes)	11
5	Result	17
6	Conclusion	19

Abstract

Video summaries provide condensed and succinct representations of the content of a video stream through a combination of still images, video segments, graphical representations and textual descriptors. Video summaries are considered as a function of the type of content they are derived from (object, event, perception or feature based) and the functionality offered to the user for their consumption (interactive or static, personalised or generic). It is argued that video summarisation would benefit from greater incorporation of external information, particularly user based information that is unobtrusively sourced, in order to overcome longstanding challenges such as the semantic gap and providing video summaries that have greater relevance to individual users.

Video summarisation techniques are considered within three broad categories: internal (analyse information sourced directly from the video stream), external (analyse information not sourced directly from the video stream) and hybrid (analyse a combination of internal and external information).

Introduction

Video Summarisation is basically the technique to make a short preview of a longer video. This technique is used in our daily lives and in our surroundings more often than we can imagine. Using q-learning technique, through this project we have hoped to achieve a short preview of our own test video via the website we have made.

Video Summarisation is used in popular video streaming websites like YouTube, Netflix, etc. In these websites, once you hover your mouse over the video icon, you are able to see a short preview of the actual video. It kind of is like a movie trailer giving you a summary of what the full video is really like. These websites use this technique to help the users decide whether the video they are thinking of watching is worth watching or not. Video streaming websites usually tend to show the more interesting and eye catching bits in these previews in order to attract the users and make them click the video icon to watch the full thing.

Our project is directed more towards a safety purpose usage of the application. We are focused on making the previews of day and hour long surveillance camera footage.

CCTV Camera footages are one of the longest real-time videos out there. The purpose of these cameras are to keep an eye on the surroundings and make sure no crimes occur. But sometimes, despite them being there, certain robberies and other crimes happen at times when no one is manning these camera. This is where our application comes in.

Take the scenario of a museum and a camera guarding a piece of diamond in the middle of the room. The diamond gets stolen somewhere during the evening when the museum is not empty but has a less number of visitors. In order to get the exact time and to spot out the robber, we need a shorter video of a day-long footage of the surveillance camera installed and focused at the diamond. Our application will be able to cut down the time spent in watching the useless, motion-less parts of the video and just give the summary of the parts with motion in it.

This is achieved by tools such as OpenCV, Python, Javascript, FFMPEG, CNN (convolutional neural network) and q-learning.

<u>Methodology Used</u>

The requirements for this project were mainly 4: NumPy, Flask, OpenCV and Imutils.
We converted the video to greyscale and divided the foreground and the background such that the foreground was white in colour and the background black. This was it easy to detect the moving frames and label them so that the final summarised video could only have the moving parts of the entire long video. This was achieved by OpenCV using CNN as one of its algorithms and techniques.

Following are the descriptions of the above mentioned four requirements in order to clear out why they are essential for our project and application:-

<u>NumPy</u>:- is the fundamental package for scientific Python computations. This is a Python library that provides a multi-dimensional array of objects, sub-objects (such as masked arrays and matrices), and a set of procedures in place to quickly work with arrays, including mathematics, logic and space manipulationWe can also use sorting, random simulation, I/O, basic linear algebra, basic statistical operations, discrete Fourier transforms, selecting and much more. It is basically a mix of C and Python where data in the form of numerals are treated as arrays for multidimensional functions and rearrangement operations. Numpy is one of the most used Python library due to its simplicity and functionality. A lot of essential libraries in Python also require it for scientific computing and these libraries also depend on NumPy arrays as its main input, and outputs. There is also a feature that makes it possible for software developers to perform basic and advanced mathematical and statistical functions in multi-dimensional arrays and matrices with a very small line of code. 'ndarray', or n-dimensional data array of the structure are the most important features of Numpy. These arrays are homogeneous, and all of the array elements must be of the same type.

Flask:- is a web framework, it is a module in Python that is used in developing web applications easily by the user. It has a small core which is easy to extend. It is basically a microframework that does not have an ORM which means "Object Relational Manager" or any other similar features. Flask was developed by Armin

Ronacher, who’s team was called Pocoo, which was made up of international Python enthusiasts. Flask is based on the Werkzeg WSGI toolkit and the Jinja2 template engine.Both are Pocoo projects. Flask is well known as micro-processor due to it being lightweight task efficient. It is to the point and only provides components that are essential for the completion of tasks making it very fast and efficient. It only provides the necessary components for web development, such as request handling, sessions, routing, and so on. The user can make use of the other functionalities such as data handling by writing a custom module or by using an extension. This approach avoids unnecessary code, which is not even being used, making the program memory efficient.

OpenCV:- It all started at Intel in 1999 by Gary Bradsky. It was not until 2000 when the first release of OpenCV was made public. Intel's Russian software OpenCV team was managed by Vadim Pisarevsky and Gary Bradsky. In 2005, OpenCV was used on Stanley, the 2005 DARPA Grand Challenge winning vehicle.
OpenCV from then has been developed and made to support a multitude of algorithms related to Machine Learning and is expanding everyday. OpenCV supports programming languages

such as C++, Python, Java, etc., and is available on multiple platforms such as Linux, Windows, OS X, Android, and iOS.
OpenCV-Python is the Python Application Interface for OpenCV, utilizing the best of the OpenCV C++ Application Interface and Python. OpenCV-Python is a library designed to solve computer vision problems on the fly.OpenCV-Python utilizes Numpy, which is a highly optimized library for numerical operations. It converts OpenCV array structures to and from Numpy arrays where Numpy Libraries are used to do various operations. This expands the usability by integrating with other libraries that use Numpy such as SciPy and Matplotlib.
Just like in our project, OpenCV is used widely for image and video analysis, like license plate reading, photo editing, facial recognition and detection,image to text recognition, etc.
Our project utilizes the OpenCV to optimize the working and image processing of each individual frame, so as to help label the frame. On successful operation, the images(or Frames) can be compared easily to notice any change(or Movement). On detecting movement via the library, the frames can be labeled differently and compiled later in the summarized video.

Imutils:- are a series of functions made available to us for our connivance in various processes involving image processing such as translation, skeletonization,rotation, resizing,translation, and also in displaying Matplotlib images easier while integrating with OpenCV. It also works well with both versions of Python, Python 2.7 and Python 3.

We use these functions to convert the frames to grayscale images for the ease of comparison between 2 consecutive frames.

Implementation

1. Basic.py (*code that makes our model*):-

```
import cv2
import os
import imutils
import numpy as np

#
def ImpTimestamp(timestamps,fps):
    # list of final output frame
    impTime = []
    for i in range(len(timestamps)-1):
      if (timestamps[i+1]-timestamps[i]- float(1/fps)) > 0.0001:
        impTime.append(timestamps[i])
        impTime.append(timestamps[i+1])

    return impTime

def FrameExtract(path,reso): #parameters are video file path and 
resolution
    vidObj = cv2.VideoCapture(path)
    fps = vidObj.get(cv2.CAP_PROP_FPS)
    g_frame=[]

    wi = reso
    hi = int(vidObj.get(cv2.CAP_PROP_FRAME_HEIGHT)/
vidObj.get(cv2.CAP_PROP_FRAME_WIDTH)*reso)
    success = True

    while success:

        success, image = vidObj.read()
        if (success == True):
          gray_img = cv2.resize(image, (wi,hi), interpolation = 
cv2.INTER_AREA)
          # Saves the frames with frame-count 
          g_frame.append(gray_img)
          image=0

    del(vidObj)

    return g_frame,fps,hi,wi

def impPt(frame,fps):
    frame_nos = []
    timestamps = []
    imp_frams = []
    fgbg = cv2.createBackgroundSubtractorKNN()
    kernel = np.ones((2, 2), np.uint8)
    kernel2 = np.ones((25, 25), np.uint8)
    for i in range(len(frame)):
        frameDelta = fgbg.apply(frame[i])
        thresh = cv2.threshold(frameDelta, 200, 255, 
cv2.THRESH_BINARY)[1]

        opening = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel)
        thresh = cv2.dilate(opening, None, iterations=4)
        closing = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, 
kernel2)

        cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, 
cv2.CHAIN_APPROX_SIMPLE)
        cnts = imutils.grab_contours(cnts)
        # Add timestamps to the foreground objects
        maxi = 0
        for c in cnts:
            if cv2.contourArea(c) > maxi:
                maxi = cv2.contourArea(c)
                cnt = c

            if maxi > 1000:
                text = float(i/fps)
                (x, y, w, h) = cv2.boundingRect(cnt)
        if maxi > 1000:
            (x, y, w, h) = cv2.boundingRect(cnt)
            cv2.putText(frame[i], '%.2f' % text, (x, h), 
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 
lineType=cv2.LINE_AA)
            frame_nos.append(i)
            imp_frams.append(frame[i])
            timestamps.append(float(text))
            
    return frame_nos,imp_frams,timestamps

#showing the difference between both graph in the webpage with 
graph drawn
def graph(inf, outf,fps):     #returns list- [input file size, 
output file size, in frames, out frames, in duration, out 
duration]
    
    graph_list = np.zeros(6)

    infs = os.path.getsize(inf)
    outfs = os.path.getsize(outf)
    graph_list[0] = ((infs >> 20))
    graph_list[1] = ((outfs >> 20))

    cap = cv2.VideoCapture(inf)
    frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    graph_list[2] = (frame_count)
    duration = frame_count/fps
    graph_list[4] = (duration)
    cap.release()

    
    cap = cv2.VideoCapture(outf)
    frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    graph_list[3] = (frame_count)
    duration = frame_count/fps
    graph_list[5] = (duration)
    cap.release()
    return graph_list.tolist()

def genImpVid(video_name, images, height, width, color, fps):
    writer = cv2.VideoWriter(video_name, 
cv2.VideoWriter_fourcc(*'mp4v'), fps, (width, height), color)
    for i in images:
        writer.write(i)

def main(vid_file):
    g_frames, fps, height, width = FrameExtract(vid_file,480)
    _, impFrams, timestamps = impPt(g_frames,fps)
    g_frames = 0
    print("this is working hahaha(●'◡'●)")
    genImpVid("static/video/output/
og.mp4",impFrams,height,width,True,fps)

    ImpTimestamp(timestamps,fps)

    os.system(
        "yes | ffmpeg -i static/video/output/og.mp4 -vcodec 
libx264 static/video/output/output.mp4")
    os.remove("static/video/output/og.mp4")

    return fps
```

2. <u>App.py</u> (*code to run the application*)**:-**

```
from flask import Flask, request, redirect, url_for, 
render_template, send_file
import os
from werkzeug.utils import secure_filename
from models import basic

UPLOAD_FOLDER = 'static/video/input/'

app = Flask(_name_)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
app.config['MAX_CONTENT_LENGTH'] = 50*1024*1024  # upload limit 
50MB
```

```python
app.config['TEMPLATES_AUTO_RELOAD'] = True


"""
    HELPERS
"""
def makedirs():
    root_path = os.path.abspath(os.path.dirname(_file_))
    static_dir = os.path.join(root_path, 'static')
    os.mkdir(os.path.join(static_dir, 'video'))
    video_dir = os.path.join(static_dir, 'video')
    folders = ['output', 'input']
    for folder in folders:
        os.mkdir(os.path.join(video_dir, folder))


"""
    ROUTES
"""
@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        if 'file' in request.files:
            file = request.files['file']
            if file.filename == '':
                return redirect(request.url)
            filename = secure_filename(file.filename)

            # create video, input & output folder in static dir
            if not os.path.exists(UPLOAD_FOLDER):
                makedirs()
            file.save(os.path.join(app.config['UPLOAD_FOLDER'], 
filename))

            # run the summarizer & get fps
            # Calls the main function in basic.py
            fps = basic.main(os.path.join(
                    app.config['UPLOAD_FOLDER'], filename))
            return redirect(url_for('processed', 
filename=filename, fps=fps))
    return render_template('index.html')


@app.route('/out')
def processed():
    filename = "video.mp4"
    fps = 30.011125336362067
    metrics = basic.graph("static/video/output/
output.mp4",os.path.join(app.config['UPLOAD_FOLDER'], filename), 
fps)
    print(filename)
    return render_template('output.html', metrics=metrics)


@app.route('/download')
def download():
    return send_file('static/video/output/output.mp4', 
as_attachment=True,
                                attachment_filename='processed-
video.mp4', cache_timeout=0)



if _name_ == '_main_':
    app.run(host='0.0.0.0',debug=True)
```

Result

We shot a 10 second long video of a moving hand against a blank background for testing purposes. The resultant video that was attained is a 3 second long summarised video with the time frames along with the moving object in the video.

Input:-

Output:-

Conclusion

Hence, from the above results we can conclude that the project is useful and does work on surveillance footages to cut short the time spent on watching them.

In the future, in order to make this application more advanced, we can use test videos with patterned backgrounds and multiple moving objects. This way by dividing and labelling the frames we shall be able to show a more real-time based result via our application.

*******Thank You*******