CSE-578 Computer Vision

Assignment 0
Niharika Vadlamudi
2018122008

January 25 2020

1 Video to Images

Code is written to extract the constituent frames of a video and write them into a specified folder. The video is opened using the `VideoCapture` module and the frames are read one-by-one using the `read` function. These frames are then written into the specified directory using the `imwrite` function.

1.1 Python Code

```
def getImages(videoPath,destinationPath):
    imgCount=0
    video=cv2.VideoCapture(videoPath)
    if(video.isOpened()== False):
        print("Unable to open video !")
        return
    else:
        print("File available!Loading the images!")
    while True:
        success, image = video.read()
        if success==0:
            break
        else:
            #Displaying the images !
            cv2.imshow('image',image)
            cv2.imwrite('{}/{}.png'.format(destinationPath,imgCount), image)
            imgCount+= 1
            # Stop if q key is pressed
            if cv2.waitKey(30) & 0xFF==ord('q'):
                break

    cv2.destroyAllWindows()
    video.release()
    print("Total frames/images extracted for given video are : ",end="")
    print(imgCount)
    return
```

1.2 Output Images

The video `spiderman.mp4` which was a GIF of 3 seconds was tested out using the above code , and it yielded **94** frames.Output images below .

(a) First Frame

(b) Last Frame

Figure 1: Output Images

Code is also written for merging a set of images from a specified folder into a single video. The folder path, desired frame rate, and desired video file name are taken as command line arguments.Then the files are read in sorted order and written into a video using the `VideoWriter` module, at the specified frame rate.

1.3 Python Code

```
def VideoCombination(destinationPath,framesRate):
    # By default it will be stored as .mp4 file format.
    path=destinationPath+'_'+str(framesRate)+'_'+'.mp4'
    fourcc = cv2.VideoWriter_fourcc(*'XVID')

    # sort the files in given directory
    image_list = sorted(os.listdir(destinationPath),key=lambda x: int(x.split('.')[0]))
    frame = cv2.imread(os.path.join(destinationPath,image_list[0]))
    size = (frame.shape[1],frame.shape[0])

    #Calling the VideoWriter Tool (OpenCV2)
    vid = cv2.VideoWriter(path,fourcc,framesRate,size)

    # will make a video from all frames in the given folder
    for filename in image_list:
        frame = cv2.imread(os.path.join(destinationPath,filename))
        vid.write(frame)

    vid.release()
    cv2.destroyAllWindows()
    return
```

The above function is tested for `captainAmerica.mp4` GIF, and 3 main output files were created with a low (0.1 fps),medium (20 fps) and high (100 fps) were generated . All the output files are present under the folder - $\texttt{Output}_{Avengers}$.

2 Capturing Images

Code is written to capture frames from the webcam (video device id 0) and write them into a specified folder. This is done using the VideoCapture module and imwrite function. The frames are captured until the 'q' key is pressed and the frames are also shown on screen using the imshow function.We mainly reuse

the code written for the first part , with the input video path as device(0) and additional entity will be frames per second part .

```
def getWebcamImages(destinationPath,webCam=1):
    imgCount=0
    if(webCam==0):
        print("Webcam device Not Activated ! ")
        return
    else:
        #For Webcam Stuff .
        video = cv2.VideoCapture(0)
    while True:
        success, image = video.read()
        if success==0:
            break
        else:
            #Displaying the images !
            cv2.imshow('image',image)
            cv2.imwrite('{}/{}.png'.format(destinationPath,imgCount), image)
            imgCount+= 1
            # Stop if q key is pressed
            if cv2.waitKey(30) & 0xFF==ord('q'):
                break
    cv2.destroyAllWindows()
    video.release()
    return
```

2.1 Output Images

These are output based images for Web Cam based video input, about 145 images were collected with the default frames per second .

(a) Webcam First Frame

(b) Webcam Last Frame

Figure 2: Webcam based Images

3 Chroma Keying

A basic Chroma Keying technique based on the constant background color assumption is implemented. Two video sequences are taken as input and a composite video sequence is created by replacing the constant green back color in the foreground video sequence with the colors from the target background sequence. This is done by first creating two binary masks, based on whether the corresponding pixels in the foreground image are green or not. These masks are then applied to the foreground and background

images respectively to cut out the foreground object from the foreground image and the corresponding pixels in the same position from the background image. These two masked-out images are then added together to form the final composite image.This technique is used heavily in film making ,an example would be VFX effects.

3.1 Code

```
def chromaKeying(foregroundvidPath,backgroundvidPath,finalPath,threshold=128,framesRate=100):
    destinationPathf=currdir+"\\"+"fore"
    destinationPathb=currdir+"\\"+"back"
    #Make each of the videos to set of Images.
    getImages(foregroundvidPath,destinationPathf,webCam=0)
    getImages(backgroundvidPath,destinationPathb,webCam=0)


    # Minimum Number of frames .
    count=min(len(foreList),len(backList))


    #Run a loop through these images and sort them.
    backList=sorted([int(temp.split('.')[0]) for temp in os.listdir(destinationPathf.split('.')[0])])
    foreList=sorted([int(temp.split('.')[0]) for temp in os.listdir(destinationPathb.split('.')[0])])


    #Main running loop.
    for i in range(count):
        fg = cv2.imread('{}/{}.png'.format(destinationPathf.split('.')[0], i))
        bg = cv2.imread('{}/{}.png'.format(destinationPathb.split('.')[0], i))
        #Resizing based on foreground image .
        h,w,c = fg.shape
        bg = cv2.resize(bg,(fg.shape[1], fg.shape[0]))
        # Extracting the Green Mask around the foreground image .
        greenMask=fg[:, :, 1]
        imgMask = (fg[:, :, 1] < threshold) | (np.amax(fg, axis=2) != green_mask)
        # Inversion.
        imgMask =1 - imgMask
        m_fg = np.copy(fg)
        m_fg[img_mask != 0] = [0,0,0]
        m_bg = np.copy(bg)
        m_bg[img_mask == 0] = [0,0,0]
        # Resultant Image .
        res_img = m_fg+m_bg
        cv2.imwrite('{}/{}.png'.format(out_path,i),res_img)
    VideoCombination(finalPath,framesRate)
    print("Chroma-Keying Done")
    return
```

3.2 Output Video

* Attached Video Link The images taken are :

(a) Background Frame

(b) Foreground Frame

Figure 3: Chroma Keying Images

4 Challenges Faced

The main difficulty was in perfectly masking out the foreground object from the background since even the foreground object has some shades of green on it. I tried to finely determine the range of green values to mask out, by writing a small interactive program with trackbars to control the range of RGB values to mask out . At the end , I gave the input range for Green with certain increased range , the results improved /

5 Learnings

I got familiar with the basic API of OpenCV2 to load, read, and write images and videos, and with manipulating their pixel values. Reading up the OpenCV2 documentation was extremely informative . I understood the Chroma Keying problem and its difficulty, and the extra constraints we need to introduce to reasonably solve the problem. I implemented and realized the weaknesses of a technique with constant back color assumption and tried out some approaches to solving them.