Computer Vision

Assignment 0 : Report

Tejas Srivastava 2021102017

Note : All the code, data and result are uploaded to the following link :- assignment0

Part1 :

Video ↔ Images: Write a program to convert a given video to its constituent images. Your output should be in a specified folder. Write another program that will merge a set of images in a folder into a single video. You should be able to control the frame rate in the video that is created.

Solution :

To achieve this, we had to explore the VideoCapture function in the opencv library. We use this function to create a video object given a video file path. Next, we try to extract relevant information about the video such as frame rate and length. In order to get the frame at some specified time-stamp, we calculate the frame id at that time-stamp as follows:

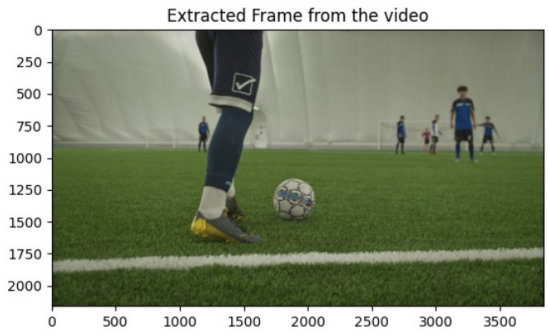
Suppose we want to find out the frame number at the time-stamp given as below

```
# given a time-stamp, calculating the frame id
minutes = 0
seconds = 0.59
frame_id = int(fps*(minutes*60 + seconds))
print('frame id =',frame_id)


# getting the frame from the video given frame-id
video.set(cv2.CAP_PROP_POS_FRAMES, frame_id)
ret, frame = video.read()

# displaying the frame
frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
```

```
frames per second :  25.0
frame id = 14
```



Now we can set the video to the frame using the frame id and the set function. After the video is set to the desired frame, we can *read()* function to obtain the desired frame.

Now, that we know how to extract the frames from the video. We can decompose a video into its constituent frames and vice versa. Following is the summary of the algorithm used.

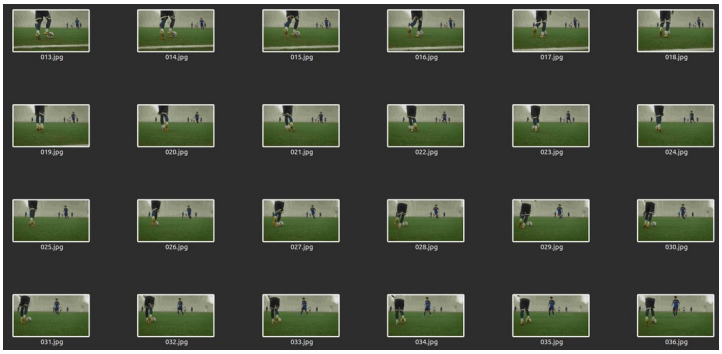
1.) The function *convertVideoToFrames()* takes a video file and a frame rate as input and extracts frames from the video at the specified frame rate. It uses OpenCV to read the video and extract frames. Here's a summary of the method:

- **Quality Configuration:** The code sets up parameters for image compression, controlling the quality of the saved frames using JPEG compression.
- **Video Information Retrieval**: It retrieves information about the video, such as frames per second (fps), total frame count, and duration.

- **Frame Extraction**: It calculates the total number of frames to be extracted based on the specified frame rate. Then, it iteratively reads frames from the video, saving them as individual JPEG images.
- **File Naming:** Frames are named sequentially with leading zeros to maintain a consistent naming format, making it easier to sort and organize and read for the following function.

2.) The *convertFramesToVideo()* function takes a folder containing image frames and a frame rate as input, then combines these frames into a video. Here's a summary of the method:

- **Image File Retrieval**: It retrieves a list of image files in the specified folder, ensuring they have the ".jpg" extension. The list is sorted for sequential processing.
- **Frame Dimensions**: It reads the first image to determine the height and width of the frames.
- **Video Creation**: It creates a VideoWriter object using the specified output path, codec ('mp4v'), frame rate, and frame dimensions.
- **Frame Writing**: It iterates through the sorted list of image files, reads each image, and writes it to the video.
- **Video Release**: After writing all frames, it releases the VideoWriter object.



The first 36 of the total frames extracted

Challenges and their solutions:

1. **Lexicographical naming** : While converting video into its frames and saving in a folder, we need to make sure that the frames are named lexicographically, so that when our function reads the images in the folder to convert them into video, it processes and adds them in the right order. Thus, the order of reading files in the **os** library should be same as the order in which we are naming out frames. This resulted in making length of the filenames equal, by adding leading zeros to frame numbers.

2. **Lossy Compression :** The original video was quite big in size. Also, the frames captured were in high quality png files by default. For the sake of the experiment, we had to lossy compress the individual frames before composing the video, so that the resultant video file is not very large in size.

3. **Bounds on FPS of decompose and compose function :** When de-composing the video into frames, we are giving the user the independence of choosing the fps of extraction. Also, while composing the video from the individual frames, we provide this parameter. However, we must be carefull that the fps of the composed video must not exceed the fps parameter in the extraction, so that the playback speed of the video composed is not changed. *Ideally,*

both the frame-rate parameters should be equal. Also, we must ensure that the frame-rate demanded in the decomposition function must not be greater than the fps of the video.

Learnings:

- How to obtain the fps and length of a video.
- How to extract running frames from a video
- How to lossy compress images before saving
- How to compose a set of frames into a video.

Part2 :

Capturing Images: Learn how to capture frames from a webcam connected to your computer and save them as images in a folder. You may use either the built-in camera of your laptop or an external one connected through USB. You should also be able to display the frames (the video) on the screen while capturing.

Solution :

We use the same *VideoCapture* function with the argument as 0 (for default webcam) to create an object of the video buffer-in from the webcam. Now, we can just put up a while-true loop to capture frames at each instant and display them on a window. We also added functionality to exit and save a frame in between using the KeyInterrupt **'s'** and **'q'** for *save* and *quit* respectively.

Code:

```
import cv2

key = cv2. waitKey(1)
webcam = cv2.VideoCapture(0)
while True:
    try:
        check, frame = webcam.read()
        cv2.imshow("Frame captured", frame)
        key = cv2.waitKey(1)

        if key == ord('q'):
            print("Quitting...")
            webcam.release()
            cv2.destroyAllWindows()
            break
        elif  key== ord('s'):
            cv2.imwrite('./results/webcam_saved_images/capture.jpg',frame)
            print("Successfully saved image to ./results/webcam_saved_images/capture.jpg")

    except(KeyboardInterrupt):
        print("quitting...")
        webcam.release()
        cv2.destroyAllWindows()
        break
```

Image Saved Image through Webcam

Challenges and their solutions:

1. **waitKey:** Using cv2.waitKey(), we were able to get promtps from the user and perform operations on the current frame of the webcam like saving the image, performing segmentation etc..

Learnings :

- How to capture streaming video frames from a video source (webcam here)
- How to utilize the waitKey function of cv2 and add interactibility to the system.
- How to perform image processing operations on each of the frames individually

---

Part3 :

Chroma Keying: Create an interesting composite of two videos using this technique, possibly with one video including yourselves

Solution :

In order to perform Chroma Keying, we first need 2 videos. One containing a green screen and another video to be overlayed on the first video. Lets call them video1 and video2 respectively. We create VideoWriter object for each of these 2 videos (called *video* and *background*). We read the video frames one by one and display them.

For each frame, we first convert the BGR image to HSV format, since thresholding in HSV provides better results for image segmentation. We also need to ensure that frames from both the video sources are of equal size. To impose this constraint, we resize both the frames to a common shape. Now, we define two pixel color values namely lower_green and upper_green, to be used for thresholding. We for now find these values using hit and trial, but the results can be optimized by creating a UI for selecting the thresholds while observing the segmented frames in real time. For now, we stick to the hit-trial values obtained (since the results are pretty optimal)

```
ret, frame = video.read()
ret_bg, bg = background.read()
frame = cv2.resize(frame,(700,400))
bg = cv2.resize(bg,(700,400))
hsv = cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)

# thresholds in hsv (found out using hit and trial)
l_green = np.array([30,88,134])
u_green = np.array([181,255,255])
```

Now, we create a mask for the background of the frame from video1. We find the masked frame using *bitwise_and()* on the original frame. Now, we subtract the masked frame from the original frame to get the foreground object. We can now impose the result on the video2 using *np.where()*.

```
# creating a mask for the segmentationof the frame
mask = cv2.inRange(hsv,l_green,u_green)
res = cv2.bitwise_and(frame,frame,mask=mask)
subtracted = frame-res
green_screen = np.where(subtracted==0,bg,subtracted)
```

Finally, we just display the resultant frame on the window and save it in the VideoWriter object.

```
# saving the frames to a video file
video_writer.write(green_screen)
```

The user can press **‘q’** to quit anytime while the video is running, althought it will interrupt the saving process leading to incomplete saving.

Code :

```
output_video_path = './results/video/chroma_keyed_output.mp4'
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
video_writer = cv2.VideoWriter(output_video_path, fourcc, 30, (700, 400))


while True:
    ret, frame = video.read()
    ret_bg, bg = background.read()
    frame = cv2.resize(frame,(700,400))
    bg = cv2.resize(bg,(700,400))
    hsv = cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)

    # thresholds in hsv (found out using hit and trial)
    l_green = np.array([30,88,134])
    u_green = np.array([181,255,255])

    # creating a mask for the segmentationof the frame
    mask = cv2.inRange(hsv,l_green,u_green)
    res = cv2.bitwise_and(frame,frame,mask=mask)
    subtracted = frame-res
    green_screen = np.where(subtracted==0,bg,subtracted)

    # displaying the frames
    cv2.imshow("Frame",frame)
    cv2.imshow("Final",green_screen)


    # saving the frames to a video file
    video_writer.write(green_screen)

    # waitkey should have fps of the video (here 30)
    key = cv2.waitKey(30)
    if key == ord('q'):
        video.release()
        background.release()
        video_writer.release()
        cv2.destroyAllWindows()
        break

video.release()
background.release()
video_writer.release()
cv2.destroyAllWindows()
```

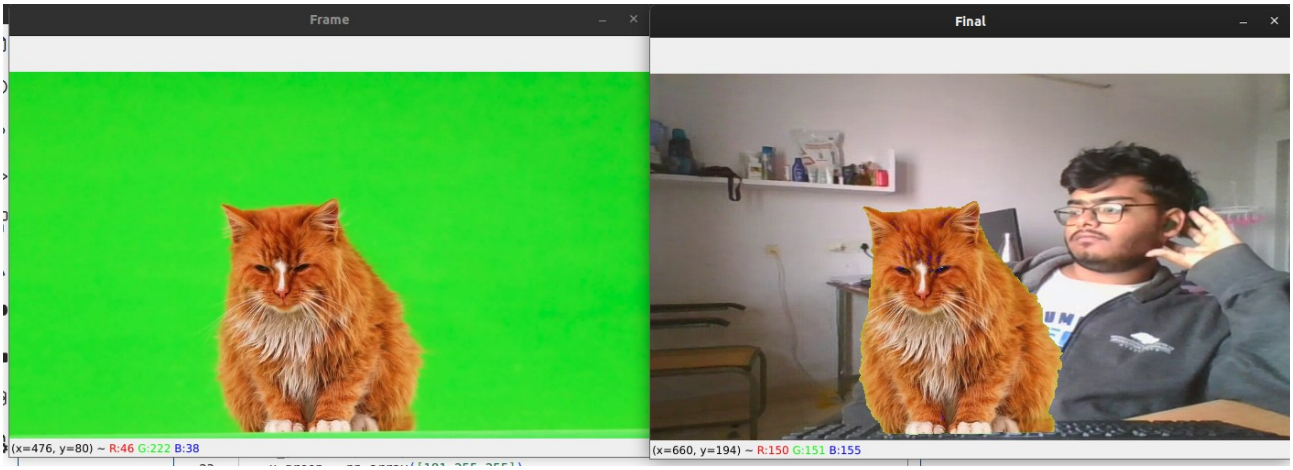
Challenges and its Solutions :

1. **HSV colorspace :** Performing thresholding for creating background mask was quite suboptimal in the RGB colorspace (BGR in cv2). After a little research, it was found that since the hue (H) channel in HSV colorspace models the color type, segmentation in HSV space gave much better results than other colorspaces. Thus the video frames needed to be converted to HSV space before processing.

2. **Finding Out thresholds :** Evaluating optimal thresholds for segmenting the green-screen was a challenging part of the question. The value was found out using hit-and-trial method after several iterations.

Learning:

- Thresholding using *cv2.inRange()*
- Masking using *cv2.bitwise_and()*
- Combining foreground and background using *np.where()* for each frame



Combining the foreground and background in videos.
