02-video_processing

September 26, 2021

1 Video Processing

In this lesson we will process video data using the OpenCV Python library.

1.1 Obtain a Video

Let's start by uploading the smallest version of this video to the Colab. Rename the video to `cars.mp4` or change the name of the video in the code below.

1.2 Reading the Video

OpenCV is an open source library for performing computer vision tasks. One of these tasks is reading and writing video frames. To read the `cars.mp4` video file, we use the VideoCapture class.

[ ]:
```
import cv2 as cv

cars_video = cv.VideoCapture('cars.mp4')
```

Once you have created a `VideoCapture` object, you can obtain information about the video that you are processing.

[ ]:
```
height = int(cars_video.get(cv.CAP_PROP_FRAME_HEIGHT))
width = int(cars_video.get(cv.CAP_PROP_FRAME_WIDTH))
fps = cars_video.get(cv.CAP_PROP_FPS)
total_frames = int(cars_video.get(cv.CAP_PROP_FRAME_COUNT))
```

```
print(f'height: {height}')
print(f'width: {width}')
print(f'frames per second: {fps}')
print(f'total frames: {total_frames}')
print(f'video length (seconds): {total_frames / fps}')
```

```
height: 360
width: 640
frames per second: 25.0
total frames: 1501
video length (seconds): 60.04
```

When you are done processing a video file, it is a good idea to release the VideoCapture to free up memory in your program.

[ ]:
```
cars_video.release()
```

We can now loop through the video frame by frame. To do this we need to know the total number of frames in the video. For each frame we set the current frame position and then read that frame. This causes the frame to be loaded from disk into memory. This is done because videos can be enormous in size, so we don't necessarily want the entire thing in memory.

You might also notice that we read the frame from the car's video, and then we check the return value to make sure that the read was successful. This is because the underlying video processing library is written in the C++ programming language, and a common practice in that language is to return a status code indicating if a function succeeds or not. This isn't very idiomatic in Python; it is just the underlying library's style leaking through into the Python wrapper.

[ ]:
```
cars_video = cv.VideoCapture('cars.mp4')

total_frames = int(cars_video.get(cv.CAP_PROP_FRAME_COUNT))

frames_read = 0

for current_frame in range(0, total_frames):
  cars_video.set(cv.CAP_PROP_POS_FRAMES, current_frame)
  ret, _ = cars_video.read()
  if not ret:
    raise Exception(f'Problem reading frame {current_frame} from video')
  if (current_frame+1) % 50 == 0:
    print(f'Read {current_frame+1} frames so far')

cars_video.release()

print(f'Read {total_frames} frames')
```

```
Read 50 frames so far
Read 100 frames so far
Read 150 frames so far
Read 200 frames so far
Read 250 frames so far
Read 300 frames so far
Read 350 frames so far
Read 400 frames so far
Read 450 frames so far
Read 500 frames so far
Read 550 frames so far
Read 600 frames so far
Read 650 frames so far
Read 700 frames so far
Read 750 frames so far
Read 800 frames so far
Read 850 frames so far
Read 900 frames so far
Read 950 frames so far
Read 1000 frames so far
Read 1050 frames so far
Read 1100 frames so far
Read 1150 frames so far
Read 1200 frames so far
Read 1250 frames so far
Read 1300 frames so far
Read 1350 frames so far
Read 1400 frames so far
Read 1450 frames so far
Read 1500 frames so far
Read 1501 frames
```

That code took a while to execute. The video is just over a minute long, and it takes a while to iterate over every frame. Consider the amount of time it would take to perform object recognition on each frame.

In practice you will be doing this kind of processing on a much bigger machine, or machines, than Colab provides for free. You can also process many frames in parallel.

For our purposes, let's just make the video shorter.

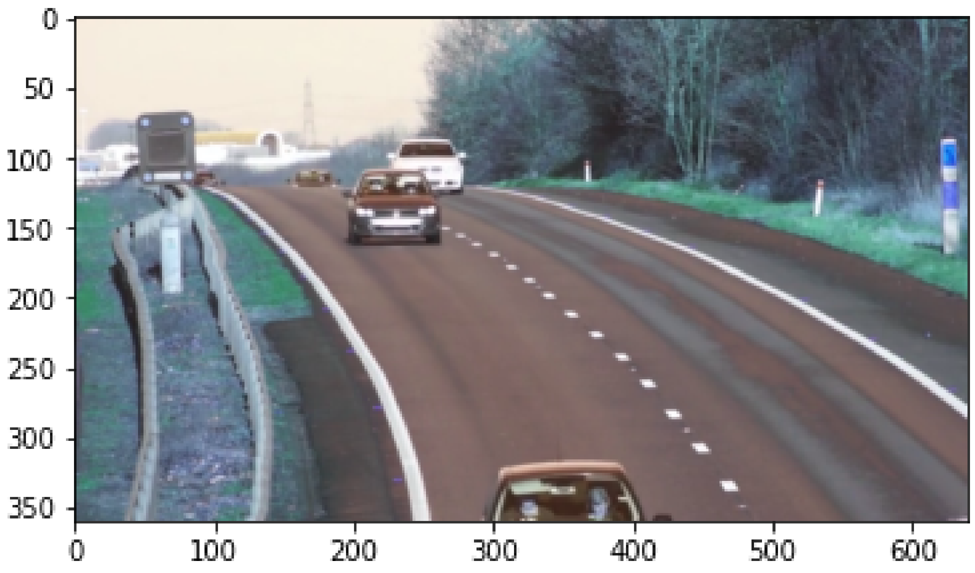
We'll load the video one more time, and then we'll read out a single frame to illustrate that the frame is just an image.

[ ]:
```python
import matplotlib.pyplot as plt

cars_video = cv.VideoCapture('cars.mp4')
cars_video.set(cv.CAP_PROP_POS_FRAMES, 123)
ret, frame = cars_video.read()
if not ret:
  raise Exception(f'Problem reading frame {current_frame} from video')

cars_video.release()
```

```
plt.imshow(frame)
```

[ ]:

```
<matplotlib.image.AxesImage at 0x7ffce9b4d290>
```



1.3 Writing a Video

OpenCV also supports writing video data. Let's loop through the long video that we have and save only one second of it into a new file.

First we need to open our input video and get information about the frame rate, height, and width.

[ ]:

```
input_video = cv.VideoCapture('cars.mp4')

height = int(input_video.get(cv.CAP_PROP_FRAME_HEIGHT))
width = int(input_video.get(cv.CAP_PROP_FRAME_WIDTH))
fps = input_video.get(cv.CAP_PROP_FPS)
```

Using that information we can create a VideoWriter that we'll use to write the shorter video.

Video can be encoded using many different formats. In order to tell OpenCV which format to use, we choose a "four character code" from fourcc. In this case we use "mp4v" to keep our input and output files consistent.

[ ]:

```
fourcc = cv.VideoWriter_fourcc(*'mp4v')
output_video = cv.VideoWriter('cars-short.mp4', fourcc, fps, (width, height))
```

Now we can loop through one second of video frames and write each frame to our output video.

[ ]:
```
for i in range(0, int(fps)):
  input_video.set(cv.CAP_PROP_POS_FRAMES, i)
  ret, frame = input_video.read()
  if not ret:
    raise Exception("Problem reading frame", i, " from video")
  output_video.write(frame)
```

Once processing is complete, be sure to release the video objects from memory.

[ ]:
```
input_video.release()
output_video.release()
```

And now we can list the directory to see if our new file was created.

[ ]:
```
import os

os.listdir('./')
```

[ ]:
```
['slides.md',
 '00-pil-key.zip',
 '.DS_Store',
 '00-pil.ipynb',
 'slides.pptx',
 '01-open_cv.ipynb',
 'running-shoe-371624_1920.jpg',
 '02-video_processing-key.zip',
 '.github',
 '.ipynb_checkpoints',
 'shiba-6292660_1920.jpg',
 'cars-short.mp4',
 'cars.mp4',
 '.git',
 '02-video_processing.ipynb',
 'car-49278_640.jpg',
 '01-open_cv-key.zip']
```

You should now see a `cars-short.mp4` file in your file browser in Colab. Download and view the video to make sure that it only lasts for a second.

Notice we have only concerned ourselves with the visual portion of the video. Videos contain both visual and auditory elements. OpenCV is only concerned with computer vision, so it doesn't handle audio processing.

2 Exercises

2.1 Exercise 1

Above we shortened our video to 1 second by simply grabbing the first second of frames from the video file. Since not much typically changes from frame to frame within a second of video, a better video processing technique is to sample frames throughout the entire video and skip some frames.

For example, it might be more beneficial to process every 10th frame or only process 1 of the frames in every second of video.

In this exercise, take the original cars video used in this Colab and reduce it to a short 25-fps (frames per second) video by grabbing the first frame of every second of video. Save the video as `cars-sampled.mp4`.

2.1.1 Student Solution

[ ]:
```
input_video = cv.VideoCapture('cars.mp4')

height = int(input_video.get(cv.CAP_PROP_FRAME_HEIGHT))
width = int(input_video.get(cv.CAP_PROP_FRAME_WIDTH))
fps = input_video.get(cv.CAP_PROP_FPS)

fourcc = cv.VideoWriter_fourcc(*'mp4v')
output_video = cv.VideoWriter('cars-sampled.mp4', fourcc, fps, (width, height))

total_frames = int(input_video.get(cv.CAP_PROP_FRAME_COUNT))

for i in range(0, total_frames, 25):
  input_video.set(cv.CAP_PROP_POS_FRAMES, i)
  ret, frame = input_video.read()
  if not ret:
    raise Exception("Problem reading frame", i, " from video")
  output_video.write(frame)

input_video.release()
output_video.release()
```