Tutorial on Video Modeling
Decord: an efficient video reader for deep learning

Yi Zhu and Zhi Zhang
06/14/2020
Motivation

- Videos have redundant frames, need video reader

Videos $\longrightarrow$ Raw frames $\longrightarrow$ Network training
Motivation

- Videos have redundant frames, need video reader

Videos $\rightarrow$ Raw frames $\rightarrow$ Network training

- Pre-processing takes time
- Data storage is huge
- IO bottleneck during training

Videos $\rightarrow$ Network training
Motivation

- Slowness in random access

```python
import numpy as np
import cv2

cap = cv2.VideoCapture(0)

while(True):
    # Capture frame-by-frame
    ret, frame = cap.read()

    # Our operations on the frame come here
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Display the resulting frame
    cv2.imshow('frame', gray)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# When everything done, release the capture
cap.release()
```
Motivation

- Slowness in random access

Segment 1: index 9
Segment 2: index 51
Segment 3: index 102

Random access > sequential read

Wang et al., Temporal Segment Networks: Towards Good Practices for Deep Action Recognition, ECCV 2016
Motivation

- Lack of flexibility or good user experience in terms of video handling

OpenCV

```python
capture.set(cv2.CAP_PROP_POS_FRAMES, 100)
print('Position:', int(capture.get(cv2.CAP_PROP_POS_FRAMES)))
_, frame = capture.read()
cv2.imshow('frame100', frame)
```

Decord

```python
frame = vr[99]
```
Decord: provide smooth experiences similar to random image loader for deep learning.
Installation

```
pip install decord
```

Supports Windows/Mac/Linux

Need to build from source to enable GPU support
Ease of Usage
Pythonic interface

Easy to get video duration

Direct access to any frames by list indexing
from decord import VideoReader
from decord import cpu, gpu

vr = VideoReader('examples/flipping_a_pancake.mkv', ctx=cpu(0))
print('video frames:', len(vr))
# 1. the simplest way is to directly access frames
for i in range(len(vr)):
    # the video reader will handle seeking and skipping in the most efficient manner
    frame = vr[i]
    print(frame.shape)

# To get multiple frames at once, use get_batch
# this is the efficient way to obtain a long list of frames
frames = vr.get_batch([1, 3, 5, 7, 9])
print(frames.shape)
# (5, 240, 320, 3)
# duplicate frame indices will be accepted and handled internally to avoid duplicate decoding
frames2 = vr.get_batch([1, 2, 3, 2, 3, 4, 3, 4, 5]).asnumpy()
print(frames2.shape)
# (9, 240, 320, 3)
from decord import VideoReader
from decord import cpu, gpu

vr = VideoReader('examples/flipping_a_pancake.mkv', ctx=cpu(0))
print('video frames:', len(vr))
# 1. the simplest way is to directly access frames
for i in range(len(vr)):
    # the video reader will handle seeking and skipping in the most efficient manner
    frame = vr[i]
    print(frame.shape)

# To get multiple frames at once, use get_batch
# this is the efficient way to obtain a long list of frames
frames = vr.get_batch([1, 3, 5, 7, 9])
print(frames.shape)
# (5, 240, 320, 3)
# duplicate frame indices will be accepted and handled internally to avoid duplicate decoding
frames2 = vr.get_batch([1, 2, 3, 2, 3, 4, 3, 4, 5]).asnumpy()
print(frames2.shape)
# (9, 240, 320, 3)
3D CNNs, loading clips instead of frames

Segment1: index [1,2,3,4,5,6,7,8,9,10,11,12]
Segment2: index [5,6,7,8,9,10,11,12,13,14,15,16,17]
Segment3: index [9,10,11,12,13,14,15,16,17,18,19,20,21]

Duplication! (OpenCV -> slow, Lintel -> X)
Batch read frames

Efficient handling of duplication

```python
from decord import VideoReader
from decord import cpu, gpu

vr = VideoReader('examples/flipping_a_pancake.mkv', ctx=cpu(0))
print('video frames:', len(vr))
# 1. the simplest way is to directly access frames
for i in range(len(vr)):
    # the video reader will handle seeking and skipping in the most efficient manner
    frame = vr[i]
    print(frame.shape)

# To get multiple frames at once, use get_batch
# this is the efficient way to obtain a long list of frames
frames = vr.get_batch([1, 3, 5, 7, 9])
print(frames.shape)
# (5, 240, 320, 3)
# duplicate frame indices will be accepted and handled internally to avoid duplicate decoding
frames2 = vr.get_batch([1, 2, 3, 2, 3, 4, 3, 4, 5]).asnumpy()
print(frames2.shape)
# (9, 240, 320, 3)
```
from decord import VideoReader
from decord import cpu, gpu

vr = VideoReader('examples/flipping_a_pancake.mkv', ctx=cpu(0))
print('video frames:', len(vr))
# 1. the simplest way is to directly access frames
for i in range(len(vr)):
    # the video reader will handle seeking and skipping in the most efficient manner
    frame = vr[i]
    print(frame.shape)

# To get multiple frames at once, use get_batch
# this is the efficient way to obtain a long list of frames
frames = vr.get_batch([1, 3, 5, 7, 9])
print(frames.shape)
# (5, 240, 320, 3)
# duplicate frame indices will be accepted and handled internally to avoid duplicate decoding
frames2 = vr.get_batch([1, 2, 3, 2, 3, 4, 3, 4, 5]).asnumpy()
print(frames2.shape)
# (9, 240, 320, 3)

# 2. you can do cv2 style reading as well
# skip 100 frames
vr.skip_frames(100)
# seek to start
vr.seek(0)
batch = vr.next()
print('frame shape:', batch.shape)
print('numpy frames:', batch.asnumpy())
Resize videos while video reading

```
vr = de.VideoReader(video, width=640, height=480)
print('Frame shape:', vr[0].shape)

Frame shape: (480, 640, 3)
```
Batch reading using `range`, reduce python overhead

```python
frame_id_list = range(0, 64, 2)
frames = vr.get_batch(frame_id_list).asnumpy()
print(faces.shape)
```

Out:

```
(32, 256, 320, 3)
```
Get all the key frames

```python
key_indices = vr.get_key_indices()
key_frames = vr.get_batch(key_indices)
print(key_frames.shape)
```

Out:

```
(1, 256, 320, 3)
```
import decord
vr = decord.VideoReader('examples/flipping_a_pancake.mkv')
print('native output:', type(vr[0]), vr[0].shape)
# native output: <class 'decord.ndarray.NDArray'>, (240, 426, 3)
# you only need to set the output type once
decord.bridge.set_bridge('mxnet')
print(type(vr[0], vr[0].shape))
# <class 'mxnet.ndarray.ndarray.NDArray'> (240, 426, 3)
# or pytorch and tensorflow(>=2.2.0)
decord.bridge.set_bridge('torch')
decord.bridge.set_bridge('tensorflow')
# or back to decord native format
decord.bridge.set_bridge('native')
Efficiency Comparison
import cv2
import time
import numpy as np

frames_list = np.arange(duration)
np.random.shuffle(frames_list)

# Decord
for i in range(11):
    if i == 1:
        start_time = time.time()
        decode_vr = VideoReader(video_fname)
        frames = decode_vr.get_batch(frames_list)
        end_time = time.time()
    print('Decord takes %4.4f seconds.' % ((end_time - start_time)/10))

# OpenCV
for i in range(11):
    if i == 1:
        start_time = time.time()
        cv2_vr = cv2.VideoCapture(video_fname)
        for frame_idx in frames_list:
            cv2_vr.set(1, frame_idx)
            _, frame = cv2_vr.read()
        cv2_vr.release()
        end_time = time.time()
    print('OpenCV takes %4.4f seconds.' % ((end_time - start_time)/10))

Out:
Decord takes 4.4514 seconds.
OpenCV takes 7.6329 seconds.
Speed Comparison

<table>
<thead>
<tr>
<th>Operation</th>
<th>OpenCV(cv2)</th>
<th>pyAV(+PIMS)</th>
<th>Discord</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential Read</td>
<td>1</td>
<td>0.63</td>
<td>2.38</td>
</tr>
<tr>
<td>Fast Random Seek</td>
<td>1</td>
<td>6.7</td>
<td>8.67</td>
</tr>
<tr>
<td>Accurate Random Seek</td>
<td>1</td>
<td>2.8</td>
<td>14.48</td>
</tr>
<tr>
<td>Real-world Training Loading Speed</td>
<td>1</td>
<td>3</td>
<td>6.00</td>
</tr>
</tbody>
</table>
Comparison to other video readers

- OpenCV and PyAV
  Slow in random access pattern

- Lintel ([https://github.com/dukebw/lintel](https://github.com/dukebw/lintel))
  can’t handle duplication, no key_frame handling

- DALI ([https://github.com/NVIDIA/DALI](https://github.com/NVIDIA/DALI))
  complicated pipeline and usage, has to use Nvidia GPU

We will provide interface to other video readers!
Previewed Features

- GPU decoding and data augmentation complete, but needs further optimization
- Video Loader
  an all-in-one solution
  [https://github.com/dmlc/decord](https://github.com/dmlc/decord)
Conclusion

- Easy to use and flexible
  
  *pythonic*

- Efficient

- Notebook
  
  *(https://github.com/dmlc/decord/blob/master/examples/video_reader.ipynb)*

- Please try Decord at *https://github.com/dmlc/decord*