Battle-hardened advice on efficient data loading for deep learning on videos.

Valentin Haenel

V.P. Engineering
http://twentybn.com

8. Jul 2018

Version: PyDataBerlin2018.02 https://github.com/TwentyBN/20bn-video-data-loading-talk

Outline

whoami

- Valentin Haenel
- V.P. Engineering @ TwentyBN
- *We teach machines to perceive the world like humans*

Quick Survey

- Who does deep learning?
- Who does deep learning on videos?
- Who would like to do deep learning on videos?

The Problem

The Starving GPUs Problem

Getting GPUs fully utilized when training neural networks on large-scale video data can be challanging.

The Reason(s)

- PCIe Bus
- Storage medium
- Data storage format and loader implementation

In this talk, I'll present what we learnt at TwentyBN over the last 2 years.

PCIe Bus

- Peripheral Component Interconnect Express
- Connects CPU and GPU
- Bandwidth (host-to-device) 16 GB/s (theoretical) 11 GB/s (measured)
- Not much we can do about it, except to throw money at the problem with NVLink
- DGX-1 etc... but we might not have access to fancy hardware

If you are building a desktop rig...

- If you are adding two GPUs:
- make sure that your CPU supports 40 PCIe lanes (not 28)
- Otherwise you can't have full bandwidth to both GPUs
- use the `bandwidthTest` if in doubt

Storage Medium

- This only applies if the data doesn't fit into memory (or filesystem cache)
- Various technologies: magnetic disks and solid state disks
- Various connectors SATA III / PCIe | U.2 | M.2
- Bandwidth (one-way): 600 MB/s | 4 GB/s
- Parallelization with RAID
- Again: throw money at the problem

What should I buy?

- Get an SSD, the bigger the better
- Connect via PCIe or M.2 or U.2

Data storage format and loader implementation

- Data storage format: how are the videos stored on disk?
- Data loader implementation: how are the videos loaded during training?
- This is all software and being smart here can have a high impact
- Assumption: faster load time to main memory implies better GPU utilization

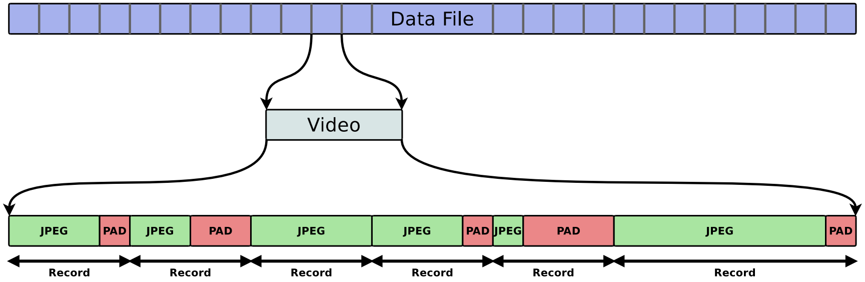
Outline

Data Formats: Bursted Frames or Encoded Videos

- Insight: A video is just a series of images
- Naive approach: decode the video into frames and store them on disk
- Slightly better: store the frames one after another in a binary file
- Maybe better: store in a video format

GulpIO

- A simple binary storage format
- Essence: concatenated JPEGs on disk
- Metadata in separate file



GulpIO

- Initial implementation for Kinetics last year, helped us get 3rd place
- https://github.com/TwentyBN/GulpIO
- https://medium.com/twentybn/introducing-gulpio-f97b07f1da58
- We are receiving pull-requests → people are using it
- Has 95 stars on github

Encoded Videos

- Principle: a video codec uses delta-compression and key frames. I.e. only certain frames are stored and the following frames are stored as a difference to these frames.
- Issue: Video Codecs might be lossy and optimized for subjective human video quality not for training

Encoded Videos

We have experience with:

- `mp4` as container and `h264` as codec
- `webm` as container and `vp9` as codec

Encoded Videos

- Assumption: smaller file size is better
- Less overhead when loading the data from disk

```
% ls -sh1 large_326ee10517267ff6e5be.*
1,3M video.mp4
400K video.webm
```

- But what about decoding?

Frames vs. Videos

```
% ffmpeg -i video.mp4 -q:v 1 -f image2
  -r 15 -vf 'scale=512x512' 'video_%04d.jpg' 2> /dev/null
% du -sch video_0*.jpg | tail -1
5,0M total
```

- Obviously depends on both frame rate (15) and resolution (512x512)

Optimizing Data for Loading

Regardless of whether you choose bursted frames or videos, you can tweak frame-rate and resolution before training.

Outline

Open-Source Packages for Loading Videos

- All of the following use `ffmpeg` and/or the underlying library `av`

- scikit-video
- moviepy
- nvvl
- lintel
- PyAV

scikit-video

- Motto: *Video Processing in Python*
- http://www.scikit-video.org/stable/
- Video loader inspired by `imageio`
- However: uses `popen` to execute either `ffmpeg` or `avconv`
- The result is a `fork` system call and multiple memory copies
- $\rightarrow$ maybe slow

MoviePY

- Motto: *Video Editing in Python*
- http://zulko.github.io/moviepy/
- Presumably suffers from the same implementation weakness as scikit-video
- $\rightarrow$ maybe slow

nvvl

- Motto: *A library that uses hardware acceleration to load sequences of video frames to facilitate machine learning training*
- https://github.com/NVIDIA/nvvl
- Should in principle decode the video *on the GPU*
- Theoretically probably the smartest approach
- However: We could never get this to work :(

Lintel

- Motto: *A Python module to decode video frames directly, using the FFmpeg C API.*
- https://github.com/dukebw/lintel
- Fairly fresh library focussed on machine (deep) learning applications
- Avoids the issues of `imamgeio`, `scikit-video` and `moviepy` (even mentions this in the readme)
- However: segfaults (currently: when reading `webm` files)
- However: submitted issues are taken care of within 24 hours
- Tricky to install as it requires compilation from source (also of the dependencies)

PyAV

- Motto: *Pythonic bindings for FFmpeg*
- http://mikeboers.github.io/PyAV/
- Fully fledged bindings using `cython`
- Easy to install with `conda`
- Can convert directly to `numpy` arrays
- Anecdotal: We have seen memory leaks however, especially with mp4 files

Micro Benchmarks

- `scikit-video` vs. `PyAV` and `mp4/h264` vs. `webm/vp9`
- This will serve as an inidication of what we found in large scale trainings
- Micro-benchmarks are always bullshit, so I encourage you to challenge me and run your own

Outline

Interlude: `nocache`

- As stated earlier, we are assuming, that the dataset doesn't fit into memory and file-system cache.
- In order to simulate this during benchmarking, we use a tool called `nocache`
- This hijacks all system-calls to the file-system cache and thus the video is loaded from the storage medium each time

Experimental Setup

- Azure instance: NCv3 / 24 CPU cores / 4 Tesla V100 GPUs / storge unclear
- Laptop: dell XPS 13 / 4 cores / random GPU / ssd
- All Software installed using `miniconda`

```
av                          0.3.3                     py36_2    conda-forge
sk-video                    1.1.10                      py_3    conda-forge
ffmpeg                      3.4.2                          0    conda-forge
```

- files are as follows:

```
% ls -sh1 *.webm *.mp4
 84K smth-smth-1.webm
1,3M video_large.mp4
400K video_large.webm
```

- Script and files available in the slides repository

Laptop with nocache

```
% nocache ipython benchmark.py
Using runner: 'skvideo_alpha' on file: 'video_large.mp4'
439 ms ± 38.2 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Using runner: 'skvideo_beta' on file: 'video_large.mp4'
417 ms ± 46.4 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Using runner: 'pyav' on file: 'video_large.mp4'
382 ms ± 8.97 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Using runner: 'skvideo_alpha' on file: 'video_large.webm'
569 ms ± 13.6 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Using runner: 'skvideo_beta' on file: 'video_large.webm'
577 ms ± 10.7 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Using runner: 'pyav' on file: 'video_large.webm'
242 ms ± 4.04 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Using runner: 'skvideo_alpha' on file: 'smth-smth-1.webm'
274 ms ± 7.62 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Using runner: 'skvideo_beta' on file: 'smth-smth-1.webm'
269 ms ± 4.23 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Using runner: 'pyav' on file: 'smth-smth-1.webm'
45.5 ms ± 903 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
nocache ipython benchmark.py  37,50s user 7,35s system 146% cpu 30,641 total
```

Laptop without nocache

```
% ipython benchmark.py
Using runner: 'skvideo_alpha' on file: 'video_large.mp4'
333 ms ± 27.2 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Using runner: 'skvideo_beta' on file: 'video_large.mp4'
325 ms ± 7.37 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Using runner: 'pyav' on file: 'video_large.mp4'
376 ms ± 9.16 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Using runner: 'skvideo_alpha' on file: 'video_large.webm'
419 ms ± 5.52 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Using runner: 'skvideo_beta' on file: 'video_large.webm'
426 ms ± 5.29 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Using runner: 'pyav' on file: 'video_large.webm'
242 ms ± 1.75 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Using runner: 'skvideo_alpha' on file: 'smth-smth-1.webm'
118 ms ± 2.78 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
Using runner: 'skvideo_beta' on file: 'smth-smth-1.webm'
115 ms ± 2.87 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
Using runner: 'pyav' on file: 'smth-smth-1.webm'
45.5 ms ± 532 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
ipython benchmark.py  46,04s user 12,80s system 142% cpu 41,294 total
```

Azure Instance with nocache

```
$ nocache ipython benchmark.py
Using runner: 'skvideo_alpha' on file: 'video_large.mp4'
1.21 s ± 11.9 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Using runner: 'skvideo_beta' on file: 'video_large.mp4'
1.19 s ± 17.7 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Using runner: 'pyav' on file: 'video_large.mp4'
410 ms ± 2.21 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Using runner: 'skvideo_alpha' on file: 'video_large.webm'
2.09 s ± 8.61 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Using runner: 'skvideo_beta' on file: 'video_large.webm'
2.09 s ± 9.91 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Using runner: 'pyav' on file: 'video_large.webm'
310 ms ± 11.7 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Using runner: 'skvideo_alpha' on file: 'smth-smth-1.webm'
1.79 s ± 7.79 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Using runner: 'skvideo_beta' on file: 'smth-smth-1.webm'
1.79 s ± 12.7 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Using runner: 'pyav' on file: 'smth-smth-1.webm'
55.3 ms ± 2.42 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Azure Instance without `nocache`

```
$ ipython benchmark.py
Using runner: 'skvideo_alpha' on file: 'video_large.mp4'
223 ms ± 3.12 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Using runner: 'skvideo_beta' on file: 'video_large.mp4'
212 ms ± 3.18 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
Using runner: 'pyav' on file: 'video_large.mp4'
366 ms ± 6.72 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Using runner: 'skvideo_alpha' on file: 'video_large.webm'
380 ms ± 3.3 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Using runner: 'skvideo_beta' on file: 'video_large.webm'
382 ms ± 6.75 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Using runner: 'pyav' on file: 'video_large.webm'
299 ms ± 3.53 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Using runner: 'skvideo_alpha' on file: 'smth-smth-1.webm'
95.5 ms ± 554 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
Using runner: 'skvideo_beta' on file: 'smth-smth-1.webm'
93.5 ms ± 408 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
Using runner: 'pyav' on file: 'smth-smth-1.webm'
54.8 ms ± 689 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Benchmarking during training

- Anecdotal evidence
- Here is something our Principle A.I. Researcher sent me
- Average video loading time

```
|Dataset | Num workers = 1 | Num workers = 10 |
| ------ |:---------------:|:----------------:|
|SkVideo |     110.4 ms    |       12.7 ms    |
|PyAV    |     33.5 ms     |        5.3 ms    |
```

- Trust me, I'm a V.P. :)

Outline

Take-Home message

- If your goal is to have the GPUs highly utilized:
- `webm/vp9` is currently superior to `mp4/h264`
- `PyAV` is currently the superior loader
- Transcode your videos to the desired spatial and temporal resolution
- All of the above may change at any time
- Run your own benchmarks...

Thanks!

I’d like to thank everyone at TwentyBN for the tests they ran and the feedback on the slides.

A shameless plug

- Last week at TwentyBN we released version 2 of our flagship dataset: `something-something`
- https://20bn.com/datasets/something-something
- Free for academic use, licences available for commercial use
- We also released code to train some baselines under MIT licence
- https://github.com/TwentyBN/something-something-v2-baseline

- Yes the files are in `webm/vp9` format and yes there is a `PyAV` based loader

- Contact: `v@20bn.com`

- Questions?