SpeedyLoader: Efficient Pipelining of Data Preprocessing and Machine Learning Training

Rahma Nouaji, Stella Bitchebe, Oana Balmau

EuroMLSys’24, Athens, Greece, April 22, 2024

https://discslab.cs.mcgill.ca
Why do we care about data preprocessing?
Why do we care about data preprocessing?

- Data sample quality is crucial for prediction accuracy.
Why do we care about data preprocessing?

- Data sample quality is crucial for prediction accuracy.
- Data preprocessing is often overlooked comparing to training algorithms.
Why do we care about data preprocessing?

- Data sample quality is crucial for prediction accuracy.
- Data preprocessing is often overlooked comparing to training algorithms.
- Dynamic datasets need online data preprocessing.
Why do we care about data preprocessing?

- Data sample quality is crucial for prediction accuracy.
- Data preprocessing is often overlooked comparing to training algorithms.
- Dynamic datasets need online data preprocessing.
- A straightforward pipelining between online data preprocessing and training with PyTorch DataLoader results in average GPU idleness of 85%.
Why do we care about data preprocessing?

- Data sample quality is crucial for prediction accuracy.
- Data preprocessing is often overlooked comparing to training algorithms.
- Dynamic datasets need online data preprocessing.
- A straightforward pipelining between online data preprocessing and training with PyTorch DataLoader results in average GPU idleness of 85%.

SpeedyLoader
Our workload

3D Image Segmentation Workload.

- The KiTS19 challenge dataset with 210 cases.
- 3D-UNet model.
Our workload

3D Image Segmentation Workload.

- The KiTS19 challenge dataset with 210 cases.
- 3D-UNet model.

Why?

- 8 data preprocessing techniques.
- Manageable dataset size (29GB).
Overview of ML data preprocessing pipeline

1. Kits19 Raw data (NIFTI, 27GB)
2. Preprocess
3. Preprocessed data (Numpy, 29GB)
4. Model training

A. Offline data preprocessing
   Done once, before the training starts

B. Training for one epoch, with online data preprocessing
Offline preprocessing overhead

![Bar chart showing processing times for different offline preprocessing techniques. The chart compares Case 00130 and Case 00165.](chart.png)
Offline preprocessing overhead

Average time in ms:
- Intensity norm: 115ms
- 3D Resampling: 380ms
- Min Padding: 55ms

[1, 53, 512, 512]
[1, 734, 512, 512]
Online preprocessing overhead

<table>
<thead>
<tr>
<th>Technique</th>
<th>Time run 1 (ms)</th>
<th>Time run 2 (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random flip</td>
<td>$2.762 \times 10^{-3}$</td>
<td>32</td>
</tr>
<tr>
<td>Cast</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Random Brightness Aug</td>
<td>6</td>
<td>$1.054 \times 10^{-3}$</td>
</tr>
<tr>
<td>Gaussian Noise</td>
<td>$2.005 \times 10^{-3}$</td>
<td>149</td>
</tr>
<tr>
<td>Random Balance Crop</td>
<td>965</td>
<td>0.172</td>
</tr>
</tbody>
</table>

Table 1. Execution time (in ms) for Online Preprocessing Techniques on case_00039 for two training runs.
What are our key takeaways from this study?

The image processing time is influenced by two factors:

1. Image size.

2. The randomness in the online preprocessing transformations.
Inefficient pipelining using PyTorch DataLoader
SpeedyLoader
SpeedyLoader

**Objective**: Enhancing training time efficiency and GPU utilization while preserving accuracy.
Objective: Enhancing training time efficiency and GPU utilization while preserving accuracy.

How?
**Objective**: Enhancing training time efficiency and GPU utilization while preserving accuracy.

**How?**

1. Combines the offline and online preprocessing into one block.
SpeedyLoader

**Objective**: Enhancing training time efficiency and GPU utilization while preserving accuracy.

**How ?**

1. Combines the offline and online preprocessing into one block.

2. Introduces a load balancer.
Objective: Enhancing training time efficiency and GPU utilization while preserving accuracy.

How?

1. Combines the offline and online preprocessing into one block.

2. Introduces a load balancer.

3. Introduces a pipelining of data loader worker threads and GPU threads via a shared producer-consumer queue.
DataLoader → CPU → Load balancer

1. Batch transfer
2. Processed batch enqueueing thread

Queue for low processing time images
Queue for high processing time images
Uniform batch queue

SpeedyLoader

1. Pulling without waiting
Experimental Environment

**System:**
- NVIDIA DGX-1 machine, with a 2.20GHz 80-core Intel Xeon processor
- 512GB of memory
- 8 NVIDIA V100 32GB GPUs
- CUDA 12.3 and PyTorch 2.1.2.

**Metrics:**
- Total training time using `time` function from Python.
- GPU usage using `nvidia-smi`, CPU usage using `top`.

**Experiment:**
- 4 Batch size, 20 queue max size, 30 workers.
**Results:** Total training time of 3D-UNet model.
Results: Total training time of 3D-UNet model.

- **SpeedyLoader** provides up to **30%** better training time.
Results: CPU and GPU usage 3D-UNet training for 5 epochs, 8 GPUs
Results: CPU and GPU usage 3D-UNet training for 5 epochs, 8 GPUs

Pytorch Dataloader

SpeedyLoader

CPU and GPU usage improves by 2x and 4.3x while maintaining 91% accuracy.
Summary
Summary

➢ Case study of data preprocessing in image segmentation workload.
Summary

➢ Case study of data preprocessing in image segmentation workload.

➢ **Bottleneck**: Inefficient pipelining of preprocessing and training.
Summary

➢ Case study of data preprocessing in image segmentation workload.

➢ **Bottleneck**: Inefficient pipelining of preprocessing and training.

➢ **Solution**: SpeedyLoader

  ○ Relies on shared queue between loading threads and GPU threads.
  ○ Implements a Load Balancer to mitigate head-of-line blocking.
Summary

➢ Case study of data preprocessing in image segmentation workload.

➢ **Bottleneck**: Inefficient pipelining of preprocessing and training.

➢ **Solution**: SpeedyLoader

  ○ Relies on shared queue between loading threads and GPU threads.
  ○ Implements a Load Balancer to mitigate head-of-line blocking.

➢ **30%** decrease in training time and a **4.3x** increase in GPU usage with 91% accuracy.
Future Work

➢ Achieve 100% GPU usage.
➢ Study other workloads.
➢ Compare to other data loaders.
➢ Support collocation of different workloads.

Check out our website:
https://discslab.cs.mcgill.ca

Contact me:
rahma.nouaji@mail.mcgill.ca