SpeedyLoader: Efficient Pipelining of Data Preprocessing and Machine Learning Training

Rahma Nouaji, Stella Bitchebe, Oana Balmau

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

McGill DISCS Lab

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.

McGill UNIVERSITY

Our workload

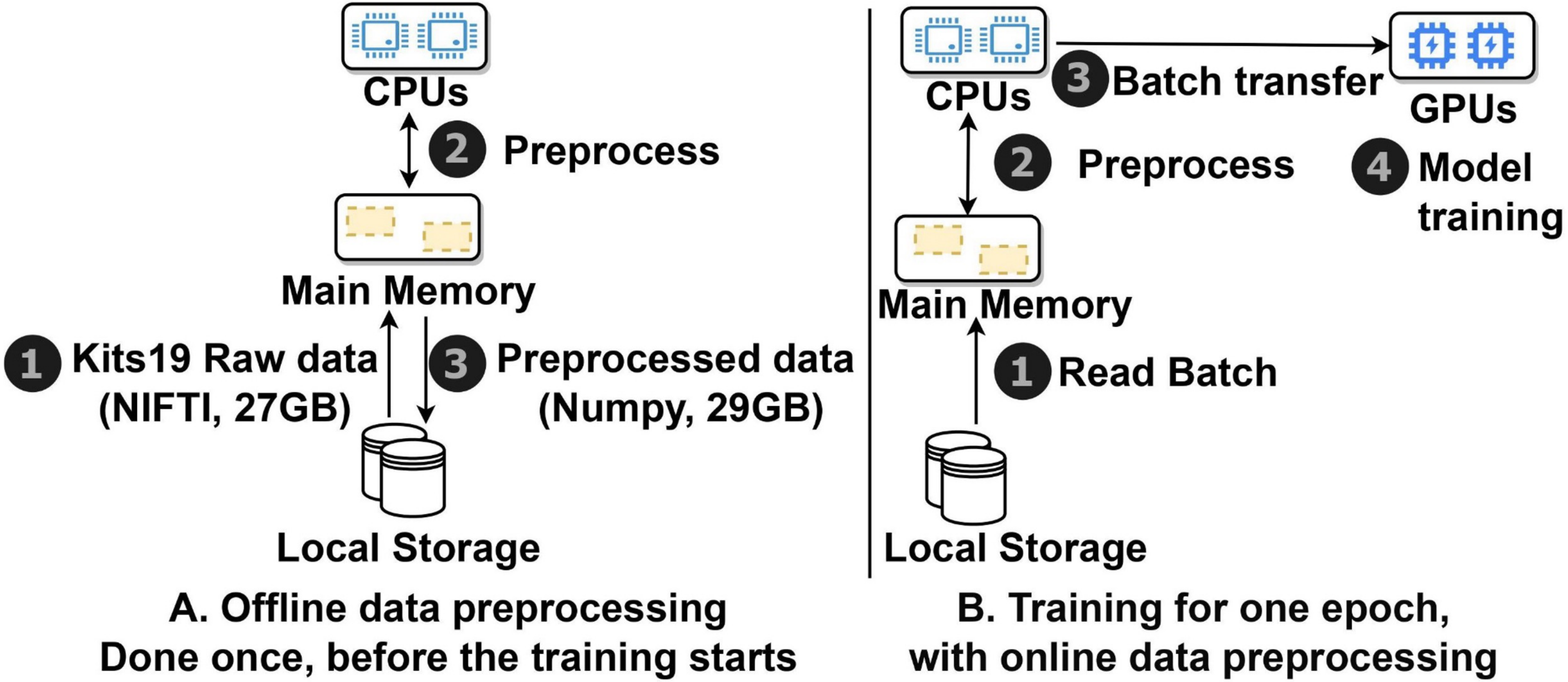
3D Image Segmentation Workload.

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

Why?

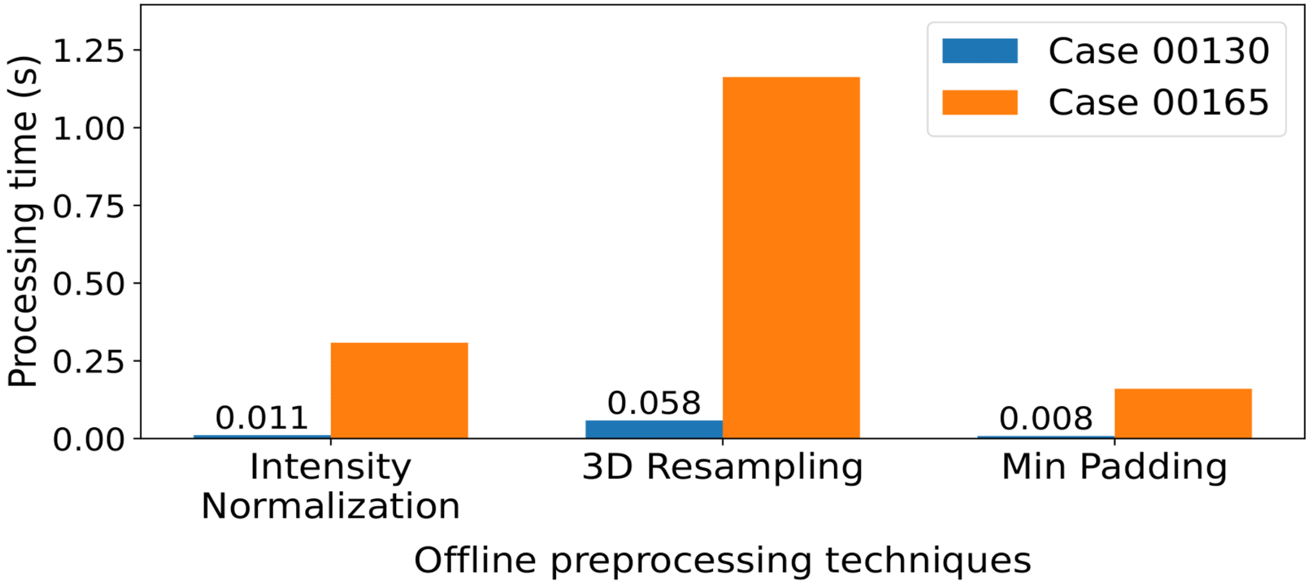
- 8 data preprocessing techniques.
- Manageable dataset size (29GB).

Overview of ML data preprocessing pipeline

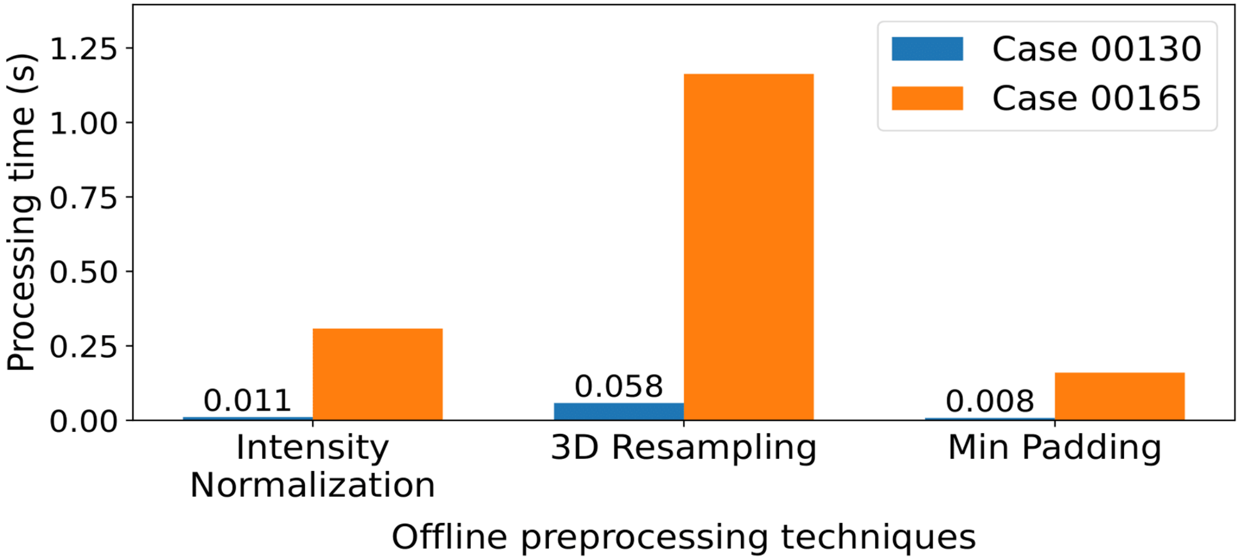


McGill UNIVERSITY

Offline preprocessing overhead



Offline preprocessing overhead



[1, 53, 512, 512]

[1, 734, 512, 512]

Average time in ms:

Intensity norm 115ms

3D Resampling 380ms

Min Padding 55ms

McGill UNIVERSITY

Online preprocessing overhead

Table 1. Execution time (in ms) for Online Preprocessing Techniques on case_00039 for two training runs.

Technique	**Time run 1 (ms)**	**Time run 2 (ms)**
Random flip	2.762×10^{-3}	32
Cast	3	3
Random Brightness Aug	6	1.054×10^{-3}
Gaussian Noise	2.005×10^{-3}	149
Random Balance Crop	965	0.172

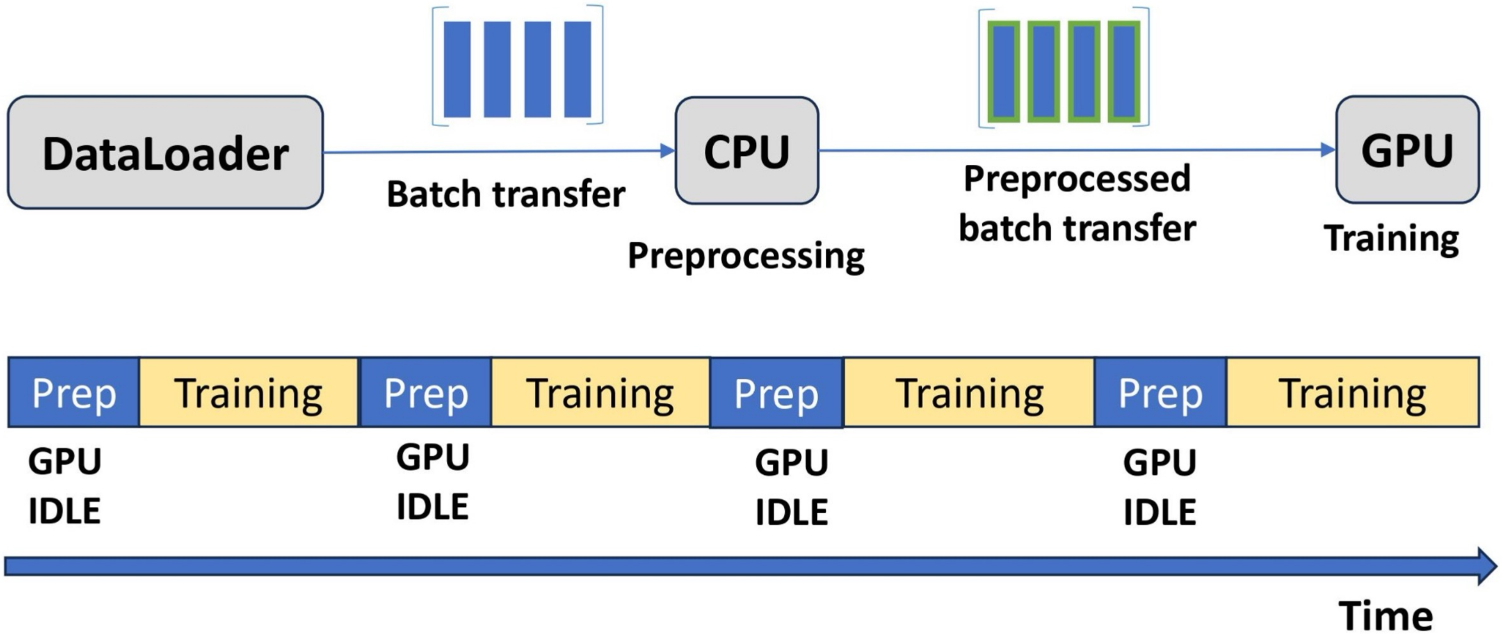
What are our key takeaways from this study?

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



McGill UNIVERSITY

SpeedyLoader

SpeedyLoader

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

SpeedyLoader

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

How ?

SpeedyLoader

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.

SpeedyLoader

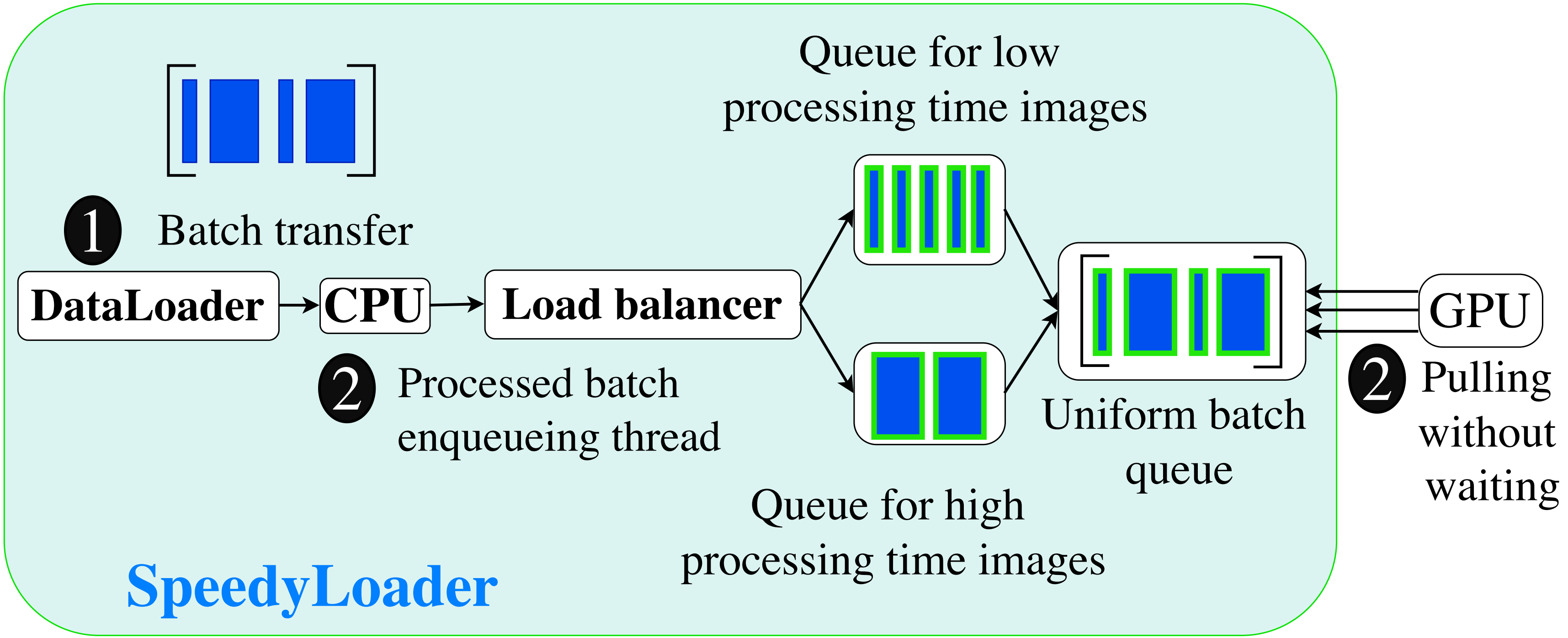
Objective: Enhancing training time efficiency and GPU utilization while preserving accuracy.

How ?

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

2. Introduces a load balancer.

3. Introduces a pipelining of data loader worker threads and GPU threads via a shared producer-consumer queue.


Queue for low
processing time images
1 Batch transfer
DataLoader
CPU
Load balancer
GPU
2 Processed batch
enqueueing thread
Uniform batch
queue
2 Pulling
without
waiting
Queue for high
processing time images
SpeedyLoader

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.

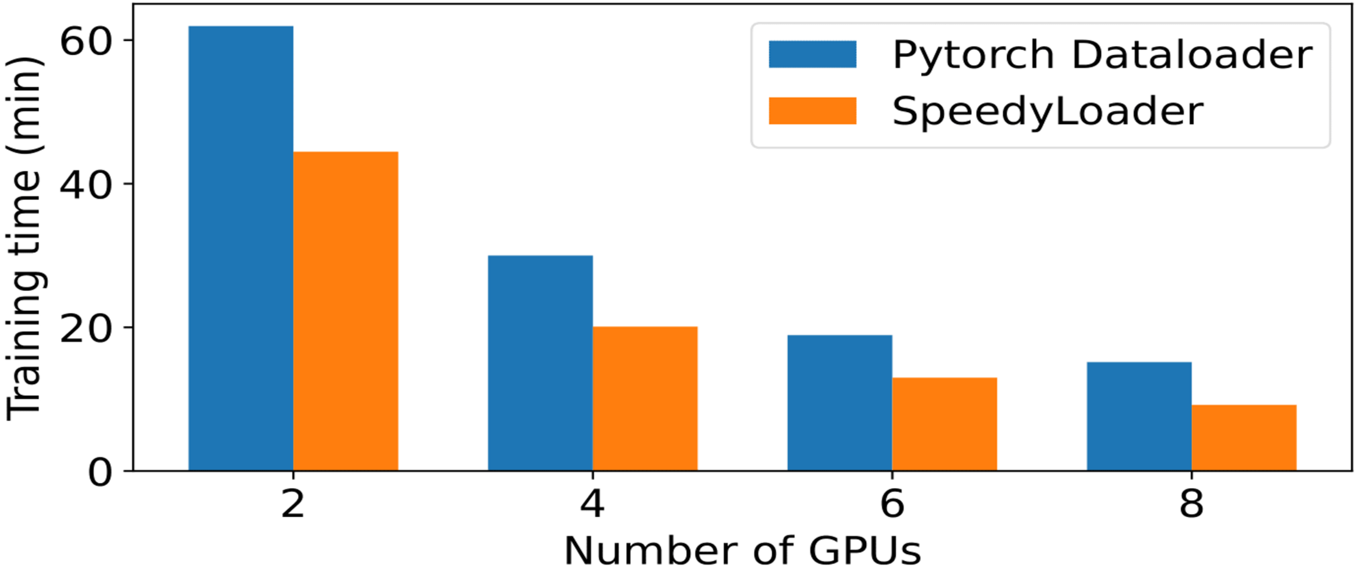
Metrics:

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

Experiment:

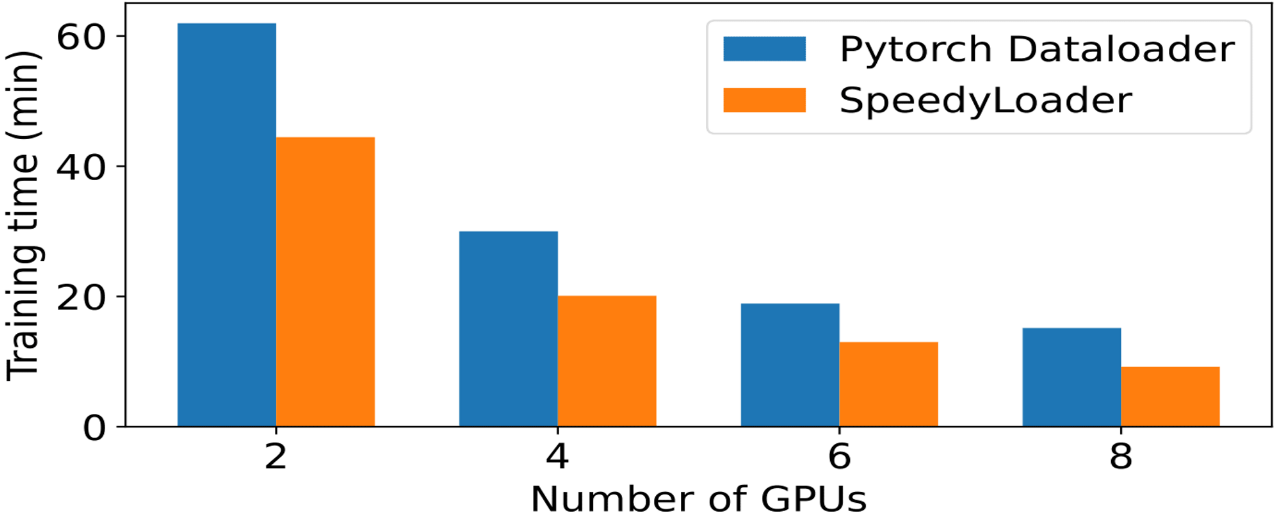
- 4 Batch size , 20 queue max size, 30 workers.

Results: Total training time of 3D-UNet model.



McGill UNIVERSITY

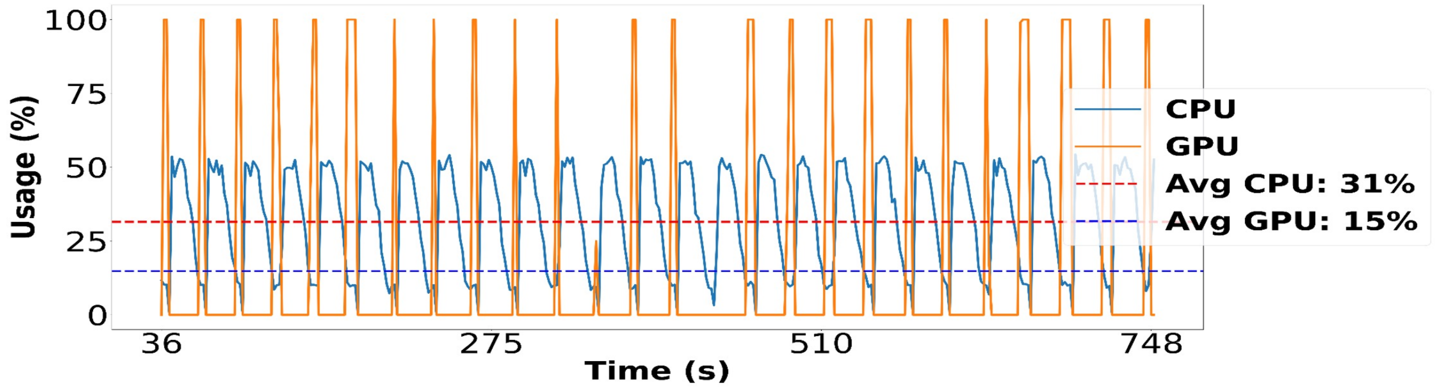
Results: Total training time of 3D-UNet model.



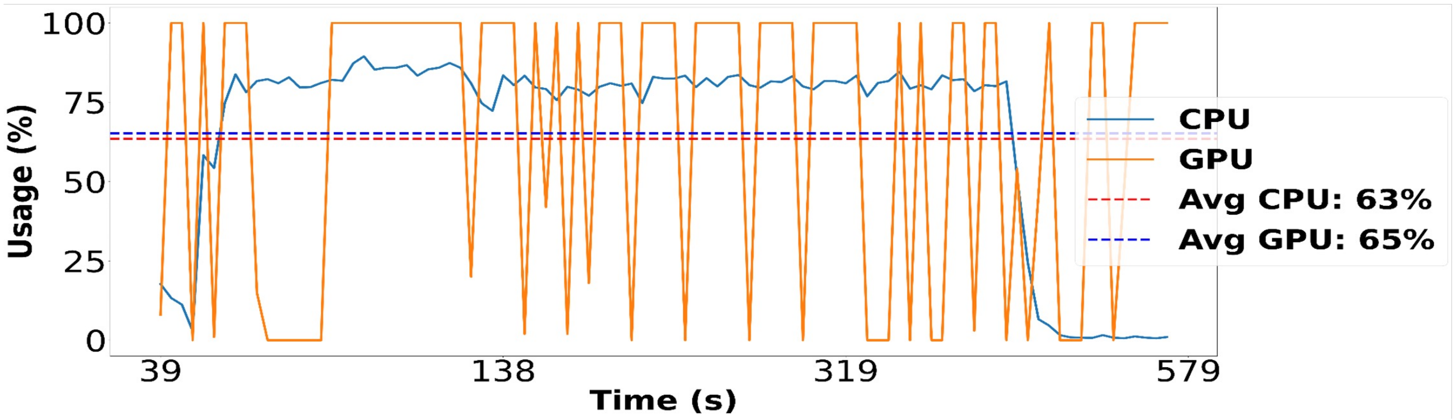
***SpeedyLoader** provides up to **30%** better training time.*

Results: CPU and GPU usage 3D-UNet training for 5 epochs, 8 GPUs

Pytorch Dataloader

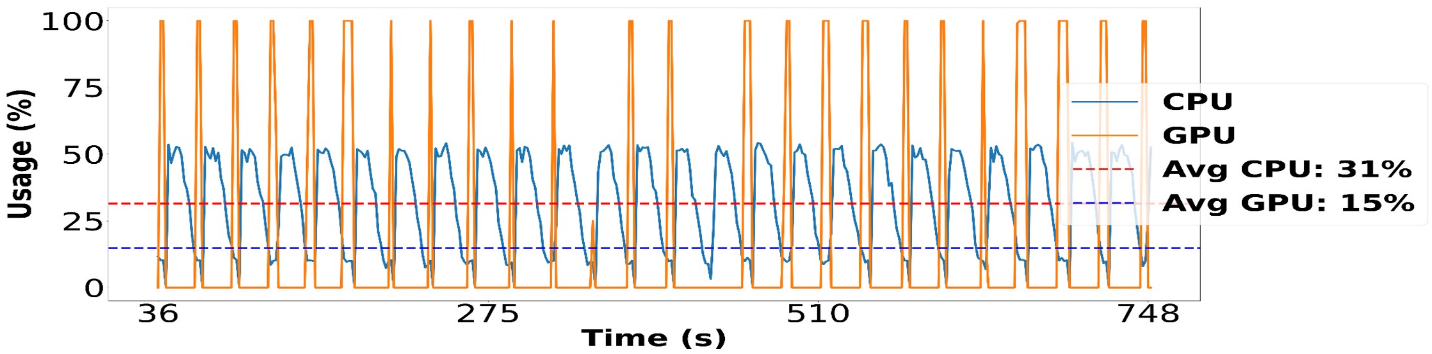


SpeedyLoader

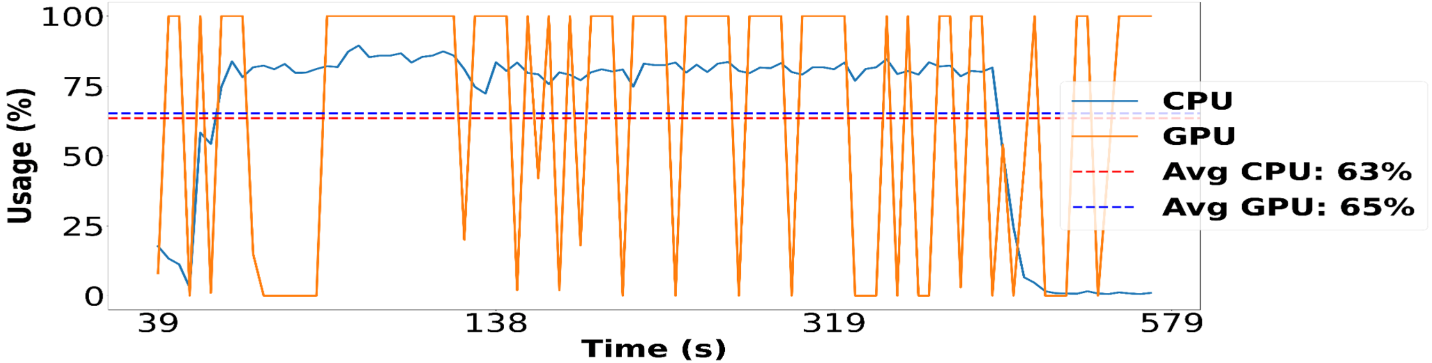


Results: CPU and GPU usage 3D-UNet training for 5 epochs, 8 GPUs

Pytorch Dataloader



SpeedyLoader



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

McGill UNIVERSITY

Summary

Summary

➢ Case study of data preprocessing in image segmentation workload.

Summary

➢ Case study of data preprocessing in image segmentation workload.

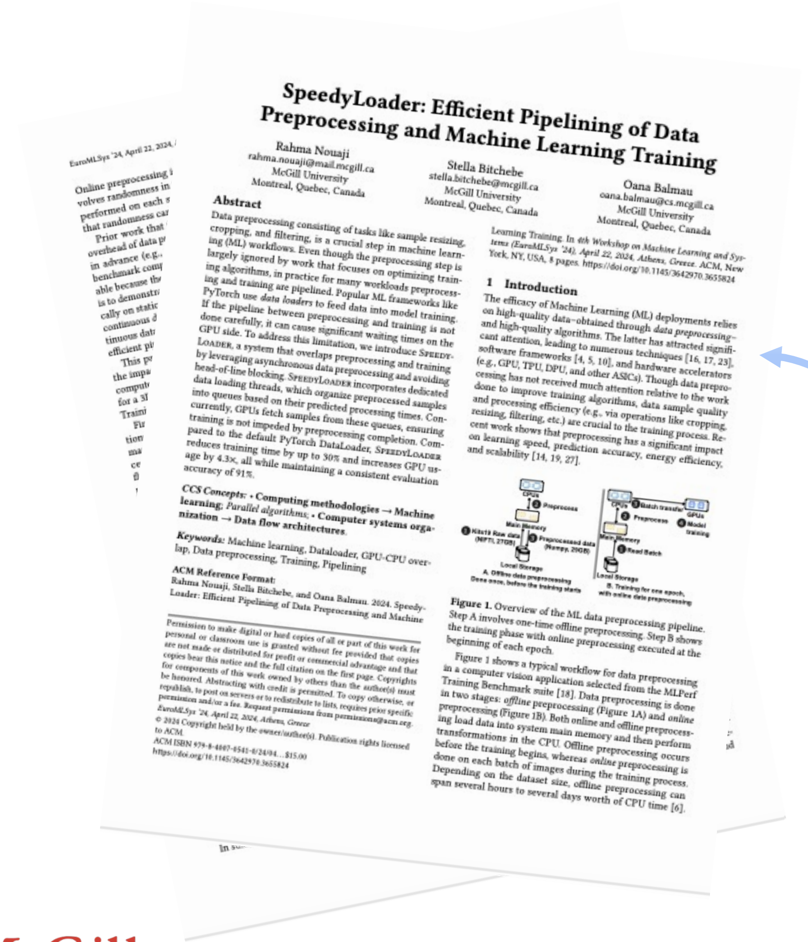
➢ **Bottleneck**: Inefficient pipelining of preprocessing and training.

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.


SpeedyLoader: Efficient Pipelining of Data Preprocessing and Machine Learning Training

Rahma Nouaji
rahma.nouaji@mail.mcgill.ca
McGill University
Montreal, Quebec, Canada

Stella Bitchebe
stella.bitchebe@mcgill.ca
McGill University
Montreal, Quebec, Canada

Oana Balmau
oana.balmau@cs.mcgill.ca
McGill University
Montreal, Quebec, Canada

Abstract

Data preprocessing consisting of tasks like sample resizing, cropping, and filtering, is a crucial step in machine learning (ML) workflows. Even though the preprocessing step is largely ignored by work that focuses on optimizing training algorithms, in practice for many workloads preprocessing and training are pipelined. Popular ML frameworks like PyTorch use *data loaders* to feed data into model training. If the pipeline between preprocessing and training is not done carefully, it can cause significant waiting times on the GPU side. To address this limitation, we introduce SpeedyLoader, a system that overlaps preprocessing and training by leveraging asynchronous data preprocessing and avoiding head-of-line blocking. SpeedyLoader incorporates dedicated data loading threads, which organize preprocessed samples into queues based on their predicted processing times. Concurrently, GPUs fetch samples from these queues, ensuring training is not impeded by preprocessing completion. Compared to the default PyTorch DataLoader, SpeedyLoader reduces training time by up to 30% and increases GPU usage by 4.3×, all while maintaining a consistent evaluation accuracy of 91%.

CCS Concepts: • Computing methodologies → Machine learning; *Parallel algorithms*; • Computer systems organization → Data flow architectures.

Keywords: Machine learning, Dataloader, GPU-CPU overlap, Data preprocessing, Training, Pipelining

ACM Reference Format:
Rahma Nouaji, Stella Bitchebe, and Oana Balmau. 2024. SpeedyLoader: Efficient Pipelining of Data Preprocessing and Machine Learning Training. In *4th Workshop on Machine Learning and Systems (EuroMLSys '24), April 22, 2024, Athens, Greece.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3642970.3655824

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
EuroMLSys '24, April 22, 2024, Athens, Greece
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0541-0/24/04...$15.00
https://doi.org/10.1145/3642970.3655824

1 Introduction

The efficacy of Machine Learning (ML) deployments relies on high-quality data–obtained through *data preprocessing*–and high-quality algorithms. The latter has attracted significant attention, leading to numerous techniques [16, 17, 23], software frameworks [4, 5, 10], and hardware accelerators (e.g., GPU, TPU, DPU, and other ASICs). Though data preprocessing has not received much attention relative to the work done to improve training algorithms, data sample quality and processing efficiency (e.g., via operations like cropping, resizing, filtering, etc.) are crucial to the training process. Recent work shows that preprocessing has a significant impact on learning speed, prediction accuracy, energy efficiency, and scalability [14, 19, 27].

Figure 1. Overview of the ML data preprocessing pipeline. Step A involves one-time offline preprocessing. Step B shows the training phase with online preprocessing executed at the beginning of each epoch.

Figure 1 shows a typical workflow for data preprocessing in a computer vision application selected from the MLPerf Training Benchmark suite [18]. Data preprocessing is done in two stages: *offline* preprocessing (Figure 1A) and *online* preprocessing (Figure 1B). Both online and offline preprocessing load data into system main memory and then perform transformations in the CPU. Offline preprocessing occurs before the training begins, whereas *online* preprocessing is done on each batch of images during the training process. Depending on the dataset size, offline preprocessing can span several hours to several days worth of CPU time [6].

Find out more in our paper!!

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

McGill UNIVERSITY