IIT Kharagpur | IIT Madras | IIT Goa | IIT Palakkad

Introduction to Deep Learning

Profiling with DLProf

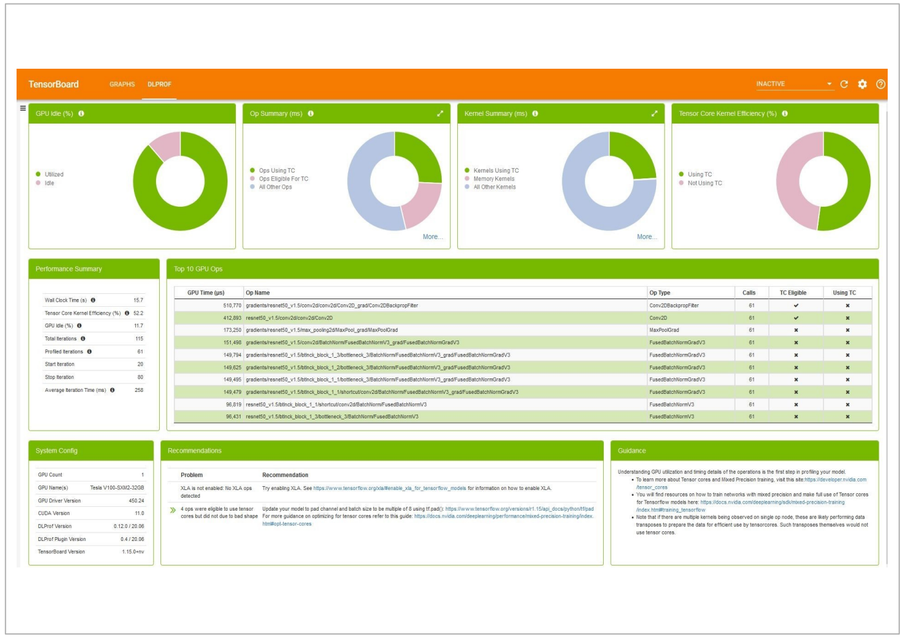
PyTorch Catalyst

Dr. Satyajit Das
Assistant Professor
Data Science
Computer Science and Engineering
IIT Palakkad

National Supercomputing Mission

Centre for Development of Advanced Computing

FW Support: TF, PyT, and TRT Lib Support: DALI, NCCL



Visualize Analysis and Recommendations

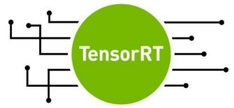
Credit: NVIDIA

Deep Learning Profiler

- **DLProf (CLI & Viewer)**
  - Helps data scientists understand and improve performance of their DL models by analyzing text reports or visualizing the profiling data
- **DLProf CLI**
  - Uses Nsight Systems profiler under the hood
  - Aggregates and correlates CPU and GPU profiling data from a training run to DL model
  - Provides accurate Tensor Core usage detection for operations and kernels
  - Identifies performance issues and provides recommendations via Expert Systems
- **DLProf Viewer**
  - Uses the results from DLProf CLI and provides visualization of the data
  - Currently exists as a TensorBoard plugin

Credit: NVIDIA

GETTING STARTED



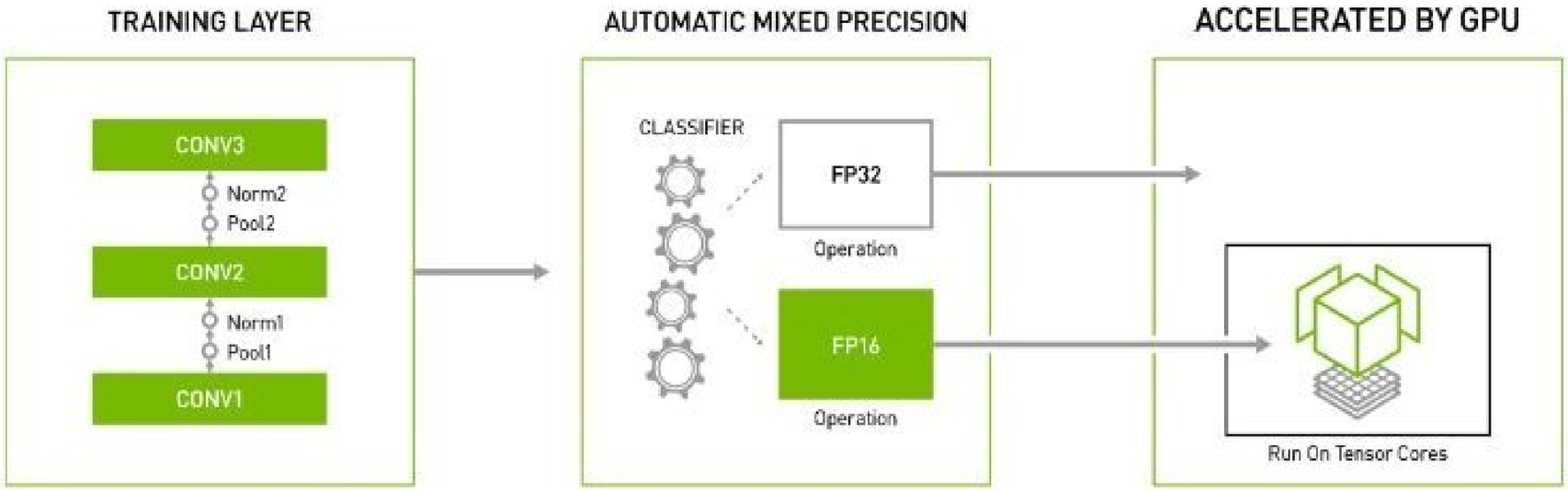
1. TensorFlow and TRT require no additional code modification
2. Profile using DLProf CLI - prepend with ***dlprof***
3. Visualize results with DLProf Viewer

1. Add few lines of code to your training script to enable **nvidia_dlprof_pytorch_nvtx** module
2. Profile using DLProf CLI - prepend with ***dlprof***
3. Visualize with DLProf Viewer

Credit: NVIDIA

AUTOMATIC MIXED PRECISION

AMP



Credit: NVIDIA

BEFORE YOU PROFILE

Do:

- make sure your code runs without an issue
- make a habit of using profiler when you make changes to your code
- Observe if changes you made improve the training performance
- get familiar with the optional arguments DLProf provides
- Iteration range, delay, duration and etc,,

Don’t:

- profile for extended periods of time. It will take very long to profile
- DL training is repetitive and you only need a couple minutes to profile to learn
- try to open DLProf database with your TensorBoard
- You need NVIDIA TB GPU plugin to visualize DLProf event files

Credit: NVIDIA

DEMO

CATALYST

INTRODUCTION

- Catalyst — A Framework for Accelerated Deep Learning R&D based on Pytorch
- Write code with PyTorch, accelerate it with Catalyst!

TRAINING LOOP

```
dl_run = ...
for stage in dl_run.stages:
    for epoch in stage.epochs:
        for loader in epoch.loaders:
            for batch in loader.batches:
                dl_run.handle_batch(batch)
```

SUPPORTED FEATURES

- Universal train/inference loop.
- Training Stages Support
- Provision for ‘callbacks’
- Configuration files for model and data hyperparameters.
- Reproducibility – all source code and environment variables are saved.
- Deep Learning best practices: SWA, AdamW, Ranger optimizer, OneCycle, and more.
- Workflow best practices: fp16 support, distributed training, slurm support, DALI loaders.
- Any hardware backend supported: AMP, Apex, DeepSpeed, FairScale, XLA.

UNIVERSAL TRAIN/INFERENCE LOOP

- This is achieved with the help of Runner Abstraction.
- Runner is an abstraction that takes all the logic of your deep learning experiment:
  - the data you are using,
  - the model you are training,
  - the batch handling logic, and
  - everything about the used metrics and monitoring systems.
- It can be used as a for – loop wrapper.

PYTORCH

```
for epoch in range(num_epochs):
        for train_batch in train_loader:
            x, y = train_batch
            x = x.view(len(x), -1)
            logits = model(x)
            loss = criterion(logits, y)
            print("train loss: ", loss.item())
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()
```

#CATALYST

```
runner = dl.SupervisedRunner()
runner.train(
    model=model,
    criterion=criterion,
    optimizer=optimizer,
    loaders={"train": train_loader,
    "valid": valid_loader},
    num_epochs=1, logdir="./logs", verbose=True
    )
```

```
class CustomRunner(dl.Runner):

    def predict_batch(self, batch):
        # model inference step
        return self.model(batch[0].to(self.device).view(batch[0].size(0), -1))

    def handle_batch(self, batch):
        x, y = batch
        x = x.view(len(x), -1)
        logits = self.model(x)
        loss = self.criterion(logits, y)
        self.batch_metrics["loss"] = loss

        if self.is_train_loader:
            loss.backward()
            self.optimizer.step()
            self.optimizer.zero_grad()
runner = CustomRunner()
# model training
runner.train(
    loaders={"train": train_loader, "valid": valid_loader},
    model=model, criterion=criterion, optimizer=optimizer,
    num_epochs=1, logdir="./logs", verbose=True,
)
```

TRAINING STAGES SUPPORT

- Provides features such as:

  ✓Logging

  ✓Model checkpointing

  ✓Evaluation metrics

```
# CATALYST

runner = dl.SupervisedRunner()
runner.train(
    model=model,
    criterion=criterion,
    optimizer=optimizer,
    loaders={"train": train_loader,
    "valid": valid_loader},
    callbacks=[dl.CheckpointCallback(logdir = '/content/
    logs/checkpoints’,
    resume = '/content/logs/checkpoints/best_full.pth')],
    num_epochs=1, logdir="./logs", verbose=True
)
```

PROVISION FOR ‘CALLBACKS’

- The Callback is an abstraction that helps you to customize the logic during your run.
- Once again, you could do anything natively with PyTorch and Catalyst as a for-loop wrapper.
- Provides reusability.

#CATALYST

```
runner = dl.SupervisedRunner()
runner.train(
    model=model, criterion=criterion,
    optimizer=optimizer, loaders=loaders,
    logdir="./logdir", num_epochs=100,
    callbacks=[
        dl.EarlyStoppingCallback(
            loader_key="valid",
            metric_key="loss",
            minimize=True,
            patience=3,
            min_delta=1e-2)])
```

DEMO

https://colab.research.google.com/drive/1YALTZg0w3CU4Nk44PfFOK77Vsr3kejZq?usp=sharing#scrollTo=mWiJrcv07HeM

https://colab.research.google.com/drive/1HbZgL33mk8NFJummKdcPLOYVKcQBmHDH?usp=sharing

CONCLUSION

- An Accelerated Framework based on Pytorch.
- Create and Research something new rather than write yet another train loop.
- Rapid experimentation, reproducibility, and codebase reuse.

"Write code with PyTorch, accelerate it with Catalyst!"

REFERENCES

- https://scitator.com/talk/2003-tea-ds/
- https://medium.com/pytorch/catalyst-a-pytorch-framework-for-accelerated-deep-learning-r-d-ad9621e4ca88
- https://docs.google.com/presentation/d/1I0iCFcqc0cbLOSXK3XFW-7_sx_U1dH_-B_lo2r3oJGI/edit#slide=id.ged67165f29_0_211
- https://analyticsindiamag.com/guide-to-catalyst-a-pytorch-framework-for-accelerated-deep-learning/

Thank You