CosmicTagger

CosmicTagger is an image segmentation application from high energy neutrino physics, and has been deployed as a benchmark application for multiple supercomputing clusters.

Introduction

Neutrinos are a fundamental particle (like electrons, muons and taus - but without electric charge) that we know exists, has mass, and not too much else. Many state of the art physics experiments probe the nature of the neutrino particle, but because the particles are so weakly interacting physicists use large, complex and high resolution detectors to image and study neutrino interactions. The application in this section is developed as a background removal technique important to analysis of some neutrino experiments.

In high energy neutrino physics experiments such as the "Short Baseline Neutrino Detector" at Fermi National Accelerator Laboratory, cosmic ray particles can be so common and pervasive that they dominate the data readout of these detectors. In this application, you will use a deep learning image segmentation technique to label individual pixels in simulated images as either cosmic-ray origin, neutrino-origin, or background pixels.

For more information about the scientific use case of this application, see https://www.frontiersin.org/articles/10.3389/frai.2021.649917/full.

The dataset is composed of 3 2D-projections of a 3D space at high resolution, 2048x1280 pixels. The high resolution data makes for a very computationally intensive training procedure. In this application, you will run CosmicTagger several ways, isolating IO, compute/scaling, and finally running the full application in both training and inference mode.

Software requirements

CosmicTagger is implemented in both tensorflow and pytorch. For the competition, either framework may be used but please note that depending on your system configuration, and site setup, one may perform better than the other.

Download the CosmicTagger code from github here:
https://github.com/coreyjadams/CosmicTagger.git

The IO Layer of the CosmicTagger application is done with a custom, sparse IO framework called `larcv`. This package may be downloaded from github:

https://github.com/DeepLearnPhysics/larcv3.git

larcv3 requires:

- hdf5
- cmake (build only)
- scikit-build (python package, build only)

Additionally, larcv uses pybind11, if you install from source please remember to run:

```
git submodule update --init
```

before building. Build larcv with these instructions:

```
python setup.py build -j 64
python setup.py install
```

Cosmic Tagger requires:

- larcv3 (see above)
- pytorch or tensorflow (your choice)
- horovod (required for tensorflow, optional for pytorch)
- hydra (for configuration)

Installation Suggestion

You may install and run everything however you please, including containers or any other method. However, you may also follow these instructions to get a working build, using miniconda.

```
# Load gcc and openmpi:
module load gcc-9.2.0
module load mpi

wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
bash Miniconda3-latest-Linux-x86_64.sh
# Accept license agreements and select the
# right install location, probably not your
# home directory since disk quota is limited.
#
# Activate conda, if you skipped it's auto initialization
source /path/to/your/conda/install/bin/activate

# Turn on the conda base environment
conda activate

# Install pytorch
conda install pytorch cudatoolkit=11.3 -c pytorch
```

```
# Install build dependencies for larcv3:
conda install cmake hdf5 scikit-build

# Install Tensorflow:
conda install cudnn -c nvidia
pip install tensorflow-gpu
# To run tensorflow with GPUs, make sure you add the conda lib directory to
your LD_LIBRARY_PATH:

# NOTE: if you don't install tensorflow, you need to pip install numpy!

# Clone larcv and install it:
git clone https://github.com/DeepLearnPhysics/larcv3.git
cd larcv3
git submodule update --init
python setup.py build -j 64
python setup.py install

# Install mpi4py:
pip install --force-reinstall mpi4py --no-cache-dir

# Install horovod with tensorflow or if you want it with pytorch:
pip install --force-reinstall horovod --no-cache-dir
```

If you use `tensorflow` for this application, please note that `horovod` is also required. `horovod` is an optional dependency for `pytorch`.

Finally, the configuration of CosmicTagger is with `hydra`: run `pip install --pre hydra-core` to install it.

Please, contact the application expert for installation problems of larcv, tensorflow/pytorch, and other packages. Software installation is not meant to be the challenging part of this application!

The application in Tasks 2 through 4 are implicitly expecting a GPU accelerator. If you do not have an accelerator, please add the following to all of your application launch commands:

run.compute_mode=CPU

Datasets

The dataset for this competition comes in several pieces:

- `cosmic_tagging_train.h5` is the train file
- `cosmic_tagging_test.h5` is used for on-the-file testing, during training and for validation in Task 4.

Additionally, you are provided two files for task4 if you choose to use them: pretrained weights for the network in either tensorflow or pytorch. You may choose to use these, or you may choose to retrain a network and attempt to reach higher inference accuracy for a scoring bonus.

Scoring

Each team will receive a score out of 100 for this application. The score breakdowns are:
- Task 1: 20 points (20%)
- Task 2: 20 points (20%)
- Task 3: 20 points (20%)
- Task 4: 20 points (20%)
- Available bonus points: 20

The value you will report for each task is a measure of throughput for each task, since this is an image processing application. Each task has been benchmarked on an independent HPC system (the details of which are not available until the completion of the competition). Your score in each task will be the ratio of your measured throughput to the benchmarked throughput. Each task will individually be curved/normalized, if necessary, to ensure the top ranking team does not exceed the total amount of points for that task.

Task 4 has an additional bonus (or penalty) to your score based upon a scientific accuracy metric. Task 4 will produced not just a throughput measurement but also a calculation of scientific accuracy of the metric, and your score will be the ratio of throughputs to the "secret" benchmark, scaled by the ratio of your precision. More details are available in the description of Task 4.

Score curving and normalization.

Since there are bonus points available, and the points awarded are based upon a comparison to unknown but measured benchmarks, it may occur that one or more teams exceed the pre-measured benchmark for a task. In this scenario, two things happen:
- The top-performing team's score becomes the new benchmark score to which everyone else's score is compared.
- Every team that beat the unknown benchmark score receives a bonus of 4 points for each task that they exceed the unknown benchmark. Bonus points are award after score normalization. You do not need to get first place to get bonus points - you only need to beat my benchmarks.

In other words: your score for each task the ratio of your throughput to my measured throughput. If you beat me, you get bonus points. If you beat everyone, you become the new benchmark and will automatically get the highest score on this task.

If no one beats my benchmarks on any task, the highest possible score will be 80 out of 100. My measurements are high but it is possible to beat them.

Competition Tasks

There are several tasks to accomplish for this application:

1) IO Performance

2) Distributed Training Scale Out (Small Images)

3) Distributed Training Scale Out (Large Images, reduced precision)

4) Inference Throughput (Accuracy Component.)

Each task is explained in more detail below.

PLEASE NOTE: The network in this application is highly configurable from the command line, but for this competition configuration changes are not allowed except as specified for each task.

What to submit

In this competition you must submit two pieces of information:
1) The log file from your run that you think is best
2) The log directory for that task.

For each task, you may configure where it stores the output directory directly from the command line, though it will also use a default location if none is provided.

```
[rest of command] run.output_dir=path/to/your/output/location/choice
```

You may run the application as many times as you like. Submit for each task only once. Duplicate submissions will be ignored. If you fail to submit a task at all, you will receive a score of 0 for that task. The tasks - while closely related - can all be completed independently except for Task 1. If you can not complete Task 1, you will not be able to complete the other 3 tasks.

General Advice

No task here is designed to take more than 1 hour to run, and all tasks can be completed faster than that. Note that in each task, the `minibatch_size` is something you must decide upon. The selected `minibatch_size` is for the entire program, and images will be divided amongts MPI ranks. Therefore, ensure that ${MINIBATCH_SIZE} / ${N_RANKS} is an even division or there will be an error.

Using a small `minibatch_size` will yield fast results for most tasks, but a larger `minibatch_size` may yield improved computational performance. You will have to experiment.

Tasks 1 and 2 use fp32 precision, while Task 3 uses mixed fp16 precision. Task 4 can use which precision you like.

Task 1: IO Performance

The first challenge in this application isolates the IO component of CosmicTagger. Run the application as follows:

```
mpirun -n $RANKS [mpi arguments] python bin/exec.py run.id=task1 mode=iotest
run.iterations=100 [run.minibatch_size=$MB]
```

Here, you may configure the minibatch size to be as large as you please. In IO mode, the application will read an image or more from file, load it into python as a numpy array, and immediately discard the data and load the next piece of data.

Please note that the larcv package uses `std::futures` to overlap CPU-based IO communication with (typically) GPU-based Machine Learning computations. Therefore the time spent in IO here may become up to 100% "invisible" in later tasks.

Report your results from this challenge based on the text output of the application. The final lines will look something like this:

```
[timestamp] - INFO - Total IO Time: [your number here]
[timestamp] - INFO - Total images read: [your selected batch size]
[timestamp] - INFO - Average Image IO Throughput: [your throughput per batch]
```

Your score here will be based upon the highest average IO throughput achieved, averaged over 100 iterations. You will have to experiment with the number of ranks to use, as well as the total number of images to use per batch. Note that the total minibatch size selected will be divided amongst all ranks evenly. The application will error if you select a minibatch size not divisible by your number of ranks.

Scoring: Task 1 is worth 20% of your total score. Any team that beats my measured throughput receives 4 bonus points. The highest of all measured throughputs (including my measurement) will be used to normalize scores into a range of 0 to 20, with bonus points added after normalization.

Task 2:

In Task 2, the objective is to perform distributed training. To allow you to achieve a reasonable quality of result in a reasonable time, for this task the images in Task 3 are *downsampled* by a factor of 4 in both X and Y. Task 3 is an MPI application, please launch your application similar to this:

```
mpirun -n $N_RANKS [mpi arguments] python bin/exec.py \
--config-name SCC_21.yaml \
run.id=task2 \
run.iterations=1000 \
data.downsample=2 \
framework=[tensorflow | torch] \
run.minibatch_size=${MINIBATCH_SIZE}
```

Here, you can use either tensorflow or pytorch by using `framework=tensorflow` or `framework=torch` as appropriate.

The minibatch size used and the number of ranks used are left to your decision. You must run for 1000 iterations and submit your output file and log directory. At the end of the run, you will see output that looks like this:

```
[timestamp] - INFO - Total time to batch_process: [numeric_value]
[timestamp] - INFO - Total time to batch process except first iteration:
[numeric_value], throughput: [numeric_value]
[timestamp] - INFO - Total time to batch process except first two iterations:
[numeric_value], throughput: [numeric_value]
[timestamp] - INFO - Total time to batch process last 40 iterations:
[numeric_value], throughput: [numeric_value]
```

Your score will be based on the value for throughput except first two iterations, so averaged over nearly the entire run. Note that in this task, the accuracy of your network does not matter. Task 2 is work 20% of your total score.

Scoring: Task 2 is worth 20% of your total score. Any team that beats my measured throughput receives 4 bonus points. The highest of all measured throughputs (including my measurement) will be used to normalize scores into a range of 0 to 20, with bonus points added after normalization.

Task 3

Task 3 is similar to Task 2 except for 2 distinct changes:
- You will use the full resolution data, images of 3x1280x2048
- You will train using reduced precision.

Because of the increased datasize, you may need to reduce the minibatch size. To be explicit, the command you run should look similar to this:

```
mpirun -n $N_RANKS [mpi arguments] python bin/exec.py \
--config-name SCC_21.yaml \
run.id=task3 \
run.iterations=1000 \
data.downsample=0 \
run.precision=mixed \
framework=[tensorflow|torch] \
run.minibatch_size=${MINIBATCH_SIZE}
```

As before, you will be scored based upon your average throughput in Images / second. Your output file should once again look like this:

```
[timestamp] - INFO - Total time to batch_process: [numeric_value]
[timestamp] - INFO - Total time to batch process except first iteration:
[numeric_value], throughput: [numeric_value]
[timestamp] - INFO - Total time to batch process except first two iterations:
[numeric_value], throughput: [numeric_value]
[timestamp] - INFO - Total time to batch process last 40 iterations:
[numeric_value], throughput: [numeric_value]
```

Task 3 is worth 20% of your total score. Any team that beats my measured throughput receives 4 bonus points. The highest of all measured throughputs (including my measurement) will be used to normalize scores into a range of 0 to 20, with bonus points added after normalization.

Task 4

Task 4 is the final task for this application. In this task, you will run full resolution inference. In this task, there is an additional constraint: the accuracy of your network matters. For both tensorflow and pytorch, there are pre-trained weights available (based on the training of Task3, but run for longer). You may download and use these weights if you like; you may also train the network to convergence if you like and try to exceed the scores of these weights. It is also plausible that you can use the pre-trained weights as a very good starting point for further training.

If you attempt to train further

If you attempt to beat the scientific accuracy, you MUST submit your trained weights with your log files. I WILL check that the accuracy you report is what the weights provided yield.

If you attempt to train further, you are permitted to vary parameters in the `mode` section beyond the usual changes in the `run` section (minibatch size, iterations, etc):

```
mode:
  checkpoint_iteration........: 500
  logging_iteration...........: 1
  name........................: train
  no_summary_images...........: False
  optimizer:
    gradient_accumulation.....: 1
    learning_rate.............: 0.0003
    loss_balance_scheme.......: light
    name......................: adam
  summary_iteration...........: 1
```

Running Task 4

Task 4 is not targeting a specific number of iterations, but instead is targeting a total amount of entries processed. You must process at least 7000 individual entries, by ensuring that the product of your `minibatch_size` and `iterations` is more than 7000. Though there is a minimum requirement, you are still scored based on average throughput as before.

You may run task 4 in whatever compute precision you would like to. You may specify a location for weights to restore from using the parameter:

```
mode.weights_location=${WEIGHTS}
```

The images used must be full resolution (`data.downsample=0`) and be sure to use `mode=inference` instead of `mode=train`. Otherwise, the command is the same as Task 3.

In this case, your throughput will be scaled according to the `Average/mIoU` metric achieved. For example, if the throughput you achieve is 10 Images per second over the whole cluster and my measurement is 15, and the mIoU on average is 0.5 while the provided weights should achieve 0.75, your score will be calculated as (0.5/0.75)x(10/15). If you exceed my pretrained network's accuracy, you do not get a scaled score but rather your throughput is used as-is, and you get a flat bonus points award.

To summarize, a Figure of Merit for Task 4 is:

```
FOM = [min(your_accuracy, my_accuracy) / my_accuracy ] * [ your_throughput / best_throughput]
```

The metric values that you should meet or beat are not disclosed here - you can discover them by running inference using the provided weights.

Task 4 is worth 20 points. Any team that beats my measured figure of merit receives 4 bonus points. The highest of all measured throughputs (including my measurement) will be used to normalize scores into a range of 0 to 20, with bonus points added after normalization.

Any team that beats my accuracy by at least 0.02 receives the final 4 bonus points. Unlike throughput, there is no renormalization across teams according to scientific accuracy - everyone competes against my pre-trained networks.