ML Model deployment using Torch

Modoranu Ionut-Cosmin

February 17, 2024

1 Setup

1.1 Trained model

The model used for performing inference is a ResNet18 trained for 30 minutes on T4 GPU on Google Colab.

1.2 Containerized inference script

1.2.1 The environment. Dockerfile.

The containerized environment is built following these steps:

1. **Base and Python Setup:** Begins with an Ubuntu 22.04 base image integrated with NVIDIA CUDA, suitable for GPU-accelerated tasks. Installs Python 3.11.0 from source for the latest feature compatibility.
2. **PyTorch and ONNX Runtime:** Installs PyTorch for Python-based machine learning and ONNX Runtime for optimized model inference across various formats.
3. **C++ Support with LibTorch:** Prepares for C++ development by setting up 'libtorch', essential for utilizing torch in C++ applications.
4. **Script Handling and Execution Flexibility:** Copies Python inference scripts and compiles the C++ inference script as well. The entry point, 'driver.sh', determines the runtime mode based on environment variables, allowing execution in both Python and C++. Driver script also handles the measurement of metrics regarding CPU usage, memory, and execution time.

```
FROM nvidia/cuda:12.3.1-base-ubuntu22.04

ENV DEVICE cpu
ENV DEBIAN_FRONTEND noninteractive

RUN apt-get update && apt-get install -y \
    software-properties-common \
    build-essential \
    wget \
    zlib1g-dev \
    libssl-dev \
    libffi-dev \
    openssl \
    libz-dev \
    libbz2-dev \
    liblzma-dev \
    libreadline-dev \
    libncursesw5-dev \
```

```
    libgdbm-dev \
    libsqlite3-dev \
    libc6-dev \
    libopenblas-dev \
    liblapack-dev \
    libblas-dev \
    cmake \
    unzip \
    time

# Download and install Python 3.11.0
WORKDIR /usr/src/app

RUN wget https://www.python.org/ftp/python/3.11.0/Python-3.11.0.tgz && \
    tar -xzf Python-3.11.0.tgz && \
    rm Python-3.11.0.tgz && \
    cd Python-3.11.0 && \
    ./configure --enable-optimizations && \
    make altinstall


RUN python3.11 -m pip install torch torchvision onnxruntime psutil

WORKDIR /usr/src/libtorch_cpu

# Download libtorch
RUN wget https://download.pytorch.org/libtorch/cpu/libtorch-cxx11-abi-shared-with-deps-2.2.0%2Bcpu.zip
    unzip libtorch-cxx11-abi-shared-with-deps-2.2.0+cpu.zip && \
    rm libtorch-cxx11-abi-shared-with-deps-2.2.0+cpu.zip

WORKDIR /usr/src/app

COPY . .

# Build the C++ code
WORKDIR /usr/src/app/cpp/example-app

RUN mv /usr/src/libtorch_cpu/libtorch /usr/src/app/cpp/example-app/libtorch && \
    rm -rf build/* && \
    cmake -S . -B build && cmake --build build --config Release

WORKDIR /usr/src/app

ENTRYPOINT /bin/bash driver.sh
```

2 Comparison

2.1 Inference script on Windows performance data

The following table shows the performance metrics for various models when the inference script is run on a Windows environment. These are the average values obtained from 30 runs.

Model Type	Accuracy	Avg. Exec. Time (ms)	Avg. Peak Memory (MB)	Avg. CPU Usage (%)
onnx_py	0.44	14730.0	336.6	49.3
torch_py	0.44	15875.0	340.2	34.0

Table 1: Performance metrics of inference script on Windows (bare metal)

This section presents the performance analysis of different model types in containerized environments, specifically in Process and Container setups. The data represents average values obtained from 30 runs.

2.2 Container level performance data

Model Type	Accuracy	Avg. Exec. Time (ms)	Avg. Peak Memory (MB)	Avg. CPU Usage (%)
torch_py	0.44	9030.5	607.75	24.5
onnx_py	0.44	9013.5	642.1	48.4
torch_cpp	0.44	3465.0	426.45	40.8

Table 2: Performance metrics of models at container level

2.3 Process level performance data

Model Type	Accuracy	Avg. Exec. Time (ms)	Avg. Peak Memory (MB)	Avg. CPU Usage (%)
torch_py	0.44	6735.0	575.45	6.55
onnx_py	0.44	6915.0	641.95	31.8
torch_cpp	0.44	3345.8	-	-

Table 3: Performance metrics of models at process level

3 Observations

3.1 Execution Time

- The model using `torch_py` reduces its execution time from 15875.0 ms in Windows to 9030.5 ms in the Containerized environment.
- For the model using `onnx_py`, execution time decreases from 14730.0 ms in Windows to 9013.5 ms in the Containerized environment.
- Performance characterized by `torch_cpp` shows the fastest execution time in the Containerized environment at 3465.0 ms.

3.2 Memory Usage

- In the Containerized environment, the models utilizing `torch_py` and `onnx_py` show increased memory usage at 607.75 MB and 642.1 MB, respectively, compared to their Windows counterparts.
- The consumption profile of `torch_cpp` indicates a lower memory footprint in the Containerized environment (426.45 MB).

3.3 CPU Usage

- CPU usage for the `torch_py`-based model is reduced in the Containerized environment (24.5%) compared to Windows (34.0%).
- The `onnx_py`-based model's CPU usage remains similar between Windows (49.3%) and the Containerized environment (48.4%).
- The model utilizing `torch_cpp` exhibits a CPU usage of 40.8% in the Containerized environment.

3.4 Model Accuracy

- The accuracy of all models remains consistent at 0.44 across both Windows and Containerized environments.

4 Conclusions

- Containerized environments significantly improve execution speed and CPU usage for models using `torch_py` and `onnx_py` compared to Windows.
- In the context of the containerized environment, The performance profile for the model using `torch_cpp` demonstrates superior efficiency in execution time and memory usage.
- While containerization enhances computational efficiency, particularly in terms of speed, it also leads to increased memory usage, which is a critical factor in resource-limited scenarios.
- These findings underline the effectiveness of containerization as a deployment strategy for machine learning models if prioritizing execution time and CPU usage. Still, such a strategy could incur more memory being used.