NVIDIA TensorRT Samples

Support Guide | NVIDIA Docs

TRM-10259-001_v8.6.0 Early Access (EA) | March 2023

Table of Contents

Chapter 1. Introduction

The following samples show how to use NVIDIA® TensorRT™ in numerous use cases while highlighting different capabilities of the interface.

Note: The TensorRT samples are provided for illustrative purposes only and are not meant to be used nor taken as examples of production quality code.

Title	TensorRT Sample Name	Description
`trtexec`	`trtexec`	A tool to quickly utilize TensorRT without having to develop your own application.
"Hello World" For TensorRT From ONNX	sampleOnnxMNIST	Converts a model trained on the MNIST dataset in ONNX format to a TensorRT network.
Building An RNN Network Layer By Layer	sampleCharRNN	Uses the TensorRT API to build an RNN network layer by layer, sets up weights and inputs/outputs and then performs inference.
Performing Inference In INT8 Precision	sampleINT8API	Sets per tensor dynamic range and computation precision of a layer.
Specifying I/O Formats	sampleIOFormats	Uses an ONNX model that was trained on theMNIST dataset and performs engine building and inference using TensorRT. The correctness of outputs is then compared to the golden reference.
Digit Recognition With Dynamic Shapes In TensorRT	sampleDynamicReshape	Demonstrates how to use dynamic input dimensions in TensorRT by creating an engine for resizing dynamically shaped inputs to the correct size for an ONNX MNIST model.
Algorithm Selection API Usage Example Based On sampleMNIST In TensorRT	sampleAlgorithmSelector	End-to-end example of how to use the algorithm selection API based on sampleMNIST.

Title	TensorRT Sample Name	Description
Introduction To Importing ONNX Models Into TensorRT Using Python	introductory_parser_samples	Uses TensorRT and its included ONNX parser, to perform inference with ResNet-50 models trained with various different frameworks.
“Hello World” For TensorRT Using PyTorch And Python	network_api_pytorch_mnist	An end-to-end sample that trains a model in PyTorch, recreates the network in TensorRT, imports weights from the trained model, and finally runs inference with a TensorRT engine.
Writing a TensorRT Plugin to Use a Custom Layer in Your ONNX Model	onnx_custom_plugin	Implements a Hardmax Layer as a TensorRT plugin and uses it to run a ONNX BiDAF question answering model in TensorRT.
Object Detection With The ONNX TensorRT Backend In Python	yolov3_onnx	Implements a full ONNX-based pipeline for performing inference with the YOLOv3-608 network, including pre and post-processing.
TensorRT Inference Of ONNX Models With Custom Layers In Python	onnx_packnet	Uses TensorRT to perform inference with a PackNet network. This sample demonstrates the use of custom layers in ONNX graphs and processing them using ONNX-graphsurgeon API.
Refitting An Engine Built From An ONNX Model In Python	engine_refit_onnx_bidaf	Builds an engine from the ONNX BiDAF model, refits the TensorRT engine with weights from the model.
Scalable And Efficient Object Detection With EfficientDet Networks In Python	efficientdet	Sample application to demonstrate conversion and execution of Google® EfficientDet models with TensorRT.
Scalable And Efficient Image Classification With EfficientNet Networks In Python	efficientnet	Sample application to demonstrate conversion and execution of a Google EfficientNet model with TensorRT.
Implementing CoordConv in TensorRT with a custom plugin using sampleOnnxMnistCoordConvAC In TensorRT	sampleOnnxMnistCoordConvAC	Contains custom CoordConv layers. It converts a model trained on the MNIST dataset in ONNX format to a TensorRT

Title	TensorRT Sample Name	Description
		network and runs inference on the network.
Object Detection with TensorFlow Object Detection API Model Zoo Networks in Python	tensorflow_object_detection_api	Demonstrates the conversion and execution of the Tensorflow Object Detection API Model Zoo models with TensorRT.
Object Detection with Detectron 2 Mask R-CNN R50-FPN 3x Network in Python	detectron2	Demonstrates the conversion and execution of the Detectron 2 Model Zoo Mask R-CNN R50-FPN 3x model with TensorRT.
Working With ONNX Models With Named Input Dimensions	sampleNamedDimensions	An example of parsing an ONNX model with named input dimensions and building the engine for it.

1.1. Getting Started With C++ Samples

You can find the C++ samples in the `/usr/src/tensorrt/samples` package directory as well as on GitHub. The following C++ samples are shipped with TensorRT.

- "Hello World" For TensorRT From ONNX
- Building An RNN Network Layer By Layer
- Performing Inference In INT8 Precision
- Specifying I/O Formats
- Digit Recognition With Dynamic Shapes In TensorRT
- Algorithm Selection API Usage Example Based On sampleMNIST In TensorRT[1]
- Implementing CoordConv in TensorRT with a custom plugin using sampleOnnxMnistCoordConvAC In TensorRT
- Working With ONNX Models With Named Input Dimensions

Getting Started With C++ Samples

Every C++ sample includes a `README.md` file in GitHub that provides detailed information about how the sample works, sample code, and step-by-step instructions on how to run and verify its output.

Running C++ Samples on Linux

If you installed TensorRT using the Debian files, copy `/usr/src/tensorrt` to a new directory first before building the C++ samples. If you installed TensorRT using the tar file,

[1] This sample is located in the release product package only.

then the samples are located in `{TAR_EXTRACT_PATH}/samples`. To build all the samples and then run one of the samples, use the following commands:

```
$ cd <samples_dir>
$ make -j4
$ cd ../bin
$ ./<sample_bin>
```

Running C++ Samples on Windows

All of the C++ samples on Windows are provided as Visual Studio Solution files. To build a sample, open its corresponding Visual Studio Solution file and build the solution. The output executable will be generated in `(ZIP_EXTRACT_PATH)\bin`. You can then run the executable directly or through Visual Studio.

1.2. Getting Started With Python Samples

You can find the Python samples in the `/usr/src/tensorrt/samples/python` package directory. The following Python samples are shipped with TensorRT.

- Introduction To Importing ONNX Models Into TensorRT Using Python
- "Hello World" For TensorRT Using PyTorch And Python
- Writing a TensorRT Plugin to Use a Custom Layer in Your ONNX Model
- Object Detection With The ONNX TensorRT Backend In Python
- TensorRT Inference Of ONNX Models With Custom Layers In Python
- Refitting An Engine Built From An ONNX Model In Python
- Scalable And Efficient Object Detection With EfficientDet Networks In Python
- Scalable And Efficient Image Classification With EfficientNet Networks In Python
- Object Detection with TensorFlow Object Detection API Model Zoo Networks in Python
- Object Detection with Detectron 2 Mask R-CNN R50-FPN 3x Network in Python

Getting Started With Python Samples

Every C++ sample includes a `README.md` file in GitHub that provides detailed information about how the sample works, sample code, and step-by-step instructions on how to run and verify its output.

Running Python Samples

To run one of the Python samples, the process typically involves two steps:

1. Install the sample requirements:
   ```
   python<x> -m pip install -r requirements.txt
   ```
   where `python<x>` is either `python2` or `python3`.

2. Run the sample code with the `data` directory provided if the TensorRT sample data is not in the default location. For example:

```
python<x> sample.py [-d DATA_DIR]
```

For more information on running samples, refer to the `README.md` file included with the sample.

Chapter 2. Cross Compiling Samples

The following sections show how to cross-compile TensorRT samples for AArch64 QNX and Linux platforms under x86_64 Linux.

2.1. Prerequisites

This section provides step-by-step instructions to ensure you meet the minimum requirements to cross-compile.

1. Install the CUDA cross-platform toolkit for the corresponding target and set the environment variable `CUDA_INSTALL_DIR`.

   ```
   $ export CUDA_INSTALL_DIR="your cuda install dir"
   ```

   Where `CUDA_INSTALL_DIR` is set to `/usr/local/cuda` by default.

   Note: If you are installing TensorRT using the network repository, then it's best if you install the `cuda-toolkit-X-Y` and `cuda-cross-<arch>-X-Y` packages first to ensure you have all CUDA dependencies required to build the TensorRT samples.

2. Install the cuDNN cross-platform libraries for the corresponding target and set the environment variable `CUDNN_INSTALL_DIR`.

   ```
   $ export CUDNN_INSTALL_DIR="your cudnn install dir"
   ```

   Where `CUDNN_INSTALL_DIR` is set to `CUDA_INSTALL_DIR` by default.

3. Install the TensorRT cross-compilation Debian packages for the corresponding target.

   Note: If you are using the tar file release for the target platform, then you can safely skip this step. The tar file release already includes the cross-compile libraries so no additional packages are required.

   QNX AArch64

   - `tensorrt-dev-cross-qnx`

   Linux AArch64

   - `tensorrt-dev-cross-aarch64`

Linux SBSA

- `tensorrt-dev-cross-sbsa`

2.2. Building Samples For QNX AArch64

This section provides step-by-step instructions to build samples for QNX users.

1. Download the QNX tool-chain and export the following environment variables.

```
$ export QNX_HOST=/path/to/your/qnx/toolchain/host/linux/x86_64
$ export QNX_TARGET=/path/to/your/qnx/toolchain/target/qnx7
```

2. Build the samples by issuing:

```
$ cd /path/to/TensorRT/samples
$ make TARGET=qnx
```

2.3. Building Samples For Linux SBSA

This section provides step-by-step instructions to build samples for Linux SBSA users.

1. Install the corresponding GCC compiler, `aarch64-linux-gnu-g++`. In Ubuntu, this can be installed using:

```
$ sudo apt-get install g++-aarch64-linux-gnu
```

2. Build the samples by issuing:

```
$ cd /path/to/TensorRT/samples
$ make TARGET=aarch64 ARMSERVER=1 DLSW_TRIPLE=aarch64-linux-gnu CUDA_TRIPLE=sbsa-linux
 CUBLAS_TRIPLE=sbsa-linux CUDA_INSTALL_DIR=<cuda-cross-dir> CUDNN_INSTALL_DIR=<cudnn-
cross-dir>
```

Chapter 3. Building Samples Using Static Libraries

The following section demonstrates how to build the TensorRT samples using the TensorRT static libraries, including cuDNN and other CUDA libraries that are statically linked. The TensorRT samples can be used as a guideline for how to build your own application using the TensorRT static libraries, if you choose.

Note: You must use the tar package if you wish to build the TensorRT samples statically because some libraries are not included in the Debian or RPM packages including some required dependent static libraries and linker scripts. Also, building the TensorRT samples statically is only supported on Linux x86 platforms and not AArch64 or PowerPC at this time.

To build the TensorRT samples using the TensorRT static libraries, you can use the following command when you are building the samples.

```
$ make TRT_STATIC=1
```

You should append any other Make arguments you would normally include, such as `TARGET` to indicate the CPU architecture or `CUDA_INSTALL_DIR` to indicate where CUDA has been installed on your system. The static sample binaries created by the `TRT_STATIC` make option will have the suffix `_static` appended to the filename in the output directory to distinguish them from the dynamic sample binaries.

3.1. Limitations

It is required that the same major.minor version of the CUDA toolkit that was used to build TensorRT is used to build your application. Since symbols cannot be hidden or duplicated in a static binary, like they can for dynamic libraries, using the same CUDA toolkit version reduces the chance of symbol conflicts, incompatibilities, or undesired behaviors.

If you are including `libnvinfer_static.a` and `libnvinfer_plugin_static.a` in your linker command line, then consider using the following linker flags to ensure that all CUDA kernels and TensorRT plug-ins are included in your final application.

```
-Wl,-whole-archive -lnvinfer_static -Wl,-no-whole-archive
-Wl,-whole-archive -lnvinfer_plugin_static -Wl,-no-whole-archive
```

When linking with the cuDNN static library, `libcudnn_static.a` should be linked with the following whole-archive linker flag for best possible performance. Refer to the NVIDIA cuDNN Release Notes for more information.

```
-Wl,-whole-archive -lcudnn_static -Wl,-no-whole-archive
```

For platforms where TensorRT was built with less than CUDA 11.6 or CUDA 11.4 on Linux x86_64, if `libnvrtc.so.*` cannot be found in the library search path, then TensorRT automatically disables some features that require NVRTC (see list below). If these features are required for your application, then you must provide the NVRTC library at runtime.

- Loops
- Boolean operations
- PointWise fusions
- Fusions that depend on PointWise fusion. For example, Convolution or FullyConnected operations fused with the subsequent PointWise operation.

When `libnvrtc_static.a`, `libnvrtc-builtins_static.a` or `libnvptxcompiler_static.a` is present in the CUDA Toolkit, it is required to link to the TensorRT static libraries. Refer to the NVIDIA NVRTC User Guide for more information.

If you are building the TensorRT samples with a GCC version less than 8.x, then you may require the RedHat Developer Toolset 8 non-shared libstdc++ library to avoid missing C++ standard library symbols during linking. You can use the following one-line command to obtain this additional static library, assuming the programs required by this command are already installed on your system.

```
$ curl -s http://mirror.centos.org/centos/7/sclo/x86_64/rh/Packages/d/devtoolset-8-libstdc++-devel-8.3.1-3.2.el7.x86_64.rpm | rpm2cpio - | bsdtar --strip-components=10 -xf - '*/libstdc++_nonshared.a'
```

If you are building the TensorRT samples with a GCC version less than 5.x (for example GCC 4.8 on RHEL/CentOS 7.x), then you may require the linker options mentioned below to ensure you are using the correct C++ standard library symbols in your application. Your application object files must come after the TensorRT static libraries and whole-archive all TensorRT static libraries when linking to ensure the newer C++ standard library symbols from the RedHat Developer Toolset are used. This change is required to avoid undefined behavior within TensorRT that may lead to a crash. Since the resulting binary will of course depends on TensorRT, both the TensorRT static libraries and any dependent object files must be linked together as a group to ensure that all symbols are resolved.

```
-Wl,--start-group -Wl,-whole-archive -lnvinfer_static -lnvinfer_plugin_static -lnvparsers_static -lnvonnxparser_static -Wl,-no-whole-archive <object_files> -Wl,--end-group
```

You may observe relocation issues during linking if the resulting binary exceeds 2 GB. This can occur if you are linking TensorRT and all of its dependencies into your application statically. To workaround this issue and move the GPU code to the end of the binary. You may require the linker script below and the following linker option `-Wl,<path/to/fatbin.ld>`.

```
SECTIONS
{
  .nvFatBinSegment : { *(.nvFatBinSegment) }
  .nv_fatbin : { *(.nv_fatbin) }
}
```

3.2. Linking With the cuDNN, cuBLAS, and cuBLASLt Stub Static Library

For the TensorRT core library, the cuDNN, cuBLAS, and cuBLASLt libraries are disabled by default. If the plugins used do not need cuDNN, cuBLAS, and cuBLASLt, to reduce CPU and GPU memory usage, the stubbed static library of cuDNN, cuBLAS, and cuBLASLt can be used to resolve symbols without linking to a real library. Add `USE_STUB_EXTERNALS=1` along with `TRT_STATIC=1` when building samples to allow the use of the stubbed library.

The stubbed cuDNN, cuBLAS, and cuBLASLt are stored in `$(TRT_LIB_DIR)/stubs`. The names of stubbed static libraries are:

- `libcudnn_static_stub_trt.a`
- `libcublas_static_stub_trt.a`
- `libcublasLt_static_stub_trt.a`

Chapter 4. Machine Comprehension

Machine comprehension systems are used to translate text from one language to another language, make predictions or answer questions based on a specific context. Recurrent neural networks (RNN) are one of the most popular deep learning solutions for machine comprehension.

Some examples of TensorRT machine comprehension samples include the following:

- Building An RNN Network Layer By Layer
- Refitting An Engine Built From An ONNX Model In Python
- Writing a TensorRT Plugin to Use a Custom Layer in Your ONNX Model

4.1. Building An RNN Network Layer By Layer

This sample, sampleCharRNN, uses the TensorRT API to build an RNN network layer by layer, sets up weights and inputs/outputs and then performs inference.

What does this sample do?

Specifically, this sample creates a CharRNN network that has been trained on the Tiny Shakespeare dataset. For more information about character level modeling, refer to char-rnn.

TensorFlow has a useful RNN Tutorial which can be used to train a word-level model. Word level models learn a probability distribution over a set of all possible word sequences. Since our goal is to train a char level model, which learns a probability distribution over a set of all possible characters, a few modifications will need to be made to get the TensorFlow sample to work. These modifications can be seen here.

Where is this sample located?

This sample is maintained under the `samples/sampleCharRNN` directory in the GitHub: sampleCharRNN repository. If using the Debian or RPM package, the sample is located at `/usr/src/tensorrt/samples/sampleCharRNN`. If using the tar or zip package, the sample is at `<extracted_path>/samples/sampleCharRNN`.

How do I get started?

For more information about getting started, refer to Getting Started With C++ Samples. For specifics about this sample, refer to the GitHub: sampleCharRNN/README.md file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

4.2. Refitting An Engine Built From An ONNX Model In Python

This sample, engine_refit_onnx_bidaf, builds an engine from the ONNX BiDAF model, and refits the TensorRT engine with weights from the model. The new refit APIs allow users to locate the weights via names from ONNX models instead of layer names and weights roles.

In the first pass, the weights "Parameter576_B_0" are refitted with empty values resulting in an incorrect inference result. In the second pass, we refit the engine with the actual weights and run inference again. With the weights now set correctly, inference should provide correct results.

Where Is This Sample Located?

This sample is maintained under the `samples/python/engine_refit_onnx_bidaf` directory in the GitHub: engine_refit_onnx_bidaf repository. If using the Debian or RPM package, the sample is located at `/usr/src/tensorrt/samples/python/engine_refit_onnx_bidaf`. If using the tar or zip package, the sample is at `<extracted_path>/samples/python/engine_refit_onnx_bidaf`.

Getting Started:

For more information about getting started, refer to Getting Started With Python Samples. For specifics about this sample, refer to the GitHub: engine_refit_onnx_bidaf/README.md file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

4.3. Writing a TensorRT Plugin to Use a Custom Layer in Your ONNX Model

This sample, onnx_custom_plugin, demonstrates how to use plugins written in C++ to run TensorRT on ONNX models with custom or unsupported layers. This sample implements a Hardmax layer and uses it to run a BiDAF question-answering model using the TensorRT ONNX Parser and Python API.

Where Is This Sample Located?

This sample is maintained under the `samples/python/onnx_custom_plugin` directory in the GitHub: onnx_custom_plugin repository. If using the Debian or RPM package, the sample is located at `/usr/src/tensorrt/samples/python/onnx_custom_plugin`. If using the tar or zip package, the sample is at `<extracted_path>/samples/python/onnx_custom_plugin`.

Getting Started:

For more information about getting started, refer to Getting Started With Python Samples. For specifics about this sample, refer to the GitHub: /onnx_custom_plugin/README.md file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

Chapter 5. Character Recognition

Character recognition, especially on the MNIST dataset, is a classic machine learning problem. The MNIST problem involves recognizing the digit that is present in an image of a handwritten digit.

Some examples of TensorRT character recognition samples include the following:

- "Hello World" For TensorRT From ONNX
- Digit Recognition With Dynamic Shapes In TensorRT
- Specifying I/O Formats
- "Hello World" For TensorRT Using PyTorch And Python
- Algorithm Selection API Usage Example Based On sampleMNIST In TensorRT
- Implementing CoordConv in TensorRT with a custom plugin using sampleOnnxMnistCoordConvAC In TensorRT

5.1. "Hello World" For TensorRT From ONNX

This sample, sampleOnnxMNIST, converts a model trained on the MNIST in ONNX format to a TensorRT network and runs inference on the network. ONNX is a standard for representing deep learning models that enables models to be transferred between frameworks.

Where is this sample located?

This sample is maintained under the `samples/sampleOnnxMNIST` directory in the GitHub: sampleOnnxMNIST repository. If using the Debian or RPM package, the sample is located at `/usr/src/tensorrt/samples/sampleOnnxMNIST`. If using the tar or zip package, the sample is at `<extracted_path>/samples/sampleOnnxMNIST`.

How do I get started?

For more information about getting started, refer to Getting Started With C++ Samples. For specifics about this sample, refer to the GitHub: sampleOnnxMNIST/README.md file

for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

5.2. Digit Recognition With Dynamic Shapes In TensorRT

This sample, sampleDynamicReshape, demonstrates how to use dynamic input dimensions in TensorRT by creating an engine for resizing dynamically shaped inputs to the correct size for an ONNX MNIST model.

What does this sample do?

This sample creates an engine for resizing an input with dynamic dimensions to a size that an ONNX MNIST model can consume.

Specifically, this sample demonstrates how to:

- Create a network with dynamic input dimensions to act as a preprocessor for the model
- Parse an ONNX MNIST model to create a second network
- Build engines for both networks and start calibration if running in INT8
- Run inference using both engines

For more information, refer to Working With Dynamic Shapes.

Where is this sample located?

This sample is maintained under the `samples/sampleDynamicReshape` directory in the GitHub: sampleDynamicReshape repository. If using the Debian or RPM package, the sample is located at `/usr/src/tensorrt/samples/sampleDynamicReshape`. If using the tar or zip package, the sample is at `<extracted_path>/samples/sampleDynamicReshape`.

How do I get started?

For more information about getting started, refer to Getting Started With C++ Samples. For specifics about this sample, refer to the GitHub: sampleDynamicReshape/README.md file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

5.3. Specifying I/O Formats

This sample, sampleIOFormats, uses an ONNX model that was trained on the MNIST dataset and performs engine building and inference using TensorRT. The correctness of outputs is then compared to the golden reference.

What does this sample do?

Specifically, it shows how to explicitly specify I/O formats for `TensorFormat::kLINEAR`, `TensorFormat::kCHW2` and `TensorFormat::kHWC8` for Float16 and INT8 precision.

`ITensor::setAllowedFormats` is invoked to specify which format is used.

Where is this sample located?

This sample is maintained under the directory `samples/sampleIOFormats` in the GitHub: sampleIOFormats repository. If using the Debian or RPM package, the sample is located at `/usr/src/tensorrt/samples/sampleIOFormats`. If using the tar or zip package, the sample is at `<extracted_path>/samples/sampleIOFormats`.

How do I get started?

For more information about getting started, refer to Getting Started With C++ Samples. For specifics about this sample, refer to the GitHub: sampleIOFormats/README.md file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

5.4. "Hello World" For TensorRT Using PyTorch And Python

This sample, network_api_pytorch_mnist, trains a convolutional model on the MNIST dataset and runs inference with a TensorRT engine.

Where Is This Sample Located?

This sample is maintained under the `samples/python/network_api_pytorch_mnist` directory in the GitHub: network_api_pytorch_mnist repository. If using the Debian or RPM package, the sample is located at `/usr/src/tensorrt/samples/python/network_api_pytorch`. If using the tar or zip package, the sample is at `<extracted_path>/samples/python/network_api_pytorch`.

Getting Started:

For more information about getting started, refer to Getting Started With Python Samples. For specifics about this sample, refer to the GitHub: /network_api_pytorch_mnist/README.md file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

5.5. Algorithm Selection API Usage Example Based On sampleMNIST In TensorRT

This sample, sampleAlgorithmSelector, shows an example of how to use the algorithm selection API based on sampleMNIST.

What does this sample do?

This sample demonstrates the usage of `IAlgorithmSelector` to deterministically build TensorRT engines. It also shows the usage of `IAlgorithmSelector::selectAlgorithms` to define heuristics for selection of algorithms.

This sample uses a Caffe model that was trained on the MNIST dataset.

To verify whether the engine is operating correctly, this sample picks a 28x28 image of a digit at random and runs inference on it using the engine it created. The output of the network is a probability distribution on the digit, showing which digit is likely to be that in the image.

Where is this sample located?

This sample is maintained under the `samples/sampleAlgorithmSelector` directory in the GitHub: sampleAlgorithmSelector repository. If using the Debian or RPM package, the sample is located at `/usr/src/tensorrt/samples/sampleAlgorithmSelector`. If using the tar or zip package, the sample is at `<extracted_path>/samples/sampleAlgorithmSelector`.

How do I get started?

For more information about getting started, refer to Getting Started With C++ Samples. For specifics about this sample, refer to the GitHub: /uff_custom_plugin/README.md file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

5.6. Implementing CoordConv in TensorRT with a custom plugin using sampleOnnxMnistCoordConvAC In TensorRT

This sample, sampleOnnxMnistCoordConvAC, converts a model trained on the MNIST dataset in Open Neural Network Exchange (ONNX) format to a TensorRT network and runs inference on the network. This model was trained in PyTorch and it contains custom CoordConv layers instead of Conv layers.

The model with the CoordConvAC layers training script and code of the CoordConv layers in PyTorch are here. The original model with the Conv layers is here.

This sample creates and runs a TensorRT engine on an ONNX model of MNIST trained with CoordConv layers. It demonstrates how TensorRT can parse and import ONNX models, as well as use plugins to run custom layers in neural networks.

Where is this sample located?

This sample is maintained under the `samples/sampleOnnxMnistCoordConvAC` directory in the GitHub:sampleOnnxMnistCoordConvAC repository. If using the Debian or RPM package, the sample is located at `/usr/src/tensorrt/samples/sampleOnnxMnistCoordConvAC`. If using the tar or zip package, the sample is at `<extracted_path>/samples/sampleOnnxMnistCoordConvAC`.

How do I get started?

For more information about getting started, refer to Getting Started With C++ Samples. For specifics about this sample, refer to the GitHub:/sampleOnnxMnistCoordConvAC/README.mdfile for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

Chapter 6. Image Classification

Image classification is the problem of identifying one or more objects present in an image. Convolutional neural networks (CNN) are a popular choice for solving this problem. They are typically composed of convolution and pooling layers.

Some examples of TensorRT image classification samples include the following:

- Performing Inference In INT8 Precision
- Introduction To Importing ONNX Models Into TensorRT Using Python
- TensorRT Inference Of ONNX Models With Custom Layers In Python
- Scalable And Efficient Image Classification With EfficientNet Networks In Python

6.1. Performing Inference In INT8 Precision

This sample, sampleINT8API, performs INT8 inference without using the INT8 calibrator; using the user-provided per activation tensor dynamic range. INT8 inference is available only on GPUs with compute capability 6.1 or 7.x and supports Image Classification ONNX models such as ResNet-50, VGG19, and MobileNet.

What does this sample do?

Specifically, this sample demonstrates how to:

- Use `nvinfer1::ITensor::setDynamicRange` to set per tensor dynamic range
- Use `nvinfer1::ILayer::setPrecison` to set computation precision of a layer
- Use `nvinfer1::ILayer::setOutputType` to set output tensor data type of a layer
- Perform INT8 inference without using INT8 calibration

Where is this sample located?

This sample is maintained under the `samples/sampleINT8API` directory in the GitHub: sampleINT8API repository. If using the Debian or RPM package, the sample is located at `/usr/src/tensorrt/samples/sampleINT8API`. If using the tar or zip package, the sample is at `<extracted_path>/samples/sampleINT8API`.

How do I get started?

For more information about getting started, refer to Getting Started With C++ Samples. For specifics about this sample, refer to the GitHub: sampleINT8API/README.md file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

6.2. Introduction To Importing ONNX Models Into TensorRT Using Python

This sample, introductory_parser_samples, is a Python sample that uses TensorRT and its included ONNX parser, to perform inference with ResNet-50 models trained with various different frameworks.

Where Is This Sample Located?

This sample is maintained under the `samples/python/introductory_parser_samples` directory in the GitHub: introductory_parser_samples repository. If using the Debian or RPM package, the sample is located at `/usr/src/tensorrt/samples/python/introductory_parser_samples`. If using the tar or zip package, the sample is at `<extracted_path>/samples/python/introductory_parser_samples`.

Getting Started:

For more information about getting started, refer to Getting Started With Python Samples. For specifics about this sample, refer to the GitHub: introductory_parser_samples/README.md file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

6.3. TensorRT Inference Of ONNX Models With Custom Layers In Python

This sample, onnx_packnet, uses TensorRT to perform inference with the PackNet network. PackNet is a self-supervised monocular depth estimation network used in autonomous driving.

What does this sample do?

This sample converts the PyTorch graph into ONNX and uses an ONNX-parser included in TensorRT to parse the ONNX graph. The sample also demonstrates how to:

- Use custom layers (plugins) in an ONNX graph. These plugins can be automatically registered in TensorRT by using `REGISTER_TENSORRT_PLUGIN` API.
- Use the ONNX GraphSurgeon (ONNX-GS) API to modify layers or subgraphs in the ONNX graph. For this network, we transform Group Normalization, upsample and pad layers to remove unnecessary nodes for inference with TensorRT.

Where is this sample located?

This sample is maintained under the `samples/python/onnx_packnet` directory in the GitHub: onnx_packnet repository. If using the Debian or RPM package, the sample is located at `/usr/src/tensorrt/samples/python/onnx_packnet`. If using the tar or zip package, the sample is at `<extracted_path>/samples/python/onnx_packnet`.

How do I get started?

For more information about getting started, refer to Getting Started With Python Samples. For specifics about this sample, refer to the GitHub: onnx_packnet/README.md file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

6.4. Scalable And Efficient Image Classification With EfficientNet Networks In Python

This sample, efficientnet, shows how to convert and execute a Google EfficientNet model with TensorRT.

What does this sample do?

The sample supports models from the original EfficientNet implementation, as well as newer EfficientNet V2 models. The sample code converts a TensorFlow saved model to ONNX and then builds a TensorRT engine with it. Inference and accuracy validation can also be performed with the helper scripts provided in the sample.

Where is this sample located?

This sample is maintained under the `samples/python/efficientnet` directory in the GitHub: efficientnet repository. If using the Debian or RPM package, the sample is located at `/usr/src/tensorrt/samples/python/efficientnet`. If using the tar or zip package, the sample is at `<extracted_path>/samples/python/efficientnet`.

How do I get started?

For more information about getting started, refer to Getting Started With Python Samples. For specifics about this sample, refer to the GitHub: efficientnet/README.md file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

Chapter 7. Object Detection

Object detection is one of the classic computer vision problems. The task, for a given image, is to detect, classify and localize all objects of interest. For example, imagine that you are developing a self-driving car and you need to do pedestrian detection - the object detection algorithm would then, for a given image, return bounding box coordinates for each pedestrian in an image.

There have been many advances in recent years in designing models for object detection.

Some examples of TensorRT object detection samples include the following:

- Object Detection With The ONNX TensorRT Backend In Python
- Scalable And Efficient Object Detection With EfficientDet Networks In Python
- Object Detection with TensorFlow Object Detection API Model Zoo Networks in Python
- Object Detection with Detectron 2 Mask R-CNN R50-FPN 3x Network in Python

7.1. Object Detection With The ONNX TensorRT Backend In Python

This sample, yolov3_onnx, implements a full ONNX-based pipeline for performing inference with the YOLOv3 network, with an input size of 608x608 pixels, including pre and post-processing.

What Does This Sample Do?

This sample is based on the YOLOv3-608 paper.

Note: This sample is not supported on Ubuntu 14.04 and older. Additionally, the `yolov3_to_onnx.py` script does not support Python 3.

Where Is This Sample Located?

This sample is maintained under the `samples/python/yolov3_onnx` directory in the GitHub: yolov3_onnx repository. If using the Debian or RPM package, the sample is

located at `/usr/src/tensorrt/samples/python/yolov3_onnx`. If using the tar or zip package, the sample is at `<extracted_path>/samples/python/yolov2_onnx`.

Getting Started:

For more information about getting started, refer to Getting Started With Python Samples. For specifics about this sample, refer to the GitHub: yolov3_onnx/README.md file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

7.2. Scalable And Efficient Object Detection With EfficientDet Networks In Python

This sample, efficientdet, demonstrates the conversion and execution of Google EfficientDet models with TensorRT.

What does this sample do?

The code converts a TensorFlow checkpoint or saved model to ONNX, adapts the ONNX graph for TensorRT compatibility, and then builds a TensorRT engine with it. Inference and accuracy validation can then be performed using the corresponding scripts provided in the sample.

Where is this sample located?

This sample is maintained under the `samples/python/efficientdet` directory in the GitHub: efficientdet repository. If using the Debian or RPM package, the sample is located at `/usr/src/tensorrt/samples/python/efficientdet`. If using the tar or zip package, the sample is at `<extracted_path>/samples/python/efficientdet`

How do I get started?

For more information about getting started, refer to Getting Started With Python Samples. For specifics about this sample, refer to the GitHub: efficientdet/README.md file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

7.3. Object Detection with TensorFlow Object Detection API Model Zoo Networks in Python

This sample, tensorflow_object_detection_api, demonstrates the conversion and execution of the Tensorflow Object Detection API Model Zoo models with TensorRT.

What does this sample do?

The code converts a TensorFlow checkpoint or saved model to ONNX, adapts the ONNX graph for TensorRT compatibility, and then builds a TensorRT engine with it. Inference and accuracy validation can then be performed using the corresponding scripts provided in the sample.

Where is this sample located?

This sample is maintained under the `samples/python/tensorflow_object_detection_api` directory in the GitHub: tensorflow_object_detection_api repository. If using the Debian or RPM package, the sample is located at `/usr/src/tensorrt/samples/python/tensorflow_object_detection_api`. If using the tar or zip package, the sample is at `<extracted_path>/samples/python/tensorflow_object_detection_api`.

How do I get started?

For more information about getting started, refer to Getting Started With Python Samples. For specifics about this sample, refer to the GitHub: tensorflow_object_detection_api/README.md file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

7.4. Object Detection with Detectron 2 Mask R-CNN R50-FPN 3x Network in Python

This sample, detectron2, demonstrates the conversion and execution of Detectron 2 Model Zoo Mask R-CNN R50-FPN 3x model with TensorRT.

What does this sample do?

The project provides steps to export Detectron 2 model to ONNX, code adapts the ONNX graph for TensorRT compatibility, and then builds a TensorRT engine with it. Inference and accuracy validation can then be performed using the corresponding scripts provided in the sample.

Where is this sample located?

This sample is maintained under the `samples/python/detectron2` directory in the GitHub: detectron2 repository. If using the Debian or RPM package, the sample is located at `/usr/src/tensorrt/samples/python/detectron2`. If using the tar or zip package, the sample is at `<extracted_path>/samples/python/detectron2`.

How do I get started?

For more information about getting started, refer to Getting Started With Python Samples. For specifics about this sample, refer to the GitHub: detectron2/README.md file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

Note: This sample cannot be run on Jetson platforms as `torch.distributed` is not available. To check whether your platform supports `torch.distributed`, open a Python shell and confirm that `torch.distributed.is_available()` returns `True`.

Chapter 8. Other Features

8.1. Working With ONNX Models With Named Input Dimensions

This sample, sampleNamedDimensions, illustrates the feature of named input dimensions.

What does this sample do?

Specifically, a simple one-layer ONNX model with named dimension parameters in the model input is generated and then passed to TensorRT for parsing and engine building.

Where is this sample located?

This sample is maintained under the `samples/sampleNamedDimensions` directory in the GitHub: sampleNamedDimensions repository. If using the Debian or RPM package, the sample is located at `/usr/src/tensorrt/samples/sampleNamedDimensions`. If using the tar or zip package, the sample is at `<extracted_path>/samples/sampleNamedDimensions`.

How do I get started?

For more information about getting started, refer to Getting Started With C++ Samples. For specifics about this sample, refer to the GitHub: sampleNamedDimensions/README.md file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation (“NVIDIA”) makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer (“Terms of Sale”). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer’s own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, “MATERIALS”) ARE BEING PROVIDED “AS IS.” NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA’s aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Arm

Arm, AMBA and Arm Powered are registered trademarks of Arm Limited. Cortex, MPCore and Mali are trademarks of Arm Limited. "Arm" is used to represent Arm Holdings plc; its operating company Arm Limited; and the regional subsidiaries Arm Inc.; Arm KK; Arm Korea Limited.; Arm Taiwan Limited; Arm France SAS; Arm Consulting (Shanghai) Co. Ltd.; Arm Germany GmbH; Arm Embedded Technologies Pvt. Ltd.; Arm Norway, AS and Arm Sweden AB.

HDMI

HDMI, the HDMI logo, and High-Definition Multimedia Interface are trademarks or registered trademarks of HDMI Licensing LLC.

Blackberry/QNX

Google

Android, Android TV, Google Play and the Google Play logo are trademarks of Google, Inc.

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com

Trademarks

NVIDIA, the NVIDIA logo, and BlueField, CUDA, DALI, DRIVE, Hopper, JetPack, Jetson AGX Xavier, Jetson Nano, Maxwell, NGC, Nsight, Orin, Pascal, Quadro, Tegra, TensorRT, Triton, Turing and Volta are trademarks and/or registered trademarks of NVIDIA Corporation in the United States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com