NVIDIA TensorRT Samples

Support Guide | NVIDIA Docs

TRM-10259-001_v8.0.2 | August 2021

Table of Contents

Chapter 1. Introduction

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

Title	TensorRT Sample Name	Description
trtexec	trtexec	A tool to quickly utilize TensorRT without having to develop your own application.
"Hello World" For TensorRT	sampleMNIST	Performs the basic setup and initialization of TensorRT using the Caffe parser.
Building A Simple MNIST Network Layer By Layer	sampleMNISTAPI	Uses the TensorRT API to build an MNIST (handwritten digit recognition) layer by layer, sets up weights and inputs/outputs and then performs inference.
Importing The TensorFlow Model And Running Inference	sampleUffMNIST	Imports a TensorFlow model trained on the MNIST dataset.
"Hello World" For TensorRT From ONNX	sampleOnnxMNIST	Converts a model trained on the MNIST dataset in ONNX format to a TensorRT network.
Building And Running GoogleNet In TensorRT	sampleGoogleNet	Shows how to import a model trained with Caffe into TensorRT using GoogleNet as an example.
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 Using Custom Calibration	sampleINT8	Performs INT8 calibration and inference. Calibrates a network for execution in INT8.
Performing Inference In INT8 Precision	sampleINT8API	Sets per tensor dynamic range and computation precision of a layer.
Object Detection With Faster R-CNN	sampleFasterRCNN	Uses TensorRT plugins, performs inference and implements a fused custom

Title	TensorRT Sample Name	Description
		layer for end-to-end inferencing of a Faster R-CNN model.
Object Detection With A TensorFlow SSD Network	sampleUffSSD	Preprocesses the TensorFlow SSD network, performs inference on the SSD network in TensorRT and uses TensorRT plugins to speed up inference.
Object Detection With SSD	sampleSSD	Preprocesses the input to the SSD network, performs inference on the SSD network in TensorRT, uses TensorRT plugins to speed up inference, and performs INT8 calibration on an SSD network.
"Hello World" For Multilayer Perceptron (MLP)	sampleMLP	Shows how to create a network that triggers the multi-layer perceptron (MLP) optimizer.
Specifying I/O Formats Using The Reformat Free I/O APIs	sampleReformatFreeIO	Uses a Caffe 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.
Adding A Custom Layer That Supports INT8 I/O To Your Network In TensorRT	sampleUffPluginV2Ext	Demonstrates how to extend INT8 I/O for a plugin that is introduced in TensorRT 6.x.x.
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.
Neural Machine Translation (NMT) Using A Sequence To Sequence (seq2seq) Model	sampleNMT	Demonstrates the implementation of Neural Machine Translation (NMT) based on a TensorFlow seq2seq model using the TensorRT API.
Object Detection And Instance Segmentation With A TensorFlow Mask R-CNN Network	sampleUffMaskRCNN	Performs inference on the Mask R-CNN network in TensorRT. Mask R-CNN is based on the Mask R-CNN paper which performs the task of object detection and object mask predictions on a target image.
Object Detection With A TensorFlow Faster R-CNN Network	sampleUffFasterRCNN	Serves as a demo of how to use a pre-trained Faster-RCNN model in Transfer Learning

Title	TensorRT Sample Name	Description
		Toolkit to do inference with TensorRT.
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.
Introduction To Importing Caffe, TensorFlow And ONNX Models Into TensorRT Using Python	introductory_parser_samples	Uses TensorRT and its included suite of parsers (the UFF, Caffe and ONNX parsers), to perform inference with ResNet-50 models trained with various different frameworks.
"Hello World" For TensorRT Using TensorFlow And Python	end_to_end_tensorflow_mnist	An end-to-end sample that trains a model in TensorFlow and Keras, freezes the model and writes it to a protobuf file, converts it to UFF, and finally runs inference using TensorRT.
"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.
Adding A Custom Layer To Your TensorFlow Network In TensorRT In Python	uff_custom_plugin	Implements a clip layer (as a CUDA kernel) wraps the implementation in a TensorRT plugin (with a corresponding plugin creator), and generates a shared library module containing its code.
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.
Object Detection With SSD In Python	uff_ssd	Implements a full UFF-based pipeline for performing inference with an SSD (InceptionV2 feature extractor) network. The sample downloads a trained `ssd_inception_v2_coco_2017_11_17` model and uses it to perform inference. Additionally, it superimposes bounding boxes on the input image as a post-processing step.

Title	TensorRT Sample Name	Description
INT8 Calibration In Python	int8_caffe_mnist	Demonstrates how to calibrate an engine to run in INT8 mode.
Refitting An Engine In Python	engine_refit_mnist	Trains an MNIST model in PyTorch, recreates the network in TensorRT with dummy weights, and finally refits the TensorRT engine with weights from the model.
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 NVIDIA TensorRT.

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
- Building A Simple MNIST Network Layer By Layer
- Importing The TensorFlow Model And Running Inference
- "Hello World" For TensorRT From ONNX
- Building And Running GoogleNet In TensorRT
- Building An RNN Network Layer By Layer
- Performing Inference In INT8 Using Custom Calibration
- Performing Inference In INT8 Precision
- Object Detection With Faster R-CNN
- Object Detection With A TensorFlow SSD Network
- Object Detection With SSD
- "Hello World" For Multilayer Perceptron (MLP)
- Specifying I/O Formats Using The Reformat Free I/O APIs

- Adding A Custom Layer That Supports INT8 I/O To Your Network In TensorRT
- Digit Recognition With Dynamic Shapes In TensorRT
- Neural Machine Translation (NMT) Using A Sequence To Sequence (seq2seq) Model
- Object Detection And Instance Segmentation With A TensorFlow Mask R-CNN Network
- Object Detection With A TensorFlow Faster R-CNN Network
- Algorithm Selection API Usage Example Based On sampleMNIST In TensorRT[1]

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, 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 Caffe, TensorFlow And ONNX Models Into TensorRT Using Python
- "Hello World" For TensorRT Using TensorFlow And Python
- "Hello World" For TensorRT Using PyTorch And Python
- Adding A Custom Layer To Your TensorFlow Network In TensorRT In Python
- Object Detection With The ONNX TensorRT Backend In Python
- Object Detection With SSD In Python

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

- INT8 Calibration In Python
- Refitting An Engine 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

Getting Started With Python Samples

Every Python sample includes a `README.md` file. Refer to the `/usr/src/tensorrt/samples/python/<sample-name>/README.md` file for 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, see the `README.md` file included with the sample.

Chapter 2. Cross Compiling Samples For AArch64 Users

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.

Procedure

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

   - `libnvinfer-dev-cross-qnx`

- libnvinfer8-cross-qnx
- libnvinfer-plugin-dev-cross-qnx
- libnvinfer-plugin8-cross-qnx
- libnvparsers-dev-cross-qnx
- libnvparsers8-cross-qnx
- libnvonnxparsers-dev-cross-qnx
- libnvonnxparsers8-cross-qnx

Linux AArch64

- libnvinfer-dev-cross-aarch64
- libnvinfer8-cross-aarch64
- libnvinfer-plugin-dev-cross-aarch64
- libnvinfer-plugin8-cross-aarch64
- libnvparsers-dev-cross-aarch64
- libnvparsers8-cross-aarch64
- libnvonnxparsers-dev-cross-aarch64
- libnvonnxparsers8-cross-aarch6

Linux SBSA

- libnvinfer-dev-cross-sbsa
- libnvinfer8-cross-sbsa
- libnvinfer-plugin-dev-cross-sbsa
- libnvinfer-plugin8-cross-sbsa
- libnvparsers-dev-cross-sbsa
- libnvparsers8-cross-sbsa
- libnvonnxparsers-dev-cross-sbsa
- libnvonnxparsers8-cross-sbsa

2.2. Building Samples For QNX AArch64

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

Procedure

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

```
$ export QNX_HOST=/path/to/your/qnx/toolchains/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 AArch64

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

Procedure

1. Install the corresponding GCC compiler, `aarch64-linux-gnu-g++`. In Ubuntu, this can be installed via:
   ```
   $ sudo apt-get install g++-aarch64-linux-gnu
   ```
2. Build the samples by issuing:
   ```
   $ cd /path/to/TensorRT/samples
   $ make TARGET=aarch64
   ```

2.4. Building Samples For Linux SBSA

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

Procedure

1. Install the corresponding GCC compiler, `aarch64-linux-gnu-g++`. In Ubuntu, this can be installed via:
   ```
   $ sudo apt-get install g++-aarch64-linux-gnu
   ```
2. Build the samples by issuing:
   ```
   $ cd /path/to/TensorRT/samples
   $ make TARGET=aarch64 ARMSERVER=1
   ```

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.

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 USE_CUDART_STATIC=1
```

When building the TensorRT samples statically using the `TRT_STATIC=1` make option, the suffix `_static` will be appended to the output binary file name.

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

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 plugins 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 cuDNN 8.x.x Release Notes for more information.

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

If `libnvrtc.so.*` cannot be found in your library search path, then TensorRT will automatically disable some TensorRT features that require NVRTC to function (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.

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're using the correct C++ standard library symbols in your application. Your application object files must come after the 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 depend on TensorRT both the TensorRT static libraries and any dependent object files must be linked together as a group to ensure all symbols are resolved.

```
-Wl,--start-group -lnvinfer_static -lnvinfer_plugin_static -lnvparsers_static -lnvonnxparser_static <object_files> -Wl,--end-group
```

You may observe relocation issues during linking if the resulting binary exceeds 2GB. 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) }
}
```

Chapter 4. Recommenders

Recommender systems are used to provide product or media recommendations to users of social networking, media content consumption, and e-commerce platforms. MLP-based Neural Collaborative Filter (NCF) recommenders employ a stack of fully-connected or matrix multiplication layers to generate recommendations.

Some examples of TensorRT recommenders samples include the following:

- "Hello World" For Multilayer Perceptron (MLP)

4.1. "Hello World" For Multilayer Perceptron (MLP)

This sample, sampleMLP, is a simple hello world example that shows how to create a network that triggers the multilayer perceptron (MLP) optimizer. The generated MLP optimizer can then accelerate TensorRT.

Where is this sample located?

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

How do I get started?

For more information about getting started, see Getting Started With C++ Samples. For specifics about this sample, refer to the GitHub: sampleMLP/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. 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:

- Neural Machine Translation (NMT) Using A Sequence To Sequence (seq2seq) Model
- Building An RNN Network Layer By Layer
- Refitting An Engine Built From An ONNX Model In Python

5.1. Neural Machine Translation (NMT) Using A Sequence To Sequence (seq2seq) Model

This sample, sampleNMT, demonstrates the implementation of Neural Machine Translation (NMT) based on a TensorFlow seq2seq model using the TensorRT API. The TensorFlow seq2seq model is an open-sourced NMT project that uses deep neural networks to translate text from one language to another language.

What does this sample do?

Specifically, this sample is an end-to-end sample that takes a TensorFlow model, builds an engine, and runs inference using the generated network. The sample is intended to be modular so it can be used as a starting point for your machine translation application.

This sample implements German to English translation using the data that is provided by and trained from the TensorFlow NMT (seq2seq) Tutorial.

Where is this sample located?

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

How do I get started?

For more information about getting started, see Getting Started With C++ Samples. For specifics about this sample, refer to the GitHub: sampleNMT/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. 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, see 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, see 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.

5.3. 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, see 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.

Chapter 6. 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
- Building A Simple MNIST Network Layer By Layer
- Importing The TensorFlow Model And Running Inference
- "Hello World" For TensorRT From ONNX
- Performing Inference In INT8 Using Custom Calibration
- Digit Recognition With Dynamic Shapes In TensorRT
- Specifying I/O Formats Using The Reformat Free I/O APIs
- Adding A Custom Layer That Supports INT8 I/O To Your Network In TensorRT
- "Hello World" For TensorRT Using TensorFlow And Python
- Refitting An Engine In Python
- INT8 Calibration In Python
- "Hello World" For TensorRT Using PyTorch And Python
- Adding A Custom Layer To Your TensorFlow Network In TensorRT In Python
- Algorithm Selection API Usage Example Based On sampleMNIST In TensorRT

6.1. "Hello World" For TensorRT

This sample, sampleMNIST, is a simple hello world example that performs the basic setup and initialization of TensorRT using the Caffe parser.

Where is this sample located?

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

How do I get started?

For more information about getting started, see Getting Started With C++ Samples. For specifics about this sample, refer to the GitHub: sampleMNIST/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. Building A Simple MNIST Network Layer By Layer

This sample, sampleMNISTAPI, uses the TensorRT API to build an engine for a model trained on the MNIST dataset.

What does this sample do?

Specifically, it creates the network layer by layer, sets up weights and inputs/outputs, and then performs inference. This sample is similar to sampleMNIST. Both of these samples use the same model weights, handle the same input, and expect similar output.

Where is this sample located?

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

How do I get started?

For more information about getting started, see Getting Started With C++ Samples. For specifics about this sample, refer to the GitHub: sampleMNISTAPI/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. Importing The TensorFlow Model And Running Inference

This sample, sampleUffMNIST, imports a TensorFlow model trained on the MNIST dataset.

What does this sample do?

The MNIST TensorFlow model has been converted to UFF (Universal Framework Format) using the explanation described in Working With TensorFlow.

The UFF is designed to store neural networks as a graph. The NvUffParser that we use in this sample parses the UFF file in order to create an inference engine based on that neural network.

With TensorRT, you can take a TensorFlow trained model, export it into a UFF protobuf file (`.uff`) using the UFF converter, and import it using the UFF parser.

Where is this sample located?

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

How do I get started?

For more information about getting started, see Getting Started With C++ Samples. For specifics about this sample, refer to the GitHub: sampleUffMNIST/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. "Hello World" For TensorRT From ONNX

This sample, sampleOnnxMNIST, converts a model trained on the MNIST in Open Neural Network Exchange (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, see 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.

6.5. Performing Inference In INT8 Using Custom Calibration

This sample, sampleINT8, performs INT8 calibration and inference.

What does this sample do?

Specifically, this sample demonstrates how to perform inference in an 8-bit integer (INT8). INT8 inference is available only on GPUs with compute capability 6.1 or 7.x. After the network is calibrated for execution in INT8, the output of the calibration is cached to avoid repeating the process. You can then reproduce your own experiments with Caffe in order to validate your results on ImageNet networks.

Where is this sample located?

This sample is maintained under the `samples/sampleINT8` directory in the GitHub: sampleINT8 repository. If using the Debian or RPM package, the sample is located at `/usr/src/tensorrt/samples/sampleINT8`. If using the tar or zip package, the sample is at `<extracted_path>/samples/sampleINT8`.

How do I get started?

For more information about getting started, see Getting Started With C++ Samples. For specifics about this sample, refer to the GitHub: sampleINT8/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.

6.6. 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, see Working With Dynamic Shapes in the TensorRT Developer Guide.

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, see 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.

6.7. Specifying I/O Formats Using The Reformat Free I/O APIs

This sample, sampleReformatFreeIO, uses a Caffe 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, this sample shows how to use reformat free I/O APIs to explicitly specify I/O formats to `TensorFormat::kLINEAR`, `TensorFormat::kCHW2` and `TensorFormat::kHWC8` for Float16 and INT8 precision.

`ITensor::setAllowedFormats` is invoked to specify which format is expected to be supported so that the unnecessary reformatting will not be inserted to convert from/to FP32 formats for I/O tensors. `BuilderFlag::kSTRICT_TYPES` is also assigned to the builder configuration to let the builder choose a reformat free path rather than the fastest path.

Where is this sample located?

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

How do I get started?

For more information about getting started, see Getting Started With C++ Samples. For specifics about this sample, refer to the GitHub: sampleReformatFreeIO/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.

6.8. Adding A Custom Layer That Supports INT8 I/O To Your Network In TensorRT

This sample, sampleUffPluginV2Ext, implements the custom pooling layer for the MNIST model (`data/samples/lenet5_custom_pool.uff`).

What does this sample do?

Since cuDNN function `cudnnPoolingForward` with float precision is used to simulate an INT8 kernel, the performance for INT8 precision does not speed up. Nevertheless, the main purpose of this sample is to demonstrate how to extend INT8 I/O for a plugin that is introduced in TensorRT 6.0. This requires the interface replacement from `IPlugin/IPluginV2/IPluginV2Ext` to `IPluginV2IOExt` (or `IPluginV2DynamicExt` if dynamic shape is required).

Where is this sample located?

This sample is maintained under the `samples/sampleUffPluginV2Ext` directory in the GitHub: sampleUffPluginV2Ext repository. If using the Debian or RPM package, the sample is located at `/usr/src/tensorrt/samples/sampleUffPluginV2Ext`. If using the tar or zip package, the sample is at `<extracted_path>/samples/sampleUffPluginV2Ext`.

How do I get started?

For more information about getting started, see Getting Started With C++ Samples. For specifics about this sample, refer to the GitHub: /sampleUffPluginV2Ext/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.

6.9. “Hello World” For TensorRT Using TensorFlow And Python

This sample, end_to_end_tensorflow_mnist, trains a small, fully-connected model on the MNIST dataset and runs inference using TensorRT.

Where Is This Sample Located?

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

Getting Started:

For more information about getting started, see Getting Started With Python Samples. For specifics about this sample, refer to the GitHub: /end_to_end_tensorflow_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.

6.10. Refitting An Engine In Python

This sample, engine_refit_mnist, trains an MNIST model in PyTorch, recreates the network in TensorRT with dummy weights, and finally refits the TensorRT engine with weights from the model. Refitting allows us to quickly modify the weights in a TensorRT engine without needing to rebuild.

Where Is This Sample Located?

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

Getting Started:

For more information about getting started, see Getting Started With Python Samples. For specifics about this sample, refer to the GitHub: /engine_refit_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.

6.11. INT8 Calibration In Python

This sample, int8_caffe_mnist, demonstrates how to create an INT8 calibrator, build and calibrate an engine for INT8 mode, and finally run inference in INT8 mode.

Where Is This Sample Located?

This sample is maintained under the `samples/python/int8_caffe_mnist` directory in the GitHub: int8_caffe_mnist repository. If using the Debian or RPM package, the sample is located at `/usr/src/tensorrt/samples/python/int8_caffe_mnist`. If using the tar or zip package, the sample is at `<extracted_path>/samples/python/int8_caffe_mnist`.

Getting Started:

For more information about getting started, see Getting Started With Python Samples. For specifics about this sample, refer to the GitHub: /int8_caffe_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.

6.12. "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, see 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.

6.13. Adding A Custom Layer To Your TensorFlow Network In TensorRT In Python

This sample, uff_custom_plugin, demonstrates how to use plugins written in C++ with the TensorRT Python bindings and UFF Parser. This sample uses the MNIST dataset.

Where Is This Sample Located?

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

Getting Started:

For more information about getting started, see Getting Started With Python 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.

6.14. 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, see 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.

Chapter 7. 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:

- Building And Running GoogleNet In TensorRT
- Performing Inference In INT8 Precision
- Introduction To Importing Caffe, TensorFlow And ONNX Models Into TensorRT Using Python
- TensorRT Inference Of ONNX Models With Custom Layers In Python

7.1. Building And Running GoogleNet In TensorRT

This sample, sampleGoogleNet, demonstrates how to import a model trained with Caffe into TensorRT using GoogleNet as an example.

What does this sample do?

Specifically, this sample builds a TensorRT engine from the saved Caffe model, sets input values to the engine, and runs it.

Where is this sample located?

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

How do I get started?

For more information about getting started, see Getting Started With C++ Samples. For specifics about this sample, refer to the GitHub: sampleGoogleNet/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. 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, see 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.

7.3. Introduction To Importing Caffe, TensorFlow And ONNX Models Into TensorRT Using Python

This sample, introductory_parser_samples, is a Python sample that uses TensorRT and its included suite of parsers (UFF, Caffe and ONNX parsers), 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, see 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.

7.4. 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, see 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.

Chapter 8. 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 SSD In Python
- Object Detection With The ONNX TensorRT Backend In Python
- Object Detection With A TensorFlow SSD Network
- Object Detection With Faster R-CNN
- Object Detection With SSD
- Object Detection And Instance Segmentation With A TensorFlow Mask R-CNN Network
- Object Detection With A TensorFlow Faster R-CNN Network
- Scalable And Efficient Object Detection With EfficientDet Networks In Python

8.1. Object Detection With SSD In Python

This sample, uff_ssd, implements a full UFF-based pipeline for performing inference with an SSD (InceptionV2 feature extractor) network.

What Does This Sample Do?

This sample is based on the SSD: Single Shot MultiBox Detector paper. The SSD network, built on the VGG-16 network, performs the task of object detection and localization in a single forward pass of the network. This approach discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. At prediction time, the network generates scores for the presence of each object category in each default box and produces adjustments to the box to better match the object shape. Additionally, the network combines predictions from multiple features with different resolutions to naturally handle objects of various sizes.

This sample is based on the TensorFlow implementation of SSD. For more information, download ssd_inception_v2_coco. Unlike the paper, the TensorFlow SSD network was trained on the InceptionV2 architecture using the MSCOCO dataset which has 91 classes (including the background class). The config details of the network can be found here.

Where Is This Sample Located?

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

Getting Started:

For more information about getting started, see Getting Started With Python Samples.
For specifics about this sample, refer to the GitHub: uff_ssd/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.

8.2. 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, see 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.

8.3. Object Detection With A TensorFlow SSD Network

This sample, sampleUffSSD, preprocesses a TensorFlow SSD network, performs inference on the SSD network in TensorRT, using TensorRT plugins to speed up inference.

What does this sample do?

This sample is based on the SSD: Single Shot MultiBox Detector paper. The SSD network performs the task of object detection and localization in a single forward pass of the network.

The SSD network used in this sample is based on the TensorFlow implementation of SSD, which actually differs from the original paper, in that it has an inception_v2 backbone. For more information about the actual model, download ssd_inception_v2_coco. The TensorFlow SSD network was trained on the InceptionV2 architecture using the MSCOCO dataset which has 91 classes (including the background class). The config details of the network can be found here.

Where is this sample located?

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

How do I get started?

For more information about getting started, see Getting Started With C++ Samples. For specifics about this sample, refer to the GitHub: sampleUffSSD/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.

8.4. Object Detection With Faster R-CNN

This sample, sampleFasterRCNN, uses TensorRT plugins, performs inference, and implements a fused custom layer for end-to-end inferencing of a Faster R-CNN model.

What does this sample do?

Specifically, this sample demonstrates the implementation of a Faster R-CNN network in TensorRT, performs a quick performance test in TensorRT, implements a fused custom

layer, and constructs the basis for further optimization, for example using INT8 calibration, user trained network, etc. The Faster R-CNN network is based on the paper Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.

Where is this sample located?

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

How do I get started?

For more information about getting started, see Getting Started With C++ Samples. For specifics about this sample, refer to the GitHub: sampleFasterRCNN/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.

8.5. Object Detection With SSD

This sample, sampleSSD, performs the task of object detection and localization in a single forward pass of the network.

What does this sample do?

This sample is based on the SSD: Single Shot MultiBox Detector paper. This network is built using the VGG network as a backbone and trained using PASCAL VOC 2007+ 2012 datasets.

Unlike Faster R-CNN, SSD completely eliminates the proposal generation and subsequent pixel or feature resampling stages and encapsulates all computation in a single network. This makes SSD straightforward to integrate into systems that require a detection component.

Where is this sample located?

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

How do I get started?

For more information about getting started, see Getting Started With C++ Samples. For specifics about this sample, refer to the GitHub: sampleSSD/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.

8.6. Object Detection And Instance Segmentation With A TensorFlow Mask R-CNN Network

This sample, sampleUffMaskRCNN, performs inference on the Mask R-CNN network in TensorRT.

What does this sample do?

Mask R-CNN is based on the Mask R-CNN paper which performs the task of object detection and object mask predictions on a target image.

This sample's model is based on the Keras implementation of Mask R-CNN and its training framework can be found in the Mask R-CNN Github repository. We have verified that the pre-trained Keras model (with backbone ResNet101 + FPN and dataset coco) provided in the v2.0 release can be converted to UFF and consumed by this sample. And, it is also feasible to deploy your customized Mask R-CNN model trained with specific backbone and datasets.

This sample makes use of TensorRT plugins to run the Mask R-CNN model. To use these plugins, the Keras model should be converted to TensorFlow `.pb` model. Then this `.pb` model needs to be preprocessed and converted to the UFF model with the help of GraphSurgeon and the UFF utility.

Where is this sample located?

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

How do I get started?

For more information about getting started, see Getting Started With C++ Samples. For specifics about this sample, refer to the GitHub: sampleUffMaskRCNN/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.

8.7. Object Detection With A TensorFlow Faster R-CNN Network

This sample, sampleUffFasterRCNN, serves as a demo of how to use the pre-trained Faster-RCNN model in Transfer Learning Toolkit to do inference with TensorRT.

What does this sample do?

This sample is a UFF TensorRT sample for Faster-RCNN in NVIDIA Transfer Learning Toolkit SDK. Besides the sample itself, it also provides two TensorRT plugins: Proposal and CropAndResize to implement the proposal layer and ROIPooling layer as custom layers in the model since TensorRT has no native support for them.

In this sample, we provide a UFF model as a demo. While in the Transfer Learning Toolkit workflow, we can't provide the UFF model. Instead, we can only get the `.tlt` model during training and the `.etlt` model after `tlt-export`. Both of them are encrypted models and the Transfer Learning Toolkit user will `use tlt-converter` to decrypt the `.etlt` model and generate a TensorRT engine file in a single step. Therefore, in the Transfer Learning Toolkit workflow, we will consume the TensorRT engine instead of a UFF model. However, this sample can still serve as a demo on how to use the UFF Faster R-CNN model regardless of its format.

Where is this sample located?

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

How do I get started?

For more information about getting started, see Getting Started With Python Samples. For specifics about this sample, refer to the GitHub: sampleUffFasterRCNN/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.

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

How do I get started?

For more information about getting started, see 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.

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.

VESA DisplayPort

DisplayPort and DisplayPort Compliance Logo, DisplayPort Compliance Logo for Dual-mode Sources, and DisplayPort Compliance Logo for Active Cables are trademarks owned by the Video Electronics Standards Association in the United States and other countries.

HDMI

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

ARM

ARM, AMBA and ARM Powered are registered trademarks of ARM Limited. Cortex, MPCore and Mali are trademarks of ARM Limited. All other brands or product names are the property of their respective holders. "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.

OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

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

Trademarks

Copyright

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