Index

Deploy Deep-learning Model of CUDA on DEBIX and NPU debugging

Chapter 1 Preparing environment

1.1 Installing CUDA and cuDNN environment

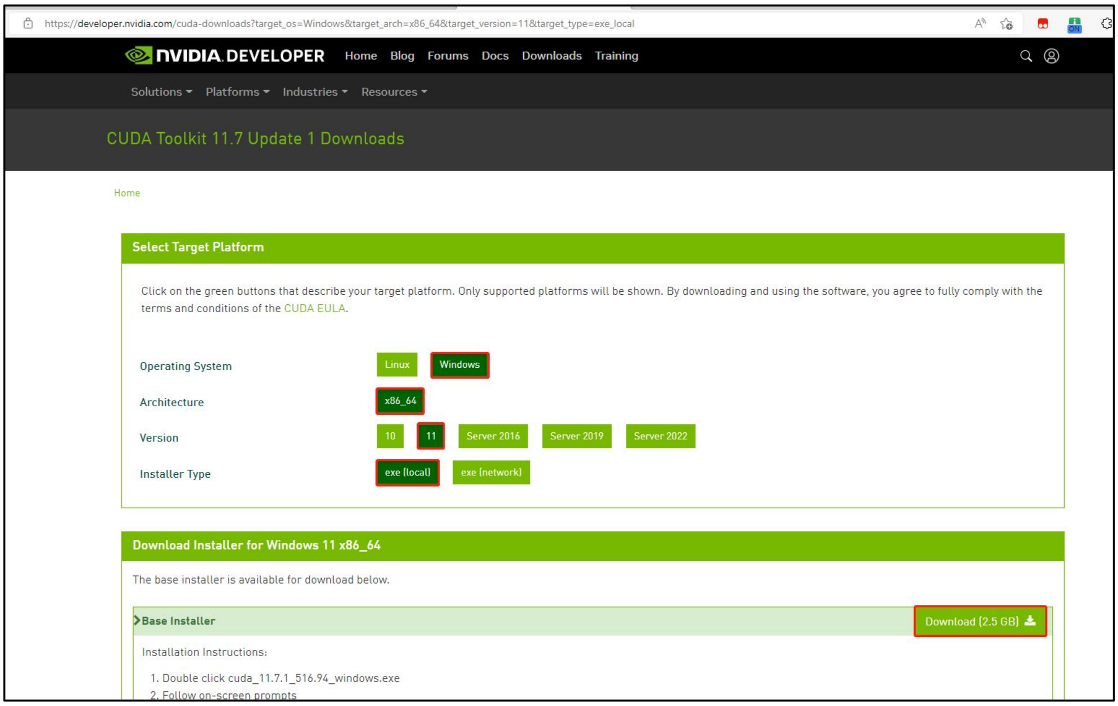
CUDA is a unified computing architecture provided by NVIDIA, CUDA software packages are the API in Application layer of CUDA architecture. This article performs the experiment on Windows10 and Debix V1.8.

1.1.1 Downloading and Installation of CUDA software packages

Download CUDA11 from https://developer.nvidia.com/cuda-downloads?target_os=Windows&target_arch=x86_64&target_version=11&target_type=exe_local and install it.

The NVIDIA® CUDA® Toolkit provides a development environment for creating high performance GPU-accelerated applications. With the CUDA Toolkit, you can develop, optimize, and deploy your applications on GPU-accelerated embedded systems, desktop workstations, enterprise data centers, cloud-based platforms and HPC super computers. The toolkit includes GPU-accelerated libraries, debugging and optimization tools, a C/C++ compiler, and a runtime library to deploy your application.

Using built-in capabilities for distributing computations across multi-GPU configurations, scientists and researchers can develop applications that scale from single GPU workstations to cloud installations with thousands of GPUs.


https://developer.nvidia.com/cuda-downloads?target_os=Windows&target_arch=x86_64&target_version=11&target_type=exe_local
NVIDIA DEVELOPER
Home
Blog
Forums
Docs
Downloads
Training
Solutions
Platforms
Industries
Resources
Home

CUDA Toolkit 11.7 Update 1 Downloads

Select Target Platform

Click on the green buttons that describe your target platform. Only supported platforms will be shown. By downloading and using the software, you agree to fully comply with the terms and conditions of the CUDA EULA.

Operating System: Linux [✓ Windows]

Architecture: [✓ x86_64]

Version: 10 [✓ 11] Server 2016 Server 2019 Server 2022

Installer Type: [✓ exe (local)] exe (network)

Download Installer for Windows 11 x86_64

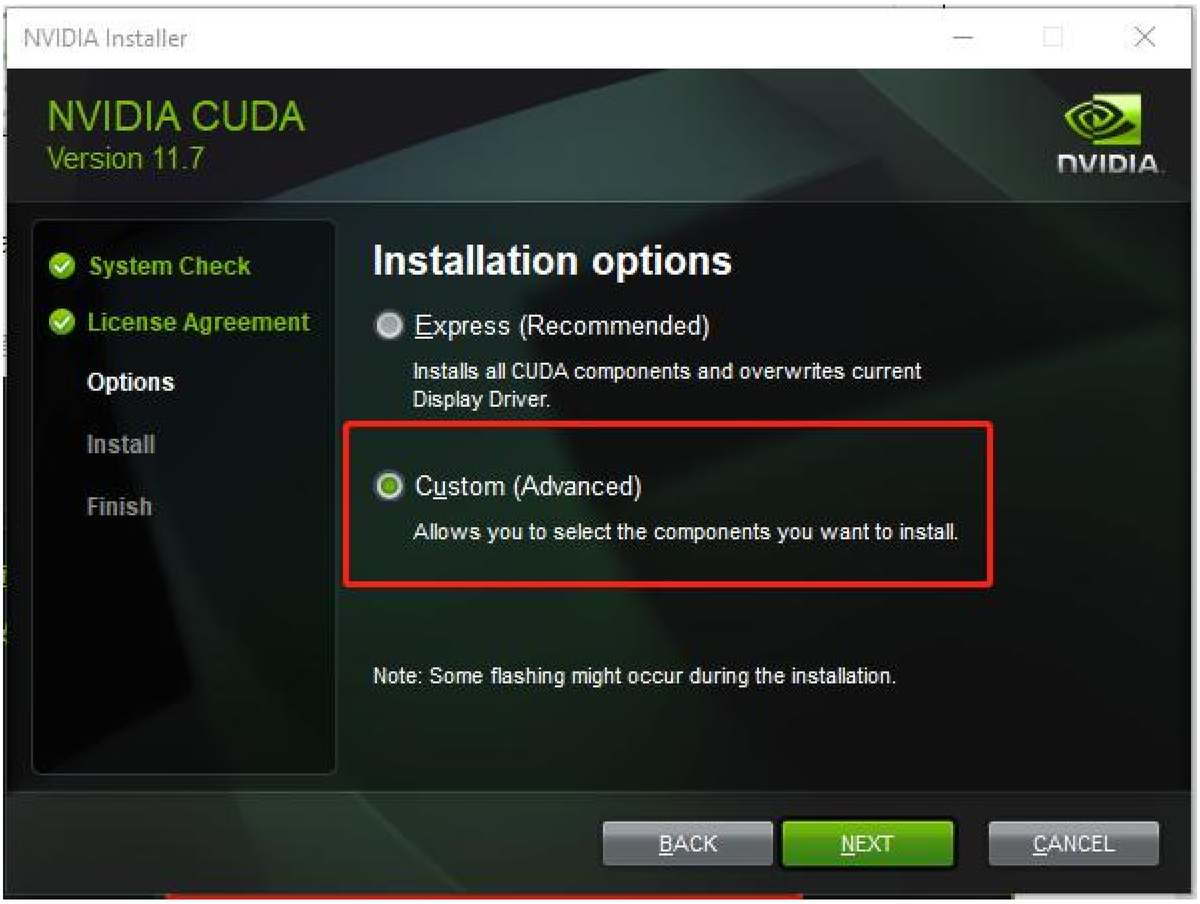
The base installer is available for download below.

Base Installer — Download (2.5 GB)

Installation Instructions:

1. Double click cuda_11.7.1_516.94_windows.exe
2. Follow on-screen prompts

In the installation process, tick the following specified option.



Installation options

○ Express (Recommended)

Installs all CUDA components and overwrites current Display Driver.

● Custom (Advanced)

Allows you to select the components you want to install.

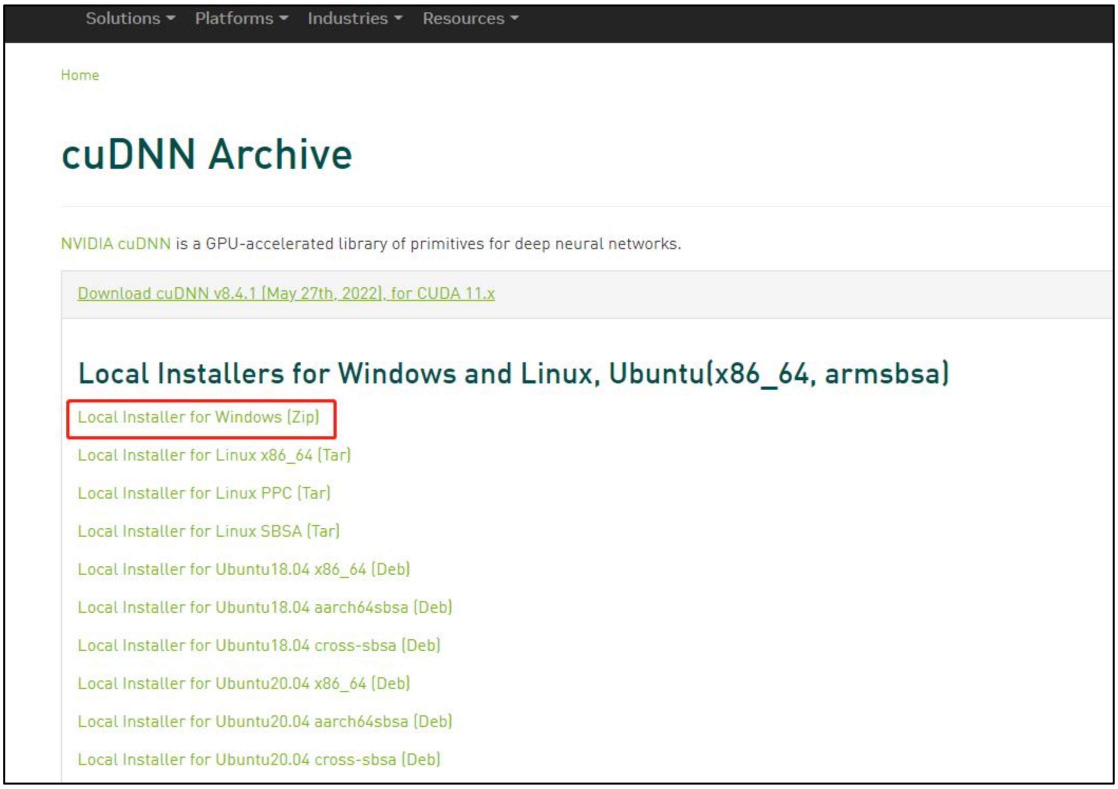
Note: Some flashing might occur during the installation.

1.1.2 Downloading and installation of cuDNN

cuDNN is the neural network operator library.
Log in to website to download https://developer.nvidia.com/rdp/cudnn-download
Once downloaded, decompressed it to any directory, and copy the three file folders(bin, include, lib) to the installation path of CUDA(C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.7)

The NVIDIA CUDA® Deep Neural Network library(cuDNN) is a GPU-accelerated library of primitives for deep neural networks. cuDNN provides highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization, and activation layers.

Deep learning researchers and framework developers worldwide rely on cuDNN for high-performance GPU acceleration. It allows them to focus on training neural networks and developing software applications rather than spending time on low-level GPU performance tuning. cuDNN accelerates widely used deep learning frameworks including Caffe2, Chainer, Keras, MATLAB, MxNet, PaddlePaddle, PyTorch, and TensorFlow. For access to NVIDIA optimized deep learning framework containers that have cuDNN integrated into frameworks, visit NVIDIA GPU CLOUD to learn more and get started.



cuDNN Archive

NVIDIA cuDNN is a GPU-accelerated library of primitives for deep neural networks.

Download cuDNN v8.4.1 (May 27th, 2022), for CUDA 11.x

Local Installers for Windows and Linux, Ubuntu(x86_64, armsbsa)

Local Installer for Windows (Zip)
Local Installer for Linux x86_64 (Tar)
Local Installer for Linux PPC (Tar)
Local Installer for Linux SBSA (Tar)
Local Installer for Ubuntu18.04 x86_64 (Deb)
Local Installer for Ubuntu18.04 aarch64sbsa (Deb)
Local Installer for Ubuntu18.04 cross-sbsa (Deb)
Local Installer for Ubuntu20.04 x86_64 (Deb)
Local Installer for Ubuntu20.04 aarch64sbsa (Deb)
Local Installer for Ubuntu20.04 cross-sbsa (Deb)

1.2 Installation of eIQ

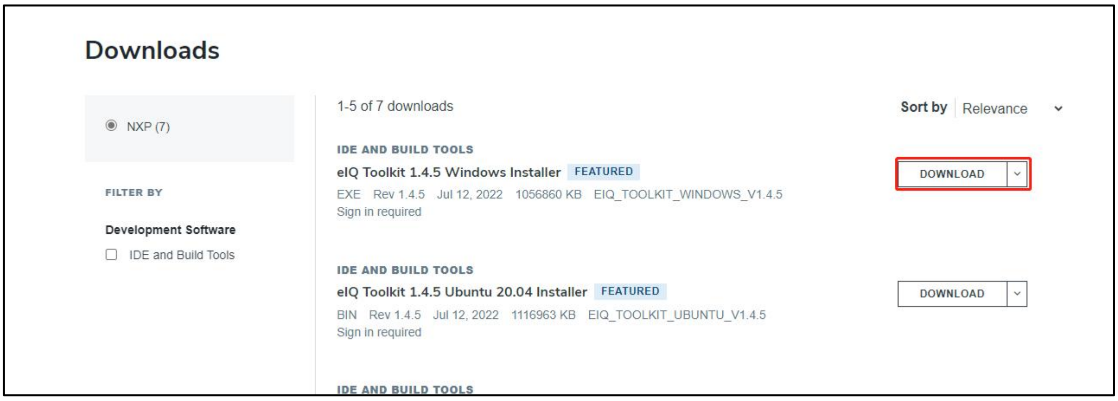
Download the latest eIQ toolbox from NXP eIQ official website:https://www.nxp.com/design/software/development-software/eiq-ml-development-environment/eiq-toolkit-for-end-to-end-model-development-and-deployment:EIQ-TOOLKIT

The eIQ Toolkit enables machine learning development with an intuitive GUI(eIQ Portal) and development workflow tools, along with command line host tool options as part of the eIQ ML software development environment. NXP’s eIQ Toolkit enables graph-level profiling capability with runtime insights to help optimize neural network architectures on target EdgeVerse^{TM} processors.

The eIQ Portal, developed in exclusive partnership with Au-Zone Technologies, is an intuitive graphical user interface(GUI) that simplifies vision based ML solutions development. Developers can create, optimize, debug and export ML models, as well as import datasets and models, rapidly train and deploy neural network models an ML workloads for vision applications.

The eIQ Portal provides output software that seamlessly feeds into DeepViewRT^{TM}, TensorFlow^{TM} Lite, TensorFlow Lite Micro, Glow and ONNX Runtime inference engines.

The eIQ Toolkit and the eIQ Portal are provided with examples demonstrating use cases and guidelines for the different process flow options such as importing pretrained models based on popular frameworks, creating, importing and augmenting datasets to develop models within the tools or integrating with users’ existing flow to leverage the supported inference engines.



Downloads

● NXP (7)

FILTER BY

Development Software

□ IDE and Build Tools

1-5 of 7 downloads

Sort by Relevance

IDE AND BUILD TOOLS
eIQ Toolkit 1.4.5 Windows Installer FEATURED
EXE Rev 1.4.5 Jul 12, 2022 1056860 KB EIQ_TOOLKIT_WINDOWS_V1.4.5
Sign in required

IDE AND BUILD TOOLS
eIQ Toolkit 1.4.5 Ubuntu 20.04 Installer FEATURED
BIN Rev 1.4.5 Jul 12, 2022 1116963 KB EIQ_TOOLKIT_UBUNTU_V1.4.5
Sign in required

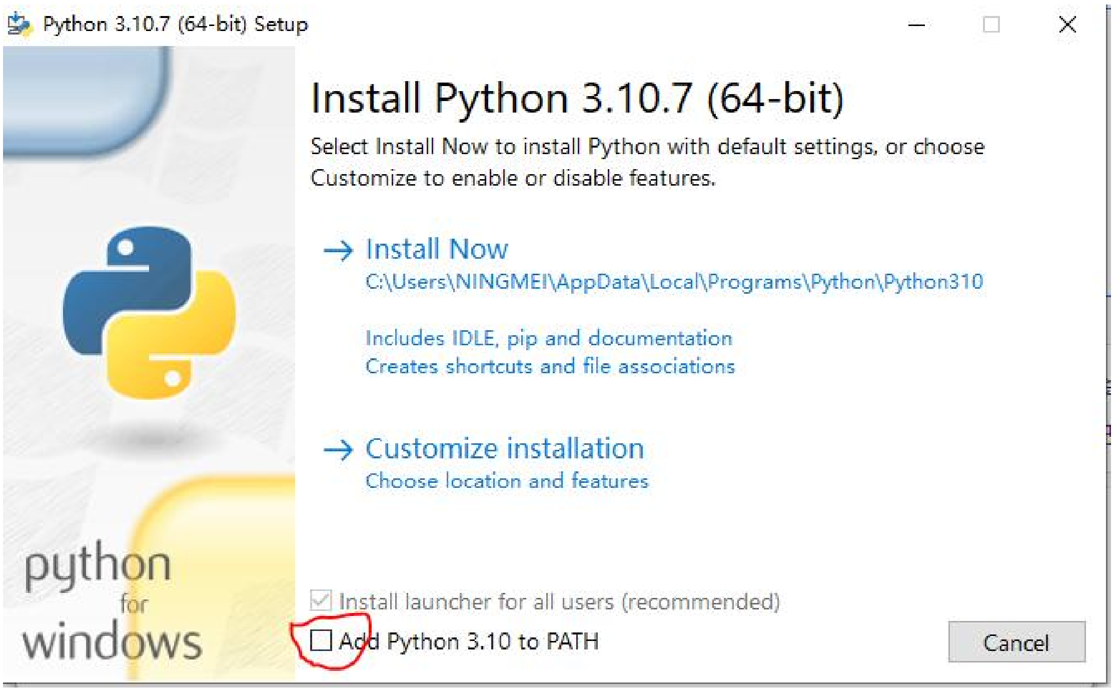
IDE AND BUILD TOOLS

Chapter 2 Deploy Yolov5s model

2.1 Download Yolov5 project

Yolov5 is a family of object detection architecture and models pretrained on the COCO dataset, and represents Ultralytics open-source research into future vision AI methods, incorporating lessons learned and best practices evolved over thousands of hours of research and development.

You need to install python on Windows from https://www.python.org/downloads/windows/
Remember to tick the following option:



Python 3.10.7 (64-bit) Setup

Install Python 3.10.7 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

→ Install Now
C:\Users\NINGMEI\AppData\Local\Programs\Python\Python310

Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
□ Add Python 3.10 to PATH

You also need to install git on windows beforehand.
Run the following command on Windows cmd

```
git clone https://github.com/ultralytics/yolov5.git
```

2.2 install required dependencies

Run the following command on Windows cmd

```
pip install -r requirements.txt
```

2.3 Transformation of pre-training model

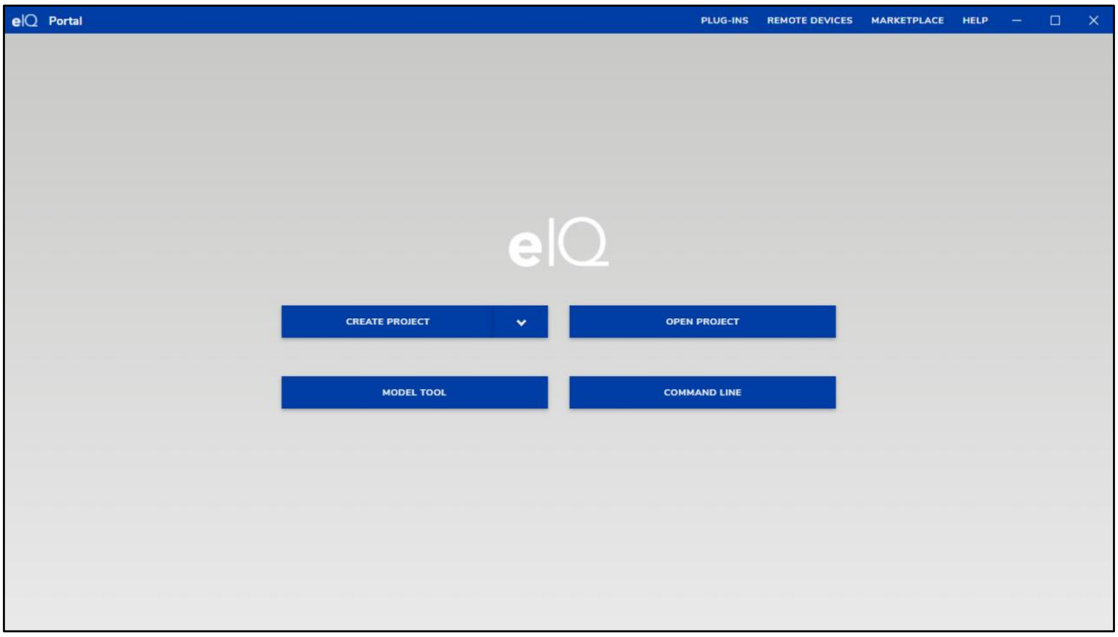
Use the following command to transfer the YOLOv5s model and the pre-training weight to TensorFlow freeze picture format.
Run the following command on Windows cmd

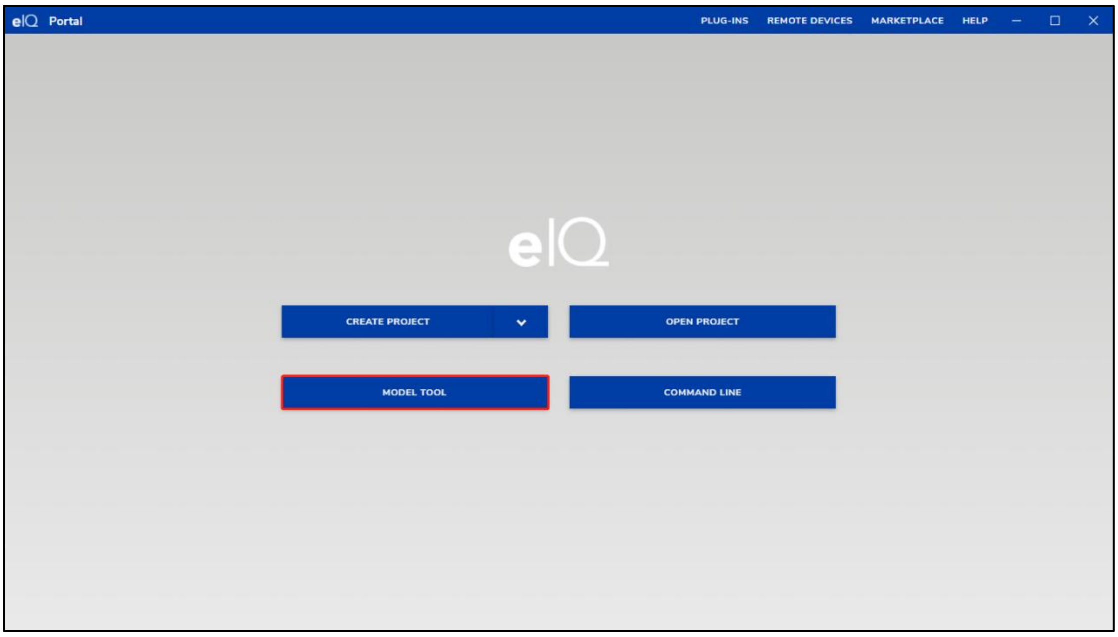
```
python export.py --weights yolov5s.pt --img-size 256 --include pb
```

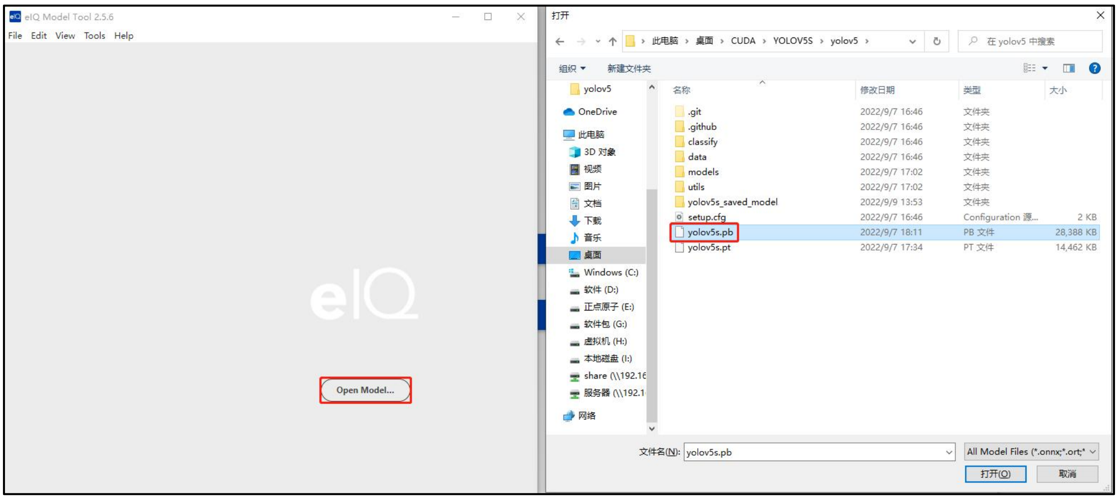
Note: You may need to download pip and install pip specifically. You may need to install tensorflow specifically.

2.4 Open eIQ and enter the model tools

2.5 Load the pd files and perform transformation



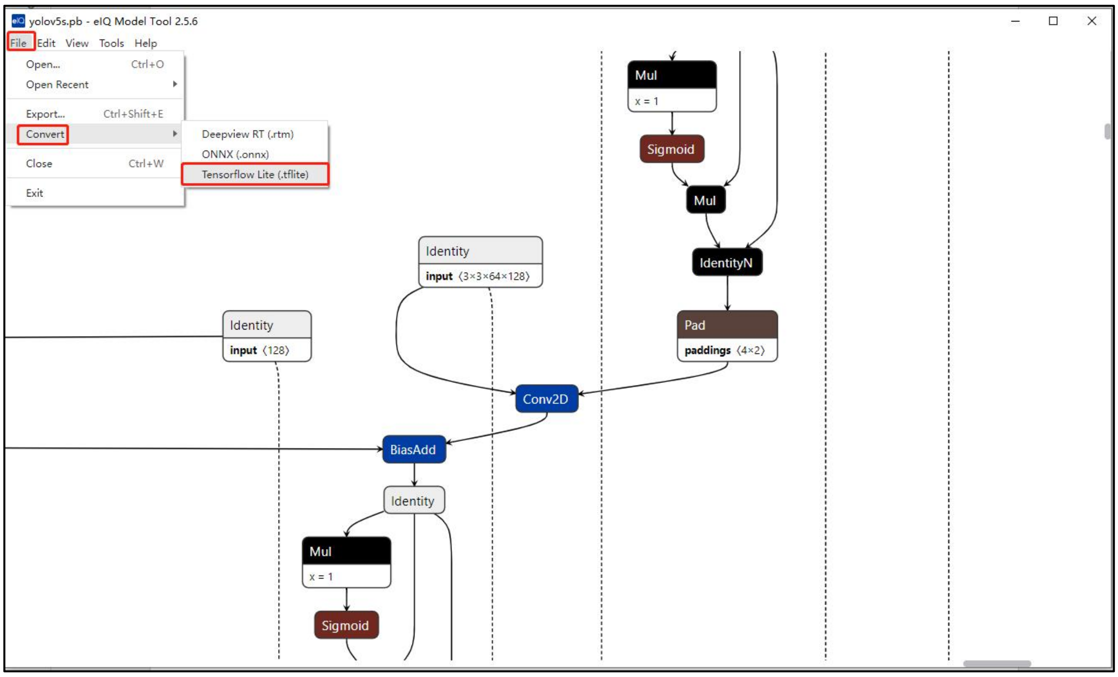


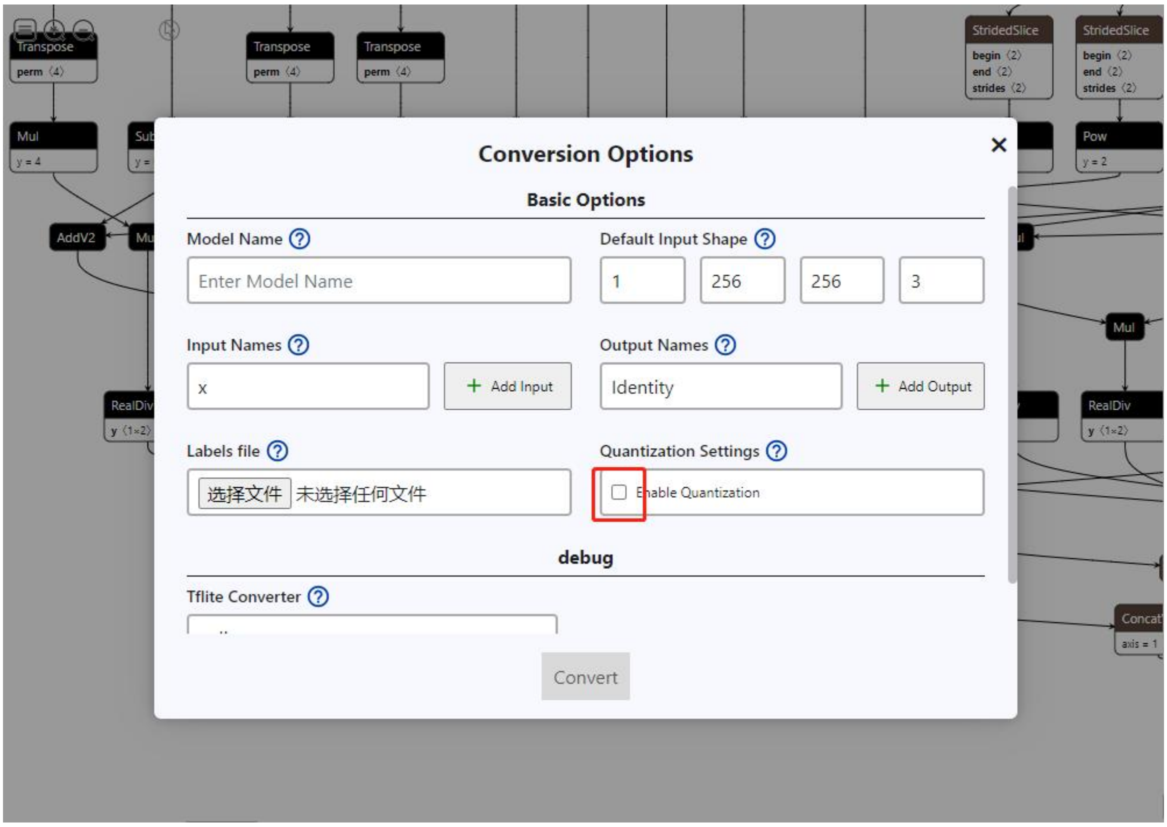


名称	修改日期	类型	大小
.git	2022/9/7 16:46	文件夹	
.github	2022/9/7 16:46	文件夹	
classify	2022/9/7 16:46	文件夹	
data	2022/9/7 16:46	文件夹	
models	2022/9/7 17:02	文件夹	
utils	2022/9/7 17:02	文件夹	
yolov5s_saved_model	2022/9/9 13:53	文件夹	
setup.cfg	2022/9/7 16:46	Configuration 源...	2 KB
yolov5s.pb	2022/9/7 18:11	PB 文件	28,388 KB
yolov5s.pt	2022/9/7 17:34	PT 文件	14,462 KB

文件名(N): yolov5s.pb

All Model Files (*.onnx;*.ort;*





Conversion Options ×

Basic Options

Model Name ?: Enter Model Name

Default Input Shape ?: 1 | 256 | 256 | 3

Input Names ?: x + Add Input

Output Names ?: Identity + Add Output

Labels file ?: 选择文件 未选择任何文件

Quantization Settings ?: □ Enable Quantization

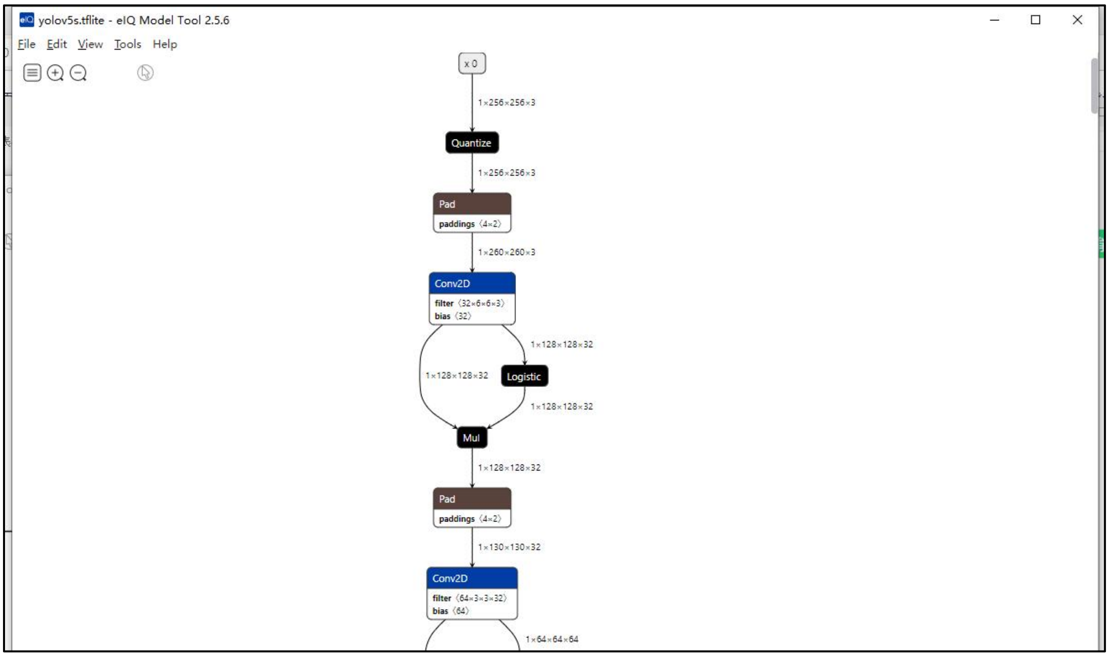
debug

Tflite Converter ?

Convert

Tick the “Enable Quantization” option, provide a Model Name, click Covert. The “.tflite” file will be generated.

The picture after transformation:



Chapter 3 Samples of testing models on DEBIX

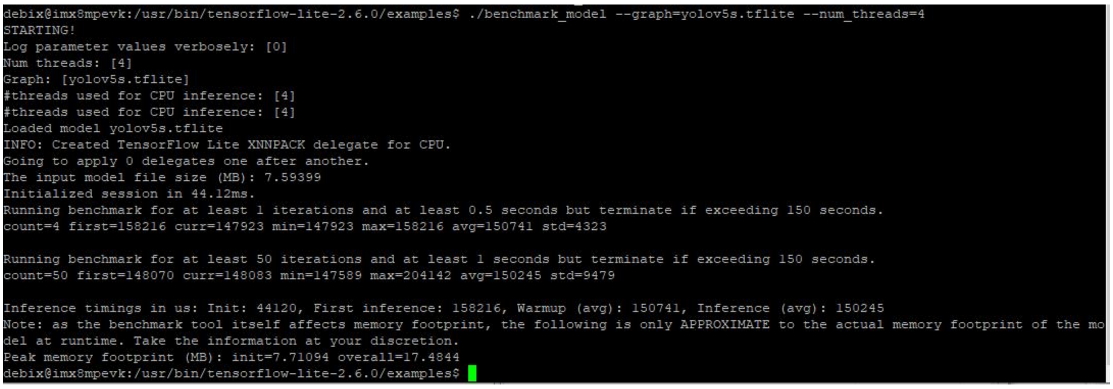
3.1 copy files

Copy the file ‘yolov5s.tflite’ (transferred from YOLOV5 project) and ‘data/image/zidane.jpg’ to the directory ‘/usr/bin/tensorflow-lite-2.6.0/examples’.

3.2 Test

CPU test

```
./benchmark_model --graph=yolov5s.tflite --num_threads=4
```



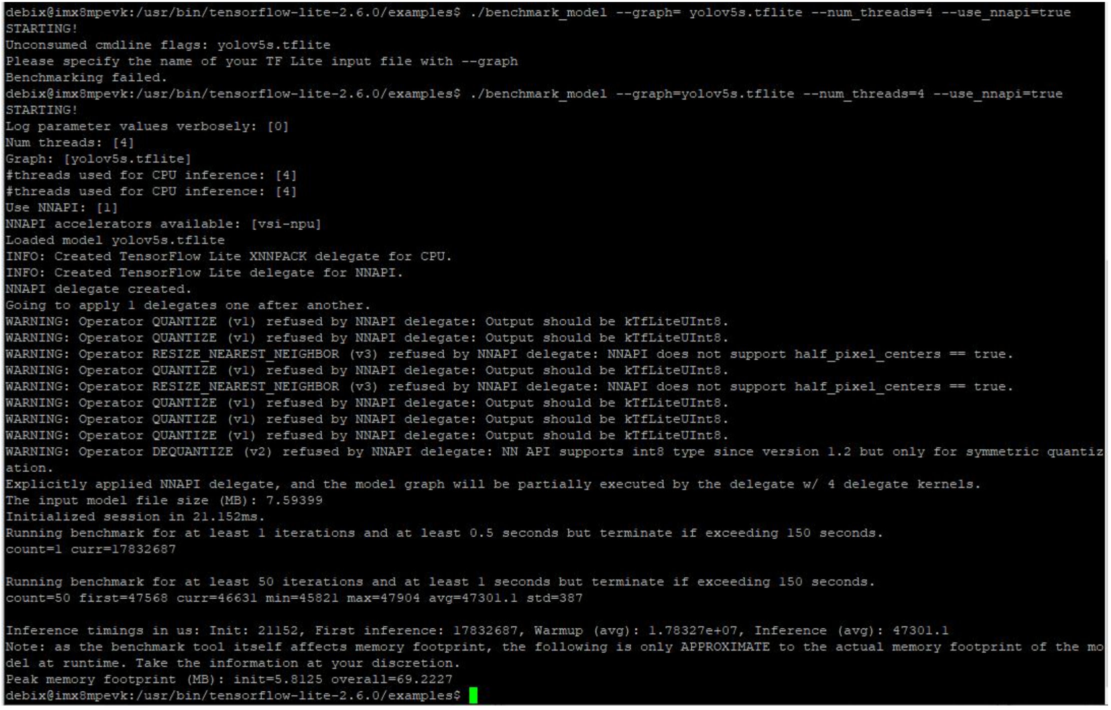
```
debix@imx8mpevk:/usr/bin/tensorflow-lite-2.6.0/examples$ ./benchmark_model --graph=yolov5s.tflite --num_threads=4
STARTING!
Log parameter values verbosely: [0]
Num threads: [4]
Graph: [yolov5s.tflite]
#threads used for CPU inference: [4]
#threads used for CPU inference: [4]
Loaded model yolov5s.tflite
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
Going to apply 0 delegates one after another.
The input model file size (MB): 7.59399
Initialized session in 44.12ms.
Running benchmark for at least 1 iterations and at least 0.5 seconds but terminate if exceeding 150 seconds.
count=4 first=158216 curr=147923 min=147923 max=158216 avg=150741 std=4323

Running benchmark for at least 50 iterations and at least 1 seconds but terminate if exceeding 150 seconds.
count=50 first=148070 curr=148083 min=147589 max=204142 avg=150245 std=9479

Inference timings in us: Init: 44120, First inference: 158216, Warmup (avg): 150741, Inference (avg): 150245
Note: as the benchmark tool itself affects memory footprint, the following is only APPROXIMATE to the actual memory footprint of the mo
del at runtime. Take the information at your discretion.
Peak memory footprint (MB): init=7.71094 overall=17.4844
debix@imx8mpevk:/usr/bin/tensorflow-lite-2.6.0/examples$
```

NPU test

```
./benchmark_model --graph=yolov5s.tflite --num_threads=4 --use_nnapi=true
```



```
debix@imx8mpevk:/usr/bin/tensorflow-lite-2.6.0/examples$ ./benchmark_model --graph= yolov5s.tflite --num_threads=4 --use_nnapi=true
STARTING!
Unconsumed cmdline flags: yolov5s.tflite
Please specify the name of your TF Lite input file with --graph
Benchmarking failed.
debix@imx8mpevk:/usr/bin/tensorflow-lite-2.6.0/examples$ ./benchmark_model --graph=yolov5s.tflite --num_threads=4 --use_nnapi=true
STARTING!
Log parameter values verbosely: [0]
Num threads: [4]
Graph: [yolov5s.tflite]
#threads used for CPU inference: [4]
#threads used for CPU inference: [4]
Use NNAPI: [1]
NNAPI accelerators available: [vsi-npu]
Loaded model yolov5s.tflite
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
INFO: Created TensorFlow Lite delegate for NNAPI.
NNAPI delegate created.
Going to apply 1 delegates one after another.
WARNING: Operator QUANTIZE (v1) refused by NNAPI delegate: Output should be kTfLiteUInt8.
WARNING: Operator QUANTIZE (v1) refused by NNAPI delegate: Output should be kTfLiteUInt8.
WARNING: Operator RESIZE_NEAREST_NEIGHBOR (v3) refused by NNAPI delegate: NNAPI does not support half_pixel_centers == true.
WARNING: Operator QUANTIZE (v1) refused by NNAPI delegate: Output should be kTfLiteUInt8.
WARNING: Operator RESIZE_NEAREST_NEIGHBOR (v3) refused by NNAPI delegate: NNAPI does not support half_pixel_centers == true.
WARNING: Operator QUANTIZE (v1) refused by NNAPI delegate: Output should be kTfLiteUInt8.
WARNING: Operator QUANTIZE (v1) refused by NNAPI delegate: Output should be kTfLiteUInt8.
WARNING: Operator QUANTIZE (v1) refused by NNAPI delegate: Output should be kTfLiteUInt8.
WARNING: Operator DEQUANTIZE (v2) refused by NNAPI delegate: NN API supports int8 type since version 1.2 but only for symmetric quantiz
ation.
Explicitly applied NNAPI delegate, and the model graph will be partially executed by the delegate w/ 4 delegate kernels.
The input model file size (MB): 7.59399
Initialized session in 21.152ms.
Running benchmark for at least 1 iterations and at least 0.5 seconds but terminate if exceeding 150 seconds.
count=1 curr=17832687

Running benchmark for at least 50 iterations and at least 1 seconds but terminate if exceeding 150 seconds.
count=50 first=47568 curr=46631 min=45821 max=47904 avg=47301.1 std=387

Inference timings in us: Init: 21152, First inference: 17832687, Warmup (avg): 1.78327e+07, Inference (avg): 47301.1
Note: as the benchmark tool itself affects memory footprint, the following is only APPROXIMATE to the actual memory footprint of the mo
del at runtime. Take the information at your discretion.
Peak memory footprint (MB): init=5.8125 overall=69.2227
debix@imx8mpevk:/usr/bin/tensorflow-lite-2.6.0/examples$
```