(ubuntu 18.04 cuda 10.2 pytorch onnx→tensorrt) 使用tensorrt加载并运行onnx模型

2021.4

0. TensorRT介绍

1. 安装tensorrt、pycuda

  1.1 安装tensorrt

  1.2 安装pycuda

2. 将pytorch模型转化为onnx模型

3. tensorrt调用onnx模型进行预测

4. 注意事项

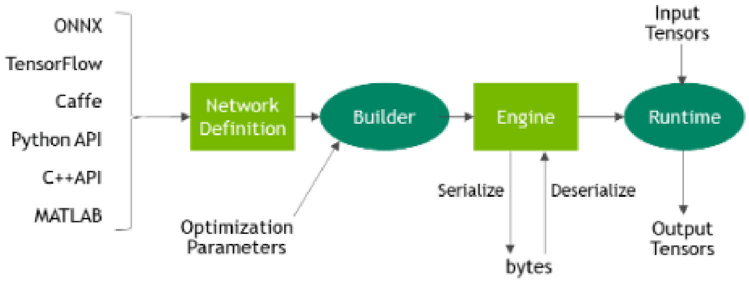
- 下述代码链接：

  https://github.com/kiyoxi2020/tensorrt_example

0. TensorRT介绍

https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html

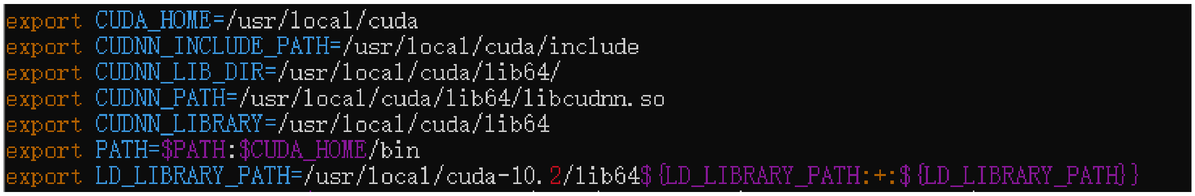
- TensorRT的核心是一个C ++库，可促进对NVIDIA图形处理单元（GPU）进行高性能推断。 它旨在与TensorFlow，Caffe，PyTorch，MXNet等深度学习框架互补地工作。它专门致力于在GPU上快速有效地运行已经受过训练的网络，以生成结果（包括检测，回归或推断任务等）。
- TensorRT根据指定的精度（FP32，FP16或INT8）组合各层，优化内核选择，通过归一化、转换为更优化的矩阵数学运算等方式，改善延迟，吞吐量和效率。
- 主要流程：



1. 安装tensorrt、pycuda

1.1 安装tensorrt

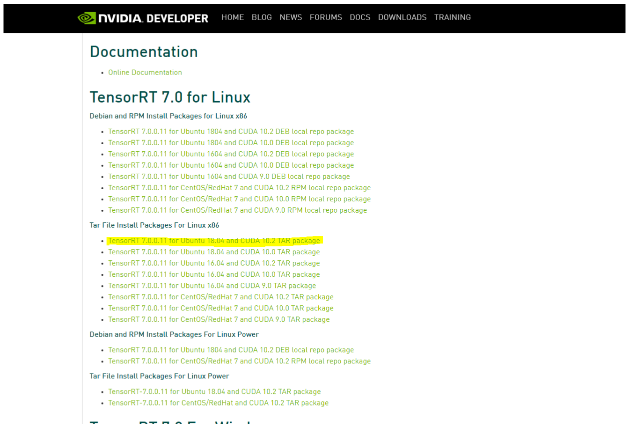
- 环境：ubuntu18.04, 2080Ti，安装cuda10.2+cudnn 7.6.5，且在~/.bashrc中添加cuda、cudnn所在位置：



```
export CUDA_HOME=/usr/local/cuda
export CUDNN_INCLUDE_PATH=/usr/local/cuda/include
export CUDNN_LIB_DIR=/usr/local/cuda/lib64/
export CUDNN_PATH=/usr/local/cuda/lib64/libcudnn.so
export CUDNN_LIBRARY=/usr/local/cuda/lib64
export PATH=$PATH:$CUDA_HOME/bin
export LD_LIBRARY_PATH=/usr/local/cuda-10.2/lib64${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}
```

- tensorrt下载链接：https://developer.nvidia.com/zh-cn/tensorrt

  需要选择tensorrt 7.0.0.11（适用于Ubuntu18.04、cuda10.2、cudnn7.6，其他更高的tensorrt版本要求cudnn8，因此此处选择tensorrt 7.0.0.11）



Documentation

- Online Documentation

TensorRT 7.0 for Linux

Debian and RPM Install Packages for Linux x86

- TensorRT 7.0.0.11 for Ubuntu 1804 and CUDA 10.2 DEB local repo package
- TensorRT 7.0.0.11 for Ubuntu 1804 and CUDA 10.0 DEB local repo package
- TensorRT 7.0.0.11 for Ubuntu 1604 and CUDA 10.2 DEB local repo package
- TensorRT 7.0.0.11 for Ubuntu 1604 and CUDA 10.0 DEB local repo package
- TensorRT 7.0.0.11 for Ubuntu 1604 and CUDA 9.0 DEB local repo package
- TensorRT 7.0.0.11 for CentOS/RedHat 7 and CUDA 10.2 RPM local repo package
- TensorRT 7.0.0.11 for CentOS/RedHat 7 and CUDA 10.0 RPM local repo package
- TensorRT 7.0.0.11 for CentOS/RedHat 7 and CUDA 9.0 RPM local repo package

Tar File Install Packages For Linux x86

- TensorRT 7.0.0.11 for Ubuntu 18.04 and CUDA 10.2 TAR package
- TensorRT 7.0.0.11 for Ubuntu 18.04 and CUDA 10.0 TAR package
- TensorRT 7.0.0.11 for Ubuntu 16.04 and CUDA 10.2 TAR package
- TensorRT 7.0.0.11 for Ubuntu 16.04 and CUDA 10.0 TAR package
- TensorRT 7.0.0.11 for Ubuntu 16.04 and CUDA 9.0 TAR package
- TensorRT 7.0.0.11 for CentOS/RedHat 7 and CUDA 10.2 TAR package
- TensorRT 7.0.0.11 for CentOS/RedHat 7 and CUDA 10.0 TAR package
- TensorRT 7.0.0.11 for CentOS/RedHat 7 and CUDA 9.0 TAR package

Debian and RPM Install Packages For Linux Power

- TensorRT 7.0.0.11 for Ubuntu 1804 and CUDA 10.2 DEB local repo package
- TensorRT 7.0.0.11 for CentOS/RedHat 7 and CUDA 10.2 RPM local repo package

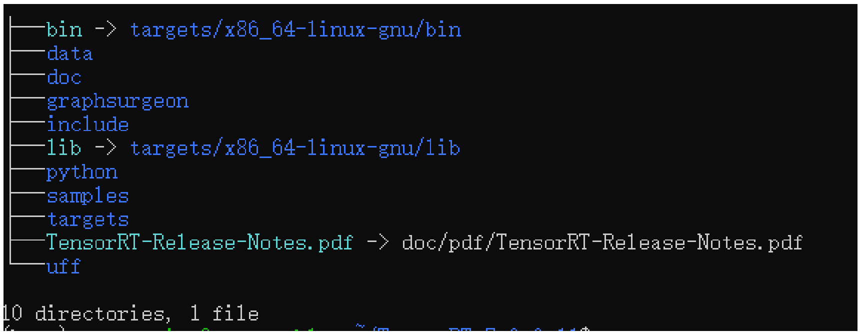
Tar File Install Packages For Linux Power

- TensorRT-7.0.0.11 for Ubuntu 18.04 and CUDA 10.2 TAR package
- TensorRT-7.0.0.11 for CentOS/RedHat and CUDA 10.2 TAR package

  - 下载之后解压：

    ```
    tar xzvf TensorRT-7.0.0.11.Ubuntu-18.04.x86_64-gnu.cuda-10.2.cudnn7.6.tar.gz
    ```

  - 解压后得到“TensorRT-7.0.0.11”文件，包含以下目录：



```
├──bin -> targets/x86_64-linux-gnu/bin
├──data
├──doc
├──graphsurgeon
├──include
├──lib -> targets/x86_64-linux-gnu/lib
├──python
├──samples
├──targets
├──TensorRT-Release-Notes.pdf -> doc/pdf/TensorRT-Release-Notes.pdf
└──uff

10 directories, 1 file
```

- 安装指导：打开**doc/pdf/TensorRT-Installation-Guide.pdf**

  由于下载的是tar安装文件，因此根据4.3 tar file installation安装即可

主要安装步骤如下：

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/TensorRT-7.0.0.11/lib   #TensorRT文件所在目录

cd TensorRT-7.0.0.11/python
pip3 install tensorrt-*-cp3x-none-linux_x86_64.whl # 根据python版本选择whl文件

cd TensorRT-7.0.0.11/graphsurgeon
pip3 install graphsurgeon-0.4.1-py2.py3-none-any.whl
```

- 验证安装：进入samples目录，该目录下有很多c++、python的tensorrt使用示例，可以详细学习一下
  - 进入samples/sampleMNIST目录

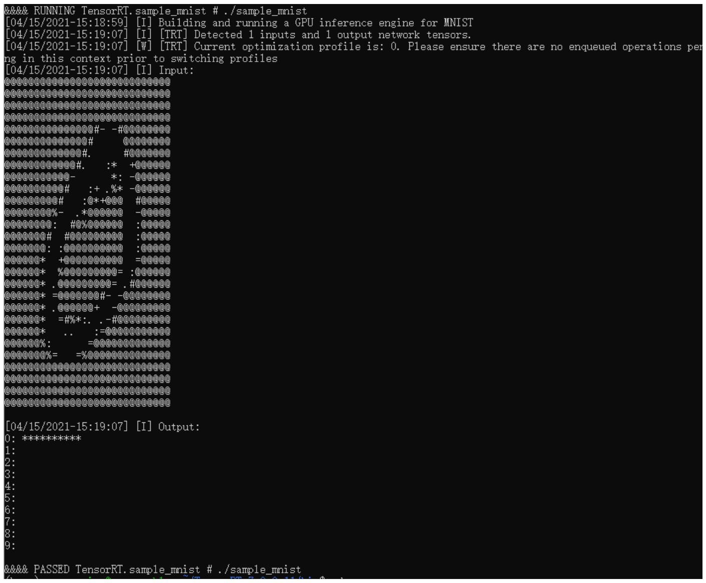
    ```
    make clean   # 可能需要sudo
    make

    cd ../../bin
    ./sample_mnist
    ```

  - 还需要准备数据：进入data/minist目录下，运行

    ```
    python download_pgms.py
    ```

    得到0.pgm~9.pgm，注意download_pgms.py里面的mnist下载链接可能无效，此时可以找一个有效的链接进行替换

  - 得到结果



```
&&&& RUNNING TensorRT.sample_mnist # ./sample_mnist
[04/15/2021-15:18:59] [I] Building and running a GPU inference engine for MNIST
[04/15/2021-15:19:07] [I] [TRT] Detected 1 inputs and 1 output network tensors.
[04/15/2021-15:19:07] [W] [TRT] Current optimization profile is: 0. Please ensure there are no enqueued operations per
ng in this context prior to switching profiles
[04/15/2021-15:19:07] [I] Input:
@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@#- -#@@@@@@@@
@@@@@@@@@@@@@@#     @@@@@@@@
@@@@@@@@@@@@@#.     #@@@@@@@
@@@@@@@@@@@@#.   :*  +@@@@@@
@@@@@@@@@@@-      *: -@@@@@@
@@@@@@@@@@#   :+ .%* -@@@@@@
@@@@@@@@@#   :@*+@@@  #@@@@@
@@@@@@@@%-  .*@@@@@@  -@@@@@
@@@@@@@@:  #@%@@@@@@  :@@@@@
@@@@@@@#  #@@@@@@@@@  :@@@@@
@@@@@@@: :@@@@@@@@@@  :@@@@@
@@@@@@*  +@@@@@@@@@@  =@@@@@
@@@@@@*  %@@@@@@@@@= :@@@@@@
@@@@@@* .@@@@@@@@@= .#@@@@@@
@@@@@@* =@@@@@@@#- -@@@@@@@@
@@@@@@* .@@@@@@+  -@@@@@@@@@
@@@@@@*  =#%*:. .-#@@@@@@@@@
@@@@@@*   ..   :=@@@@@@@@@@@
@@@@@@%:      =@@@@@@@@@@@@@
@@@@@@@%=   =%@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@

[04/15/2021-15:19:07] [I] Output:
0: **********
1:
2:
3:
4:
5:
6:
7:
8:
9:

&&&& PASSED TensorRT.sample_mnist # ./sample_mnist
```

1.2 安装pycuda

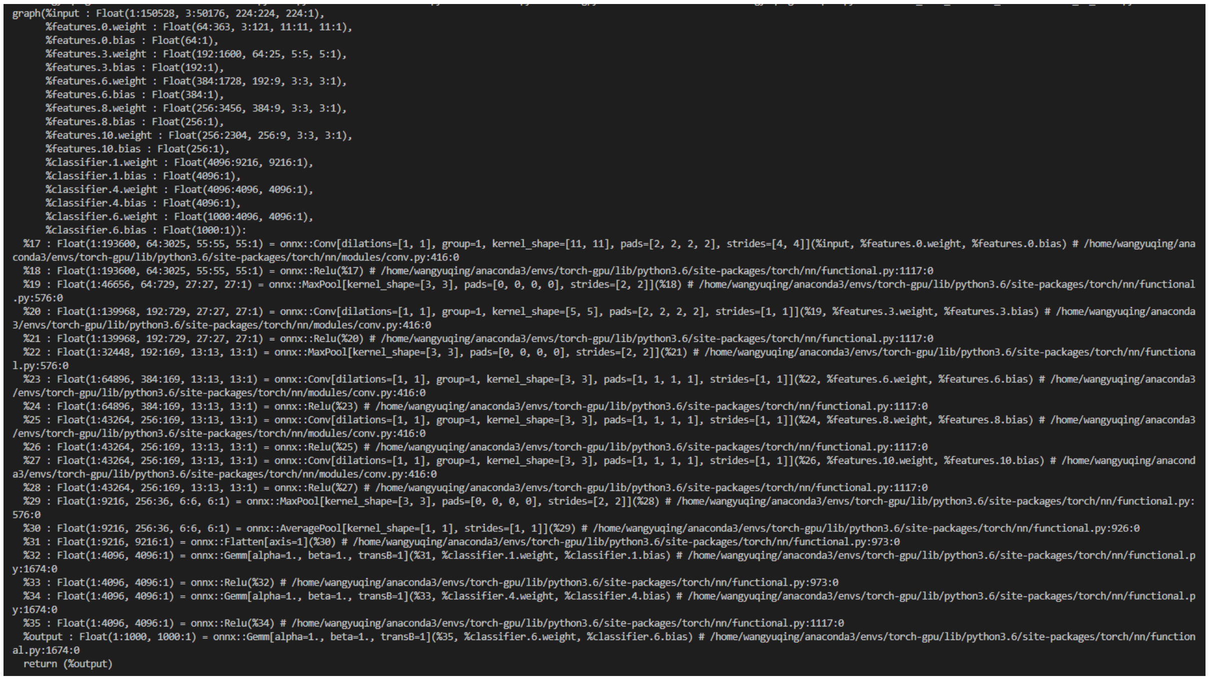
```
pip install pycuda
```

2. 将pytorch模型转化为onnx模型

- tensorrt无法直接读取pytorch模型，因此需要先转化为onnx
- 以alexnet为例，主要代码：

  ```
  import torch
  import torchvision

  def get_model():
      model = torchvision.models.alexnet(pretrained=True).cuda()
      resnet_model = model.eval()
      return model

  def get_onnx(model, onnx_save_path, example_tensor):
      example_tensor = example_tensor.cuda()
      _ = torch.onnx.export(model,
                          example_tensor,
                          onnx_save_path,
                          verbose=True,
                          training=False,
                          do_constant_folding=False,
                          input_names=['input'],
                          output_names=['output'])

  if __name__ == '__main__':
      model = get_model()
      onnx_save_path = "alexnet.onnx"
      example_tensor = torch.randn(1, 3, 224, 224, device='cuda')

      get_onnx(model, onnx_save_path, example_tensor)
  ```

  会有onnx模型的输出：



```
graph(%input : Float(1:150528, 3:50176, 224:224, 224:1),
      %features.0.weight : Float(64:363, 3:121, 11:11, 11:1),
      %features.0.bias : Float(64:1),
      %features.3.weight : Float(192:1600, 64:25, 5:5, 5:1),
      %features.3.bias : Float(192:1),
      %features.6.weight : Float(384:1728, 192:9, 3:3, 3:1),
      %features.6.bias : Float(384:1),
      %features.8.weight : Float(256:3456, 384:9, 3:3, 3:1),
      %features.8.bias : Float(256:1),
      %features.10.weight : Float(256:2304, 256:9, 3:3, 3:1),
      %features.10.bias : Float(256:1),
      %classifier.1.weight : Float(4096:9216, 9216:1),
      %classifier.1.bias : Float(4096:1),
      %classifier.4.weight : Float(4096:4096, 4096:1),
      %classifier.4.bias : Float(4096:1),
      %classifier.6.weight : Float(1000:4096, 4096:1),
      %classifier.6.bias : Float(1000:1)):
  %17 : Float(1:193600, 64:3025, 55:55, 55:1) = onnx::Conv[dilations=[1, 1], group=1, kernel_shape=[11, 11], pads=[2, 2, 2, 2], strides=[4, 4]](%input, %features.0.weight, %features.0.bias) # /home/wangyuqing/ana
conda3/envs/torch-gpu/lib/python3.6/site-packages/torch/nn/modules/conv.py:416:0
  %18 : Float(1:193600, 64:3025, 55:55, 55:1) = onnx::Relu(%17) # /home/wangyuqing/anaconda3/envs/torch-gpu/lib/python3.6/site-packages/torch/nn/functional.py:1117:0
  %19 : Float(1:46656, 64:729, 27:27, 27:1) = onnx::MaxPool[kernel_shape=[3, 3], pads=[0, 0, 0, 0], strides=[2, 2]](%18) # /home/wangyuqing/anaconda3/envs/torch-gpu/lib/python3.6/site-packages/torch/nn/functional
.py:576:0
  %20 : Float(1:139968, 192:729, 27:27, 27:1) = onnx::Conv[dilations=[1, 1], group=1, kernel_shape=[5, 5], pads=[2, 2, 2, 2], strides=[1, 1]](%19, %features.3.weight, %features.3.bias) # /home/wangyuqing/anaconda
3/envs/torch-gpu/lib/python3.6/site-packages/torch/nn/modules/conv.py:416:0
  %21 : Float(1:139968, 192:729, 27:27, 27:1) = onnx::Relu(%20) # /home/wangyuqing/anaconda3/envs/torch-gpu/lib/python3.6/site-packages/torch/nn/functional.py:1117:0
  %22 : Float(1:32448, 192:169, 13:13, 13:1) = onnx::MaxPool[kernel_shape=[3, 3], pads=[0, 0, 0, 0], strides=[2, 2]](%21) # /home/wangyuqing/anaconda3/envs/torch-gpu/lib/python3.6/site-packages/torch/nn/functiona
l.py:576:0
  %23 : Float(1:64896, 384:169, 13:13, 13:1) = onnx::Conv[dilations=[1, 1], group=1, kernel_shape=[3, 3], pads=[1, 1, 1, 1], strides=[1, 1]](%22, %features.6.weight, %features.6.bias) # /home/wangyuqing/anaconda3
/envs/torch-gpu/lib/python3.6/site-packages/torch/nn/modules/conv.py:416:0
  %24 : Float(1:64896, 384:169, 13:13, 13:1) = onnx::Relu(%23) # /home/wangyuqing/anaconda3/envs/torch-gpu/lib/python3.6/site-packages/torch/nn/functional.py:1117:0
  %25 : Float(1:43264, 256:169, 13:13, 13:1) = onnx::Conv[dilations=[1, 1], group=1, kernel_shape=[3, 3], pads=[1, 1, 1, 1], strides=[1, 1]](%24, %features.8.weight, %features.8.bias) # /home/wangyuqing/anaconda3
/envs/torch-gpu/lib/python3.6/site-packages/torch/nn/modules/conv.py:416:0
  %26 : Float(1:43264, 256:169, 13:13, 13:1) = onnx::Relu(%25) # /home/wangyuqing/anaconda3/envs/torch-gpu/lib/python3.6/site-packages/torch/nn/functional.py:1117:0
  %27 : Float(1:43264, 256:169, 13:13, 13:1) = onnx::Conv[dilations=[1, 1], group=1, kernel_shape=[3, 3], pads=[1, 1, 1, 1], strides=[1, 1]](%26, %features.10.weight, %features.10.bias) # /home/wangyuqing/anacond
a3/envs/torch-gpu/lib/python3.6/site-packages/torch/nn/modules/conv.py:416:0
  %28 : Float(1:43264, 256:169, 13:13, 13:1) = onnx::Relu(%27) # /home/wangyuqing/anaconda3/envs/torch-gpu/lib/python3.6/site-packages/torch/nn/functional.py:1117:0
  %29 : Float(1:9216, 256:36, 6:6, 6:1) = onnx::MaxPool[kernel_shape=[3, 3], pads=[0, 0, 0, 0], strides=[2, 2]](%28) # /home/wangyuqing/anaconda3/envs/torch-gpu/lib/python3.6/site-packages/torch/nn/functional.py:
576:0
  %30 : Float(1:9216, 256:36, 6:6, 6:1) = onnx::AveragePool[kernel_shape=[1, 1], strides=[1, 1]](%29) # /home/wangyuqing/anaconda3/envs/torch-gpu/lib/python3.6/site-packages/torch/nn/functional.py:926:0
  %31 : Float(1:9216, 9216:1) = onnx::Flatten[axis=1](%30) # /home/wangyuqing/anaconda3/envs/torch-gpu/lib/python3.6/site-packages/torch/nn/functional.py:973:0
  %32 : Float(1:4096, 4096:1) = onnx::Gemm[alpha=1., beta=1., transB=1](%31, %classifier.1.weight, %classifier.1.bias) # /home/wangyuqing/anaconda3/envs/torch-gpu/lib/python3.6/site-packages/torch/nn/functional.p
y:1674:0
  %33 : Float(1:4096, 4096:1) = onnx::Relu(%32) # /home/wangyuqing/anaconda3/envs/torch-gpu/lib/python3.6/site-packages/torch/nn/functional.py:973:0
  %34 : Float(1:4096, 4096:1) = onnx::Gemm[alpha=1., beta=1., transB=1](%33, %classifier.4.weight, %classifier.4.bias) # /home/wangyuqing/anaconda3/envs/torch-gpu/lib/python3.6/site-packages/torch/nn/functional.p
y:1674:0
  %35 : Float(1:4096, 4096:1) = onnx::Relu(%34) # /home/wangyuqing/anaconda3/envs/torch-gpu/lib/python3.6/site-packages/torch/nn/functional.py:1117:0
  %output : Float(1:1000, 1000:1) = onnx::Gemm[alpha=1., beta=1., transB=1](%35, %classifier.6.weight, %classifier.6.bias) # /home/wangyuqing/anaconda3/envs/torch-gpu/lib/python3.6/site-packages/torch/nn/function
al.py:1674:0
  return (%output)
```

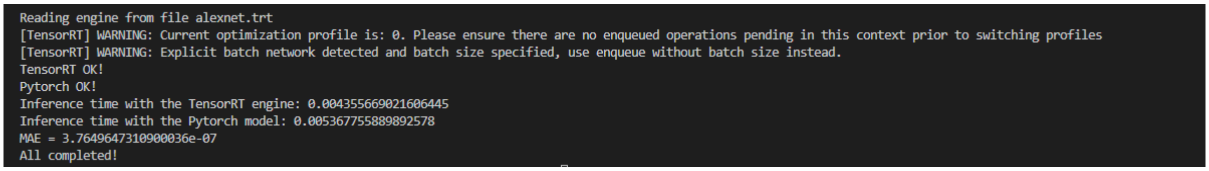
3. tensorrt调用onnx模型进行预测

- 参考1：https://zhuanlan.zhihu.com/p/88318324
- 参考2：samples/python/yolov3_onnx
- 参考3：doc/pdf/TensorRT-Developer-Guide.pdf (Using The Python API章节)
- 主要代码：

```
import pycuda.autoinit
import numpy as np
import pycuda.driver as cuda
import tensorrt as trt
import torch
import os
import time
import torchvision

max_batch_size = 1
onnx_model_path = 'alexnet.onnx'
trt_engine_path = 'alexnet.trt'

TRT_LOGGER = trt.Logger()


class HostDeviceMem(object):
    def __init__(self, host_mem, device_mem):
        self.host = host_mem
        self.device = device_mem

    def __str__(self):
        return "Host: \n" + str(self.host) + '\nDevice:\n' + str(self.device)

    def __repr__(self):
        return self.__str__()
```

```
def allocate_buffers(engine):
    inputs = []
    outputs = []
    bindings = []
    stream = cuda.Stream()
    for binding in engine:
        size = trt.volume(engine.get_binding_shape(binding)) * engine.max_batch_size
        dtype = trt.nptype(engine.get_binding_dtype(binding))
        host_mem = cuda.pagelocked_empty(size, dtype)
        device_mem = cuda.mem_alloc(host_mem.nbytes)
        bindings.append(int(device_mem))
        if engine.binding_is_input(binding):
            inputs.append(HostDeviceMem(host_mem, device_mem))
        else:
            outputs.append(HostDeviceMem(host_mem, device_mem))

    return inputs, outputs, bindings, stream


def get_engine(max_batch_size=1, onnx_file_path="", engine_file_path="", save_engine=False):

    def build_engine(max_batch_size, save_engine):
        with trt.Builder(TRT_LOGGER) as builder, \
            builder.create_network(1 << (int)(trt.NetworkDefinitionCreationFlag.EXPLICIT_BATCH)) as network, \
                trt.OnnxParser(network, TRT_LOGGER) as parser:
            builder.max_workspace_size = 1 << 30
            builder.max_batch_size = max_batch_size

            builder.fp16_mode = False
            builder.int8_mode = False
            if not os.path.exists(onnx_file_path):
                quit('ONNX file {} not found'.format(onnx_file_path))
            print('Loading ONNX file from path {}...'.format(onnx_file_path))
            with open(onnx_file_path, 'rb') as model:
                print('Beginning ONNX file parsing')
                if not parser.parse(model.read()):
                    for error in range(parser.num_errors):
                        print(parser.get_error(error))
            last_layer = network.get_layer(network.num_layers - 1)
            print('Completed parsing of ONNX file')
            print('Building an engine from file {}; this may take a while...'.format(onnx_file_path))

            engine = builder.build_cuda_engine(network)
            print('Completed creating Engine')

            if save_engine:
                with open(engine_file_path, "wb") as f:
                    f.write(engine.serialize())
            return engine
    if os.path.exists(engine_file_path):
        print('Reading engine from file {}'.format(engine_file_path))
        with open(engine_file_path, "rb") as f, trt.Runtime(TRT_LOGGER) as runtime:
            return runtime.deserialize_cuda_engine(f.read())
    else:
        return build_engine(max_batch_size, save_engine)

def do_inference(context, bindings, inputs, outputs, stream, batch_size=1):
    [cuda.memcpy_htod_async(inp.device, inp.host, stream) for inp in inputs]
    context.execute_async(batch_size=batch_size, bindings=bindings, stream_handle=stream.handle)
    [cuda.memcpy_dtoh_async(out.host, out.device, stream) for out in outputs]
    stream.synchronize()
    return [out.host for out in outputs]

def postprocess_the_outputs(h_outputs, shape_of_output):
    h_outputs = h_outputs.reshape(*shape_of_output)
    return h_outputs


def main():
    img = np.ones([1,3,224,224]) * 0.5
    img = img.astype(dtype=np.float32)

    fp16_mode = False
    int8_mode = False
    engine = get_engine(max_batch_size, onnx_model_path, trt_engine_path, save_engine=True)

    context = engine.create_execution_context()

    inputs, outputs, bindings, stream = allocate_buffers(engine)

    shape_of_output = (max_batch_size, 1000)
    inputs[0].host = img.reshape(-1)

    t1 = time.time()
    trt_outputs = do_inference(context, bindings=bindings, inputs=inputs, outputs=outputs, stream=stream)
    t2 = time.time()
    output_trt = postprocess_the_outputs(trt_outputs[0], shape_of_output)
    print('TensorRT OK!')
    # print(np.argmax(output_trt))

    model = torchvision.models.alexnet(pretrained=True).cuda()
    alexnet_model = model.eval()

    input_for_torch = torch.from_numpy(img).cuda()
    t3 = time.time()
    output_torch = alexnet_model(input_for_torch)
    t4 = time.time()
    output_torch = output_torch.cpu().data.numpy()
    print('Pytorch OK!')
    # print(np.argmax(output_torch))

    mae = np.mean(abs(output_trt - output_torch))
    print('Inference time with the TensorRT engine: {}'.format(t2-t1))
    print('Inference time with the Pytorch model: {}'.format(t4-t3))
    print('MAE = {}'.format(mae))

    print('All completed!')

if __name__ == '__main__':
    main()
```

- 运行结果



```
Reading engine from file alexnet.trt
[TensorRT] WARNING: Current optimization profile is: 0. Please ensure there are no enqueued operations pending in this context prior to switching profiles
[TensorRT] WARNING: Explicit batch network detected and batch size specified, use enqueue without batch size instead.
TensorRT OK!
Pytorch OK!
Inference time with the TensorRT engine: 0.004355669021606445
Inference time with the Pytorch model: 0.005367755889892578
MAE = 3.7649647310900036e-07
All completed!
```

4. 注意事项

- tensorrt只支持zero pad

  https://github.com/NVIDIA/TensorRT/issues/195

- tensorrt 7.0.0.11不支持batchnorm层，可以从 doc/pdf/TensorRT-Developer-Guide.pdf (Appendix A.1. TensorRT Layers章节) 查看tensorrt支持的层