深蓝学院
shenlanxueyuan.com

TensorRT 介绍

课程目标

理论部分

- TensorRT 基本概念
- TensorRT 基础使用教程
- TensorRT demo代码讲解
- TensorRT 模型转换工具

课程大纲

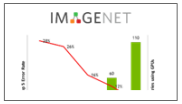
基础	1. TensorRT是什么 2. TensorRT工作流程介绍 3. TensorRT优化策略介绍 4. TensorRT的组成 5. TensorRT基本使用流程 6. TensorRT demo 代码-SampleMNIST
进阶	7. Dynamic Shape模式介绍 8. TensorRT模型转换 9. TensorRT版本选择

TensorRT是什么

COMPUTER VISION

OBJECT DETECTION

IMAGE CLASSIFICATION



SPEECH & AUDIO

VOICE RECOGNITION

LANGUAGE TRANSLATION

NATURAL LANGUAGE PROCESSING

RECOMMENDATION ENGINES

SENTIMENT ANALYSIS

深度学习训练框架

Caffe

Chainer

DL4J

Deeplearning4j

Mocha.jl

julia

KERAS

Microsoft CNTK

mxnet

Purine

TensorFlow

torch

MatConvNet

MINERVA

OpenDeep

Pylearn2

theano

深度学习inference框架

TensorRT

ONNX RUNTIME

TNN

MNN

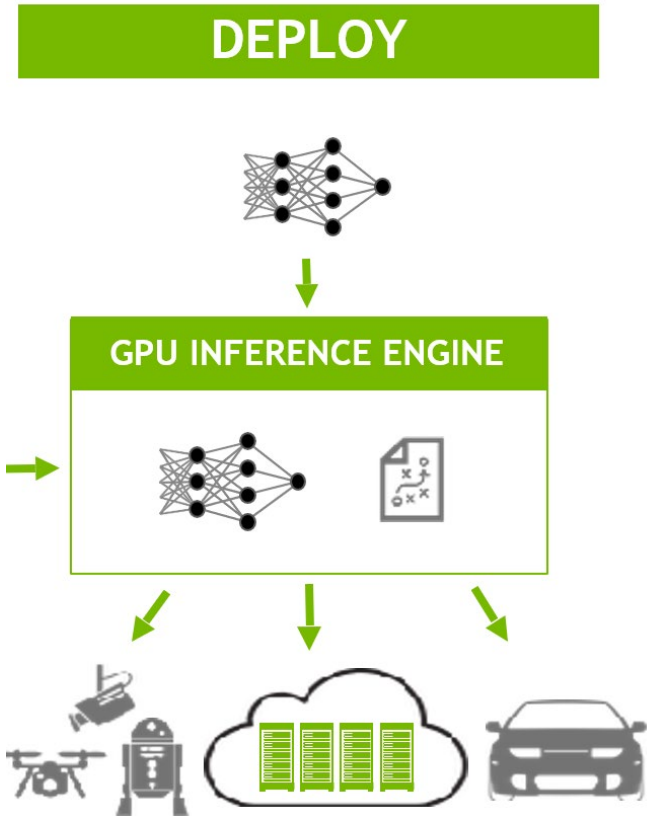
Mobile Neural Network

TensorFlow C

libtorch

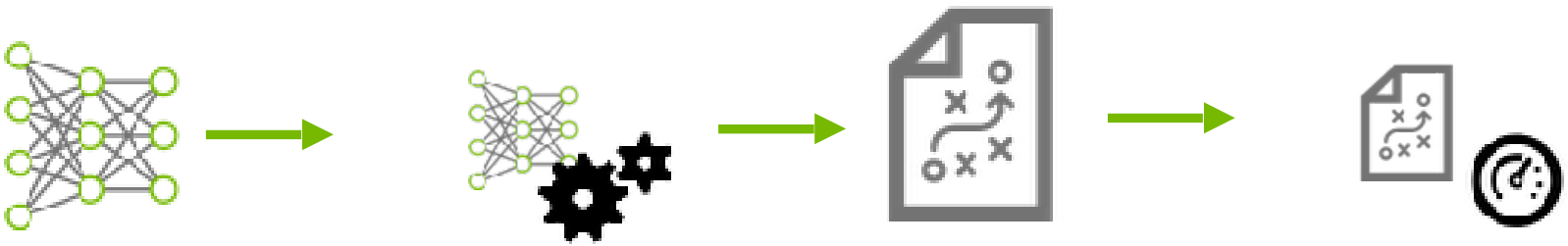
TensorRT是什么

1. 高性能深度学习推理 优化器和加速库;
2. 低延迟和高吞吐量;
3. 部署到超大规模数据中心、嵌入式或汽车产品。



TensorRT工作流程介绍

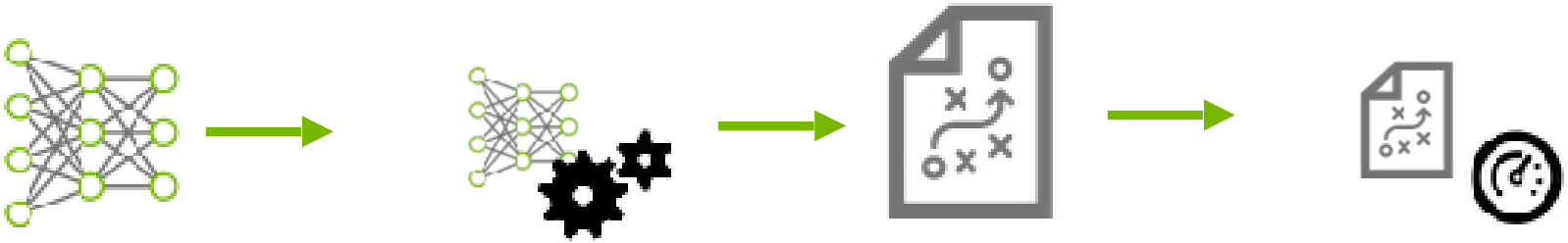
保存好的TRT模型文件可以从磁盘重新加载到TRT执行引擎中，不需要再次执行优化步骤。



32位模型　　TRT转换优化引擎　　TRT模型文件　　TRT执行引擎

TensorRT工作流程介绍

保存好的TRT模型文件可以从磁盘重新加载到TRT执行引擎中，不需要再次执行优化步骤。



32位模型

TRT转换优化引擎

- 模型转换
- Kernel自动调优
- 算子融合
- 低精度

TRT模型文件

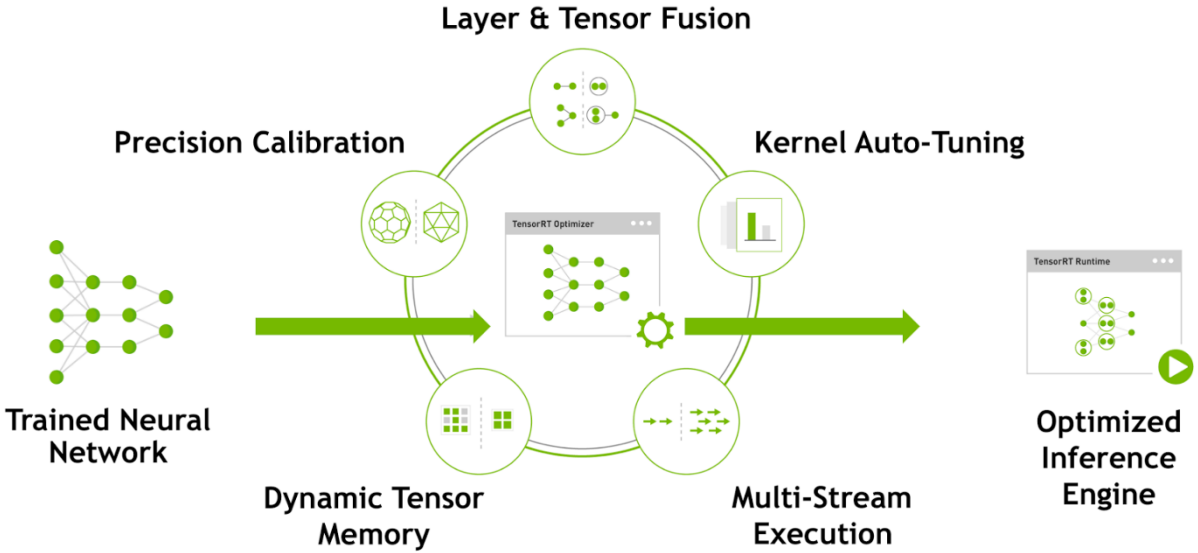
- 不同NVIDIA显卡架构间不兼容

TRT执行引擎

- 多流运行
- 显存优化

TensorRT优化策略介绍

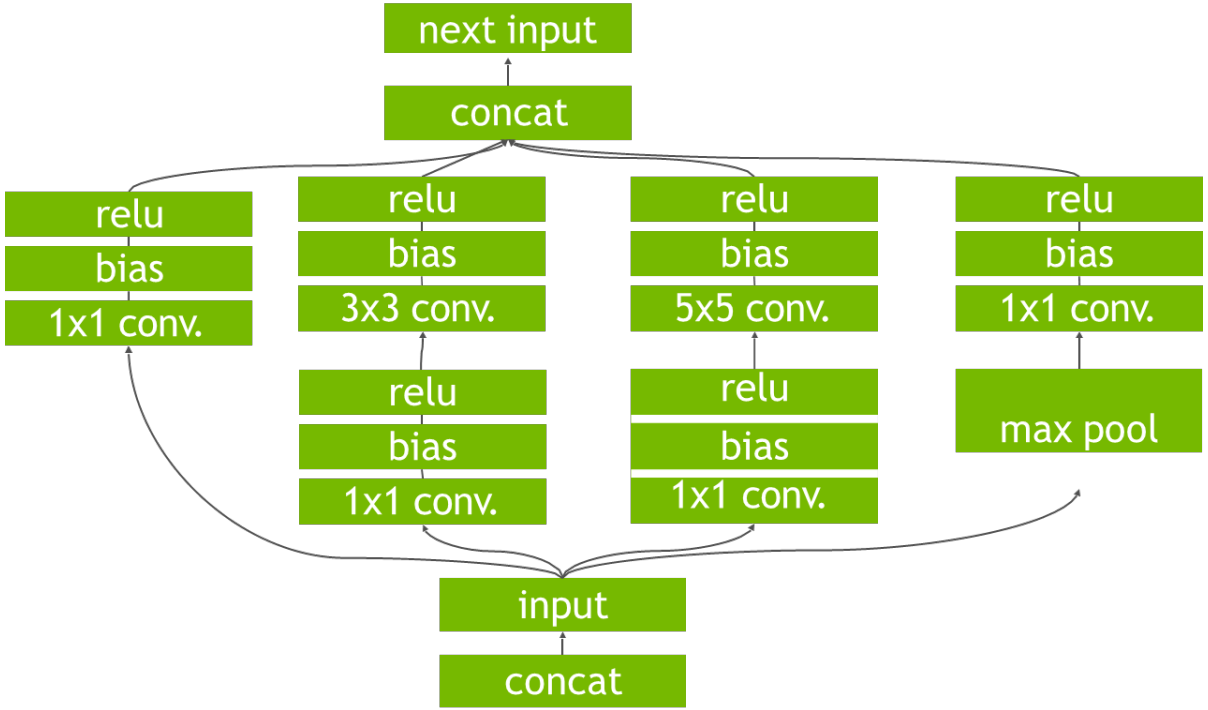
1. 低精度优化
2. Kernel 自动调优
3. 算子融合
4. 多流运行
5. 显存优化



TensorRT使用的优化策略介绍

3. 算子融合
4. 多流运行

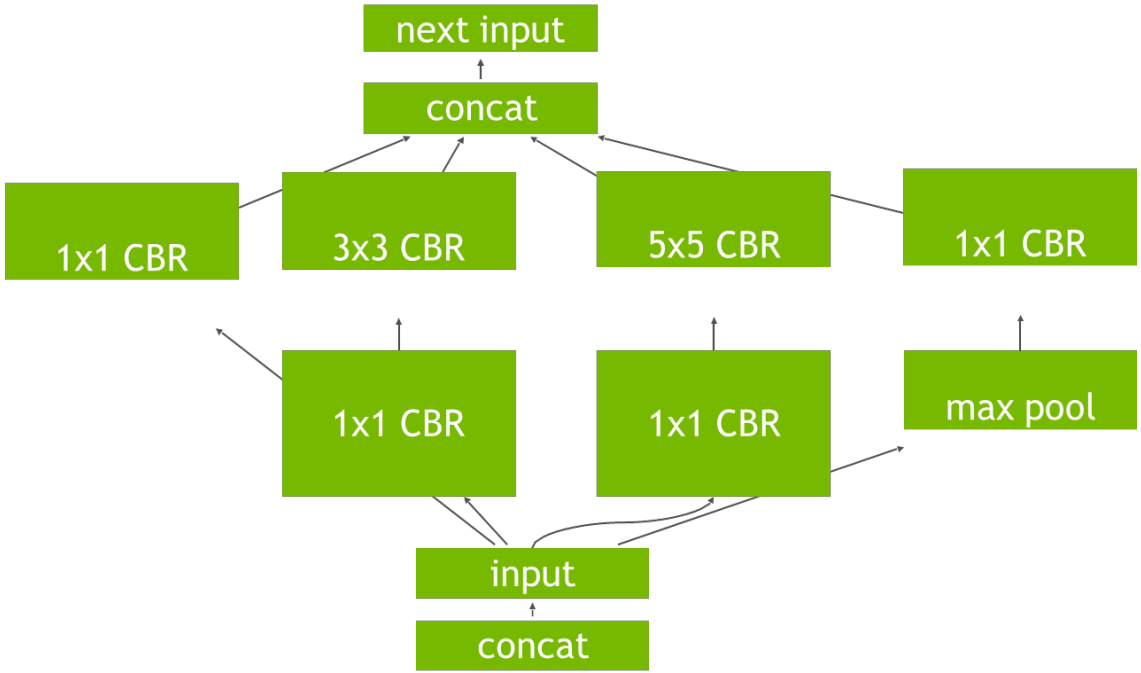
GoogleNet的初始结构



TensorRT使用的优化策略介绍

3. 算子融合
4. 多流运行

优化后的结构



CBR： conv, bias and ReLU

TensorRT的组成

官方提供的库

闭源，是TRT的核心部分

```
[wgyang@VM-121-9-centos ~/TRT_DIY_Inference/third_party/TensorRT]$ ls lib/ bin/ include/
bin/:
giexec  trtexec

include/:
NvCaffeParser.h  NvInferPlugin.h       NvInferRuntime.h        NvInferVersion.h  NvOnnxParser.h
NvInfer.h        NvInferPluginUtils.h  NvInferRuntimeCommon.h  NvOnnxConfig.h    NvOnnxParserRunti

lib/:
libnvcaffe_parser.a        libnvinfer.so         libnvinfer_plugin.so.6      libnvonnxparser.so
libnvcaffe_parser.so       libnvinfer.so.6       libnvinfer_plugin.so.6.0.1  libnvonnxparser.so.6
libnvcaffe_parser.so.6     libnvinfer.so.6.0.1   libnvinfer_plugin_static.a  libnvonnxparser.so.6
libnvcaffe_parser.so.6.0.1 libnvinfer_plugin.so  libnvinfer_static.a         libnvonnxparser_runt
[wgyang@VM-121-9-centos ~/TRT_DIY_Inference/third_party/TensorRT]$
```

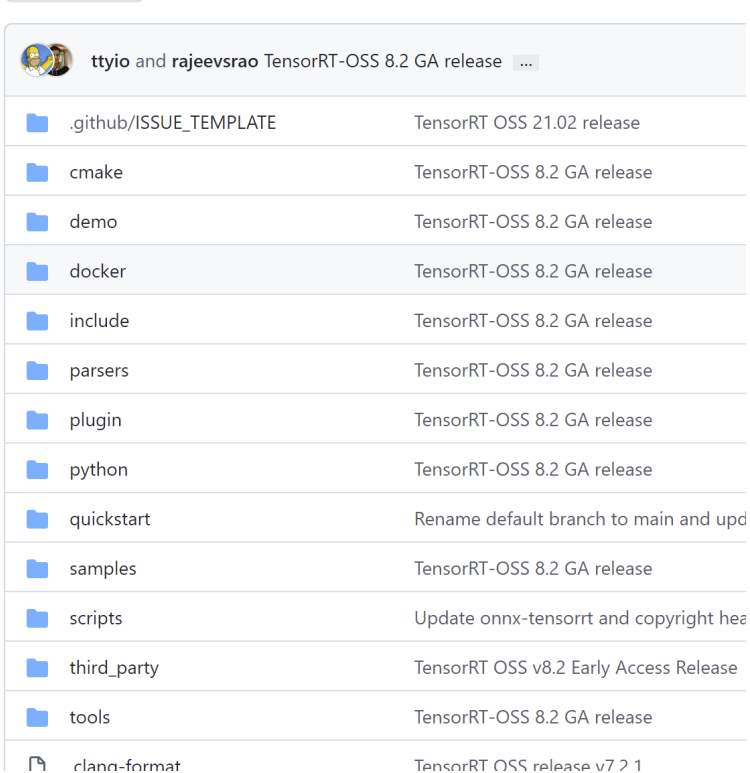
TensorRT的组成

官方提供的库

闭源，是TRT的核心部分

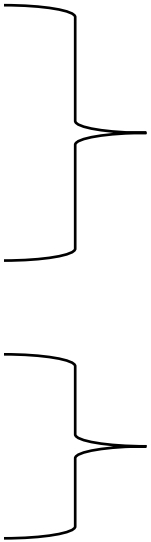
Github开源代码

- 模型解析器（caffe, onnx）
- 代码样例
- Plugin样例



TensorRT基本使用流程

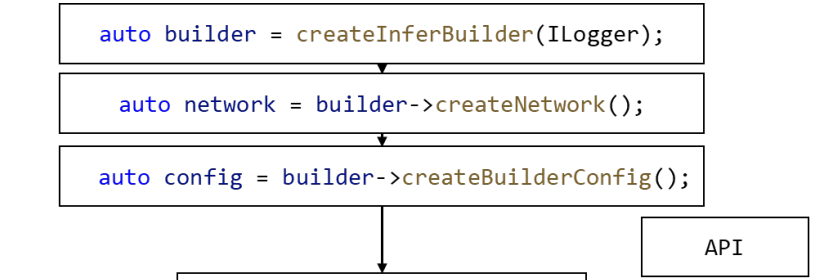
1. 创建Builder
2. 创建Network
3. 使用API or Parser 构建network
4. 优化网络
5. 序列化和反序列化模型
6. 传输计算数据（host->device）
7. 执行计算
8. 传输计算结果（device->host）



TRT转换优化引擎

TRT执行引擎

TensorRT基本使用流程



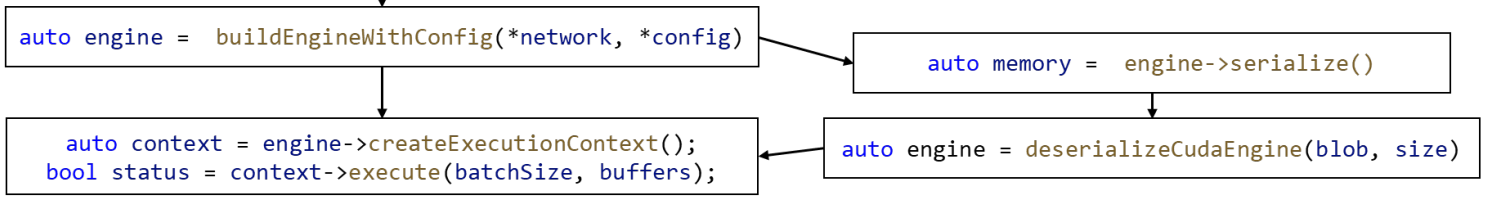
API

Parser

```
// Add convolution layer with 20 outputs and a 5x5 filter.
IConvolutionLayer* conv1 = network->addConvolution(
    *scale_1->getOutput(0), 20, DimsHW{5, 5},
    mWeightMap["conv1filter"], mWeightMap["conv1bias"]);
assert(conv1);
conv1->setStride(DimsHW{1, 1});
```

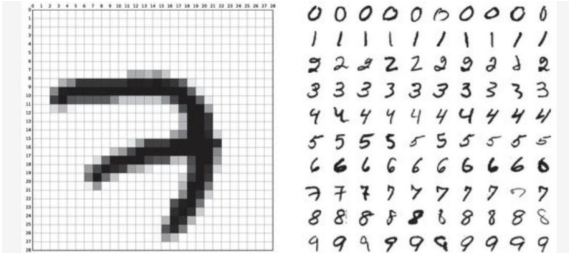
```
mEngine = nullptr;
const nvcaffeparser1::IBlobNameToTensor* blobNameToTensor
    = parser->parse(locateFile(mParams.prototxtFileName, mParams.dataDirs).c_str(),
        locateFile(mParams.weightsFileName, mParams.dataDirs).c_str(), *network,
        dataType == DataType::kINT8 ? DataType::kFLOAT : dataType);
```

```
// Build engine
builder->setMaxBatchSize(mParams.batchSize);
config->setMaxWorkspaceSize(16_MiB);
if (mParams.fp16)
{
    config->setFlag(BuilderFlag::kFP16);
}
```

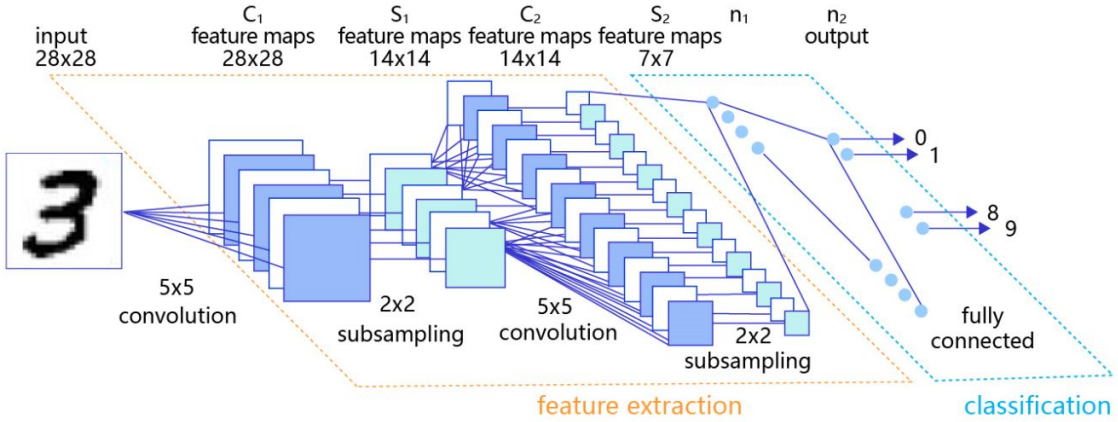


TensorRT demo 代码-SampleMNIST

MNIST是一个手写数字识别模型，输入一张手写数字的图像，然后识别图像中手写的是哪个数字。



网络结构



TensorRT demo 代码-SampleMNIST

```cpp
bool SampleMNIST::build() {
    auto builder = SampleUniquePtr<nvinfer1::IBuilder>(nvinfer1::createInferBuilder(gLogger.getTRTLogger()));
    auto network = SampleUniquePtr<nvinfer1::INetworkDefinition>(builder->createNetwork());
    auto config = SampleUniquePtr<nvinfer1::IBuilderConfig>(builder->createBuilderConfig());
    auto parser = SampleUniquePtr<nvcaffeparser1::ICaffeParser>(nvcaffeparser1::createCaffeParser());

    constructNetwork(parser, network);

    builder->setMaxBatchSize(mParams.batchSize);
    config->setMaxWorkspaceSize(16_MiB);
    config->setFlag(BuilderFlag::kGPU_FALLBACK);
    config->setFlag(BuilderFlag::kSTRICT_TYPES);
    if (mParams.fp16)
        config->setFlag(BuilderFlag::kFP16);
    if (mParams.int8)
        config->setFlag(BuilderFlag::kINT8);

    // samplesCommon::enableDLA(builder.get(), config.get(), mParams.dlaCore);

    mEngine = std::shared_ptr<nvinfer1::ICudaEngine>(
        builder->buildEngineWithConfig(*network, *config), samplesCommon::InferDeleter());

    return true;
}
```

TensorRT demo 代码-SampleMNIST

Caffe Parser方式构建

```
void SampleMNIST::constructNetwork(
    SampleUniquePtr<nvcaffeparser1::ICaffeParser>& parser, SampleUniquePtr<nvinfer1::INetworkDefinition>& network)
{
    const nvcaffeparser1::IBlobNameToTensor* blobNameToTensor = parser->parse(
        mParams.prototxtFileName.c_str(), mParams.weightsFileName.c_str(), *network, nvinfer1::DataType::kFLOAT);

    for (auto& s : mParams.outputTensorNames)
    {
        network->markOutput(*blobNameToTensor->find(s.c_str()));
    }
```

TensorRT demo 代码-SampleMNIST

API方式构建

```
bool SampleMNISTAPI::constructNetwork(SampleUniquePtr<nvinfer1::IBuilder>& builder,
    SampleUniquePtr<nvinfer1::INetworkDefinition>& network, SampleUniquePtr<nvinfer1::IBuilderConfig>& config)
{
    // Create input tensor of shape { 1, 1, 28, 28 }
    ITensor* data = network->addInput(
        mParams.inputTensorNames[0].c_str(), DataType::kFLOAT, Dims3{1, mParams.inputH, mParams.inputW});
    assert(data);

    // Create scale layer with default power/shift and specified scale parameter.
    const float scaleParam = 0.0125f;
    const Weights power{DataType::kFLOAT, nullptr, 0};
    const Weights shift{DataType::kFLOAT, nullptr, 0};
    const Weights scale{DataType::kFLOAT, &scaleParam, 1};
    IScaleLayer* scale_1 = network->addScale(*data, ScaleMode::kUNIFORM, shift, scale, power);
    assert(scale_1);

    // Add convolution layer with 20 outputs and a 5x5 filter.
    IConvolutionLayer* conv1 = network->addConvolution(
        *scale_1->getOutput(0), 20, DimsHW{5, 5}, mWeightMap["conv1filter"], mWeightMap["conv1bias"]);
    assert(conv1);
    conv1->setStride(DimsHW{1, 1});
```

TensorRT demo 代码-SampleMNIST

```cpp
bool SampleMNIST::infer() {
    // Create RAII buffer manager object
    samplesCommon::BufferManager buffers(mEngine, mParams.batchSize);
    auto context = SampleUniquePtr<nvinfer1::IExecutionContext>(mEngine->createExecutionContext());

    // Read the input data into the managed buffers

    // Create CUDA stream for the execution of this inference.
    cudaStream_t stream;
    CHECK(cudaStreamCreate(&stream));

    // Asynchronously copy data from host input buffers to device input buffers
    buffers.copyInputToDeviceAsync(stream);

    // Asynchronously enqueue the inference work
    if (!context->enqueue(mParams.batchSize, buffers.getDeviceBindings().data(), stream, nullptr))
        return false;
    // Asynchronously copy data from device output buffers to host output buffers
    buffers.copyOutputToHostAsync(stream);

    // Wait for the work in the stream to complete
    cudaStreamSynchronize(stream);

    // Release stream
    cudaStreamDestroy(stream);
```

进阶：Dynamic Shape模式介绍

TRT 6.0 版本之前，只支持固定大小输入。

implicit(隐式) batch

Build 阶段设置：

IBuilder:: createNetwork();

IBuilder:: setMaxBatchSize(maxBatchSize);

Infer阶段：

enqueue(batchSize, data, stream, nullptr);

进阶：Dynamic Shape模式介绍

TRT6.0 后，支持动态大小输入。

explicit(显式) batch

Build 阶段设置：

```
IBuilder::createNetworkV2(1U <<
static_cast<int>(NetworkDefinitionCreationFlag::kEXPLICIT_BATCH))
builder->setMaxBatchSize(maxBatchSize);
IOptimizationProfile* profile = builder.createOptimizationProfile();
    profile->setDimensions("foo", OptProfileSelector::kMIN, Dims3(3,100,200);
    profile->setDimensions("foo", OptProfileSelector::kOPT, Dims3(3,150,250);
    profile->setDimensions("foo", OptProfileSelector::kMAX, Dims3(3,200,300);
    config.addOptimizationProfile(profile)

context.setOptimizationProfile(0)
```

进阶：Dynamic Shape模式介绍

TRT6.0 后，支持动态大小输入。

Infer阶段：

```
context->setBindingDimensions(i, input_dim);
context->allInputDimensionsSpecified();
context->enqueueV2(data, stream, nullptr);
```

进阶： TensorRT模型转换

ONNX： https://github.com/NVIDIA/TensorRT/tree/main/parsers

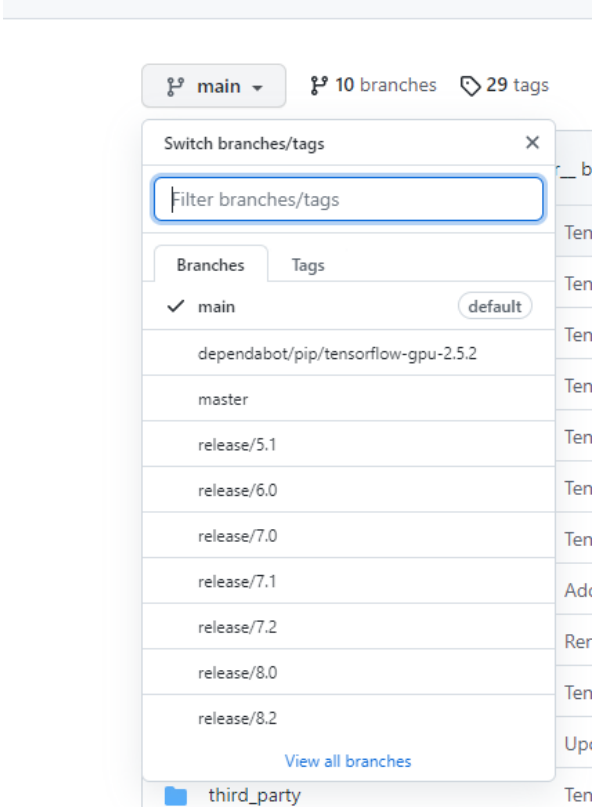
Pytorch： https://github.com/NVIDIA-AI-IOT/torch2trt

TensorFlow：

https://github.com/tensorflow/tensorflow/tree/1cca70b80504474402215d2a4e55bc44621b691d/tensorflow/compiler/tf2tensorrt

Tencent Forward： https://github.com/Tencent/Forward

进阶： TensorRT版本选择



- 5.1 不建议使用
- Tesla P4 6.0
- Tesla T4 7.2
- NVIDIA A10 8.2

深蓝学院
shenlanxueyuan.com

感谢聆听！

Thanks for Listening