OCR Acceleration

and NVIDIA Ecosystem

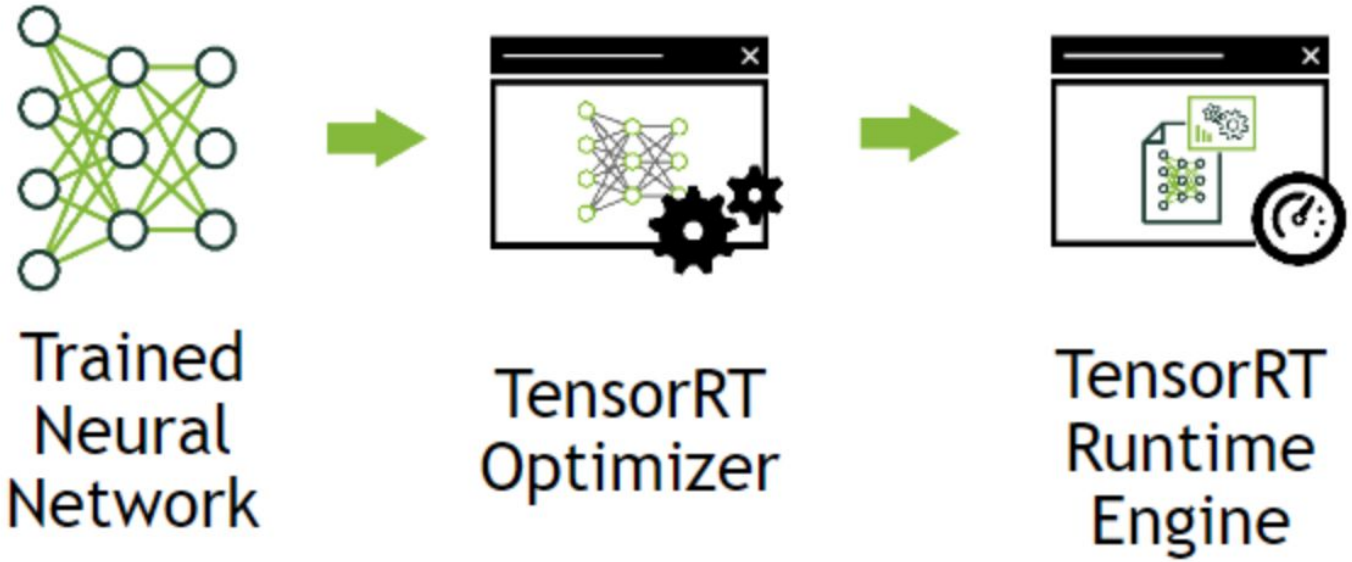
Yang.Bai

Outline:

- Using **TensorRT** to Accelerate OCR on NVIDIA GPUs
  - Text Detection Acceleration
  - RCNN Recognition Acceleration
  - Configuration of Acceleration
- Using **MNN** to Accelerate OCR on mobile devices
- **NVIDIA** Ecosystem

What is TensorRT?

A high-performance neural network inference optimizer and runtime engine for production deployment.



Why TensorRT?

1. Using the training framework to perform inference is easy, but lower performance on a given GPU
2. Training frameworks tend to implement more general purpose code which stress generality, focus on efficient training
3. Labor-intensive and specialized knowledge to reach a high-level of performance on a modern GPU, not translate fully to other GPUs

TensorRT: by combining a high-level API that abstracts away specific hardware details and optimizes inference

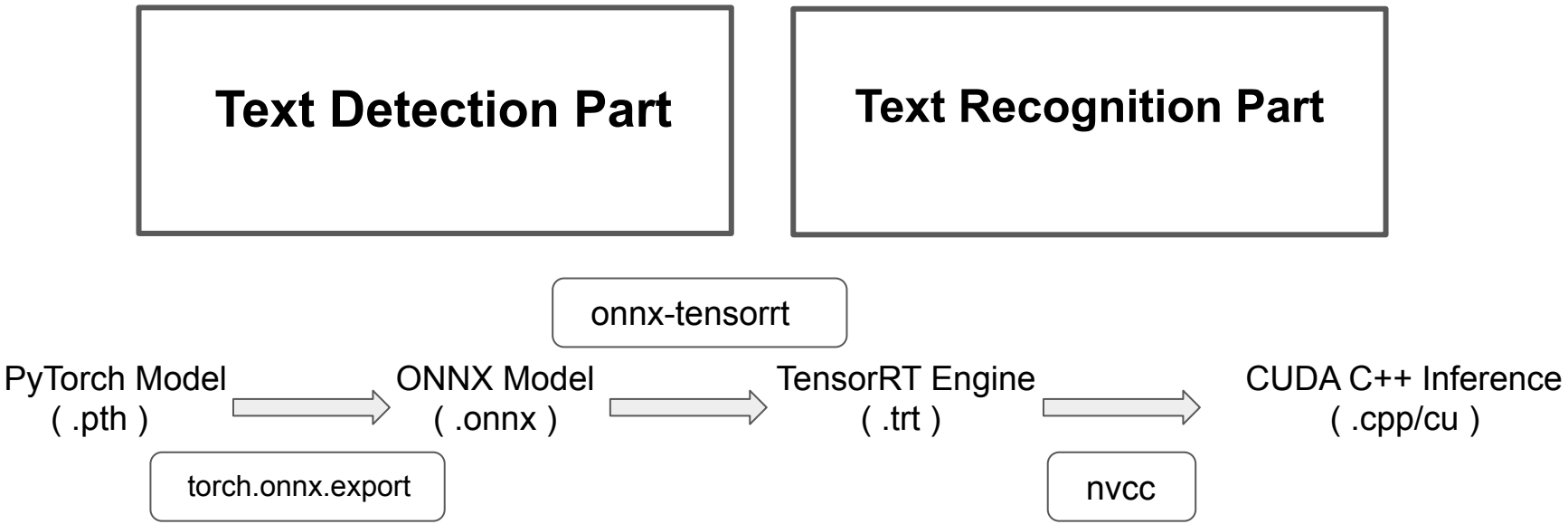
TensorRT Optimizations and Performance

1. Weight & Activation Precision Calibration
2. Layer & Tensor Fusion
3. Kernel Auto-Tuning
4. Dynamic Tensor Memory
5. Multi-Stream Execution

5 Critical factors to measure software:

1. **Throughput**
   a. The volume of output within a given period
   b. Measured in inference / second or samples / second
2. **Efficiency**
   a. Amount of throughput delivered per unit-power
   b. Expressed as performance/watt
3. **Latency**
   a. Time to execute an inference
   b. Measured in milliseconds
4. **Accuracy**
   a. Deliver the correct answer
   b. Different tasks: top-5 or top-1 or mAP
5. **Memory Usage**
   a. Host and Device memory

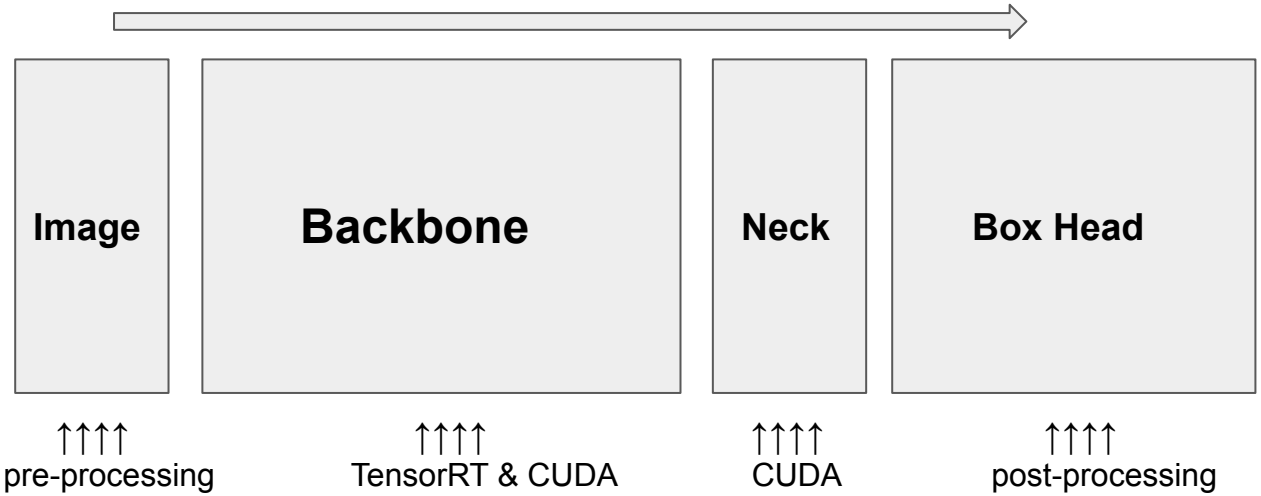
OCR Acceleration Pipeline by TensorRT



Text Detection Configuration

- input batch size = 1
- input shape = (1, 3, 672, 768)
- output shape = (1, 3, 672, 768)

↑↑↑↑: Accelerate



Performance Comparison

1. **Image Pre-processing**
   a. BGR -> RGB
   b. HWC -> CHW
   c. Normalization
2. **Backbone**
   a. Upsampling Operator implemented by CUDA
   b. Computation Graph optimized by TensorRT
3. **Neck**
   a. text computation
   b. seg_map2 computation
4. **Box Head**
   a. resize
   b. addWeight
   c. findContours
   d. drawContours

1. **Image pre-processing**
   a. **PyTorch with opencv python:** 0.09889s
   b. **TensorRT :**
      i. **cuda on gpu:** 0.016267s
      ii. **opencv on cpu:** 0.026738s
   c. **Speedup: 6.079x**
2. **Backbone**
   a. **Upsampling operator is compulsory**
3. **Neck**
   a. **Pytorch with opencv python:** 0.51290s
   b. **C++ on CPU:** 0.062755
   c. **CUDA on GPU:** 0.052585
   d. **Speedup: 9.753x**
4. **Box Head for visualization**
   a. **Slightly slow (CPU v.s. GPU)**

Text detection (mobilenet_v2_035)

Input Image

TensorRT Problem

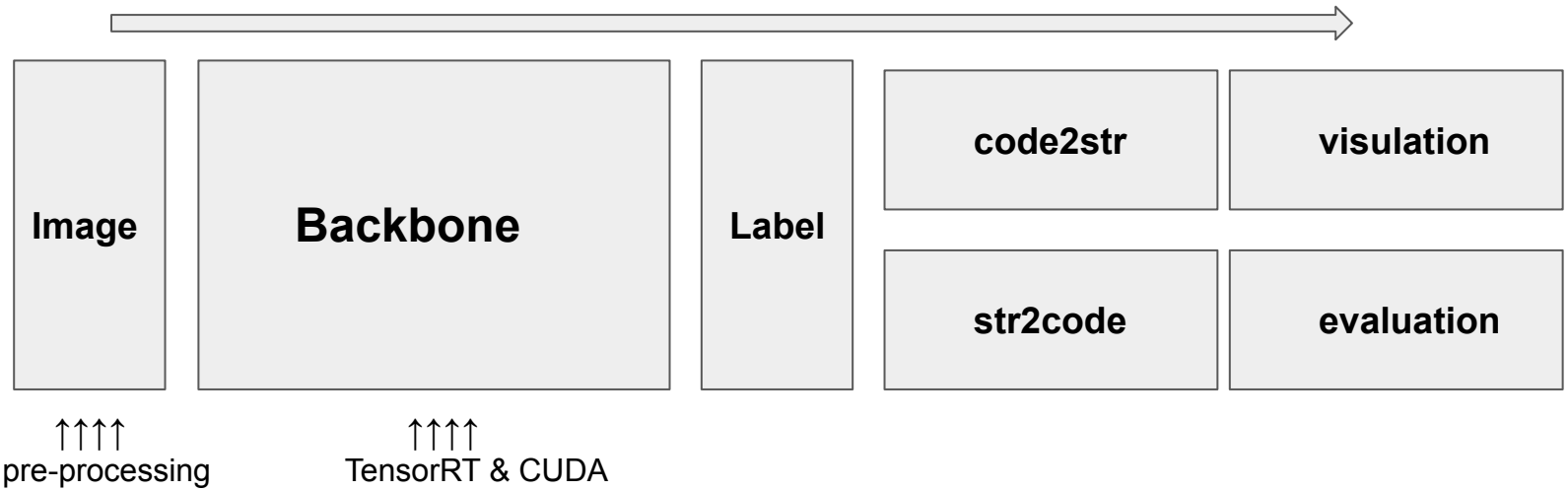
TensorRT Result

PyTorch Result

Text Recognition Configuration

- input_batch_size = 1
- input_shape = (1, 3, 32, 400)
- output_shape = (6141, 1) -> depends on **alphabets.txt**

↑↑↑↑: Accelerate



Text Recognition(reslite18)

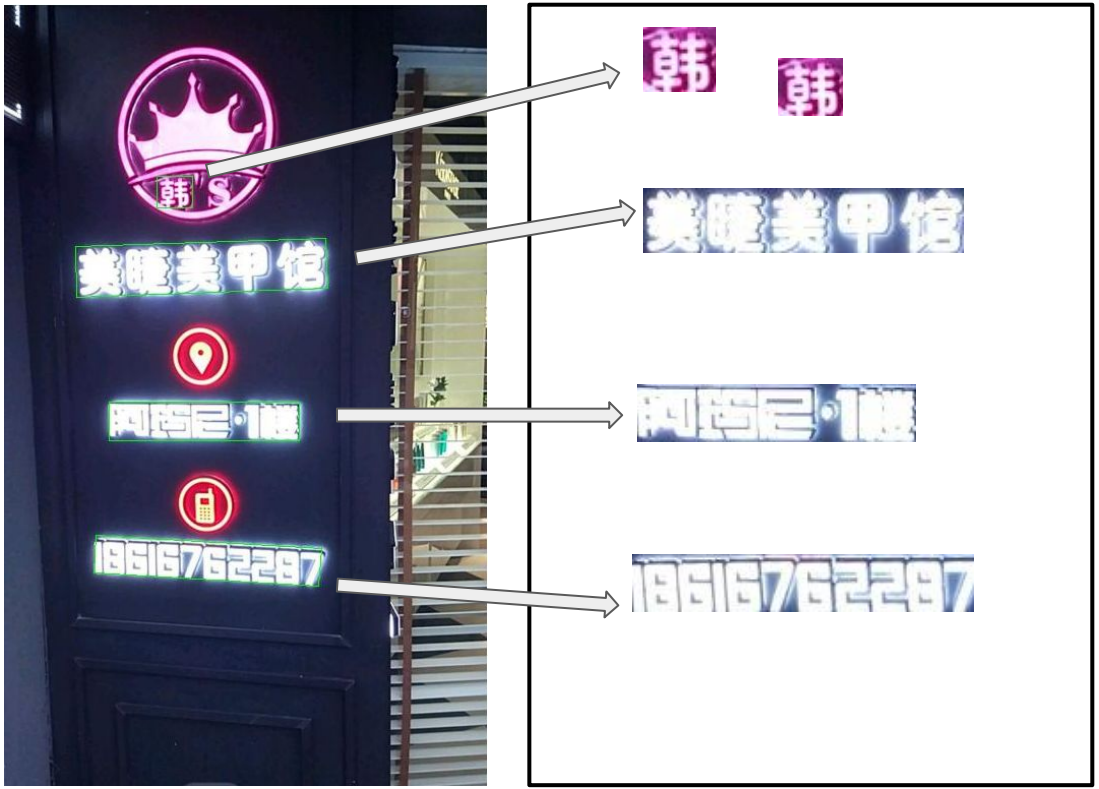
Speedup: **10.411x**

PyTorch: 0.48974s

232,251,285,251,285,297,232,297★韩
112,356,482,343,484,417,115,430★美睛美甲馆
162,583,443,583,443,640,162,640★阿已楼
143,780,475,800,472,855,140,835★旧日7日2297

TensorRT: 0.04704s

```
(base) root@640ee047b271:/home/Yang.Bai/OCR-TensorRT#
Loading engine file:./engine/ctrcnn-fp32.trt
 Create the engine from ./engine/ctrcnn-fp32.trt succe
[07/31/2020-18:02:10] [W] [TRT] Current optimization p
i=0 tensor's name:input_rcnn dim:(1,3,32,400) size:384
i=1 tensor's name:output_rcnn dim:(1,6141,1,100) size:
buffers'index, input id: 0 output id: 1
det_out/det_box0.jpg
 starting inference ...
[07/31/2020-18:02:10] [W] [TRT] Explicit batch network
prediction_label: 韩
det_out/det_box1.jpg
 starting inference ...
[07/31/2020-18:02:10] [W] [TRT] Explicit batch network
prediction_label: 美睛美甲馆
det_out/det_box2.jpg
 starting inference ...
[07/31/2020-18:02:10] [W] [TRT] Explicit batch network
prediction_label: 阿已
det_out/det_box3.jpg
 starting inference ...
[07/31/2020-18:02:10] [W] [TRT] Explicit batch network
prediction_label: 旧日7日2287
total time:
```

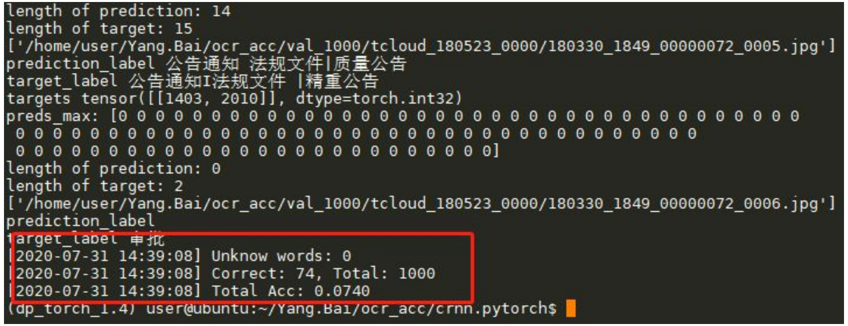


Detected Image

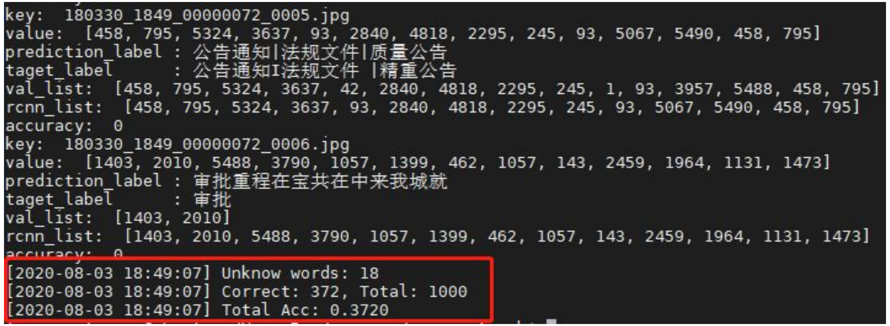
Cropped Bbox

Evaluation for Accuracy

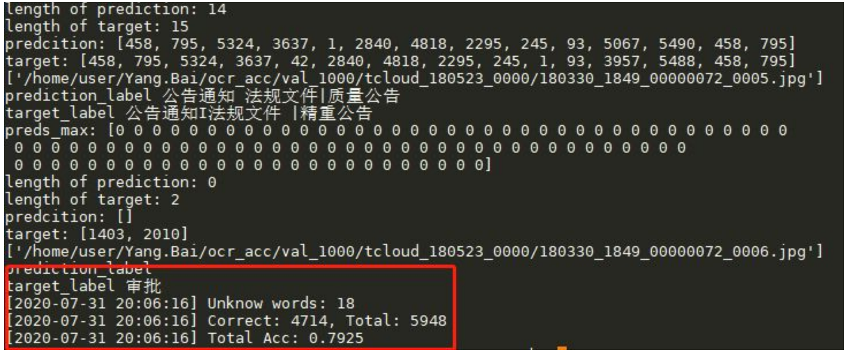
PyTorch：



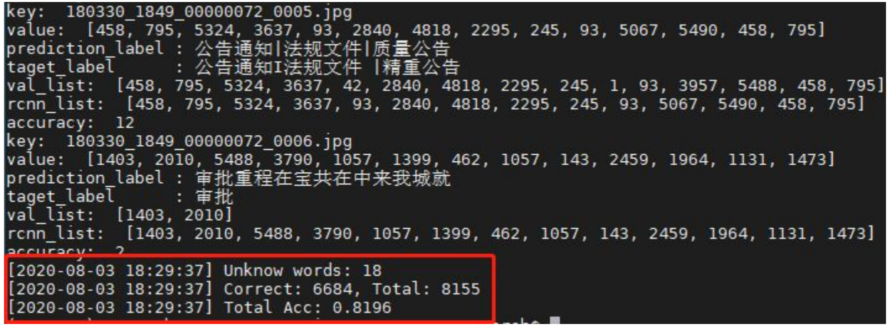
TensorRT:



Change the Criteria



Change the Criteria



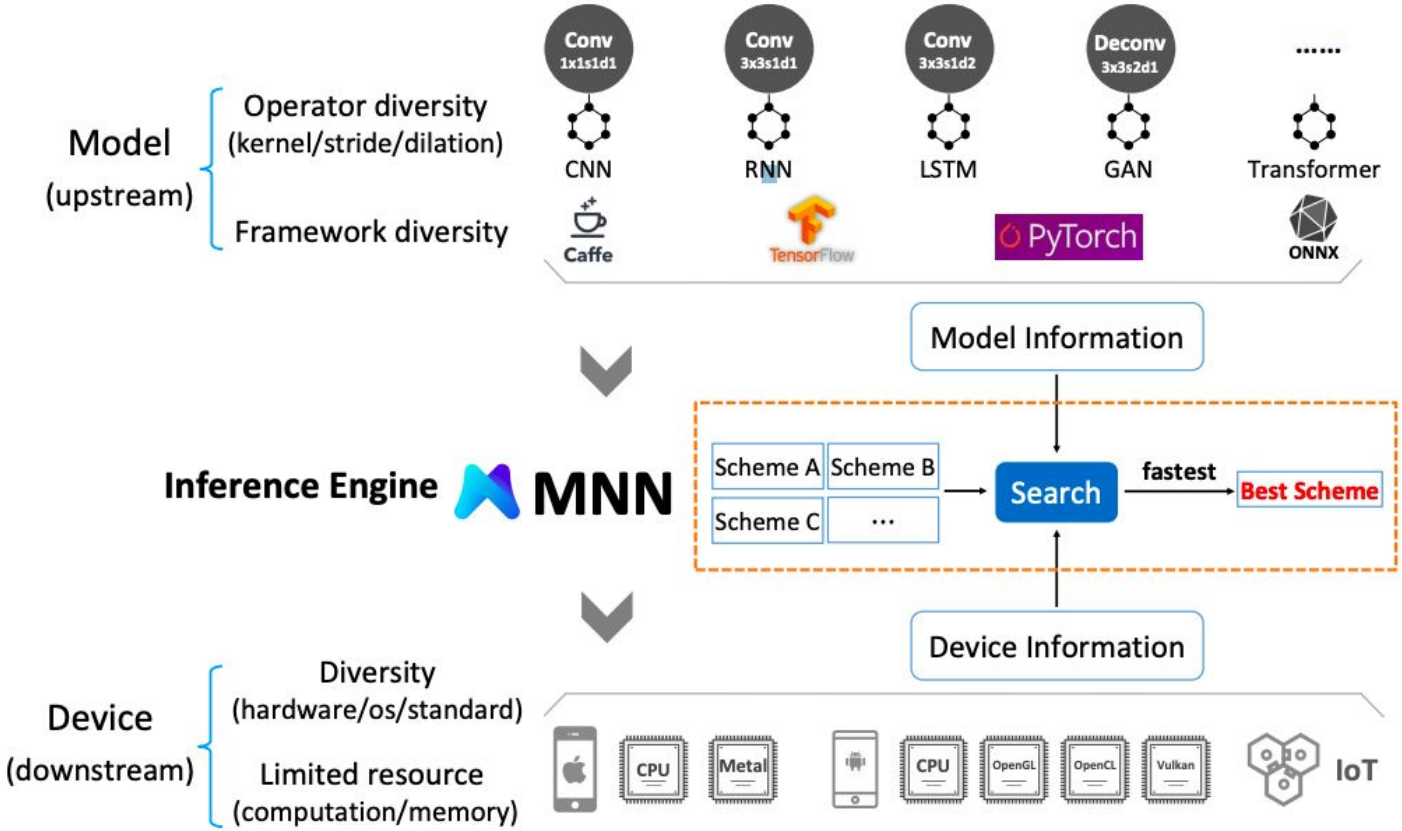
Configuration of Acceleration for NVIDIA GPUs

1. Original Version: Pytorch1.4 + CUDA 10.1 with cuDNN

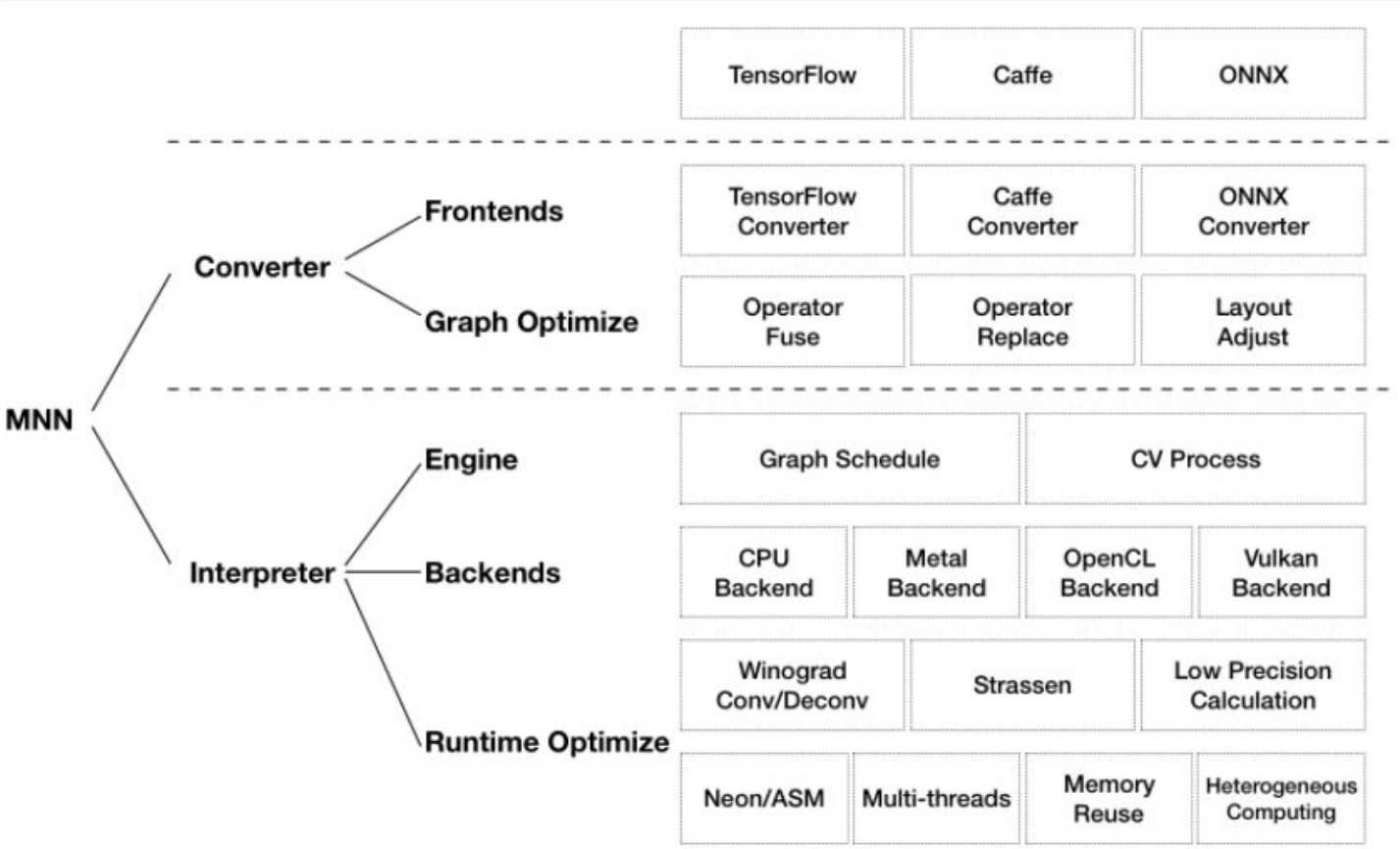
2. Accelerated Version: TensorRT 7.0 + CUDA 10.0 with cuDNN

3. More details can be found: OCR_Acceleration_Documentation

MNN: Mobile Neural Network



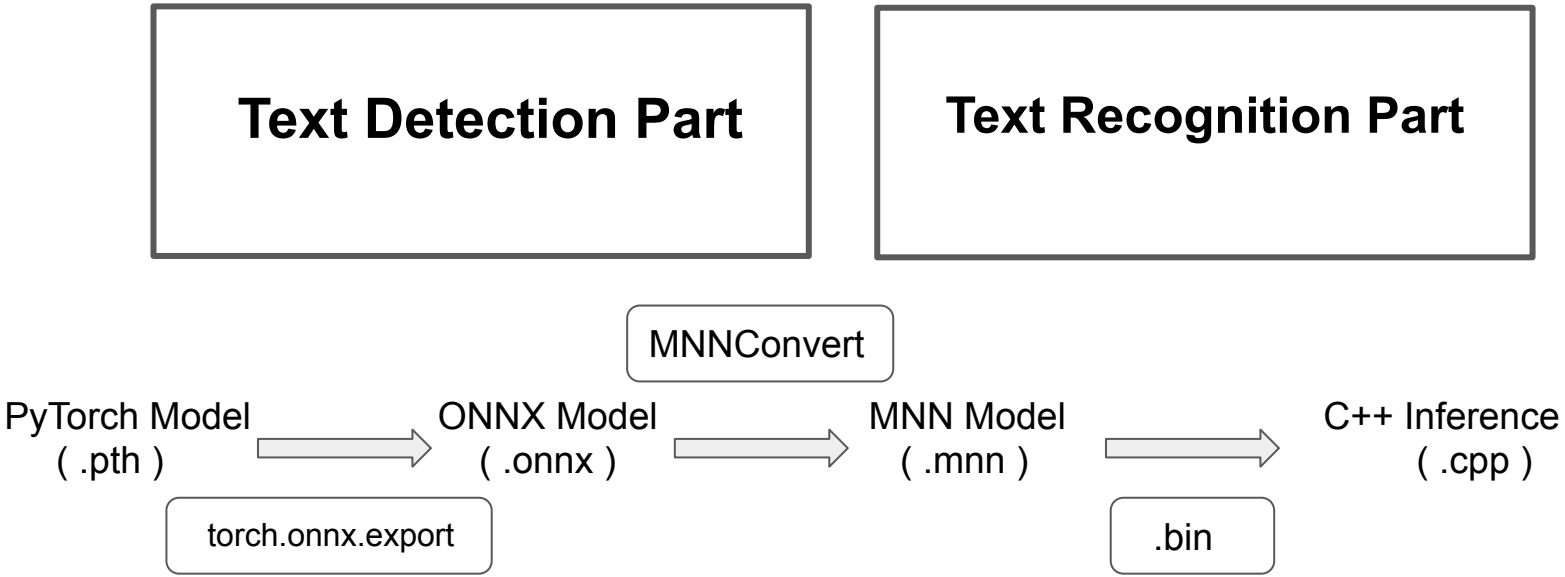
MNN Architecture



Converter and Interpreter

- Converter:
  - Frontends and Graph Optimize
    - Fronteds: Tensorflow, Tensorflow lite, Caffe and ONNX
    - Graph Optimize: Operator fusion, substituion, layout adjustment
- Interpreter:
  - Engine and Backends
    - loading of the model and the scheduling of the computational graph
    - **Backends (so important)**:
      - Winograd Algorithm in Conv & Deconv
      - Strassen Algorithm in GEMM
      - Low-precision Calculation
      - Neon Optimization
      - Hand-written assembly
      - Multi-thread optimization
      - Memory Reuse
      - Heterogeneous Computing

OCR Acceleration Pipeline by MNN



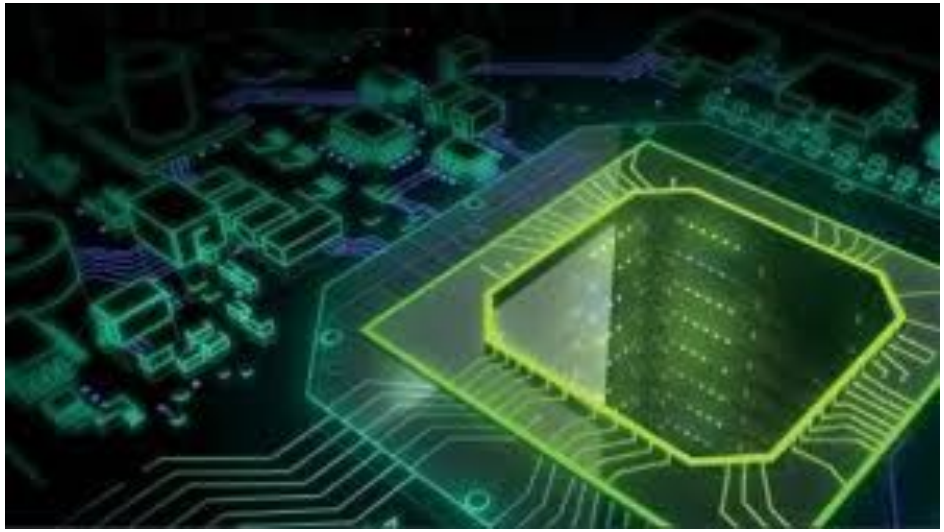
Summary about Acceleration and planning

SmartAccel Project:

1. PyTorch model -> ONNX model -> TensorRT Engine
2. Dynamic shapes for memory allociation
3. High Performance C++ Deep Learning Library for Acceleration

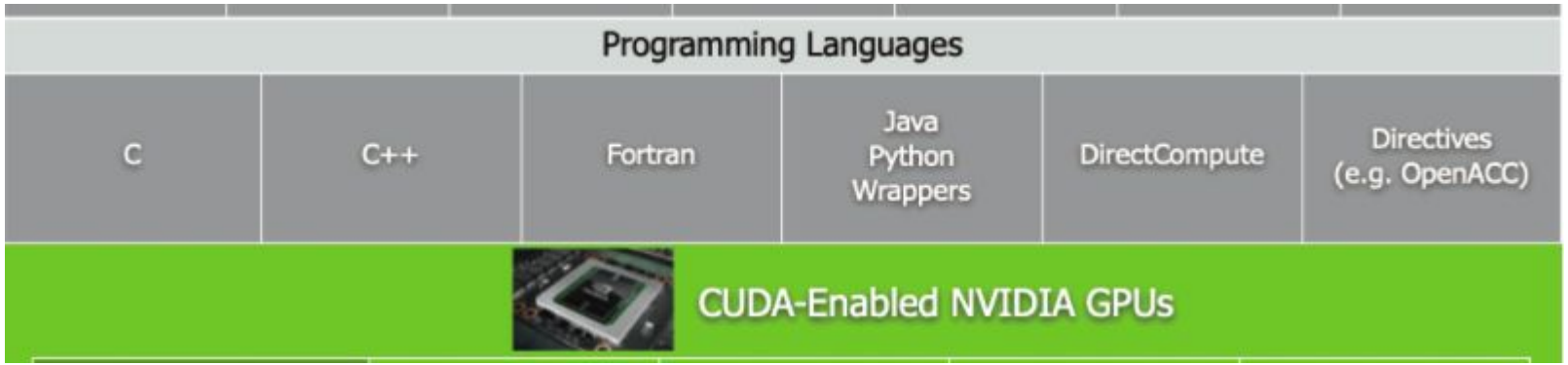
NVIDIA Ecosystem

- CUDA
- OpenCL
- TensorRT
- TensorCore
- cuDNN
- cuBLAS



CUDA (Compute Unified Device Architecture)

CUDA is a parallel computing platform and programming model developed by NVIDIA for general computing on grahical processing units (GPUs)



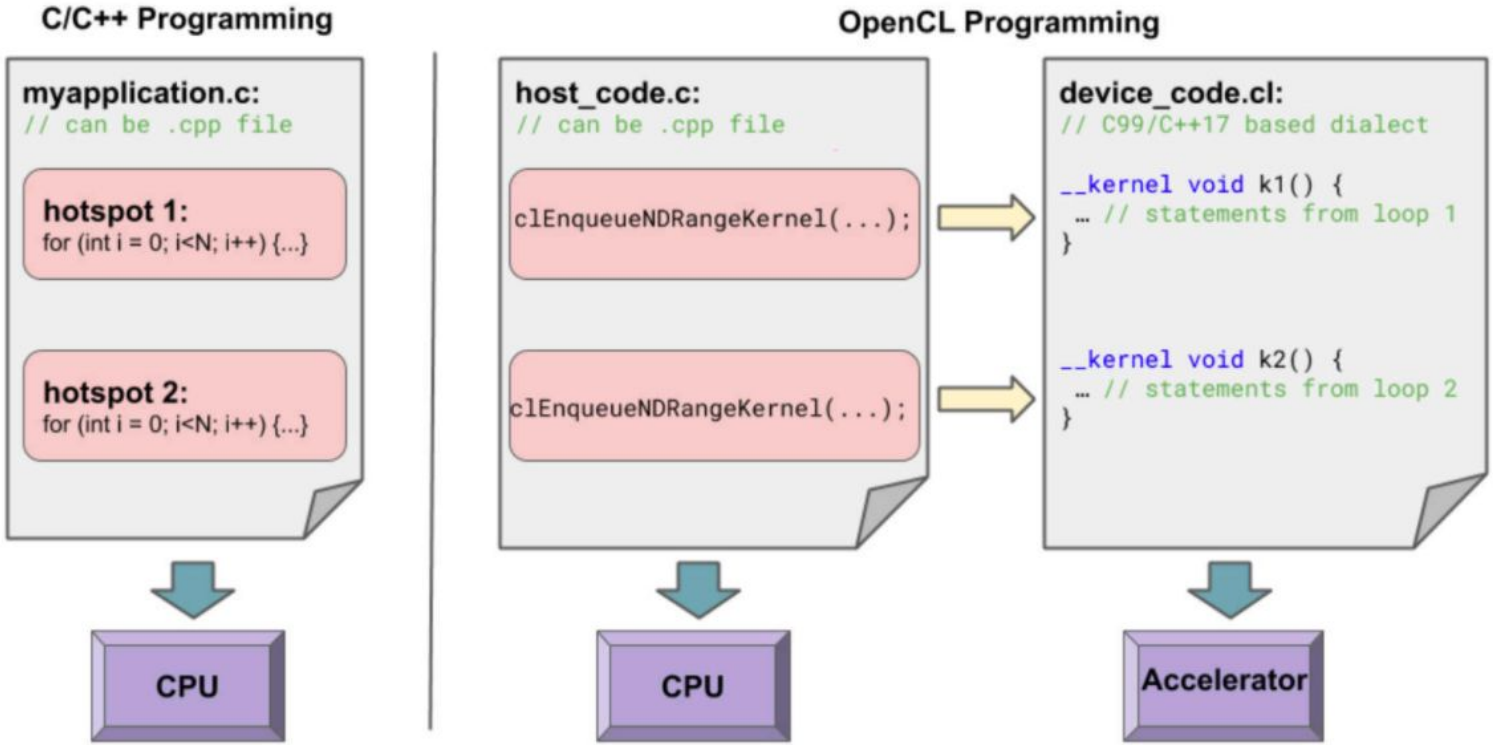
OpenCL

Open standard for parallel programming of heterogeneous systems

What's the difference between CUDA and OpenCL:

- CUDA -> NVIDIA GPUs
- OpenCL -> Massively Parallel Processor
    - CPU
    - GPU
    - FPGA
    - DSP
    - AI/Tensor HW
    - Custom Hardware

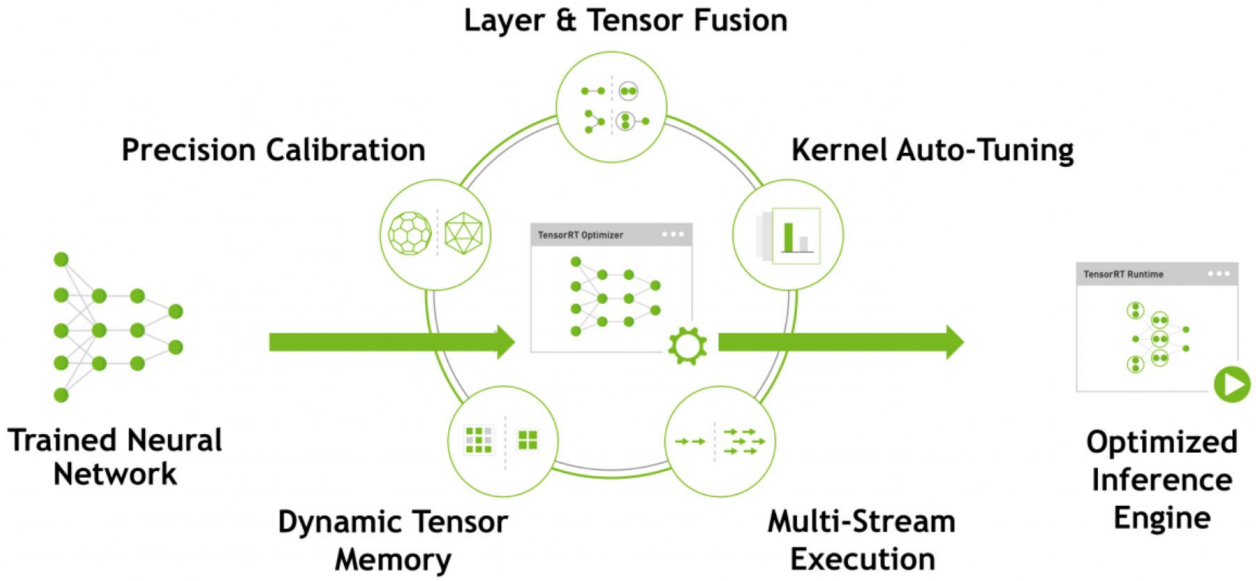
Programming Model are same!



Traditional vs OpenCL programming paradigm

TensorRT

A high-performance neural network inference optimizer and runtime engine for production deployment.



TensorCore

- 1st Generation, Volta Architecture
  - InputDataType: FP16
  - Accumulator: FP16 or FP32
  - Matrix Scale: 8 * 8 * 4 (m * n * k)
- 2nd Generation, Turing Architecture
  - New Support: int8, int4, int1
  - Matrix Scale: 8 * 8 * 4 (m * n * k) & 16 * 8 * 8
- 3rd Generation, Ampere Architecture
  - Domain: Specific for Deep Learning and High Performance Computing
  - New Support: bfloat (BF16), tensorfloat32(TF32), double(FP64)
  - New Feature: Sparsity expect for int1 & double(FP64)

TensorCore

Typically, the notion is that CUDA cores are slower, but offer more significant precision. Whereas a Tensor cores are lightning fast, however lose some precision along the way.

	NVIDIA A100	NVIDIA Turing	NVIDIA Volta
Supported Tensor Core Precisions	FP64, TF32, bfloat16, FP16, INT8, INT4, INT1	FP16, INT8, INT4, INT1	FP16
Supported CUDA® Core Precisions	FP64, FP32, FP16, bfloat16, INT8	FP64, FP32, FP16, INT8	FP64, FP32, FP16, INT8

cuDNN / cuBLAS

- cuDNN: a GPU-accelerated library of primitives for DNN. It provides highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization, and activation layers.

- cuBLAS: an implementation of BLAS (Basic Linear Algebra Subprograms) on top of the CUDA runtime.

Thanks