# 英伟达-阿里云异构计算

# TensorRT 加速AI推理Hackathon 2021

# 项目报告书

**参赛主题：**SOLOv2\_R101\_3x模型的tensorRT压缩和优化

**参赛单位：格灵深瞳有限公司、中国科学院光电技术研究所、北京航空航天大学、广州理工事业有限公司**

**团队名称：牛魔王们**

**团队成员：**孙启昌、邵林松、罗嗣梧、王睿

## 摘要

本项目选择SOLOv2分割网络重的SOLOv2\_R101\_3x模型。使用TensorRT API来搭建网络，有些层需要plugin，例如group norm层。

（扼要描述，包括：

* 选择了什么模型，
* 进行TensorRT加速的技术路线，例如：onnx导入加plugin开放，INT8量化等。
* 取得的成果，例如，在batchsize=16时，达到了最高加速比，时延/吞吐时多少。
* 工作的难点和创新点是什么，例如，开发了3个plugin，支持模型变长输入等。
* 后续计划等

）

## 一．背景介绍

（建议使用计划书里面关于模型的介绍即可。）

SOLOv2构建一种简单、直接、快速的性能强大的实例分割框架。沿用SOLO的框架，改进了分割头。将分割分支分解为mask kernel branch和mask feature branch， 最后通过一个简单的卷积得到最终的分割实例。同时，提出了Matrix NMS来提高inference的推理时间。

## 二．开发内容

### 2.1 模型结构和基本情况

（介绍初始模型的训练框架，模型结构，模型精度，模型包括的主要网络层，模型在原框架中的性能（此处可简要提及，后边有详细对比）。）

SOLOv2使用ResNet101作为骨干网络来进行特征提取；neck采用FPN，更好地融合骨干网络所给出的特征；

SOLOv2是SOLO基础上，引入动态机制，动态学习目标分割器的mask head。将mask branch解耦为mask kernel branch以及mask feature branch，学习卷积核权重。如图1所示。

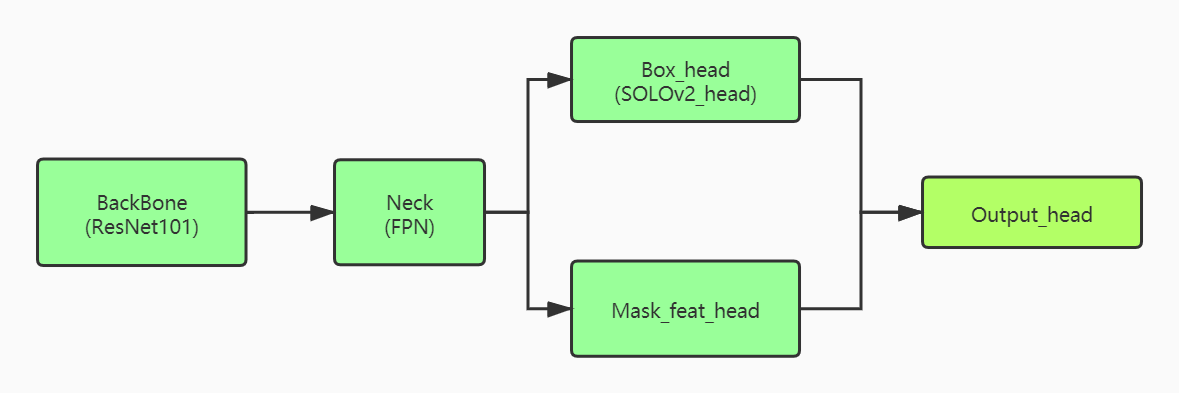
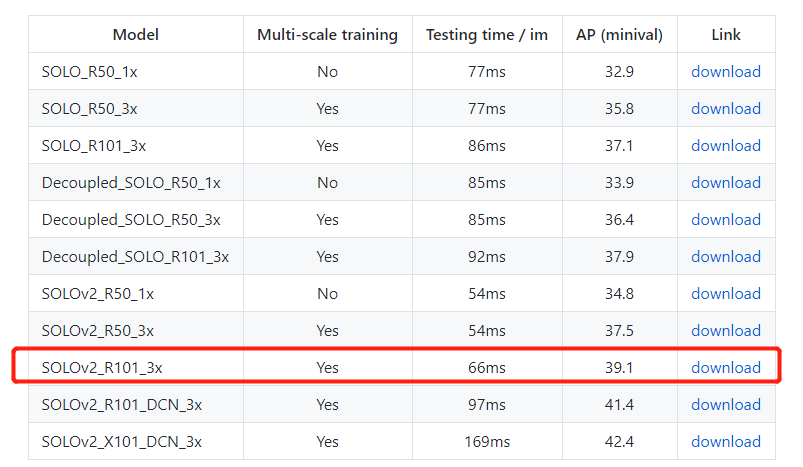


图1 SOLO结构

原始模型性能



### 2.2 技术路线选择

（介绍选择了什么样的技术路线，为什么。可以从几个方面说明（选取1到若干个原因解释）：

1. 现有技术路线的比较，onnx，framework-trt，api等。考虑选择某一个路线的原因，例如，时间原因，可以复用已有公开成果，etc.
2. 创新性考虑，例如，重点放在量化，所以模型导入不是工作重点，etc.
3. 性能考虑，例如，进行了profiling，发现问题主要是在小batchsize情况性能不好，所以希望通过TensorRT改善小batch inference的性能等。
4. 可扩展性考虑，虽然参加比赛，但是考虑到未来工作的可扩展性等。
5. 实用性考虑，例如考虑到未来上线环境，c++环境更友好等。

）

采用tensorRT API来实现推理，由于SOLOv2网络结构中包含一些无法解析的层，需要使用plugin，所以不适用onnx解析网络。

该分割网络中使用动态学习mask head和Matrix NMS，这在使用tensorRT部署时是关键结构，也是我们的创新所在。

为了部署推理更快，未来将对网络进行量化处理。

### 2.3 开发工作和遇到的困难

（重点描述做了哪些工作，工作时间的安排，例如第一周做了什么，第二周做了什么，第三周做了什么。遇到了哪些困难，如何解决的（可以是workaround，或者是feature的扩展/plugin的编写等。可以用流程图，示意图等描述。）

第一周：骨干网络的搭建；困难：group normalization层的实现；解决：在normalization层的tensorRT源码上修改成group normalization层

第二周：head的实现；困难：动态mask head的实现

第三周：后处理；困难：Matrix NMS的实现以及mask/point\_extract plugin

## 三．性能结果

### 3.1 测试环境

（详细描述原始模型的运行环境，加速后模型的测试环境，包括软硬件环境）

运行环境：

Python3.7

PyTorch1.4.0

mmcv0.2.16

mmdetection1.0.0

测试环境：

Python3.7

TensorRT 7.2.2

### 3.2 正确性验证

（描述验证方法，可以是单层1对1比对，误差范围（相对误差）是多少；也可以是最终的预测效果的数据，如果是以最终预测效果为准，请说明测试数据集规模，和最终效果计算的方法，例如top5的accuracy等。如果涉及多个精度（并且您想强调该工作支持多种精度），请分开说明。）

### 3.3 性能测试

提供完整性能对比表格，并附上运行时“nvidia-smi“输出的gpu频率和memory 频率，确保测试性能时没有别的数据在运行，可以提供运行时GPU memory使用情况。

可以提供end-to-end的性能，也可以只提供模型预测部分的性能。例如，既可以是从图片输入，预处理，到最后结果统计的时间；也可以是raw data（例如RGB normalized数据）输入，到最后出inference结果的时间。

需要提供完整的latency vs throughput 表格和图形，可以按照实际情况取batchsize，计算方法如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Batchsize | Latency, ms | Throughput  (1000/latency\*batchsize) | Latency Speedup  (TRT latency / original latency) | Throughput speedup (TRT throughput / original thoughput) |
| PyTorch | 1 |  |  | 1x |  |
| 8 |  |  |  |  |
| 16 |  |  |  |  |
| 32 |  |  |  |  |
| TensorRT | 1 |  |  |  |  |
| 8 |  |  |  |  |
| 16 |  |  |  |  |
| 32 |  |  |  |  |

绘制Latency vs. throughput 图形，横坐标latency (ms)，纵坐标throughput。指出你认为最应该选择的batchsize并说明原因（可能是实际应用对latency有要求，可能是这个情况性能最好，等等）

## 四．其他结果

（例如开发了通用工具，提交了trt bug等，或者详细的profiling过程等。你认为除了性能以外需要特别提及的亮点）

## 五．未来计划

（如果后续进一步工作，或者有更长时间，还有什么进一步优化的计划）

SOLOv2的量化部署实现

## 六．代码位置

（具体提供开源代码的地址，branch。注意，比赛结果提交的时候需要代码开源，并且创建专门的tag：”TRT2021”，后续你可以继续对你的代码进行优化，我们的评奖按照这个tag来。添加Readme文件， 说明编译步骤和运行环境， 比如使用了哪个docker image。性能测试/evaluation的代码也需要放在repo里面。）

代码连接：https://github.com/SunQiChang/deeplearning-tensorRT

## 七．TensorRT改进建议

（可以列举一些目前还不支持的特性或是能够使得工具更具易用性和通用性的技术反馈。）

## 参考文献

（如果有的话）

[1] Wang, Xinlong, et al. "Solo: Segmenting objects by locations." *European Conference on Computer Vision*. Springer, Cham, 2020.

[2] Wang, Xinlong, et al. "SOLOv2: Dynamic and fast instance segmentation." Advances in Neural Information Processing Systems (2020).

# 附件：评分标准，和对应报告的部分（不用包含到报告之中）

1. ⽂档描述清晰。20分 => 对应整个文档
2. 代码整⻬，逻辑清晰。30分 => 对应最后公开的代码，参见 “六.代码位置”部分
3. 模型顺利运⾏，有加速效果。30分 => 参见“三.性能结果”部分
4. 如果模型开发过程中发现TensorRT 的bug，提交bug，得到确认。5分每个bug。 => 在 “2.3 开发工作和遇到的困难” ，“四. 其他结果”和“七. TensorRT 改进建议” 中体现。
5. 开发了相应 Plugin 或 CUDA 代码。20分 => 在代码和“2.2 技术路线选择” 和“2.3 开发工作和遇到的困难”中描述。
6. 进⾏了 Profiling，对性能优化提出进⼀步的意⻅。20分 => 在“2.2 技术路线选择”，“2.3 开发工作和遇到的困难”和“四. 其他结果”中体现。
7. 进⾏了 INT8 量化的⼯作。20分 => 代码，“三.性能结果”， “2.2 技术路线选择” 和“2.3 开发工作和遇到的困难”中描述。