Tina V853 NPU Yolact 模型部署实战

版本号: 1.0

发布日期: 2021.07.21

版本历史

版本号	日期	制/修订人	内容描述
1.0	2021.07.21	PDC	NPU Yolact 模型部署实战

目　录

插　图

1 前言

1.1 读者对象

本文档（本指南）主要适用于以下人员：

- 技术支持工程师
- 软件开发工程师
- AI 应用案客户

1.2 约定

1.2.1 符号约定

本文中可能出现的符号如下：

警告

警告

技巧

1. 技巧
2. 小常识

说明

说明

2 正文

2.1 NPU 开发简介

- 支持 int8/uint8/int16 量化精度，运算性能可达 1TOPS.
- 相较于 GPU 作为 AI 运算单元的大型芯片方案，功耗不到 GPU 所需要的 1%.
- 可直接导入 Caffe, TensorFlow, Onnx, TFLite，Keras, Darknet, pyTorch 等模型格式.
- 提供 AI 开发工具：支持模型快速转换、支持开发板端侧转换 API、支持 TensorFlow, TF Lite, Caffe, ONNX, Darknet, pyTorch 等模型.
- 提供 AI 应用开发接口：提供 NPU 跨平台 API.

2.2 开发流程

NPU 开发完整的流程如下图所示：

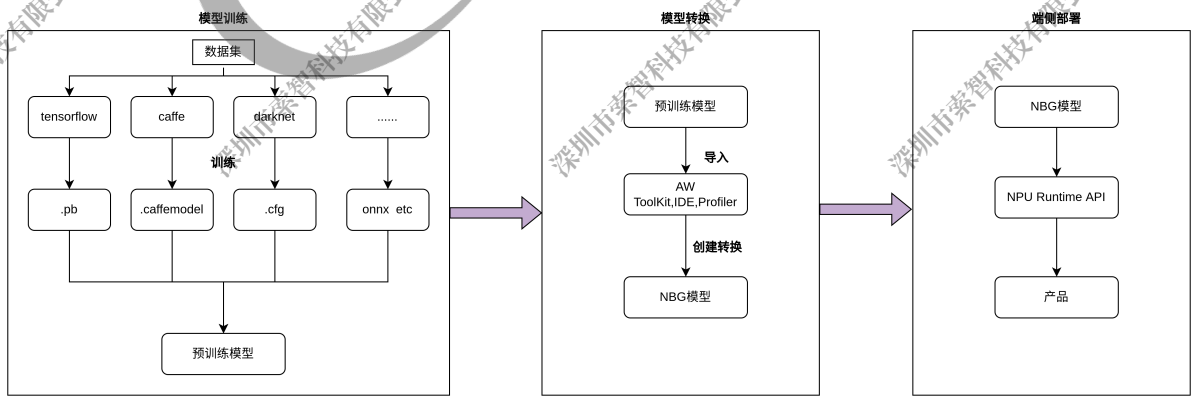

图 2-1：npu_1.png

2.3 获取 YOLACT 原始模型

YOLACT 模型获取方式有很多种，可以基于项目 https://github.com/dbolya/yolact 产生 onnx 格式的 yolact 模型，本文档假设你已经产生了 yolact.onnx 模型

2.4 模型部署工作目录结构

yolact-sim.onnx 有 120M，还是蛮大的.

图 2-2: npu_workspace

2.5 导入模型

```
pegasus import onnx --model yolact-sim.onnx --output-model yolact-sim.json --output-data yolact-sim
.data
```

导入模型的目的是将开放模型转换为符合 VIP 模型网络描述文件 (.json) 和权重文件 (.data)

图 2-3: npu_import

2.6 创建 YML 文件

YML 文件对网络的输入和输出进行描述，比如输入图像的形状，归一化系数 (均值，零点)，图像格式，输出 tensor 的输出格式，后处理方式等等，命令如下：

```
pegasus generate inputmeta --model yolact-sim.json --input-meta-output yolact-sim-inputemeta.yml

pegasus generate postprocess-file --model yolact-sim.json --postprocess-file-output yolact-sim-
postprocess-file.yml
```

图 2-4: npu_yml

修改 input meta 文件中的的 scale 参数为 0.0039(1/256)，和 yolov3 的改法完全一致。

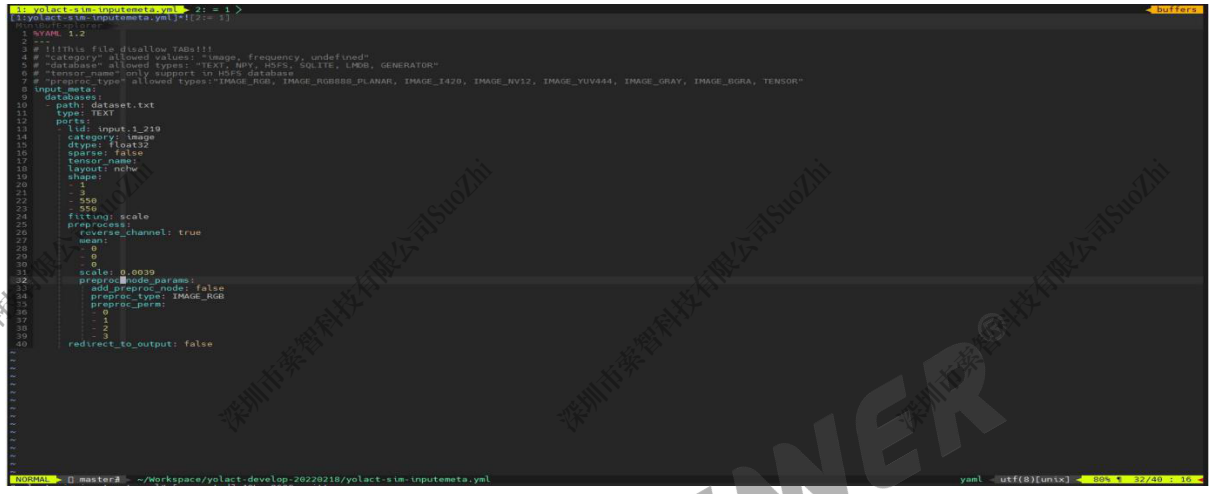

图 2-5: npu_scale

2.7 量化

量化命令：

```
pegasus quantize --model yolact-sim.json  --model-data yolact-sim.data --batch-size 1 --device CPU
--with-input-meta yolact-sim-inputemeta.yml --rebuild --model-quantize yolact-sim.quantilize --
quantizer asymmetric_affine --qtype uint8
```

命令执行后，创建了量化表文件

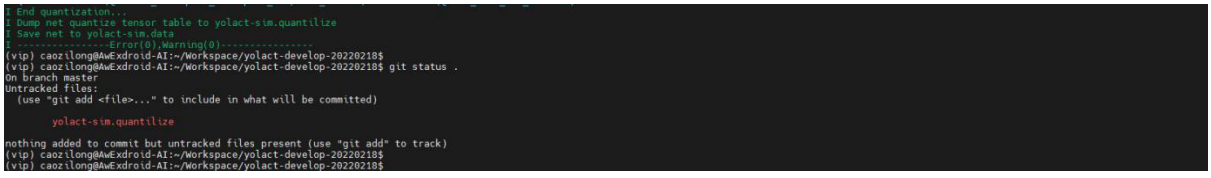

图 2-6: npu_quantilize

2.8 预推理

```
pegasus inference --model yolact-sim.json --model-data yolact-sim.data --batch-size 1 --dtype
 quantized --model-quantize yolact-sim.quantilize --device CPU --with-input-meta yolact-sim-
inputemeta.yml --postprocess-file yolact-sim-postprocess-file.yml
```

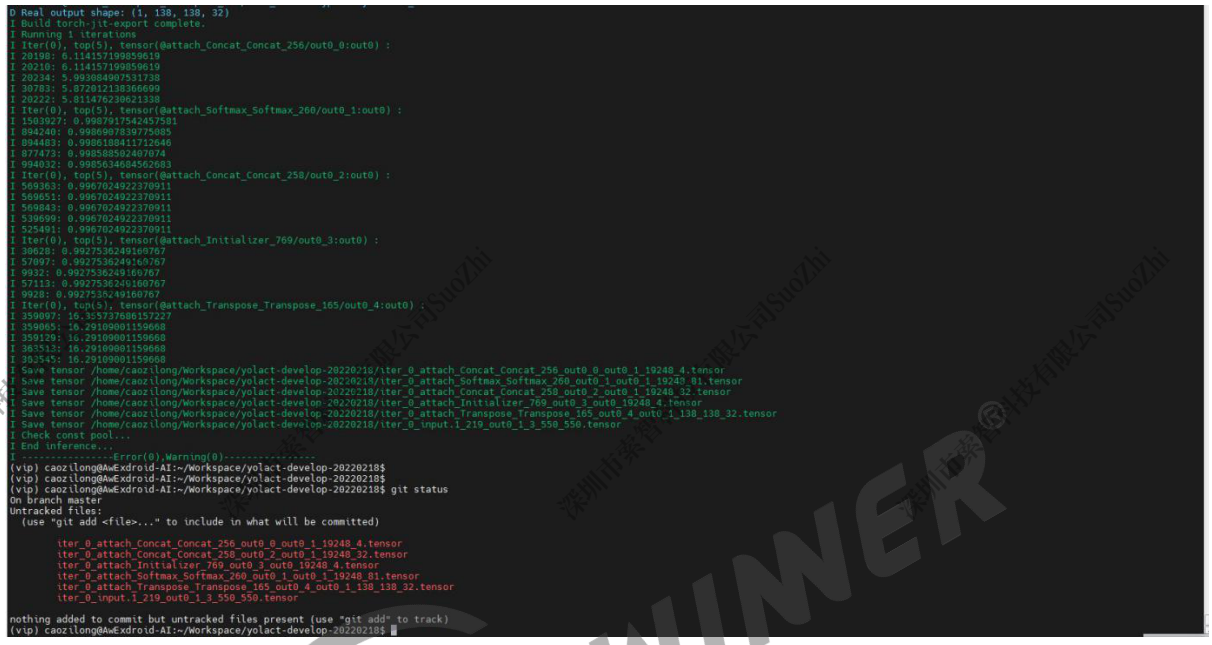

图 2-7: npu_inf

2.9 导出代码和 NBG 文件

```
pegasus export ovxlib --model yolact-sim.json --model-data yolact-sim.data --dtype quantized --
model-quantize yolact-sim.quantilize --batch-size 1 --save-fused-graph --target-ide-project '
linux64' --with-input-meta yolact-sim-inputemeta.yml --postprocess-file yolact-sim-postprocess-file
.yml --output-path ovxlib/yolact/yolact --pack-nbg-unify  --optimize "VIP9000PICO_PID0XEE" --viv-
sdk ${VIV_SDK}
```

```
pegasus export ovxlib --model yolact-sim.json --model-data yolact-sim.data --dtype quantized --
model-quantize yolact-sim.quantilize --batch-size 1 --save-fused-graph --target-ide-project '
linux64' --with-input-meta yolact-sim-inputemeta.yml --postprocess-file yolact-sim-postprocess-file
.yml --output-path ovxlib/yolact/yolact --pack-nbg-viplite --optimize "VIP9000PICO_PID0XEE" --viv-
sdk ${VIV_SDK}
```

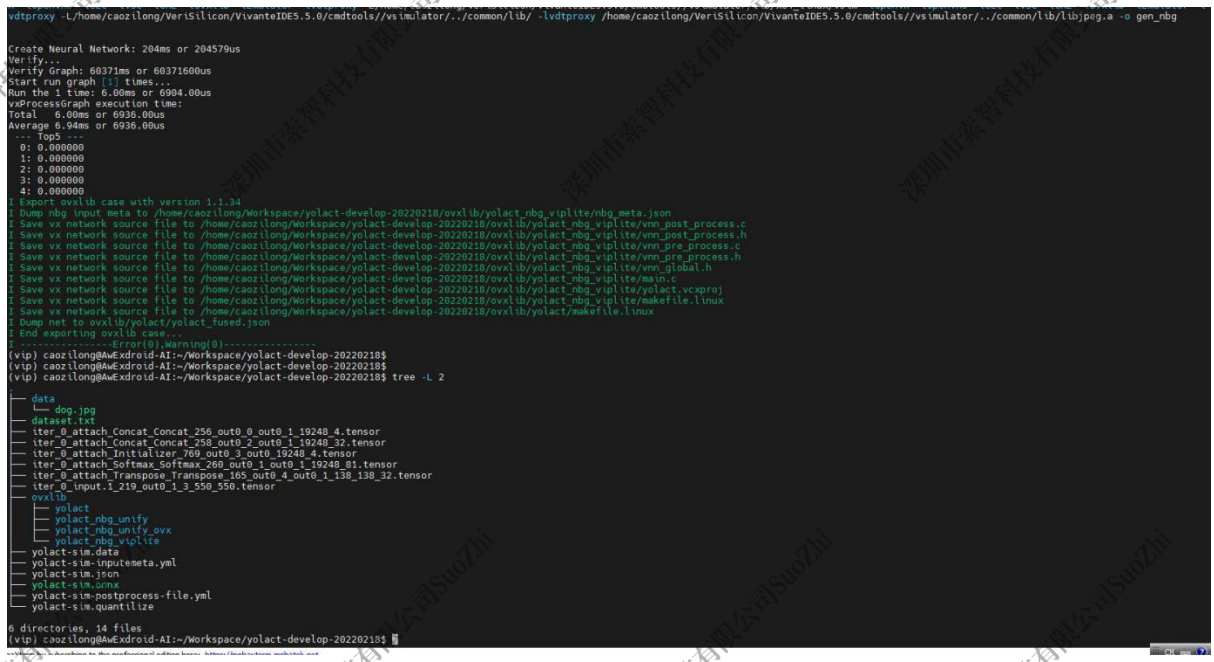

图 2-8: npu_export

生成的 NBG 文件, 可以部署到端侧:

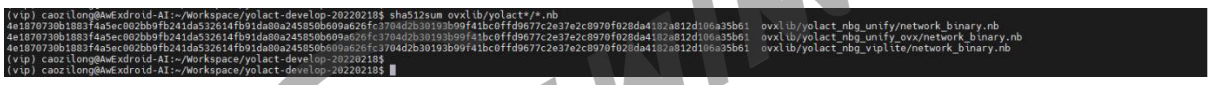

图 2-9: npu_nbg

2.10 模型算力测量

```
pegasus measure --model yolact-sim.json
```

图 2-10: npu_measure

对比 yolov3 的 32.99G MACC 算力和 yolov3-tiny 的 2.79G macc 的算力需求，YOLACT 明显对算力的需求大很多. 题外话,1TOPS=1000GOPS, 对于 yolov3-tiny 来说，我们以 3G 的算力需求来计算，就是 0.003T，在 500M 频率下，NPU 的理论算力是 0.5T, 所以，YOLOV3 的帧率为：0.5T/0.003T=166 帧。这个值和仿真值是比较接近的.

2.11 后处理验证

yolact 的后处理 C 代码在如下位置：

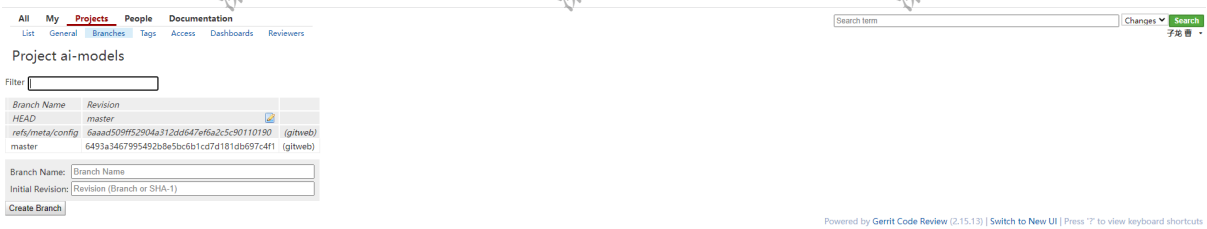

图 2-11: npu_gerit

编译后处理模型：

图 2-12: npu_post

out 目录生成了 yolact 的后处理程序：

图 2-13: npu_postprocess

yolact 模型有五个输出层,inference 阶段生成了 5 个 output tensor.

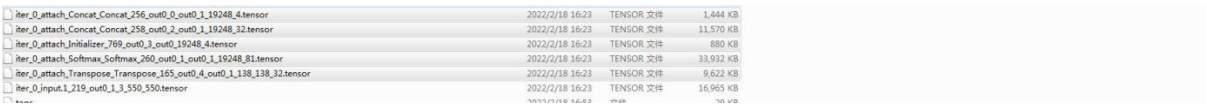

图 2-14: npu_tensor

tensor之间的对应关系,0是location, 1是confidence, 2是mask，3暂时无用，4是maskmaps.

output0_4_19248_1.dat <---->iter_0_attach_Concat_Concat_256_out0_0_out0_1_19248_4.tensor

output1_81_19248_1.dat<---->iter_0_attach_Softmax_Softmax_260_out0_1_out0_1_19248_81.tensor

```
output2_32_19248_1.dat<---->iter_0_attach_Concat_Concat_258_out0_2_out0_1_19248_32.tensor

output3_4_19248.dat  <---->iter_0_attach_Initializer_769_out0_3_out0_19248_4.tensor

output4_32_138_138_1.dat<---->iter_0_attach_Transpose_Transpose_165_out0_4_out0_1_138_138_32.tensor
```

执行后处理

```
(vip) caozilong@AwExdroid-AI:~/ai-models/yolact/postprocess$ ./out/postprocess dog.jpg
time : 0.035 Sec
2 = 0.96443 at 99.48 111.53 478.22 x 329.83
17 = 0.76052 at 128.47 212.00 196.70 x 339.42
17 = 0.70652 at 130.96 150.66 341.01 x 358.92
8 = 0.52484 at 472.83 76.51 207.34 x 92.95
3 = 0.52384 at 472.83 75.79 207.34 x 94.07
(vip) caozilong@AwExdroid-AI:~/ai-models/yolact/postprocess$
```

图 2-15: npu_exepost

后处理结果：

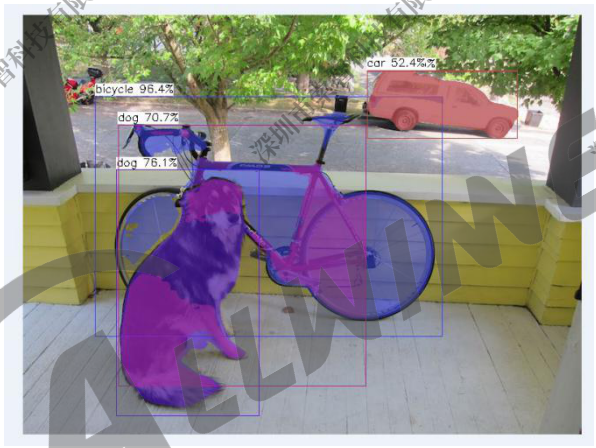

图 2-16: npu_yolact

图中识别除了两只 dog，有两个狗的信息，其他都能很好的输出结果，dog 70.7% 不应该出现的，大概率是量化导致输出结果精度精度降低。

修改归一化参数后重新部署，得到的结果如下，可以看到，之前的问题消失：

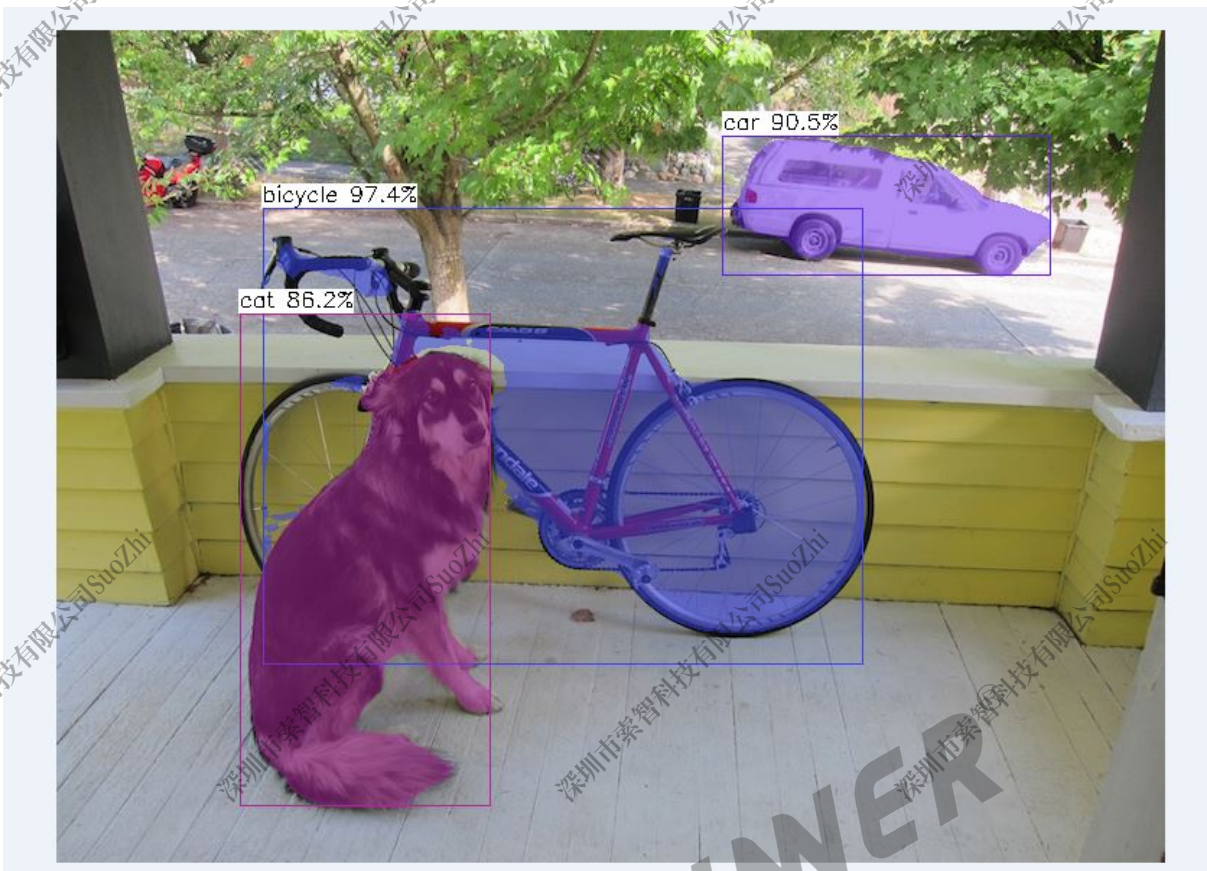

图 2-17: npu_yolact_ok

至此，yoloact 模型导入完成.

2.12 结束

著作权声明

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

商标声明


、 

、 
（不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。