软件开发手册

版本 1.0　　时间：2024.04.06

一. NPU 介绍

NPU 是一款面向实时、高性能、多应用场景要求的神经网络处理器，可满足智慧感知、智能决策、智能控制等场景智能处理和算法加速需求。NPU 采用自主设计的架构，主要包括 GEMM 模块、ALU 模块、池化模块、激活函数模块等，系统支持单精度浮点 fp32 和 Int8 精度计算，500M 时钟频率下算力可以达到 2TOPS，集成了两路 AXI 总线接口，目前支持的单个算子主要包括通用卷积、深度卷积、全连接、最大值池化、均值池化、常用的激活函数和 ALU 操作等，支持常见的二十余种卷积神经网络算法。

二. 工程目录结构介绍

src/

```
├── config.h
├── hw
├── layer
├── lscript.ld
├── main.cpp
├── ncnn
├── net
├── README.txt
├── testLayer
└── utils
```

1. **config.h:**存放了整个系统工程全部的宏定义，下面介绍一些重要的宏定义；

//关于计算平台选择的宏定义

```
#define FORWARD_ON_NPU_CONV      //卷积算子在 NPU 上运算
#define FORWARD_ON_NPU_POOL      //池化算子在 NPU 上运算
#define FORWARD_ON_NPU_ALU       //ALU 算子在 NPU 上运算
#define PRINT_ALL                //打印提示信息
```

//单个算子仿真测试

```
#define TEST_LAYER_CONV
#define TEST_LAYER_POOL
```

```
#define TEST_LAYER_ELTWISE
#define TEST_LAYER_BINARYOP
#define TEST_LAYER_ABSVAL
#define TEST_LAYER_BIAS
#define TEST_LAYER_DROPOUT
#define TEST_LAYER_SCALE
#define TEST_LAYER_THRESHOLD
#define TEST_LAYER_RELU
#define TEST_LAYER_TANH
#define TEST_LAYER_CLIP
#define TEST_LAYER_SIGMOID
#define TEST_LAYER_SWISH
#define TEST_LAYER_ELU
#define TEST_LAYER_SELU
#define TEST_LAYER_HARDSIGMOID
#define TEST_LAYER_HARDSWISH
#define TEST_LAYER_INNERPROD
```

//网络推理宏定义

```
#define TEST_YOLOV3
#define TEST_YOLOV3_TINY
#define TEST_YOLOV4_TINY
#define TEST_YOLOV5S
#define TEST_YOLOV6N
#define TEST_YOLOV7
#define TEST_YOLOV7_TINY
```

需要注意单个算子测试和网络推理不能同时进行

2. **hw**：存放了与硬件相关的函数文件，包括系统的初始化函数，以及底层算子推理的寄存器配置函数；
3. **layer**：存放了 ncnn 的 layer 定义代码；
4. **main.cpp**：存放了系统工程的主函数；
5. **net**：存放了网络推理函数；
6. **testLayer**：存放了单个算子的仿真测试文件，包括卷积、池化、激活函数、ALU 等，每种算子都包含了大量仿真 case；
7. **utils**：存放了系统工程中调用的一些常用函数；

三．典型使用方式——编程示例

对于神经网络算法的部署，软件层的工作主要分为两个部分：网络模型配置

文件的生成和网络推理。

1. 网络模型配置文件的生成

NPU 目前支持来自不同训练框架（如 PyTorch、Caffe、ONNX 等）的模型文件转换为 NPU 专用的二进制文件格式。典型的过程包括：框架转换、模型优化以及模型量化。

（1） 框架转换

框架转换旨在将来自不同训练框架下的预训练权重统一转换至 NPU 专用的框架。对于 PyTorch 训练框架下的训练权重，直接用 torch.onnx.export 就能输出.onnx 文件，官方同时开源了对于 ONNX 转为 NCNN 框架的指导教程，用户可以参考教程进行转换。

https://github.com/Tencent/ncnn

（2） 模型优化

模型优化过程完成对计算图的优化，主要包括卷积类算子的融合、层的替换和优化等，相关的实现已经集成到 Simlution 工程中。

首先切换到工程的 modles 路径下，

```
$ cd models/
```

修改该路径下 Makefile 文件中的 NET_NAME 以及与优化、量化相关的参数，其中 NET_NAME 代表当前需要转换的网络名称，并在 modles 目录下建立相同名称的文件夹，将上一步框架转换完成后的文件复制到该文件夹中，然后运行指令：

```
$ make tools
$ make opt
```

等待优化完成后，会在对应文件夹下生成 NET_NAME_opt.param 和 NET_NAME_opt.bin 文件。

（3） 模型量化

网络模型的量化算法采用 NCNN 的量化方式，在量化之前需要首先生成量化校准表，需要运行如下指令：

```
$ make table
```

生成标准表之后，进行量化：

```
$ make quant
```

等待量化完成后，会在对应文件夹下生成 NET_NAME_int8.param 和 NET_NAME_int8.bin 文件，至此网络模型配置文件已经成功生成。

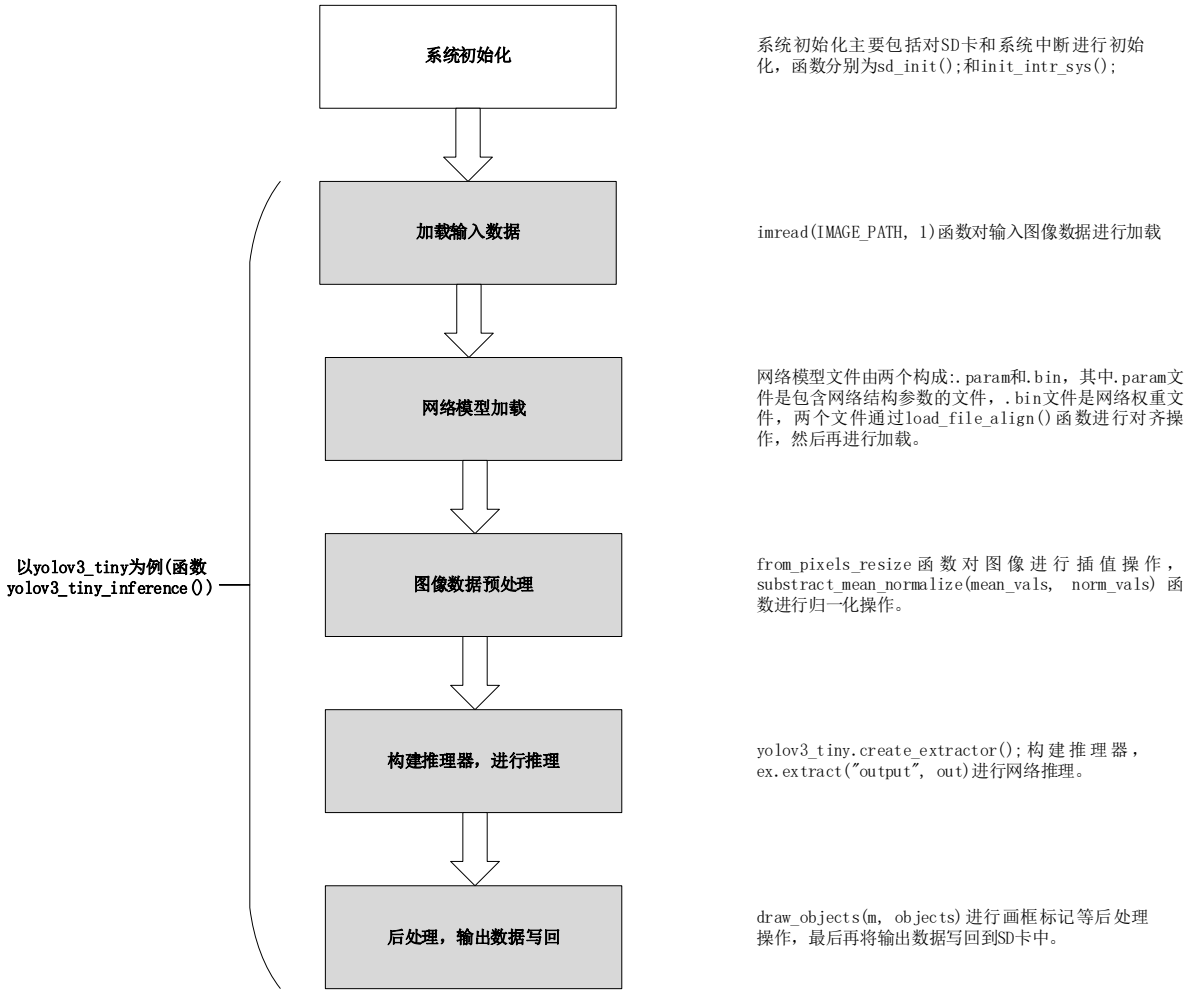
当然在 Simlution 工程文件夹中也已经附带了部分已经量化完成的算法文件，可供用户直接使用。其他算法的参数文件可以从网盘链接获取：

链接: https://pan.baidu.com/s/1BMMpRjHi4iafZVosZ3sc3A

提取码: 4056

2. 网络推理

目前 NPU 已经成功适配了包括 yolov3、yolov4、yolov5、resnet18、resnet50、squeezenet、mobilenet_ssd 在内的二十余种网络，大部分网络的运行流程基本相似，下图展示了 yolov3_tiny_inference() API 的运行流程。



其中在对应模型的 API 函数中可以通过修改下面的部分去配置不同的网络权重参数文件：

```
if (yolov3_tiny.load_param_mem(load_file_align("yolov3_tiny_int8.param")))
        exit(-1);
if (yolov3_tiny.load_model_mem(load_file_align("yolov3_tiny_int8.bin")))
        exit(-1);
```

下面的表格列出了目前所支持的 API 函数列表，用户只需要调用相应的 API 函数即可。

序号	网络名称	API
1	yolov3	yolov3_inference()
2	yolov3_tiny	yolov3_tiny_inference()
3	yolov4_tiny	yolov4_tiny_inference()
4	yolov5s	yolov5s_inference()
5	yolov6n	yolov6n_inference()
6	yolov7	yolov7_inference()
7	yolov7_tiny	yolov7_tiny_inference()
8	mobilenet_ssd	mobilenet_ssd_inference()
9	mobilenet_yolo	mobilenet_yolo_inference()
10	mobilenetv2_yolo	mobilenetv2_yolo_inference()
11	squeezenet	squeezenet_inference()
12	shufflenetv2	shufflenetv2_inference()
13	resnet18	resnet18_inference()
14	resnet50	resnet50_inference()
15	resnet101	resnet101_inference()
16	googlenet	googlenet_inference()
17	retinaface	retinaface_inference()