**软件开发手册**

版本1.0 时间：2024.04.06

# 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。

# 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**

1. **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；
2. **layer：**存放了ncnn的layer定义代码；
3. **main.cpp：**存放了系统工程的主函数；
4. **net：**存放了网络推理函数；
5. **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；
6. **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。

## 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。

### 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

### 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。

1. 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

## 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() |