FPGA 深度学习加速器工具链说明文档

一、工具链职能

为实现对深度学习模型的加速，FPGA 需要将网络模型中的每一层操作拆分出来做计算。本工具链所做的就是为 FPGA 深度学习加速器提供每一层的数据信息、指令控制信息以及对比结果。

本文档仅介绍工具链如何使用，代码的注释中有更详细的细节。

二、工具链总体架构

1. 目录结构

工具链由四个软件包和三个 python 文件组成：

main.py:工具链主函数，负责将网络模型按照网络结构进行拆分

transit.Py：分发器，根据每一层网络所做的操作来调用不同的算子

shared_variable.Py:定义整个工具链的共享变量

conv_operator 软件包：存放 conv 操作算子

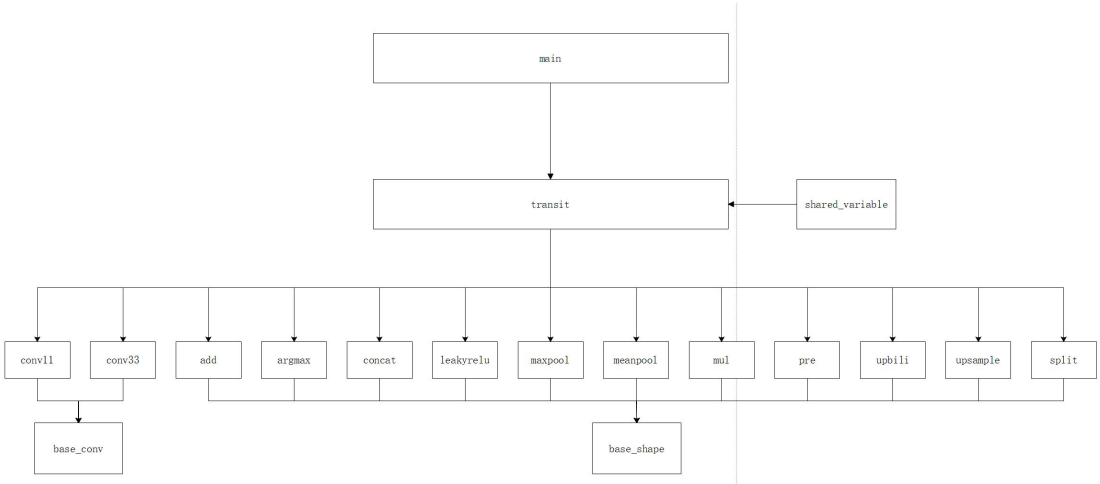
shape_operator 软件包：存放 shape 操作算子

lib 软件包:存放算子操作过程中所需要的方法

utils 软件包:存放数据信息、指令控制信息以及对比结果生成后所需要的一些便捷工具

2. 工具链总体设计

工具链的设计思路如下图所示：



main 函数根据网络层数会调用多个 transit 分发器，所有的 transit 都共

用一个共享变量对象（shared_variable）。同时 transit 会根据操作类型选择不同的算子，其中 conv11 算子和 conv33 算子继承了 base_conv 父类，其余算子继承了 base_shape 父类。

三、主函数编写规范

主函数编写由两部分组成。首先需要将网络模型的每一层拆分出来，之后将拆分出来的网络输入输出、提取出来的量化参数、本层操作传入 transit 分发器。

1. 拆分网络模型

1.1 导入量化好的模型

```
model = torch.jit.load(shared.model_path)
model.eval()
```

shared.model_path 是量化后模型的文件路径，使用 torch 导入模型

1.2 导入需要处理的输入图片

```
img = picture_load(shared.img_path)
```

shared.img_path 是输入图片路径，使用 picture_load 导入输入图片并做一次预处理。预处理一般是将图片转为灰度图并做一次归一化。

归一化的公式为[(图片数组/255)-数据集图片均值]/数据集图片方差。

1.3 按照网络结构拆分网络

```
# ------------------------ContextPath------------------------------
ContextPath = model.cp
# ------------------------cp.resnet18------------------------------
cp_Resnet18 = ContextPath.resnet
cp_Resnet18_conv1 = cp_Resnet18.conv1
cp_Resnet18_bn1 = cp_Resnet18.bn1
cp_Resnet18_relu = cp_Resnet18.relu

quant_feature_f = model.quant(img)

cp_Resnet18_conv1_feature = cp_Resnet18_conv1(quant_feature_f)
cp_Resnet18_bn1_feature = cp_Resnet18_bn1(cp_Resnet18_conv1_feature)
cp_Resnet18_relu_feature = cp_Resnet18_relu(cp_Resnet18_bn1_feature)
```

以 Bisenet 网络为例，该网络第一层为 3*3 卷积，并且做了特征融合，将 conv、bn 融合为一层。

先从模型中拆出第一层的三个操作（conv1、bn1、relu），再将输入图片经

过量化，最后将量化图片经过 conv、bn1、relu 处理，得到输出特征图 cp_Resnet18_relu_feature。这就拆分出了网络结构的第一层，其他层也类似此操作。

2. 规范调用 transit 分发器

2.1 conv 操作调用 transit 分发器

若为正常 conv 操作 ：

```
# layer ↓ 1
Transit(para1='quant', para2='cp.resnet.conv1', feature=[quant_feature_f, cp_Resnet18_relu_feature],
        option=['Conv33', 2, 1, 1])
```

para1=’输入层的 npy 文件前缀名’,

para2=’本层的 npy 文件前缀名’,

feature=[输入特征图, 输出特征图],

option=[’卷积操作名’,stride,padding,是否使用激活函数]

其中 para 参数中的 npy 文件是量化时提取出来的 weight、scale、zp、bias 参数，conv 操作都有这些参数。

若为分块 conv 操作：

```
block(para1='quant', para2='cp.resnet.conv1', feature=[quant_feature_f, cp_Resnet18_relu_feature],
      option=['Conv33', 2, 1, 1, 2])
```

para1=’输入层的 npy 文件前缀名’,

para2=’本层的 npy 文件前缀名’,

feature=[输入特征图, 输出特征图],

option=[’卷积操作名’,stride,padding,是否使用激活函数,分块数量]

分块 conv 操作只比正常 conv 操作多需要一个分块数量参数，并且根据分块数量生成 2*分块数量-1 层指令、权重、对比结果(满二叉树)。

该函数本质上还是调用 Transit 分发器。

2.2 Shape 操作调用 Transit 分发器

```
# layer ↓ 5
cp_Resnet18_layer1_b0_out_feature = cp_Resnet18_layer1_b0_qf.add(cp_Resnet18_layer1_b0_bn2_feature,
                                                                 cp_Resnet18_maxpool_feature)
Transit(para1='cp.resnet.layer1.0.conv2', para2='cp.resnet.layer1.0.qf', para3='cp.resnet.conv1',
        feature=[cp_Resnet18_layer1_b0_bn2_feature, cp_Resnet18_layer1_b0_out_feature,
                 cp_Resnet18_maxpool_feature],
        option=['Add'])
```

para1='左输入层的 npy 文件前缀名',

para2='本层的 npy 文件前缀名',

para3='右输入层的 npy 文件前缀名',

feature=[左输入特征图,输出特征图,右输入特征图],

option=['shape 操作名']

Shape 操作调用 Transit 分发器的特殊形式：

```
# layer ↓ 6
cp_Resnet18_layer1_b0_relu_feature = cp_Resnet18_layer1_b0_relu2(cp_Resnet18_layer1_b0_out_feature)
Transit(para1='cp.resnet.layer1.0.qf', para2='cp.resnet.layer1.0.qf',
        feature=[cp_Resnet18_layer1_b0_out_feature, cp_Resnet18_layer1_b0_relu_feature], option=['LeakyRelu'])
```

由于 leakyRelu 操作不需要 npy 文件提供数据,所以本层的 para2 用 para1 代替。

由于 leakyRelu 操作只需要一个输入，因此 para3 可省略，feature[3]可省略。

四、共享变量

1. 生成模式相关变量

```
# 生成模式有两种，
# [0,层数]：测单层，用于单层仿真,生成单层指令、权重、中间结果
# [1,层数]：联测,用于侧板子,生成0-本层的连续指令、权重、中间结果
generate_mode = [1, 65]
# 这三个变量需要配合生成模式使用
gen_ins = True  # 是否需要指令
gen_weight = False  # 是否需要生成权重
gen_result = False  # 是否需要生成中间结果,联测生成一次后建议关掉,太占用时间
```

以图中这种配置方式为例：

generate_mode:表示生成到 65 层，并且每一层数据都囊括了之前的数据（1 代表联测）。gen_ins 为 true，gen_weight 为 false，gen_result 为 false 表示本次只生成指令。

若 generate_mode=[0,64]表示生成到 64 层，并且每一层数据不包括之前的数据（0 代表测单层）。

2. 配置相关变量

```
picture_address = 0  # 输入图片地址,一般记0
write_address = 409600  # 特征图起始地址,一般接在输入图片之后
weight_address = 64103424  # 权重起始地址,一般接在所有特征图之后

parallel = 16  # 指示生成数据采用16进16出还是8进8出
img_path = '../bisenet/img/234.jpg'  # 输入图片路径
model_path = '../bisenet/model/5class_quantization_post.pth'  # 量化模型路径
mean = 0.0558  # 整个数据集图片的均值,无均值填0
std = 0.1208  # 整个数据集图片的方差,无方差填1
file_path = "../sim_data/"  # 生成指令、权重、对比结果路径
para_path = "../para_68/"  # 提取npy文件路径
start_op = 0  # 用于首次卷积时指示输入是否要做特殊处理,不推荐使用,预处理层可以做得更好
layer_count = 0  # 用于维护网络层数,0层一般为为预处理层,也负责生成输入文件
# shape状态字典,用于维护shape操作的状态码
shape_control = {
    'MaxPool': 1,
    'Split': 2,
    'UpSample': 3,
    'Concat': 4,
    'Add': 5,
    'LeakyRelu': 6,
    'MeanPool': 7,
    'Mul': 8,
    'ArgMax': 9,
    'UpBili': 10,
    'Pre': 11,
    'YoloSig': 12
}
address_table = {}  # 读地址字典,用于维护特征图读地址变换
```

picture_address:记录了输入图片的起始地址

write_address:记录了特征图(每一层的计算结果)的起始地址

weight_address:记录了权重数据的起始地址

paraller:指示了 FPGA 的通道并行数，一般为 8 或 16

shape_control:用于维护 shape 操作的状态码，由于每个项目用到的 shape 操作各不相同，因此每个项目的 shape 状态码都不一样。将项目中使用到的 n 个 shape 操作从 1 到 n 排序，未使用到的 shape 操作排到 n 之后可以优化时序。

address_table:用于记录工具链中的读地址，从而实现地址变换，此字典是自动更新的，不需要填任何信息。