ESWIN

ENNP 用户手册

EIC7x 系列 AI SoC

Engineering Draft / Rev1.2

2025-01-06

ESWIN Confidential

声明

版权所有©2024，北京奕斯伟计算技术有限公司及其关联公司（以下简称“ESWIN”）。保留所有权利。

ESWIN 在此软件中保留所有知识产权和专有权利。未经 ESWIN 授权，不得发布、复制、分发、转载、修改、改编、翻译或制作此软件的衍生作品，无论是全部还是部分。

对于本文档的任何修改，ESWIN 无需承担通知义务且不代表 ESWIN 做出任何承诺。除非另有说明，本文档中的所有陈述、信息及建议均按“现状”提供，ESWIN 不对此做出任何形式的担保、保证与承诺，无论明示或默示。（本段落用于文档中）

“ESWIN”logo 和其他 ESWIN 标识，为北京奕斯伟计算技术有限公司及（或）其母公司所有的商标。

本文档提及的其他所有商标或商品名称，由其各自的所有权人所有。

产品安装中可能包含开源软件和/或自由软件的许可声明。

ESWIN Confidential

修订记录

文档版本	日期	说明
v0.3	2024/02/22	初始版本
v0.4	2024/03/25	修改 NPU Runtime 概要描述和调用图等。
v0.5	2024/07/17	修改章节顺序，修改 EsAAC 和 EsQuant 工具说明。
v0.6	2024/08/09	完善 docker 的使用说明。
v0.7	2024/08/28	增加从 GitHub 导出 ONNX 模型的示例。
v0.8	2024/09/10	完善 yolo v3 示例说明。
v0.9	2024/10/12	增加 NPU Runtime 实现动态 Batch size 和复合模型的说明。
v0.9.2	2024/10/22	修订了 NPU Runtime 实现动态 Batch size 和复合模型的说明。
v1.0	2024/11/18	增加用户量化流程使用自定义预处理函数的说明。
v1.1	2024/12/26	修订内容： 1、新增 EsAAC 混合模型生成方式说明； 2、增加用户自定义 dsp 算子开发示例； 3、完善了 dsp 支持算子列表。 影响范围： 本版本更新，在第 3.2 节增加了混合模型的编译说明；增加了第 8 章自定义 dsp 算子开发内容；在第 9 章附录添加了新支持的 dsp 算子。对其余章节无增删，不影响用户之前的使用方式。
v1.2	2025/01/06	修订内容： 1、新增 EsAAC 对 vit 模型的编译选项--enable-uosp 说明。 影响范围： 本版本更新，在第 3.1 节新增了编译选项说明，对其余章节无增删。用户在编译 vit 模型时，请添加编译选项--enable-uosp；其余场景下，用户延续之前使用方式即可。
v1.3	2025/01/21	新增 EsAAC 编译选项--lutdsp 的说明。

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

目录

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

ESWIN Confidential

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

表目录

ESWIN Confidential

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

图目录

ESWIN Confidential

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

1. 概述

1.1 ENNP 简介

ENNP（ESWIN Neural Network Processing）平台是奕斯伟媒体处理芯片智能计算异构加速平台。开发者可以基于 ENNP 提供的 API 接口调用 EIC7700 内部的硬件加速模块实现神经网络模型的推理、图像的变换处理及其他需要硬件加速的自定义功能。ENNP 平台能提供基于 HAE、GPU、DSP、NPU 等硬件加速模型的推理，实现目标识别、对象检测、图像分类等应用场景。

ENNP 包括离线开发工具套件和运行时软件框架，基于 ENNP 的开发分为离线开发与在线开发。

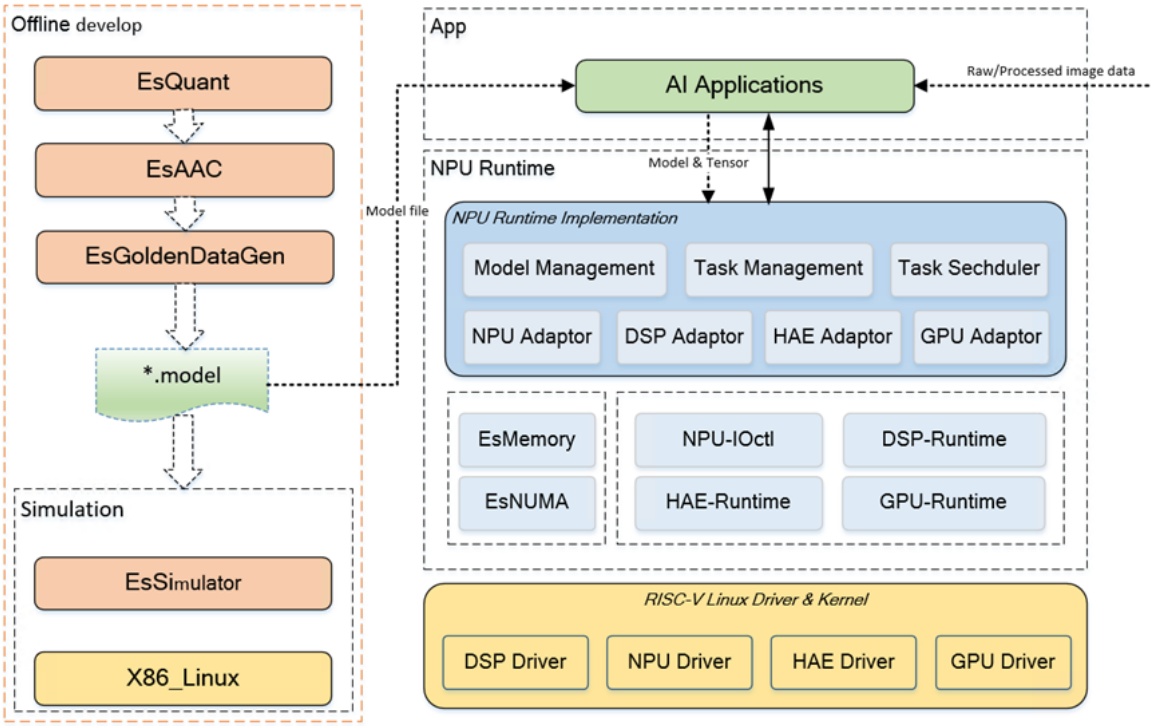

图 1-1 ENNP 软件栈框图

图 1-1 中的左边框图“Offline develop”中包含 EsQuant 量化工具，EsAAC 模型编译工具，EsGoldenDataGen 参考数据生成工具以及 EsSimulator 仿真验证工具。这些属于 ENNP 离线开发工具套件。利用上述工具套件，可以将人工智能框架（TensorFlow、 PyTorch、Caffe 与 ONNX 等）训练好的算法模型转换成统一的中间表达（IR）并生成离线模型，同时提供端到端的模型优化、离线模型产生及验证等功能。每个套件工具的详细信息可以参考对应的章节。

模型生成后，可以基于 ESSDK 编写运行时软件，调用 NPU Runtime 的接口利用 NPU 子系统硬件（NPU、DSP、HAE、GPU 等）进行推理基于芯片硬件加速实现图像分类，目标检测，图像分割，自然语言处理等功能。

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

1.2 运行时框架

NPU 运行时即下文提到的 NPU Runtime，是 ENNP 提供的一套加载 NN 模型的运行时系统。它可以直接加载基于 ENNP 工具量化编译的离线模型并进行推理完成目标识别、图像分类等功能。用户基于 NPU Runtime 开发智能分析方案，由编译器自动分析并优化执行过程，并生成优化好的离线模型，最大化复用 NPU，DSP，HAE、GPU 等硬件，提升硬件利用率并优化系统功耗。

NPU Runtime 是 ENNP 向用户应用程序提供模型推理能力的 API 接口，用户可以直接调用 NPU Runtime 丰富的 API 来实现实际应用的推理场景。目前 NPU Runtime 支持 ENNP 平台模型编译器 EsAAC 生成的各种网络模型（具体见：本手册第 3 章 EsAAC 使用指南）。NPU Runtime 支持 HAE 硬件前处理（Resize，Normalization，CVT），支持 DSP 丰富且性能优异的算子（算子列表详见：附录）运算，支持 ESWIN 的 NPU 硬件加速单元。

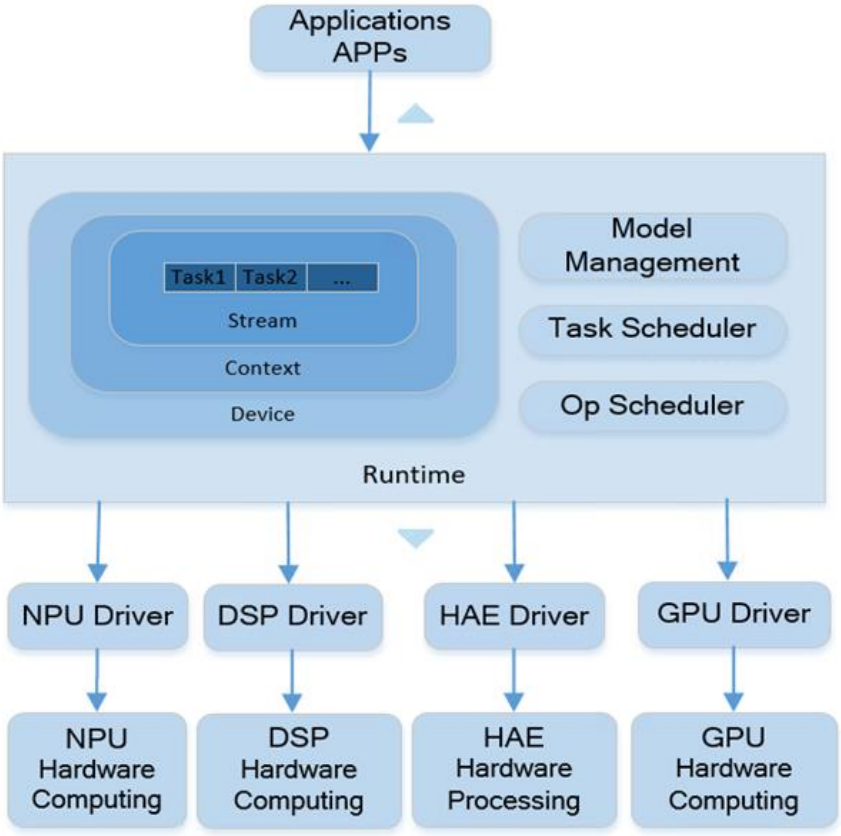

图 1-2 NPU Runtime 框图

- 模型推理接口：支持不同硬件加速单元上多个模型的推理，通过任务调度器协调多个任务的执行顺序。
  - Device：表示一个 NPU 推理硬件执行单元，不同 Device 上的 Task 是并行执行的。
  - Context：描述了 NPU 运行的一个配置上下文，方便管理对应的内存分配及硬件资源；同时它作为一个容器，管理了在对应 Context 上创建对象的生命周期，用于管理异步提交的任务。一个 Context 可以关联到一个进程的多个线程，不同 Context 的 Stream 是完全隔离

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

的，基于同一个 Context 上提交的任务都会按照先后顺序进行调度。在进程或线程中调用 ES_NPU_CreateContext 接口创建一个 Context。

- Stream：用于维护异步操作任务完成后等待用户取回的队列。在进程或线程中调用 ES_NPU_CreateStream 接口创建一个 Stream。Stream 隶属于 Context。
- Task：任务的执行载体，包含执行一个任务所需的必要信息。

- 推理加速硬件：可以使用 NPU、DSP、HAE、GPU 等硬件实现对应的加速算法。利用 ENNP 将 NN 模型量化编译后，可以直接加载到运行时，并由框架软件及固件自动且高效地调度到各个硬件单元执行。也可以通过 AcceleratorKit 单独调用对应的算子执行单个计算。

1.3 软件开发流程

基于 ENNP 的开发流程可以细分为两大部分：离线工具使用与在线开发验证。

1.3.1 离线工具使用

相关工具有：量化工具 EsQuant、模型编译工具 EsAAC、参考数据生成工具 EsGoldenDataGen、仿真验证工具 EsSimulator，其开发步骤如下：

- ENNP 工具环境配置：在 PC 端按照指南安装 ENNP 所需的工具套件。
- 模型获取：利用 PyTorch 框架搭建并训练神经网络，之后将模型导出为 ONNX 格式（.onnx 文件），以便进行后续处理。
- 量化参数获取：EsQuant 基于 ONNX 文件与量化配置文件，生成量化参数，其保存方式为 json 格式。
- 模型生成：EsAAC 工具基于 ONNX 文件和量化参数 json 文件，编译生成.model 文件
- 仿真验证：基于 ONNX 文件使用 EsGoldenDataGen 生成参考数据，并通过 EsSimulator 进行仿真验证，确保模型精度符合预期。

1.3.2 在线开发验证

- C 语言接口编程：基于 ESSDK 提供的 C 接口 API，编写适配的 C 语言代码，并修改 CMakeLists.txt 文件以适配编译流程。
- 执行验证：使用交叉编译器编译上述代码，生成可执行程序。将此程序拷贝至 EIC7700 开发板上执行，以验证程序的实际运行效果与模型性能。

通过上述步骤，开发者可以高效地进行模型的开发、优化、编译、验证，并最终实现在目标硬件设备上的部署与运行，从而完成从模型设计到实际应用的全流程开发。

1.4 开发环境

操作系统版本、SDK 版本介绍等详见第 6 章。

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

1.5 相关文档

ENNP 开发所依赖的相关文档，包括《ENNP 开发者手册》等。

2. EsQuant 使用指南

2.1 EsQuant 工具介绍

EsQuant 工具提供 python 接口供用户使用，使用 python 接口形式量化模型的优势是用户可以自定义预处理函数，同时预先提供了 imagenet 和 yolo 的预处理函数，方便分类模型和检测模型的量化工作。

EsQuant 可以方便地完成以下工作：

- 量化功能：支持将浮点数模型量化成定点化模型，支持对称量化，不同层支持混合精度量化；
- 量化精度分析：该功能将给出模型量化后每一层推理结果与浮点数推理结果的余弦相似度，包含累计误差分析和重置误差分析，根据累计误差分析可以便于分析量化误差是如何出现的，为提高量化模型的精度提供思路。

2.2 模型量化说明

2.2.1 量化配置文件说明

配置文件为 json 文件，以 yolov3 模型为例，格式如下：

ESWIN Confidential

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

```
eswin@debian:~# cat config.json
{
    "version": "1.3",
    "model": {
        "model_path": "/yolov3.onnx",
        "save_path": "/home/yolov3/",
        "images_list": "/home/yolov3/cal_lists.txt",
        "analysis_list": "/home/yolov3/file_list_1.txt"
    },
    "quant": {
        "quantized_method": "per_channel",
        "quantized_dtype": "int8",
        "requant_mode": "mean",
        "quantized_algorithm": "AT_EIC",
        "optimization_option": "auto",
        "bias_option": "absmax",
        "nodes_option1": [],
        "nodes_option2": [],
        "nodes_i8": [],
        "nodes_i16": ["conv1"],
        "mean": [
            0,0,0
        ],
        "std": [
            1,1,1
        ],
        "norm": true,
        "scale_path": "",
        "enable_analyse": true,
        "device": "cpu",
        "output_variables_dtype": {
            "326": "int16",
            "392": "int16",
            "458": "int16"
        }
    },
    "preprocess": {
        "input_format": "RGB",
        "keep_ratio":false,
        "resize_shape": [
          416,416
        ],
        "crop_shape": [
          416,
          416
        ]
    }
}
```

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

参数说明见表 2-1。

表 2-1 配置文件参数说明

配置	示例	说明
model_path (必配参数)	/path/resnet.onnx	需要量化的模型路径，目前支持的模型为 ONNX 模型，不配置将报错并中止程序。
save_path (必配参数)	/path/save/	中间数据（包括 table.json）以及量化精度分析结果的保存路径，其中量化精度分析结果为 txt 文件，不配置将报错并中止程序。
images_list (必配参数)	/path/image_list.txt	txt 文件路径，计算 table 时需要统计多张图在网络推理过程的数据的分布。 该 txt 中保存了所有需要计算的图片的路径，例如 example/路径下包含 1000 张 jpg 图片，则将路径下的图片都写到 txt 中。建议检测和分类模型提供 1000 张数据。
analysis_list (必配参数)	/path/image_list_2.txt	用于精度分析的 txt 文件路径。
scale_path	/path/save/scale.json	1. 每层 tensor 统计信息的保存路径，可重复使用，以 json 的形式保存; 2. 如不配置将在 save_path 路径下自动增加 scale.json 文件。 3. 流程：第一次运行 quantize 时，统计网络 tensor 得到原始 scale，保存到 json 中。在精度分析阶段，通过 config.json 调整量化方法或者切换某一层数据类型时，会自动读取 scale_path 下的 scale.json，不会再重复统计信息以减少量化时间; 4. scale_path 路径下的 scale.json 文件必须与模型对应，否则将报错终止程序运行。
quantized_method	per_channel	可配置的值有：per_channel、per_layer，默认 per_channel，目前只支持 per_channel。 量化不同 channel 的数据时（如 conv 的 weight、bias），单独统计各 channel 的分布计算 scale（per_channel）然后分别量化，或者统计所有 channel 的分布统一量化（per_layer）。
quantized_dtype	int8	可配置的值有：int8、int16，默认 int8。 量化网络时选择的全局计算精度，即未被

ESWIN Confidential

配置	示例	说明
		nodes_i8、nodes_i16 特别标记时，网络层计算时数据所选择的数据类型。 注意：设置成 int16 时，需要手动更改 table.json 中的 input 和 output 的 datatype 为 int16
requant_mode	max	可配置的值有：max、mean，默认 max。 每个网络层的输入输出 tensor 均会统计出 scale，以便对 tensor 进行量化。但每层输出的 scale 往往存在差异，需要对量化的整数乘除各层的 scale 实现恒等变形，通过插入 requant 实现（insert）；但这需要额外的计算开销，通过统一临近几层网络层的 scale 避免额外的乘法，统一 scale 的方法有取均值（mean）和取最大值（max）。
quantized_algorithm	at_eic	计算每一层的量化参数时采用的量化方法。 可配置的值有：at_eic(adaptive threshold)，mse_eic，默认 at_eic。 计算 table 时需统计各层输入输出 tensor 的数值分布，排除离群值确定有效的数值范围，根据数值范围确定 scale。 Yolov5 模型推荐使用 mse_eic，分类模型大多 at_eic 效果更好。
optimization_option	auto	模型优化等级，可配置的值有：option1、option2、auto，默认 auto。 算子的排列融合顺序对计算精度、速度都有影响，大部分情况下 option2 策略的计算速度、精度均更高，但对于某些网络层仍存在 option1 更优的情况；auto 则会根据网络结构自动调整优化策略。
bias_option	absmax	统计 bias 范围模式，可配置的值有：absmax、max，默认 absmax。 absmax 在大部分模型效果更优。
enable_analyse	true	是否开启精度分析，开启则 true，否则 false，默认开启
device	cuda	字符串，使用 cpu 填写"cpu"，使用 gpu 填写"cuda"，默认使用 gpu。
nodes_option1	[“conv_1”]	对于全局 option2 或 auto 量化的网络，可指定特

FSW Confidential

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

配置	示例	说明
		定 conv 节点为 option1 量化。
nodes_option2	["conv_1"]	对于全局 option1 或 auto 量化的网络，可指定特定 conv 节点为 option2 量化。
nodes_i8	["conv_1"]	对于全局 int16 量化的网络，某些层即使替换为 int8 计算仍能保持很高的计算精度，可以将其指定为 int8 量化，加快计算速度。
nodes_i16	["conv_1"]	对于全局 int8 量化的网络，某些层量化后计算精度降低很多，为了提高计算精度，可将其指定为 int16 量化，尤其对于检测模型多输出的最后一层 conv，为了获得最大的精度，手动填写对应的 node-name。 注意： 1. 对于存在同一 tensor 被不同分支共用的情况，不要更换分支第一层 layer 的数据类型。
output_variables_dtype	{"326": "int16","392": "int16", "458": "int16"}	字典数据类型，关键字是输出的 tensor 名称，值是数据类型，一般只有目标检测模型需要配置，yolov5 建议配置成 int16，保证检测精度。
mean (必配参数)	[0.485, 0.456, 0.406]	预处理中各通道数值需减去的均值(除 255)。参数格式是一个列表，即使用户使用自定义的预处理函数也需要将其配置到 config 中，归一化时，配置 Min； 客户在自定义预处理函数时，注意在配置文件中设置均值参数需要与预处理参数一致。
std (必配参数)	[0.229, 0.224, 0.225]	预处理中各通道数值需除以的方差(除 255)。参数格式是一个列表，即使用户使用自定义的预处理函数也需要将其配置到 config 中， 归一化时，配置 1/(Max-Min)； 客户在自定义预处理函数时，注意在配置文件中设置方差参数需要与预处理参数一致。
input_format (必配参数)	BGR	可配置的值有：RGB、BGR，网络所要求的输入图片的颜色层的顺序。
keep_ratio (必配参数)	true	布尔类型参数，若为 ture 则在 resize 过程中保持长宽比，否则不保持。
norm (必配参数)	true	预处理是否进行归一化，即对输入先除以 255；默认是 true，当是 true 情况下，均值方差数值必须在 0-1 之间。

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

配置	示例	说明
channel_first	true	根据输入 ONNX 的布局进行设置，当输入布局是 NHWC 则在配置文件中设置成 false； 用于可执行程序采用默认预处理阶段，对于 python 客户自定义预处理则不需设置。
resize_shape (必配参数)	[1024,2048]	图片缩放的长和宽的尺寸。 keep_ratio 为 true 时要求 resize_shape 为 1 维向量，为 false 时则是 2 维向量。
crop_shape (必配参数)	[1024,2048]	图片截取的长和宽的尺寸，目前为中心截取， python 接口中 resize_shape 需要大于 crop_shape。

2.2.2 量化命令说明

量化命令如下所示，具体的环境配置见 6.2.1。

```
eswin@debian:~# python Example_with_config.py --config_path /config.json --preprocess_name Yolo
```

上述命令中--preprocess_name 参数为用户选择的预处理函数。EsQuant 预先提供 Yolo、ImagenetOpencv 和 ImagenetTorch 的预处理函数。对于分类模型包括 ImagenetOpencv 和 ImagenetTorch 两种，ImagenetOpencv 使用 opencv 实现，ImagenetTorch 则是标准的 pytorch 实现方式。均值方差以及 shape 按需根据用户的模型和预处理方式进行调整。Yolo 预处理函数适用所有官方提供的 Yolo 检测网络。对于其他图像任务，客户可以自定义预处理函数，只需要补全预处理函数即可以使用，命令示例如下所示。

```
eswin@debian:~# python Example_with_config.py --config_path /config.json --preprocess_name ImagenetOpencv_custom --is_custom_preproc=True --custom_preproc_path= datasets/preprocess_custom.py
```

其中，--is_custom_preproc 参数指定了是否使用客户自定义的预处理函数，--custom_preproc_path 表示客户自定义预处理函数实现的 python 文件路径。对于自定义预处理函数，需在对应 python 文件中定义其实现类，并完善对应的__init___和__call__方法，另外需要定义一个名为 preprocess_registry 的字典，字典中元素的 key 即为--preprocess_name 参数中输入的预处理函数名，value 为对应预处理函数在 python 文件中实现的类名。

2.3 量化精度分析说明

该功能主要进行浮点数推理和量化推理并产生每层的数据，然后比较两者每层数据的余弦相似度，根据余弦相似度结果进行量化精度分析。

精度分析会获得两个 txt 文件，precision_accumulate_result.txt 统计的信息为量化模型和浮点数模

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

型逐层累计的余弦相似度误差；precision_reset_result.txt 统计的信息为量化模型和浮点数模型每层均重置输入获得的余弦相似度误差。文件内部会记录余弦相似度信息。

详细内容如下：

```
eswin@debian:~# cat precision_reset_result.txt
conv2_1     : 0.999249001344045
relu2_1     : 0.999094545841217
conv2_2     : 0.9989643030696445
relu2_2     : 0.9991797871059842
pool2       : 0.9991780718167623
```

首先查看 precision_accumulate_result.txt 中最后的 op-name（网络结构中最后一层 operation 名称）对应的 cos_dist(mean)是否大于 95%，如果满足大于阈值则视为误差在可接受范围内。如不满足需要查看另外一个文件即 precision_reset_result.txt。此时需要定位哪一层精度损失下降明显，然后在配置文件 config.json 中将此层替换成 int16（不要更换第一层 operation 和最后输出的 operation 的 datatype）。操作即在配置文件中找到 nodes_int16，在其中填写 node-name 字段。重新量化并开展精度分析，再排查量化精度误差。该过程结束后一般都可以获得较高的精度，如果精度还是低于阈值，则需要排查配置参数是否正确。

如客户使用自定义预处理函数，首先排查 input_format 参数，客户自定义预处理的颜色通道（客户自定义预处理函数）与模型实际输入是否一致；其次排查输入布局，EsQuant 默认接受 NCHW 的 layout，需要排查自定义预处理输入与模型要求是否一致，如果设置错误会导致累积误差极低。

如果此时精度依旧不达要求，则考虑配置 optimization_option 和 quantized_method 是否是按照建议配置设置。建议优先采用默认的最佳量化方法。

当最后的精度结果满足需求后，采用量化工具生成的 table.json（实际量化表名由 onnx 文件名以及上级目录构成）文件和模型编译工具即可生成在硬件上运行的模型。

2.4 EsGoldenDataGen 工具介绍

EsGoldenDataGen 工具通过开源框架 ONNXRuntime 对 ONNX 模型进行浮点数推理，产生输入和与之匹配的输出的二进制数据文件，用于对 EsACC 生成的量化模型进行验证，验证其编译生成过程和量化过程的精度正确性。注意此过程仅用于验证量化过程，提供的数据必须是 calibration 的标定数据。

EsGoldenDataGen 使用方法：

```
eswin@debian:~# python -m esquant.es_goldendata_gen.EsGoldenDataManager --config config_cls.json
```

配置文件为 json 文件，格式如下：

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

```
eswin@debian:~# cat config_cls.json
{
  "model": {
    "model_path":                    # 模型文件路径
      "/yolov3_sim_extract_416_notranspose_noreshape.onnx",
    "save_path": "/output/",          # 保存输入输出文件路径
    "batchsize": 4                   # batchsize
  },
  "dataset": {
    "data_root": "/image/",           # 图片文件路径
    "transform_cfgs": [
      {
        "type": "ToRGB"               # 预处理参数配置;自定义写 CustomPreProcess;默认提供的预处理包括
Imagenet 等
      },
      {
        "type": "Resize",
        "shape": 256
      },
      {
        "type": "CenterCrop",
        "crop_size": 224
      },
      {
        "type": "Normalize",
        "mean": [123.675, 116.28, 103.53],
        "std": [58.394161, 57.120009, 57.375638],
        "norm": false
      },
      {
        "type": "ToCHW"
      },
      {
        "type": "TransPrecision",
        "precision": "float32"
      }
    ]
  },
  "quant": {                          # 量化相关参数,
    "input": {                       # 输入相关参数
      "format": {
        "input1": [1,3,224,224,1],    # 与 EsAAC 输入保持一致
        "input2": [1,3,224,224,1]     # 与 EsAAC 输入保持一致
      },
      "color_format_convert": true,     # 表示是否调整 dump 数据的颜色通道顺序
      "scale": {
        "input1": 0.02042430216871847, # 输入 1 对应 table 中的 int8 或 int16 的 step 数值
        "input2": 0.02042430216871847, # 输入 1 对应 table 中的 int8 或 int16 的 step 数值
      },
      "data_type": 0,                 # 转化的数据类型:0 代表 int8，1 代表 int16
      "channel_first": true
```

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

```
    },
    "output": {
      "format": {
        "output1": [1,255,20,20,1]      # 支持格式为 NCHW
        "output2": [1,255,40,40,1]      # 支持格式为 NCHW
      },
      "scale": {
        "output1": 0.1886681467294693  # 输出 1 对应 table 中的 int8 或 int16 的 step 数值
        "output2": 0.1886681467294693  # 输出 2 对应 table 中的 int8 或 int16 的 step 数值
      },
      "data_type": 0                  # 转化的数据类型:0 代表 int8，1 代表 int16
    }
  }
}
```

- 多输入多输出在后面增加对应的 tensor-name 和相关 layout 以及 scale 信息。
- shape 信息需要和 EsAAC 的输入 shape 保持一致，如输入是 NHWC，对于输入的数据是图片调整[1,1,224,224,3]；如输入是 NCHW，对于输入数据信息就调整为[1,3,224,224,1]。
- 对于多 batch 模型，shape 中的 N 需要修改成 batch 数量,比如 batch=6 情况下 shape=[6,255,20,20,1]，并且提供的 ONNX 模型也必须为多 batch。
- 保存的数据是指定路径下二进制文件，文件名称由 tensor 名称和图片名称构成，中间由"_"字符连接。
- 客户自定义预处理，需要在客户在 esquant/es_goldendata_gen/CustomPreProcess.py 中实现 custom_preprocess(img)函数，并在配置文件中将 transform_cfgs 的 type 用 Custom。
- Scale 信息是通过量化生成的 table 中获取，即按照 2.2 生成的 table 文件得到；
  输出的 datatype 必须和量化模型一致，对于 detection 任务建议 datatype 为 1（int16）。
- color_format_convert 表示在 dump 数据阶段是否调整模型输入的颜色通道顺序，如果模型输入是 bgr，但是想要获取 rgb 的数据则使用 true，否则填写 false，其他情况同理。

3. EsAAC 使用指南

3.1 EsAAC 工具介绍

EsAAC 是 ESWIN 面向自研芯片的深度学习编译器，可以将主流人工智能框架（TensorFlow、Pytorch、Caffe、ONNX 等）的网络模型转换成统一的中间表达(IR)，同时进行基于 EIC7700 硬件的模型优化，产生 EIC7700 可以加载运行的离线模型，即集转换、优化、编译于一体。EsAAC 适用的网络模型包括图像分类、目标检测、图像分割等。

具体支持的功能包括：

- 支持图像分类，目标检测，图像分割等领域模型的优化和编译。支持输入的网络模型格式：ONNX
- 支持多个 batch 模型的编译,最大 batchsize 为 16（具体数量取决于实际模型），离线模型只支

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

持静态多 batch；

- 支持利用不同异构计算单元，并设置各个计算单元最大的使用资源情况；
- 支持利用片上高速存储 SRAM 进行缓存优化；
- 支持用户定制的前处理或后处理计算（包括输入图片的颜色转换和归一化，以及输出非最大值抑制等计算）。

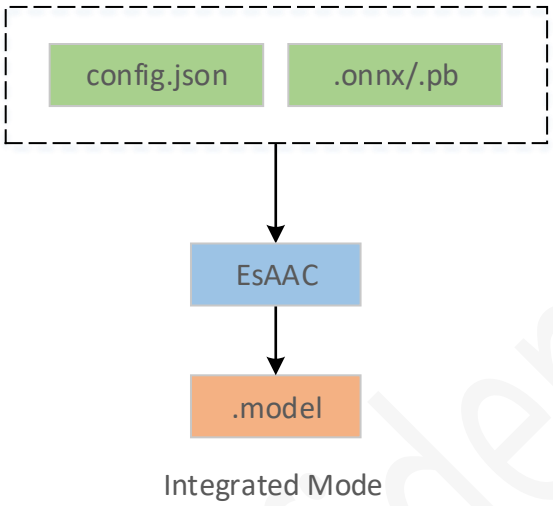

图 3-1 EsAAC 编译流程

EsAAC 的编译主要流程如图 3-1 所示，主要分为以下步骤：

- EsAAC 读取用户输入的已训练好的模型文件，根据配置文件和命令行参数开始编译。
- 用户可以选择配置 SRAM 可用数量利用 SRAM 缓存中间结果。EsAAC 采用高效算法调度与分配 SRAM 空间，提高访存效率。
- EsAAC 对训练好的 ONNX 模型解析并将计算配置映射到硬件单元，生成可由 Runtime 运行的离线模型，此模型是序列化保存的二进制文件，文件名为 xxx.model。

3.2 生成模型功能介绍

EsAAC 工具提供名为 EsAAC 的可执行文件，用于将 ONNX 格式的模型转换为可以在硬件上执行的.model 格式文件。

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

表 3-1 命令行参数说明

参数	说明	默认值	是否必选
--input-model	输入模型文件	无	是
--input-nodes	模型的输入节点	无	否
--input-shapes	模型输入各节点的形状，若配置此参数，则必须配置 input-nodes	模型各节点自带形状	否
--output-nodes	模型的输出节点	无	否
--output-shapes	模型输出各节点的形状， 若配置此参数，则必须配置 output-nodes	无	否
--quant-stats	量化统计文件	无	否
--equant-config-path	量化配置文件，配置量化相关的参数（参考其他章节）	无	否
--loadable-name	输出模型文件的名称	输入模型文件的名称.model	否
--run-weight-csc	是否对 weight 进行 csc 压缩处理	no	否
--sram-capacity	SRAM 的空间大小，单位 KB	4096	否
--dsp-core	指定 DSP 的个数，1~4	1	否
--inference-datatype	需要生成模型的数据类型，当使用量化文件或配置文件时，此参数不起作用，可不配置，可选值为 DT_QINT8、DT_QINT16	DT_QINT8	否
--reverse-input-channel	是否切换输入模型的通道位置	no	否
--input-format	设置输入形状格式是 NCHW 或 NHWC	NCHW	否
--convert-to-qm	设置是否将输入模型转换为 qm 格式，提供给 EsQuant 使用	no	否
--run-device	指定生成模型的推理硬件类型，可不配置，可选值为 npu、dsp 和 all	npu	否
--enable-uosp	编译 vit 模型需加上此选项进行 shape 变换，大语言模型无需使能。	False	否
--lutdsp	设置是否使用 DSP 的 lut 算子。	False	否

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

注意：目前支持通过" --run-device all"命令生成 throughput 模型的有：[Inception_V1, inception-v3, inception-v4, resnet18, resnet50, segnet, squeezenet_v1.0, squeezenet_v1.1, mobilenet_v1, mobilenet_v2, densenet121, densenet201, yolov2, yolov3, senet50, yolov5s]，并且通过"—input-model"传入的文件名需要和上述名称保持一致，如："—input-model resnet50.onnx"。

3.3 EsSimulator 工具介绍

EsSimulator 模型离线测试工具，用于比较生成的模型是否与原始模型计算结果一致。EsSimulator 接收由 EsACC 生成的离线模型，根据 EsGoldenDataGen 生成的输入和预期的输出 Tensor 数据来验证模型的正确性。EsSimulator 是一个离线验证工具，可快速验证模型正确性，方便调试定位模型问题及评估模型效果。

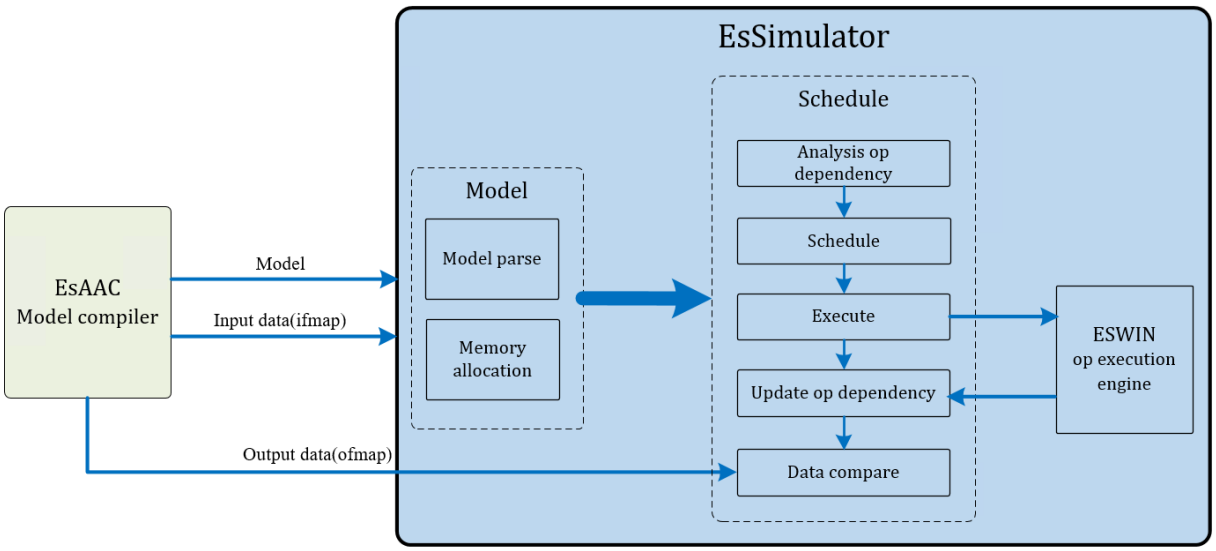

图 3-2 EsSimulator 执行架构

EsSimulator 的使用方法可以通--help 进行显示。

```
eswin@debian:~# ./EsSimulator –help
Usage:
EsSimulator --model=<xx.model> --input=<ifmap.bin> --output=<ofmap.bin> [--tolerance=xx, default:0.05] [--dsp_op_dir=xx, default:./dsp_kernels]
EsSimulator --model=<xx.model> --input=<ifmap1.bin,ifmap2.bin,ifmap3.bin> --output=<ofmap1.bin,ofmap2.bin,ofmap3.bin> [--tolerance=xx, default:0.05] [--dsp_op_dir=xx, default:./dsp_kernels]
```

对 EsSimulator 参数说明如下：

--model 必选参数，指定输入的 model 文件

--input 必须参数，指定输入的 ifmap，支持多个 ifmap

--output 必须参数，指定模型输出文件 ofmap，支持多个 ofmap

--tolerance 可选参数，指定误差容忍度，默认 0.009，当模型数据为 float32 或 float16 时有效

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

4. NPU Runtime 开发指南

NPU Runtime 是 ENNP 提供的一套加载 NN 模型的运行时系统。可以直接加载基于 ENNP 工具量化编译的离线模型，用于实现目标识别、图像分类等功能。用户基于 NPU Runtime 开发智能分析方案，由编译器自动分析并优化执行过程，并自动生成优化好的离线模型，最大化复用 NPU 硬件，提升硬件利用率并优化系统功耗，NPU Runtime 的架构框图参考: 图 1-2 NPU Runtime 框图。

4.1 输入模型要求

NPU Runtime 输入的模型必须是 ESWIN EsAAC 模型编译器产生的网络模型，模型生成之后建议使用 ESWIN EsGoldenDataGen / EsSimulator 工具来校验该模型的正确性和有效性。NPU Runtime 支持的模型具体参考：EsAAC 使用指南。

4.2 开发流程

4.2.1 NPU Runtime 接口调用

用户可以根据需求场景调用 NPU Runtime 的 API 来实现对应的功能，最新的 NPU Runtime 实现了动态 Batch size 和复合模型功能提了 NPU Runtime 应用开发的灵活性和推理的性能。

- 动态 Batch size 功能可以使用户在任何时候输入任意张要推理的图片，就是说一次可以提交 N 个 Task 进行推理（一个张图对应一个 Task），N 可以是任意值。
- 复合模型会最大利用 ENNP 硬件能力来提高运算的速度。

  说明：复合模型支持 NPU 和 DSP 硬件并行推理。

NPU Runtime 的接口调用如图 4-1 所示。

ESWIN Confidential

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

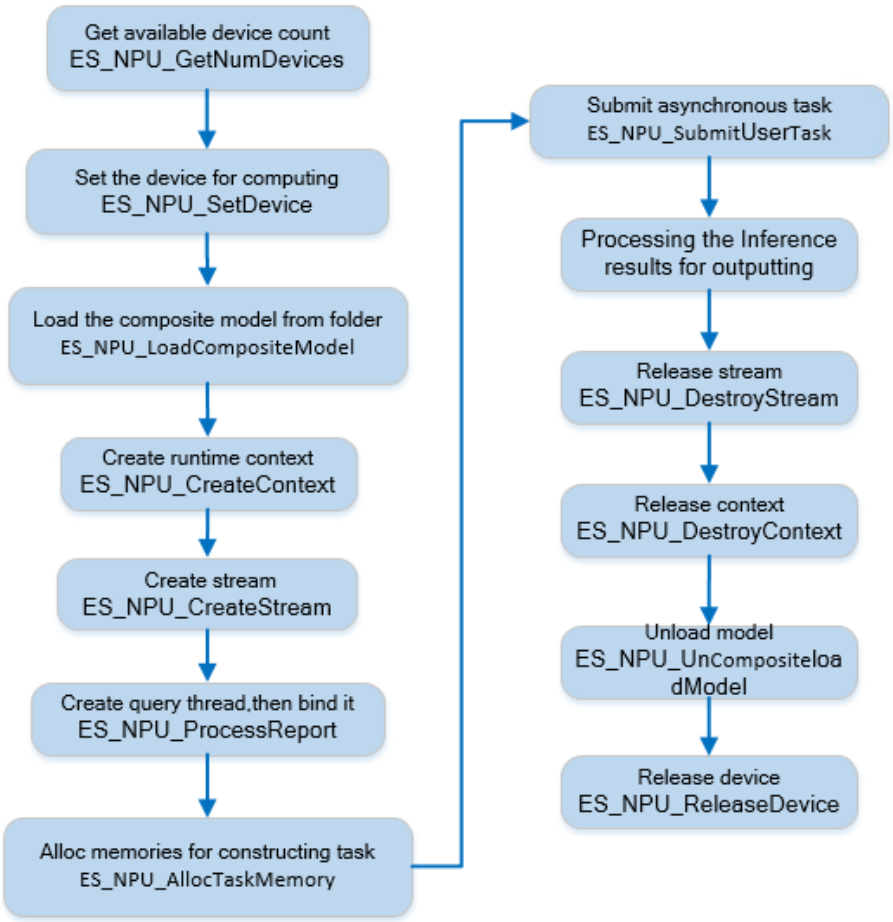

图 4-1 NPU 接口调用流程

4.2.2 使用 es_run_model 进行模型评测

为了方便用户调用 NPU Runtime 测评模型，NPU Runtime 提供了 es_run_model 工具。该工具提供了若干选项，可以很方便地测试模型速度和精度等信息。es_run_model 的参数信息如下：

```
usage: es_run_model --model=string [options] ...
options:
-M, --mode              the mode of es_run_model(int [=0], 0 is sync mode, 1 is nbatch mode.)
-m, --model             path to a model file (string)
-b, --batch             the model batch count (int [=1])
-r, --repeat            repeat times running a model (int [=1])
-w, --warmup            repeat times before running a model to warming up (int [=0])
-i, --input-dir         the directory of each inputs (folders) located (string [=])
-o, --output-dir        the directory of each outputs (folders) will saved in (string [=])
-l, --list              the list of inputs which will test (string [=])
-v  --verify            verify outputs after running model
-s  --save-perf         save performance result(min, max, avg) as a json file
-?, --help              print this message
```

详细应用示例见 6.6.1 节内容。

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

4.3 NPU Runtime 调试

NPU Runtime 目前提供了基于 Release 的调试信息，默认的调试信息是 Error 级别。

```
[1970-01-01 08:01:01] [E] [ES_NPU] [-1466355936] [readModelToMem] [186] couldn't open /IBU_8T/IBU_SOFTWARE/npu_release_models/npu_release/opt/eswin/data/models/mobilenet_v1_int8_1x224x224x3I

[1970-01-01 08:01:01] [E] [ES_NPU] [-1466355936] [loadModel] [34] read model to memory error, err=-1609605117
```

另外，我们也可以通过返回值的错误码来进行调试，具体参考《ENNP 开发者手册》。

5. AcceleratorKit 开发指南

AcceleratorKit 是用于调用未编译入 ONNX 模型的算子的 API 库，如一些较为复杂的前处理、后处理算子等。AcceleratorKit 提供了可在不同类型设备上执行的计算加速功能，以便最大化发挥不同类型设备的计算加速能力。

5.1 软件框架

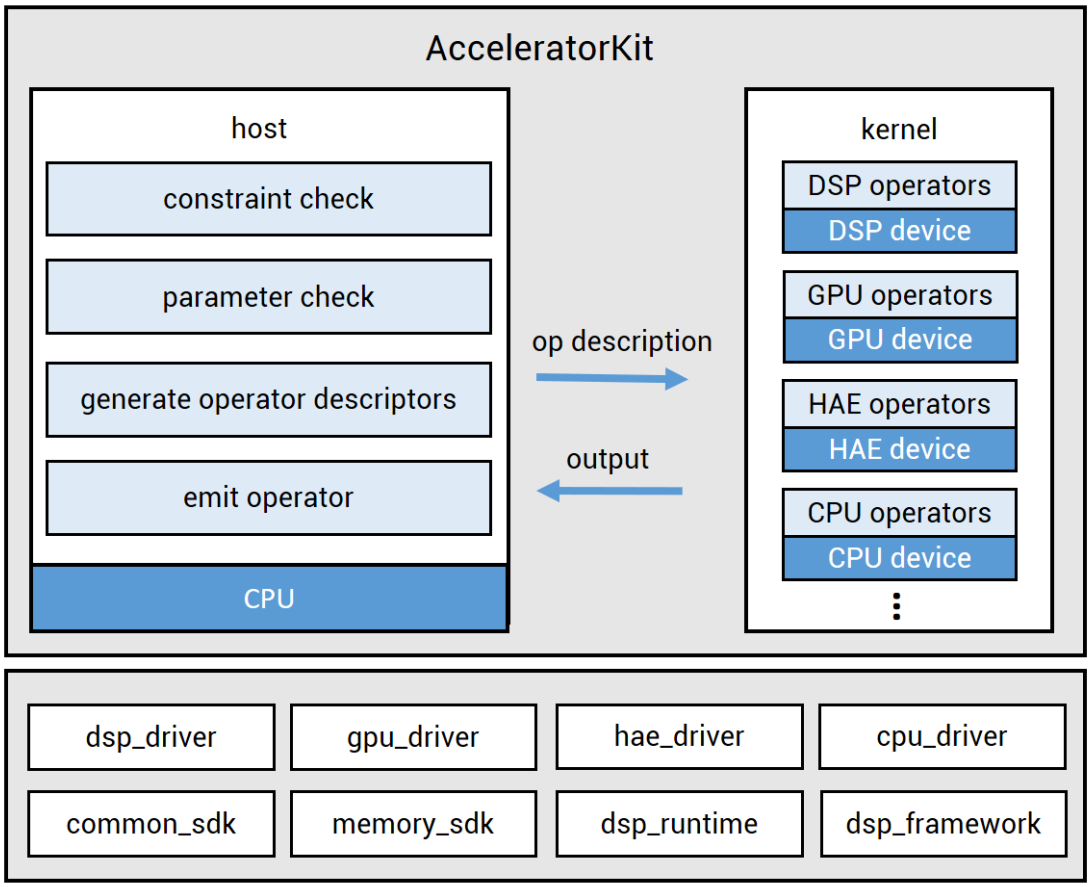

图 5-1 AcceleratorKit 软件架构

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

当前支持的算子如表 5-1 所示。

表 5-1 AcceleratorKit 支持的算子列表

算子名称	算子功能
ES_AK_DSP_CosDistance	计算两组数据之间的余弦距离。
ES_AK_DSP_Argmax	进行 Argmax 计算，对输入的数据的指定维度进行最大排序。
ES_AK_DSP_DetectionOut	对检测网络的输出 feature map 做后处理。
ES_AK_DSP_Softmax	进行 Softmax 计算。
ES_AK_DSP_PerspectiveAffine	对图像做透视变换。
ES_AK_DSP_WarpAffine	对图像做仿射变换。
ES_AK_CPU_SimilarityTransform	计算相似变换矩阵。

算子的详细定义和使用说明请参考《ENNP 开发者手册》，或咨询技术支持。

5.2 调用流程

5.2.1 依赖模块

- DSP 相关模块，包括 dsp_compiler、dsp_runtime、dsp_driver、dsp_fw、dsp_kernels.
- common_sdk，提供公共的数据类型。
- memory_sdk，提供 memory 管理工具。

5.2.2 开发步骤

- 调用 ES_AK_GetVersion 等接口完成获取版本信息、设置日志等级等操作；
- 调用 ES_AK_Init 初始化硬件设备资源；
- 根据具体需求调用 ES_AK_SetDevice 配置运算设备；
- 调用 ES_AK_GetDevice 获取运算设备；
- 通过 ES_SYS_MemAlloc 配置算子的输入输出内存；
- 根据配置的具体硬件设备，调用对应的算子接口，如 ES_AK_DSP_DetectionOut；
- 算子计算完成调用 ES_SYS_MemFree 释放配置的输入输出内存；
- 所有计算结束后需要调用 ES_AK_Deinit 释放设备资源；

详细的 API 使用参考《ENNP 开发者手册》中 AcceleratorKit 章节。

ESWIN Confidential

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

6. 模型开发示例

本章介绍 ONNX 模型转换、编译及开发板部署运行的完整操作流程。完整的 ENNP 流程如下：

- 使用基于 x86 平台的 PC 端的 EsQuant 工具，ONNX 原始模型和量化配置(Json 文件)生成量化表 table.json（实际量化表名由 onnx 文件名以及上级目录构成）；
- 使用基于 x86 平台的 PC 端的 EsAAC 模型编译，ONNX 原始模型和量化表 table.json（实际量化表名由 onnx 文件名以及上级目录构成）生成 ENNP 硬件支持的模型；
- 使用基于 x86 平台的 PC 端的工具 EsSimulator 仿真 ENNP 硬件验证模型正确性；
- 使用基于 RSIC-V 平台的工具 es_run_model 在硬件上验证模型的功能并统计其性能等；
- 使用基于 RSIC-V 平台调用 NPU Runtime 接口开发用户 App 并在硬件上执行任务推理。另外，NPU Runtime 提供了 sample_npu 参考代码加速用户 App 的开发。

6.1 Sample 文件说明

Sample 文件涉及到基于 x86 平台 PC 端的工具和基于 RSIC-V 平台开发板的开发 Sample，因此 ENNP 的 Sample 分为 pc sample 和 board sample 进行描述。

6.1.1 pc sample

pc sample 目录如下所示，EsNNTools 文件夹中包含所需工具链的 docker 镜像文件，sample/yolov3 文件夹中存放了量化、编译等环节所需要的输入文件。

ESWIN Confidential

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

```
└─── EsNNTools
  ├─── esquant-1.0-py3-none-any.whl
    ├─── esquant_docker.tar
    ├─── esaac_essimulator_docker.tar
    ├─── Example_with_config.py
    ├─── libs
    │      ├─── libconv_model_creator.so
    │      ├─── libmanager.so
    │      ├─── libmapper.so
    │      └─── libtimeloop-model.so
  └─── sample
        ├─── yolov3
              │    ├─── esaac
              │    │    └─── table.json
              │    ├─── esquant
              │    │    ├─── alys_list.txt
              │    │    ├─── config.json
              │    │    └─── img_list.txt
              │    ├─── essimulator
              │    │    ├─── ifmap.bin
              │    │    ├─── ofmap0.bin
              │    │    ├─── ofmap1.bin
              │    │    ├─── ofmap2.bin
              │    │    └─── yolov3.json
              │    └─── model
              │         └─── yolov3_sim_extract_416_notranspose_noreshape.onnx
              └─── yolov5
          ├─── eaac
          │    └─── table.json
          ├─── esquant
          │    ├─── alys_list.txt
          │    ├─── config.json
          │    └─── img_list.txt
          ├─── essimulator
          │    ├─── ifmap.bin
```

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

```
|   ├── ofmap0.bin
|   ├── ofmap1.bin
|   ├── ofmap2.bin
|   └── yolov5.json
└── model
    └── yolov5s_416-sim_extract.onnx
```

6.1.2 board sample

以 root 帐户安装 sample 的 deb 包，需要链接以太网：

- 安装 npu sample 包：

  apt install es-sdk-sample-npu

  安装后 sample 位于/opt/eswin/sample-code/npu_sample/npu_runtime_sample 目录

- 安装 npu resnet50 包：

  apt install es-sdk-sample-npu-resnet50

  安装后 sample 位于/opt/eswin/sample-code/ npu_sample /npu_resnet50_sample 目录

- 安装 npu mobilenetv2 包：
  apt install es-sdk-sample-npu-mobilenetv2
  安装后 sample 位于/opt/eswin/sample-code/npu_sample /npu_mobilenetv2_sample 目录

6.1.2.1 es-sdk-sample-npu

NPU Runtime 提供的示例目录(npu_runtime_sample)主要包括调用 NPU Runtime 接口的示例代码和模型目录：src 和 models，详细目录结构如下：

ESWIN Confidential

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

```
├─── models
│   └─── yolov3
│       ├─── es_yolov3_classes.txt
│       ├─── es_yolov3_post_process.json
│       ├─── es_yolov3_pre_process.json
│       ├─── git_yolov3_416_mix_1x3x416x416_dyn_latency.model
│       ├─── git_yolov3_416_mix_1x3x416x416_dyn_latency.ofmap_order.txt
│       ├─── git_yolov3_416_mix_1x3x416x416_dyn.model
│       ├─── input
│       └─── model.json
└─── src
    ├─── build.sh
    ├─── CMakeLists.txt
    ├─── README.md
    ├─── sample_npu_comm.cpp
    ├─── sample_npu_comm.h
    ├─── sample_npu.cpp
    └─── utils
```

- 示例代码目录(src)

NPU Runtime 提供的示例代码的入口是 sample_npu.cpp 文件，这个文件代码包含了一系列示例的 Case 例如：同步，异步，多 Stream，Context 以及动态 batchsize 的 Case 等，每种 Case 在代码中都有单独的入口函数，用户可以轻松找到适合自己应用的 Sample。

README.md 文档中包含编译 sample_npu 代码方法以及运行 sample_npu 命令行示例。

Utils 中主要包含一些辅助的工具库。例如：图片的前处理(preprocess)和推理后数据的后处理(postprocess)等。

- 模型目录（models）
  - 配置文件介绍：

    es_yolov3_classes.txt 包含类别标签信息。

    es_yolov3_pre_process.json 包含预处理的策略和参数。

    es_yolov3_post_process.json 包含后处理算子相关参数。
  - input 目录：

    该目录包含 Pictures 和 Preprocessed 目录。Pictures 目录存放待推理的原始图片文件。

    Preprocessed 目录存放已经预处理(Decode，Resize，Normalization 和 CVT)后的二进制

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

数据文件。

- output 目录：

  output 目录在这个 Sample 示例中暂时没有用到，es_run_model 工具会使用 output 目录中的数据来校验 NPU Runtime 推理的结果正确性。

6.1.2.2 es-sdk-sample-npu-resnet50

NPU resnet50 提供的示例目录(npu_resnet50_sample)主要包括调用 NPU Runtime 接口的示例代码和模型目录、二进制程序：src 和 models、 bin，详细目录结构如下：

ESWIN Confidential

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

```
├─── bin
│   └─── es_ai_inference
├─── models
│   ├─── es_resnet50_classes.txt
│   ├─── es_resnet50_post_process.json
│   ├─── es_resnet50_pre_process.json
│   ├─── git_resnet50_mix_4x3x224x224_dyn_latency.model
│   ├─── git_resnet50_mix_4x3x224x224_dyn_latency.ofmap_order.txt
│   ├─── git_resnet50_mix_4x3x224x224_dyn.model
│   ├─── git_resnet50_mix_4x3x224x224_dyn_throughput.model
│   ├─── git_resnet50_mix_4x3x224x224_dyn_throughput.ofmap_order.txt
│   └─── input
└─── src
    ├─── build.sh
    ├─── CMakeLists.txt
    ├─── common
    │   └─── utils
    ├─── es_ai_inference.cpp
    ├─── es_ai_inference.h
    ├─── list.txt
    ├─── main.cpp
    ├─── README.md
    ├─── runtime
    │   └─── include
    └─── utils
        ├─── json
        ├─── npu_test_utils.cpp
        ├─── npu_test_utils.h
        ├─── postprocess
        └─── preprocess
```

- 示例代码目录(src)

  NPU resnet50 提供的示例代码的入口是 es_ai_inference.cpp 文件。

  README.md 文档中包含编译 resnet50 sample 代码编译方法以及运行命令。

  Utils 中主要包含一些辅助的工具库。例如：图片的前处理(preprocess)和推理后数据的后处理(postprocess)等。

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

- 模型目录（models）
  - 配置文件介绍：

    es_resnet50_classes.txt 包含类别标签信息。

    es_resnet50_pre_process.json 包含预处理的策略和参数。

    es_resnet50_post_process.json 包含后处理算子相关参数。
  - input 目录：

    该目录存放待推理的原始图片文件。
- 二进制程序目录（bin）

  该目录存放二进制可执行程序。

6.1.2.3 es-sdk-sample-npu-mobilenetv2

NPU mobilenetv2 提供的示例目录(npu_mobilenetv2_sample)主要包括调用 NPU Runtime 接口的示例代码和模型目录、二进制程序：src 和 models、 bin，详细目录结构如下：

ESWIN Confidential

```
├─── bin
│   └─── es_ai_inference
├─── models
│   ├─── es_mobilenet_classes.txt
│   ├─── es_mobilenet_post_process.json
│   ├─── es_mobilenet_pre_process.json
│   ├─── git_mobilenetv2_mix_4x3x224x224_dyn_latency.model
│   ├─── git_mobilenetv2_mix_4x3x224x224_dyn_latency.ofmap_order.txt
│   ├─── git_mobilenetv2_mix_4x3x224x224_dyn.model
│   ├───git_mobilenetv2_mix_4x3x224x224_dyn_throughput.model
│   ├─── git_mobilenetv2_mix_4x3x224x224_dyn_throughput.ofmap_order.txt
│   └─── input
└─── src
  ├─── build.sh
  ├─── CMakeLists.txt
  ├─── common
  │   └─── utils
  ├─── es_ai_inference.cpp
  ├─── es_ai_inference.h
  ├─── list.txt
  ├─── main.cpp
  ├─── README.md
  ├─── runtime
  │   └─── include
  └─── utils
    ├─── json
    ├─── npu_test_utils.cpp
    ├─── npu_test_utils.h
    ├─── postprocess
    └─── preprocess
```

- 示例代码目录(src)

  NPU mobilenetv2 提供的示例代码的入口是 es_ai_inference.cpp 文件。

  README.md 文档中包含编译 mobilenetv2 sample 代码编译方法以及运行命令。

  Utils 中主要包含一些辅助的工具库。例如：图片的前处理(preprocess)和推理后数据的后处理(postprocess)等。

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

- 模型目录（models）
  - 配置文件介绍：

    es_mobilenet_classes.txt 包含类别标签信息。

    es_mobilenet_pre_process.json 包含预处理的策略和参数。

    es_mobilenet_post_process.json 包含后处理算子相关参数。
  - input 目录：

    该目录存放待推理的原始图片文件。
- 二进制程序目录（bin）

  该目录存放二进制可执行程序。

6.2 安装开发环境

本节介绍使用 ENNP 工具链前的开发环境准备工作。为方便用户部署安装工具链，ENNP 提供 docker 容器进行工具链集成，其中 EsQuant 打包为一个镜像，EsAAC 和 EsSimulator 打包为一个镜像，共两个 docker 镜像，请用户注意区分。下面分别介绍这两个容器镜像的安装和启动，所涉及到的镜像文件均在 pc sample 的 EsNNTools 文件夹下。

6.2.1 EsQuant 部署

EsQuant 以 docker 形式发布，整体安装运行 docker 容器流程如下：

- 加载 docker 镜像。

```
eswin@debian:~# docker load -i  ${EsNNTools}/esquant_docker.tar
```

- 查看 docker 镜像。

```
eswin@debian:~# docker images

REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
esquant      v0.7      c711e4a379ca   4 days ago    18.2GB
```

- 创建容器。

```
# 启动 gpu 镜像
# --privieged=true 特权模式：赋予容器几乎与主机相同的权限
eswin@debian:~# docker run --name [container_name] -it --runtime=nvidia --privieged=true --gpus all -v /hostPath:containerPath esquant:v0.7 /bin/bash
eswin@debian:~# docker run --name [container_name] -it --runtime=nvidia --gpus all -v /hostPath:containerPath esquant:v0.7 /bin/bash

# 启动 cpu 镜像
eswin@debian:~# docker run --name [container_name] -it -v /hostPath:containerPath esquant:v0.7 /bin/bash
```

- 安装 whl 包。

```
root@d76a884a0d56:~$cd EsNNTools/
root@d76a884a0d56:~$pip3 install esquant*.whl
```

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

- 查看版本信息和 git-hash 信息以确认是否部署成功。

```
root@d76a884a0d56:~$python3
import esquant
esquant.version()

version: 1.0.0
sha id: af6da07d5c205583f1a71cfee12339ed10623d08
```

6.2.2 EsAAC 和 EsSimulator 部署

EsAAC 和 EsSimulator 以同一 docker 形式发布，运行流程如下：

- docker 镜像：

```
eswin@debian:~# docker load -i ${EsNNTools}/esaac_essimulator_docker.tar
```

- 查看 docker 镜像：

```
eswin@debian:~# docker images

REPOSITORY              TAG       IMAGE ID       CREATED         SIZE
esaac_essimulator       latest    26c09ab3286b   7 minutes ago   4.02GB
```

- 创建容器：

```
eswin@debian:~# docker run [-v hostPath:containerPath] --name [container_name] -it --rm esaac_essimulator:latest /bin/bash
```

- 查看 EsAAC 版本信息：

```
eswin@d76a884a0d56:~$ ls
EsAAC  EsSimulator  EsSimulator_copyright.txt  dsp_kernels

eswin@ d76a884a0d56:~$ ./EsAAC --version
eaac version: 0.0.3
```

- 查看 EsSimulator 版本信息：

```
eswin@d76a884a0d56:~$ ls
EsAAC  EsSimulator  EsSimulator_copyright.txt  dsp_kernels

eswin@d76a884a0d56:~$ ./EsSimulator --version
EsSimulator version: 0.0.3(Fri Jul 19 14:06:01 2024 +0800).
```

6.3 ONNX 模型导出

本节介绍如何从官网导出 yolov3 模型以及裁剪模型。

6.3.1 Yolov3 模型导出

从 GitHub 链接上下载官网代码，并按照下述流程进行导出 yolov3 模型，量化工具支持的 opset 版本 11-14，导出模型时需要选择 11-14 版本：

注意官方在 requirements.txt 中将 onnx 库和 onnx-simplifier 库注释掉，安装环境时记得修改并安装；另外 torch 和 torchvision 版本需要分别安装 torch==1.12.0 和 torchvision==0.13.0 版本。

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

```
git clone https://github.com/ultralytics/yolov3  # clone
cd yolov3
pip install -r requirements.txt  # install

mkdir model_file
wget https://github.com/ultralytics/yolov3/releases/download/v9.6.0/yolov3.pt
python3 export.py --weights model_file/yolov3.pt --img-size 416 --simplify --opset 13 --include onnx
```

6.3.2 Yolov3 模型裁剪

由于导出的模型包含部分后处理操作，且量化工具暂时不支持后处理操作，所以需要裁剪掉后处理部分。注意裁剪部分为 conv 后的 operations。裁剪脚本填写 input_names 和输出的 output_names 即可。

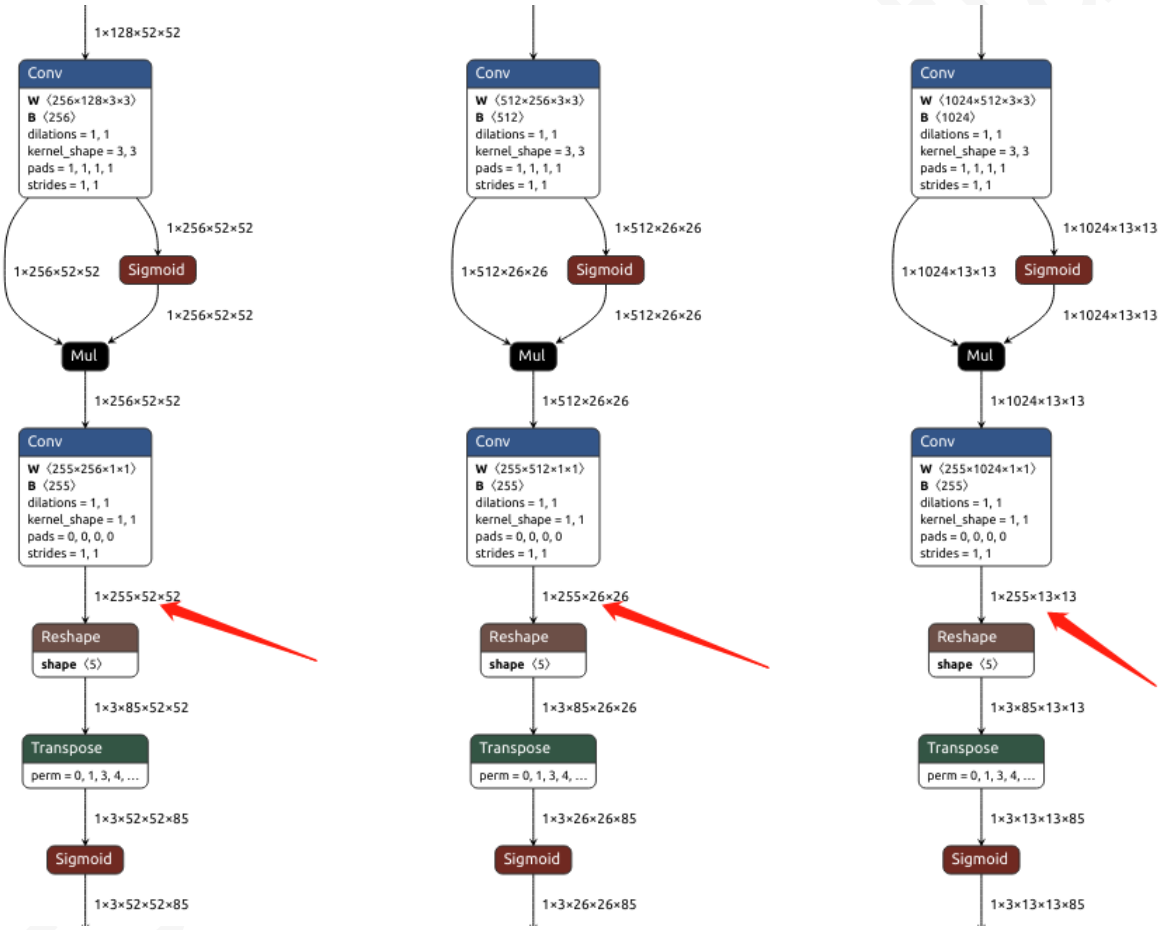

图 6-1 模型裁剪示例

```
import onnx

input_onnx = "yolov3.onnx"
input_names = ["images"]
output_names = ["onnx::Shape_406", "onnx::Shape_461", "onnx::Reshape_516"]

output_onnx = input_onnx
cut_suffix = "_sim_extract_416_notranspose_noreshape." + input_onnx.split('.')[-1]
new_output_onnx = output_onnx.replace(".onnx", cut_suffix)
print(new_output_onnx)

onnx.utils.extract_model(input_onnx, new_output_onnx, input_names, output_names)
```

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

6.4 ONNX 模型量化及精度分析

本节介绍使用 EsQuant 工具执行量化和量化精度分析的流程。

6.4.1 使用 EsQuant 生成量化模型

6.4.1.1 配置 config.json 文件

针对 yolov3 模型，推荐配置参数设置如下，其中路径相关的参数请用户根据开发环境实际情况进行修改。各参数的含义请参考 2.2.1。完整的 config.json、img_list.txt、alys_list.txt 文件均可在 EsNNTools/sample/yolov3/esquant 下获取。img_list.txt 中所用到的图片为 coco2017 数据集，请用户自行下载。

注意标定数据需要在 1000 张图片以内，且要根据是否存在 gpu 在配置文件中配置。

ESWIN Confidential

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

```
root@d76a884a0d56:~$ cat config.json
{
    "version": "1.3",
    "model": {
        "model_path": "/yolov3_sim_extract_416_notranspose_noreshape.onnx",
        "save_path": "/home/yolov3/",
        "images_list": "/home/yolov3/cal_lists.txt",
        "analysis_list": "/home/yolov3/file_list.txt"
    },
    "quant": {
        "quantized_method": "per_channel",
        "quantized_dtype": "int8",
        "requant_mode": "mean",
        "quantized_algorithm": "at_eic",
        "optimization_option": "auto",
        "bias_option": "absmax",
        "nodes_option1": [],
        "nodes_option2": [],
        "nodes_i8": [],
        "nodes_i16": [],
        "mean": [
            0,0,0
        ],
        "std": [
            1,1,1
        ],
        "norm": true,
        "scale_path": "",
        "enable_analyse": true,
        "device": "cpu",
        "output_variables_dtype": {
            "onnx::Shape_406": "int16",
            "onnx::Shape_461": "int16",
            "onnx::Reshape_516": "int16"
        }
    },
    "preprocess": {
        "input_format": "RGB",
        "keep_ratio":false,
        "resize_shape": [
          416,416
        ],
        "crop_shape": [
          416,
          416
        ]
    }
}
```

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

6.4.1.2 执行量化

运行以下脚本执行量化，这里使用内置的 Yolo 预处理方式，用户也可以基于提供的样例自行定义预处理函数，样例代码在 EsNNTools 目录下获取。

```
root@d76a884a0d56:~$python Example_with_config.py --config_path /config.json --preprocess_name Yolo
```

运行代码后在终端得到如下显示表示量化完成。

```
正准备量化你的网络，检查下列设置:
TARGET PLATFORM : EIC
NETWORK INPUTSHAPE : [1, 3, 224, 224] [2024-01-30 19:56:02][warn][config.cpp:66]:Configuration
[10:05:12] ESQUANT Quantize Simplify Pass Running ... Finished.
[10:05:12] ESQUANT Quantization Fusion Pass Running ... Finished.
Calibration Progress(Phase 1): 100%|████████████████| 1000/1000 [00:13<00:00, 75.10it/s]
Calibration Progress(Phase 2): 100%|████████████████| 1000/1000 [00:16<00:00, 62.34it/s]
Finished.
[10:05:54] ESQUANT Quantization Fusion Pass Running ... [Warning] More than 1 pattern root was found, Complext Pattern might cause memory overflow ...
[Warning] More than 1 pattern root was found, Complext Pattern might cause memory overflow ...
[Warning] More than 1 pattern root was found, Complext Pattern might cause memory overflow ...
Finished.
[10:05:54] ESQUANT Parameter Quantization Pass Running ... Finished.
[10:05:54] EIC Parameter Quantization Pass Running ...
[10:05:54] EIC Input Tensor Quantization Pass Running ... Finished.
[10:05:54] ESQUANT Passive Parameter Quantization Running ... Finished.
[10:05:54] ESQUANT Quantization Alignment Pass Running ... Finished.
[10:05:54] ESQUANT Parameter Baking Pass Running ... Finished.
Network Quantization Finished.
正计算网络量化误差(COSINE)，最后一层的误差应小于 0.1 以保证量化精度:
Analysing Graphwise Quantization Error(Phrase 1):: 100%|██████████| 8/8 [00:01<00:00, 6.30it/s]
Analysing Graphwise Quantization Error(Phrase 2):: 100%|██████████| 8/8 [00:01<00:00, 6.04it/s]
网络量化结束，正在生成目标文件:
[Warning] File /quantized-yolov3-eic-at-0715.onnx is already existed, Exporter will overwrite it.
[Warning] File /quantized-yolov3-eic-at-0715.json is already existed, Exporter will overwrite it.
```

量化执行完成后，进入 config.json 文件中定义的"save_path"目录，可看到生成的 table.json（实际量化表名由 onnx 文件名以及上级目录构成）文件，进行图融合和后的 onnx 文件和精度分析结果文件 precision_accumulate_result.txt、precision_reset_result.txt。文档后续 table.json 即为量化信息表，不再特殊说明文件名称。config.json 文件中"save_path"参数为必配参数，如果不配置将报错。table.json 文件是模型编译的输入之一，将在模型编译章节详细介绍。

量化生成的 onnx 进行部分了图融合操作，在编译模型时可以使用原始 onnx 模型也可以使用量化后的 onnx 模型；使用 EsGoldenDataGen 工具时，需要使用原始 onnx 文件。

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

```
root@d76a884a0d56:~$ cd ${save_path}
root@d76a884a0d56:~$ ls

precision_accumulate_result.txt
precision_reset_result.txt
table.json
yolov3.onnx
```

6.4.2 量化模型精度分析

精度分析依赖于量化过程中生成的 precision_accumulate_result.txt 和 precision_reset_result.txt 文件。precision_accumulate_result.txt 统计的信息为量化模型和浮点数模型逐层累计的余弦相似度误差；precision_reset_result.txt 统计的信息为量化模型和浮点数模型每层均重置输入获得的余弦相似度误差。文件内部会记录余弦相似度信息。

详细内容如下：

```
root@d76a884a0d56:~$ cat precision_accumulate_result.txt
Conv_0     : 0.9973047614097595
Sigmoid_1  : 0.999542486667633
Mul_2      : 0.9864129543304443
Conv_3     : 0.9846434354782104
Sigmoid_4  : 0.9976542234420777
Mul_5      : 0.9705418705940246
```

首先查看 precision_accumulate_result.txt 中最后的 op-name（网络结构中最后一层 operation 名称）对应的 cos_dist(mean)是否大于 95%，如果满足大于阈值则视为误差在可接受范围内。如不满足需要查看另外一个文件即 precision_reset_result.txt。此时需要定位哪一层精度损失下降明显，然后在配置文件 config.json 中将此层替换成 int16（不要更换第一层 operation 和最后输出的 operation 的 datatype）。操作即在配置文件中找到 nodes_int16，在其中填写 node-name 字段。重新量化并开展精度分析，再排查量化精度误差。该过程结束后一般都可以获得较高的精度，如果精度还是低于阈值，则需要排查配置参数是否正确。更多的精度分析方法及精度提升方法，请参考 2.3 节。

6.4.3 GoldenData 生成

EsGoldenDataGen 用于生成参考数据，包含 input feature map 和 output feature map，然后将工具生成的数据提供给 EsSimulator 然后进行验证。

EsGoldenDataGen 基于 ONNXRuntime 框架运行浮点数模型然后进行量化，可以作为真实数据。EsSimulator 运行量化模型并将量化结果和 ONNXRuntime 的结果进行余弦相似度比较。目前余弦相似度设置的阈值是 95%，如果达到阈值要求视为量化精度损失基本满足需求。

注意此过程仅用于验证量化过程，提供的数据必须是 calibration 的标定数据，提供一张图片即可。

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

6.4.3.1 配置 yolov3 运行文件

完整配置文件可在 EsNNTools/sample/yolov3/essimulator 目录下获取，建议客户以实际情况配置 esgolden_data_gen 参数，详细可以参考 EsGoldenDataGen 工具介绍。

ESWIN Confidential

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

```
root@d76a884a0d56:~$ cat yolov3.json
{
    "model": {
        "model_path": "/yolov3/yolov3_sim_extract_416_notranspose_noreshape.onnx",
        "save_path": "/Model/yolov3/",
        "batchsize": 1
    },
    "dataset": {
        "data_root": "/img_list/",
        "transform_cfgs": [
      {
        "type": "ToRGB"
      },
      {
        "type": "LetterBox",
        "new_shape": [
          416,
          416
        ],
        "auto": false
      },
      {
        "type": "Normalize",
        "norm": true
      },
      {
        "type": "ToCHW"
      },
      {
        "type": "TransPrecision",
        "precision": "float32"
      }
        ]
    },
    "quant": {
        "input": {
            "format": {
                "images": [
                    1,
                    3,
                    416,
                    416,
                    1
                ]
            },
            " color_format_convert ": true,
            "scale": {
                "images": 0.007874015718698502
            },
            "data_type": 0,
        },
```

```
        "output": {
            "format": {
                "onnx::Shape_406": [1,255,52,52,1],
                "onnx::Shape_461": [1,255,26,26,1],
                "onnx::Reshape_516": [1,255,13,13,1]
            },
            "scale": {
                "onnx::Shape_406": 0.13676564395427704,
                "onnx::Shape_461": 0.12968941032886505,
                "onnx::Reshape_516": 0.12968522310256958
            },
          "data_type": 0
       }
    }
}
```

在 EsQuant 容器中运行 EsGoldenDataGen 命令生成 ifmap 和 ofmap。

```
root@d76a884a0d56:~$ python -m esquant.es_goldendata_gen.EsGoldenDataManager --config yolov3.json
```

在 yolov3.json 中定义的"save_path"路径下会生成对应的输入的 ifmap 和 ofmap，文件名称由 operation 名称和图片名称构成，中间由"_"字符连接，如下所示。

```
root@d76a884a0d56:~$ cd ${save_path}
root@d76a884a0d56:~$ ls

Conv_248_000000092416.bin    Conv_292_000000092416.bin
Conv_336_000000092416.bin    images_000000092416.bin
```

6.5 模型编译

6.5.1 使用 EsAAC 编译模型

- 准备量化文件 table.json

执行编译需要量化文件 table.json 和 ONNX 模型文件作为输入。在 6.3.1 中成功执行量化后会生成模型的量化文件 table.json，用户也可以在 EsNNTools/sample/yolov3/EsAAC 路径下获取预置的 table.json。

将 table.json 和 ONNX 模型文件拷贝到 EsAAC 和 EsSimulator 容器的挂载目录${workdir}下，在容器中执行 EsAAC 的编译命令。

- 运行 EsAAC 命令行

```
eswin@d76a884a0d56:~$ ./EsAAC --input-model ${workdir}/yolov3_sim_extract_416_notranspose_noreshape.onnx --quant-stats ${workdir}/table.json
```

运行时打印提示信息：

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

```
[EsAAC] Model is starting to compile ...
[EsAAC] Model parsing time: 1s:17ms:454us
[EsAAC] Model conversion time: 7ms:360us
[EsAAC] Model generation time: 11s:346ms:23us
[EsAAC] Model simulation running time: 12us
[EsAAC] Model compilation total time: 12s:465ms:287us
[EsAAC] Model compilation completed.
```

打印出 Mode compilation completed 表示运行成功，运行后在当前挂载目录下产生量化后的离线模型文件${workdir}/ yolov3_sim_extract_416_notranspose_noreshape.model。

6.5.2 使用 EsSimulator 仿真验证

基于 EsSimulator 做基本测试，需要指定 model、input、output 参数。将 6.4.3 得到的 ifmap 和 ofmap 文件拷贝到 EsAAC 和 EsSimulator 容器的挂载目录${workdir}下，执行下述命令：

```
eswin@d76a884a0d56:~$ ./EsSimulator --model=${workdir}/yolov3_sim_extract_416_notranspos
e_noreshape.model --input=${workdir}/ifmap.bin --output=${workdir}/ofmap0.bin,
${workdir}/ofmap1.bin,${workdir}/ofmap2.bin
```

这里需要注意：对于多输出模型输出的 ofmap 顺序可能和模型结构上面输出不一致，送入测试程序的时候会报错，这时 ofmap 需要指定顺序。EsACC 会在当前目录下产生一个 model_name.ofmap_order.txt 的文件来描述输出 ofmap 的顺序,文件格式如下所示(只有一个输出的模型不需要关注此文件)：

```
5   Conv_292-conv_biasadd
6   Conv_336-conv_biasadd
7   Conv_248-conv_biasadd
```

因此对于多输出的模型，在进行 EsSimulator 仿真验证的时候,需要把多个 ofmap 数据按照 model_name.ofmap_order.txt 所描述的顺序送给 EsSimulator 测试程序。

在 EsSimulator 运行完成后，会有相关打印提示模型运行成功与否，便于定位模型问题，以下为 yolov3 模型的测试示例打印。

```
EsSimulator version: 0.0.3(Mon Jul 22 14:36:18 2024 +0800).
yolov3_sim_extract_416_notranspose_noreshape.model test successful.
EsSimulator(version: Mon Jul 22 14:36:18 2024 +0800) finished.
```

当模型数据基于 float32 或 float16，可设置误差容忍度，模型进行数据比较时，当绝对误差和相对误差小于容忍度时，模型比较都为正确，配置容忍度示例如下：

```
eswin@d76a884a0d56:~$ ./EsSimulator --model=${workdir} yolov3_sim_extract_416_notransp
ose_noreshape.model --input=${workdir}/ifmap.bin --output=${workdir}/ofmap0.bin,
${workdir}/ofmap1.bin,${workdir}/ofmap2.bin --tolerance=0.008
```

6.6 模型推理

模型部署包含调用 NPU 相关 api 接口，以及配置后处理相关算子的 json 文件，最后运行后处理相关代码等步骤。

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

6.6.1 es_run_model 模型评测工具使用

es_run_model 工具主要是验证模型在硬件上的功能、精度并统计模型的性能。

本节演示 es_run_model 调试工具的使用。如下命令直接拷贝的可能存在格式问题，建议拷贝后格式化一下（手动去掉换行符）。示例中所使用到的模型是复合模型，模型可以由 tar 包解压并拷贝到测试目录，如/opt/eswin/data/npu/yolov3/。用户切换到 root 帐户后直接执行，也可将模型替换为 6.5.1 编译得到的离线模型。命令行中参数的详细含义，请参考 4.2.2 节。

- 示例 1：单纯测试模型运行速度

```
root@rockos-eswin:/#/opt/eswin/bin/es_run_model  -m /opt/eswin/data/npu/yolov3/ -r 1000
------------------------------------------------------------
avg = 11.6540 ms, fps = 85.8074 frames/s ----------------------------------------------------
-------------
```

这种情况下可以通过模型来确定输入数据的形状和精度，然后生成模型的输入数据, 从而对模型进行测试，用户不用指定输入文件夹。

- 示例 2：指定了输入、输出文件夹参数

这种模式下不需要指定 list 参数，es_run_model 会从输入目录中查找输入数据，并将推理结果保存在输出目录中，当没有指定输出目录时不会保存结果。

es_run_model 测试命令如下：

```
root@rockos-eswin:/# /opt/eswin/bin/es_run_model -m /opt/eswin/data/npu/yolov3/ -r 10 -i
/opt/eswin/data/npu/yolov3/input/preprocessed/0/ -o /opt/eswin/data/npu/yolov3/output
------------------------------------------------------------
avg = 11.6852 ms, fps = 85.5783 frames/s
------------------------------------------------------------
```

说明：上述-o 参数指定 output 的输出目录可能还不存在，在这里指定后会自动创建，也也可以在。

当指定了 verify 参数时，会从指定输出目录中查找是否有参考数据用来对比，如果找不到则报错。此时需要注意，对于多输出模型，输出文件夹的多个输出文件需要重命名以满足 ofmap 的顺序，这里基于 EsAAC 产生的 model_name.ofmap_order.txt 文件所描述的 ofmap 顺序，将多输出文件按顺序重命名，例如重命名为 ofmap0.bin、ofmap1.bin、ofmap2.bin 等，满足顺序后才能使得多输出模型的 verify 正确。

- 示例 3：指定了输入、输出文件夹并指定 list 参数

在指定输入和输出目录基础上增加了 list 参数，当输入目录下还有子目录的时候，我们需要通过 list 指定需要从输入目录的哪些子目录中查找输入文件。 这种模式下需要指定 list 参数，list 就是表示目录中有哪些子目录作为输入数据。

如下目录结构:

```
root@rockos-eswin:/# cat /opt/eswin/data/npu/yolov3/input/preprocessed/list.txt
0
1
```

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

es_run_model 测试命令如下：

```
root@rockos-eswin:/#  /opt/eswin/bin/es_run_model  -m  /opt/eswin/data/npu/yolov3/  -i
/opt/eswin/data/npu/yolov3/input/preprocessed/  -o  /opt/eswin/data/npu/yolov3/output  -l
/opt/eswin/data/npu/yolov3/input/preprocessed/list.txt
------------------------------------------------------------
avg = 11.8720 ms, fps = 84.2318 frames/s ----------------------------------------------------
-----------
```

当加上--save-perf 参数之后，会将性能测试数据保存到一个 json 文件中，内容如下：

```
root@rockos-eswin:/# cat model_perf_data.json
{
    "info": {
        "average_inference_time_ms":11.6080,
        "max_time_cost":11.6480,
        "min_time_cost":11.5680,
        "model_path": "yolov3_sim_extract_416_notranspose_noreshape.model",
        "repeat_times":1
    }
}
```

- 示例 4：使用动态 batch 测试模型运行速度

动态 batch 接口允许用户对图像进行批处理，每批次的数量可以动态调整。通过-M 参数设置测试模式，-b 参数设置 batch 大小。（注：1. 动态 batch 支持设置输入输出。2. 如果出现内存不足错误，需要调整环境变量 ES_NPU_MEMPOOL_CAPABILITY 到大于 128 的值）

```
root@rockos-eswin:/# /opt/eswin/bin/es_run_model  -m /opt/eswin/data/npu/yolov3/ -b 4 -r 1000
------------------------------------------------------------
  avg = 11.6896 ms, fps = 85.5463 frames/s
------------------------------------------------------------
```

动态 batch 使用异步方式测试的多个任务的平均值，故没有最大最小值。

6.6.2 模型推理

该部分主要介绍如何参考 NPU Runtime 提供的 sample_npu 源代码来开发用户自己的智能 App 以及如何执行 sample_npu 程序并输出推理结果等。

接下来将介绍 sample_npu 源代码调用 NPU Runtime 接口的简单流程。sample_npu 源代码简单的分为图像前处理，调用 NPU Runtime 接口的简单流程和输出后处理三部分。

- 图像前处理

配置前处理参数的 json 文件，参考 6.1.2 展示的 es_yolov3_pre_process.json 文件。示例代码使用 opencv 对图像进行前处理，详细代码可以参考 6.1.2 中的 sample 源码，此处不做详细介绍。

- NPU runtime 调用流程

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

```
# 设置设备号
ES_NPU_GetNumDevices(&deviceNum);
# 设置设备 ID
ES_NPU_SetDevice(deviceId);
# 加载模型文件
ES_NPU_LoadModelFromFile(&modelId, modelPaths[modelIdx].c_str());
# 创建 context
ES_NPU_CreateContext(&context, 0);
# 创建 stream
ES_NPU_CreateStream(&stream);
# 获取输入 tensor 数量
ES_NPU_GetNumInputTensors(modelId, &numInputs);
# 获取输入 tensor 描述
ES_NPU_GetInputTensorDesc(modelId, inputTensorId, &tensor);
# 根据输入 tensor 描述分配 tensor 内存
ES_SYS_MemAlloc(&task.inputFd[inputTensorId].memFd, SYS_CACHE_MODE_NOCACHE,
        "npu_sample", "mmz_nid_0_part_0", tensor.bufferSize);
# 拷贝前处理后的图像内容到 tensor 内存中
memcpy(inputBuf, int8Img.data, outDataLen);
# 同步提交任务
ES_NPU_SubmitSync(task, 1);
# 异步提交任务
ES_NPU_SubmitAsync(task, 1, stream);
# 查询任务
ES_NPU_ProcessReport(info->stream, info->queryMs);
```

此外，除了以上传统的单模型同步和异步任务推理外，NPU runtime API 还提供了动态 batch 接口和复合模型功能。

动态 batch 就是用户可以按照需求提交任意数量的 Task，并且提交的 Task 数量等可以灵活处理。

复合模型通过真正的 NPU 和 DSP 硬件并行运算提高了推理的性能。

动态 batch 接口允许用户按批处理任务，且批次大小不需要跟模型绑定。动态 batch 由 runtime 分配输入输出 tensor 内存，和复合模型一起使用方式如下。

```
# 调用 runtime 接口分配内存，第三个参数为批大小
ES_NPU_LoadCompositeModel (&modelId, modelPaths);
ES_NPU_AllocTaskMemory(modelId, stream, 1, &taskMem)
# 按批次提交任务
ES_NPU_SubmitFlexibleTask (task, 1, stream);
# 任务完成后调用 runtime 释放内存
ES_NPU_ReleaseTaskMemory(task.modelId, stream, 1, &taskMem);
```

- 输出后处理

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

示例代码采用 AcceleratorKit 模块提供的 DetectionOut 算子对模型推理结果进行后处理。首先，增加处理结果的类别标签文件，参考 6.1.2 展示的 es_yolov3_classes.txt 文件。同时，配置后处理算子信息的 es_yolov3_post_process.json 文件，该算子的 json 示例及注意事项如下：

```
{
    "config_version": "1.0.0",
    "kernel_name": "detection_out",
    "detection_net": "yolov3_u",
    "in_tensor_num": 3,
    "out_tensor_num": 2,
    "input_shape": "[1,255,13,13],[1,255,26,26],[1,255,52,52]",
    "output_shape": "[1,7,1,1024],[1,1,1,1]",
    "input_data_type": "S8,S8,S8",
    "output_data_type": "F32,I32",
    "anchors_num": 3,
    "anchor_scale_table": "[116,90, 156,198, 373,326], [30,61, 62,45, 59,119], [10,13, 16,30, 33,23]",
    "cls_num": 80,
    "img_h": 416,
    "img_w": 416,
    "input_scale": "[0.18310,0.14447,0.15155]",
    "nms_method": "hard_nms",
    "iou_method": "iou",
    "out_box_type": "xminyminxmaxymax",
    "coord_norm_flag": false,
    "max_boxes_per_class": 200,
    "max_boxes_per_batch": 500,
    "score_threshold": 0.3,
    "iou_threshold": 0.3,
    "soft_nms_sigma": 0.6,
    "effec_img_offset_y": 0,
    "effec_img_offset_x": 0
}
```

注意事项：

1、该配置 json 中的 detection_net 详细信息请参考《ENNP 开发者手册》第 3.4.4 节 ES_DET_NETWORK_E 枚举；

2、该配置 json 中的 input_shape 需要按照 tensor size 升序排列，同时 anchor_scale_table 和 input_scale 的配置顺序应该和 input_shape 保持一致；

3、该配置 json 中的 output_shape 为两个输出的 shape，两个输出分别是 outBoxInfo 和 outBoxNum。Shape 依次表示 NCHW 维度信息，第一个 tensor 的 shape 中“7”表示输出的每个检测框有 batch id、class id 等 7 个信息；“1024”表示本次推理最多输出 1024 个检测框(这个值可由用户自定义)；第二个 tensor 存放的数值表示本次推理的每个 batch 的实际 boxes 个数，假设本次进行 8 batch 的推理，则配置 shape 为“[8,1,1,1]”即可，该 tensor 存放的第 i 个数的数值即为第 i 张图的实际 boxes 个数。详细信息请参考《ENNP 开发者手册》第 3.3.12 节对 ES_AK_DSP_DetectionOut 接口的描述；

4、anchor_scale_table 的数值一般采用上述参数即可，如果网络经过微调或者自己重新训练，需要更换为自己的 anchor_scale，具体数值可以从未被裁剪的 onnx 中获取，例如下述模型的 anchor_scale

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

为：

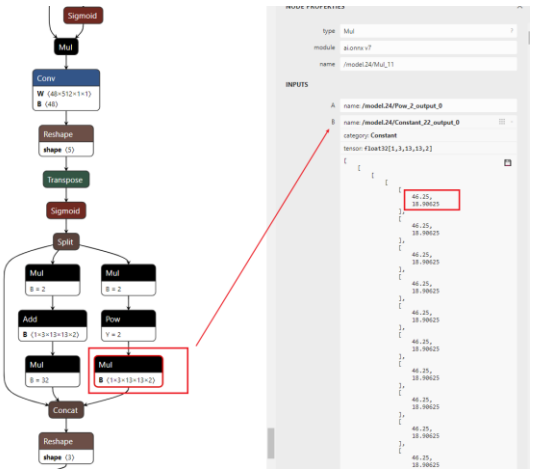

图 6-2 anchor_scale 获取位置示例

5、input_scale 的数值，需要通过 input_shape 的大小到 onnx 中获取对应的节点名称，再从该模型量化所得的 table.json 中获取对应节点的量化系数。

6、该配置 json 中的 nms_method 详细信息请参考《ENNP 开发者手册》第 3.4.5 节 ES_NMS_METHOD_E 枚举；

7、该配置 json 中的 iou_method 详细信息请参考《ENNP 开发者手册》第 3.4.6 节 ES_IOU_METHOD_E 枚举。

ESWIN Confidential

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

```
# 根据模型推理结果填充后处理算子输入
for (ES_S32 tensorId = 0; tensorId < task.outputFdNum; tensorId++) {
    NPU_TENSOR_S tensor;
    ES_TENSOR_S esTensor;

    ret = ES_NPU_GetOutputTensorDesc(task.modelId, tensorId, &tensor);
    if (ret != ES_SUCCESS) {
        SAMPLE_NPU_DEBUG(ret, "ES_NPU_GetOutputTensorDesc failed\n");
        return ret;
    }

    esTensor.pData = task.outputFd[tensorId];
    esTensor.dataType = postProcess->convertDataType(tensor.dataType);
    if (processType == EsPostProcess::CLASSIFY) {
        esTensor.shapeDim = 6;
    } else if (processType == EsPostProcess::DETECTION_OUT) {
        esTensor.shapeDim = 5;
    } else {
        esTensor.shapeDim = 5;
    }
    esTensor.shape[0] = tensor.dims.n;
    esTensor.shape[1] = tensor.dims.c;
    esTensor.shape[2] = tensor.dims.h;
    esTensor.shape[3] = tensor.dims.w;
    esTensor.shape[4] = 1;
    if (processType == EsPostProcess::CLASSIFY) {
        esTensor.shape[5] = esTensor.shape[4] * esTensor.shape[1];
    }
    postTensors.push_back(esTensor);
}
# 根据 配置文件调整 tensor 顺序
std::sort(postTensors.begin(), postTensors.end(), [](ES_TENSOR_S &t1, ES_TENSOR_S &t2) {
    return t1.pData.size < t2.pData.size;
});
# 准备输出 tensor
memset(&output, 0x00, sizeof(output));
memset(&outputCount, 0x00, sizeof(outputCount));
output.shapeDim = 5;
output.shape[0] = 1;
…
outputCount.shapeDim = 5;
outputCount.shape[0] = mDetectOutConfigs.inputShape[0][0];
outputCount.shape[1] = 1;
…
# 调用 DetectionOut 算子
ES_AK_DSP_DetectionOut(
            inTensors.data(), inTensors.size(), output,outputCount,
            (ES_DET_NETWORK_E)mDetectOutConfigs.detectNet, mDetectOutConfigs.anchorsNum,
            mDetectOutConfigs.anchorScale.data(), mDetectOutConfigs.imgH,
        mDetectOutConfigs.imgW, mDetectOutConfigs.clsNum,
            mDetectOutConfigs.inputScale.data(),
        (ES_NMS_METHOD_E)mDetectOutConfigs.nmsMethod,
            (ES_IOU_METHOD_E)mDetectOutConfigs.iouMethod,
        (ES_BOX_TYPE_E)mDetectOutConfigs.outBoxType,    (ES_BOOL)mDetectOutConfigs.coordNorm,
            mDetectOutConfigs.maxBoxesPerClass,
```

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

```
        mDetectOutConfigs.maxBoxesPerBatch, mDetectOutConfigs.scoreThreshold,
    mDetectOutConfigs.iouThreshold, mDetectOutConfigs.softNmsSigma,
        mDetectOutConfigs.effecImgOffsetX, mDetectOutConfigs.effecImgOffsetY);
```

完整代码可参考客户开发包 board sample 目录，编译及运行方式请参考 board sample/src/README.md 文件。

NPU Runtime 提供的 sample_npu 源代码在编译成可执行文件后，需要用户拷贝到 EIC7700 的开发板上运行。另外，EIC7700 开发板默认集成了 sample_npu 的可执行文件，下面是运行开发板自带 sample_npu 的执行命令和执行结果：

```
eswin@debian:~# /opt/eswin/bin/sample_npu -s 2 -m /opt/eswin/data/npu/yolov3/ -i /opt/eswin/data/npu/yolov3/input/pictures
/opt/eswin/data/npu/yolov3/input/pictures/input0/bus.jpg: box count(5)
class_id: 0.000000( ('person')), score: 0.843750, x_min: 26.156250, y_min: 153.250000, x_max: 123.125000, y_max: 348.000000
class_id: 0.000000( ('person')), score: 0.838379, x_min: 343.500000, y_min: 146.500000, x_max: 413.750000, y_max: 344.250000
class_id: 0.000000( ('person')), score: 0.827637, x_min: 118.812500, y_min: 160.000000, x_max: 174.625000, y_max: 325.500000
class_id: 5.000000( ('bus')), score: 0.791504, x_min: 8.250000, y_min: 89.187500, x_max: 407.750000, y_max: 283.750000
class_id: 0.000000( ('person')), score: 0.367432, x_min: 0.000000, y_min: 163.500000, x_max: 39.468750, y_max: 391.500000
/opt/eswin/data/npu/yolov3/input/pictures/input0/dog.jpg: box count(3)
class_id: 16.000000( ('dog')), score: 0.874023, x_min: 71.500000, y_min: 171.750000, x_max: 168.500000, y_max: 383.500000
class_id: 1.000000( ('bicycle')), score: 0.787598, x_min: 64.625000, y_min: 98.625000, x_max: 308.500000, y_max: 296.500000
class_id: 7.000000( ('truck')), score: 0.608887, x_min: 254.750000, y_min: 53.312500, x_max: 374.000000, y_max: 122.687500
```

上述打印每列（以逗号隔开）的类别依次为：分类信息，置信度，左上角坐标 X，左上角坐标 Y，右下角坐标 X 和右下角坐标 Y。

注：如果执行 sample 时加了-o 参数，sample_npu 还会将处理后带框的图像保存在-o 参数设置的路径下。

7. ENNP 双 die 模型推理架构

7.1 双 die 芯片简介

- 双 die 的芯片是在 soc 上有两套一模一样的硬件，从硬件性能上来说意味着性能翻倍。
- 双 die 的 soc 在都运行在同一个操作系统上，且硬件地址在系统中是线性化的，CPU 可以直接进行访问。

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

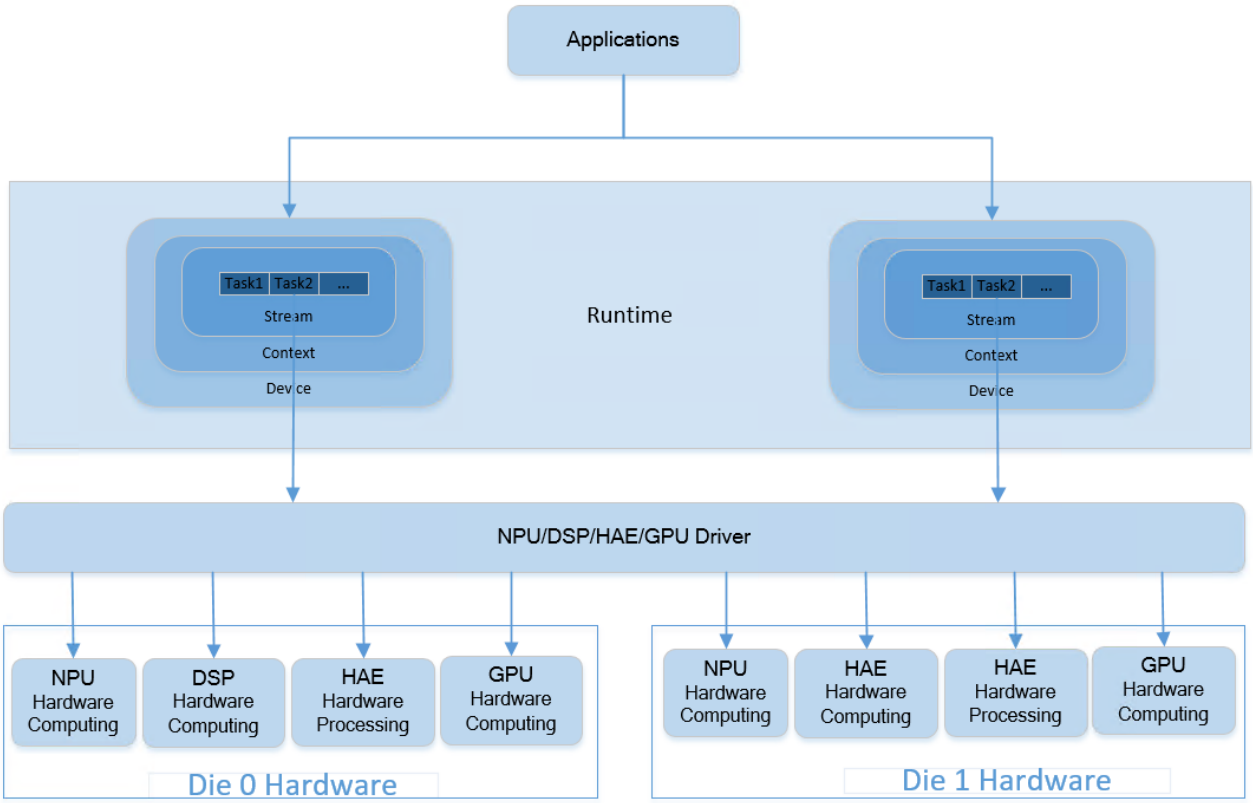

图 7-1 双 Die NPU 架构

7.2 应用场景

- 因为双 die 环境下多了一套硬件设备，因此性能整体会提高很多。可以使 NPU 能够推理更多的任务。
- 在双 DIE 环境下，由于跨 DIE 数据访问的性能可能受到影响，因此在系统运行时，应尽量减少跨 DIE 的数据传输，以提升整体运行效率。
- 对于性能跑分场景，批量的推理可以均匀分配到两个 die 上面分别运行，这样跑分结果性能会大大提高。

7.3 双 die 推理框架

- 为避免跨 die 之前的数据访问影响性能，NPU 在两个 die 的推理完全并行，中间不进行 die to die 的数据交互。
- NPU Runtime 为每个 pipeline 分配一个 NPU Context，分别对应不同的 NPU die。
- NPU 驱动根据当前双 die 的环境挂载两个设备节点，并分别创建两个设备数据结构用于分别存储各个 die 的数据。
- NPU 驱动分别于各自的 die 上硬件（E31）进行通信，任务下发和推理。

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

8. 自定义 dsp 算子开发示例

本章介绍用户如何基于 dsp 硬件，自定义模型所需的特殊算子的完整操作流程。完整的自定义 dsp 算子开发流程如下：

- 搭建 ESWIN 交叉编译环境；
- 搭建 Cadence Vision Q7 DSP 编译环境；
- 开发自定义 dsp 算子；
- 编译自定义开发的 dsp 算子；
- 板端运行 dsp 算子。

注意：更为详细的开发指南，请参考 dsp_sample/kernel_sample 中的 README.md 和 sample 示例。

8.1 Sample 文件说明

ESWIN 提供了 dsp_sample/kernel_sample 示例，用于帮助用户快速上手自定义算子开发。具体目录结构如下所示：

ESWIN Confidential

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

```
kernel_sample
├─── kernels
│    ├─── bev_pool
│    │    ├─── bev_pool.c
│    │    ├─── bev_pool.h
│    │    └─── CMakeLists.txt
│    ├─── build.sh
│    ├─── CMakeLists.txt
│    ├─── include
│    │    ├─── common
│    │    └─── xtensa
│    └─── user_add
│         ├─── CMakeLists.txt
│         ├─── user_add.c
│         └─── user_add.h
├─── README.md
└─── test
     ├─── build.sh
     ├─── CMakeLists.txt
     ├─── data
     │    ├─── input_0.bin
     │    └─── input_1.bin
     └─── src
          ├─── common.c
          ├─── common.h
          └─── dsp_kernel_test.c
```

主要包括以下内容：

README.md：详细的开发和编译指南；

kernels：dsp 侧代码，以 user_add 算子为例，提供了 dsp 侧算子开发的 sample 示例；

test/src：host 侧代码，以调用 user_add 算子为例，提供了自定义算子的调用示例；

tesr/data：以二进制形式，存放了 user_add 所需的输入数据，用户可以使用该数据完成自定义算子 sample 自测。

8.2 搭建 ESWIN 交叉编译环境

用户编译环境分 20.04 和 22.04 两种，区别在于生成交叉编译目录 rootfs，20.04 只能用 eswin 提供的 docker 编译生成 rootfs，22.04 可以直接在 x86 编译生成 rootfs。

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

以下都操作都基于 ubuntu 20.04。

8.2.1 获取 docker 和 tar.gz

从 ESWIN 获取 es_debian docker image 和 eswin-sdk-20241024.tar.gz（或最新版本）。

8.2.2 编译 eswin 交叉编译环境

- 参考《软件入门指南_CN_v1.3.docx》中“3.3 系统编译”章节，并执行到“make_all”命令完成。
- 在 Host 执行

```
cd eswin-sdk-20241024/
sudo chmod 777 -R EIDS200B/
cd EIDS200B/output
mkdir rootfs
sudo apt-get install e2fsprogs
e2fsck -f root-EIDS200B-20241025-020530.ext4
resize2fs root-EIDS200B-20241025-020530.ext4 12G
sudo mount root-EIDS200B-20241025-020530.ext4 rootfs/

sudo chroot rootfs/
```

- 在 rootfs 执行

```
root@ibudev12:/# cd home/eswin/
root@ibudev12:/# apt install -y gcc build-essential cmake
root@ibudev12:/# apt install es-sdk-dsp
root@ibudev12:/# apt install es-sdk-sample-dsp
root@ibudev12:/# apt install es-sdk-common
root@ibudev12:/# cd /usr/include/
root@ibudev12:/usr/include# ln -s riscv64-linux-gnu/bits bits
root@ibudev12:/usr/include# ln -s riscv64-linux-gnu/sys sys
root@ibudev12:/usr/include# ln -s riscv64-linux-gnu/gnu gnu
root@ibudev12:/usr/include# ln -s riscv64-linux-gnu/asm asm
root@ibudev12:/usr/include# cd /usr/lib
root@ibudev12:/usr/lib# cp riscv64-linux-gnu/* ./ -rf
root@ibudev12:/usr/lib# exit
```

8.2.3 获取 riscv 交叉编译工具

```
wget  https://github.com/riscv-collab/riscv-gnu-toolchain/releases/download/2024.04.12/riscv64-glibc-ubuntu-20.04-gcc-nightly-2024.04.12-nightly.tar.gz
sudo tar -xvf riscv64-glibc-ubuntu-20.04-gcc-nightly-2024.04.12-nightly.tar.gz -C /opt
sudo cp /opt/riscv/sysroot/usr/lib/crt1.o eswin-sdk-20241024/EIDS200B/output/rootfs/usr/lib
```

注意：上述的 opt 目录，需要在后续的 host 编译环节和脚本保持一致 。

8.3 Cadence Vision Q7 DSP 编译环境搭建

EIC7700 集成了 4 个 DSP 核，使用的是 Cadence 公司的 Vision Q7 DSP IP。Cadence DSP 开发环

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

境安装可参考 Cadence 提供的官方文档《dev_tools_install_guide.pdf》。

注意：DSP 开发依赖 Cadence 提供的集成开发环境，需要正确设置 license 之后才可以使用 Xtensa Develop Tools，EIC7700 不提供 Cadence license，需要用户单独购买。

8.4 自定义算子开发

8.4.1 host 侧准备

用户需要在 host 完成 op desc 生成、inputs 和 outputs 的内存申请和数据加载，其余步骤一般无需修改。

- op desc 填充方法参考 test/src/dsp_kernel_test.c 的 prepare_add_op_desc() 函数实现；
- inputs 的内存申请和数据加载参考 test/src/dsp_kernel_test.c 的 loadInputsData() 函数实现；
- outputs 的内存申请参考 test/src/dsp_kernel_test.c 的 prepareOutputBuffer() 函数实现。

8.4.2 dsp 侧算子开发

本手册简述 dsp 侧算子开发的常用接口，更为详细的开发指南和说明，请参考 dsp_sample/kernel_sample 中的 README.md 和提供的 sample 示例。

- dsp 侧算子函数接口

在 dsp 侧，共有 prepare、eval、get_prepare、get_eval 这 4 个函数，其函数名、参数均为固定接口，不能被修改。同时 prepare、get_prepare、get_eval 这 3 个函数的内容也不能被修改。用户只需要在 eval 函数中完成自己算子的实现代码即可。

- 获取片上内存起始地址及其大小

```
char *dram0_base_ptr = (char *)((ES_U32)get_available_dram0_base());
ES_U32 dram0_size = get_available_dram0_size();
```

注意：

1、 在 dsp 开发时，不建议直接对 ddr 进行访存操作，可能会因为 cache 导致数据不一致，另外大量访存 ddr 也会极大拖慢算子的计算性能。建议使用 dma 将数据拷贝到 dram 上进行操作；

2、每个 dsp 均有两个片上内存： dram0 和 dram1，两个 dram 的原始大小均为 128k。由于固件占用了部分空间，目前 dram0 起始地址为 0x28105000，可用空间为 108k；dram1 起始地址为 0x28120000，可用空间为 120k。

- 发起 dma 拷贝

```
idma_copy_3d_desc64(CHANNEL, &dst_ptr, &src_ptr, IDMA_DISABLE_INT, row_sz, nrows, ntiles,
                    src_row_pitch, dst_row_pitch, src_tile_pitch, dst_tile_pitch);
```

- 等待 dma 拷贝结束

```
while (idma_hw_num_outstanding(CHANNEL) > 0) {
}
```

- 日志打印

在 dsp 侧，请使用 es_printf 函数进行日志打印。

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

```
es_printf("this is eval of add op\n");
```

注意：

1、请登录串口，查看 dsp 侧的打印日志，例如 (下面命令仅供参考，具体命令请根据自身开发板连接方式进行调整)：

```
ssh 192.168.1.100 -l hostname
输入密码：password
sudo hostname -D /dev/ttyUSB3
```

登录串口后，再执行可执行程序调用算子，即可在串口获取到 es_printf 打印的内容。

2、在 dsp 侧使用 es_printf 打印日志，会对算子性能造成较大影响。根据打印的日志长度，大致会导致耗时增加数十~数百微妙。

- dsp 性能优化

1、将数据从 ddr 使用 dma 拷贝到 dram 进行计算，避免大量访问 ddr；

2、建议优先使用 int8 数据类型，其次是 int16 和 f16 数据类型，尽可能不要使用 float32 或者 double 数据类型；

3、使用 dsp 提供的 SIMD 指令，优化计算速度；

4、使用时间戳，统计每部分耗时，对性能瓶颈部分进行针对性优化。

8.5 算子编译

参考 dsp_sample/kernel_sample 中的 README.md，在算子编译前，根据实际情况更改“dsp_sample/kernel_sample/kernels/build.sh”函数 “xtensa_env_setup”中的 “xtensa 配置”。

注意：由于 XTENSA_CORE 和 XTENSA_TOOLS_VERSION 可能存在差异，如编译遇到问题请及时沟通解决。

- 编译 host 侧的测试程序

```
cd /path/to/your/dsp_sample/kernel_sample/test
./build.sh  /path/to/your/eswin-sdk-20241024/EIDS200B/output/rootfs  /path/to/your/kernel_sample/kernels/user_add
```

- 编译 dsp 侧的自定义算子

```
cd /path/to/your/dsp_sample/kernel_sample/kernels/
./build.sh /path/to/your/eswin-sdk-20241024/EIDS200B/output/rootfs
```

8.6 自定义算子测试

- 将编译好的算子 es_dsp_kernel_add.pkg.lib 拷贝到开发板/lib/firmware/dsp_kernels/；
- 将测试程序 dsp_kernel_test 和测试文件 input_0.bin, input_1.bin 拷贝到开发板任意位置，如：/opt/eswin/；
- 执行测试命令：

```
cd /opt/eswin/
./dsp_kernel_test -c 0 -i input_0.bin input_1.bin
```

Eswin Confidential

- 正确性验证：

执行结束后，会在 input_0.bin 所在目录，生成 output.bin。将其拷出到 Linux 下，使用：

```
md5sum output.bin
```

查看其 md5 码，如果其值为：4aae41d770845ae89f0cf33be15d0a77，则表示结果正确。

9. 附录

表 9-1 DSP 支持算子列表

算子名称	算子功能
Add	支持对两个输入进行加法操作
ArgMax	对输入求取最大值及其索引
AveragePool	对 kernel 范围内的数据求均值
BatchNormalization	对输入进行 BatchNormalization 操作
CropAndResize	对输入图片中提取特定区域进行缩放
Concat	对多个输入在指定轴上进行拼接
Conv	对输入进行卷积计算
Cos	计算输入的 Cos 数值
ConvTranspose	对输入进行反卷积计算
DepthwiseConv	对输入进行深度卷积计算
Focus	支持 yolo 检测网络中的 Focus 结构
Gather	从输入数据中收集 indices 对应的数据
Gelu	计算输入的 Gelu 激活函数输出
Gemm	对输入数据做矩阵乘
GlobalAveragePool	对输入数据每个通道分别求均值
HardSigmoid	计算输入的 HardSigmoid 激活函数输出
HardSwish	计算输入的 HardSwish 激活函数输出
InstanceNormalization	对输入进行 InstanceNormalization 操作
LayerNormalization	对输入进行 LayerNormalization 操作
LeakyRelu	计算输入的 LeakyRelu 激活函数输出

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。

Log	计算输入的 Log 数值
MatMul	对两个数据矩阵做矩阵乘
MaxPool	对 kernel 范围内的数据求最大值
Mish	计算输入的 Mish 激活函数输出
NonMaxSuppression	对检测网络的 boxes 进行 NMS 操作
Pad	对输入在不同 dims 上进行 Pad 操作
PRelu	计算输入的 PRelu 激活函数输出
ReduceMax	对输入在指定维度上求取最大值
ReduceSum	对输入在指定维度上求取累加和
Relu	计算输入的 Relu 激活函数输出
Relu6	计算输入的 Relu6 激活函数输出
Resize	对输入的张量数据进行缩放
RoiAlign	对输入的感兴趣区域进行双线性插值
MaxRoiPool	执行 MaxRoiPool 池化操作
ROPE	在 LLM 模型 attention 模块前插入位置信息
Sigmoid	计算输入的 Sigmoid 激活函数输出
Silu	计算输入的 Silu 激活函数输出
Sin	计算输入的 Sin 数值
Slice	将单个输入根据 attribute 信息拆分为多个输出
Softmax	计算输入的 Softmax 作为输出
TopK	对输入在指定维度上求取前 K 个较大值及其索引
Transpose	根据 perm 参数，对输入做转置操作

ESWIN Confidential

Copyright@北京奕斯伟计算技术股份有限公司所有。未经许可，严禁复制和分发。