



深度学习模型部署流程

汇报人：张文剑

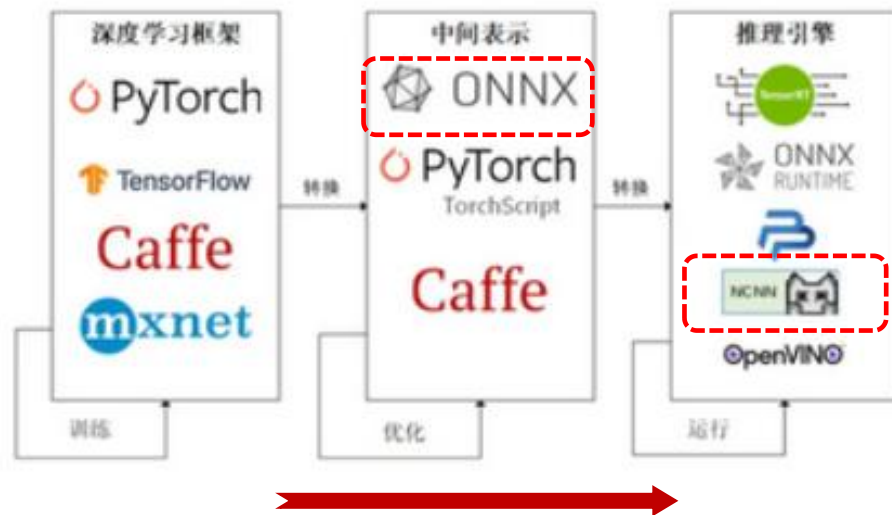


模型部署

简单来说，深度学习模型的部署就是指训练好的模型在特定环境中运行的过程。相比于软件工程定义中应用程序的部署，深度学习模型的部署会有以下一些难点：

- 模型运行所需的环境难以配置。深度学习模型通常是由PyTorch、TensorFlow框架编写，由于框架规模和依赖环境的限制，这些**框架本身并不适合在手机、开发板等边缘计算环境中安装**；
- 深度学习模型的网络**结构一般较为庞大**，需要大量的算力才能满足一些实时运行的需求，模型的运行效率要求较高；

在工业界中模型部署有了一条**流行的流水线**：



ONNX计算图

ONNX (Open Neural Network Exchange) 2017年由微软和Facebook联合推出的开源项目，旨在为深度学习模型提供一个**跨平台、跨框架的标准化表示**。它能够标准地描述深度学习**模型的计算图**，允许AI开发人员将训练好的模型从一个深度学习框架迁移到另一个框架，保证模型的跨框架互操作性。

因此，ONNX 被当成了深度学习框架到推理引擎的桥梁，为高效地进行跨框架的模型**迁移与部署**提供了一个解决方案。

```
torch.onnx.export(  
    model,                # 模型实例  
    img,                  # 模型输入数据，可以是一个元组或列表  
    "/data/zhangwj/lmf-output/onnx/model.onnx", # 输出文件路径  
    input_names=["image"], # 输入节点名称，  
    output_names=["output"], # 输出节点名称  
    opset_version=11,      # 使用 ONNX opset 11  
    verbose=True,          # 打印详细的转换信息  
    # export_params=True  
)
```

构建ONNX图

定义输入输出向量

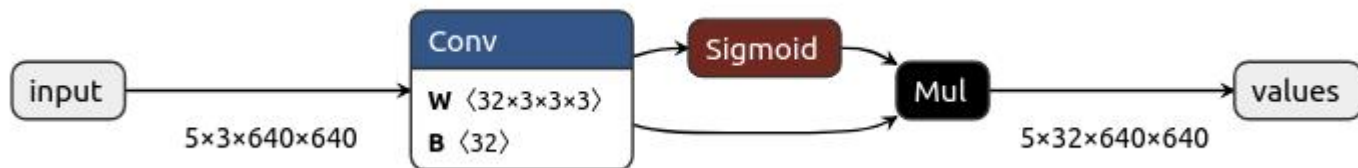
定义节点

构建图

构建和检测模型

保存模型

Netron工具可视化计算图:



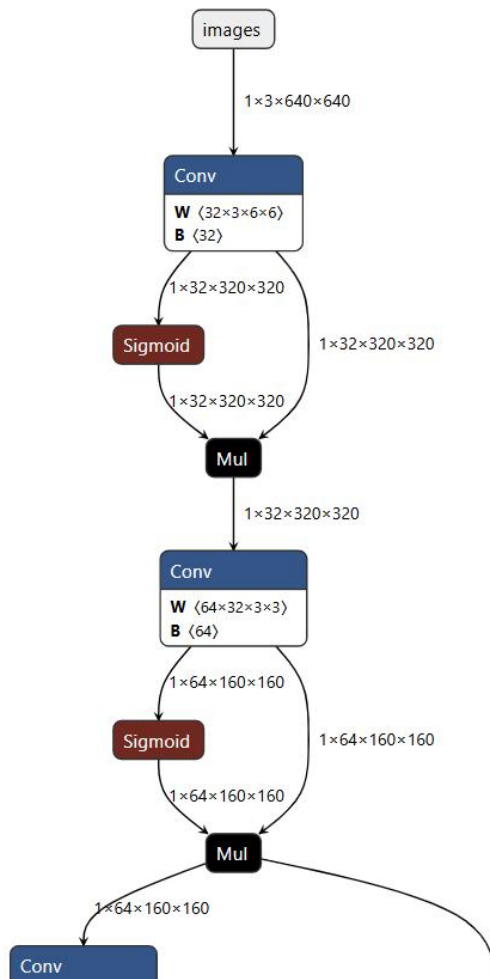
```

# Create the input tensor
input_tensor = helper.make_tensor_value_info('input', TensorProto.FLOAT, [5, 3, 640, 640])
# Create the weight and bias tensors for the convolution layer
weights_data = np.random.randn(*conv_weight_shape).astype(np.float32)
bias_data = np.random.randn(*conv_bias_shape).astype(np.float32)
weights_tensor = helper.make_tensor('weights', input_dtype, [32, 3, 3, 3], weights_data)
bias_tensor = helper.make_tensor('bias', input_dtype, [32], bias_data)

# Create the convolution node
conv_node = helper.make_node(
    'Conv', # onnx算子
    inputs=['input', 'weights', 'bias'], # 输入向量名称
    outputs=['conv_output'], # 输出向量名称
    kernel_shape=[3, 3],
    pads=[1, 1, 1, 1],
    strides=[1, 1]
)
# Create the sigmoid node
sigmoid_node = helper.make_node(
    'Sigmoid',
    inputs=['conv_output'],
    outputs=['sigmoid_output']
)
# Create the multiplication node
mul_node = helper.make_node(
    'Mul',
    inputs=['conv_output', 'sigmoid_output'],
    outputs=['values']
)
# Create the graph
graph_def = helper.make_graph(
    [conv_node, sigmoid_node, mul_node], # 节点名称
    'example_model', # 计算图名称
    [input_tensor], # 输入向量
    ['values'],
    initializer=[weights_tensor, bias_tensor]
)
# Create the model
model_onnx = helper.make_model(graph_def, producer_name='onnx-example')
onnx.checker.check_model(model_onnx) # 检测模型的准确性
# Save the model to a file
onnx.save(model_onnx, 'example.onnx')
  
```

ONNX计算图-yolov5s

模型结构图



MODEL PROPERTIES

format ONNX v7

producer pytorch 1.10

version 0

imports ai.onnx v12

graph torch-jit-export

METADATA

stride 32

names

['person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'train', 'truck', 'boat', 'traffic light', 'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow', 'elephant', 'bear', 'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie', 'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball', 'kite', 'baseball bat', 'baseball glove', 'skateboard', 'surfboard', 'tennis racket', 'bottle', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple', 'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', 'cake', 'chair', 'couch', 'potted plant', 'bed', 'dining table', 'toilet', 'tv', 'laptop', 'mouse', 'remote', 'keyboard', 'cell phone', 'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'book', 'clock', 'vase', 'scissors', 'teddy bear', 'hair drier', 'toothbrush']

INPUTS

images name: images
tensor: float32[1, 3, 640, 640]

OUTPUTS

output name: output
tensor: float32[1, 3, 80, 80, 85]

353 name: 353
tensor: float32[1, 3, 40, 40, 85]

367 name: 367
tensor: float32[1, 3, 20, 20, 85]

转换ONNX文件

```
# Upsample lr feat to hr feat
inp = F.grid_sample(
    feat, coords.flip(-1).unsqueeze(1),
    mode='nearest', align_corners=False)[: , :, 0, :] \
    .permute(0, 2, 1)
```

```
torch.onnx.export(
File "/home/zhangwenjian/miniconda3/envs/test/lib/python3.8/site-packages/torch/onnx/utils.py", line 506, in export
  _export(
File "/home/zhangwenjian/miniconda3/envs/test/lib/python3.8/site-packages/torch/onnx/utils.py", line 1548, in _export
    graph, params_dict, torch_out = _model_to_graph(
File "/home/zhangwenjian/miniconda3/envs/test/lib/python3.8/site-packages/torch/onnx/utils.py", line 1117, in _model_to_graph
    graph = _optimize_graph(
File "/home/zhangwenjian/miniconda3/envs/test/lib/python3.8/site-packages/torch/onnx/utils.py", line 665, in _optimize_graph
    graph = _C._jit_pass_onnx(graph, operator_export_type)
File "/home/zhangwenjian/miniconda3/envs/test/lib/python3.8/site-packages/torch/onnx/utils.py", line 1901, in _run_symbolic_function
    raise errors.UnsupportedOperatorError(
torch.onnx.errors.UnsupportedOperatorError: Exporting the operator 'aten::grid_sampler' to ONNX opset version 11 is not supported. Support for this operator was added in version 16, try exporting with this version.
```

调用export()方法时转 ONNX 时最容易出现的问题就是**算子不兼容**了，转换普通的 torch.nn.Module 模型时，PyTorch 会用跟踪法执行前向推理，把遇到的算子整合成计算图，同时会把遇到的每个算子翻译成 ONNX 中定义的算子。

- 该算子可以一对一地翻译成一个 ONNX 算子；
- 该算子在 ONNX 中没有直接对应的算子，会翻译成一至多个 ONNX 算子；
- 该算子没有定义翻译成 ONNX 的规则，报错。

主要特点

处理器内核 8G

- 四核 ARM Cortex A55@1.4GHz
 - 32KB I-Cache, 32KB D-Cache /512KB L3 cache
 - 支持Neon加速, 集成FPU处理单元
- 内置 32bit MCU@500MHz
 - 32KB I-Cache, 32KB D-Cache /64KB TCM
- 图像分析加速引擎, 高达 10.4Tops@INT8 算力
 - 双内核异构引擎
 - 引擎 1 支持 4.8Tops 算力, 支持 INT4/INT8/FP16
 - 引擎 2 支持 5.6Tops 算力, 支持 INT8/INT16

数字图像处理 (ISP)

- ISP 支持分时复用处理多路 sensor 输入视频
- 支持 3A (AE/AWB/AF) 功能, 3A 的控制用户可调节
- 支持去固定模式噪声 (FPN)
- 支持坏点校正、镜头阴影校正;
- 最高支持三帧 WDR 及 Advanced Local Tone Mapping
- 支持多级 3D 去噪、图像边缘增强、去雾、动态对比度增强等处理功能
- 支持 3D-LUT 色彩调节
- 支持镜头畸变校正, 支持鱼眼矫正
- 支持 6-DoF 数字防抖及 Rolling-Shutter 校正
- 支持图像 Mirror、Flip、90 度/270 度旋转
- 提供 PC 端 ISP 调节工具

内置AI ISP功能

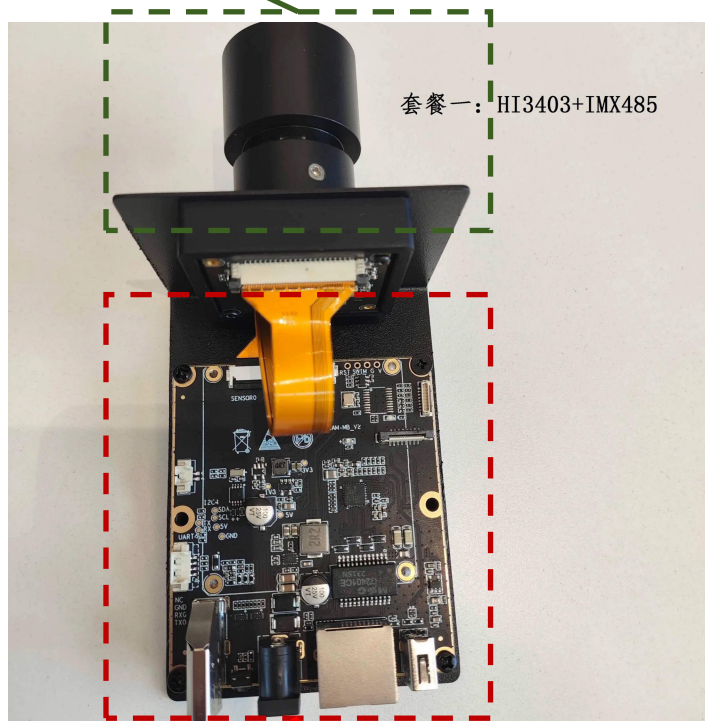


AI ISP卓越的全天候图像效果

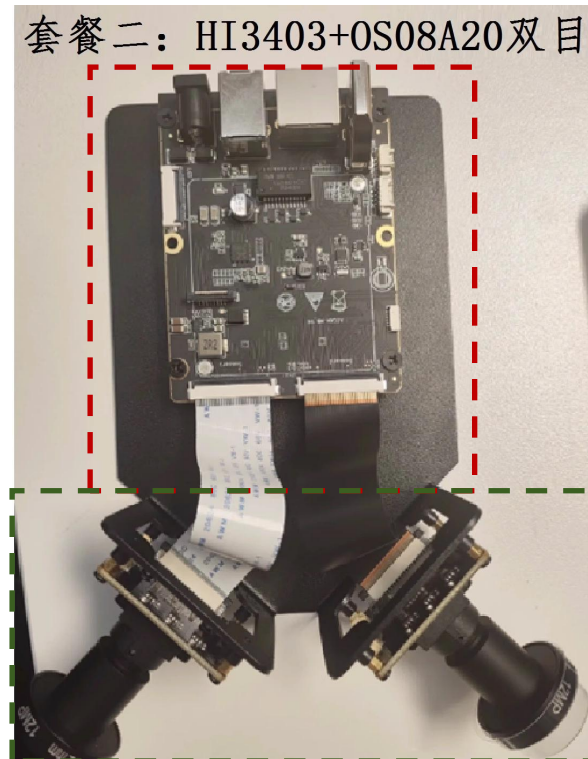
- 超感光降噪, 12dB信噪比提升
- 超级宽动态, 基于场景感知的极限收光; 快速准确跟踪复杂光线下的成像效果

- Hi3403V100是一颗面向监控市场推出的专业 Ultra-HD Smart IP Camera SOC, 该芯片最高支持四路 sensor输入, 支持最高4K60的ISP图像处理能力;
- Hi3403V100集成了高效的神经网络推理单元, 最高10TOPS INT8, 并支持业界主流的神经网络框架。

配件: Sensors



套餐二: HI3403+OS08A20双目



SoC主体

目前Hi3403V100网上经销商售价大概是4000

SoC激活的主要步骤

1. 安装交叉编译工具
2. 编译SDK
3. 烧录固件



om文件

“.om”文件是Huawei Ascend AI 处理器（hisilicon）的模型文件格式，全称为“**offline model**”。Huawei Ascend AI是一种专门用于人工智能计算的芯片，旨在提高 AI 计算效率和性能。因此，om文件用于在 Ascend AI 推理引擎上部署和运行的模型文件。

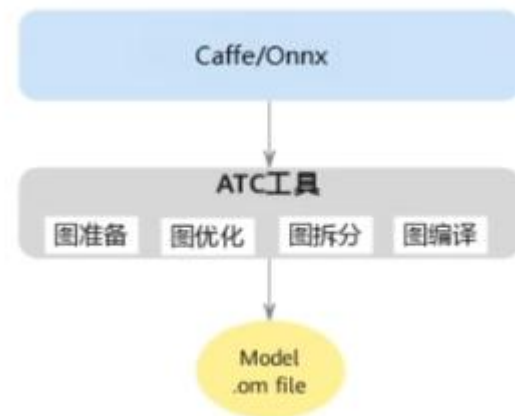
为了适配这种芯片，Huawei 开发了一种名为 MindSpore 的 AI 框架，可以将训练好的模型转换为 “.om” 格式的模型文件，并在 Ascend AI 处理器上进行高效的推理计算。

转换方式：SoC开发板中SDK文档中带的ATC**模型转换工具**

名称	修改日期
 Ascend-cann-toolkit_5.20.t6.2.b060_linux-x86_64.run	2022/12/26 11:43
 CANN-amct-5.20.t6.2.b060-ubuntu18.04.x86_64.tar.gz	2022/12/26 11:43

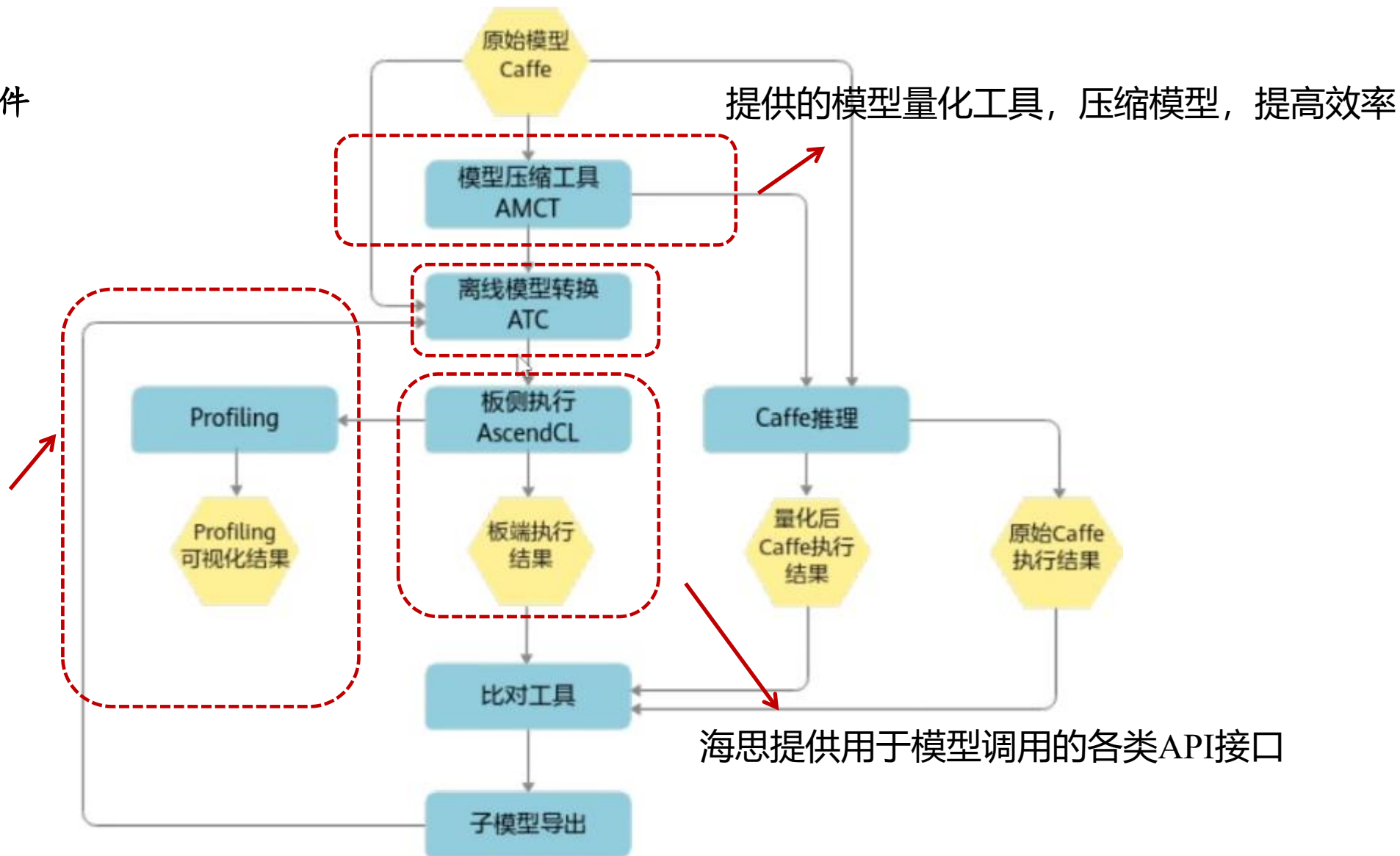
每一种型号的开发板自带的ATC转换工具不同，适应不同开发板的特点。

图1-1 ATC 工具功能架构



Caffe模型转换om文件

Profiling 性能分析工具用于采集和分析运行在 SOC 上的推理业务（应用或算子）各个运行阶段的关键性能指标，用户可根据输出的性能数据针对关键性能瓶颈做出优化以实现产品的极致性能。



om模型转换

```
atc --model=lmliif.onnx --framework=5 --output=lmliif --soc_version="OPTG"
--output_type=FP32 --insert_op_conf=./op.cfg
```

5 算子规格说明	82
5.1 Caffe框架算子规格	82
5.2 Onnx框架算子规格	102

```
(atc) wenjian@qihaoran-server:~/Ascend/ascend-toolkit/svp_latest/x86_64-linux$ atc --model=/data/zhangwj/lmf-output/LMF0113-v11.onnx --framework=5 --output=LMF0113-v11 --soc_version="SS928V100" --output_type=FP32
Mapper Version 1.0.0.0_B010 (PICO_1.0) 221230032147ce09523(CPU) (INST_2.0.9)
framework:5
model:/data/zhangwj/lmf-output/LMF0113-v11.onnx
output:LMF0113-v11
output_type:FP32
soc_version:SS928V100
begin net parsing....
[ERROR][LoadLibrary][46] ERROR: dlopen libpath[libsfp_custom.so] failed
[ERROR][ParseLayer][179] Unknown Op[ScatterND]. Known Ops[Abs, AbsVal, Acos, Acosh, Add, And, ArgMax, ArgMin, Asin, Asinh, Atan, Atanh, AveragePool, BILSTM, BIRNN, BNLL, BatchNorm, BatchNormalization, Bias, BinaryMath, BitShift, CReLU, Ceil, Celu, Clip, Cmp, Compress, Concat, CondEnd, ConstantOfShape, ConvTranspose, Convolution, Cos, Cosh, Crop, CumSum, Custom, DecBBBox, Deconvolution, DepthToSpace, DepthwiseConv, Det, DetectionOutput, Div, Dropout, ELU, Einsum, ElseBegin, Eltwise, Equal, Exp, Expand, Extract, ExtractSlice, EyeLike, Filter, Flatten, Floor, GRU, Gather, GatherElements, Gemm, GlobalAveragePool, GlobalLpPool, GlobalMaxPool, Greater, GreaterOrEqual, HardSigmoid, Hardmax, Hswish, Identity, If, InnerProduct, Input, InstanceNormalization, Interp, LRN, LSTM, LeakyRelu, Less, LessOrEqual, Log, Logical, Loop, LpNorm, LpPool, MVN, MatMul, Max, MaxPool, MaxRoiPool, MaxUnpool, Mean, MeanVarianceNormalization, Min, Mish, Mod, Mul, NMS, Neg, NonMaxSuppression, NonZero, Nop, Normalize, Not, OneHot, Or, PReLU, PRelu, PSROI, PSROIPooling, Pad, Parameter, PassThrough, Passthrough, Permute, Pooling, PoolingMask, Power, QLinearMatMul, RNN, ROIpooling, RReLU, Range, ReLU, ReLU6, Reciprocal, ReduceL1, ReduceL2, ReduceLogSum, ReduceLogSumExp, ReduceMax, ReduceMean, ReduceMin, ReduceProd, ReduceSum, ReduceSumSquare, Reduction, Reorg, Reshape, Resize, Reverse, Round, SPP, Scale, Scan, Scatter, ScatterElements, Selu, Shape, Shrink, ShuffleChannel, Sigmoid, Sign, Silence, Sin, Sinh, Size, Slice, Softmax, Softplus, Softsign, Sort, SpaceToDepth, Split, Sqrt, Squeeze, Sub, Sum, Tan, TanH, Tanh, ThenEnd, Threshold, ThresholdedRelu, Tile, TopK, Transpose, Unique, Unsqueeze, Upsample, Where, Xor], ExtendedOps[].
```

难点：涉及未知算子的问题，开发文档中提供了算子自定义流程，过程复杂且耗时。

npu

└─ Makefile

└─ sample_npu_main.c

└─ sample_svp_npu

└─ sample_npu_model.c

└─ sample_npu_process.c

└─ include

└─ sample_npu_model.h

└─ sample_npu_process.h

└─ data

└─ model

└─ mobilenet_v3_dynamic_batch.om

└─ resnet50.om

└─ image

└─ yuv_sp420_224_224.sp420

└─ caffe_model

└─ resnet50.prototxt

```

/* function : show the sample of npu resnet50 */
hi_void sample_svp_npu_acl_resnet50(hi_void)
{
    hi_void *data_buf = HI_NULL;
    size_t buf_size;
    hi_s32 ret;

    const char *om_model_path = "./data/model/resnet50.om";
    ret = sample_svp_npu_acl_prepare_init(om_model_path);
    if (ret != HI_SUCCESS) {
        return;
    }

    ret = sample_svp_npu_load_model(om_model_path, 0, HI_FALSE);
    if (ret != HI_SUCCESS) {
        goto acl_process_end0;
    }

    ret = sample_svp_npu_dataset_prepare_init(0);
    if (ret != HI_SUCCESS) {
        goto acl_process_end0;
    }

    ret = sample_svp_npu_get_input_data(&data_buf, &buf_size, 0);
    if (ret != HI_SUCCESS) {
        sample_svp_trace_err("execute create input fail.\n");
        goto acl_process_end1;
    }

    ret = sample_svp_npu_create_input_databuf(data_buf, buf_size, 0);
    if (ret != HI_SUCCESS) {
        sample_svp_trace_err("memcpy_s device buffer fail.\n");
        goto acl_process_end2;
    }

    ret = sample_npu_model_execute(0);
    if (ret != HI_SUCCESS) {
        sample_svp_trace_err("execute inference fail.\n");
        goto acl_process_end3;
    }

    sample_npu_output_model_result(0);

acl_process_end3:
    sample_svp_npu_destroy_input_databuf(1);
acl_process_end2:
    sample_svp_npu_release_input_data(&data_buf, &buf_size, 1);
acl_process_end1:
    sample_svp_npu_dataset_prepare_exit(1);
acl_process_end0:
    sample_svp_npu_acl_prepare_exit(1);
}

```

板端Resnet50 Sample模型推理

- 模型初始化
- 模型加载
- 数据初始化
- 数据加载
- 模型推理
- 推理结果后处理

V:\0.158.1.234\mm\code\ssdk\ssdk28\SSDK_V2.0.22\sample\c++\mm\sample\sample\sample

5.海思3403开发板检测符法移植(板端环境搭建+NPUSample运行)

网络 > 10.168.1.234 > mm > code > sdk_ss28 > SSDK_V2.0.22 > sample > dss_mux > npp > sample > svp

名称	修改日期	类型	大小
delete	2022/12/23 12:00	文件夹	
nmmlite	2022/12/23 12:00	文件夹	
sample_svp_npu	2024/5/16 0:11	文件夹	
Makefile	2022/12/23 12:00	文件	1 KB
sample_npu_main	2024/5/16 0:11	文件	3,981 KB
sample_npu_main.h	2022/12/23 12:00	C 源文件	4 KB
sample_npu_main.o	2021/3/16 0:11	0 文件	20 KB

显示被隐藏的文件。

今天

余文强的共享桌面

01:40 / 05:15

1080P 高清

设置

1.5x

静音

6.19

2024/5/15


主要特点

Rk3588系列是瑞芯微电子22年发布的一款旗舰芯片，该系列主要包括两款型号：

RK3588 EVB及RK3588S EVB。

- RK3588 EVB：面向ARM PC、NVR、服务器、IPC、大屏显示设备等**AIoT行业类应用产品**；
- RK3588S EVB：面向高端平板、AR/VR、个人移动互联网设备等**消费类电子产品**；
- 8nm先进制程，8核64位架构，高性能，低功耗
- ARM Mali-G610 MC4 GPU, 专用2D图形加速模块
- 6TOPs NPU，赋能各类AI场景
- 8K 视频编解码，8K显示输出
- 内置多种显示接口，支持多屏异显
- 超强影像处理能力，48MP ISP, 支持多摄像头输入
- 丰富的高速接口(PCIe, TYPE-C, SATA, 千兆以太网)，易于扩展
- Android 和Linux OS



 **Radxa**
@RadxaComputer

RK3688 is unveiled, ARM V9.3/128bit DDR/UFS 4/16T NPU, what feature do you expect on #Radxa ROCK 6 SBC?

翻译帖子

阴阳结合，持续新品，助力AIOT应用

下一代旗舰处理器RK3688

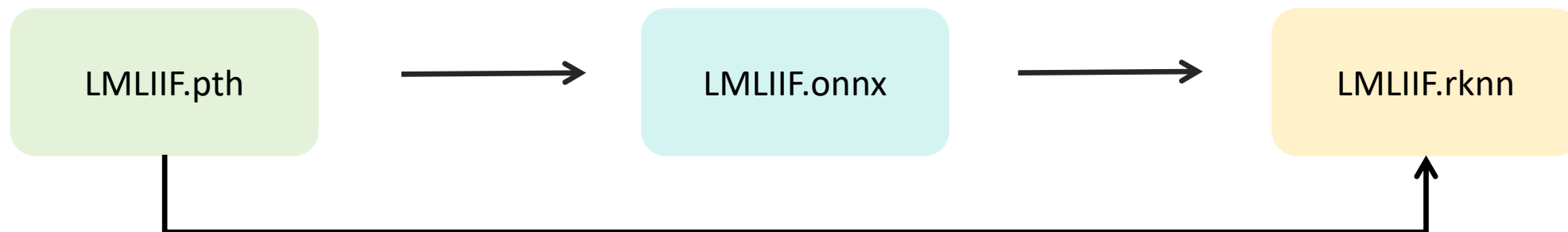
- 先进制程
- CPU ~ 250KDMIPS (Cortex-A7XX 全大核, ARMV9.3指令集)
- GPU > 1TGFLOPS
- NPU ~ 16TOPS
- 128-bit LPDDR4/4x5, UFS 4.0

新一代高性价比处理器RK35xx

- CPU : 2xA72@2.0G+6xA53@1.8G
- GPU : G310@800MHz
- NPU : 2TOPs
- 32-bit LPDDR4/4x5, eMMC5.1 HS400/UFS3.0

大算力协处理器

- 多核CPU,支持256bits矢量运算
- 多核NPU,16TOPs INT8算力
- 高性能嵌入式片内DRAM, 带宽高达1TBps



RKNN 是 Rockchip npu 平台使用的模型类型，以.rknn后缀结尾的模型文件。Rockchip 提供了完整了模型转换 Python工具——**RKNN-Toolkit2工具包**，方便用户将自主研发的算法模型转换成 RKNN 模型，同时 Rockchip 也提供了C/C++和Python API 接口。

RKNN-Toolkit 是为用户提供在 PC、Rockchip NPU 平台上进行模型转换、推理和性能评估的开发套件（相当于海思SDK文档），用户通过该工具提供的 Python 接口可以完成模型转换、量化功能、自定义算子和可视化功能等。

RKNN进行模型转换

虚拟机安装Ubuntu系统

安装依赖的虚拟环境 (python、依赖库)

安装RKNN-Toolkit2工具

执行模型转换流程

初始化



配置



加载



构建



导出模型



释放资源

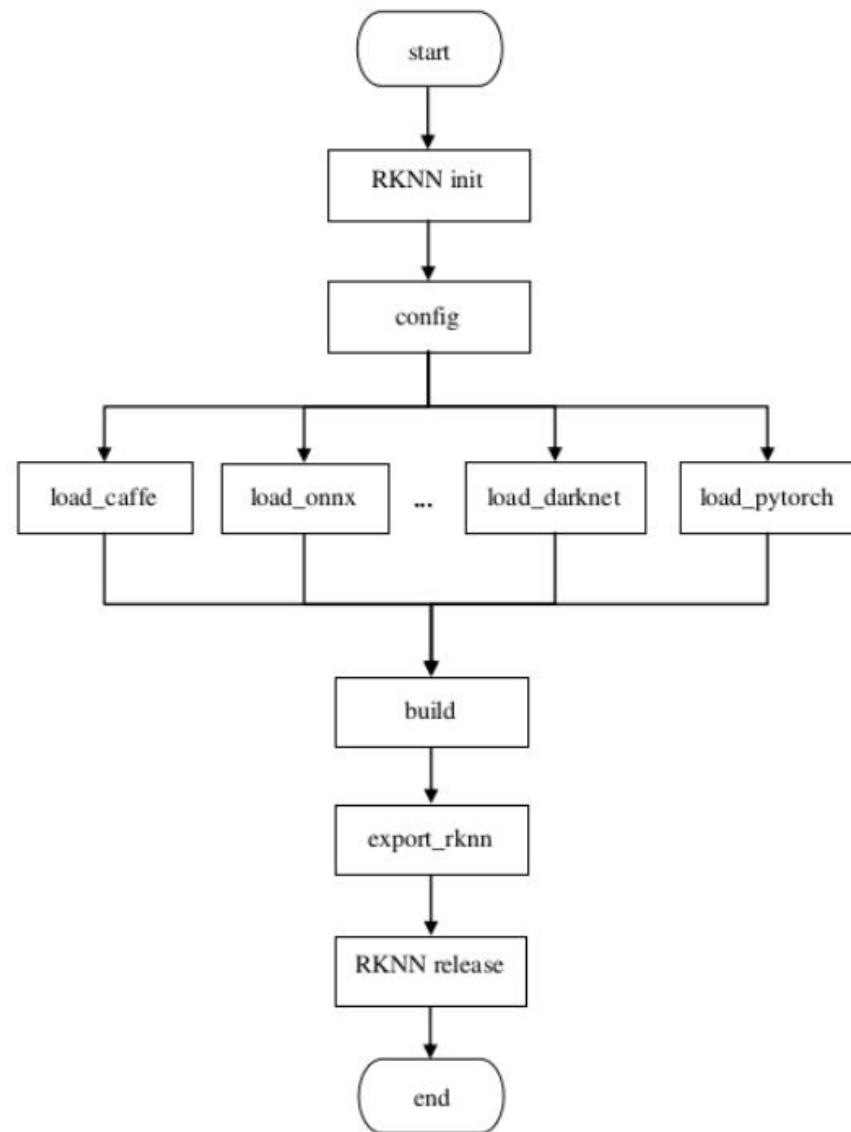


图3-1 模型转换流程图

将yolov5_onnx模型转为yolov5.rknn模型

RKNN-Toolkit工具包

```
(atc) zhwj@ubuntu:~$ atc --model=./Downloads/LMF_0102_epoch-best_3.onnx --framework=5 --output=./Downloads/LMF_0102_epoch-best_3 --soc_version="OPTG" --input_fp16_nodes="image"
ATC start working now, please wait for a moment.
ATC run failed, Please check the detail log, Try 'atc --help' for more information
E19022: Model featuremap requires [27977189376] memory, which exceeds system limit [27917287424].
        build graph failed, graph id:0, ret:245000[FUNC:BuildModel][FILE:ge_generator.cc][LINE:1327]
```

ONNX模型转RKNN模型

导出RKNN模型

```
if __name__ == '__main__':
    # 解析命令行参数
    model_path, platform, do_quant, output_path = parse_arg()

    # 创建RKNN对象
    rknn = RKNN(verbose=False)

    # 配置预处理参数
    print('--> Config model')
    rknn.config(mean_values=[[0, 0, 0]], std_values=[[255, 255, 255]], target_platform=platform)
    print('done')

    # 加载ONNX模型
    print('--> Loading model')
    ret = rknn.load_onnx(model=model_path)
    if ret != 0:
        print('Load model failed!')
        exit(ret)
    print('done')

    # 构建RKNN模型
    print('--> Building model')
    ret = rknn.build(do_quantization=do_quant, dataset=DATASET_PATH)
    if ret != 0:
        print('Build model failed!')
        exit(ret)
    print('done')

    # 导出RKNN模型
    print('--> Export rknn model')
    ret = rknn.export_rknn(output_path)
    if ret != 0:
        print('Export rknn model failed!')
        exit(ret)
    print('done')
```

板端NPU进行推理

PC端通过运行build-linux_RK356X.sh，将调用模型的C程序、转换得到的.rknn模型参数编译生成一个install和build文件夹，得到可执行文件

```
root@df6432c0c799:/rockchip/NPU/rknpu2/examples/rknn_yolov5_demo_v5# ./build-linux_RK356X.sh
-- The C compiler identification is GNU 6.3.1
-- The CXX compiler identification is GNU 6.3.1
-- Check for working C compiler: /rockchip/SDK/rk3566-rk3568-linux/prebuilts/gcc/linux-x86/aarch64/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-gcc
-- Check for working C compiler: /rockchip/SDK/rk3566-rk3568-linux/prebuilts/gcc/linux-x86/aarch64/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /rockchip/SDK/rk3566-rk3568-linux/prebuilts/gcc/linux-x86/aarch64/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-g++
-- Check for working CXX compiler: /rockchip/SDK/rk3566-rk3568-linux/prebuilts/gcc/linux-x86/aarch64/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-g++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found OpenCV: /rockchip/NPU/rknpu2/examples/3rdparty/opencv/opencv-linux-aarch64 (found version "3.4.5")
-- Configuring done
-- Generating done
-- Build files have been written to: /rockchip/NPU/rknpu2/examples/rknn_yolov5_demo_v5/build/build_linux_aarch64
Scanning dependencies of target rknn_yolov5_demo
[ 33%] Building CXX object CMakeFiles/rknn_yolov5_demo.dir/src/postprocess.cc.o
[ 66%] Building CXX object CMakeFiles/rknn_yolov5_demo.dir/src/main.cc.o
[100%] Linking CXX executable rknn_yolov5_demo
[100%] Built target rknn_yolov5_demo
[100%] Built target rknn_yolov5_demo
Install the project...
```

名称

convert_rknn_demo

include

model

src

utils

build-android.sh

build-linux.sh

CMakeLists.txt

README.md

README_CN.md

主文件夹

NPU

rknpu2

examples

rknn_yolov5_demo_v5

Q



build



convert_rknn_demo



include



install



model



src



build-android_RK356X.sh



build-android_RK3588.sh



build-linux_RK356X.sh



build-linux_RK3588.sh



CMakeLists.txt



README.md

CSDN @多万里

板端NPU进行推理

运行编译生成的可执行文件rknn_yolov5_demo，输入模型权重、图像路径运行程序获取推理结果

```
1 [root@RK356X:/mnt/rknn_yolov5_demo_Linux]# ./rknn_yolov5_demo ./model/RK356X/yolov5s-640-640.rknn ./model/bus.jpg
2 post process config: box_conf_threshold = 0.25, nms_threshold = 0.45
3 Read ./model/bus.jpg ...
4 img width = 640, img height = 640
5 Loading mode...
6 sdk version: 1.4.0 (a10f100eb@2022-09-09T09:07:14) driver version: 0.4.2
7 model input num: 1, output num: 3
8   index=0, name=images, n_dims=4, dims=[1, 640, 640, 3], n_elems=1228800, size=1228800, fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
9   index=0, name=334, n_dims=4, dims=[1, 255, 80, 80], n_elems=1632000, size=1632000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=77, scale=0.080445
10  index=1, name=353, n_dims=4, dims=[1, 255, 40, 40], n_elems=408000, size=408000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=56, scale=0.080794
11  index=2, name=372, n_dims=4, dims=[1, 255, 20, 20], n_elems=102000, size=102000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=69, scale=0.081305
12 model is NHWC input fmt
13 model input height=640, width=640, channel=3
14 once run use 62.407000 ms
15 loadLabelName ./model/coco_80_labels_list.txt
16 person @ (114 235 212 527) 0.819099
17 person @ (210 242 284 509) 0.814970
18 person @ (479 235 561 520) 0.790311
19 bus @ (99 141 557 445) 0.693320
20 person @ (78 338 122 520) 0.404960
21 loop count = 10 , average run 77.491700 ms
22
```