AWNN 使用指南

文档版本号：v0.9.2

发布日期：2020-10-19

商标声明


、全志和其他全志商标均为珠海全志科技股份有限公司的商标。本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受全志公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，全志公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保

前言

概述

本文档为基于 AWNN 开发算法模型的工程师提供开发指导。

产品版本

与本文档对应的产品版本。

产品名称	产品版本
V831/V833	AWNN V0.9.2

读者对象

本文档（本指南）主要适用于以下工程师：

使用 AWNN 进行开发的软件/算法开发工程师

名词解释

API——应用程序接口,全称：Application Programming Interface

SDK——软件开发包，全称：Software Development Kit

AWNN——AW 网络级中间件总称

修订记录

版本号	修订日期	修订内容
V0.8	**2020-7-15**	第一次发布。
V0.8.1	**2020-7-20**	修订部分内容。
V0.9.0	**2020-7-27**	修订部分内容。
V0.9.1	**2020-9-22**	增加了部分算子的使用约束说明； 增加了图像数据处理模块的说明； 增加了部分示例说明； 增加了模型转换建议。
V0.9.2	**2020-10-19**	增加了模型加密功能； 隐藏用户 first layer 设置； 修复了转换工具卷积算子不支持 bias_term==0 的 bug； 修复了转换工具算子多个输出对应 scale 的显示问题； 增加了转换工具对灰度图像输入的支持；

目录

1. AWNN 介绍

AWNN 是 AW AI 网络级套件的总称，其中分为两大模块，离线工具和推理引擎，二者配合使用。离线工具兼容 NCNN （开源框架，https://github.com/Tencent/ncnn） 的模型定义。除此之外，AWNN 还包含一个图像数据处理模块（硬件加速）。

2. AWNN 概述

2.1. 数据格式

AWNN 目前版本只支持 INT8（有符号 8 位）运算，主要是针对权重，特征图以及输入图像。

2.2. 算子支持

算子	描述
conv (卷积)	卷积算子[1]。
inner_product (内积运算)	全连接算子。
pooling (池化)	池化算子[2]，池化核支持 MAX 与 AVG[3] 两种模式，kernel ≤ 7x7。
eltwise (元素对位运算)	支持加法，乘法运算。
activation (激活运算)	支持 ReLU，PReLU（layer-wise/channel-wise）。
bn (批归一化)	采用通道模式进行批归一化(Batch Normalization)操作。
split （分离）	将 feature map 复制为多份。
concat（连接）	将多个 feature map[4]连接为一个。
interp（插值）	支持 nearest neighbor 插值。

2.3. 框架支持

当前版本兼容 NCNN 的模型与参数定义，其它框架模型需转换至 NCNN 格式。离线工具内已封装 MXNet 至 NCNN 的转换工具，其它框架转换请参考 https://github.com/Tencent/ncnn/tree/master/tools。

[1] 卷积算子对极少尺寸有限制，在运行时会直接退出并输出日志信息。

[2] 池化算子对输入宽度 w > 512 的 feature map 有限制，在运行时会直接退出并输出日志信息。

[3] Average Pool 算子在 padding 的情况下，边界处总以 kernel size （例如 7x7=49）计算平均值。

[4] 目前仅支持每个输入的 feature map 的通道数为 8 的倍数。

3. 图像数据处理模块

AWNN 网络级套件的图像数据处理模块，包括图像预处理（硬件加速）、量化和反量化。

3.1. 结构体

3.1.1. AWNNRect

【说明】

描述矩形的结构体。

【定义】

```
struct AWNNRect
{
    int tl_x;
    int tl_y;
    int br_x;
    int br_y;
    int width;
    int height;
};
```

【变量】

变量	含义	类型	取值范围
tl_x	矩形左上角像素点的横坐标	int	/
tl_y	矩形左上角像素点的纵坐标	int	/
br_x	矩形右下角像素点的横坐标	int	/
br_y	矩形右下角像素点的纵坐标	int	/
width	矩形的宽度	int	/
height	矩形的高度	int	/

【参考头文件】

AWNN_image.h

【备注】

3.1.2. AWNNSize

【说明】

描述矩形大小的结构体。

【定义】

```
struct AWNNSize
{
    int width;
    int height;
};
```

【变量】

变量	含义	类型	取值范围
width	矩形的宽度	int	/
height	矩形的高度	int	/

【参考头文件】

AWNN_image.h

【备注】

3.1.3. AWNNYuvType

【说明】

yuv 数据格式。

【定义】

```
typedef enum {
    AWNN_NONE_YUV,
    AWNN_YUV420SP_NV21
}AWNNYuvType;
```

【变量】

类型	含义	类型	取值范围
AWNN_NONE_YUV	初始默认值	enum	/
AWNN_YUV420SP_NV21	YUV420SP_NV21 格式	enum	/

【参考头文件】

AWNN_image.h

【备注】

3.1.4. AWNNColorSpace

【说明】

图像格式类型。

【定义】

```
typedef enum {
    AWNN_NONE_SPACE,
    AWNN_RGB
}AWNNColorSpace;
```

【变量】

类型	含义	类型	取值范围
AWNN_NONE_SPACE	初始默认值	enum	/
AWNN_RGB	图像数据按 R-G-B 逐像素顺序排列	enum	/

【参考头文件】

AWNN_image.h

【备注】

3.1.5. AWNNYuv

【说明】

描述 yuv 数据的结构体。

【定义】

```
struct AWNNYuv
{
    int w;
    int h;
    unsigned char *data;
    AWNNYuvType yuv_type;
};
```

【变量】

变量	含义	类型	取值范围
w	yuv 中的 y 分量的宽度	int	/
h	yuv 中的 y 分量的高度	int	/
data	指向 yuv 数据	unsigned char *	[0, 255]
yuv_type	yuv 数据格式	enum	/

【参考头文件】

AWNN_image.h

【备注】

3.1.6. AWNNImg

【说明】

描述图像数据的结构体。

【定义】

```
struct AWNNImg
{
    int w;
    int h;
    int c;
    unsigned char *data;
    AWNNColorSpace c_space;
};
```

【变量】

变量	含义	类型	取值范围
w	图像的宽度	int	/
h	图像的高度	int	/
c	图像的通道数	int	/
data	指向图像像素数据的指针	unsigned char *	[0, 255]
c_space	图像的颜色空间	enum	/

【参考头文件】

AWNN_image.h

【备注】

3.2. 函数说明

3.2.1. awnn_make_blank_image

【描述】

创建像素数据为空的 AWNNImg 结构体指针（为像素数据分配了内存）。

【原型】

```
AWNNImg *awnn_make_blank_image(int w, int h, AWNNColorSpace c_space);
```

【参数】

参数名称	类型	描述	输入/输出
w	int	所创建图像的宽度	输入
h	int	所创建图像的高度	输入
c_space	enum	所创建图像的颜色空间	输入

【返回值】

参数类型	描述	返回值
struct *	指向所创建的图像结构体 AWNNImg 的指针	/

【参考头文件】

AWNN_image.h

【例程】

【备注】

3.2.2. awnn_make_yuv_image

【描述】

根据跟定的 yuv 数据和格式，及其 y 分量的宽度和高度，创建并返回指向 AWNNYuv 结构体的指针。

【原型】

```
AWNNYuv *awnn_make_yuv_image(const unsigned char *yuv_buffer, int w, int h,
AWNNYuvType yuv_type);
```

【参数】

参数名称	类型	描述	输入/输出
yuv_buffer	const unsigned char *	指向 yuv 数据的指针	输入
w	int	y 分量的宽度	输入
h	int	y 分量的高度	输入
yuv_type	enum	yuv 的数据类型	输入

【返回值】

参数类型	描述	返回值
struct *	指向所创建的结构体 AWNNYuv 的指针	/

【参考头文件】

AWNN_image.h

【例程】

【备注】

3.2.3. awnn_make_rgb_image

【描述】

根据给定的像素数据和图像大小、颜色空间，创建并返回 AWNNImg 结构体指针。

【原型】

```
AWNNImg *awnn_make_rgb_image(const unsigned char *rgb_buffer, int w, int h,
AWNNColorSpace c_space);
```

【参数】

参数名称	类型	描述	输入/输出
rgb_buffer	const unsigned char *	指向 rgb 像素数据	输入
w	int	所创建图像的宽度	输入
h	int	所创建图像的高度	输入
c_space	enum	所创建图像的颜色空间	输入

【返回值】

参数类型	描述	返回值
struct *	指向所创建的图像结构体指针	/

【参考头文件】

AWNN_image.h

【例程】

【备注】

3.2.4. awnn_make_rect

【描述】

根据矩形对角两个像素点的坐标，创建并返回 AWNNRect 结构体。

【原型】

```
AWNNRect  awnn_make_rect(int tl_x, int tl_y, int br_x, int br_y);
```

【参数】

参数名称	类型	描述	输入/输出
tl_x	int	矩形左上角像素点的横坐标	输入
tl_y	int	矩形左上角像素点的纵坐标	输入
br_x	int	矩形右下角像素点的横坐标	输入
br_y	int	矩形右下角像素点的纵坐标	输入

【返回值】

参数类型	描述	返回值
struct	所创建的 AWNNRect 结构体	/

【参考头文件】

AWNN_image.h

【例程】

【备注】

3.2.5. awnn_yuv2rgb

【描述】

输入 yuv 格式的像素数据，转换并生成 rgb 排列格式的图像结构体。

【原型】

```
int awnn_yuv2rgb(const AWNNYuv *src_yuv, AWNNImg *dst_img);
```

【参数】

参数名称	类型	描述	输入/输出
src_yuv	const struct *	指向源 yuv 数据结构体	输入
dst_img	struct *	指向图像结构体	输入/输出

【返回值】

参数类型	描述	返回值
int	0：成功；-1：失败	/

【参考头文件】

AWNN_image.h

【例程】

【备注】

3.2.6. awnn_resize

【描述】

图像尺度缩放。

【原型】

```
int awnn_resize(const AWNNImg *src_img, AWNNImg *dst_img);
```

【参数】

参数名称	类型	描述	输入/输出
src_img	const struct *	指向源图像结构体	输入
dst_img	struct *	指向缩放后的图像结构体	输入/输出

【返回值】

参数类型	描述	返回值
int	0：成功；-1：失败	/

【参考头文件】

AWNN_image.h

【例程】

【备注】

3.2.7. awnn_crop

【描述】

图像剪裁。

【原型】

```
int awnn_crop(const AWNNImg *src_img, AWNNRect rect, AWNNImg *dst_img);
```

【参数】

参数名称	类型	描述	输入/输出
src_img	const struct *	指向源图像结构体	输入
rect	struct	用于剪裁的框信息	输入
dst_img	struct *	指向缩放后的图像结构体	输入/输出

【返回值】

参数类型	描述	返回值
int	0：成功；-1：设置	/

【参考头文件】

AWNN_image.h

【例程】

【备注】

3.2.8. awnn_crop_resize

【描述】

对图像剪裁，并根据预设的图像宽度和高度进行尺度缩放。

【原型】

```
int awnn_crop_resize(const AWNNImg *src_img, AWNNRect rect, AWNNImg *dst_img);
```

【参数】

参数名称	类型	描述	输入/输出
src_img	const struct *	指向源图像结构体	输入
rect	struct	用于剪裁的框信息	输入
dst_img	struct *	指向缩放后的图像结构体	输入/输出

【返回值】

参数类型	描述	返回值
int	0：成功；-1：失败	/

【参考头文件】

AWNN_image.h

【例程】

【备注】

3.2.9. awnn_yuv2rgb_resize

【描述】

把 yuv 格式的像素数据转换为 rgb 排列格式的数据，并根据预设的图像大小进行尺度缩放。

【原型】

```
int awnn_yuv2rgb_resize(const AWNNYuv *src_yuv, AWNNImg *dst_img);
```

【参数】

参数名称	类型	描述	输入/输出
src_img	const struct *	指向源 yuv 数据结构体	输入
dst_img	struct *	指向缩放后的图像结构体	输入/输出

【返回值】

参数类型	描述	返回值
int	0：成功；-1：失败	/

【参考头文件】

AWNN_image.h

【例程】

【备注】

3.2.10. awnn_yuv2rgb_crop_resize

【描述】

把 yuv 格式的像素数据转换为 rgb 排列格式的数据，按照矩形框信息进行剪裁，最后根据预设的图像大小进行尺度缩放。

【原型】

int awnn_yuv2rgb_crop_resize(const AWNNYuv *src_yuv, AWNNRect rect, AWNNImg *dst_img);【参数】

参数名称	类型	描述	输入/输出
src_yuv	const struct *	指向源 yuv 数据结构体	输入
rect	struct	用于剪裁的框信息	输入
dst_img	struct *	指向缩放后的图像结构体	输入/输出

【返回值】

参数类型	描述	返回值
int	0：成功；-1：失败	/

【参考头文件】

AWNN_image.h

【例程】

【备注】

3.2.11. awnn_normalize

【描述】

数据归一化，减去均值，除以方差。

【原型】

```
void awnn_normalize(const unsigned char *src_data, int size,
std::vector<float> &mean_vals, std::vector<float> &norm_vals, float *dst_data);
```

【参数】

参数名称	类型	描述	输入/输出
src_img	const unsigned char *	指向源图像结构体	输入
size	int	源图像数据的数量	输入
mean_vals	std::vector<float>	均值	输入
norm_vals	std::vector<float>	方差	输入
dst_data	float *	指向归一化数据的指针	输出

【返回值】

参数类型	描述	返回值
int	/	/

【参考头文件】

AWNN_image.h

【例程】

【备注】无硬件加速。

3.2.12. awnn_quantize

【描述】

数据量化（float -> int8）。

【原型】

```
void awnn_quantize(const float *src_data, int size, float input_scale,
signed char *dst_data);
```

【参数】

参数名称	类型	描述	输入/输出

src_img	const float *	指向源数据	输入
size	int	源数据的数量	输入
input_scale	float	输入层 scale	输入
dst_data	signed char *	指向量化后的数据	输出

【返回值】

参数类型	描述	返回值
int	/	/

【参考头文件】

AWNN_image.h

【例程】

【备注】无硬件加速。

3.2.13. awnn_dequantize

【描述】

数据的反量化（int8 -> float）。

【原型】

```
void awnn_dequantize(const signed char *src_data, int size, float output_scale,
float *dst_data);
```

【参数】

参数名称	类型	描述	输入/输出
src_img	const signed char *	指向源数据	输入
size	int	源数据的数量	输入
output_scale	float	反量化 scale	输入
dst_data	float *	指向反量化后的数据	输出

【返回值】

参数类型	描述	返回值
int	/	/

【参考头文件】

AWNN_image.h

【例程】

【备注】无硬件加速。

3.2.14. awnn_free_image

【描述】

释放结构体 AWNNImg 占用的内存。

【原型】

```
void awnn_free_image(AWNNImg *img);
```

【参数】

参数名称	类型	描述	输入/输出
img	struct *	指向图像结构体	输入

【返回值】

参数类型	描述	返回值
int	/	/

【参考头文件】

AWNN_image.h

【例程】

【备注】

3.3. 应用示例

3.3.1. awnn_yuv2rgb 示例

```
void test_yuv2rgb() {

    int w = 320;
    int h = 240;
    int buffer_len = w*h*3/2;
    unsigned char *yuv_buffer = (unsigned char*)calloc(buffer_len, sizeof(unsigned char));

    FILE *fp;
    fp = fopen("test.yuv", "rb");
    if (fp == NULL) {
        printf("Open file failed\n");
    }
    fread(yuv_buffer, buffer_len, 1, fp);
    fclose(fp);

    AWNNYuv *src_yuv = awnn_make_yuv_image(yuv_buffer, w, h, AWNN_YUV420SP_NV21);
    AWNNImg *dst_img = awnn_make_blank_image(w, h, AWNN_RGB);
    awnn_yuv2rgb(src_yuv, dst_img);

    char name[100] = {0};
    sprintf(name, "yuv2rgb_%dx%d.rgb", w, h);

    fp = fopen(name, "wb");
    if (fp == NULL) {
        printf("Open file failed\n");
    }
    fwrite(dst_img->data,  w*h*3, 1, fp);
    fclose(fp);

    awnn_free_image(dst_img);
    free(yuv_buffer);
    free(dst_img);
    free(src_yuv);
}
```

3.3.2. awnn_yuv2rgb_crop_resize 示例

```
void test_yuv2rgb_crop_resize() {
    int w = 320;
    int h = 240;
    int buffer_len = w*h*3/2;
    unsigned char *yuv_buffer = (unsigned char*)calloc(buffer_len, sizeof(unsigned char));

    FILE *fp;
    fp = fopen("test.yuv", "rb");
    if (fp == NULL) {
        printf("Open file failed\n");
        return 1;
    }
    fread(yuv_buffer, buffer_len, 1, fp);
    fclose(fp);

    int resized_w = 600;
    int resized_h = 600;
    AWNNRect rect = awnn_make_rect(10, 25, 150, 150);

    AWNNYuv *src_yuv = awnn_make_yuv_image(yuv_buffer, w, h, AWNN_YUV420SP_NV21);
    AWNNImg *dst_img = awnn_make_blank_image(resized_w, resized_h, AWNN_RGB);

    awnn_yuv2rgb_crop_resize(src_yuv, rect, dst_img);

    char name[100] = {0};
    sprintf(name, "yuv2rgb_crop_resize_%dx%d.rgb", resized_w, resized_h);
    fp = fopen(name, "wb");
    if (fp == NULL) {
        printf("Open file failed\n");
    }
    fwrite(dst_img->data, dst_img->w * dst_img->h* 3, 1, fp);
    fclose(fp);

    awnn_free_image(dst_img);
    free(yuv_buffer);
    free(src_yuv);
    free(dst_img);
}
```

4. AWNN 在线推理引擎

4.1. API 说明

4.1.1. AWNNInit

【描述】

硬件模块和 AWNN 初始化函数。

【语法】

```
void AWNNInit();
```

【参数】

参数名称	描述	输入/输出
/	/	/

【返回值】

参数类型	描述	返回值
/	/	/

【参考头文件】

```
AWNN_interface.h
```

【例程】

【备注】**AWNNInit 只需执行一次**

4.1.2. AWNNDeinit

【描述】

硬件模块和 AWNN 释放函数。

【语法】

```
void AWNNDeinit();
```

【参数】

参数名称	描述	输入/输出
/	/	/

【返回值】

参数类型	描述	返回值
/	/	/

【参考头文件】

AWNN_interface.h

【例程】

【备注】**AWNNDeinit 只需执行一次**

4.1.3. AWNNInstance

【描述】

AWNNInstance 类的构造函数。

【语法】

AWNNInstance:: AWNNInstance();

【参数】

参数名称	描述	输入/输出
/	/	/

【返回值】

参数类型	描述	返回值
/	/	/

【参考头文件】

AWNN_interface.h

【例程】

【备注】

4.1.4. create

【描述】

构造网络前向计算图实例。

【语法】

int AWNNInstance::create(const AWNNConfig& config);

【参数】

参数名称	描述	输入/输出
config	网络级配置的描述结构体，具体参数描述请参考结构体部分。	输入

【返回值】

参数类型	描述	返回值
int	网络是否创建成功的标志	创建成功返回 0

【参考头文件】

AWNN_interface.h

【例程】

【备注】

4.1.5. inference

【描述】

网络前向推理计算。

【语法】

```
int AWNNInstance::inference(AWNNSessionConfig& sessConfig);
```

【参数】

参数名称	描述	输入/输出
sessConfig	网络 runtime 配置的描述结构体，具体参数描述请参考结构体部分。	输入

【返回值】

参数类型	描述	返回值
int	网络前向计算是否成功的标志	计算成功返回 0

【参考头文件】

AWNN_interface.h

【例程】

【备注】

4.1.6. destroy

【描述】

销毁网络前向计算图实例。

【语法】

```
void AWNNInstance::destroy()
```

【参数】

参数名称	描述	输入/输出
/	/	/

【返回值】

参数类型	描述	返回值
/	/	/

【参考头文件】

AWNN_interface.h

【例程】

【备注】

4.1.7. getTensorScale

【描述】

获取 tensor 对应的量化 scale。

【语法】

```
int AWNNInstance::getTensorScale(const std::string& tensorName, float& tensorScale);
int AWNNInstance::getTensorScale(const int& tensorID, float& tensorScale);
```

【参数】

参数名称	描述	输入/输出
tensorName	欲获取 tensor 的名字	输入
tensorID	欲获取 tensor 的索引	输入
tensorScale	Tensor 对应的量化 scale	输入

【返回值】

参数类型	描述	返回值
int	获取 scale 是否成功的标志	计算成功返回 0

【参考头文件】

AWNN_interface.h

【例程】

【备注】

4.2. 结构体说明

4.2.1. AWNNTensorDims

【说明】

数据维度结构体。

【定义】

```
struct AWNNTensorDims
{
    uint32_t h = 0;
    uint32_t w = 0;
    uint32_t c = 0;
};
```

【变量】

变量	含义	类型	取值范围
h	输入数据的高度。	uint32_t	/
w	输入数据的宽度。	uint32_t	/
c	输入数据的通道数。	uint32_t	/

【参考头文件】

AWNN_interface.h

【备注】

4.2.2. AWNNTensorDesc

【说明】

数据描述结构体。

【定义】

```
struct AWNNTensorDesc
{
    AWNNTensorLayout layout = LAYOUT_HWC;
    AWNNTensorDataType dataType = DATA_TYPE_INT8;
    AWNNTensorDims dims;
    std::string tensorName;
    int tensorID = -1;
    // tensor address
    void* data = NULL;
```

```
    // tensor size
    uint32_t size = 0;
    // quantization scale of tensor
    float tensorScale = 1.0;
};
```

【变量】

变量	含义	类型	取值范围
layout	数据的排序格式	AWNNTensorLayout	目前仅支持 hwc 格式
dataType	数据的类型	AWNNTensorDataType	目前仅支持 int8
dims	数据的维度	AWNNTensorDims	/
tensorName	数据的名字	std::string	/
tensorID	数据的索引	int	/
data	数据的首地址指针	void*	/
size	数据的大小	uint32_t	/
tensorScale	数据的量化 scale	float	/

【参考头文件】

AWNN_interface.h

【备注】

4.2.3. AWNNConfig

【说明】

网络级配置结构体。

【定义】

```
struct AWNNConfig
{
    bool paramInvisible = false;

    // network model path
    std::string paramPath;
    std::string modelPath;

    // network model pointer
    unsigned char* paramBuffer = NULL;
```

```
    unsigned char* modelBuffer = NULL;
};
```

变量】

变量	含义	类型	取值范围
paramInvisible	网络配置是否不可见标志	bool	True/false
paramPath	网络配置路径	std::string	/
modelPath	网络参数路径	std::string	/
paramBuffer	网络配置指针	unsigned char*	/
modelBuffer	网络参数指针	unsigned char*	/

【参考头文件】

AWNN_interface.h

【备注】

4.2.4. AWNNSessionConfig

【说明】

网络 runtime 配置结构体。

【定义】

```
struct AWNNSessionConfig
{
    // inference device type
    AWNNInferenceType type = AWNN_FORWARD_IPU;

    // input/output tensor names of model
    std::vector<std::string> inputNames;
    std::vector<std::string> outputNames;
    // input/output tensor indexes of model
    std::vector<int> inputIDs;
    std::vector<int> outputIDs;

    // input/output tensors description
    std::vector<AWNNTensorDesc> inputTensors;
    std::vector<AWNNTensorDesc> outputTensors;
};
```

【变量】

变量	含义	类型	取值范围
type	计算设备类型	AWNNInferenceType	目前仅支持 IPU
inputNames	网络输入数据的名称	std::vector<std::string>	/
outputNames	网络输出数据的名称	std::vector<std::string>	/
inputIDs	网络输入数据的索引	std::vector<int>	/
outputIDs	网络输出数据的索引	std::vector<int>	/
inputTensors	网络输入数据	std::vector<AWNNTensorDesc>	/
outputTensors	网络输出数据	std::vector<AWNNTensorDesc>	/

【参考头文件】

AWNN_interface.h

【备注】

4.3. 应用示例

4.3.1. 准备

神经网络硬件单元仅支持 int8 权重（特定格式）与 int8 输入数据进行推理计算。进行推理计算之前，必须进行如下准备工作：

模型文件：转换[5]（MXNet，ONNX，TensorFlow，PyTorch 至 NCNN），优化，校正（产生量化数据表）以及量化为 8bit 模型文件[6]。

图像数据：图像的预处理[7]（例如 YUV2RGB，Crop，Resize），归一化以及量化[8]。

4.3.2. 初始化

首先进行初始化（AWNNInit）、内存分配，如下图所示。

```
AWNNInit();

signed char* inputBuffer = (signed char*)malloc(48 * 48 * 3);
signed char* outputBuffer = (signed char*)malloc(2 + 4 + 10);
```

若以模型加密方式进行 inference，需要提前预分配内存，并将模型加载到内存，如下图所示。

```
const char* paramPath = "../data/model_test/onet/ONet.param.bin";
const char* modelPath = "../data/model_test/onet/det3_int8.bin";
size_t paramBufferSize = 0;
size_t modelBufferSize = 0;
getBinSize(paramPath, paramBufferSize);
getBinSize(modelPath, modelBufferSize);
unsigned char* paramBuffer = (unsigned char*)malloc(paramBufferSize);
unsigned char* modelBuffer = (unsigned char*)malloc(modelBufferSize);
loadFromBin(paramPath, paramBufferSize, paramBuffer);
loadFromBin(modelPath, modelBufferSize, modelBuffer);
```

4.3.3. 配置网络

主要对模型 param 和 model 的地址或者指针进行配置。**值得注意的是，paramInvisible = false 时应该配置模型路径，paramInvisible = true 时应该配置模型指针。**

若以 paramInvisible = false 配置网络，如下图所示。

```
AWNNConfig config;
config.paramInvisible = false;
config.paramPath = "../data/model_test/onet/det3_int8.param";
config.modelPath = "../data/model_test/onet/det3_int8.bin";
```

[5] 模型的转换请参见附录：模型转换。

[6] 模型的优化，矫正与量化请参见：AWNN 离线工具。

[7] 图像处理请参见：图像数据处理模块。

[8] 数据的量化与反量化请参见：图像数据处理模块中的 awnn_quantize，awnn_dequantize 函数。

若以 paramInvisible = true 配置网络，如下图所示。

```
AWNNConfig config;
config.paramInvisible = true;
config.paramBuffer = paramBuffer;
config.modelBuffer = modelBuffer;
```

4.3.4. 配置 session

主要配置网络 runtime 参数，输入输出 tensor 名称和 input tensors 与 output tensors。

```
AWNNSessionConfig sessConfig;
sessConfig.type = AWNN_FORWARD_IPU;
sessConfig.inputNames =  { "data" };
sessConfig.outputNames = { "conv6_1", "conv6_2", "conv6_3" };

AWNNTensorDesc input;
input.dims.w = 48;
input.dims.h = 48;
input.dims.c = 3;
input.size = 48 * 48 * 3;
input.data = (void*)inputBuffer;
sessConfig.inputTensors.push_back(input);

AWNNTensorDesc output1, output2, output3;
output1.data = (void*)outputBuffer;
output2.data = (void*)(outputBuffer + 2);
output3.data = (void*)(outputBuffer + 2 + 4);
sessConfig.outputTensors.push_back(output1);
sessConfig.outputTensors.push_back(output2);
sessConfig.outputTensors.push_back(output3);
```

以模型加密方式配置 session 时，如下图所示。值得注意的是，需要用户在应用程序中 include 转换后的*id.h 头文件，这样才可正确配置输入输出数据索引。

```
AWNNSessionConfig sessConfig;
sessConfig.type = AWNN_FORWARD_IPU;
sessConfig.inputIDs =  { ONet_param_id::BLOB_data };
sessConfig.outputIDs = { ONet_param_id::BLOB_conv6_1,
    ONet_param_id::BLOB_conv6_2, ONet_param_id::BLOB_conv6_3 };

AWNNTensorDesc input;
input.dims.w = 48;
input.dims.h = 48;
input.dims.c = 3;
input.size = 48 * 48 * 3;
input.data = (void*)inputBuffer;
sessConfig.inputTensors.push_back(input);

AWNNTensorDesc output1, output2, output3;
output1.data = (void*)outputBuffer;
output2.data = (void*)(outputBuffer + 2);
output3.data = (void*)(outputBuffer + 2 + 4);
sessConfig.outputTensors.push_back(output1);
sessConfig.outputTensors.push_back(output2);
sessConfig.outputTensors.push_back(output3);
```

4.3.5. 运行

最后创建网络实例(create)、前向推理(inference)以及销毁实例(destroy)。在对多张图像进行前向推理时，创建实例和销毁实例只需执行一次，前向推理循环执行多次。

值得注意的是，这里假设用户已经对数据（data_fake_inputs.bin）进行了预处理与量化。例如，对于图像数据，往往首先需要进行归一化操作（均值、方差），然后进行从 float 类型到 int8 类型的量化[9]。

其中 fp_scale 为该 tensor 对应的量化因子。在创建完网络实例后，用户可以调用 getTensorScale() 函数得到对应 tensor 的 fp_scale。值得注意的是，输出结果存储在 output tensors 中，并配有相应的 fp_scale。可根据该值，反量化[10]获得输出内容的 float 数据。

[9] 数据的量化请参见：图像数据处理模块中的 awnn_quantize 函数。

[10] 数据的反量化请参见：图像数据处理模块中的 awnn_dequantize 函数。

```
/* create AWNNInstance */
AWNNInstance caseNet;
int createFlag = caseNet.create(config);

/* inference pipeline */
int loopNumber = 500;
if (createFlag == 0)
{
    for (int index = 0; index < loopNumber; index++)
    {
        // load data
        const char* binPath = "../data/model_test/onet/data_fake_inputs.bin";
        loadFromBin(binPath, 48 * 48 * 3, inputBuffer);

        // inference
        int inferenceFlag = caseNet.inference(sessConfig);

        // compare
        if (inferenceFlag == 0)
        {
            std::vector<std::string> resultPaths;
            resultPaths.push_back("../data/model_test/onet/conv6_1_int8.bin");
            resultPaths.push_back("../data/model_test/onet/conv6_2_int8.bin");
            resultPaths.push_back("../data/model_test/onet/conv6_3_int8.bin");
            for (size_t i = 0; i < resultPaths.size(); i++)
            {
                compareResult(resultPaths[i].c_str(), (signed char*)sessConfig.outputTensors[i].data,
                    sessConfig.outputTensors[i].size);
            }
        }
    }
}
caseNet.destroy();
free(inputBuffer);
free(outputBuffer);
AWNNDeinit();
```

5. AWNN 离线工具

AWNN 的离线工具（awnntools.exe）以可执行文件的形式提供，包含 4 个功能，分别是模型转换、模型优化、基于数据的矫正以及 int8 的量化。请在**命令行中**以如下形式调用：

```
>> awnntools.exe  command  param#1  param#2  ...
```

5.1.转换：convert

转换工具将 MXNet 模型转换为 NCNN 模型，调用方式如下：

```
>> awnntools.exe  convert  mxnet.json  mxnet.params  target.param  target.bin
```

5.2.优化：optimize

优化工具对转换后的 NCNN 模型进行优化，调用方式如下：

```
>> awnntools.exe  optimize  target.param  target.bin  opt.param  opt.bin
```

5.3.矫正：calibrate

矫正工具根据给定的数据（图像），产生用于量化的 table 文件，调用方式如下：

```
>> awnntools.exe  calibrate  -p=xxx  -b=xxx  -i=xxx  -o=xxx  -c=swapRB  ...
```

其中，参数设置方式如下：

参数路径	-p=opt.param
权重路径	-b=opt.bin
矫正图像路径	-i=/image_path/
输出 table 文件	-o=opt.table
均值	-m=127.5,127.5,127.5 （默认 104.0，117.0， 123.0） （若打开 GARY 开关，-m=127.5）
归一化因子	-n=0.0078125,0.0078125,0.0078125 （默认 1.0， 1.0， 1.0） （若打开 RGB 开关，-n=0.0078125）
图像 resize 的目标尺寸	-s=48,48 (默认 w=224, h=224)
GRAY 格式开关	-g,--gray（若不配置则默认为 BGR，GRAY 格式开关与 RGB 格式开

	关不可同时开启）
RGB 格式开关	-c,--swapRB （若不配置则默认为 BGR，GRAY 格式开关与 RGB 格式开关不可同时开启）
处理线程数	-t=4 （默认 4，请根据计算机配置自行调整）
矫正图片集可考虑直接采用训练中的验证数据集（以 >5000 张，覆盖真实场景为宜），并务必保证矫正时图像的预处理方式与训练和部署时一致。	

5.4.量化：quantize

量化工具根据给定的量化 table 文件，量化 Float 模型为 int8 的 IPU 格式：

```
>> awnntools.exe quantize opt.param opt.bin opt_int8.param opt_int8.bin opt.table
```

5.5.加密：encrypt

加密工具对量化好的模型进行加密，调用方式如下：

```
>> awnntools.exe encrypt opt_int8.param opt_int8.id.h
```

附录

模型转换建议

- ✓ MXNet to NCNN：请参见章节 5.1 转换：convert。
- ✓ ONNX to NCNN：访问 https://convertmodel.com/ 。请不要复选任何优化选项（如左下图所示）。若提示 NCNN 算子不支持，可先进行 ONNX 模型的简化（如右下图所示）。

- ✓ PyTorch to NCNN：建议从 PyTorch -> ONNX -> NCNN 间接转换。
- ✓ TensorFlow to NCNN：建议从 TensorFlow pb -> ONNX -> NCNN 间接转换[11]。

模型推理耗时

Model	Input Size	Number of Inference	Average Time (ms)
VGG16	224x224x3	500	282.78
ResNet18-V2	224x224x3	500	40.68
ResNet18-V2 (PRELU)	224x224x3	500	43.80
ResNet50-V1	224x224x3	500	99.49
MX_R34	112x112x3	500	76.16
LResNet100E-IR	112x112x3	500	179.37

[11] TensorFlow 至 ONNX 转换请访问： https://github.com/onnx/tensorflow-onnx ；值得注意的是，由于 TensorFlow 与 ONNX 数据排布不同，转换时请强制保持 NCHW 排布：https://github.com/onnx/tensorflow-onnx/blob/master/README.md#--inputs-as-nchw 。