Tengine

用户使用手册

2018-11-04

OPEN AI LAB

变更记录

日期	版本	说明	作者
2018-10-10	0.1.0	初版	Cheng Meng
2018-11-04	0.1.1	修改拼写	路明

目录

1 前言

1.1 目的

本文档主要介绍由 OPEN AI LAB 开发的嵌入式深度学习推理框架 Tengine 的功能、特点和使用方法，旨在帮助开发人员更快，更有效的掌握 Tengine 的开发方法。

Tengine 由开放智能实验室(OPEN AI Lab)开发的一个精简的、高性能的、用于嵌入式设备的模块化推理引擎。

本手册旨在帮助用户来了解如何使用 Tengine 构建 AI 推理应用程序。

1.2 术语

- **EAIDK**：Embedded AI Development Kit。嵌入式人工智能开发套件。
- **AID**：AI Distro。AID 是 OPEN AI LAB 开发的一个面向嵌入式平台前端智能，跨 SoC 的 AI 核心软件平台。
- **BladeCV**：BladeCV 是 OPEN AI LAB 开发的，在嵌入式平台上替代 OpenCV 的计算机视觉开发包，包含计算机视觉算法、图像获取和图形界面三部分。
- **MIPI**：Mobile Industry Processor Interface，移动产业处理器接口。
- **eDP**：Embedded DisplayPort，嵌入式数码音视讯传输接口
- **CTIA**: Cellular Telecommunications and Internet Association. 美国无线通信和互联网协会。该协会制定的 AHJ（American Headset Jack）定义了二合一音频插孔的标准。

2 支持范围

2.1 硬件支持

本版本包含对 ARM64 CPU 的支持。

2.2 OS 支持

支持 Android 上的应用开发。

3 产品构成

本版本软件包主要包含以下内容:

- 一个 C 语言的头文件（tengine_c_api.h）
- 一个动态库（libtengine.so）：由 C/C++语言和汇编代码实现的库。
- 还包含其他:

  一个基于 Tengine 写图像分类应用的样例工程

  解析 Caffe 模型和 Tensorflow 模型需要的 protobuf 库

4 接口说明

4.1 类型说明

```
typedef void * user_context_t;
typedef void * workspace_t;
typedef void * graph_t;
typedef void * tensor_t;
typedef void * node_t;
```

4.2 接口说明

下面列出了开发一个应用程序时，用到的主要接口。

更多的接口定义和说明，请查看头文件 tengine_c_api.h 中的注释。

- init_tengine_library：初始化 tengine
- release_tengine_library：释放 tengine
- get_tengine_version：获取 Tengine 版本号
- request_tengine_version：请求 Tengine 版本号
- load_model：通过文件加载模型
- remove_model：移除模型
- create_runtime_graph：创建运行图
- destroy_runtime_graph：销毁运行图
- prerun_graph：准备运行图需要的资源
- run_graph：运行图，执行推理。可以反复多次调用。
- postrun_graph：释放运行图需要的资源
- get_graph_tensor：获取图的张量
- get_graph_input_tensor：获取图的输入张量
- get_graph_output_tensor：获取图的输出张量
- put_graph_tensor：释放张量句柄
- set_tensor_shape：设置张量的形状。
- get_tensor_shape：获取张量的形状
- set_tensor_buffer：设置张量的缓存区
- get_tensor_buffer：获取张量的缓存区

4.2.1 Tengine 创建和销毁

4.2.1.1 init_tengine_library

接口名称	`init_tengine_library`	
接口说明	使用Tengine库前，首先需要执行`init_tengine_library()`，对Tengine库进行初始化操作。	
参数	无	
返回值	int	0： 表示成功。 -1： 表示失败。

接口声明	int init_tengine_library(void);

4.2.1.2 release_tengine_library

接口名称	release_tengine_library	
接口说明	执行release_tengine_library()，对Tengine库进行资源释放操作。	
参数	无	
返回值	无	
接口声明	void release_tengine_library(void);	

4.2.1.3 get_tengine_version

接口名称	get_tengine_version	
接口说明	获取Tengine库的版本信息。	
参数	无	
返回值	const char *	字符串指针。
接口声明	const char * get_tengine_version(void);	

4.2.1.4 request_tengine_version

接口名称	request_tengine_version	
接口说明	初始化Tengine后，通过该接口获取对版本的兼容情况。	
参数	const char * version	版本号。
返回值	int	1： 表示支持。 0： 表示不支持。
接口声明	int request_tengine_version(const char * version);	

4.2.2 模型的加载和移除

4.2.2.1 load_model

接口名称	load_model	
接口说明	加载已保存成文件的模型到系统中。	
参数	const char * model_name	指定的模型名称，用于创建运行图时指定模型用。
	const char * model_format	模型文件格式。例如： “caffe", "tensorflow", "onnx","tengine"。
	const char * fname	文件名称。根据模型格式的不同，需要填入的文件名也不一样。
返回值	int	0： 表示成功。 -1： 表示失败。
接口声明	int load_model(const char * model_name, const char * model_format, const char * fname, ...);	

4.2.2.2 remove_model

接口名称	remove_model	
接口说明	移除模型。	
参数	const char * model_name	模型名称。例如： “squeezenet”、“mobilenet”等。
返回值	int	0： 表示成功。 -1： 表示失败。
接口声明	int remove_model(const char * model_name);	

4.2.3 图的操作

4.2.3.1 create_runtime_graph

接口名称	create_runtime_graph
接口说明	创建运行图。

参数	const char * graph_name	运行图名称。例如：“graph”等。
	const char * model_name	模型名称。在load_model()时指定的名称。
	workspace_t ws	工作空间句柄；可以传入NULL，使用默认工作空间。
返回值	graph_t	运行图的句柄。失败时返回NULL
接口声明	graph_t create_runtime_graph(const char * graph_name, const char * model_name, workspace_t ws);	

4.2.3.2 destroy_runtime_graph

接口名称	destroy_runtime_graph	
接口说明	销毁运行图。	
参数	graph_t graph	运行图的句柄。
返回值	int	0：表示成功。 -1：表示失败。
接口声明	int destroy_runtime_graph(graph_t graph);	

4.2.3.3 prerun_graph

接口名称	prerun_graph	
接口说明	图的预运行。为图的运行准备资源	
参数	graph_t graph	运行图的句柄。
返回值	int	0：表示成功。 -1：表示失败。
接口声明	int prerun_graph(graph_t graph);	

4.2.3.4 run_graph

接口名称	run_graph
接口说明	运行图。

参数	graph_t graph	运行图的句柄。
	int block	0：需要调用wait_graph来等待图运行结束 1：阻塞运行。
返回值	int	0 ： 表示成功。 -1 ： 表示失败。
接口声明	int run_graph(graph_t graph, int block);	

4.2.3.5 postrun_graph

接口名称	postrun_graph	
接口说明	释放运行图的资源。	
参数	graph_t graph	运行图的句柄。
返回值	int	0 ： 表示成功。 -1 ： 表示失败。
接口声明	int postrun_graph(graph_t graph);	

4.2.4 张量的操作

4.2.4.1 get_graph_tensor

接口名称	get_graph_tensor	
接口说明	通过张量名称获取成张量句柄。	
参数	graph_t graph	运行图的句柄。
	const char * tensor_name	张量名称
返回值	tensor_t	张量句柄，失败时返回NULL。 需要调用put_graph_tensor来释放句柄。
接口声明	tensor_t get_graph_tensor(graph_t graph, const char * tensor_name);	

4.2.4.2 get_graph_input_tensor

接口名称	get_graph_input_tensor	
接口说明	获取图的输入节点的张量句柄。	
参数	graph_t graph	运行图的句柄。
	int input_node_idx	输入节点索引值。
	int tensor_idx	张量索引值。
返回值	tensor_t	张量句柄，失败时返回NULL 需要调用put_graph_tensor来释放句柄。
接口声明	tensor_t get_graph_input_tensor(graph_t graph, int input_node_idx, int tensor_idx);	

4.2.4.3 get_graph_output_tensor

接口名称	get_graph_output_tensor	
接口说明	获取图的输出节点的张量句柄。	
参数	graph_t graph	运行图的句柄。
	int output_node_idx	输出节点索引值。
	int tensor_idx	张量索引值。
返回值	tensor_t	张量句柄，失败时返回NULL 需要调用put_graph_tensor来释放句柄。
接口声明	tensor_t get_graph_output_tensor(graph_t graph, int output_node_idx, int tensor_idx);	

4.2.4.4 put_graph_tensor

接口名称	put_graph_tensor	
接口说明	释放张量句柄。	
参数	tensor_t tensor	张量句柄。
返回值	无	
接口声明	void put_graph_tensor(tensor_t tensor);	

4.2.4.5 set_tensor_shape

接口名称	set_tensor_shape	
接口说明	设置张量形状。	
参数	tensor_t tensor	张量句柄。
	int dims[]	表示维度的int数组。
	int dim_number	维度的个数。
返回值	无	
接口声明	int set_tensor_shape(tensor_t tensor, int dims[], int dim_number);	

4.2.4.6 get_tensor_shape

接口名称	get_tensor_shape	
接口说明	获取张量形状。	
参数	tensor_t tensor	张量句柄。
	int dims[]	表示维度的int数组
	int dim_number	维度的个数。
返回值	int	大于等于1表示维度的个数，-1表示获取失败。
接口声明	int get_tensor_shape(tensor_t tensor, int dims[], int dim_number);	

4.2.4.7 set_tensor_buffer

接口名称	set_tensor_buffer	
接口说明	设置张量数据缓存区。	
参数	tensor_t tensor	张量句柄。
	void * buffer	数据缓存区指针。该内存仍然由调用者负责释放。
	int buffer_size	缓存区大小。
返回值	无	

接口声明	int set_tensor_buffer(tensor_t target_tensor, void * buffer, int buffer_size);

4.2.4.8 get_tensor_buffer

接口名称	get_tensor_buffer	
接口说明	获取张量数据缓存区。	
参数	tensor_t tensor	张量句柄。
返回值	void *	数据缓存区指针。NULL表示数据缓存区还未申请。
接口声明	void * get_tensor_buffer(tensor_t tensor);	

5 使用示例

5.1 classificatioin 示例程序

1. 初始化 Tengine 库，并判断当前的 Tengine 库是否兼容“0.7”的 Tengine 版本。

```
    // init tengine
    init_tengine_library();
    if (request_tengine_version("0.7") < 0)
        return false;
```

2. 从文件加载模型。

   Tengine 支持 caffe/onnx/tensorflow/Tengine 格式模型的直接加载。

   caffe 格式的模型只支持新 caffe.proto 定义的格式。如果是旧格式，请用 caffe 包里面的 **upgrade_net_proto_binary/ upgrade_net_proto_binary** 升级模型。

   Tensorflow/ONNX 模型只支持单个 Binary PB 文件。

```
    /* model format should be "caffe", "tensorflow", "onnx", "tengine" */

    // load model
    if (model_fnum == 2)
    {
        if (load_model(model_name.c_str(), model_format, proto_file, model_file)
< 0)
```

```
        return false;
    }
    else // model_fnum == 1
    {
        if (load_model(model_name.c_str(), model_format, model_file) < 0)
            return false;
    }
    std::cout << "Load model done.\n";
```

3. 创建一个运行图

```
    // create graph
    graph_t graph = create_runtime_graph("graph", model_name.c_str(), NULL);
    if (!graph)
    {
        std::cerr << "Create graph0 failed.\n";
        return false;
    }
```

4. 获取图的输入张量，并设置张量的形状。此步骤可以省略，如果从模型中可以获取输入张量的形状。

```
    // input
    int img_size = img_h * img_w * 3;
    int dims[] = {1, 3, img_h, img_w};
    float *input_data = (float *)malloc(sizeof(float) * img_size);

    tensor_t input_tensor = get_graph_input_tensor(graph, 0, 0);
    set_tensor_shape(input_tensor, dims, 4);
```

5. 预运行。

```
    // prerun
    prerun_graph(graph);
```

6. 设置输入数据和张量缓存数据区，运行图。

```
    /* run */
    get_input_data(image_file, input_data, img_h, img_w, mean, scale);
    set_tensor_buffer(input_tensor, input_data, img_size * 4);

    run_graph(graph, 1);
```

7. 获取输出数据，并打印结果。

```
     // print output
    tensor_t output_tensor = get_graph_output_tensor(graph, 0, 0);
    float *data = (float *)get_tensor_buffer(output_tensor);
```

```
    PrintTopLabels(label_file, data);
```

8. 释放资源。

```
    put_graph_tensor(output_tensor);
    put_graph_tensor(input_tensor);

    free(input_data);
    postrun_graph(graph);
    destroy_runtime_graph(graph);
    remove_model(model_name);
```