SDK 开发指南

文档版本 01

发布日期 2019-11-04

南京天数智芯科技有限公司

商标声明

和其他天数商标均为南京天数智芯科技有限公司的商标。

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

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，南京天数智芯科技有限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。

南京天数智芯科技有限公司

地址：　南京市雨花台区宁双路 19 号云密城 2 号楼第 11 层

电话：　025-52600501

目录

修订记录

文档版本 V1.0.0

文档版本	发布日期	修改说明
V1.0.0	2019-11-04	第一次初稿。
V1.0.0	2019-11-06	第二次修改，内容如下： ● 补充 TFlex 的功能和接口介绍。 ● 删除原来的 3.4/3.5/3.6 章节。

1 SDK 简介

Iluvatar CoreX I SDK 包含 Iluvatar CoreX I PCIe Driver、TensorFlow-TFlex、TFlex 三个部分，但上层真正提供 Iluvatar CoreX I 相关使用接口的其实就只有 TFLEX 部分。

2 开发接口

2.1 Iluvatar CoreX I PCIe Driver

Iluvatar CoreX I 芯片的 Linux kernel driver，用于提供访问 Iluvatar CoreX I 的设备接口。

输入参数

无

返回

无

使用举例

无

2.2 TensorFlow-TFlex

基于 tensorflow 加入 Iluvatar CoreX I device 支持的推理框架。

输入参数

无

返回

无

使用举例

无

2.3 TFlex

基于 tensorflow-tflex 之上提供 Iluvatar CoreX I 模型转换、查看、导入等功能。

2.3.1 概述

TFlex 是嵌入式设备上的边缘推理库，其中包含由 iluvatar.ai 设计的 Edge Iluvatar CoreX I 协处理器。它是为需要深度学习（DL）模型的设备上快速推理的新项目制作原型的理想选择。

为了方便起见，TFlex 库提供了三个命令行工具：tflexconverter，tflexviewer 和 tflexverify，用于转换，查看和验证 Iluvatar CoreX I 支持的 DL 模型。

- tflexconverter，它将 SavedModel/.pb/.h5 模型转换为 Iluvatar CoreX I 支持的.tflex 模型。
- 基于 Tensorboard 的 tflexviewer 可以更直观地显示神经网络体系结构（均支持.pb 和.tflex 文件）。
- tflexverify 验证模型是否成功转换，并以 PDF 格式返回比较推理结果。

2.3.2 工作流程

使用 TFlex 库的开发工作流程涉及以下步骤。

安装

通过在 Linux（x86_64）/ MacOS / Wins 上使用 python-pip3 可以轻松安装 TFlex，并且将自动安装自定义的 tensorflow-tflex 库。

- X86 系统下：安装预发布的 tflex。

```
# pip3 install --pre tflex
```

此外，如果要安装在 ARM（例如 aarch64）系统上，由于 pypi.org 上的版本不受支持，因此需要分别安装 tensorflow-tflex 和 tflex。

- ARM 系统下：安装 tensorflow-tflex 和 tflex

首先下载基于 ARM 的 tensorflow-tflex 软件包。

下载链接：https://github.com/SkyAI/tensorflow-tflex/releases/tag/v1.13.1-rc2

```
# pip3 install tensorflow_tflex-1.13.1rc1-cp27-cp27mu-linux_aarch64.whl
# pip3 install --pre tflex
```

转换

准备好预训练或定制的模型后，开发人员可以直接使用 tflexconverter 命令行工具（或 Python API）将模型转换为 Iluvatar CoreX I 格式（即.tflex 模型）。

推理

将转换后的模型部署到包含 Edge Iluvatar CoreX I 的嵌入式设备，并实现 Edge 推理。

转换

TFlexconverter 用于将基本的 32 位浮点模型从每种受支持的数据格式转换为 Iluvatar CoreX I 支持的 16 位浮点模型，提供了自定义类 Converter 来使用。

Converter 接受以下文件格式：

- Frozen tf.GraphDef —包含变量的 tf.GraphDef 的子类。
- from tf.Session —从 tf.Session 获取的任何模型。
- SavedModel —带有签名的 GraphDef 和检查点，用于标记模型的输入和输出参数。
- tf.keras —一个 HDF5 文件，其中包含具有权重，tf.keras 生成的输入，输出参数的模型。

2.3.2.1 Python API

TFlex 提供了四种方法来将模型的不同原始格式初始化为.tflex 模型。并且为实际转换提供了 convert（）方法。

2.3.2.1.1 from_frozen_graph

从包含固化的 GraphDef 的文件中创建 Converter 类。

```
@classmethod
def from_frozen_graph(cls,
                      graph_def_file,
                      input_arrays,
                      output_arrays,
                      input_shapes=None):
```

【参数描述】

- graph_def_file：包含固化的 GraphDef 的文件的路径。
- input_arrays：固化图的输入张量的列表。
- output_arrays：固化图的输出张量列表。
- input_shapes：表示输入张量名称的字符串的字典到表示输入形状的整数列表（例如{“ input”：[1、229、229、3]}）。

【返回】

转换器类。

【结果】

- IOEorror：找不到文件。 无法解析输入文件。
- ValueError：
  - 该图未固化。
  - input_arrays 或 output_arrays 不存在或包含无效的张量名称。
  - 必要时未正确定义 input_shapes。

2.3.2.1.2 from_session

从 TensorFlow 会话创建 Converter 类。

```
@classmethod
def from session(cls,
                sess,
                input tensors,
                output_tensors):
```

【参数描述】

- sess：Tensorflow 会话。
- input_tensors：输入张量的列表。
- output_tensors：输出张量的列表。

【返回】

转换器类。

2.3.2.1.3 from_saved_model

从 SavedModel 创建 Converter 类。

```
@classmethod
def from saved model(cls,
                    saved model dir,
                    input arrays=None,
                    input shapes=None,
                    output_arrays=None):
```

【参数描述】

- saved_model_dir：要转换的 SavedModel 目录。

- input_arrays：固化图的输入张量的列表。
- input_shapes：表示输入张量名称列出表示输入形状整数串的字典（例如，{ “输入”：[1，299，299，3]}）。 输入形状为“无”时自动确定。
- output_arrays：固化图的输出张量列表。

【返回】

转换器类。

2.3.2.1.4 from_keras_model

从 tf.keras 模型文件创建 Converter 类。

```
@classmethod
def from keras model(cls,
                model file,
                input arrays=None,
                input shapes=None,
                output_arrays=None):
```

【参数描述】

- model_file：包含 tf.keras 模型的 HDF5 文件的路径。
- input_arrays：固化图的输入张量的列表。
- input_shapes：表示输入张量名称列出表示输入形状整数串的字典（例如，{ “输入”：[1，299，299，3]}）。 输入形状为“无”时自动确定。
- output_arrays：固化图的输出张量列表。

【返回】

转换器类。

2.3.2.1.5 convert

根据实例变量转换一个 tensorflow graphdef。

```
def convert(self, save_path, device='/device:Iluvatar CoreX I:0', level=4,
strict padding=False,
          start_node_names=None, end_node_names=None):
```

【参数描述】

以序列化格式转换的数据。

- save_path：保存成功优化的 Graphdef 的路径名，例如 models / mobilenet.tflex。
- 设备：分配给指定操作的 EPU 设备，例如 Conv2D，MaxPool，Pad 等。默认值为/device：Iluvatar CoreX I：0。

- 级别：选择 5 种最佳组合级别。例如，level = 1：基础优化；level = 2：基础优化和批归一化；级别= 3：基础优化，批归一化和 Iluvatar CoreX I 核心优化；级别= 4：基础优化，批归一化，Iluvatar CoreX I 核心和 Iluvatar CoreX I 高级优化；级别 = 5：基础优化，批归一化，Iluvatar CoreX I 核心，Iluvatar CoreX I 高级和其他优化。默认级别= 4 表示将执行前四个优化。每个优化的具体实现如下所示。

基础优化	批归一化	Iluvatar CoreX I 核心优化	Iluvatar CoreX I 高级优化	附加优化
placeholder_modify strip_unused_reserve_input_node_shape remove_training_nodes append_identity_output fold_multiple_relus standardize_leakyRelu clear_map standardize_concat	standardize_batchNorm merge_bias_BN fold_batch_norms	merge_pad replace_matmul_to_conv2d add_conv2D_before_BN add_to_bias remove_add_ensue_bias fold_conv2d_bias_relu fuse_second_input_ops_2 pad_depthwise pad_slice_conv2d2 reshape_conv2d_weights move_const_to_epu fp32_to_fp16	replace_avgpool_to_conv2d replace_conca_v1 replace_depthwise_to_conv2d	make_op_async

- strict_padding：Iluvatar CoreX I MaxPool 仅支持填充为 SAME 或 VALID（W / H 中没有剩余）。如果设置为 True，它将在 CPU 上执行。
- start_node_names：将起始节点设置为用于优化的初始输入节点，以灵活地控制优化范围内的起始位置。
- end_node_names：将结束节点设置为最优化的最后一个输出节点，以灵活地控制优化范围内的结束位置。

【返回值】

- 优化了 graphdef。
- .tflex 文件，包含序列化的优化 graphdef。
- map.txt 文件，用于 tflexverify 来验证模型转换的正确性。

【举例】

以下代码显示了如何在 Python 中使用上述 API。

```
import tflex

# Converting a GraphDef from session.
converter = tflex.Converter.from session(sess, input tensors, output tensors)
tflex model = converter.convert(save path, device='/device:Iluvatar CoreX I:1') //
output model.tflex file simultaneously.

# Converting a GraphDef from file.
converter = tflex.Converter.from frozen graph(graph def file, input arrays,
output arrays)
tflex model = converter.convert(save path, device='/device:Iluvatar CoreX I:1') //
output model.tflex file simultaneously.

# Converting a SavedModel.
converter = tflex.Converter.from saved model(saved model dir)
tflex model = converter.convert(save path, device='/device:Iluvatar CoreX I:1') //
output model.tflex file simultaneously.

# Converting a tf.keras model.
converter = tflex.Converter.from keras model(keras model)
tflex model = converter.convert(save path, device='/device:Iluvatar CoreX I:1') //
output model.tflex file simultaneously.
```

2.3.2.2 命令行工具

2.3.2.2.6 tflexconverter

为方便起见，TFlex 还提供了 tflexconverter 命令行工具来在终端中转换模型。以下示例介绍了 tflexconverter 命令行工具的参数，并指导您在终端中使用 tflexconverter 将固化的 graph_def mobilenet_v1_1.0_224_frozen.pb 转换为 Iluvatar CoreX I 支持的 mobilenet.tflex。

```
$ tflexconverter -h
Using TensorFlow backend.
usage: tflexconverter.py [-h] [--keras model KERAS MODEL]
                    [--frozen model FROZEN MODEL]
                    [--saved model SAVED MODEL] [--save path SAVE PATH]
                    [--input arrays INPUT ARRAYS]
                    [--output arrays OUTPUT ARRAYS] [--device DEVICE]
                    [--level LEVEL] [--strict padding]
                    [--start node names START NODE NAMES]
                    [--end node names END NODE NAMES]

Convert source model(.pb,.h5,SavedModel) to target model(.tflex graph)
supported on Iluvatar CoreX I.

optional arguments:
  -h, --help          show this help message and exit
  --keras_model KERAS_MODEL, -k KERAS_MODEL
```

```
                    Source model with .h5 file to be converted.
  --frozen model FROZEN MODEL, -f FROZEN MODEL
                    Source model with .pb file to be converted.
  --saved model SAVED MODEL, -s SAVED MODEL
                    Source SavedModel with .pb file and variables to be
                    converted.
  --save path SAVE PATH, -p SAVE PATH
                    Pathname to save the optimized graph(e.g.,
                    ./model.tflex).
  --input arrays INPUT ARRAYS, -i INPUT ARRAYS
                    String of input node names. If your model has more
                    inputs, please use tflexconverter -i input 1,input 2.
  --output arrays OUTPUT ARRAYS, -o OUTPUT ARRAYS
                    String of output node names. If your model has more
                    outputs, please use tflexconverter -o
                    output 1,output 2.
  --device DEVICE, -d DEVICE
                    Iluvatar CoreX I devices assigned to the Conv2D, MaxPool and
Pad
                    ops, default is /device:Iluvatar CoreX I:0.
  --level LEVEL, -l LEVEL
                    Selection of 5 optimal combination levels. For
                    example, `level=1`: fundamental
                    optimization;`level=2`: fundamental and
                    batchnormalization optimization; `level=3`:
                    fundamental, batchnormalization and Iluvatar CoreX I core
                    optimization; `level=4`: fundamental,
                    batchnormalization, Iluvatar CoreX I core and Iluvatar CoreX I
Advanced
                    optimization; `level=5`: fundamental,
                    batchnormalization, Iluvatar CoreX I core, Iluvatar CoreX I
Advanced and
                    additional optimization. Default `level=4` means that
                    the first four optimizations will be executed.
  --strict padding, -r  Iluvatar CoreX I MaxPool only support padding is SAME or
                    VALID(without remnant in W/H). If True is set, it will
                    execute on the CPU.
  --start node names START NODE NAMES, -t START NODE NAMES
                    The starting nodes are set as the initial input nodes
                    for optimization to flexibly control the starting
                    position in the scope of optimization.
  --end node names END NODE NAMES, -e END NODE NAMES
                    The ending nodes are set as the last output nodes for
                    optimization to flexibly control the ending position
                    in the scope of optimization.

$ tflexconverter -f mobilenet/mobilenet_v1_1.0_224_frozen.pb \
             -i input \
             -o MobilenetV1/Predictions/Reshape_1 \
             -p mobilenet/mobilenet.tflex \
             -d /device:Iluvatar CoreX I:1 \
             2>&1 | tee mobilenet/logs/mobilenet.log
```

为了验证您的模型是否成功转换，TFlex 还提供了两个附加的命令行工具：tflexviewer 和 tflexverify。

2.3.2.2.7 tflexviewer

tflexviewer 用于基于张量板可视化.pb 或.tflex 模型的体系结构。以下示例是 tflexviewer 命令行工具的格式和参数介绍，并指导您在终端中使用 tflexviewer 可视化 mobilenet.tflex 的结构。

```
$ tflexviewer -h
Using TensorFlow backend.
usage: tflexviewer.py [-h] --graph GRAPH [--logdir LOGDIR] [--depth DEPTH]
                  [--port PORT]

Visualization of deep learning models(.pb and .tflex file are supported).

optional arguments:
  -h, --help          show this help message and exit
  --graph GRAPH, -g GRAPH
                      Import protobuf graphDef file (.pb or .tflex file) or
                      model dir(with pb or tflex files inside) to the
                      tensorboard.
  --logdir LOGDIR, -l LOGDIR
                      Log directory specified by user to save tensorboard
                      logs.
  --depth DEPTH, -d DEPTH
                      Recursion depth for model dir.
  --port PORT, -p PORT  Port to serve TensorBoard on, default port is 6006.

$ tflexviewer -g mobilenet/mobilenet.tflex \
           -l mobilenet/logDir
```

此外，如果为-g 参数指定目录 mobilenet /，则 tflexviewer 将递归搜索该目录中的所有模型，默认递归深度为 3。如果在指定目录中找到多个模型（例如，mobilenet_v1_1.0_224_frozen.pb 和 mobilenet）。tflex 在目录 mobilenet /中被观察到，将基于张量板查看以下多个模型。

```
$ tflexviewer -g mobilenet/ \
           -l mobilenet/logDir \
           -d 3
```

2.3.2.2.8 tflexverify

tflexverify 用于验证转换是否正确，并以 PDF 格式返回源.pb 模型和优化的.tflex 模型之间的比较推理结果。以下示例展示了如何使用 tflexverify 命令行工具。

```
$ tflexverify -h
Using TensorFlow backend.
usage: tflexverify.py [-h] [--frozen model FROZEN MODEL]
                  [--optimized model OPTIMIZED MODEL]
                  [--map file MAP FILE] [--image file IMAGE FILE]

Verify the correctness of the source frozen model(.pb) conversion.
```

```
optional arguments:
  -h, --help          show this help message and exit
  --frozen model FROZEN MODEL, -f FROZEN MODEL
                      Source frozen model with .pb file.
  --optimized model OPTIMIZED MODEL, -o OPTIMIZED MODEL
                      Target optimized model with .tflex file.
  --map file MAP FILE, -m MAP FILE
                      The map.txt file generated automatically during model
                      conversion process.
  --image file IMAGE FILE, -i IMAGE FILE
                      An image (e.g., ILSVRC2012 val 00013716.JPEG) is
                      required for inference.

$ tflexverify -f mobilenet/mobilenet v1 1.0 224 frozen.pb \
           -o mobilenet/mobilenet.tflex \
           -m map.txt \
           -i ILSVRC2012_val_00013716.JPEG
```

2.3.3 Inference

完成上述步骤后，您现在应该拥有一个.tflex 文件，可以在 Iluvatar CoreX I 设备上进行推理。基于 EPU 的设备上的模型推理通常遵循以下步骤：

- 图导入：从.tflex 文件导入推理模型的执行图以创建会话。
- 输入/输出张量获取：根据导入的图形获取输入和输出张量。
- 模型推理：针对特定应用执行模型并从模型推理中检索结果。例如，模型可以返回概率列表。应用程序开发人员应将它们有意义地映射到相关类别，然后将其呈现给用户。

2.3.3.1 Python API

TFlex 提供了 utils 模块，包括 import_graph，get_tensors，set_device api，以帮助开发人员方便地实现模型推理。详细的接口定义如下所述：

2.3.3.1.1 import_graph

导入由 tflexconverter 命令行工具或 Python API 转换的优化的 graphdef。

```
def import_graph(tflex_file, device=None):
```

【参数描述】

- tflex_file：包含 graphdef 的.tflex 文件的完整文件路径已成功转换。
- device：可以将指定的操作（例如 Conv2DEPU，MaxPoolEPU，PadEPU 等）重置为用户指定的新设备名称。

【返回值】

- 图。
- 输入张量的名称列表。
- 输出张量的名称列表。

2.3.3.1.2 get_tensors

返回张量列表。

```
def get_tensors(graph, tensor_names):
```

【参数描述】

- graph：要启动的图。
- device：用户指定的设备名称。

【返回值】

原始图形，其中 op.device 已被重置，包括 Conv2DEPU，MaxPoolEPU，PadEPU 等。

【举例】

以下是使用 mobilenet.tflex 进行图像识别的示例，该模型基于 TFlex 和定制的 tensorflow-tflex 库分析了模型推理过程。

```
import tflex
import tensorflow as tf

# import graph from `.tflex` model file converted successfully by `tflexconverter`
or python api.
r_graph, input_arrays, output_arrays = tflex.utils.import_graph("mobilenet.tflex")

# if you want to inference on `/device:Iluvatar CoreX I:1`, default is
`/device:Iluvatar CoreX I:0`.
r_graph = tflex.utils.set_device(r_graph, '/device:Iluvatar CoreX I:1')

# obtain input/output tensors.
r_input = tflex.utils.get_tensors(r_graph, input_arrays)
r_output = tflex.utils.get_tensors(r_graph, output_arrays)

# Execute inference.
with tf.Seesion(graph=r_graph) as sess:
   # data is a list, which has the same length as the r_input.
   data = [np.array(img1), np.array(img2),...]
   sess.run(r_output, feed_dict={i : d for i, d in zip(r_input, data)})
```

另外，TFlex 库支持多线程并行推理，双重 EPU 推理示例实现如下。

```
import tflex
```

```
import tensorflow as tf

def main( ):
   # import graph from `mobilenet.tflex` model file.
   graph mobilenet, input arrays, output arrays =
tflex.utils.import graph("mobilenet.tflex",
                                                        device='/device:Iluvatar
CoreX I:0')
   # obtain input/output tensors.
   input mobilenet = tflex.utils.get tensors(graph mobilenet, input arrays)
   output mobilenet = tflex.utils.get tensors(graph mobilenet, output arrays)

   # Create session for graph mobilenet.
   sess mobilenet = tf.Session(graph=graph mobilenet)

   # import graph from `resnet.tflex` model file
   graph resnet, input arrays, output arrays =
tflex.utils.import graph("resnet.tflex",
                                                        device='/device:Iluvatar
CoreX I:1')
   # obtain input/output tensors.
   input resnet = tflex.utils.get tensors(graph resnet, input arrays)
   output resnet = tflex.utils.get tensors(graph resnet, output arrays)
   # Create session for graph resnet.
   sess resnet = tf.Session(graph=graph resnet)

   # Thread-1 for mobilenet inference on '/device:Iluvatar CoreX I:0'.
   def epu0 inference(sess mobilenet, input mobilenet, output mobilenet):
      data = [np.array(img1), np.array(img2),...]
      sess  mobilenet.run(output  mobilenet, feed dict={i : d for i, d in
zip(input  mobilenet, data)})

   # Thread-2 for resnet inference on '/device:Iluvatar CoreX I:1'
   def epu1 inference(sess resnet, input resnet, output resnet):
      data = [np.array(img1), np.array(img2),...]
      sess resnet.run(output resnet, feed dict={i : d for i, d in
zip(input resnet, data)})

   # Thread-1 Creation.
   t1 = threading.Thread(target=epu0 inference)
   # Thread-2 Creation.
   t2 = threading.Thread(target=epu1 inference)
   t1.start()
   t2.start()
   t1.join()
   t2.join()

if __name__ == '__main__':
   tf.app.run()
```