

CANN  
6.3.RC2

# TensorFlow 2.6.5 在线推理指南

文档版本	01
发布日期	2023-07-28



**版权所有 © 华为技术有限公司 2023。保留一切权利。**

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

## **商标声明**



HUAWEI和其他华为商标均为华为技术有限公司的商标。

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、服务或特性等应受华为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，华为公司对本文档内容不做任何明示或暗示的声明或保证。

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。

---

# 目 录

---

1 概述.....	1
2 环境准备.....	2
3 迁移示例.....	3
4 完整代码示例.....	6
5 参考.....	8
5.1 安装 7.3.0 版本 gcc.....	8

# 1 概述

---

## 简介

在线推理是在AI框架内执行推理的场景，例如在Tensorflow框架上，加载模型后，通过`tf.compat.v1 session.run`执行推理。相比于离线推理场景，使用在线推理可以方便将原来基于Tensorflow 2.6.5框架做推理的应用快速迁移到昇腾AI处理器，适用于数据中心推理场景。

## 读者对象

本文档适用于将训练好的TensorFlow网络模型加载到昇腾AI处理器进行在线推理的AI工程师。

掌握以下经验和技能可以更好地理解本文档：

- 熟练的Python语言编程能力
- 熟悉TensorFlow API
- 对机器学习、深度学习有一定的了解

## 支持的芯片型号

昇腾910 AI处理器

昇腾910B AI处理器

# 2 环境准备

请参见《[CANN 软件安装指南](#)》进行环境搭建。

- 请依据实际在下列场景中选择一个进行在线推理依赖包安装路径的环境变量设置。具体如下（以HwHiAiUser安装用户为例）：
  - 场景一：昇腾设备安装部署开发套件包Ascend-cann-toolkit(此时开发环境可进行推理任务)。  

```
./home/HwHiAiUser/Ascend/ascend-toolkit/set_env.sh
```
  - 场景二：昇腾设备安装部署软件包Ascend-cann-nnae。  

```
./home/HwHiAiUser/Ascend/nnae/set_env.sh
```
- 设置tfplugin插件包的环境变量。  

```
./home/HwHiAiUser/Ascend/tfplugin/set_env.sh
```
- 若运行环境中存在多个python3版本时，需要在环境变量中配置python的安装路径。如下配置以安装python3.7.5为例，可根据实际修改。  

```
export PATH=/usr/local/python3.7.5/bin:$PATH  
export LD_LIBRARY_PATH=/usr/local/python3.7.5/lib:$LD_LIBRARY_PATH
```

## 说明

若所在系统环境需要升级gcc（例如Centos、Debian和BCLinux系统），则“LD\_LIBRARY\_PATH”配置项此处动态库查找路径需要添加“\${install\_path}/lib64”，其中“{install\_path}”为gcc升级安装路径。请参见[5.1 安装7.3.0版本gcc](#)。

# 3 迁移示例

下面，我们以ResNet50模型为例，介绍如何将基于TensorFlow 2.6.5框架和CPU/GPU环境运行的在线推理应用迁移到昇腾AI处理器上。

## 迁移前准备

准备基于TensorFlow 2.6.5框架的CPU/GPU在线推理代码和数据集，并在CPU/GPU环境跑通。

## 熟悉 CPU/GPU 在线推理流程

基于TensorFlow 2.6.5框架的在线推理代码主要包括：

1. 准备resnet50.pb模型、输入节点、输出节点、数据集。
2. 调用sess.run()执行推理。sess.run()中的feed\_dict的作用是给使用placeholder创建出来的tensor赋值，我们可以提供feed（输入）数据，作为run()调用的参数。

关键推理代码为：

```
def load_graph(frozen_graph):
    with tf.io.gfile.GFile(frozen_graph,"rb") as f:
        graph_def = tf.compat.v1.GraphDef()
        graph_def.ParseFromString(f.read())

    with tf.Graph().as_default() as graph:
        tf.import_graph_def(graph_def,name='')
    return graph

def NetworkRun(modelPath,inputPath,outputPath):
    graph = load_graph(modelPath)
    input_nodes = graph.get_tensor_by_name('Input:0')
    output_nodes = graph.get_tensor_by_name('Identity:0')

    with tf.compat.v1.Session(graph=graph) as sess:
        files = os.listdir(inputPath)
        files.sort()
        for file in files:
            if file.endswith(".bin"):
                input_img = np.fromfile(inputPath+"/"+file,dtype="float32").reshape(1,224,224,3)
                t0 = time.time()
                out = sess.run(output_nodes, feed_dict= {input_nodes: input_img,})
                t1 = time.time()
                out.tofile(outputPath+"/"+ "cpu_out "+file)
                print("%s, Inference time: %.3f ms".format(file,(t1-t0)*1000))
```

## 推理脚本迁移到昇腾 AI 处理器

1. 迁移到昇腾AI处理器需要引入NPU的相关配置库。

```
import npu_device
from npu_device.compat.v1.npu_init import *
npu_device.compat.enable_v1()
from tensorflow.core.protobuf.rewriter_config_pb2 import RewriterConfig
```

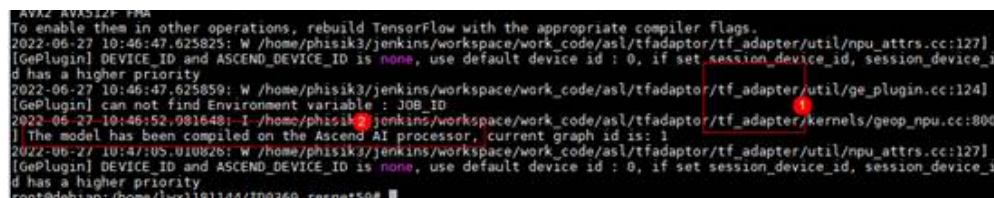
2. 在sess.run之前增加NPU的相关config配置。

```
config_proto = tf.compat.v1.ConfigProto()
custom_op = config_proto.graph_options.rewrite_options.custom_optimizers.add()
custom_op.name = "NpuOptimizer"
custom_op.parameter_map["precision_mode"].s = tf.compat.as_bytes("allow_mix_precision")
config_proto.graph_options.rewrite_options.remapping = RewriterConfig.OFF
tf_config = npu_config_proto(config_proto=config_proto)
```

## 查看是否迁移成功

在昇腾AI处理器执行迁移后的在线推理脚本，其执行成功的标志和训练成功打印一致，方法包括：

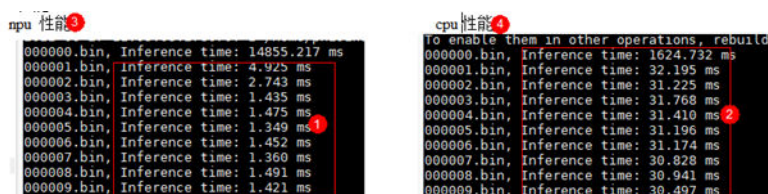
1. 出现tf\_adapter和The model has been compiled on the Ascend AI processor的关键字打印。



2. 或者打开Dump计算图开关DUMP\_GE\_GRAPH，看能否产生Dump计算图。

## 检查推理性能和精度

推理性能是通过sess.run()前后时间打点差值来计算的。从本样例推理结果看，npu的性能远优于cpu的性能。



推理精度是将输出bin文件转成txt进行比较，从本样例推理结果可以看出npu与cpu精度差别不大。

2022/6/22 11:12:17	2022/6/22 11:12:17
0.000046	0.000046
0.000478	0.000475
0.000170	0.000170
0.000132	0.000132
0.000342	0.000346
0.000110	0.000110
0.000074	0.000074
0.000260	0.000257
0.000039	0.000039
0.000685	0.000684
0.002399	0.002396
0.002861	0.002827
0.000782	0.000774
0.000612	0.000607
0.000384	0.000380
0.000384	0.000382
0.001260	0.001254
0.000292	0.000289
0.001643	0.001645
0.000614	0.000603
0.000985	0.000990
0.038574	0.039103
0.005112	0.005170
0.003851	0.003905
0.000743	0.000745
0.000099	0.000098
0.000205	0.000204
0.000492	0.000488
0.000189	0.000188



# 4 完整代码示例

迁移到昇腾AI处理器的推理脚本：

```
import npu_device
from npu_device.compat.v1.npu_init import *
npu_device.compat.enable_v1()
from tensorflow.core.protobuf.rewriter_config_pb2 import RewriterConfig

import tensorflow as tf
import numpy as np
from tensorflow.python.profiler import model_analyzer
from tensorflow.python.profiler import option_builder
from tensorflow.python.client import timeline
import os
import time
import argparse

# np.random.seed(10)

def load_graph(frozen_graph):
    with tf.io.gfile.GFile(frozen_graph, "rb") as f:
        graph_def = tf.compat.v1.GraphDef()
        graph_def.ParseFromString(f.read())

    with tf.Graph().as_default() as graph:
        tf.import_graph_def(graph_def, name='')
    return graph

def NetworkRun(modelPath, inputPath, outputPath):
    graph = load_graph(modelPath)
    input_nodes = graph.get_tensor_by_name('Input:0')
    output_nodes = graph.get_tensor_by_name('Identity:0')
    #适配npu
    config_proto = tf.compat.v1.ConfigProto()
    custom_op = config_proto.graph_options.rewrite_options.custom_optimizers.add()
    custom_op.name = "NpuOptimizer"
    custom_op.parameter_map["precision_mode"].s = tf.compat.as_bytes("allow_mix_precision")
    config_proto.graph_options.rewrite_options.remapping = RewriterConfig.OFF
    tf_config = npu_config_proto(config_proto=config_proto)

    with tf.compat.v1.Session(config=tf_config, graph=graph) as sess:
        files = os.listdir(inputPath)
        files.sort()
        for file in files:
            if file.endswith(".bin"):
                input_img = np.fromfile(inputPath+"/"+file, dtype="float32").reshape(1, 224, 224, 3)
                t0 = time.time()
                out = sess.run(output_nodes, feed_dict= {input_nodes: input_img,})
                print('out---', out)
                t1 = time.time()
```

```
        out.tofile(outputPath+"/"+ "cpu_out_" +file)
        #print("%s, Inference time: %.3f ms".format(file,(t1-t0)*1000)

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument("--model", type=str, default="./resnet50_tf2.pb")
    parser.add_argument("--input", type=str, default="./input_bin/")
    parser.add_argument("--output", type=str, default="./npu_output/")
    args = parser.parse_args()
    if not os.path.isdir(args.output):
        os.mkdir(args.output)
    NetworkRun(args.model,args.input,args.output)
```

# 5 参考

## 安装7.3.0版本gcc

### 5.1 安装 7.3.0 版本 gcc

以下步骤请在root用户下执行。

**步骤1** 下载gcc-7.3.0.tar.gz，下载地址为<https://mirrors.tuna.tsinghua.edu.cn/gnu/gcc/gcc-7.3.0/gcc-7.3.0.tar.gz>。

**步骤2** 安装gcc时候会占用大量临时空间，所以先执行下面的命令清空/tmp目录：

```
rm -rf /tmp/*
```

**步骤3** 安装依赖。

centos/bclinux执行如下命令安装。

```
yum install bzip2
```

ubuntu/debian执行如下命令安装。

```
apt-get install bzip2
```

**步骤4** 编译安装gcc。

1. 进入gcc-7.3.0.tar.gz源码包所在目录，解压源码包，命令为：

```
tar -zxvf gcc-7.3.0.tar.gz
```

2. 进入解压后的文件夹，执行如下命令下载gcc依赖包：

```
cd gcc-7.3.0  
./contrib/download_prerequisites
```

如果执行上述命令报错，需要执行如下命令在“gcc-7.3.0/”文件夹下下载依赖包：

```
wget http://gcc.gnu.org/pub/gcc/infrastructure/gmp-6.1.0.tar.bz2  
wget http://gcc.gnu.org/pub/gcc/infrastructure/mpfr-3.1.4.tar.bz2  
wget http://gcc.gnu.org/pub/gcc/infrastructure/mpc-1.0.3.tar.gz  
wget http://gcc.gnu.org/pub/gcc/infrastructure/isl-0.16.1.tar.bz2
```

下载好上述依赖包后，重新执行以下命令：

```
./contrib/download_prerequisites
```

如果上述命令校验失败，需要确保依赖包为一次性下载成功，无重复下载现象。

3. 执行配置、编译和安装命令：

```
./configure --enable-languages=c,c++ --disable-multilib --with-system-zlib --prefix=/usr/local/  
linux_gcc7.3.0  
make -j15 # 通过grep -w processor /proc/cpuinfo|wc -l查看cpu数，示例为15，用户可自行设置相应参  
数。  
make install
```

### 注意

其中“--prefix”参数用于指定linux\_gcc7.3.0安装路径，用户可自行配置，但注意不要配置为“/usr/local”及“/usr”，因为会与系统使用软件源默认安装的gcc相冲突，导致系统原始gcc编译环境被破坏。示例指定为“/usr/local/linux\_gcc7.3.0”。

### 步骤5 配置环境变量。

当用户执行训练时，需要用到gcc升级后的编译环境，因此要在训练脚本中配置环境变量，通过如下命令配置。

```
export LD_LIBRARY_PATH=${install_path}/lib64:${LD_LIBRARY_PATH}
```

其中\${install\_path}为3.中配置的gcc7.3.0安装路径，本示例为“/usr/local/gcc7.3.0/”。

### 说明

本步骤为用户在需要用到gcc升级后的编译环境时才配置环境变量。

----结束