华为昇腾-昇思MindSpore《AI计算系统》精品课程

HCIA-AI计算系统

精品课程

实验指导手册

版本:1.0



华为技术有限公司

|  |
| --- |
| 版权所有 © 华为技术有限公司 2021。 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  C:\Users\jwx341670\Desktop\华为标志 Huawei Logo 2018\竖版标志Vertical Version\PNG\HW_POS_RBG_Vertical-150ppi.png 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。 |

|  |  |
| --- | --- |
| 华为技术有限公司 | |
| 地址： | 深圳市龙岗区坂田华为总部办公楼 邮编：518129 |
| 网址： | http://[e](http://e.huawei.com/).huawei.com |

目录

[3 基于ACL实现图片分类应用 4](#_Toc184544800)

[3.1实验介绍 4](#_Toc678067332)

[3.1.1 数据集的介绍 4](#_Toc1729417942)

[3.1.2 AscendCL介绍 5](#_Toc136189049)

[3.1.3 ATC介绍 5](#_Toc1859262488)

[3.1.4 实验目录结构介绍 6](#_Toc590088319)

[3.2 实验环境要求 6](#_Toc534895587)

[3.3 实验详细设计与实现 7](#_Toc623584367)

[3.3.1 实验步骤 7](#_Toc862258809)

[3.4 实验总结 15](#_Toc764152907)

[3.5 实验任务与参考解答任务 15](#_Toc1165698889)

[3.5.1 实验任务 15](#_Toc407915842)

# 基于ACL实现图片分类应用

## 3.1实验介绍

本实验实现的是基于ACL（Ascend Computing Language）搭建VGG16模型实现图片分类应用。整体流程如下：

* 将ONNX模型转为适配Ascend310推理的OM模型。
* 定义数据预处理模块。
* 定义推理模块。
* 读取给定图片并进行推理，输出各项类别置信度。

### 数据集的介绍

我们示例中用到的图像花卉数据集中的四张图片，分别为雏菊、玫瑰、向日葵和郁金香。数据目录结构如下：

data

├── daisy\_demo.jpg

├── roses\_demo.jpg

├── sunflowers\_demo.jpg

├── tulips\_demo.jpg

玫瑰 雏菊

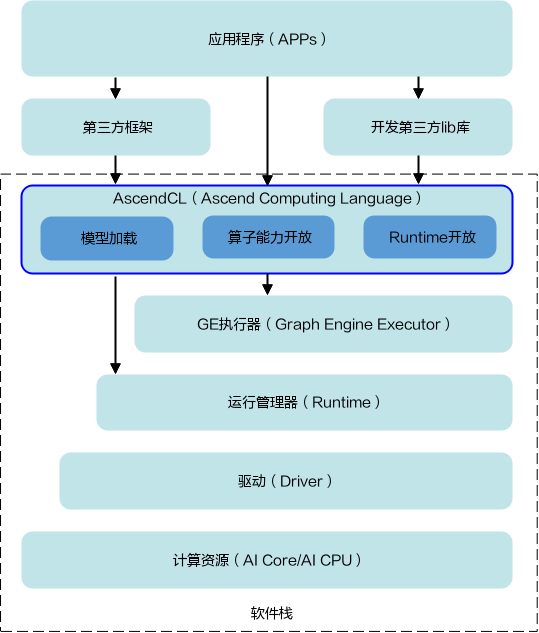
向日葵 郁金香

### AscendCL介绍

AscendCL（Ascend Computing Language）提供Device管理、Context管理、Stream管理、内存管理、模型加载与执行、算子加载与执行、媒体数据处理等C语言API库供用户开发深度神经网络应用，用于实现目标识别、图像分类等功能。用户可以通过第三方框架调用AscendCL接口，以便使用昇腾AI处理器的计算能力；用户还可以使用AscendCL封装实现第三方lib库，以便提供昇腾AI处理器的运行管理、资源管理能力。

在运行应用时，AscendCL调用GE执行器提供的接口实现模型和算子的加载与执行、调用运行管理器的接口实现Device管理/Context管理/Stream管理/内存管理等。

计算资源层是昇腾AI处理器的硬件算力基础，主要完成神经网络的矩阵相关计算、完成控制算子/标量/向量等通用计算和执行控制功能、完成图像和视频数据的预处理，为深度神经网络计算提供了执行上的保障。



### ATC介绍

当前昇腾AI处理器以及昇腾AI软件栈是没有办法直接拿比如Caffe，TensorFlow，Pytorch等开源框架网络模型来直接进行推理的，想要进行推理则需要做一步模型转换的步骤，将开源框架的网络模型转换成Davinci架构专用的模型。而此处模型转换的步骤就是通过本手册所要介绍的ATC工具完成的。

昇腾张量编译器（Ascend Tensor Compiler，简称ATC）是昇腾CANN架构体系下的模型转换工具：

* 它可以将开源框架的网络模型（如Caffe、TensorFlow、Pytorch等）以及Ascend IR定义的单算子描述文件转换为昇腾AI处理器支持的离线模型。
* 模型转换过程中，ATC会进行算子调度优化、权重数据重排、内存使用优化等具体操作，对原始的深度学习模型进行进一步的调优，从而满足部署场景下的高性能需求，使其能够高效执行在昇腾AI处理器上。

### 实验目录结构介绍

EXP

├── data

│ ├── daisy\_demo.jpg // 测试雏菊图片

│ ├── roses\_demo.jpg // 测试玫瑰图片

│ ├── sunflowers\_demo.jpg // 测试向日葵图片

│ ├── tulips\_demo.jpg // 测试郁金香图片

├── model

│ ├── vgg16.onnx // VGG16网络的模型文件(\*.onnx)

│ ├── vgg16.om // VGG16网络的模型文件(\*.om)（待生成）

├── convert\_model.sh // 模型转换的脚本

├── app.py // 运行应用脚本

├── process.py // 将测试图片进行预处理

## 实验环境要求

* 操作系统：Ubuntu X86
* Python版本：Python3.7.5
* 芯片：Ascend310
* 已完成昇腾AI软件栈在开发环境的部署（CANN环境，需要完成驱动及CANN软件的安装，关于CANN环境的安装参考官方文档）

## 实验详细设计与实现

### 实验步骤

导入Python库&模块并配置运行信息

模型转换模块实现ONNX模型转为适配Ascend310推理的OM模型，使之可以高效的运行在Ascend硬件上。

代码：

# 添加环境变量

export install\_path=/usr/local/Ascend/ascend-toolkit/latest

export PATH=/usr/local/python3.7.5/bin:${install\_path}/atc/ccec\_compiler/bin:${install\_path}/atc/bin:$PATH

export PYTHONPATH=${install\_path}/atc/python/site-packages:$PYTHONPATH

export LD\_LIBRARY\_PATH=${install\_path}/atc/lib64:${install\_path}/acllib/lib64:$LD\_LIBRARY\_PATH

export ASCEND\_OPP\_PATH=${install\_path}/opp

export ASCEND\_AICPU\_PATH=/usr/local/Ascend/ascend-toolkit/latest

# 模型转换

/usr/local/Ascend/ascend-toolkit/latest/atc/bin/atc --framework=5 --model="./model/vgg16.onnx" \

--output="model/vgg16" --input\_format=NCHW \

--input\_shape="image:1,3,224,224" \

--log=debug \

--soc\_version=Ascend310

- --framework：原始框架类型。5表示onnx。

- --model：vgg16网络的模型文件（\*.onnx）的路径。

- --output：vgg16.om模型文件名

- --input\_format：输入格式为（批次、通道、长度、宽度）

- --input\_shape：输入的shape，“image:"代表onnx输入节点的名称。

- --soc\_version：昇腾AI处理器的版本。昇腾310 AI处理器，此处配置为Ascend310。

- --precision\_mode：force\_fp16表示对该模型进行float16量化处理。

数据预处理模块

对数据图片进行预处理

代码：

# 导入相关依赖

import cv2

import sys

import numpy as np

# 缩放图片

def resize\_image(image, target\_size):

h, w = image.shape[:2]

th, tw = target\_size

# 获取等比缩放后的尺寸

scale = min(th / h, tw / w)

oh, ow = round(h \* scale), round(w \* scale)

# 缩放图片，opencv缩放传入尺寸为（宽，高），这里采用线性差值算法

image = cv2.resize(image, (ow, oh), interpolation=cv2.INTER\_LINEAR).astype(np.uint8)

# 将剩余部分进行填充

new\_image = np.ones((th, tw, 3), dtype=np.uint8) \* 114

new\_image[:oh, :ow, :] = image

return new\_image

# 预处理流程

def process\_image(img\_path):

# 读取图片，opencv读图后格式是BGR格式，需要转为RGB格式

image = cv2.imread(img\_path, cv2.IMREAD\_COLOR)

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

# 将图片等比resize至(224x224)

image = resize\_image(image, (224, 224))

image = np.array(image, dtype=np.float32)

# 将图片标准化

image -= [125.307, 122.961, 113.8575]

image /= [51.5865, 50.847, 51.255]

# (h,w,c) -> (c,h,w)

image = image.transpose((2, 0, 1))

return image

搭建推理模块

我们定义一个ACLEngine类用于实现推理流程，在这个类中，需要实现以下函数

* 资源初始化。
* 模型加载。
* 图片读取。
* 推理函数。
* 推理结果处理。
* 模型卸载。
* 释放内存。
* 资源去初始化。

代码：

# 导入依赖库

import struct

import time

import acl

from process import process\_image

import numpy as np

# 设置全局变量

CLASSES = ("daisy", "roses", "sunflowers", "tulips")

ACL\_MEM\_MALLOC\_HUGE\_FIRST = 0

ACL\_MEMCPY\_HOST\_TO\_DEVICE = 1

ACL\_MEMCPY\_DEVICE\_TO\_HOST = 2

NPY\_BYTE = 1

class ACLEngine(object):

def \_\_init\_\_(self, model\_path, device\_id=0):

self.device\_id = device\_id

self.context = None

self.model\_id = None

self.model\_desc = None

self.load\_input\_dataset, self.load\_output\_dataset = None, None

self.input\_data, self.output\_data = None, None

# --------------- pyACL初始化 ---------------

print('Start init resource')

self.\_init\_resource()

self.\_load\_model(model\_path)

self.\_prepare\_inputs()

def inference(self, img\_path):

""" 模型推理及后处理模块 """

# 1.读取并预处理图片

img = process\_image(img\_path)

# 2.准备模型推理的输入数据，运行模式默认为运行模式为ACL\_HOST，当前实例代码中模型只有一个输入。

bytes\_data = img.tobytes()

np\_ptr = acl.util.bytes\_to\_ptr(bytes\_data)

start\_time = time.time()

# 将图片数据从Host传输到Device。

ret = acl.rt.memcpy(self.input\_data[0]["buffer"], self.input\_data[0]["size"], np\_ptr,

self.input\_data[0]["size"], ACL\_MEMCPY\_HOST\_TO\_DEVICE)

# 3.执行模型推理。

# self.model\_id表示模型ID，在模型加载成功后，会返回标识模型的ID。

ret = acl.mdl.execute(self.model\_id, self.load\_input\_dataset, self.load\_output\_dataset)

# 4.处理模型推理的输出数据，输出置信度的类别编号。

inference\_result = []

for i, item in enumerate(self.output\_data):

buffer\_host, ret = acl.rt.malloc\_host(self.output\_data[i]["size"])

# 将推理输出数据从Device传输到Host。

ret = acl.rt.memcpy(buffer\_host, self.output\_data[i]["size"], self.output\_data[i]["buffer"],

self.output\_data[i]["size"], ACL\_MEMCPY\_DEVICE\_TO\_HOST)

bytes\_out = acl.util.ptr\_to\_bytes(buffer\_host, self.output\_data[i]["size"])

data = np.frombuffer(bytes\_out, dtype=np.byte)

inference\_result.append(data)

tuple\_st = struct.unpack("4f", bytearray(inference\_result[0]))

vals = np.array(tuple\_st).flatten()

top\_k = vals.argsort()[-1:-6:-1]

print("\n======== inference results: =============")

for i, j in enumerate(top\_k):

print("top %d: class:[%s]: probability:[%f]" % (i, CLASSES[j], vals[j]))

end\_time = time.time()

print('inference cost time: {:.1f}ms\n'.format((end\_time-start\_time)\*1000))

def release\_resource(self):

""" 资源释放模块 """

self.\_unload\_model()

self.\_unload\_picture()

self.\_destroy\_resource()

print('Resource destroyed successfully')

def \_init\_resource(self):

# pyACL初始化

ret = acl.init()

# 运行管理资源申请

# 指定运算的Device。

self.device\_id = 0

ret = acl.rt.set\_device(self.device\_id)

# 显式创建一个Context，用于管理Stream对象。

self.context, ret = acl.rt.create\_context(self.device\_id)

def \_load\_model(self, model\_path):

# 加载离线模型文件，返回标识模型的ID。

self.model\_id, ret = acl.mdl.load\_from\_file(model\_path)

# 根据加载成功的模型的ID，获取该模型的描述信息。

self.model\_desc = acl.mdl.create\_desc()

ret = acl.mdl.get\_desc(self.model\_desc, self.model\_id)

def \_prepare\_inputs(self):

# 1.准备模型推理的输入数据集。

# 创建aclmdlDataset类型的数据，描述模型推理的输入。

self.load\_input\_dataset = acl.mdl.create\_dataset()

# 获取模型输入的数量。

input\_size = acl.mdl.get\_num\_inputs(self.model\_desc)

self.input\_data = []

# 循环为每个输入申请内存，并将每个输入添加到aclmdlDataset类型的数据中。

for i in range(input\_size):

buffer\_size = acl.mdl.get\_input\_size\_by\_index(self.model\_desc, i)

# 申请输入内存。

buffer, ret = acl.rt.malloc(buffer\_size, ACL\_MEM\_MALLOC\_HUGE\_FIRST)

data = acl.create\_data\_buffer(buffer, buffer\_size)

\_, ret = acl.mdl.add\_dataset\_buffer(self.load\_input\_dataset, data)

self.input\_data.append({"buffer": buffer, "size": buffer\_size})

# 2.准备模型推理的输出数据集。

# 创建aclmdlDataset类型的数据，描述模型推理的输出。

self.load\_output\_dataset = acl.mdl.create\_dataset()

# 获取模型输出的数量。

output\_size = acl.mdl.get\_num\_outputs(self.model\_desc)

self.output\_data = []

# 循环为每个输出申请内存，并将每个输出添加到aclmdlDataset类型的数据中。

for i in range(output\_size):

buffer\_size = acl.mdl.get\_output\_size\_by\_index(self.model\_desc, i)

# 申请输出内存。

buffer, ret = acl.rt.malloc(buffer\_size, ACL\_MEM\_MALLOC\_HUGE\_FIRST)

data = acl.create\_data\_buffer(buffer, buffer\_size)

\_, ret = acl.mdl.add\_dataset\_buffer(self.load\_output\_dataset, data)

self.output\_data.append({"buffer": buffer, "size": buffer\_size})

def \_unload\_model(self):

# 卸载模型。

ret = acl.mdl.unload(self.model\_id)

# 释放模型描述信息。

if self.model\_desc:

ret = acl.mdl.destroy\_desc(self.model\_desc)

self.model\_desc = None

# 释放Context。

if self.context:

ret = acl.rt.destroy\_context(self.context)

self.context = None

def \_unload\_picture(self):

# 释放输出资源，包括数据结构和内存。

while self.output\_data:

item = self.output\_data.pop()

ret = acl.rt.free(item["buffer"])

output\_number = acl.mdl.get\_dataset\_num\_buffers(self.load\_output\_dataset)

for i in range(output\_number):

data\_buf = acl.mdl.get\_dataset\_buffer(self.load\_output\_dataset, i)

if data\_buf:

ret = acl.destroy\_data\_buffer(data\_buf)

ret = acl.mdl.destroy\_dataset(self.load\_output\_dataset)

def \_destroy\_resource(self):

# 释放Device。

ret = acl.rt.reset\_device(self.device\_id)

# pyACL去初始化。

ret = acl.finalize()

进行推理

完成模型转换、数据预处理、和推理模块的实现之后，可以加载模型进行推理

代码：

# 构建推理模块

engine = ACLEngine('./model/vgg16.om')

# 进行推理

engine.inference('./data/daisy\_demo.jpg')

engine.inference('./data/roses\_demo.jpg')

engine.inference('./data/sunflowers\_demo.jpg')

engine.inference('./data/tulips\_demo.jpg')

# 释放资源

engine.release\_resource()

输出：

Start init resource

======== inference results: =============

top 0: class:[daisy]: probability:[0.952637]

top 1: class:[roses]: probability:[0.026199]

top 2: class:[tulips]: probability:[0.012085]

top 3: class:[sunflowers]: probability:[0.008987]

inference cost time: 9.5ms

======== inference results: =============

top 0: class:[roses]: probability:[0.866211]

top 1: class:[tulips]: probability:[0.108032]

top 2: class:[daisy]: probability:[0.023865]

top 3: class:[sunflowers]: probability:[0.001881]

inference cost time: 9.2ms

======== inference results: =============

top 0: class:[sunflowers]: probability:[0.997559]

top 1: class:[tulips]: probability:[0.001055]

top 2: class:[daisy]: probability:[0.000991]

top 3: class:[roses]: probability:[0.000452]

inference cost time: 9.2ms

======== inference results: =============

top 0: class:[tulips]: probability:[0.992188]

top 1: class:[roses]: probability:[0.007118]

top 2: class:[sunflowers]: probability:[0.000441]

top 3: class:[daisy]: probability:[0.000272]

inference cost time: 9.2ms

Resource destroyed successfully

## 实验总结

掌握使用AscendCL（Ascend Computing Language）实现模型应用的开发。将以训练好的VGG16网络模型(onnx格式)转换为Davinci架构专用的模型，使VGG16网络推理过程可以高效的运行在Ascend硬件上。并对指定图片进行推理输出指定结果，搭建一个实时的图片分类应用。

## 实验任务与参考解答任务

### 实验任务

在Notebook上，基于上述操作说明，熟悉推理过程，完成python版本的推理，实现VGG16图片分类应用，并正确输出预测结果。

并根据CPP版本的文件代码，在main函数中正确按操作手册中介绍的推理顺序组合函数，完成CPP版本的推理，输出正确结果。