AI 计算系统
实验操作手册

实验序号：三

实验名称：基于 ACL 搭建 VGG16 实现图片分类应用

教　　师：朱光辉

学　　校：南京大学

时　　间：2022 年 8 月 23 日

三 基于ACL搭建VGG16实现图片分类应用

3.1 实验目的

本实验的目的是：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。

3.2 背景介绍

3.2.1 ONNX

Open Neural Network Exchange（ONNX，开放神经网络交换）格式，是一个用于表示深度学习模型的标准，可使模型在不同框架之间进行转移。

> **ONNX**（Open Neural Network Exchange）是一种针对机器学习所设计的开放式的文件格式，用于存储训练好的模型。它使得不同的人工智能框架（如Pytorch、MXNet）可以采用相同格式存储模型数据并交互。 ONNX的规范及代码主要由微软，亚马逊，Facebook和IBM等公司共同开发，以开放源代码的方式托管在Github上。[2][3][4] 目前官方支持加载ONNX模型并进行推理的深度学习框架有： Caffe2, PyTorch, MXNet，ML.NET，TensorRT 和 Microsoft CNTK，并且 TensorFlow 也非官方的支持ONNX。---维基百科

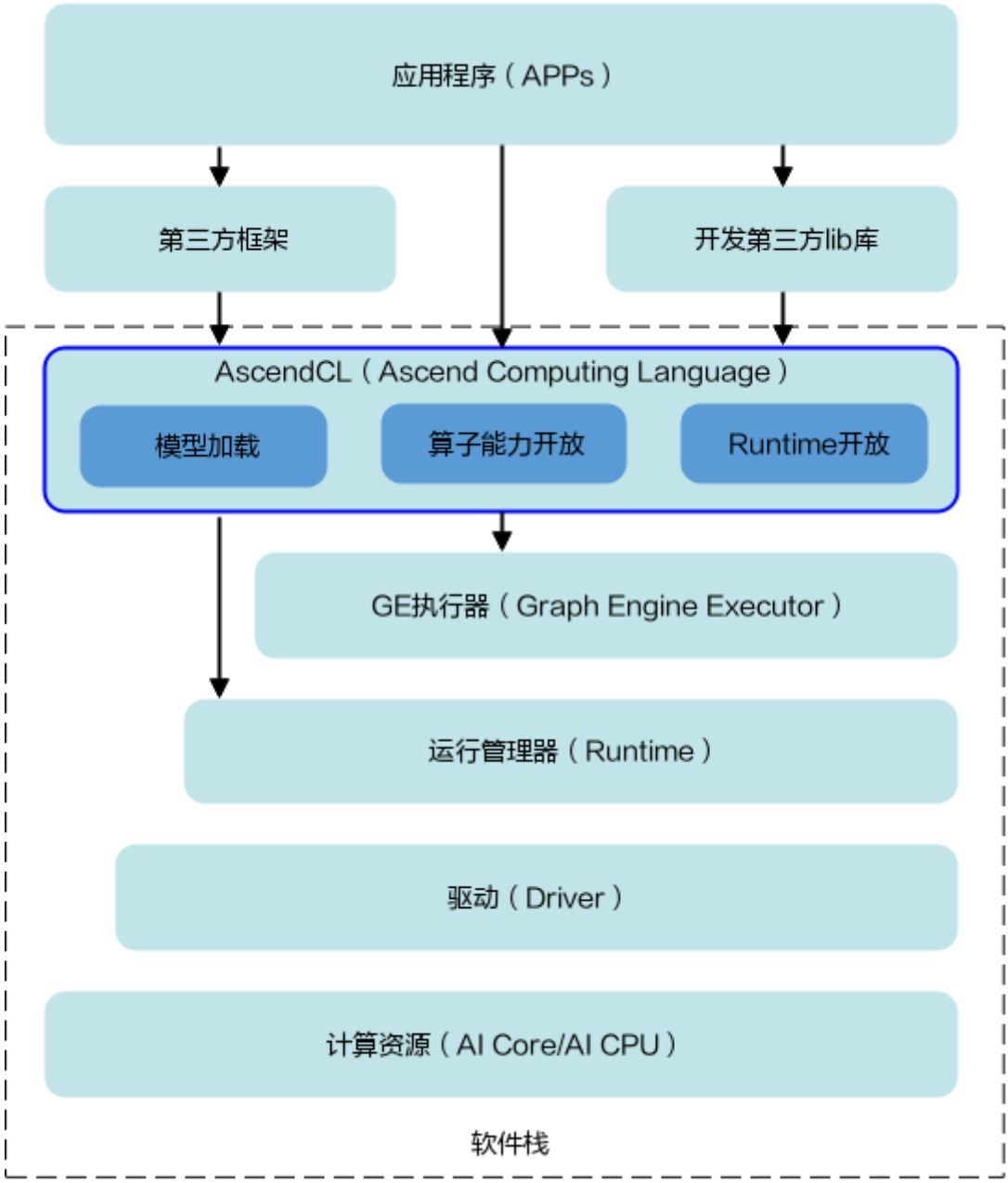
onnx模型可以上传至 https://netron.app/ 进行网络结构可视化。

3.2.2 AscendCL

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。



3.2.3 ATC

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。

3.3 实验环境

环境：ModelArts弹性云服务器，包含x86_64CPU和昇腾310推理芯片

操作系统：Ubuntu

依赖	版本
python	3.7.5
c++	7.5.0
cmake	3.10.2
numpy	1.19.4
scipy	1.5.4
opencv-python	4.5.3.56

3.4 实验内容

本实验基于ACL实现 `VGG16` 的推理过程，并将其打包成一个实时图片分类应用。并对提供的花卉测试数据图片进行推理，输出推理性能以及推理结果。首先利用ATC工具将 `VGG16` 模型 `*.onnx` 转换为适配 `Ascend310` 专用模型 `*.om`，然后将图片进行预处理后送入到 `VGG16` 模型进行推理，最后输出推理结果。

本实验的重点是利用AscendCL接口，充分发挥昇腾AI处理器的计算能力，并结合C++或Python API库开发一个实时的深度神经网络应用。

3.5 实验步骤

3.5.1 模型转换模块

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。（注：次实验为应用部署实验，只涉及推理，故VGG16后面包含Softmax模块，方便直接输出概率。）

1.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_LIBRA
RY_PATH
export ASCEND_OPP_PATH=${install_path}/opp
export ASCEND_AICPU_PATH=/usr/local/Ascend/ascend-toolkit/latest
```

2.模型转换

使用 atc 工具将 onnx 模型转换为 om 模型，文件工具使用方法可以参考CANN V100R020C10 开发辅助工具指南 (推理) 01

```
# 如果没有atc工具则生成atc软连接
ln -s /usr/local/Ascend/ascend-toolkit/latest/atc/bin/atc ./
```

```
# fp16量化
./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 \
      --precision_mode=force_fp16

# 正常导出
./atc --framework=5 --model="./model/vgg16.onnx" \
      --output="model/vgg16_fp32" --input_format=NCHW \
      --input_shape="image:1,3,224,224" \
      --log=debug \
      --soc_version=Ascend310 \
      --precision_mode=force_fp32
```

- --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。

3.5.2 数据预处理模块

数据预处理模块主要是将`jpg`格式的图片经过预处理之后，存储为二进制形式，从而送入卷积神经网络进行推理。预处理流程与实验二完全一致，故在此不多做说明。

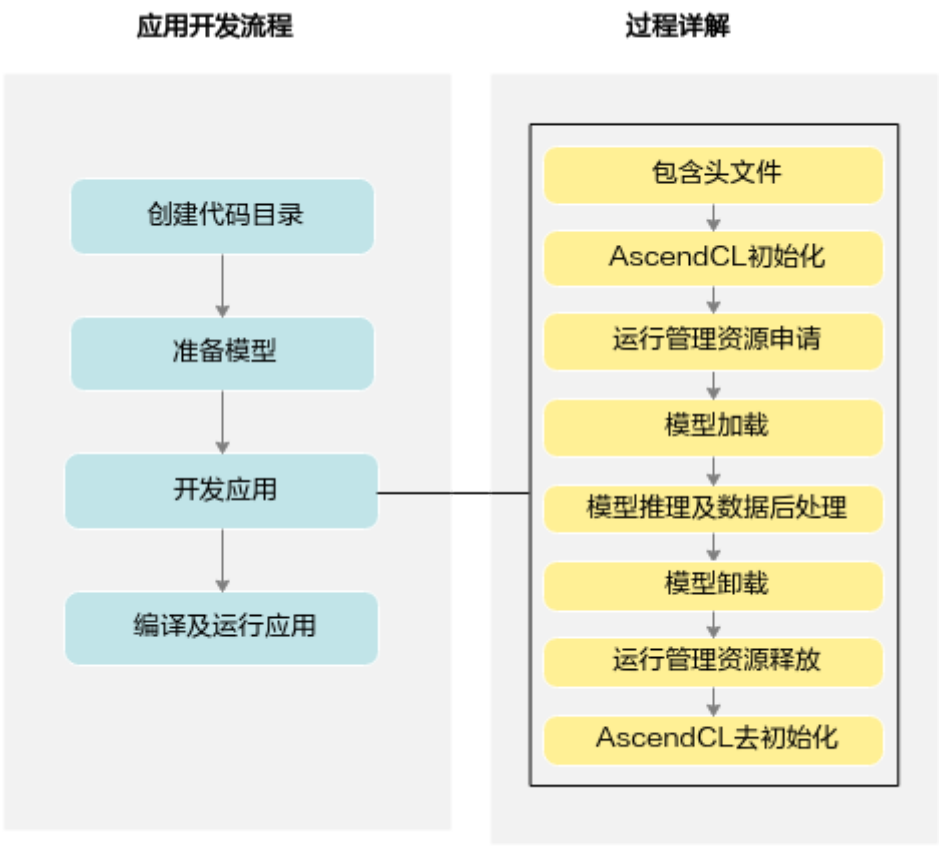
```
# file: process.py
import cv2
import sys
import numpy as np


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
```

```python
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


def process_2_bin():
    img_paths = ['./data/daisy_demo.jpg',
                 './data/roses_demo.jpg',
                 './data/sunflowers_demo.jpg',
                 './data/tulips_demo.jpg']
    for img_path in img_paths:
        # 数据预处理
        img = process_image(img_path)
        # 将处理好的图片存为bin格式，以便推理
        img.tofile(img_path.replace('jpg', 'bin'))


if __name__ == "__main__":
    process_2_bin()
```

3.5.3 推理模块（C++）

整体推理流程实现如下所示，其中定义了以下成员函数：

- 资源初始化：用于AscendCL初始化、运行管理资源申请（指定计算设备）
- 模型加载：加载图片分类的模型，用于后续推理使用
- 图片读取：将测试图片数据读入内存，并传输到Device侧，用于后续推理使用
- 推理函数：使用Ascend310执行推理
- 推理结果处理：用于在终端上屏显测试图片的top5置信度的类别编号
- 模型卸载：卸载图片分类的模型
- 释放内存：用于释放内存、销毁推理相关的数据类型，防止内存泄露
- 资源去初始化：用于AscendCL去初始化、运行管理资源释放（释放计算设备）



3.5.3.1 资源初始化

```
// ---------------- 资源初始化 ----------------
int32_t deviceId = 0;
void InitResource()
{
    aclError ret = aclInit(nullptr);
    ret = aclrtSetDevice(deviceId);
}
```

API接口说明：

- aclInit：AscendCL初始化函数
- aclrtSetDevice：指定当前进程的Device，同时创建Context。

3.5.3.2 模型加载

```
// ---------------- 模型加载 ----------------
uint32_t modelId;
void LoadModel(const char* modelPath)
{
    aclError ret = aclmdlLoadFromFile(modelPath, &modelId);
}
```

API接口说明：

- aclmdlLoadFromFile：加载离线模型

3.5.3.3 图片读取

```
// ---------------- 图片读取 ----------------
size_t pictureDataSize = 0;
void *pictureHostData;
void *pictureDeviceData;

//申请内存，使用C/C++标准库的函数将测试图片读入内存
void ReadPictureTotHost(const char *picturePath)
{
    string fileName = picturePath;
    ifstream binFile(fileName, ifstream::binary);
    binFile.seekg(0, binFile.end);
    pictureDataSize = binFile.tellg();
    binFile.seekg(0, binFile.beg);
    aclError ret = aclrtMallocHost(&pictureHostData, pictureDataSize);
    binFile.read((char*)pictureHostData, pictureDataSize);
    binFile.close();
}

//申请Device侧的内存，再以内存复制的方式将内存中的图片数据传输到Device
void CopyDataFromHostToDevice()
{
    aclError ret = aclrtMalloc(&pictureDeviceData, pictureDataSize,
ACL_MEM_MALLOC_HUGE_FIRST);
    ret = aclrtMemcpy(pictureDeviceData, pictureDataSize, pictureHostData,
pictureDataSize, ACL_MEMCPY_HOST_TO_DEVICE);
}

void LoadPicture(const char* picturePath)
{
    ReadPictureTotHost(picturePath);
    CopyDataFromHostToDevice();
}
```

API接口说明：

- aclrtMallocHost：应用在Device上运行时，调用该接口申请的是Device内存。
- aclrtMalloc：在Device上申请size大小的线性内存，通过*devPtr返回已分配内存的指针。
- aclrtMemcpy：实现Host内、Host与Device之间、Device内、Device间的同步内存复制。

3.5.3.4 推理函数

```
// ---------------- 推理函数 ----------------
aclmdlDataset *inputDataSet;
aclDataBuffer *inputDataBuffer;
aclmdlDataset *outputDataSet;
aclDataBuffer *outputDataBuffer;
aclmdlDesc *modelDesc;
size_t outputDataSize = 0;
void *outputDeviceData;

// 准备模型推理的输入数据结构
void CreateModelInput()
{
```

```
    // 创建aclmdlDataset类型的数据，描述模型推理的输入
    inputDataSet = aclmdlCreateDataset();
    inputDataBuffer = aclCreateDataBuffer(pictureDeviceData, pictureDataSize);
    aclError ret = aclmdlAddDatasetBuffer(inputDataSet, inputDataBuffer);
}

// 准备模型推理的输出数据结构
void CreateModelOutput()
{
   // 创建模型描述信息
    modelDesc =  aclmdlCreateDesc();
    aclError ret = aclmdlGetDesc(modelDesc, modelId);
    // 创建aclmdlDataset类型的数据，描述模型推理的输出
    outputDataSet = aclmdlCreateDataset();
    // 获取模型输出数据需占用的内存大小，单位为Byte
    outputDataSize = aclmdlGetOutputSizeByIndex(modelDesc, 0);
    // 申请输出内存
    ret = aclrtMalloc(&outputDeviceData, outputDataSize,
ACL_MEM_MALLOC_HUGE_FIRST);
    outputDataBuffer = aclCreateDataBuffer(outputDeviceData, outputDataSize);
    ret = aclmdlAddDatasetBuffer(outputDataSet, outputDataBuffer);
}

// 执行模型
void Inference()
{
    CreateModelInput();
    CreateModelOutput();
    aclError ret = aclmdlExecute(modelId, inputDataSet, outputDataSet);
}
```

API接口说明：

- aclmdlCreateDataset：创建aclmdlDataset类型的数据，该数据类型用于描述模型推理时的输入数据、输出数据。
- aclCreateDataBuffer：创建aclDataBuffer类型的数据，该数据类型用于描述内存地址、大小等内存信息。
- aclmdlAddDatasetBuffer：向aclmdlDataset中增加aclDataBuffer。
- aclmdlCreateDesc：创建aclmdlDesc类型的数据，表示模型描述信息。
- aclmdlGetDesc：根据模型ID获取该模型的模型描述信息。
- aclmdlGetOutputSizeByIndex：根据模型描述信息获取指定输出的大小，单位为Byte。
- aclmdlExecute：执行模型推理，返回推理结果。

3.5.3.5 推理结果处理

```
// ---------------- 推理结果处理 ----------------
void *outputHostData;
void PrintResult()
{
    // 获取推理结果数据
    aclError ret = aclrtMallocHost(&outputHostData, outputDataSize);
    ret = aclrtMemcpy(outputHostData, outputDataSize, outputDeviceData,
outputDataSize, ACL_MEMCPY_DEVICE_TO_HOST);
    // 将内存中的数据转换为float类型
    float* outFloatData = reinterpret_cast<float *>(outputHostData);
```

```
    // 屏显测试图片的置信度的类别编号
    map<float, unsigned int, greater<float>> resultMap;
    for (unsigned int j = 0; j < outputDataSize / sizeof(float);++j)
    {
        resultMap[*outFloatData] = j;
        outFloatData++;
    }

    int cnt = 0;
    // 类别集合
    const char *ClassMap[4] = {"daisy", "roses", "sunflowers", "tulips"};
    for (auto it = resultMap.begin();it != resultMap.end();++it,++cnt)
    {
        printf("top %d: class[%s] probability[%lf] \n", cnt, ClassMap[it-
>second], it->first);
    }
}
```

3.5.3.6 模型卸载

```
// ---------------- 模型卸载 ----------------
void UnloadModel()
{
    // 释放模型描述信息
    aclmdlDestroyDesc(modelDesc);
    // 卸载模型
    aclmdlUnload(modelId);
}
```

API接口说明：

- aclmdlDestroyDesc：销毁通过aclmdlCreateDesc接口创建的aclmdlDesc类型的数据。
- aclmdlUnload：系统完成模型推理后，可调用该接口卸载模型，释放资源，同步接口。

3.5.3.7 释放内存

```
// ---------------- 释放内存 ----------------
void UnloadPicture()
{
    aclError ret = aclrtFreeHost(pictureHostData);
    pictureHostData = nullptr;
    ret = aclrtFree(pictureDeviceData);
    pictureDeviceData = nullptr;
    aclDestroyDataBuffer(inputDataBuffer);
    inputDataBuffer = nullptr;
    aclmdlDestroyDataset(inputDataSet);
    inputDataSet = nullptr;

    ret = aclrtFreeHost(outputHostData);
    outputHostData = nullptr;
    ret = aclrtFree(outputDeviceData);
    outputDeviceData = nullptr;
    aclDestroyDataBuffer(outputDataBuffer);
```

```
    outputDataBuffer = nullptr;
    aclmdlDestroyDataset(outputDataSet);
    outputDataSet = nullptr;
}
```

API接口说明：

- aclrtFreeHost：释放内存。
- aclrtFree：释放Device上的内存。
- aclDestroyDataBuffer：销毁通过aclCreateDataBuffer接口创建的aclDataBuffer类型的数据。
- aclmdlDestroyDataset：销毁通过aclmdlCreateDataset接口创建的aclmdlDataset类型的数据。

3.5.3.8 资源去初始化

```
// ---------------- 资源去初始化 ----------------
void DestroyResource()
{
    aclError ret = aclrtResetDevice(deviceId);
    aclFinalize();
}
```

API接口说明：

- aclrtResetDevice：复位当前运算的Device，释放Device上的资源，包括默认Context、默认Stream以及默认Context下创建的所有Stream。

3.5.4 推理模块（Python）

具体模块如上一致，这里只介绍相应的代码及API接口部分。

- acl.init：初始化pyACL配置
- acl.finalize：实现pyACL去初始化。
- acl.rt.set_device：指定用于运算的Device。
- acl.rt.get_run_mode：获取昇腾AI软件栈的运行模式
- acl.rt.reset_device：复位当前运算的Device
- acl.rt.create_context：创建Context
- acl.rt.destroy_context：销毁Context
- acl.rt.create_stream：创建Stream
- acl.rt.destroy_stream：销毁Stream
- acl.rt.malloc：申请Device上的内存
- acl.rt.free：释放Device上的内存。
- acl.rt.memcpy：：将数据从Host传输到Device上
- acl.mdl.load_from_file：从*.om文件加载模型
- acl.mdl.execute：执行模型推理，同步接口。
- acl.mdl.unload：卸载模型。
- acl.mdl.get_dataset_num_buffers：从输出aclmdlDataset获取aclDataBuffer的个数num。
- acl.mdl.get_dataset_buffer：获取aclDataBuffer
- acl.get_data_buffer_addr：获取aclDataBuffer中数据
- acl.get_data_buffer_size_v2：获取aclDataBuffer中数据大小。
- acl.rt.malloc_host、acl.rt.memcpy：做数据搬移，从Device->Host。
- acl.util.ptr_to_numpy：将指针转换numpy对象

3.5.4.1 资源初始化

```
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)
```

3.5.4.2 模型加载

```
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)
```

3.5.4.3 准备图片输入

```
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})
```

3.5.4.4 模型推理和处理

```
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))
```

3.5.4.5 模型卸载

```
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
```

3.5.4.6 释放内存

```
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)
```

3.5.4.7 资源去初始化

```
def _destroy_resource(self):
    # 释放Device。
    ret = acl.rt.reset_device(self.device_id)
    # pyACL去初始化。
    ret = acl.finalize()
```

3.5.5 实验运行

代码目录介绍

```
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
|   ├── daisy_demo.bin          // 测试雏菊图片处理后二进制文件（待生成）
|   ├── roses_demo.bin          // 测试玫瑰图片处理后二进制文件（待生成）
|   ├── sunflowers_demo.bin     // 测试向日葵图片处理后二进制文件（待生成）
|   ├── tulips_demo.bin         // 测试郁金香图片处理后二进制文件（待生成）

├── 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）
|   ├── vgg16_fp16.om           // VGG16网络的模型文件(*.om)（待生成）
```

```
├── src
|   ├── CMakeLists.txt          // cmake编译脚本
|   ├── main.cpp                // 主函数，图片分类功能的实现文件

├── 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，包括将*.jpg转换为*.bin。
├── sample.sh                   // 运行应用的脚本
```

3.5.5.1 基于C++运行

1.模型转换

```
bash convert_model.sh
```

2.实现代码

```
vim src/main.cpp
```

3.运行实验

在`main.cpp`中选择float16量化前后的模型分别进行测试。

```
bash sample.sh
```

3.5.5.2 基于Python运行

1.模型转换

```
bash convert_model.sh
```

2.激活环境

```
source /usr/local/Ascend/ascend-toolkit/set_env.sh
```

3.实现代码

```
vim app.py
```

4.运行实验

在`app.py`中选择float16量化前后的模型分别进行测试。

```
python3.7 app.py
```

3.6 评分指标

80：基于Python编程语言实现VGG16图片分类应用，并正确输出预测结果。

100：基于C++编程语言实现VGG16图片分类应用，并正确输出预测结果。

3.7 实验思考

上述应用程序还有哪些性能瓶颈？如何进一步提升推理速度？

对于多张图片推理，可以采取哪些方案进行改进？