Atlas 200I DK A2 开发者套件
23.0.RC3

MindX SDK 应用开发入门

文档版本	01
发布日期	2023-11-14

华为技术有限公司

商标声明


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： https://www.huawei.com

客户服务邮箱： support@huawei.com

客户服务电话： 4008302118

安全声明

漏洞声明

华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该政策可参考华为公司官方网站的网址：https://www.huawei.com/cn/psirt/vul-response-process。
如企业客户须获取漏洞信息，请访问：https://securitybulletin.huawei.com/enterprise/cn/security-advisory。

目 录

1 图像分类应用样例开发介绍（Python）

样例介绍

本文以MindX SDK来开发一个简单的图像分类应用，图像分类模型推理流程如**图1 分类模型推理流程图**所示。

图 1-1 分类模型推理流程图

本例中使用的是Caffe框架的ResNet-50模型。可以直接使用训练好的开源模型，也可以基于开源模型的源码进行修改、重新训练，还可以基于算法、框架构建适合的模型。

模型的输入数据与输出数据格式：

- 输入数据：RGB格式、224*224分辨率的输入图片。
- 输出数据：图片的类别标签及其对应置信度。

获取代码

步骤1 获取代码文件。

单击**获取链接**或使用wget命令，下载代码文件压缩包，以**root**用户登录开发者套件。

```
wget https://ascend-repo.obs.cn-east-2.myhuaweicloud.com/Atlas%20200I%20DK%20A2/DevKit/models/sdk_cal_samples/resnet50_sdk_python_sample.zip
```

步骤2 将“resnet50_sdk_python_sample.zip”压缩包上传到开发者套件，解压并进入解压后的目录。

```
unzip resnet50_sdk_python_sample.zip
cd resnet50_sdk_python_sample
```

代码目录结构如下所示，按照正常开发流程，需要将框架模型文件转换成昇腾AI处理器支持推理的om格式模型文件，鉴于当前是入门内容，用户可直接获取已转换好的om模型进行推理。

```
resnet50_sdk_python_sample
├── data
│   ├── test.jpg            # 测试图片
├── model
```

```
│   ├── resnet50.om           # ResNet-50网络的om模型
├── utils
│   ├── resnet50.cfg          # 用于后处理的配置文件，包含类别数量和是否包含softmax操作，具体可打开文件查看
│   ├── resnet50_clsidx_to_labels.names          # 类别标签文件
├── main.py                   # 运行程序的脚本
```

步骤3 准备用于推理的图片数据。

如**步骤2**所示文件结构，内置测试图片为“test.jpg”，用户也可从imagenet数据集中获取其它图片。

----结束

代码解析

开发代码过程中，在“resnet50_sdk_python_sample/main.py”文件中已包含读入数据、前处理、推理、后处理等功能，串联整个应用代码逻辑，此处仅对代码进行解析。

1. 在“main.py”文件的开头有如下代码，用于导入需要的第三方库以及MindX SDK推理所需文件。
```
import numpy as np  # 用于对多维数组进行计算
import cv2  # 图片处理三方库，用于对图片进行前后处理

from mindx.sdk import Tensor  # mxVision 中的 Tensor 数据结构
from mindx.sdk import base  # mxVision 推理接口
from mindx.sdk.base import post  # post.Resnet50PostProcess 为 resnet50 后处理接口
```
2. 初始化资源和模型相关变量，如图片路径、模型路径、配置文件路径、标签路径等。
```
'''初始化资源和变量'''
base.mx_init()  # 初始化 mxVision 资源
pic_path = 'data/test.jpg'  # 单张图片
model_path = "model/resnet50.om"  # 模型路径
device_id = 0  # 指定运算的Device
config_path='utils/resnet50.cfg'  # 后处理配置文件
label_path='utils/resnet50_clsidx_to_labels.names'  # 类别标签文件
img_size = 256
```
3. 对输入数据进行前处理。先使用opencv读入图片，得到三维数组，再进行相应的图片裁剪、缩放、转换颜色空间等处理，并将其转化为MindX SDK推理所需要的数据格式（Tensor类型）。
```
'''前处理'''
img_bgr = cv2.imread(pic_path)
img_rgb = img_bgr[:,:,::-1]
img = cv2.resize(img_rgb, (img_size, img_size))  # 缩放到目标大小
hw_off = (img_size - 224) // 2  # 对图片进行切分，取中间区域
crop_img = img[hw_off:img_size - hw_off, hw_off:img_size - hw_off, :]
img = crop_img.astype("float32")  # 转为 float32 数据类型
img[:, :, 0] -= 104  # 常数 104,117,123 用于将图像转换到Caffe模型需要的颜色空间
img[:, :, 1] -= 117
img[:, :, 2] -= 123
img = np.expand_dims(img, axis=0)  # 扩展第一维度，适应模型输入
img = img.transpose([0, 3, 1, 2])  # 将 (batch,height,width,channels) 转为 (batch,channels,height,width)
img = np.ascontiguousarray(img)  # 将内存连续排列
img = Tensor(img) # 将numpy转为转为Tensor类
```
4. 使用MindX SDK接口进行模型推理，得到模型输出结果。
```
'''模型推理'''
model = base.model(modelPath=model_path, deviceId=device_id)  # 初始化 base.model 类
output = model.infer([img])[0]  # 执行推理。输入数据类型：List[base.Tensor]， 返回模型推理输出的 List[base.Tensor]
```
5. 对模型输出进行后处理。利用MindX SDK所带的后处理插件，可直接得到预测类别及其置信度，并将其画在原图上。


```
'''后处理'''
postprocessor = post.Resnet50PostProcess(config_path=config_path, label_path=label_path)  # 获取后处理对象
pred = postprocessor.process([output])[0][0]  # 利用SDK接口进行后处理，pred：<ClassInfo classId=... confidence=... className=...>
confidence = pred.confidence  # 获取类别置信度
className = pred.className  # 获取类别名称
print('{}: {}'.format(className, confidence))  # 打印出结果

'''保存推理图片'''
img_res = cv2.putText(img_bgr, f'{className}: {confidence:.2f}', (20, 20), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 1)  # 将预测的类别与置信度添加到图片
cv2.imwrite('result.png', img_res)
print('save infer result success')
```

运行推理

步骤1 配置环境变量。

- Ubuntu OS：
  ```
  . /usr/local/Ascend/mxVision/set_env.sh
  ```
- openEuler OS：
  ```
  . $HOME/Ascend/mxVision/set_env.sh
  ```

步骤2 运行主程序。

```
python main.py
```

命令行输出如下，表明运行成功。

```
Standard Poodle: 0.98583984375
save infer result success
```

推理完成后，在当前文件夹下生成“result.png”文件，如图1-2所示：

图 1-2 result.png 文件

----结束

样例总结与扩展

以上代码包括以下几个步骤：

1. 前处理：对图片进行 缩放、裁剪、转换颜色空间、转换维度、连续排列内存、转为base.Tensor操作。

2. 推理：利用Model或者base.model初始化模型，并用infer进行推理。

3. 后处理：利用MindX SDK的后处理接口直接得到预测结果，提取结果后，将其打印到终端，顺便保存到图片。

MindX SDK接口分类总结：

分类	接口函数	描述
推理相关	base.model(model_path, device_id)	初始化模型
	model.infer([img])	通过输入Tensor列表进行模型推理
后处理相关	post.Resnet50PostProcess(config_path, label_path)	ResNet-50后处理

理解各个接口含义后，用户可进行灵活运用。除此外，此样例中只示范了图片推理，若需要对视频流数据进行推理，可用两种方式输入视频流数据：USB摄像头、手机摄像头。具体使用方式可参考《**摄像头拉流**》，用户只需将前处理、推理及后处理代码放入摄像头推理代码的循环中即可，注意修改的逻辑如下：

- 引入三方库部分，加入import cv2。在初始化变量后，加入了USB摄像头读写相关代码，并在将图片前处理、推理、后处理相关代码放入了try...except...结构中。除此外，还相应的修改了数据前处理和后处理代码。
- 对于前处理，原来代码是利用opencv从路径中读入图片数据，而此处是直接从摄像头中获取图像帧，两者获取的数据都为bgr格式，所以前处理步骤相同。
- 对于后处理，代码也保持不变。

下面以USB摄像头为例，运行代码后，可在主目录保存结果视频“video_result.mp4”。其他摄像头使用方式可按相应逻辑修改：

```
import numpy as np  # 用于对多维数组进行计算
import cv2

from mindx.sdk import Tensor  # mxVision 中的 Tensor 数据结构
from mindx.sdk import base  # mxVision 推理接口
from mindx.sdk.base import post  # post.Resnet50PostProcess 为 resnet50 后处理接口


'''初始化变量'''
base.mx_init()  # 初始化 mxVision 资源
pic_path = 'data/test.jpg'  # 单张图片
model_path = "model/resnet50.om"  # 模型路径
device_id = 0  # 指定运算的Device
config_path='utils/resnet50.cfg'  # 后处理配置文件
label_path='utils/resnet50_clsidx_to_labels.names'  # 类别标签文件


# 打开摄像头
cap = cv2.VideoCapture(0)  # 打开摄像头

# 获取保存视频相关变量
fps = cap.get(cv2.CAP_PROP_FPS)
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
outfile = 'video_result.mp4'
video_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
video_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
writer = cv2.VideoWriter(outfile, fourcc, fps, (video_width, video_height))
```

```
try:
    while(cap.isOpened()):  # 在摄像头打开的情况下循环执行
        ret, frame = cap.read()  # 此处 frame 为 bgr 格式图片

        '''前处理'''
        img_size = 256
        img_rgb = frame[:,:,::-1]  # bgr to rgb
        img = cv2.resize(img_rgb, (img_size, img_size))  # 缩放到目标大小
        hw_off = (img_size - 224) // 2  # 对图片进行切分，取中间区域
        crop_img = img[hw_off:img_size - hw_off, hw_off:img_size - hw_off, :]
        img = crop_img.astype("float32")  # 转为 float32 数据类型
        img[:, :, 0] -= 104  # 常数 104,117,123 用于将图像转换到Caffe模型需要的颜色空间
        img[:, :, 1] -= 117
        img[:, :, 2] -= 123
        img = np.expand_dims(img, axis=0)  # 扩展第一维度，适应模型输入
        img = img.transpose([0, 3, 1, 2])  # 将 (batch,height,width,channels) 转为
(batch,channels,height,width)
        img = np.ascontiguousarray(img)  # 将内存连续排列
        img = Tensor(img) # 将numpy转为转为Tensor类

        '''模型推理'''
        model = base.model(modelPath=model_path, deviceId=device_id)  # 初始化 base.model 类
        output = model.infer([img])[0]  # 执行推理。输入数据类型：List[base.Tensor]， 返回模型推理输出的
List[base.Tensor]

        '''后处理'''
        postprocessor = post.Resnet50PostProcess(config_path=config_path, label_path=label_path)  # 获取后
处理对象
        pred = postprocessor.process([output])[0][0]  # 利用sdk接口进行后处理，pred：<ClassInfo classId=267
confidence=0.935546875 className=Standard Poodle>
        confidence = pred.confidence  # 获取类别置信度
        className = pred.className  # 获取类别名称

        '''保存推理帧到结果视频'''
        print('{}: {}'.format(className, confidence))  # 打印出结果
        img_res = cv2.putText(frame, f'{className}: {confidence:.2f}', (20, 20), cv2.FONT_HERSHEY_SIMPLEX,
0.5, (255, 255, 255), 1)
        writer.write(img_res)  # 将推理结果写入视频

except KeyboardInterrupt:
    cap.release()
    writer.release()
    print('save infer result success')
finally:
    cap.release()
    writer.release()
    print('save infer result success')
```


2 目标检测应用样例开发介绍（Python）

样例介绍

本文以MindX SDK来开发一个简单的目标检测应用，目标检测模型推理流程如**图1 目标检测模型推理流程图**所示。

图 2-1 目标检测模型推理流程图

本例中使用的是pytorch框架的yolov5模型。可以直接使用训练好的开源模型，也可以基于开源模型的源码进行修改、重新训练，还可以基于算法、框架构建适合的模型。

模型的输入数据与输出数据格式：

- 输入数据：RGB格式图片，分辨率为 640*640，输入形状为（1,3,640,640），即（batchsize，channel，height，width），对应每个batch的图片数量、图片的RGB维度、图片高度、图片宽度。
- 输出数据：目标检测框的坐标值、置信度、类别。

前期准备

步骤1 获取代码文件。

单击**获取链接**或使用wget命令，下载代码文件压缩包，以**root**用户登录开发者套件。

```
wget https://ascend-repo.obs.cn-east-2.myhuaweicloud.com/Atlas%20200I%20DK%20A2/DevKit/models/sdk_cal_samples/yolo_sdk_python_sample.zip
```

步骤2 将“yolo_sdk_python_sample.zip”压缩包上传到开发者套件，解压并进入目录。

```
unzip yolo_sdk_python_sample.zip
cd yolo_sdk_python_sample
```

代码目录结构如下所示，按照正常开发流程，需要将框架模型文件转换成昇腾AI处理器支持推理的om格式模型文件，鉴于当前是入门内容，用户可直接获取已转换好的om模型进行推理。

```
yolo_sdk_python_sample
├── main.py              # 运行程序的脚本
├── coco_names.txt          # coco数据集所有类别名
├── det_utils.py         # 模型相关前后处理函数
```

```
├── world_cup.jpg          # 测试图片
├── model
│   ├── yolov5s_bs1.om      # 已有om模型
```

----结束

代码解析

开发代码过程中，在“yolo_sdk_python_sample/main.py”文件中已包含读入数据、前处理、推理、后处理等功能，串联整个应用代码逻辑，此处仅对代码进行解析。

1. 在“main.py”文件的开头有如下代码，用于导入需要的第三方库以及MindX SDK推理所需文件。

```
import cv2  # 图片处理三方库，用于对图片进行前后处理
import numpy as np  # 用于对多维数组进行计算
import torch  # 深度学习运算框架，此处主要用来处理数据

from mindx.sdk import Tensor  # mxVision 中的 Tensor 数据结构
from mindx.sdk import base  # mxVision 推理接口

from det_utils import get_labels_from_txt, letterbox, scale_coords, nms, draw_bbox  # 模型前后处理相关函数
```

2. 初始化资源、定义模型相关变量，如图片路径、模型路径、设备id等。

```
# 初始化资源和变量
base.mx_init()  # 初始化 mxVision 资源
DEVICE_ID = 0  # 设备id
model_path = 'model/yolov5s_bs1.om'  # 模型路径
image_path = 'world_cup.jpg'  # 测试图片路径
```

3. 对输入数据进行前处理。先使用opencv读入图片，再进行相应的图片大小缩放填充、通道转换等处理，并将其转化为mindx sdk推理所需要的数据集格式（Tensor类）。

```
# 数据前处理
img_bgr = cv2.imread(image_path, cv2.IMREAD_COLOR)  # 读入图片
img, scale_ratio, pad_size = letterbox(img_bgr, new_shape=[640, 640])  # 对图像进行缩放与填充，保持长宽比
img = img[:, :, ::-1].transpose(2, 0, 1)  # BGR to RGB, HWC to CHW，将形状转换为 channel first
img = np.expand_dims(img, 0).astype(np.float32)  # 得到(1, 3, 640, 640)，即扩展第一维为 batchsize
img = np.ascontiguousarray(img) / 255.0  # 转换为内存连续存储的数组
img = Tensor(img) # 将numpy转为转为Tensor类
```

4. 使用MindX SDK接口进行模型推理，得到模型输出结果。

```
# 模型推理, 得到模型输出
model = base.model(modelPath=model_path, deviceId=DEVICE_ID)  # 初始化 base.model 类
output = model.infer([img])[0]  # 执行推理。输入数据类型：List[base.Tensor]， 返回模型推理输出的List[base.Tensor]
```

5. 对模型输出进行后处理。将 base.tensor类并转换为利于处理的numpy数组，再进行非极大值抑制、缩放图片、画出检测框等步骤(所涉及到的 nms、scale_coords及draw_bbox函数都可参见 det_utils.py)，得到最终可以用于显示的的目标检测结果，最后保存图片文件。

```
# 后处理
output.to_host()  # 将Tensor数据转移到内存
output = np.array(output)  # 将数据转为 numpy array 类型
boxout = nms(torch.tensor(output), conf_thres=0.4, iou_thres=0.5)  # 利用非极大值抑制处理模型输出，conf_thres 为置信度阈值，iou_thres 为iou阈值
pred_all = boxout[0].numpy()  # 转换为numpy数组
scale_coords([640, 640], pred_all[:, :4], img_bgr.shape, ratio_pad=(scale_ratio, pad_size))  # 将推理结果缩放到原始图片大小
labels_dict = get_labels_from_txt('./coco_names.txt')  # 得到类别信息，返回序号与类别对应的字典
img_dw = draw_bbox(pred_all, img_bgr, (0, 255, 0), 2, labels_dict)  # 画出检测框、类别、概率

# 保存图片到文件
cv2.imwrite('result.png', img_dw)
print('save infer result success')
```


运行推理

步骤1 配置环境变量。

- Ubuntu OS：
  ```
  . /usr/local/Ascend/mxVision/set_env.sh
  ```
- openEuler OS：
  ```
  . $HOME/Ascend/mxVision/set_env.sh
  ```

步骤2 运行主程序。

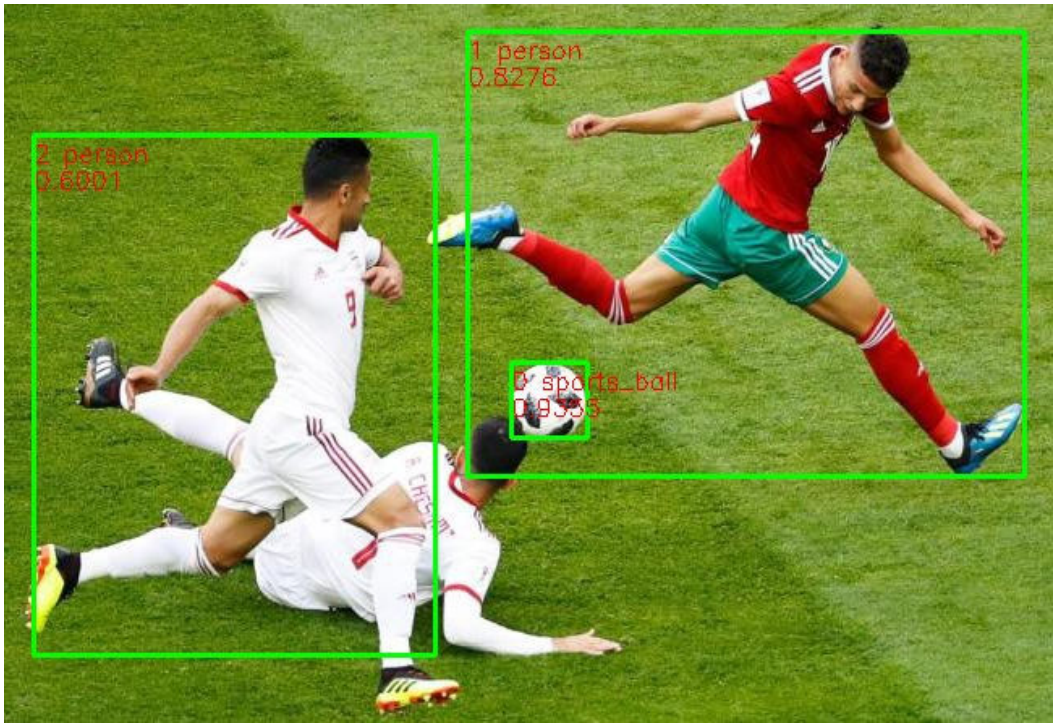
```
python main.py
```

命令行输出如下，表明运行成功：

```
save infer result success
```

推理完成后，在当前文件夹下生成“result.png”文件，如图2-2所示：

图 2-2 “result.png”文件



----结束

样例总结与扩展

以上代码包括以下几个步骤：

1. 前处理：对图片进行 缩放填充，维度转换、连续内存排列、以及转化为base.Tensor操作。

2. 推理：利用Model或者base.model 初始化模型，并用infer进行推理。

3. 后处理：用 非极大值抑制函数 处理模型输出结果，并将检测框坐标对应到原图上，再将结果画到原图上并保存。

MindX SDK接口分类总结：

分类	接口函数	描述
推理相关	base.model(model_path, device_id)	初始化模型
	model.infer([img])	通过输入Tensor列表进行模型推理

理解各个接口含义后，用户可进行灵活运用。除此外，此样例中只示范了图片推理，若需要对视频流数据进行推理，可用两种方式输入视频流数据：USB摄像头、手机摄像头。具体使用方式可参考《**摄像头拉流**》，用户只需将前处理、推理及后处理代码放入摄像头推理代码的循环中即可，注意有些细节地方需进行相应修改。

其中注释与图片读入和保存相关代码，并将前处理、推理及后处理代码加到了参考《**摄像头拉流**》所示的循环中。cv2.VideoCapture(url)这种读取方式会出现延迟高，掉帧多等现象，所以手动将帧率fps设置为5，用户也可根据实际情况设置fps。除此外，也可利用多线程来缓解此问题，由于本案例是入门案例，不再向外拓展。

下面以手机摄像头的rtsp拉流为例，改动后的代码如下：

```
# coding=utf-8
import cv2  # 图片处理三方库，用于对图片进行前后处理
import numpy as np  # 用于对多维数组进行计算
import torch  # 深度学习运算框架，此处主要用来处理数据
from mindx.sdk import Tensor  # mxVision 中的 Tensor 数据结构
from mindx.sdk import base  # mxVision 推理接口
from det_utils import get_labels_from_txt, letterbox, scale_coords, nms, draw_bbox  # 模型前后处理相关函数

# 变量初始化
base.mx_init()  # 初始化 mxVision 资源
DEVICE_ID = 0  # 设备id
model_path = 'model/yolov5s_bs1.om'  # 模型路径
# image_path = 'world_cup.jpg'  # 测试图片路径

# 利用手机ip摄像头
url = 'rtsp://admin:password@192.168.0.102:8554/live'  # 这里需要替换为自己的链接
cap = cv2.VideoCapture(url)

# 获取保存视频相关变量
fps = 5  # 使用rtsp推流时，不能使用cap.get(cv2.CAP_PROP_FPS)来获取帧率，且由于延迟较高，手动指定帧率，可以根据实际情况调节
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
outfile = 'video_result.mp4'
video_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
video_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
writer = cv2.VideoWriter(outfile, fourcc, fps, (video_width, video_height))

try:
    while(cap.isOpened()):  # 在摄像头打开的情况下循环执行
        ret, frame = cap.read()  # 此处 frame 为 bgr 格式图片

        # 数据前处理
        # img_bgr = cv2.imread(image_path, cv2.IMREAD_COLOR)  # 读入图片
        # img, scale_ratio, pad_size = letterbox(img_bgr, new_shape=[640, 640])  # 对图像进行缩放与填充，保持长宽比
        img, scale_ratio, pad_size = letterbox(frame, new_shape=[640, 640])  # 对图像进行缩放与填充，保持长宽比
        img = img[:, :, ::-1].transpose(2, 0, 1)  # BGR to RGB, HWC to CHW
        img = np.expand_dims(img, 0).astype(np.float32)  # 将形状转换为 channel first (1, 3, 640, 640)，即扩展第一维为 batchsize
        img = np.ascontiguousarray(img) / 255.0  # 转换为内存连续存储的数组
        img = Tensor(img) # 将numpy转为转为Tensor类

        # 模型推理, 得到模型输出
```

```
        model = base.model(modelPath=model_path, deviceId=DEVICE_ID)  # 初始化 base.model 类
        output = model.infer([img])[0]  # 执行推理。输入数据类型：List[base.Tensor]， 返回模型推理输出的
List[base.Tensor]

        # 后处理
        output.to_host()  # 将 Tensor 数据转移到 Host 侧
        output = np.array(output)  # 将数据转为 numpy array 类型
        boxout = nms(torch.tensor(output), conf_thres=0.4, iou_thres=0.5)  # 利用非极大值抑制处理模型输出，
conf_thres 为置信度阈值，iou_thres 为iou阈值
        pred_all = boxout[0].numpy()  # 转换为numpy数组
        scale_coords([640, 640], pred_all[:, :4], frame.shape, ratio_pad=(scale_ratio, pad_size))  # 将推理结果缩
放到原始图片大小
        labels_dict = get_labels_from_txt('./coco_names.txt')  # 得到类别信息，返回序号与类别对应的字典
        img_dw = draw_bbox(pred_all, frame, (0, 255, 0), 2, labels_dict)  # 画出检测框、类别、概率

        # 将推理结果写入视频
        writer.write(img_dw)
except KeyboardInterrupt:
    cap.release()
    writer.release()
finally:
    cap.release()
    writer.release()
# 保存图片到文件
print('save infer result success')
```


3 图像分割应用样例开发介绍（Python）

样例介绍

本文以MindX SDK来开发一个简单的图像分割应用，图像分割模型推理流程如**图1 分割模型推理流程图**所示。

图 3-1 分割模型推理流程图

本例中使用的是**开源数据集**训练的Unet++图片分割模型（参见**ModelZoo-PyTorch-Unet++**）。可以直接使用训练好的开源模型，也可以基于开源模型的源码进行修改、重新训练，还可以基于算法、框架构建适合的模型。

模型的输入数据与输出数据格式：

- 输入数据：RGB格式图片，分辨率为96*96，输入shape为(1,3,96,96)，即（batchsize，channel，height，width），也即每个batch的图片数量、图片的RGB维度、图片高度、图片宽度。
- 输出数据：分割结果灰度图，分辨率为96*96，输入shape为(1,1,96,96)，即（batchsize，channel，height，width），也即每个batch的图片数量、图片的RGB维度、图片高度、图片宽度。

前期准备

步骤1 获取代码文件。

单击**获取链接**或使用wget命令，下载代码文件压缩包，以**root**用户登录开发者套件。

```
wget https://ascend-repo.obs.cn-east-2.myhuaweicloud.com/Atlas%20200I%20DK%20A2/DevKit/models/sdk_cal_samples/unetplusplus_sdk_python_sample.zip
```

步骤2 将“unetplusplus_sdk_python_sample.zip”压缩包上传到开发者套件，解压并进入解压后的目录。

```
unzip unetplusplus_sdk_python_sample.zip
cd unetplusplus_sdk_python_sample
```

代码目录结构如下所示，按照正常开发流程，需要将框架模型文件转换成昇腾AI处理器支持推理的om格式模型文件，鉴于当前是入门内容，用户可直接获取已转换好的om模型进行推理。

```
|-- unetplusplus_sdk_python_sample
   |-- main.py                    # 主程序
   |-- img.png                    # 测试图片
   |-- model
      |-- unetplusplus.om             # om模型
```

步骤3 准备用于推理的图片数据。

如**步骤2**所示文件结构，内置测试图片为“img.png”，用户也可从**开源数据集**中获取其它图片。

----结束

代码解析

开发代码过程中，在“unetplusplus_sdk_python_sample/main.py”文件中已包含读入数据、前处理、推理、后处理等功能，串联整个应用代码逻辑，此处仅对代码进行解析。

1. 在“main.py”文件的开头有如下代码，用于导入需要的第三方库以及MindX SDK推理所需文件。

```
import numpy as np  # 用于对多维数组进行计算
import cv2  # 图片处理三方库，用于对图片进行前后处理
from albumentations.augmentations import transforms  # 数据增强库，此处用于对像素值进行值域变换

from mindx.sdk import Tensor  # mxVision 中的 Tensor 数据结构
from mindx.sdk import base  # mxVision 推理接口
```

2. 初始化资源、定义模型相关变量，如图片路径、模型路径、类别数量、指定device。

```
# 初始化资源和变量
base.mx_init()  # 初始化 mxVision 资源
pic_path = 'img.png'  # 单张图片
model_path = "model/unetplusplus.om"  # 模型路径
num_class = 1  # 类别数量, 需要根据模型结构、任务类别进行改变; 此处只分割出细胞, 即为一分类
device_id = 0  # 指定运算的Device
```

3. 对输入数据进行前处理。先使用OpenCV读入图片，得到三维数组，再进行相应的图片大小缩放、像素值缩放等处理，并将其转化为MindX SDK推理所需要的数据集格式（Tensor类）。

```
# 前处理
img_bgr = cv2.imread(pic_path)  # 读入图片
img = cv2.resize(img_bgr, (96, 96))  # 将原图缩放到 96*96 大小
img = transforms.Normalize().apply(img)  # 将像素值标准化（减去均值除以方差）
img = img.astype('float32') / 255  # 将像素值缩放到 0~1 范围内
img = img.transpose(2, 0, 1)  # 将形状转换为 channel first (3, 96, 96)
img = np.expand_dims(img, 0)  # 将形状转换为 (1, 3, 96, 96)，即扩展第一维为 batchsize
img = np.ascontiguousarray(img)  # 将内存连续排列
img = Tensor(img) # 将numpy转为转为Tensor类
```

4. 使用MindX SDK接口进行模型推理，得到模型输出结果。

```
# 模型推理
model = base.model(modelPath=model_path, deviceId=device_id)  # 初始化 base.model 类
outputs = model.infer([img])  # 执行推理。输入数据类型：List[base.Tensor]， 返回模型推理输出的List[base.Tensor]
```

5. 对模型输出进行后处理。将base.tensor类并转换为利于处理的numpy数组，再进行后处理。后处理步骤即将像素值变换到0~1范围内后，再将其画在原图上，得到最终可以用于显示的的分割结果。

```
# 后处理
model_out_msk = outputs[0]  # 取出模型推理结果, 推理结果形状为 (1, 1, 96, 96),即（batchsize, num_class, height, width）
model_out_msk.to_host()  # 移动tensor到内存中
model_out_msk = np.array(model_out_msk)  # 将 base.Tensor 类转为numpy array
model_out_msk = sigmoid(model_out_msk[0][0])  # 利用 sigmoid 将模型输出变换到 0~1 范围内
img_to_save = plot_mask(img_bgr, model_out_msk)  # 将处理后的输出画在原图上, 并返回
```



```
# 保存图片到文件
cv2.imwrite('result.png', img_to_save)
print('save infer result success')
```

以上步骤为模型推理的整体流程，代码中的函数调用定义在代码最后。

6. 主体函数定义。

   其中sigmoid函数用于将矩阵的每个元素变换到0~1范围内，plot_mask用于将得到的分割结果画在原图上。

   以下代码定义了这两个函数功能。

```
def sigmoid(x):
    y = 1.0 / (1 + np.exp(-x))  # 对矩阵的每个元素执行 1/(1+e^(-x))
    return y

def plot_mask(img, msk):
    """ 将推理得到的 mask 覆盖到原图上 """
    msk = msk + 0.5  # 将像素值范围变换到 0.5~1.5, 有利于下面转为二值图
    msk = cv2.resize(msk, (img.shape[1], img.shape[0]))  # 将 mask 缩放到原图大小
    msk = np.array(msk, np.uint8)  # 转为二值图, 只包含 0 和 1

    # 从 mask 中找到轮廓线, 其中第二个参数为轮廓检测的模式, 第三个参数为轮廓的近似方法
    # cv2.RETR_EXTERNAL 表示只检测外轮廓，cv2.CHAIN_APPROX_SIMPLE 表示压缩水平方向、
    # 垂直方向、对角线方向的元素, 只保留该方向的终点坐标, 例如一个矩形轮廓只需要4个点来保存轮廓信
息
    # contours 为返回的轮廓（list）
    contours, _ = cv2.findContours(msk, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # 在原图上画出轮廓, 其中 img 为原图, contours 为检测到的轮廓列表
    # 第三个参数表示绘制 contours 中的哪条轮廓, -1 表示绘制所有轮廓
    # 第四个参数表示颜色,（0, 0, 255）表示红色, 第五个参数表示轮廓线的宽度
    cv2.drawContours(img, contours, -1, (0, 0, 255), 1)

    # 将轮廓线以内（即分割区域）覆盖上一层红色
    img[..., 2] = np.where(msk == 1, 255, img[..., 2])

    return img
```

运行推理

步骤1 配置环境变量。

- Ubuntu OS：

  ```
  . /usr/local/Ascend/mxVision/set_env.sh
  ```

- openEuler OS：

  ```
  . $HOME/Ascend/mxVision/set_env.sh
  ```

步骤2 运行主程序。

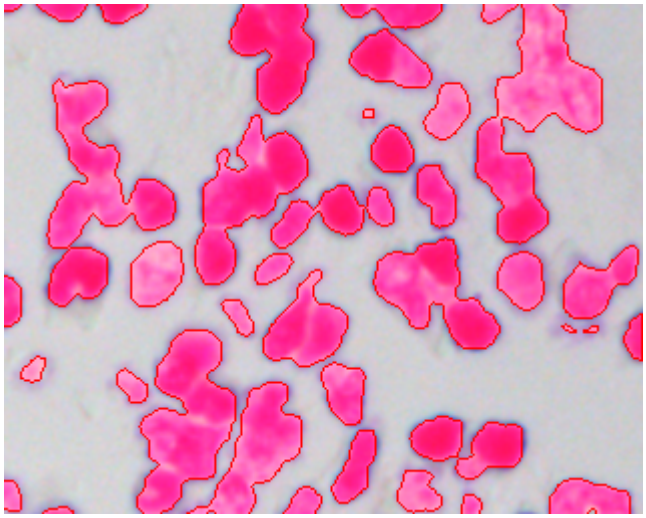
```
python main.py
```

系统回显如下，则表明运行成功。

```
save infer result success
```

推理完成后，在当前文件夹下生成“result.png”文件，如图3-2所示。

图 3-2 “result.png”文件



----结束

样例总结与扩展

以上代码包括以下几个步骤：

1. 前处理：对图片进行缩放到 96*96、标准化、像素值变换、维度转换、连续排列内存、转为Tensor类型等操作。

2. 推理：利用Model或者base.model 初始化模型，并用infer进行推理。

3. 后处理：将模型输出mask进行sigmoid变换，并画到原图上。

MindX SDK接口分类总结：

分类	接口函数	描述
推理相关	base.model(model_path, device_id)	初始化模型
	model.infer([img])	通过输入Tensor列表进行模型推理

理解各个接口含义后，用户可进行灵活运用。除此外，此样例中只示范了图片推理，若需要对视频流数据进行推理，可用两种方式输入视频流数据：USB摄像头、手机摄像头。具体使用方式可参考《**摄像头拉流**》，用户只需将前处理、推理及后处理代码放入摄像头推理代码的循环中即可，注意有些细节地方需进行相应修改。

其中，加入了USB摄像头读写相关代码，并在将图片前处理、推理、后处理相关代码放入了try...except...结构中。其中sigmoid与plot_mask函数与原样例定义相同。运行代码后，可在主目录保存结果视频video_result.mp4。

下面以USB摄像头为例，其他摄像头使用方式可按相应逻辑修改：

```
import numpy as np  # 用于对多维数组进行计算
import cv2  # 图片处理三方库，用于对图片进行前后处理
from albumentations.augmentations import transforms  # 数据增强库，此处用于对像素值进行值域变换

from mindx.sdk import Tensor  # mxVision 中的 Tensor 数据结构
from mindx.sdk import base  # mxVision 推理接口
```



```
# 初始化变量
base.mx_init()  # 初始化 mxVision 资源
pic_path = 'img.png'  # 单张图片
model_path = "model/unetplusplus.om"  # 模型路径
num_class = 1  # 类别数量, 需要根据模型结构、任务类别进行改变; 此处只分割出细胞, 即为一分类
device_id = 0  # 指定运算的Device


# 打开摄像头
cap = cv2.VideoCapture(0)  # 打开摄像头

# 获取保存视频相关变量
fps = cap.get(cv2.CAP_PROP_FPS)
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
outfile = 'video_result.mp4'
video_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
video_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
writer = cv2.VideoWriter(outfile, fourcc, fps, (video_width, video_height))

try:
    while(cap.isOpened()):  # 在摄像头打开的情况下循环执行
        ret, frame = cap.read()  # 此处 frame 为 bgr 格式图片

        # 前处理
        img = cv2.resize(frame, (96, 96))  # 将原图缩放到 96*96 大小
        img = transforms.Normalize().apply(img)  # 将像素值标准化（减去均值除以方差）
        img = img.astype('float32') / 255  # 将像素值缩放到 0~1 范围内
        img = img.transpose(2, 0, 1)  # 将形状转换为 channel first (3, 96, 96)
        img = np.expand_dims(img, 0)  # 将形状转换为 (1, 3, 96, 96)，即扩展第一维为 batchsize
        img = np.ascontiguousarray(img)  # 将内存连续排列
        img = Tensor(img) # 将numpy转为转为Tensor类

        # 模型推理
        model = base.model(modelPath=model_path, deviceId=device_id)  # 初始化 base.model 类
        outputs = model.infer([img])  # 执行推理。输入数据类型：List[base.Tensor]， 返回模型推理输出的
List[base.Tensor]

        # 后处理
        model_out_msk = outputs[0]  # 取出模型推理结果, 推理结果形状为 (1, 1, 96, 96),即（batchsize,
num_class, height, width）
        model_out_msk.to_host()  # 移动tensor到host内存中
        model_out_msk = np.array(model_out_msk)  # 将 base.Tensor 类转为numpy array
        model_out_msk = sigmoid(model_out_msk[0][0])  # 利用 sigmoid 将模型输出变换到 0~1 范围内
        img_to_save = plot_mask(frame, model_out_msk)  # 将处理后的输出画在原图上, 并返回
        print('infer current frame success')

        # 保存图片到文件
        writer.write(img_to_save)  # 将推理结果写入视频

except KeyboardInterrupt:
    cap.release()
    writer.release()
    print('save infer result success')
finally:
    cap.release()
    writer.release()
    print('save infer result success')
```


4 更多代码样例

更多的开发者套件代码样例请访问**昇腾社区应用样例**。