**基于GAN的AR阴影生成开发指导书**

**1.31版本**



华为技术有限公司

目录

[基于GAN的AR阴影生成 2](#_Toc29297303)

[1项目介绍 2](#_Toc29297304)

[2实验目的 2](#_Toc29297305)

[3预备知识 2](#_Toc29297306)

[4实验环境 2](#_Toc29297307)

[5运行AR阴影生成示例 3](#_Toc29297308)

[6实验原理及流程 4](#_Toc29297309)

[6.1实验原理 4](#_Toc29297310)

[6.2实验流程 4](#_Toc29297311)

[7实验任务及步骤 5](#_Toc29297312)

[任务一 模板工程创建 5](#_Toc29297313)

[任务二 模型转换 7](#_Toc29297314)

[任务三 应用代码编写 8](#_Toc29297315)

[任务四 应用运行 13](#_Toc29297316)

# 基于GAN的AR阴影生成

## 1项目介绍

本文档主要介绍基于GAN的AR阴影生成项目代码开发并部署在Atlas 200DK开发板上执行的方法。通过Atlas 200DK开发板来实现AR阴影生成推理实验，从本地读取AR合成图像数据作为输入，由GAN网络推断生成包含虚拟物体阴影的图像，并将结果保存，可用于展示。用户可以通过基于GAN的AR阴影生成项目对Atlas 200DK开发板在AI方面的应用有全面的认识。

## 2实验目的

了解熟悉基于GAN的AR阴影生成应用代码的编写（python语言）。

掌握将应用部署在Atlas 200DK开发板上的操作。

## 3预备知识

具备一定的深度学习理论知识，并对TensorFlow深度学习框架框架有一定了解。

具备Python脚本开发能力、一定的Linux Shell脚本编写能力。

了解Linux操作系统的基本使用。

## 4实验环境

实验环境需要从硬件和软件两个方面进行准备：

（1） 硬件配件准备环境：

使用Atlas 200DK前，需自行购买相关配件，包含制作Atlas 200DK启动系统的micro SD卡、读卡器，与Ubuntu虚拟机相连接的Type-C数据线及摄像头等配件，详细的配件信息如表4.1所示：

表4-1硬件配件清单

|  |  |  |
| --- | --- | --- |
| 配件名称 | 描述 | 推荐型号 |
| SD卡 | 用于制作Atlas 200DK开发者板启动系统。 | 推荐使用经过测试的SD卡：  三星UHS-I U3 CLASS 10 64G  金士顿UHS-I U1 CLASS 10 64G |
| 读卡器 | 使用读卡器制作SD卡的场景。 | 支持USB3.0协议 |
| Type-C连接线 | 用于将开发板与Mind Studio所在服务器通过USB方式连接。 | 支持USB3.0的Type-C连接线 |

（2）软件部署环境：

已在Ubuntu机器上搭建好Mind Studio1.3版本的环境（参考文档：<https://ascend.huawei.com/doc/mindstudio/2.1.0(beta)/zh/zh-cn_topic_0188160926.html>）。

已配置部署好Atlas 200DK环境（参考文档：<https://ascend.huawei.com/doc/Atlas%20200%20DK/1.31.0.0(beta)/zh/zh-cn_topic_0199239495.html>）。

## 5运行AR阴影生成示例

开发者可以先将从此链接<https://pan.baidu.com/s/120f8pHF2_Tn1AqGcFo35zw> (提取码: 2j7b)下载“ARShadowDemo”，其中包含部署指导书文件“部署指导说明书.doc”。读者可以根据该指导书所述运行本案例的示例代码，观察AR阴影生成的效果。

## 6实验原理及流程

### 6.1实验原理

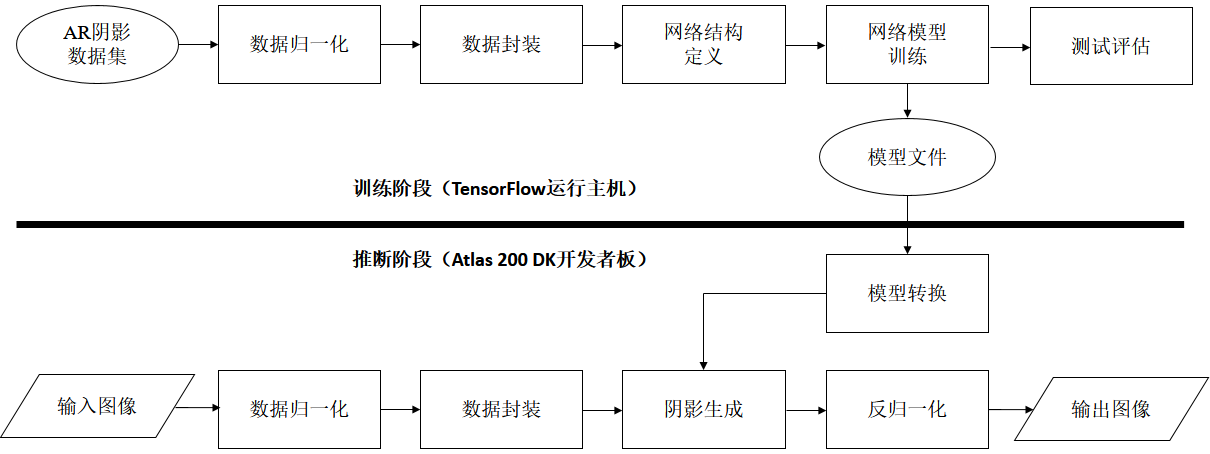


图6.1 基于GAN的AR阴影生成原理图

由原理图可知，本实验的网络训练阶段运行在部署了TensorFlow深度学习框架的主机上，而训练完成的网络模型（包括计算图与权重文件）经过转换后在Atlas 200DK开发者板上执行网络推断过程。网络推断阶段的图像读取、归一化与返归一化、网络推断与结果保存等步骤全部在Atlas 200DK开发者板上执行。输出结果默认保存在开发者板，读者可根据实际需要自行将结果文件拷贝到主机查看或进行后处理。

本实验重点介绍基于GAN的AR阴影生成案例在Atlas 200DK开发板的移植环节，因此读者需要重点网络推断阶段的数据读取、数据处理与格式封装以及网络推断实现过程。

### 6.2实验流程

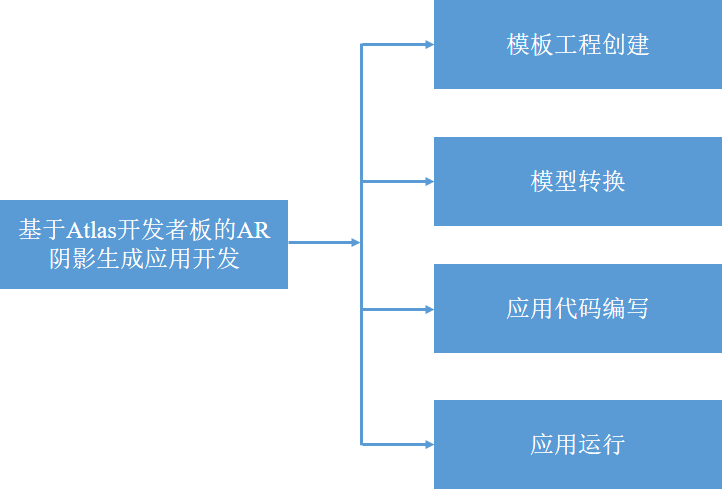


图 6.2 AR阴影生成案例移植流程图

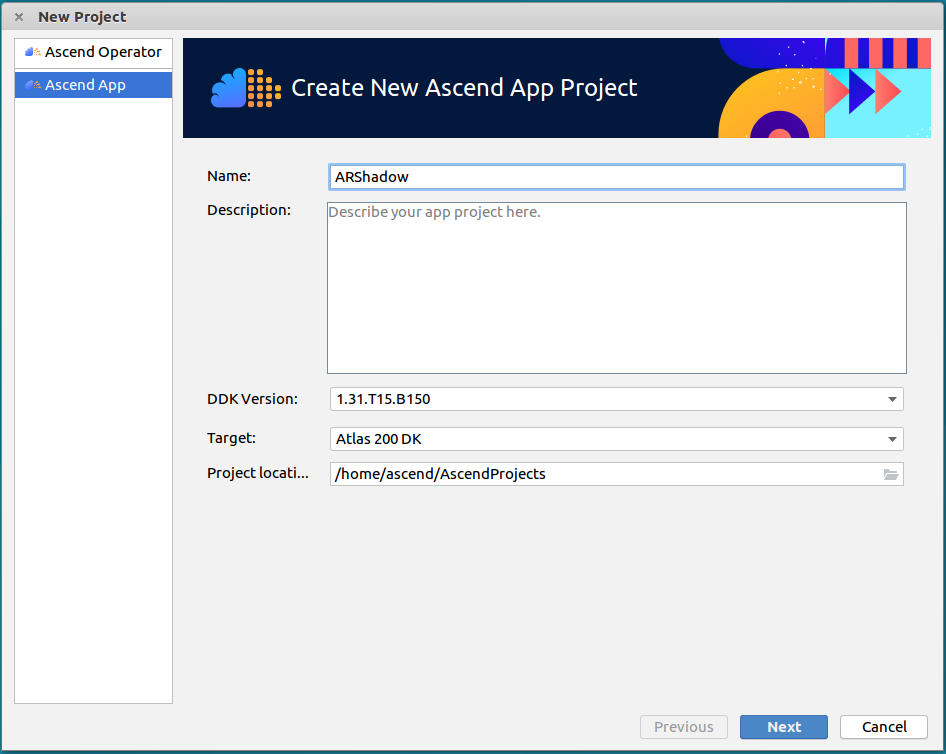
在本实验中，默认已完成硬件环境和软件环境的准备工作，在此基础上进行AR阴影生成应用项目的实验操作，由上图可知，本实验需要分别在Ubuntu主机PC端完成基于Python的应用程序代码的编写工作，以及AR阴影生成GAN网络模型的转换，最后在Atlas 200DK开发板上进行项目部署执行工作。

本案例移植的源代码编写以百度云提供的源代码为例进行说明，实验任务及步骤将围绕图6.2所示四个方面分别展开介绍。

## 7实验任务及步骤

### 任务一 模板工程创建

打开Mind Studio，创建Atlas 200DK场景的自定义的工程（File->New->New Project），工程名没有特殊要求，本案例将其命名为“ARShadow”，如下图所示:



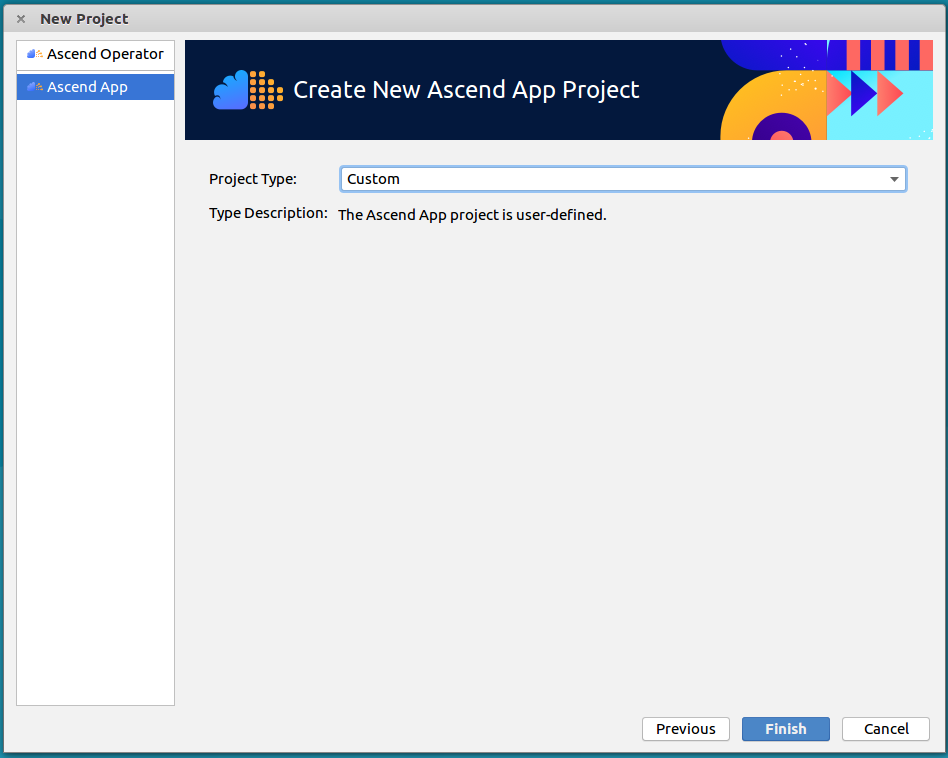


图7.1.1创建自定义工程示意图

如图7.1.2所示，自动生成的模板工程包括main.cpp、graph.config以及CMakeList.txt等文件，将CMakeLists.txt中的环境变量DDK\_PATH修改为DDK\_HOME后可以在Mind Studio菜单选择Build直接编译模板工程，将会生成一个.so共享链接库文件。

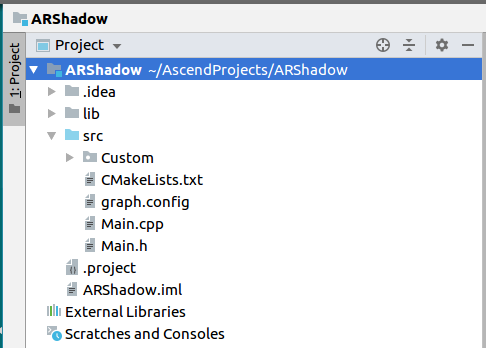


图7.1.2自定义工程源码目录

注意：本案例创建模板工程的目的是为了直观地进行模型转换的设置与完成模型转换，而不进一步使用C++接口进行应用开发，如果读者对C++接口开发感兴趣可自行尝试。

### 任务二 模型转换

此步骤将TensorFlow框架训练的PB模型文件转换为Ascend 310芯片直接支持运行的OM模型。我们启动Mind Studio，打开任务一创建的模板工程ARShadow，从菜单的“Tools-> Model Convert”打开OMG模型管理工具，选择PB文件，设置数据格式为NHWC，其中N代表一个batch中的图像数目，H与W代表图像的高度与宽度，C代表图像的通道数。如图7.2.1，在我们即将运行的网络推断阶段，N设置为1，H与W设置为256，无虚拟阴影AR图像为BGR格式，其通道数设置为3，而虚拟物体掩码为灰度图像，通道数设置为1。本实验不启用Quantization与AIPP预处理设置，而是通过编写应用代码实现图像数据的浮点数转换与归一化处理。设置完成后点击Finish按钮，如果模型转换成功则可以看到如图7.2.2所示的输出日志，OM模型文件默认被保存在~/modelzoo/目录下。

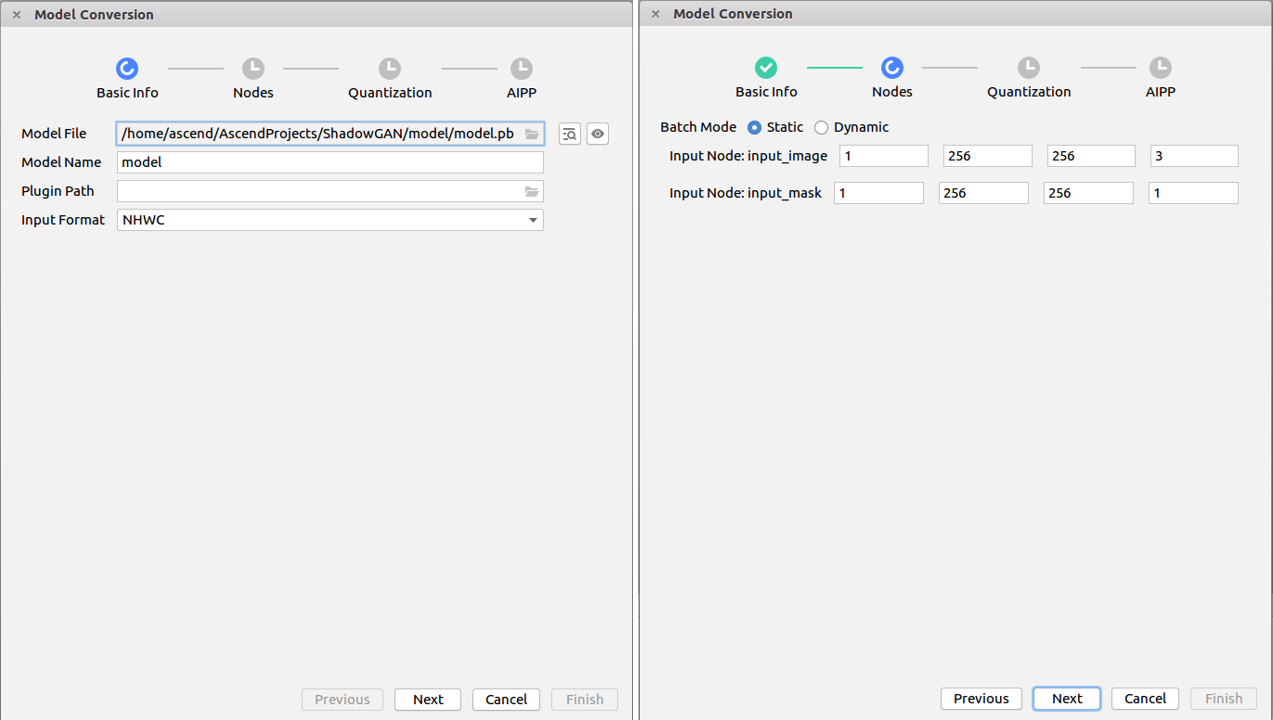


图7.2.1 模型转换参数设置

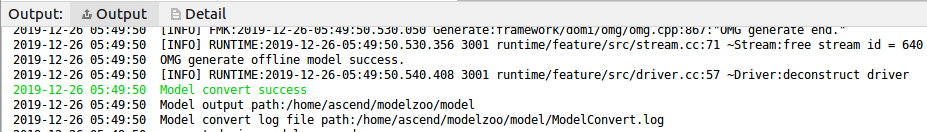


图7.2.2 模型转换成功输出日志

如果模型转换失败，可从日志中查看转换失败的原因，如果遇到不支持的算子，可考虑在TensorFlow框架定义网络结构与训练时用其他支持的算子代替。本案例建议尽量使用支持的算子。

注意，OMG模型管理工具支持转换的TensorFlow模型只有PB模型，如果保存的文件为ckpt文件，需要运行本案例提供的convert.py脚本把ckpt文件转换成PB模型。

### 任务三 应用代码编写

至此，AR阴影生成所需要的网络模型文件已经准备好了，接下来我们编写能在Atlas 200DK开发者板上运行网络推断过程的应用代码。应用代码基于python接口开发，需要在Atlas 200DK开发者板上安装python版的opencv、numpy与hiai。我们参考<https://bbs.huaweicloud.com/forum/thread-28882-1-1.html> 在Atlas开发者板上安装网络推断应用代码所需要的依赖与运行环境。

首先，我们通过ssh命令登录Atlas开发者板，在超级用户模式下用vim命令修改/etc/apt/sources.list，如图7.3.1所示，我们将sources.list的内容替换成中科大的arm软件源。

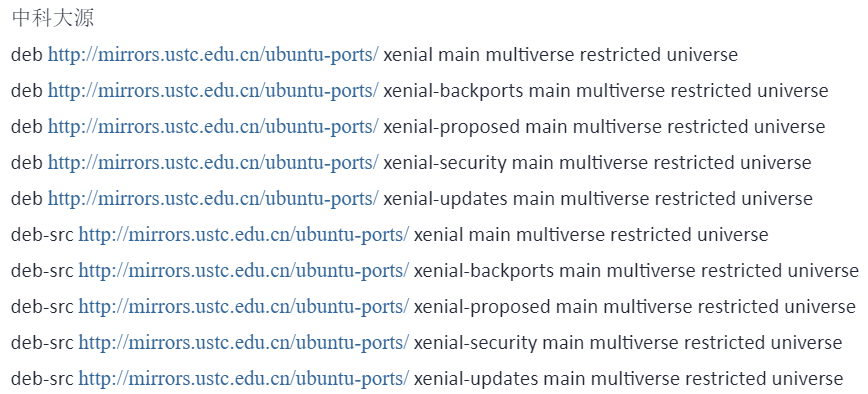


图7.3.1 Atlas 200DK端中科大软件源

修改完毕后按ESC键退出编辑模式，按:wq保存后退出。退出后在终端执行” apt-get update”命令更新软件源，然后运行如下命令安装依赖库：

apt-get install python-setuptools python-dev build-essential python-pip

pip install numpy==1.11.2 enum34==1.1.6 future==0.17.1 funcsigs==1.0.2

依赖库安装成功后，把脚本<https://gitee.com/Atlas200DK/sample-pythonClassifyDemo/raw/master/scripts/python2_hiai_install.sh> 拷贝到Atlas开发者板上，然后在终端执行如下命令安装hiai的python接口：

bash python2\_hiai\_install.sh

最后运行如下命令安装opencv的python接口：

apt-get install python-opencv

安装完毕之后在python下运行如下import命令，若无报错则说明opencv、numpy与hiai的python接口安装成功：

import hiai

import numpy as np

import cv2 as cv

依赖与运行库安装完毕之后我们就可以开始应用程序代码的编写工作了。应用代码实现数据读取、归一化、数据封装、计算图构建、网络模型推断、后处理与反归一化功能。接下来我们详细介绍代码的编写与功能。

首先导入本案例需要使用的程序包，numpy用于实现数据的归一化与内存排布格式转换，cv2用于实现本地图像的读取与保存，hiai用于实现网络计算图的创建与网络推断的执行。

import os  
import os.path as osp  
import numpy as np  
import cv2 as cv  
import time  
import hiai  
from hiai.nn\_tensor\_lib import DataType

接下来定义OM模型的存储目录omModelName，待处理图像数据的目录testDataDir，数据目录下要求包含noshadow与mask两个子目录，分别存放相应的图像与掩码，且需要具有相同的名称。生成结果的保存目录为outputDir。输入图像的尺寸与训练时相同，为256x256。

# OM 模型路径  
omModelName = osp.join('models', 'model.om')

# 图像数据目录  
testDataDir = 'data'

# 保存输出结果的目录  
outputDir = 'output'

# 输入图像的尺寸  
inputWidth = 256  
inputHeight = 256

我们简单地封装读取数据的接口ReadImage，输入参数fileName为图像的路径，channels为图像应该具有的通道数，在本案例中无虚拟阴影AR图像通道数应为3，虚拟物体掩码通道数应为1。对于读取的数据我们做尺寸与维度两个方面的检查：如果图像尺寸与网络接受的输入不符，则通过双三次插值把图像缩放为标准尺寸；opencv默认省略单通道图像的通道数维度（即HW），但是这与网络的输入格式不符，因此这种情况下需要把维度扩充为3个维度（即HWC）。

# 读取图像数据  
def ReadImage(fileName, channels):  
 if channels != 1 and channels != 3:  
 print("[ERROR] Wrong image channel parameter.")  
 return None  
 if channels == 3:  
 image = cv.imread(fileName, cv.IMREAD\_COLOR)  
 else:  
 image = cv.imread(fileName, cv.IMREAD\_GRAYSCALE)  
 if image.shape[0] != inputHeight or image.shape[1] != inputWidth:  
 image = cv.resize(image, (inputWidth, inputHeight), interpolation=cv.INTER\_CUBIC)  
 if image.ndim == 2:  
 image = np.expand\_dims(image, axis=2)  
 return image.astype(np.float32)

我们简单地封装hiai提供的接口函数为CreateGraph实现网络计算流程图的定义。CreateGraph的输入参数model为hiai.AIModelDescription()返回的网络模型描述。在通过hiai.hiai.\_global\_default\_graph\_stack.get\_default\_graph()创建了默认图之后，我们通过网络模型描述model安排其具体的计算流程。计算流程与我们提供的OM模型一一对应，我们在接下来的代码编写中会介绍。

# 创建计算图  
def CreateGraph(model):  
 # 创建图  
 myGraph = hiai.hiai.\_global\_default\_graph\_stack.get\_default\_graph()  
 if myGraph is None:  
 print("[ERROR] Fail to get default graph.")  
 return None  
  
 # 定义计算流程  
 nnTensorList = hiai.NNTensorList()  
 hiai.inference(nnTensorList, model, None)  
  
 if hiai.HiaiPythonStatust.HIAI\_PYTHON\_OK == myGraph.create\_graph():  
 print("[OK] Finish creating graph.")  
 return myGraph  
 else:  
 print("[ERROR] Fail to create graph, please check Davinc log.")  
 return None

定义好了网络计算流程，只要把数据打包成与网络输入相匹配的张量列表就可以直接输入网络执行推断。graphHandle为上述CreateGraph返回的计算图对象，inputTensorList为网络执行推断所需要的输入数据张量列表，本案例包括无虚拟阴影AR图像与虚拟物体的掩码。Graph.proc()返回的网络推断结果为一个列表，本案例的推断结果为该列表的第0项，为NWHC格式，但内存排布为NCHW，后续需要处理成我们所需要的格式。可参考Atlas 200DK的API说明：<https://ascend.huawei.com/doc/Atlas200DK/1.3.0.0/zh/zh-cn_topic_0161025328.html>（API参考->Matrix API参考->Python接口->数据发送与接收->Graph类->Graph.proc接口）。

# 执行网络推断并返回结果  
def GraphInference(graphHandle, inputTensorList):  
 if not isinstance(graphHandle, hiai.Graph):  
 print("[ERROR] graphHandle is not a Graph object.")  
 return None  
  
 # 网络直接输出resultList[0]为NWHC格式，但是内存排布是NCHW  
 resultList = graphHandle.proc(inputTensorList)  
 return resultList

获取了网络推断的结果之后，我们先通过numpy.reshape()将数据的内存排布转换为NCHW，然后再通过numpy.transpose().copy()把NCHW格式转换为我们所需要的NHWC格式。通过反归一化把数据转换为标准的无符号八位整数格式，最后通过cv2.imwrite()把结果保存为BGR彩色图像。

# 保存网络输出的结果  
def SaveResult(resultList, fileName):  
 # 网络直接输出为NWHC格式，但是内存排布是NCHW，需要转换成NCHW  
 resultArray = resultList[0].reshape([1, 3, inputHeight, inputWidth])  
   
 # 再将NCHW格式转换成NHWC格式  
 resultImage = resultArray[0].transpose([1, 2, 0]).copy()  
 resultImage = ((resultImage + 1.0) \* 127.5).astype(np.uint8)  
 cv.imwrite(osp.join(outputDir, fileName), resultImage)

完整的一次网络推断流程首先需要用hiai.AIModelDescription()接口从OM模型创建模型描述，然后调用以上模块完成网络计算流程图的创建。注意到我们的模型支持处理的数据为32位浮点数张量，而我们在模型转换时没有启用预处理，所以在读取数据后我们通过numpy完成浮点类型转换与归一化处理。处理后的数据用hiai.NNTensor()封装成网络模型推断支持的张量类型，该接口函数只支持NCHW格式的输入，需要通过numpy.transpose().copy()进行格式转换。详情可参考<https://ascend.huawei.com/doc/Atlas200DK/1.3.0.0/zh/zh-cn_topic_0161025273.html> （API参考->Matrix API参考->Python接口->基本数据类型->NNTensor类->NNTensor.\_\_init\_\_接口）。如前所述，在我们的案例中输入包括无虚拟阴影AR图像与虚拟物体掩码，完整的应用代码如下所示：

def main():  
 inferenceModel = hiai.AIModelDescription('ShadowGAN', omModelName)  
 myGraph = CreateGraph(inferenceModel)  
 if myGraph is None:  
 exit(0)  
  
 if not osp.exists(outputDir):  
 os.makedirs(outputDir)  
  
 # 读取物虚拟阴影的AR图像与虚拟物体mask  
 inputImage = ReadImage(osp.join(testDataDir, 'noshadow', 'demo.jpg'), channels=3)  
 if inputImage is None:  
 print("[ERROR] No input image.")  
 DestroyGraph()  
 exit(0)  
 inputMask = ReadImage(osp.join(testDataDir, 'mask', 'demo.jpg'), channels=1)  
 if inputMask is None:  
 print("[ERROR] No input mask.")  
 DestroyGraph()  
 exit(0)  
  
 # 归一化到[-1.0, 1.0], mask取反  
 inputImage = inputImage / 127.5 - 1.0  
 inputMask = 1.0 - inputMask / 127.5  
  
 # hiai.NNTensor() 的输入必须是NCHW格式, 输入数据从HWC格式转换成CHW  
 inputImage = inputImage.transpose([2, 0, 1]).copy()  
 inputMask = inputMask.transpose([2, 0, 1]).copy()  
 inputImageTensor = hiai.NNTensor(inputImage, inputWidth, inputHeight, 3,

'input\_image', DataType.FLOAT32\_T,

inputWidth \* inputHeight \* 3)  
 inputMaskTensor = hiai.NNTensor(inputMask, inputWidth, inputHeight, 1,

'input\_mask', DataType.FLOAT32\_T,

inputWidth \* inputHeight \* 1)  
 nntensorList = hiai.NNTensorList([inputImageTensor, inputMaskTensor])  
  
 print("Inference start...")  
 startTime = time.time()  
 resultList = GraphInference(myGraph, nntensorList)  
 if resultList is None:  
 print("[ERROR] Inference failed.")  
 DestroyGraph()  
 exit(0)  
 endTime = time.time()  
 inferenceTime = endTime - startTime  
 SaveResult(resultList, 'demo.jpg')  
  
 DestroyGraph()  
 print("Inference finished. Inference time: %.3fms" % (inferenceTime \* 1000))

至此，单张图像的网络推断代码编写完成，实现从数据目录读取一张图像与掩码，处理完成后在输出目录保存名为demo.jpg的图像结果。完整的代码见云盘代码的demo.py文件，读者可以很容易地修改相应的接口函数将其功能从单张图像处理扩展为批量图像处理。

### 任务四 应用运行

AR阴影生成案例的示例代码运行较为简单，有两种方式可供选择：

1. 通过ssh登录Atlas 200DK开发者板，进入代码目录，在python环境下运行demo.py脚本。这种方法需要事先在数据目录下准备好待处理的图像。运行效果如图7.4.1。
2. 在Ubuntu主机端通过bash运行demo.sh远程执行网络推断生成阴影AR图像再将生成结果传回主机。这种方法要求Ubuntu主机端安装python版的opencv用于输入与输出结果的显示。远程运行输入与输出效果分别如图7.4.2与图7.4.3。

我们在文档“readme-1.31-mindtudio模式部署.doc”中详细提供了远程运行本案例的脚本使用方法与各参数的意义供读者阅读与实验。

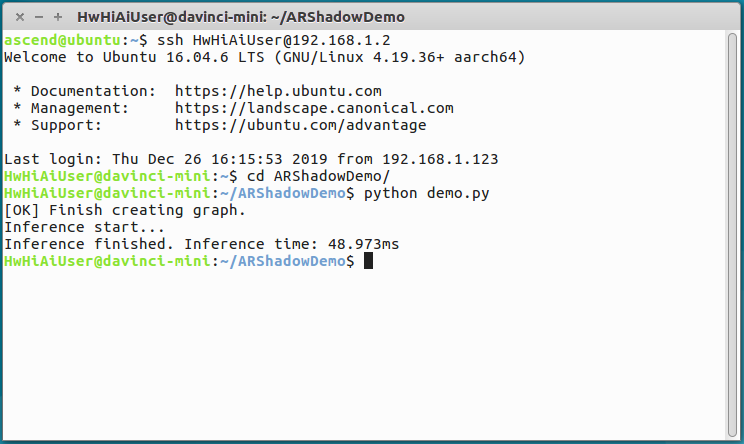


图7.4.1 样例在Atlas 200DK开发者板运行的效果

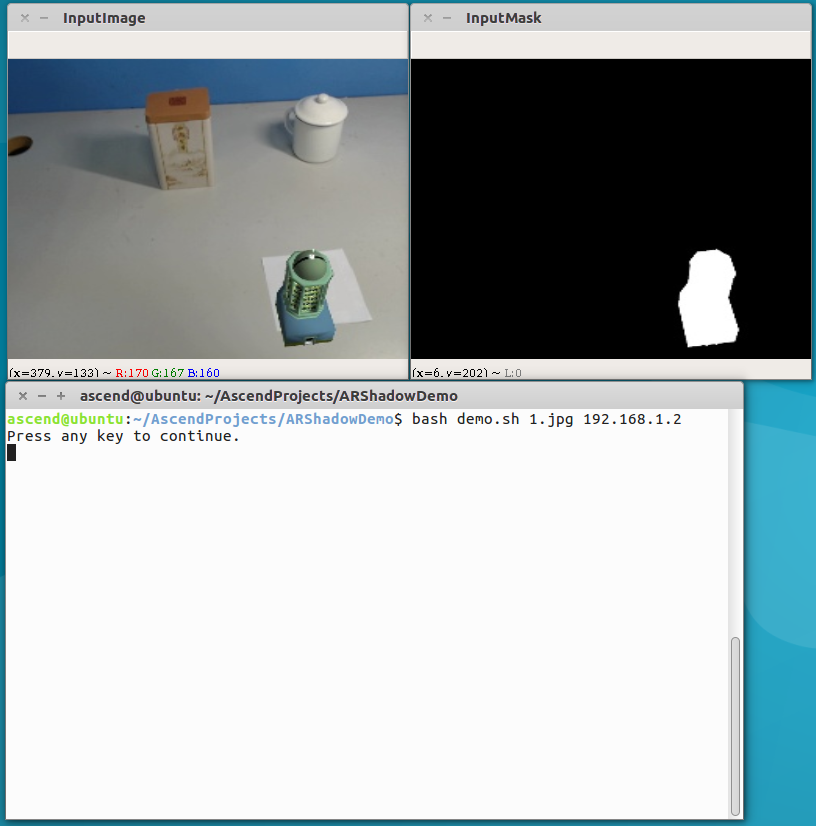


图7.4.2样例在Ubuntu端远程运行输入

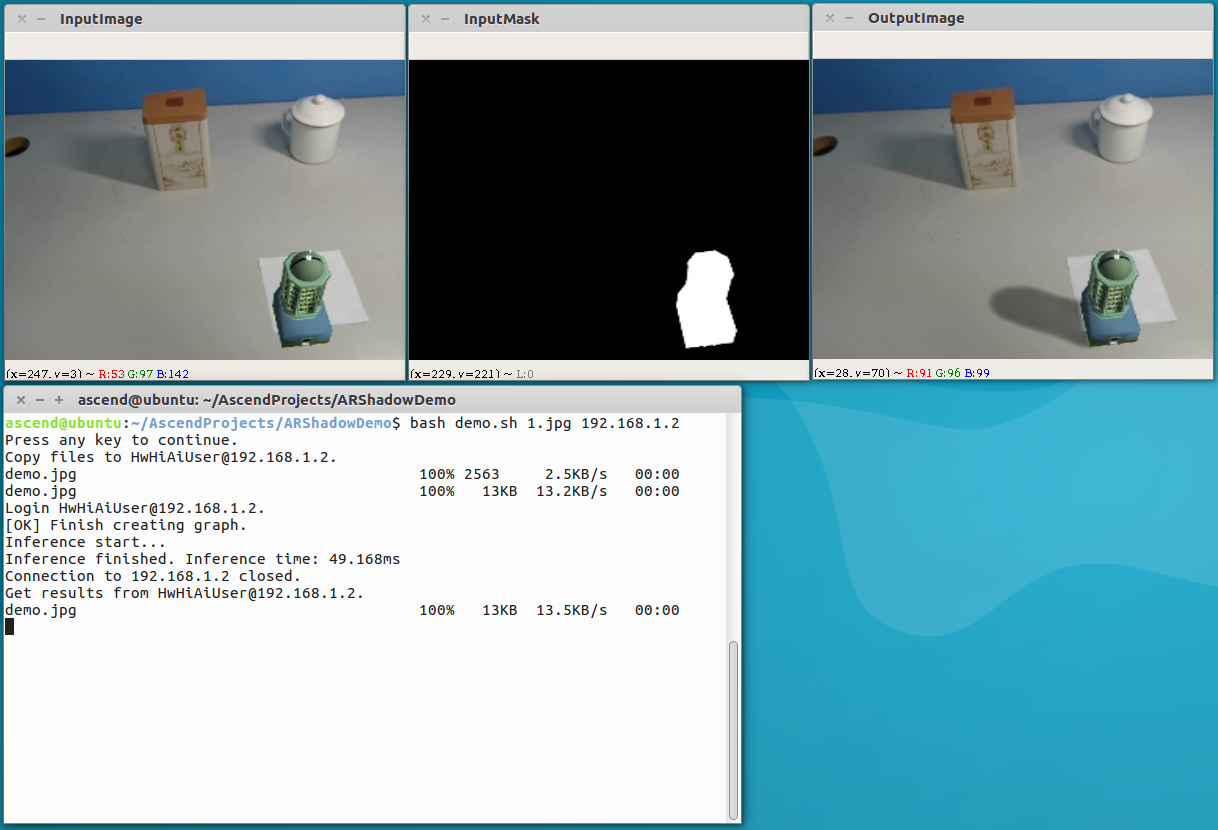


图7.4.2样例在Ubuntu端远程运行输出