华为“智能基座”系列课程

基于昇思MindSpore+Orange AIpro的Pix2Pix实现图像转换训推全流程

版本：1.0



华为技术有限公司

|  |
| --- |
| 版权所有 © 华为技术有限公司 2024。 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

[1 基于昇思MindSpore+Orange AIpro的Pix2Pix实现图像转换训练推全流程 2](#_Toc175757859)

[1.1 实验介绍 2](#_Toc175757860)

[1.1.1 Pix2Pix介绍 2](#_Toc175757861)

[1.1.2 数据集介绍 2](#_Toc175757862)

[1.1.3 基础原理 2](#_Toc175757863)

[1.1.4 实验环境 4](#_Toc175757864)

[1.1.5 实验目的 4](#_Toc175757865)

[1.2 MindSpore模型训练 4](#_Toc175757866)

[1.2.1 环境准备 4](#_Toc175757867)

[1.2.2 准备数据 5](#_Toc175757868)

[1.2.3 创建网络 6](#_Toc175757869)

[1.2.4 模型训练 11](#_Toc175757870)

[1.2.5 模型推理 13](#_Toc175757871)

[1.2.6 模型导出 14](#_Toc175757872)

[1.3 香橙派 AI Pro端侧部署 16](#_Toc175757873)

[1.3.1 环境准备 16](#_Toc175757874)

[1.3.2 模型转换 17](#_Toc175757875)

[1.3.3 推理运行 17](#_Toc175757876)

[1.4 实验小结 21](#_Toc175757877)

# 基于昇思MindSpore+Orange AIpro的Pix2Pix实现图像转换训推全流程

## 实验介绍

### Pix2Pix介绍

Pix2Pix（像素到像素的映射）是基于条件生成对抗网络（cGAN, Condition Generative Adversarial Networks ）实现的一种深度学习图像转换模型，该模型是由Phillip Isola等作者在2017年CVPR上提出的，可以实现语义/标签到真实图片、灰度图到彩色图、航空图到地图、白天到黑夜、线稿图到实物图的转换。Pix2Pix是将cGAN应用于有监督的图像到图像翻译的经典之作，其包括两个模型：生成器和判别器。

传统上，尽管此类任务的目标都是相同的从像素预测像素，但每项都是用单独的专用机器来处理的。而Pix2Pix使用的网络作为一个通用框架，使用相同的架构和目标，只在不同的数据上进行训练，即可得到令人满意的结果，鉴于此许多人已经使用此网络发布了他们自己的艺术作品。

我们使用外墙（facades）数据集，基于Ascend910和Ascend310B算力平台，实现图像转换模型训练+推理部署的全流程实验。首先在Ascend910算力平台使用迁移学习训练得到图像转换模型，接下来将其部署在香橙派AI Pro上，使用Ascend310B算力平台实现端侧推理。

### 数据集介绍

Facades数据集是一个用于图像分割任务的数据集，其中包含了建筑物外墙的图像和相应的二值分割图像。该数据集旨在帮助研究人员和开发者进行建筑物分割相关的算法研究和模型训练。

Facades数据集包含了106个建筑物外墙图像，每个图像的分辨率为512x512像素。对于每个图像，都有相应的二值分割图像，用于标注建筑物的区域。分割图像中的建筑物区域用白色表示（像素值为255），背景区域用黑色表示（像素值为0）。

本实验是采用已经处理好的Facades数据集。

### 基础原理

条件生成对抗网络（cGAN, Condition Generative Adversarial Networks ）的生成器与传统GAN的生成器在原理上有一些区别，cGAN的生成器是将输入图片作为指导信息，由输入图像不断尝试生成用于迷惑判别器的“假”图像，由输入图像转换输出为相应“假”图像的本质是从像素到另一个像素的映射，而传统GAN的生成器是基于一个给定的随机噪声生成图像，输出图像通过其他约束条件控制生成，这是cGAN和GAN的在图像翻译任务中的差异。Pix2Pix中判别器的任务是判断从生成器输出的图像是真实的训练图像还是生成的“假”图像。在生成器与判别器的不断博弈过程中，模型会达到一个平衡点，生成器输出的图像与真实训练数据使得判别器刚好具有50%的概率判断正确。

在教程开始前，首先定义一些在整个过程中需要用到的符号：

观测图像数据

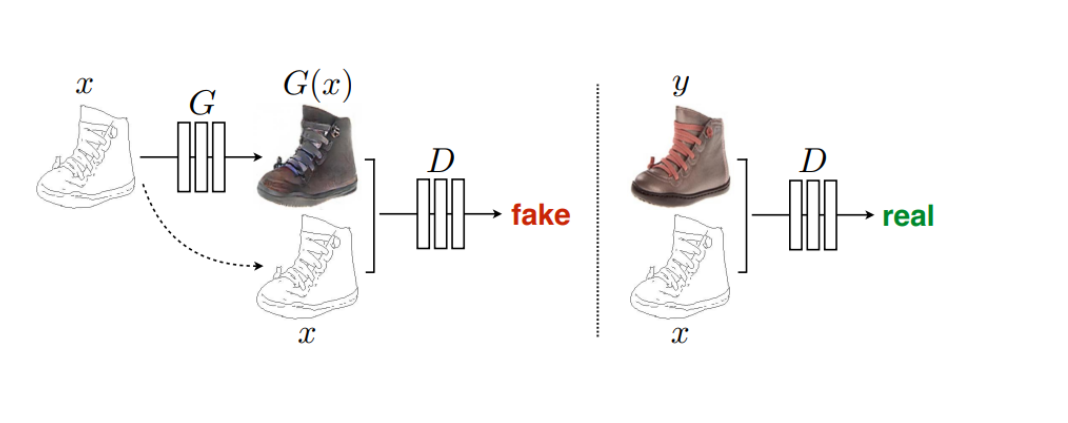
随机噪声数据

生成器网络，给出由观测图像与随机噪声生成的“假”图片，其中来自于训练数据而非生成器。

判别器网络，给出图像判定为真实图像的概率，其中来自于训练数据，来自于生成器。

cGAN的目标可以表示为：

该公式是cGAN的损失函数，想要尽最大努力去正确分类真实图像与“假”图像，也就是使参数最大化；而G尽最大努力用生成的“假”图像欺骗，避免被识破，也就是使参数最小化。cGAN的目标可简化为：



为了对比cGAN和GAN的不同，我们将GAN的目标也进行了说明：

从公式可以看出，GAN直接由随机噪声生成“假”图像，不借助观测图像的任何信息。过去的经验告诉我们，GAN与传统损失混合使用是有好处的，判别器的任务不变，依旧是区分真实图像与“假”图像，但是生成器的任务不仅要欺骗判别器，还要在传统损失的基础上接近训练数据。假设cGAN与L1正则化混合使用，那么有:

进而得到最终目标：

图像转换问题本质上其实就是像素到像素的映射问题，Pix2Pix使用完全一样的网络结构和目标函数，仅更换不同的训练数据集就能分别实现以上的任务。

### 实验环境

本实验分为两部分：

MindSpore模型训练：在（华为云ModelArts的）Ascend910开发环境中执行。

香橙派 AIpro端侧部署：在香橙派 AIpro上的Ascend310B推理环境中执行。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验环境 | 实验平台 | AI 计算框架 | AI处理器/算力 | 软件 |
| MindSpore模型训练 | 华为云 ModelArts | MindSpore2.2.14 | Ascend 910 | Notebook环境，  Python3.9，  MindSpore2.2.14 |
| 香橙派 AI Pro端侧部署 | 香橙派 AIpro | MindSpore2.2.14 | Ascend 310B | Ubuntu 22.04 LTS Arm64，  npu-drive 23.0.RC2， CANN6.2.RC2，  MobaXterm 23 |

环境要求

MindSpore模型训练部分请参考官网：

https://www.mindspore.cn/tutorials/application/zh-CN/r2.2/generative/pix2pix.html

香橙派 AI Pro端侧部署请参考官网：

https://github.com/mindspore-courses/orange-pi-mindspore/tree/master/infer/08-Pix2Pix

### 实验目的

熟悉基于MindSpore的 UNet网络构建；

掌握ATC模型转换工具的基本操作；

掌握AscendCL开发垃圾分类的推理应用；

掌握基于Ascend 910和Ascend310B算力的训练+推理全流程实践。

## MindSpore模型训练

### 环境准备

请参考下方《MindSpore实验环境搭建手册》：



升级MindSpore

本实验需要在MindSpore2.2.14运行，手动升级MindSpore

%env no\_proxy='a.test.com,127.0.0.1,2.2.2.2'

pip install mindspore==2.2.14

安装download库

pip install download -i https://pypi.tuna.tsinghua.edu.cn/simple

### 准备数据

下载数据集

在本实验中，我们将使用[指定数据集](https://mindspore-website.obs.cn-north-4.myhuaweicloud.com/notebook/models/application/dataset_pix2pix.tar)，该数据集是已经经过处理的外墙（facades）数据，可以直接使用mindspore.dataset的方法读取。

from download import download

url = "https://mindspore-website.obs.cn-north-4.myhuaweicloud.com/notebook/models/application/dataset\_pix2pix.tar"

download(url, "./dataset", kind="tar", replace=True)

数据展示

调用Pix2PixDataset和create\_train\_dataset读取训练集，这里我们直接下载已经处理好的数据集。

from mindspore import dataset as ds

import matplotlib.pyplot as plt

dataset = ds.MindDataset("./dataset/dataset\_pix2pix/train.mindrecord", columns\_list=["input\_images", "target\_images"], shuffle=True)

data\_iter = next(dataset.create\_dict\_iterator(output\_numpy=True))

# 可视化部分训练数据

plt.figure(figsize=(10, 3), dpi=140)

for i, image in enumerate(data\_iter['input\_images'][:10], 1):

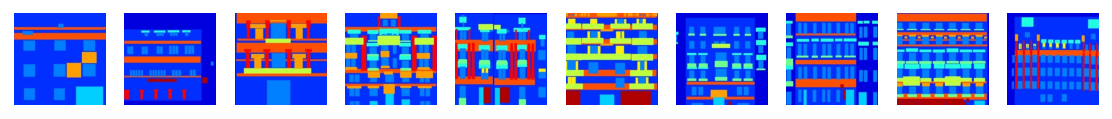
plt.subplot(3, 10, i)

plt.axis("off")

plt.imshow((image.transpose(1, 2, 0) + 1) / 2)

plt.show()

输出：



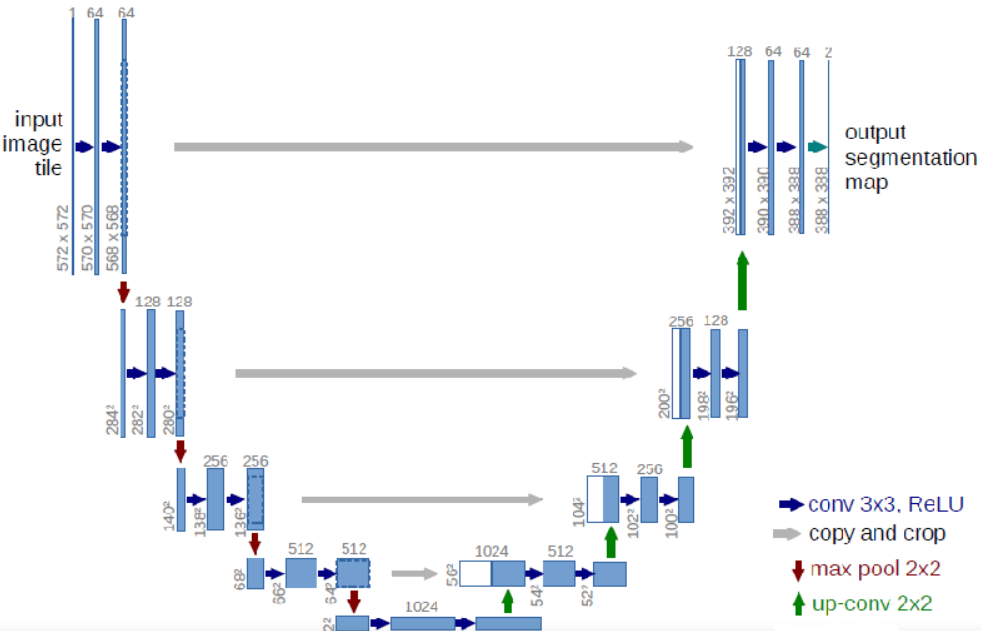
### 创建网络

当处理完数据后，就可以来进行网络的搭建了。网络搭建将逐一详细讨论生成器、判别器和损失函数。生成器G用到的是U-Net结构，输入的轮廓图编码再解码成真是图片，判别器D用到的是作者自己提出来的条件判别器PatchGAN，判别器D的作用是在轮廓图的条件下，对于生成的图片

判断为假，对于真实判断为真。

#### 生成器G结构

U-Net是德国Freiburg大学模式识别和图像处理组提出的一种全卷积结构。它分为两个部分，其中左侧是由卷积和降采样操作组成的压缩路径，右侧是由卷积和上采样组成的扩张路径，扩张的每个网络块的输入由上一层上采样的特征和压缩路径部分的特征拼接而成。网络模型整体是一个U形的结构，因此被叫做U-Net。和常见的先降采样到低维度，再升采样到原始分辨率的编解码结构的网络相比，U-Net的区别是加入skip-connection，对应的feature maps和decode之后的同样大小的feature maps按通道拼一起，用来保留不同分辨率下像素级的细节信息。



定义UNet Skip Connection Block

import mindspore

import mindspore.nn as nn

import mindspore.ops as ops

class UNetSkipConnectionBlock(nn.Cell):

def \_\_init\_\_(self, outer\_nc, inner\_nc, in\_planes=None, dropout=False,

submodule=None, outermost=False, innermost=False, alpha=0.2, norm\_mode='batch'):

super(UNetSkipConnectionBlock, self).\_\_init\_\_()

down\_norm = nn.BatchNorm2d(inner\_nc)

up\_norm = nn.BatchNorm2d(outer\_nc)

use\_bias = False

if norm\_mode == 'instance':

down\_norm = nn.BatchNorm2d(inner\_nc, affine=False)

up\_norm = nn.BatchNorm2d(outer\_nc, affine=False)

use\_bias = True

if in\_planes is None:

in\_planes = outer\_nc

down\_conv = nn.Conv2d(in\_planes, inner\_nc, kernel\_size=4,

stride=2, padding=1, has\_bias=use\_bias, pad\_mode='pad')

down\_relu = nn.LeakyReLU(alpha)

up\_relu = nn.ReLU()

if outermost:

up\_conv = nn.Conv2dTranspose(inner\_nc \* 2, outer\_nc,

kernel\_size=4, stride=2,

padding=1, pad\_mode='pad')

down = [down\_conv]

up = [up\_relu, up\_conv, nn.Tanh()]

model = down + [submodule] + up

elif innermost:

up\_conv = nn.Conv2dTranspose(inner\_nc, outer\_nc,

kernel\_size=4, stride=2,

padding=1, has\_bias=use\_bias, pad\_mode='pad')

down = [down\_relu, down\_conv]

up = [up\_relu, up\_conv, up\_norm]

model = down + up

else:

up\_conv = nn.Conv2dTranspose(inner\_nc \* 2, outer\_nc,

kernel\_size=4, stride=2,

padding=1, has\_bias=use\_bias, pad\_mode='pad')

down = [down\_relu, down\_conv, down\_norm]

up = [up\_relu, up\_conv, up\_norm]

model = down + [submodule] + up

if dropout:

model.append(nn.Dropout(p=0.5))

self.model = nn.SequentialCell(model)

self.skip\_connections = not outermost

def construct(self, x):

out = self.model(x)

if self.skip\_connections:

out = ops.concat((out, x), axis=1)

return out

基于UNet的生成器

class UNetGenerator(nn.Cell):

def \_\_init\_\_(self, in\_planes, out\_planes, ngf=64, n\_layers=8, norm\_mode='bn', dropout=False):

super(UNetGenerator, self).\_\_init\_\_()

unet\_block = UNetSkipConnectionBlock(ngf \* 8, ngf \* 8, in\_planes=None, submodule=None,

norm\_mode=norm\_mode, innermost=True)

for \_ in range(n\_layers - 5):

unet\_block = UNetSkipConnectionBlock(ngf \* 8, ngf \* 8, in\_planes=None, submodule=unet\_block,

norm\_mode=norm\_mode, dropout=dropout)

unet\_block = UNetSkipConnectionBlock(ngf \* 4, ngf \* 8, in\_planes=None, submodule=unet\_block,

norm\_mode=norm\_mode)

unet\_block = UNetSkipConnectionBlock(ngf \* 2, ngf \* 4, in\_planes=None, submodule=unet\_block,

norm\_mode=norm\_mode)

unet\_block = UNetSkipConnectionBlock(ngf, ngf \* 2, in\_planes=None, submodule=unet\_block,

norm\_mode=norm\_mode)

self.model = UNetSkipConnectionBlock(out\_planes, ngf, in\_planes=in\_planes, submodule=unet\_block,

outermost=True, norm\_mode=norm\_mode)

def construct(self, x):

return self.model(x)

备注：原始cGAN的输入是条件x和噪声z两种信息，这里的生成器只使用了条件信息，因此不能生成多样性的结果。因此Pix2Pix在训练和测试时都使用了dropout，这样可以生成多样性的结果。

#### 基于PatchGAN的判别器

判别器使用的PatchGAN结构，可看做卷积。生成的矩阵中的每个点代表原图的一小块区域（patch）。通过矩阵中的各个值来判断原图中对应每个Patch的真假。

import mindspore.nn as nn

class ConvNormRelu(nn.Cell):

def \_\_init\_\_(self,

in\_planes,

out\_planes,

kernel\_size=4,

stride=2,

alpha=0.2,

norm\_mode='batch',

pad\_mode='CONSTANT',

use\_relu=True,

padding=None):

super(ConvNormRelu, self).\_\_init\_\_()

norm = nn.BatchNorm2d(out\_planes)

if norm\_mode == 'instance':

norm = nn.BatchNorm2d(out\_planes, affine=False)

has\_bias = (norm\_mode == 'instance')

if not padding:

padding = (kernel\_size - 1) // 2

if pad\_mode == 'CONSTANT':

conv = nn.Conv2d(in\_planes, out\_planes, kernel\_size, stride, pad\_mode='pad',

has\_bias=has\_bias, padding=padding)

layers = [conv, norm]

else:

paddings = ((0, 0), (0, 0), (padding, padding), (padding, padding))

pad = nn.Pad(paddings=paddings, mode=pad\_mode)

conv = nn.Conv2d(in\_planes, out\_planes, kernel\_size, stride, pad\_mode='pad', has\_bias=has\_bias)

layers = [pad, conv, norm]

if use\_relu:

relu = nn.ReLU()

if alpha > 0:

relu = nn.LeakyReLU(alpha)

layers.append(relu)

self.features = nn.SequentialCell(layers)

def construct(self, x):

output = self.features(x)

return output

class Discriminator(nn.Cell):

def \_\_init\_\_(self, in\_planes=3, ndf=64, n\_layers=3, alpha=0.2, norm\_mode='batch'):

super(Discriminator, self).\_\_init\_\_()

kernel\_size = 4

layers = [

nn.Conv2d(in\_planes, ndf, kernel\_size, 2, pad\_mode='pad', padding=1),

nn.LeakyReLU(alpha)

]

nf\_mult = ndf

for i in range(1, n\_layers):

nf\_mult\_prev = nf\_mult

nf\_mult = min(2 \*\* i, 8) \* ndf

layers.append(ConvNormRelu(nf\_mult\_prev, nf\_mult, kernel\_size, 2, alpha, norm\_mode, padding=1))

nf\_mult\_prev = nf\_mult

nf\_mult = min(2 \*\* n\_layers, 8) \* ndf

layers.append(ConvNormRelu(nf\_mult\_prev, nf\_mult, kernel\_size, 1, alpha, norm\_mode, padding=1))

layers.append(nn.Conv2d(nf\_mult, 1, kernel\_size, 1, pad\_mode='pad', padding=1))

self.features = nn.SequentialCell(layers)

def construct(self, x, y):

x\_y = ops.concat((x, y), axis=1)

output = self.features(x\_y)

return output

#### Pix2Pix的生成器和判别器初始化

import mindspore.nn as nn

from mindspore.common import initializer as init

g\_in\_planes = 3

g\_out\_planes = 3

g\_ngf = 64

g\_layers = 8

d\_in\_planes = 6

d\_ndf = 64

d\_layers = 3

alpha = 0.2

init\_gain = 0.02

init\_type = 'normal'

net\_generator = UNetGenerator(in\_planes=g\_in\_planes, out\_planes=g\_out\_planes,

ngf=g\_ngf, n\_layers=g\_layers)

for \_, cell in net\_generator.cells\_and\_names():

if isinstance(cell, (nn.Conv2d, nn.Conv2dTranspose)):

if init\_type == 'normal':

cell.weight.set\_data(init.initializer(init.Normal(init\_gain), cell.weight.shape))

elif init\_type == 'xavier':

cell.weight.set\_data(init.initializer(init.XavierUniform(init\_gain), cell.weight.shape))

elif init\_type == 'constant':

cell.weight.set\_data(init.initializer(0.001, cell.weight.shape))

else:

raise NotImplementedError('initialization method [%s] is not implemented' % init\_type)

elif isinstance(cell, nn.BatchNorm2d):

cell.gamma.set\_data(init.initializer('ones', cell.gamma.shape))

cell.beta.set\_data(init.initializer('zeros', cell.beta.shape))

net\_discriminator = Discriminator(in\_planes=d\_in\_planes, ndf=d\_ndf,

alpha=alpha, n\_layers=d\_layers)

for \_, cell in net\_discriminator.cells\_and\_names():

if isinstance(cell, (nn.Conv2d, nn.Conv2dTranspose)):

if init\_type == 'normal':

cell.weight.set\_data(init.initializer(init.Normal(init\_gain), cell.weight.shape))

elif init\_type == 'xavier':

cell.weight.set\_data(init.initializer(init.XavierUniform(init\_gain), cell.weight.shape))

elif init\_type == 'constant':

cell.weight.set\_data(init.initializer(0.001, cell.weight.shape))

else:

raise NotImplementedError('initialization method [%s] is not implemented' % init\_type)

elif isinstance(cell, nn.BatchNorm2d):

cell.gamma.set\_data(init.initializer('ones', cell.gamma.shape))

cell.beta.set\_data(init.initializer('zeros', cell.beta.shape))

class Pix2Pix(nn.Cell):

"""Pix2Pix模型网络"""

def \_\_init\_\_(self, discriminator, generator):

super(Pix2Pix, self).\_\_init\_\_(auto\_prefix=True)

self.net\_discriminator = discriminator

self.net\_generator = generator

def construct(self, reala):

fakeb = self.net\_generator(reala)

return fakeb

### 模型训练

训练分为两个主要部分：训练判别器和训练生成器。训练判别器的目的是最大程度地提高判别图像真伪的概率。训练生成器是希望能产生更好的虚假图像。在这两个部分中，分别获取训练过程中的损失，并在每个周期结束时进行统计（由于时间问题，训练过程只进行了100个epoch）。

import numpy as np

import os

import datetime

from mindspore import value\_and\_grad, Tensor

epoch\_num = 100

ckpt\_dir = "results/ckpt"

dataset\_size = 400

val\_pic\_size = 256

lr = 0.0002

n\_epochs = 100

n\_epochs\_decay = 100

def get\_lr():

lrs = [lr] \* dataset\_size \* n\_epochs

lr\_epoch = 0

for epoch in range(n\_epochs\_decay):

lr\_epoch = lr \* (n\_epochs\_decay - epoch) / n\_epochs\_decay

lrs += [lr\_epoch] \* dataset\_size

lrs += [lr\_epoch] \* dataset\_size \* (epoch\_num - n\_epochs\_decay - n\_epochs)

return Tensor(np.array(lrs).astype(np.float32))

dataset = ds.MindDataset("./dataset/dataset\_pix2pix/train.mindrecord", columns\_list=["input\_images", "target\_images"], shuffle=True, num\_parallel\_workers=16)

steps\_per\_epoch = dataset.get\_dataset\_size()

loss\_f = nn.BCEWithLogitsLoss()

l1\_loss = nn.L1Loss()

def forword\_dis(reala, realb):

lambda\_dis = 0.5

fakeb = net\_generator(reala)

pred0 = net\_discriminator(reala, fakeb)

pred1 = net\_discriminator(reala, realb)

loss\_d = loss\_f(pred1, ops.ones\_like(pred1)) + loss\_f(pred0, ops.zeros\_like(pred0))

loss\_dis = loss\_d \* lambda\_dis

return loss\_dis

def forword\_gan(reala, realb):

lambda\_gan = 0.5

lambda\_l1 = 100

fakeb = net\_generator(reala)

pred0 = net\_discriminator(reala, fakeb)

loss\_1 = loss\_f(pred0, ops.ones\_like(pred0))

loss\_2 = l1\_loss(fakeb, realb)

loss\_gan = loss\_1 \* lambda\_gan + loss\_2 \* lambda\_l1

return loss\_gan

d\_opt = nn.Adam(net\_discriminator.trainable\_params(), learning\_rate=get\_lr(),

beta1=0.5, beta2=0.999, loss\_scale=1)

g\_opt = nn.Adam(net\_generator.trainable\_params(), learning\_rate=get\_lr(),

beta1=0.5, beta2=0.999, loss\_scale=1)

grad\_d = value\_and\_grad(forword\_dis, None, net\_discriminator.trainable\_params())

grad\_g = value\_and\_grad(forword\_gan, None, net\_generator.trainable\_params())

def train\_step(reala, realb):

loss\_dis, d\_grads = grad\_d(reala, realb)

loss\_gan, g\_grads = grad\_g(reala, realb)

d\_opt(d\_grads)

g\_opt(g\_grads)

return loss\_dis, loss\_gan

if not os.path.isdir(ckpt\_dir):

os.makedirs(ckpt\_dir)

g\_losses = []

d\_losses = []

data\_loader = dataset.create\_dict\_iterator(output\_numpy=True, num\_epochs=epoch\_num)

for epoch in range(epoch\_num):

for i, data in enumerate(data\_loader):

start\_time = datetime.datetime.now()

input\_image = Tensor(data["input\_images"])

target\_image = Tensor(data["target\_images"])

dis\_loss, gen\_loss = train\_step(input\_image, target\_image)

end\_time = datetime.datetime.now()

delta = (end\_time - start\_time).microseconds

if i % 2 == 0:

print("ms per step:{:.2f} epoch:{}/{} step:{}/{} Dloss:{:.4f} Gloss:{:.4f} ".format((delta / 1000), (epoch + 1), (epoch\_num), i, steps\_per\_epoch, float(dis\_loss), float(gen\_loss)))

d\_losses.append(dis\_loss.asnumpy())

g\_losses.append(gen\_loss.asnumpy())

if (epoch + 1) == epoch\_num:

mindspore.save\_checkpoint(net\_generator, ckpt\_dir + "Generator.ckpt")

输出：

ms per step:532.31 epoch:1/100 step:0/25 Dloss:0.6940 Gloss:38.1245

ms per step:304.35 epoch:1/100 step:2/25 Dloss:0.6489 Gloss:39.4826

ms per step:299.15 epoch:1/100 step:4/25 Dloss:0.5506 Gloss:36.7634

ms per step:301.06 epoch:1/100 step:6/25 Dloss:1.6741 Gloss:47.7600

ms per step:299.72 epoch:1/100 step:8/25 Dloss:0.4604 Gloss:39.7121

...... ......

ms per step:290.44 epoch:100/100 step:16/25 Dloss:0.6009 Gloss:9.1915

ms per step:289.95 epoch:100/100 step:18/25 Dloss:0.4617 Gloss:9.8740

ms per step:290.24 epoch:100/100 step:20/25 Dloss:0.4402 Gloss:8.2490

ms per step:287.70 epoch:100/100 step:22/25 Dloss:0.3814 Gloss:9.3652

ms per step:289.41 epoch:100/100 step:24/25 Dloss:0.4199 Gloss:9.2418

### 模型推理

获取上述训练过程完成后的ckpt文件，通过load\_checkpoint和load\_param\_into\_net将ckpt中的权重参数导入到模型中，获取数据进行推理并对推理的效果图进行演示

from mindspore import load\_checkpoint, load\_param\_into\_net

param\_g = load\_checkpoint(ckpt\_dir + "Generator.ckpt")

load\_param\_into\_net(net\_generator, param\_g)

dataset = ds.MindDataset("./dataset/dataset\_pix2pix/train.mindrecord", columns\_list=["input\_images", "target\_images"], shuffle=True)

data\_iter = next(dataset.create\_dict\_iterator())

predict\_show = net\_generator(data\_iter["input\_images"])

plt.figure(figsize=(10, 3), dpi=140)

for i in range(10):

plt.subplot(2, 10, i + 1)

plt.imshow((data\_iter["input\_images"][i].asnumpy().transpose(1, 2, 0) + 1) / 2)

plt.axis("off")

plt.subplots\_adjust(wspace=0.05, hspace=0.02)

plt.subplot(2, 10, i + 11)

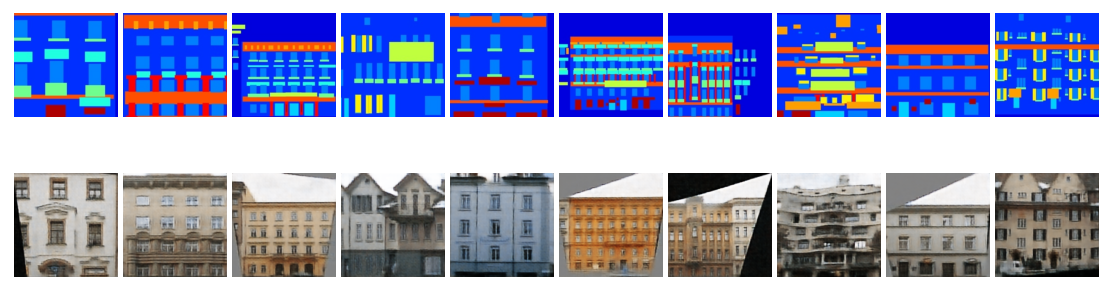
plt.imshow((predict\_show[i].asnumpy().transpose(1, 2, 0) + 1) / 2)

plt.axis("off")

plt.subplots\_adjust(wspace=0.05, hspace=0.02)

plt.show()

输出：



### 模型导出

导出MINDIR文件

指定CKPT转MINDIR格式，用于后续在香橙派 AI Pro上的模型转换与推理。

from mindspore import load\_checkpoint, load\_param\_into\_net

param\_g = load\_checkpoint(ckpt\_dir + "Generator.ckpt")

load\_param\_into\_net(net\_generator, param\_g)

inputs =Tensor(np.ones([16,3,256,256]).astype(np.float32))

mindspore.export(net\_generator,inputs,file\_name='pix2pix',file\_format="MINDIR")

保存到OBS

因模型文件超出100MB，需用OBS中转下载。

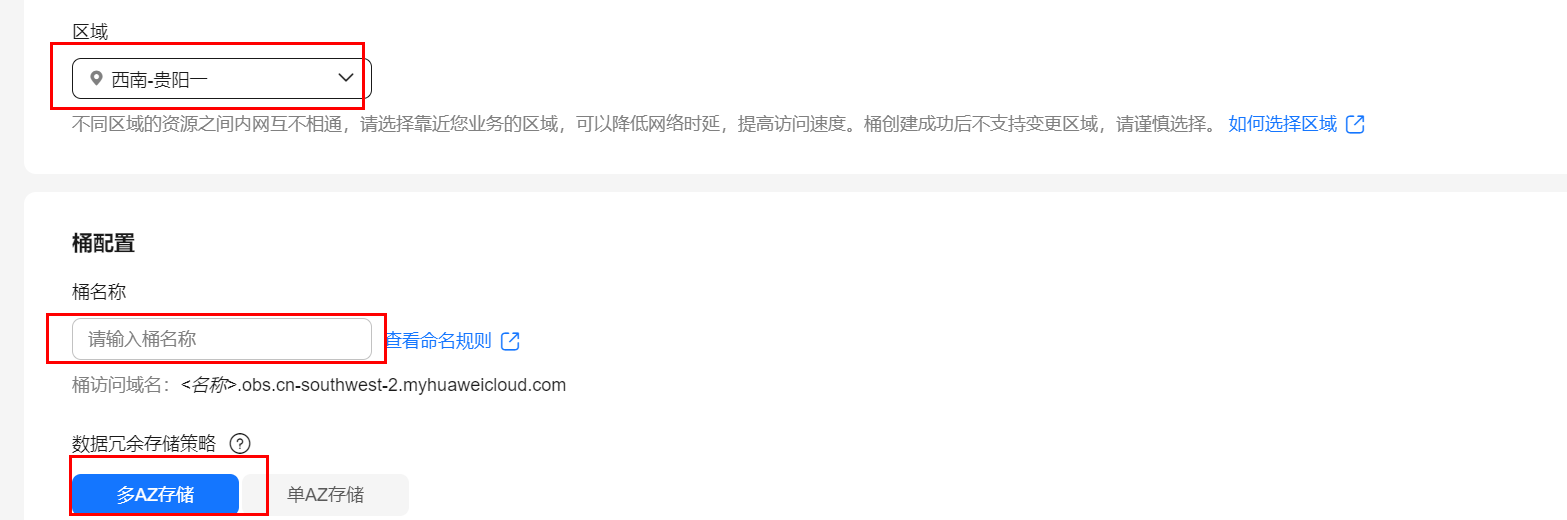
创建OBS桶

在[华为云OBS主页](https://www.huaweicloud.com/product/obs.html)，点击“管理控制台”进入OBS的管理页面。

在页面右上角选择“创建桶”：



进入OBS创建页面，“区域”选择“西南-贵阳一”，“桶名称”自定义：



其他选项选择默认配置即可，之后选择“立即创建”，创建成功之后即可在桶列表查看。



在OBS桶创建文件夹

点击“新建文件夹”并命名，点击“确定”。



mindir文件保存到OBS桶

在notebook中输入如下命令, 模型会保存在OBS桶内（红色字体需要更换自己桶名称）

import moxing as mox

mox.file.copy\_parallel('pix2pix.mindir', 'obs://pix2pix22/mindir/pix2pix.mindir')

mindir 文件下载

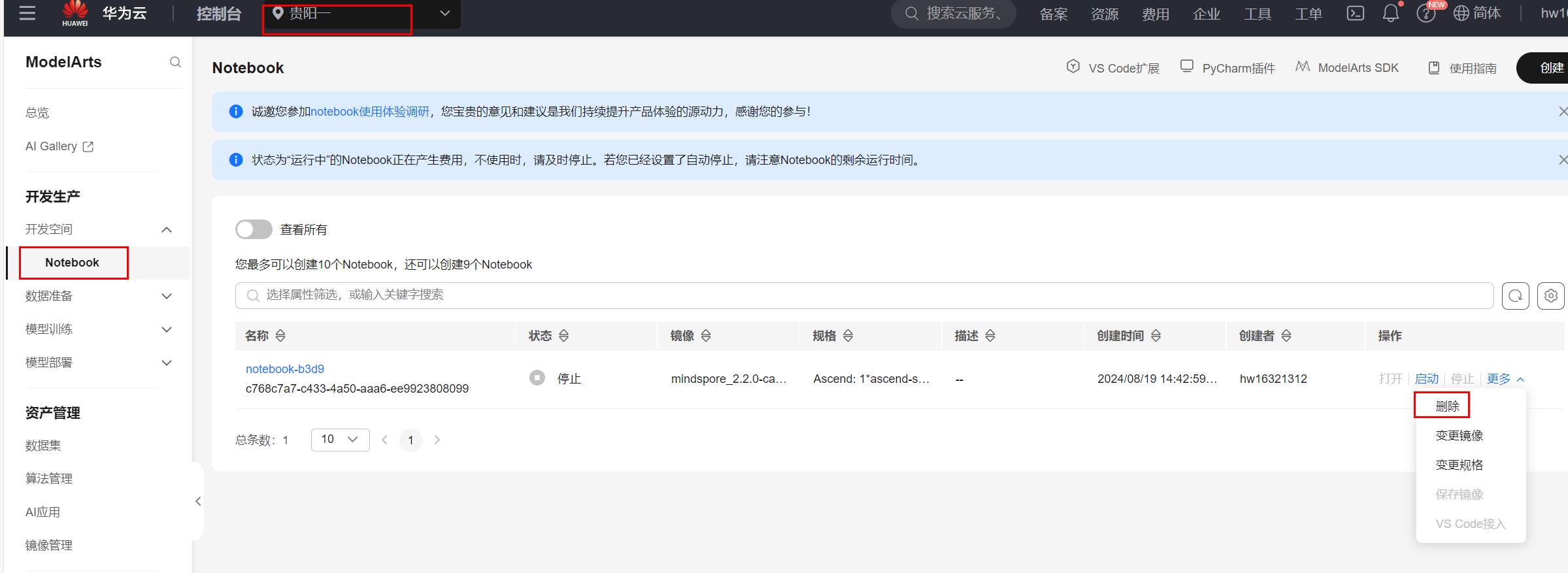
将保存的pix2pix.mindir模型下载到本地,用于后续的推理实验。



关闭云环境

实验完成之后，请及时关闭华为云ModelArts的Notebook开发环境，避免资源浪费。

关闭方式：登录[华为云ModelArts控制台](https://console.huaweicloud.com/modelarts/?region=cn-southwest-2#/dev-container)，在“操作”栏选择“停止”或“更多—>删除”操作。



及时关闭/删除云环境

## 香橙派 AI Pro端侧部署

本实验使用昇腾CANN在香橙派 AI Pro上开发垃圾分类推理应用，首先对模型进行模型转换，接下来使用ACLLite开发推理代码，最后利用plt将风格转换后的图象可视化。

实验主要步骤：

初始化acl资源：在调用acl相关资源时，必须先初始化AscendCL，否则可能会导致后续系统内部资源初始化出错。

对图片进行前处理：在此样例中，我们首先根据图片路径，使用mindspore.dataset.MindDataset接口下加载测试用数据集。然后基于数据集对象使用mindspore中的create\_dict\_iterator接口创建数据迭代器。

推理：利用AclLiteModel.execute接口对数据进行推理。

可视化图片：利用plt将结果画出。

### 环境准备

本实验是在新一代开发者套件香橙派 AI Pro上执行，因此需要部署香橙派 AI Pro的实验环境，请参考《手把手教你搭建Orange Pi AI Pro开发环境》实验手册。



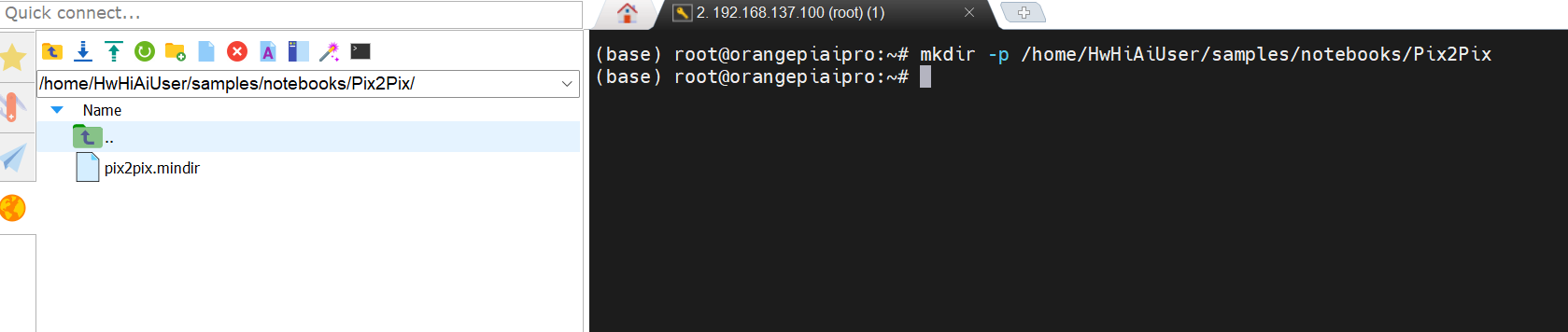
按上面手册完成：登录开发板、开发板网络连接、AclLite安装、MindSpore版本升级为2.2.14。

### 模型转换

上传mindir模型文件

在“/home/HwHiAiUser/samples/noteboooks”目录下创建Pix2Pix文件夹，将训练获得的mindir模型文件放入该文件夹。

mkdir -p /home/HwHiAiUser/samples/notebooks/Pix2Pix



mindir模型文件转换为om模型

在“/home/HwHiAiUser/samples/noteboooks”目录下运行如下命令，生成om模型文件。

#获取bash.sh文件

wget <https://mindspore-courses.obs.cn-north-4.myhuaweicloud.com/orange-pi-mindspore/package/bash.sh>

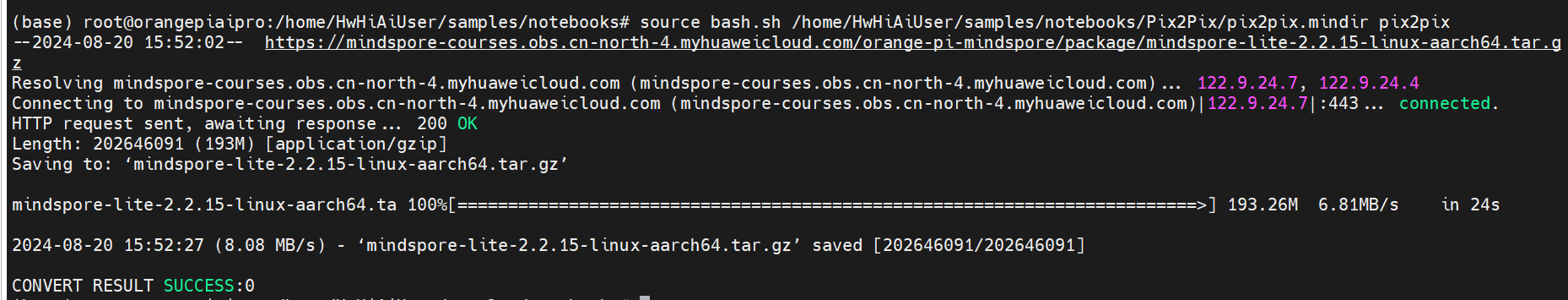
#执行bash.sh文件

source bash.sh /home/HwHiAiUser/samples/notebooks/Pix2Pix/pix2pix.mindir /home/HwHiAiUser/samples/notebooks/Pix2Pix/pix2pix

注：bash.sh文件执行时需要传入两个参数，如上述第二个命令所示：

第一个参数是开发板上存放的MINDIR文件的绝对路径；

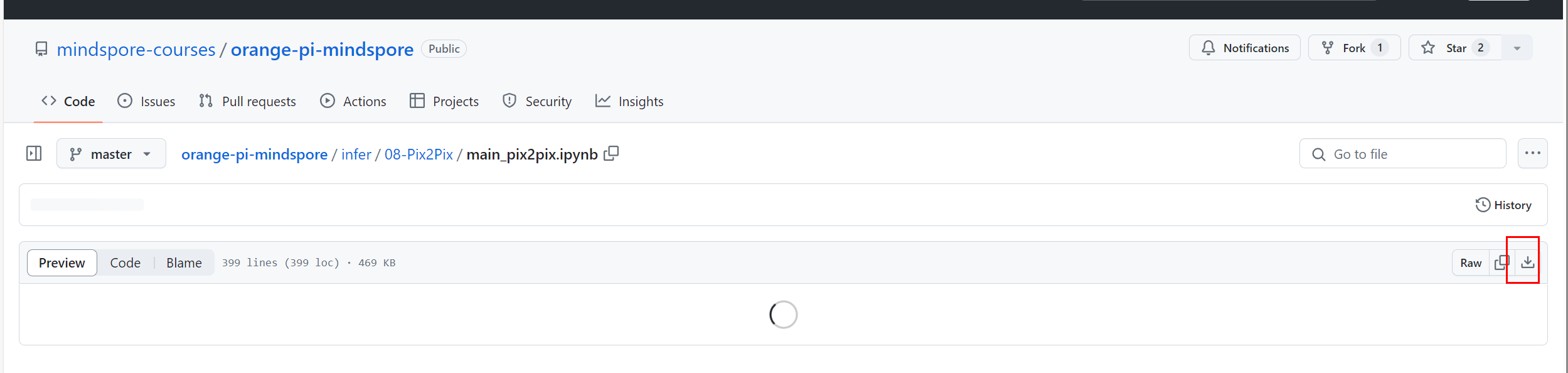
第二个参数是生成的om文件的名称及路径；



### 推理运行

下载推理代码

<https://github.com/mindspore-courses/orange-pi-mindspore/blob/master/infer/08-Pix2Pix/main_pix2pix.ipynb>



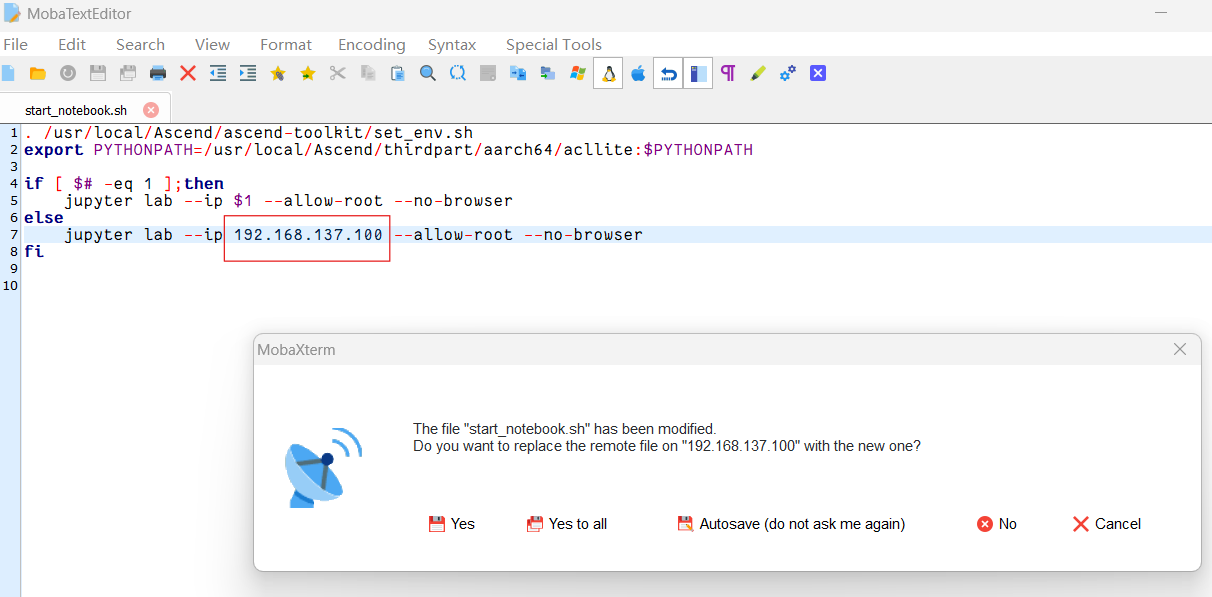
下载后上传到Pix2Pix文件夹下



启动notebook运行环境执行推理应用

修改ip参数

打开start\_notebook.sh文件，修改ip参数为192.168.137.100，如下所示：



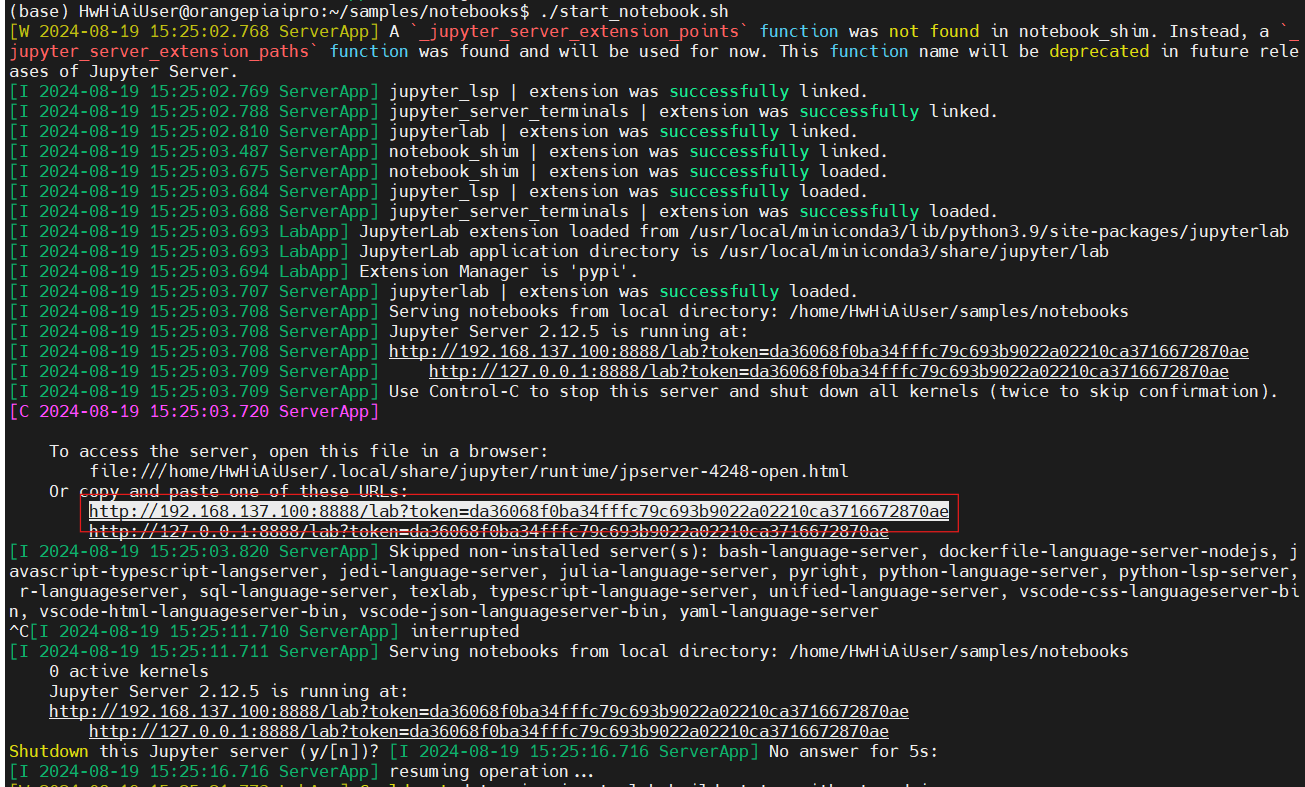
运行start\_notebook.sh文件

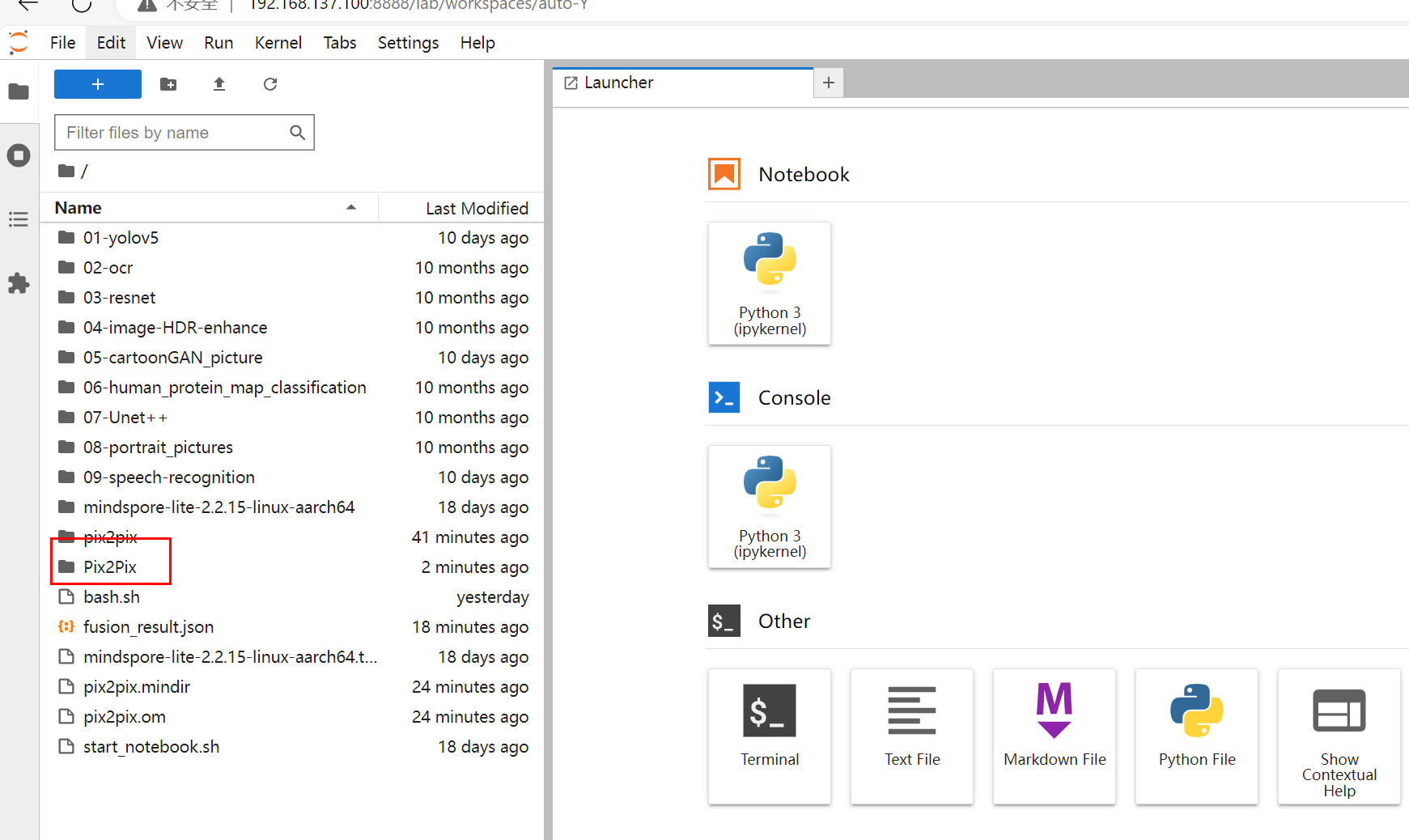
使用如下命令运行start\_notebook.sh文件

cd /home/HwHiAiUser/samples/notebooks

./start\_notebook.sh

打开notebook运行环境，可以看到创建的Pix2Pix项目文件夹。



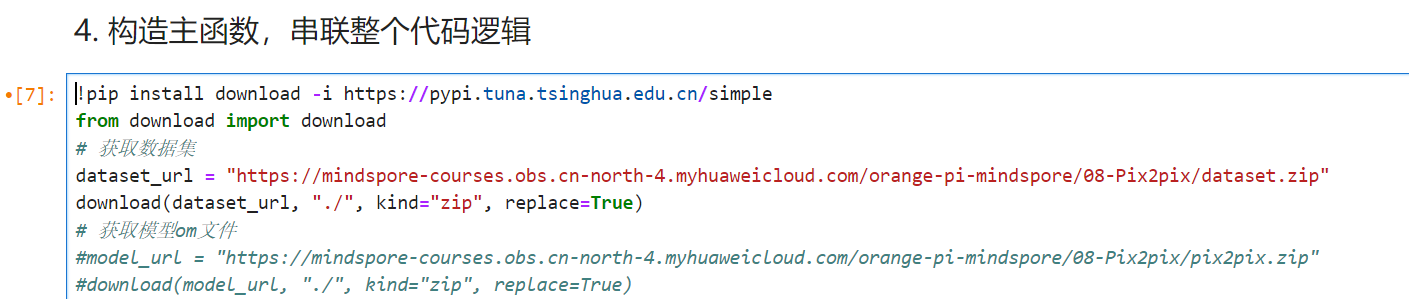


修改推理代码

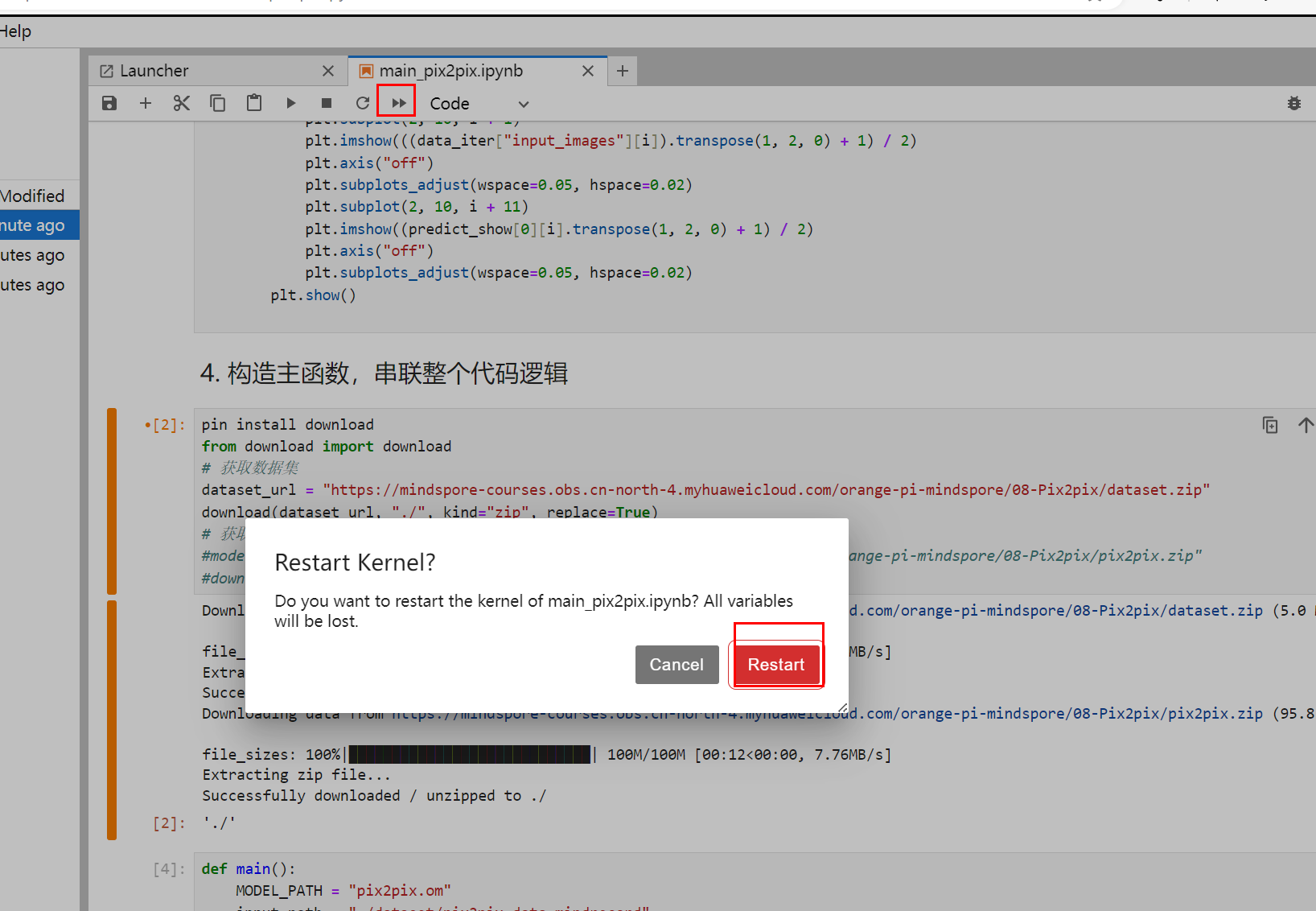
打开Pix2Pix/main\_pix2pix.ipynb文档，在下载环节，注释掉om文件下载的代码，保留数据集下载的代码。

#新增代码

!pip install download -i https://pypi.tuna.tsinghua.edu.cn/simple



执行推理



输出：

init resource stage:

Init resource success

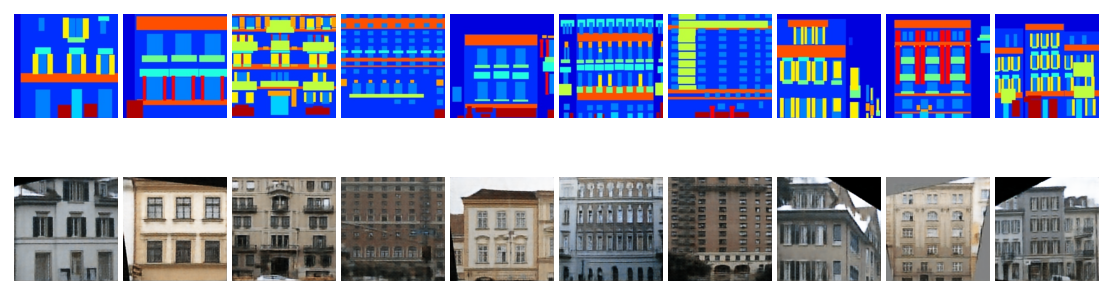
Init model resource start...

[AclLiteModel] create model output dataset:

malloc output 0, size 12582912

Create model output dataset success

Init model resource success

acl resource release all resource

AclLiteModel release source success

acl resource release stream

acl resource release context

Reset acl device 0

Release acl resource success

## 实验小结

本实验是基于Ascend910和Ascend310B算力平台，实现pix2pix训练+推理部署的全流程实验。在Ascend910算力平台使用迁移学习训练得到pix2pix模型，将其部署在香橙派 AI Pro上，使用Ascend310B算力平台实现端侧推理。通过本实验使学员掌握MindSpore的算法开发，昇腾CANN的推理应用开发，Ascend910和Ascend310B算力平台的使用。