《计算机视觉》实验单元四

目标跟踪

|  |  |
| --- | --- |
| https://www.scut.edu.cn/_upload/article/images/93/f1/da8bef494e929b2303b75fcae24a/395da264-74a6-4841-ae9f-3530ed01cf84.jpg | C:\Users\jwx341670\AppData\Local\Microsoft\Windows\INetCache\Content.Word\HW_POS_RBG_Vertical-150ppi.png |
| 华南理工大学 | 华为技术有限公司 |

版本说明：2022年春季华南理工版，

修订依据：华为智能基座《计算机视觉》课件包v2.1

使用者： 软件工程专业 2019级 数字媒体方向 第六学期《计算机视觉》课

修订人： 华南理工大学软件学院 彭绍武，张熠鹏，李泽淇，赵鹏程，黄锟城，毛玺羽

教师：彭绍武

助教：赵鹏程，李泽淇

**目录**

[1 实验介绍 1](#_Toc107782137)

[1.1 实验目的 1](#_Toc107782138)

[1.2 实验清单 1](#_Toc107782139)

[1.3 参考内容 1](#_Toc107782140)

[1.4 实验开发环境 2](#_Toc107782141)

[2 KCF目标跟踪算法 3](#_Toc107782142)

[2.1 KCF目标跟踪算法实验介绍 3](#_Toc107782143)

[2.2 实验总体设计 3](#_Toc107782144)

[2.3 实验详细设计与实现 3](#_Toc107782145)

[2.3.1 创建实验环境 3](#_Toc107782146)

[2.3.2 准备数据以及实验 5](#_Toc107782147)

[2.3.3 测试并评估KCF 8](#_Toc107782148)

[2.4 思考题 13](#_Toc107782149)

[2.5 实验总结 13](#_Toc107782150)

[3 DCFNET目标跟踪网络 14](#_Toc107782151)

[3.1 实验介绍 14](#_Toc107782152)

[3.2 实验准备 14](#_Toc107782153)

[3.3 实验详细设计与实现 15](#_Toc107782154)

[3.3.1 数据以及环境准备 15](#_Toc107782155)

[3.3.2 实验步骤 16](#_Toc107782156)

[3.4 思考题 28](#_Toc107782157)

[3.5 实验总结 28](#_Toc107782158)

[4 附录 28](#_Toc107782159)

[4.1 利用OBS进行OTB数据集的上传 29](#_Toc107782160)

# 实验介绍

目标跟踪是计算机视觉的一个热门子领域，本章主要是围绕单目目标跟踪任务，验证基于相关滤波器的方法，以及卷积神经网络方法。

相关滤波器方法实验：KCF目标跟踪算法

卷积神经网络方法实验：DCFNET目标跟踪网络

## 实验目的

本章实验的主要目的是掌握目标跟踪相关基础知识点，了解目标跟踪的相关基础知识，认识代表性的跟踪方法在该视觉任务上的表现。掌握神经网络在不同视觉任务中的架构及设计原理。

同时，本单元熟悉使用Pytorch深度学习框架实现深度学习实验的一般流程，与前几单元的Mindespore、TensorFlow框架进行对比。

## 实验清单

表格：实验、简述、难度、软件环境、硬件环境。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验 | 简述 | 难度 | 软件环境 | 开发环境 |
| KCF目标跟踪算法 | 利用KCF算法实现目标跟踪 | 初级 | Python3.7  PyTorch 1.4 | PC 64bit  ModelArts |
| DCFNET目标跟踪网络 | 利用DCFNET网络实现目标跟踪 | 中级 | Python3.7.5  PyTorch 1.4 | PC 64bit  ModelArts |

## 参考内容

《ModelArts一站式AI开发平台》

## 实验开发环境

Pytorch 1.4（云端/本地）

因在华为云ModelArts上快速搭建开发环境，可参考第三单元实验手册的文末附录：ModelArts开发环境搭建，此时是借助云端服务器、Pytorch1.4框架，以及相关的notebook编辑环境展开实验。同学们亦可参考昇腾学习资料中关于本地安装pytorch的信息，在本地完成开发环境的部署。

本次实验计算压力不大，可以选择CPU服务器进行。如果机时券充沛，亦可进行GPU实验的对比。

# KCF目标跟踪算法

## KCF目标跟踪算法实验介绍

KCF全称为Kernel Correlation Filter，核相关滤波算法。是在2014年由Joao F.Henriques等人提出，KCF算法无论是在跟踪效果上还是跟踪速度上都有十分亮眼的表现，因此当时引起了一批相近类型算法的研究与工业界的应用探索。

【实验环境要求】:

1. python3.7.5
2. ModelArts平台

## 实验总体设计

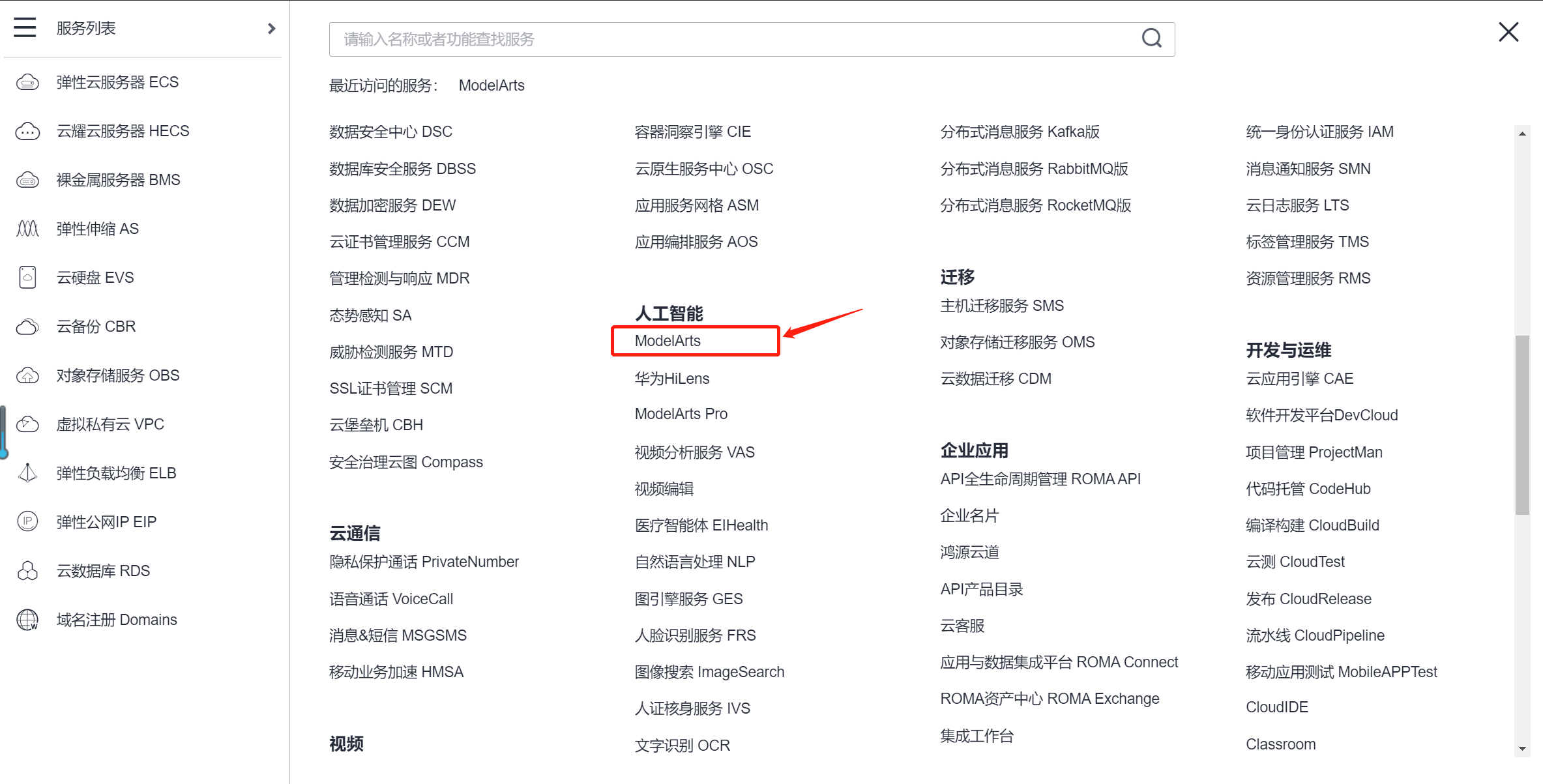
该实验引导学生在华为云ModelArts上实现KCF算法，并利用OTB数据集进行算法效果的评估。

## 实验详细设计与实现

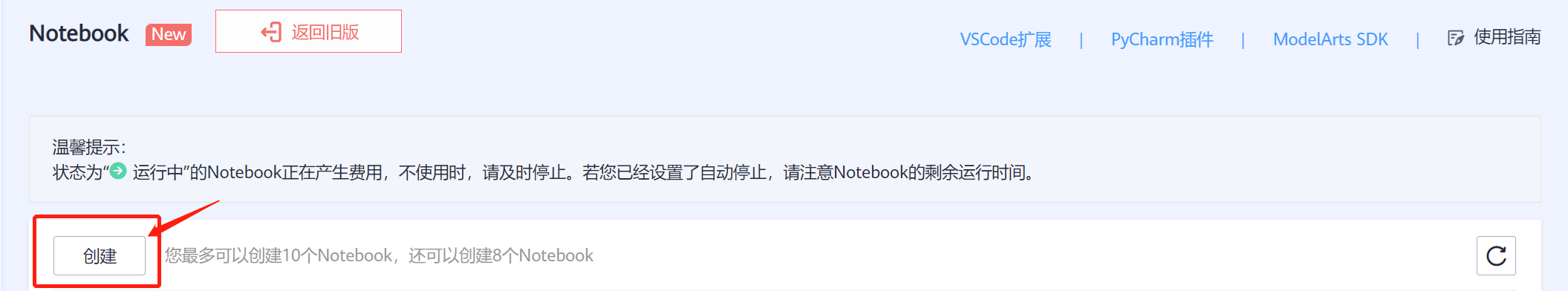
本节将详细介绍实验D的设计与实现。2.3.1节创建实验环境；2.3.2节准备数据以及实验；2.3.3节测试并评估KCF算法；

### 创建实验环境

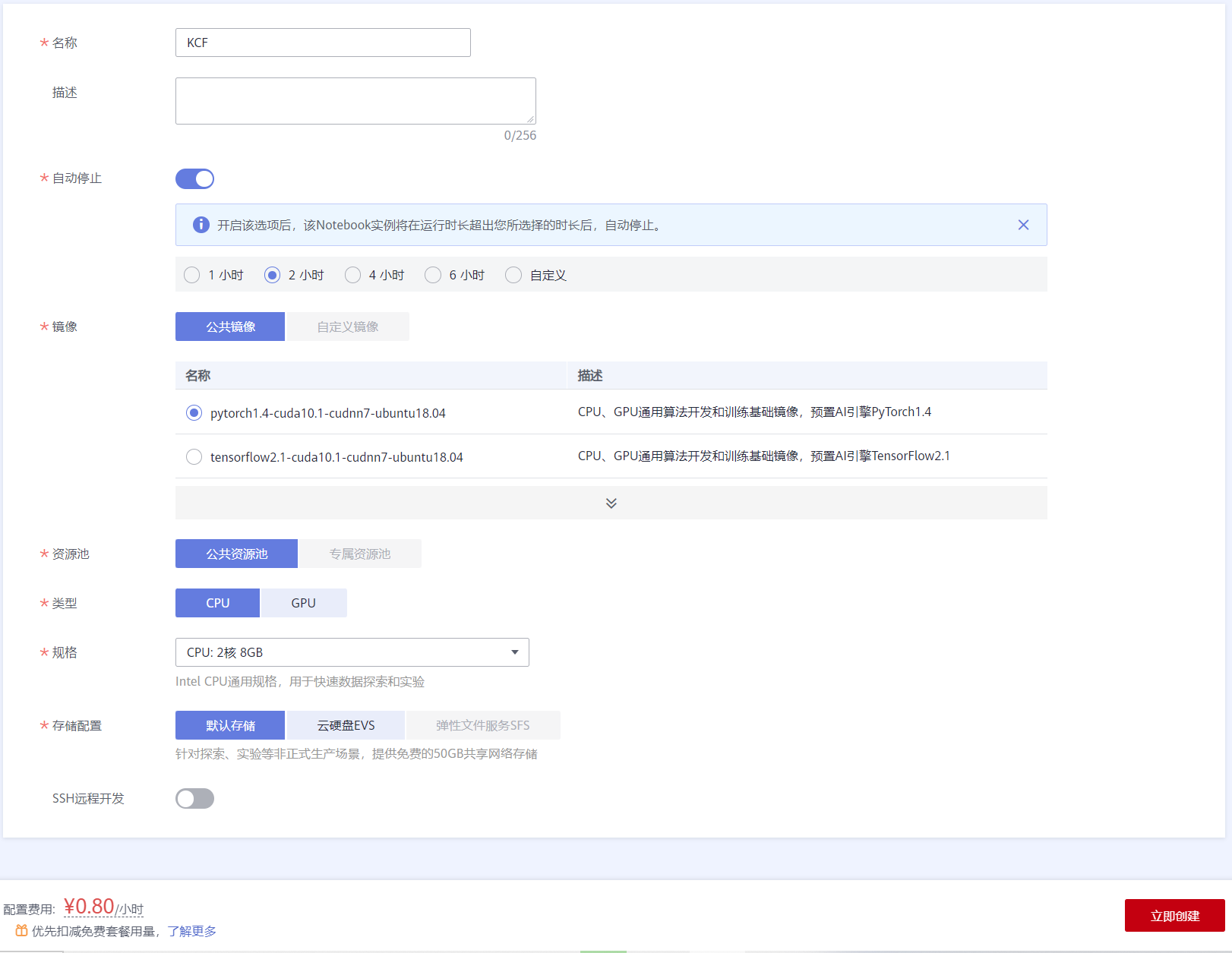
打开华为云，选择ModelArts服务



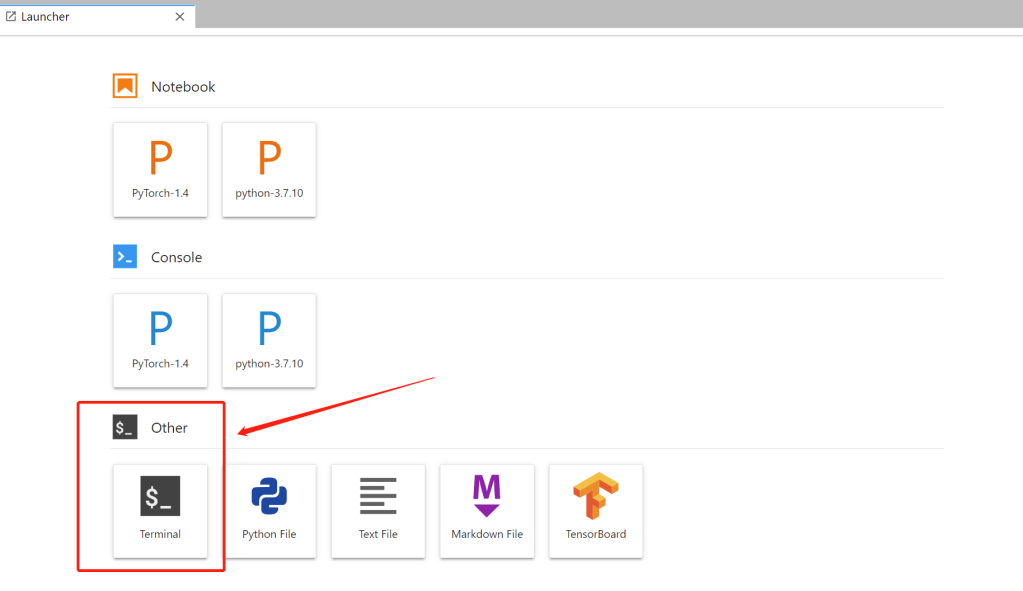
进入notebook界面，点击创建



以以下配置创建notebook环境，等待创建完成后打开notebook，完成创建实验环境



### 准备数据以及实验

点击Terminal进入终端

在终端上利用指令下载OTB数据集（如果无法下载数据集，可看手册4.1节利用OBS进行数据集的上传）

进入工作路径

cd work

创建OTB100文件夹并进入

mkdir OTB100 && cd OTB100

下载数据集，等待下载完成（大致花费5分钟左右）

wget "http://cvlab.hanyang.ac.kr/tracker\_benchmark/datasets.html"

cat datasets.html | grep '\.zip' | sed -e 's/\.zip".\*/.zip/' | sed -e s'/.\*"//' >files.txt

cat files.txt | xargs -n 1 -P 8 -I {} wget -c "http://cvlab.hanyang.ac.kr/tracker\_benchmark/{}"

数据集解压

ls \*.zip | xargs -n 1 unzip

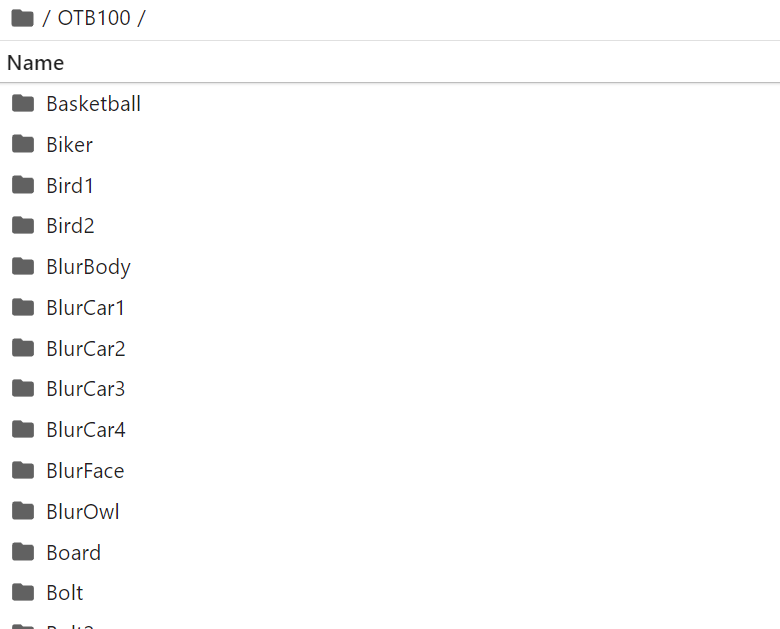
如果下载出现问题可以通过QQ群中的分享的文件，或者百度云下载：

链接：https://pan.baidu.com/s/1Pgdwt9ZqBHTPxy0oZTwKpQ

提取码：z2qm

然后参考后文，或者之前的手册，创建OBS桶，使用mox API导入数据。

数据集配置完毕后，OTB100文件夹如下图将所示



将KCF文件夹下的文件上传，上传后文件结构如下所示

##### 1643444145(1)

fhog.py用于计算hog特征

Kcftracker.py是KCF的跟踪器类

打开kcf.ipynb文件，下载需要的库

注意确保运行环境为python3.7.10



#下载opencv库

!pip install opencv-python

#下载numba库

!pip install numba

配置环境

#环境配置

import cv2

import sys

import json

import numpy as np

from os.path import join, isdir

from IPython.display import clear\_output, Image, display, HTML

import time

import ipywidgets as widgets

import kcftracker

读取数据集和配置文件

#读取dataset和配置文件

dataset = "OTB100"

base\_path = join(dataset)

json\_path = join(dataset + '.json')

annos = json.load(open(json\_path, 'r'))

videos = sorted(annos.keys())

选择想要测试的数据并初始化KCFtracker

testvideo= ['car4']

# 可以从下面任意选择放入testvideo中

# ['basketball','biker','bird1','bird2','blurBody','blurCar1','blurCar2','blurCar3','blurCar4','blurFace','blurOwl','board','bolt',

# 'bolt2','box','boy','car1','car2','car24','car4','carDark','carScale','clifBar','coke','couple','coupon','crossing','crowds','dancer','dancer2',

# 'david','david2','david3','deer','diving','dog','dog1','doll','dragonBaby','dudek','faceocc1','faceocc2','fish','fleetface','football','football1','freeman1',

# 'freeman3' 'freeman4','girl','girl2','gym','human2','human3','human4\_2','human5','human6','human7','human8','human9','ironman','jogging\_1','jogging\_2','jump',

# 'jumping','kiteSurf', 'lemming','liquor','man','matrix','mhyang','motorRolling','mountainBike','panda','redTeam','rubik','shaking','singer1','singer2',

# 'skater','skater2', 'skating1','skating2\_1','skating2\_2','skiing','soccer','subway','surfer','suv','sylvester','tiger1','tiger2','toy','trans','trellis','twinnings','vase','walking','walking2','woman']

#

#初始化kcftracker

tracker = kcftracker.KCFTracker(True, True, True)

### 测试并评估KCF

测试KCFtracker

#测试KCF Tracker

for video\_id, video in enumerate(testvideo):

video\_path\_name = annos[video]['name'] #获取视频名称

init\_rect = np.array(annos[video]['init\_rect']) #获取初始帧的bounding box

image\_files = [join(base\_path, video\_path\_name, 'img', im\_f) for im\_f in annos[video]['image\_files']] #获取视频中所有图片的路径

n\_images = len(image\_files)#获取视频长度

first\_im = cv2.imread(image\_files[0]) # 读取图片

tracker.init(init\_rect,first\_im) #利用初始图像和boundingbox初始化跟踪器

imgbox = widgets.Image(format='jpg', height=first\_im.shape[0], width=first\_im.shape[1]) #创建图片展示套件

display(imgbox)

for count in range(1 , n\_images):

gt = annos[video]['gt\_rect'][count] #获取gt

img = cv2.imread(image\_files[count])#读取图片

boundingbox = tracker.update(img)#获取计算得到的boundingbox

boundingbox = list(map(int, boundingbox))#将boundingbox中的数字转为int

#绘制预测的boundingbox 框颜色为红色

cv2.rectangle(img, (boundingbox[0],boundingbox[1]), (boundingbox[0]+boundingbox[2],boundingbox[1]+boundingbox[3]),(0,0,255), 3)

#绘制groudtruth作为对比 框颜色为绿色

cv2.rectangle(img, (gt[0],gt[1]), (gt[0]+gt[2],gt[1]+gt[3]),(0, 255, 0), 3)

#图像更新

imgbox.value = cv2.imencode('.jpg', img)[1].tobytes()

在OTB数据集选取部分视频用于评估KCF，计算iou与中心点距离误差

#计算在OTB上KCF跟踪算法的效果

evaluate\_save = dict()

#用于存储每个视频的iou和center\_error平均值

iou\_means = []

center\_error\_means = []

#选取6个视频作为评价指标

videos =['blurCar1','basketball','skater2','sylvester','trellis','freeman3']

for name in videos:

#视频名

video\_path\_name = annos[name]['name']

#图像名

imgs\_name = annos[name]['image\_files']

#groundtruth

imgs\_gt = annos[name]['gt\_rect']

#初始groudtruth

init\_rect = annos[name]['init\_rect']

#视频路径

image\_files = [join(base\_path, video\_path\_name, 'img', im\_f) for im\_f in imgs\_name]

first\_image = cv2.imread(image\_files[0])

imgbox = widgets.Image(format='jpg', height=first\_image.shape[0], width=first\_image.shape[1]) #创建图片展示套件

display(imgbox)

#初始化tracker

tracker.init(init\_rect,first\_image)

#存放每一帧的iou

video\_ious = []

#存放每一帧的中心坐标差

video\_center\_errors = []

for num , path\_name in enumerate(image\_files[1:]):

im = cv2.imread(path\_name)

boundingbox = tracker.update(im)

gt = imgs\_gt[num+1]

#OTB部分gt有误，跳过

if imgs\_gt[num+1][2]==0 or imgs\_gt[num+1][3]==0:

continue

#计算iou

iou = compute\_iou(boundingbox,gt)

#计算center\_error

center\_error = compute\_center\_error(boundingbox,gt)

# # 跟踪框绘制

res = [int(l) for l in boundingbox]

cv2.rectangle(im, (res[0], res[1]), (res[0] + res[2], res[1] + res[3]), (0, 255, 255), 3)

cv2.rectangle(im, (imgs\_gt[num+1][0], imgs\_gt[num+1][1]), (imgs\_gt[num+1][0] + imgs\_gt[num+1][2], imgs\_gt[num+1][1] + imgs\_gt[num+1][3]), (255, 0, 0), 3)

cv2.putText(im, str(iou), (40, 40), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 255), 2, cv2.LINE\_AA)

#

# # 显示跟踪结果

imgbox.value = cv2.imencode('.jpg', im)[1].tobytes()

cv2.waitKey(1)

#储存iou与center\_error

video\_ious.append(iou)

video\_center\_errors.append(center\_error)

#计算iou和中心偏差的平均值

iou\_mean = np.mean(np.array(video\_ious))

center\_error\_mean = np.mean(np.array(video\_center\_errors))

#存储

evaluate\_save[name]={

"name":video\_path\_name,

"video\_ious":video\_ious,

"iou\_mean":iou\_mean,

"video\_center\_errors":video\_center\_errors,

"center\_error\_mean":center\_error\_mean

}

iou\_means.append(iou\_mean)

center\_error\_means.append(center\_error\_mean)

print("video\_name:{}, avg\_iou:{}%, avg\_center\_error:{}".format(video\_path\_name,(iou\_mean\*100),center\_error\_mean))

#计算在所有视频中iou和中心偏差的平均值

total\_iou\_mean = np.mean(np.array(iou\_means))

total\_center\_error\_mean = np.mean(np.array(center\_error\_means))

print("total\_avg\_iou:{}%, total\_avg\_center\_error:{}".format((total\_iou\_mean\*100),total\_center\_error\_mean))

#储存计算结果

jsonsave("OTB\_evaluate\_save.json",evaluate\_save)

最终输出KCF算法在这六个视频的平均iou和中心误差值

1645513745(1)

绘制算法iou与中心点距离误差图

#绘制iou和中心点误差图

import matplotlib.pyplot as plt

from scipy.interpolate import make\_interp\_spline

annos = json.load(open("OTB\_evaluate\_save.json", 'r'))

#################绘制video\_ious#########################

al = []

al = np.array(al)

for i in annos:

al = np.append(al , annos[i]["video\_ious"])

al = np.ndarray.flatten(al)

X = np.linspace(0, 1, 21, endpoint=True)

Y = np.zeros([21])

plt.axis([0,1,0,1])

for i in range(21):

Y[i] = np.size(np.where(al>=X[i]))/np.size(al)

x\_smooth = np.linspace(X.min(),X.max(),300) #300 represents number of points to make between T.min and T.max

# power\_smooth = make\_interp\_spline(x\_suc,L,xnew)

y\_smooth = make\_interp\_spline(X,Y)(x\_smooth)

# 加网格线

plt.grid()

# 标题 横纵坐标名

plt.title(u"Success plots of OPE")

plt.xlabel(u"Overlap threshold")

plt.ylabel(u"Success rate")

plt.plot(x\_smooth,y\_smooth)

plt.show()

##############################绘制video\_center\_errors##########################

al = []

al = np.array(al)

for i in annos:

al = np.append(al , annos[i]["video\_center\_errors"])

al = np.ndarray.flatten(al)

X = np.linspace(0, 50, 51, endpoint=True)

Y = np.zeros([51])

plt.axis([0,50,0,1])

for i in range(51):

Y[i] = np.size(np.where(al<=X[i]))/np.size(al)

x\_smooth = np.linspace(X.min(),X.max(),300) #300 represents number of points to make between T.min and T.max

# power\_smooth = make\_interp\_spline(x\_suc,L,xnew)

y\_smooth = make\_interp\_spline(X,Y)(x\_smooth)

# 加网格线

plt.grid()

# 标题 横纵坐标名

plt.title(u"Precision plots of OPE")

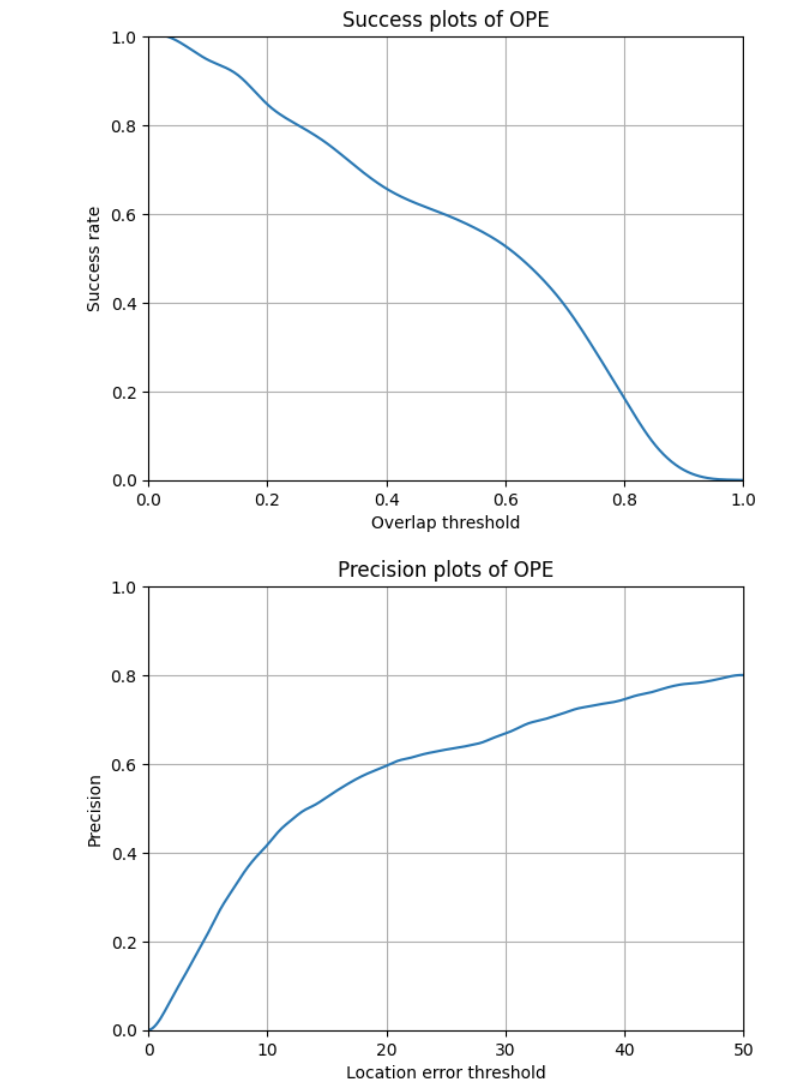
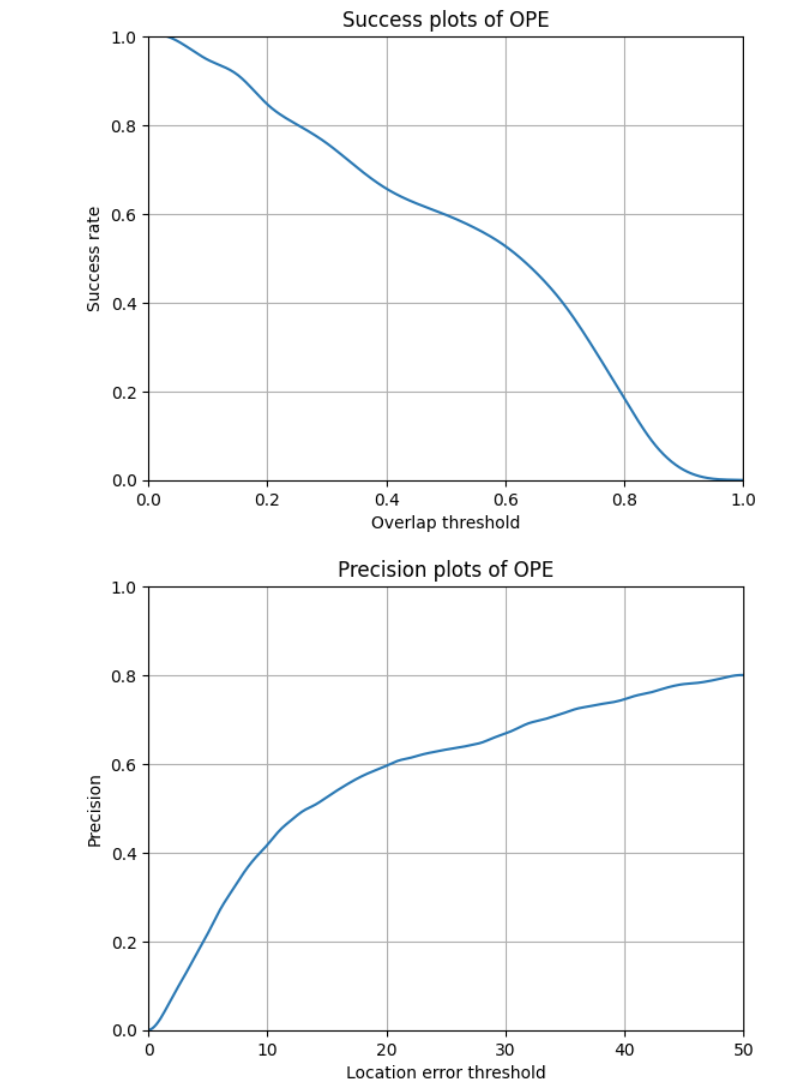
plt.xlabel(u"Location error threshold")

plt.ylabel(u"Precision")

plt.plot(x\_smooth,y\_smooth)

plt.show()

结果图例



## 思考题

学习[http://cvlab.hanyang.ac.kr/tracker\_benchmark/datasets.html 上关于OTB100](http://cvlab.hanyang.ac.kr/tracker_benchmark/datasets.html%20上关于OTB100)数据集中不同评价指标的定义，如IV, SV, OCC, DEF, MB… ，尝试根据标签选择具有对应指标特性的序列进行对比测试、验证并分析说明这些评价指标对应不同视频的跟踪情况。实验方案要针对特定评价指标进行设计，选择相应的典型与非典型测试序列，绘制曲线，并给出具体跟踪目标的典型成功与失败案例图示与说明。

## 实验总结

本章提供了一个基于华为ModelArts平台的目标跟踪实验。该实验演示了如何利用华为云ModelArts完成目标跟踪任务。本章阐明了整个实验的运行流程，详细解释了如何下载配置数据，处理数据以及如何评估算法。实验结果表明KCF算法具有较快的推断速度和较好的跟踪性能。

# DCFNET目标跟踪网络

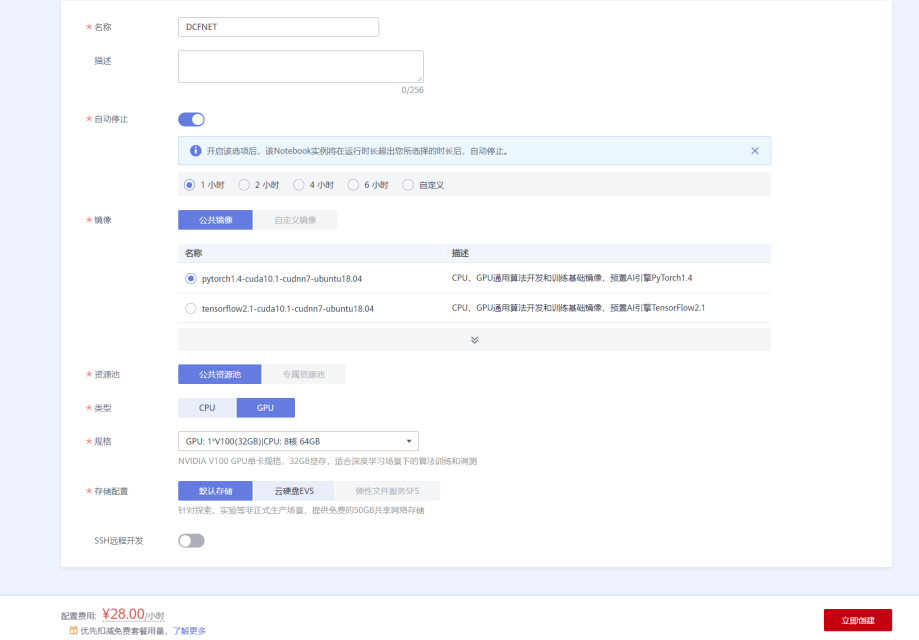
## 实验介绍

本例子会实现DCFNET网络的搭建、训练以及评估，整体流程如下：

1. 处理需要的数据集，这里使用了OTB数据集。
2. 定义一个网络，这里我们使用DCFNET网络。
3. 定义损失函数和优化器。
4. 加载数据集并进行训练，训练完成后，将模型文件保存
5. 加载保存的模型，进行推理测试。
6. 验证模型，加载测试数据集和训练后的模型，验证网络效果。

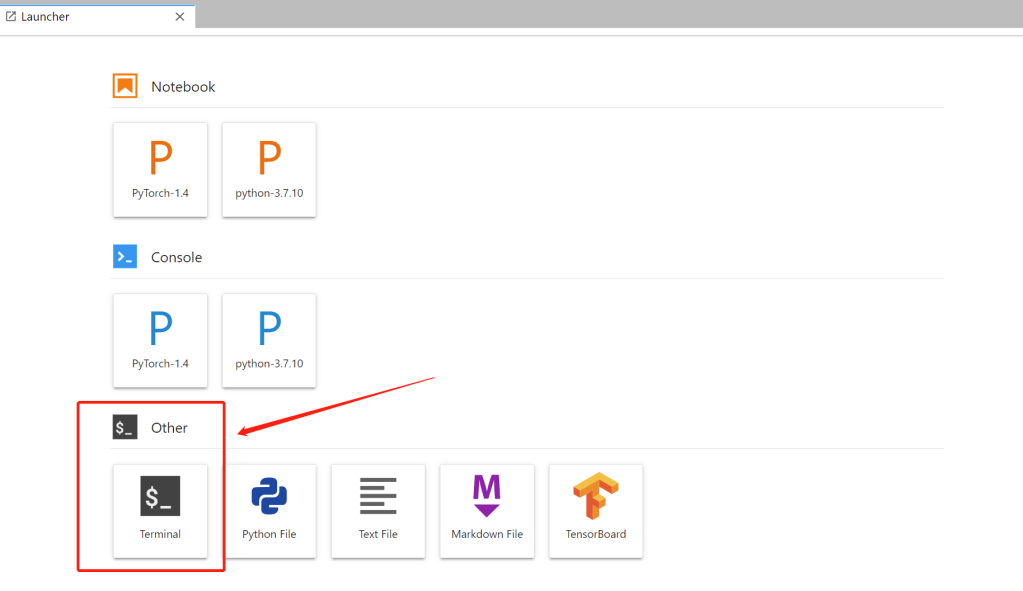
## 实验准备

打开华为云的modelarts平台，新建notebook，配置如下图



## 实验详细设计与实现

### 数据以及环境准备

点击Terminal进入终端

在终端上利用指令下载OTB数据集

进入工作路径

cd work

创建OTB100文件夹并进入

mkdir OTB100 && cd OTB100

下载数据集，等待下载完成（大致花费5分钟左右）

wget "http://cvlab.hanyang.ac.kr/tracker\_benchmark/datasets.html"

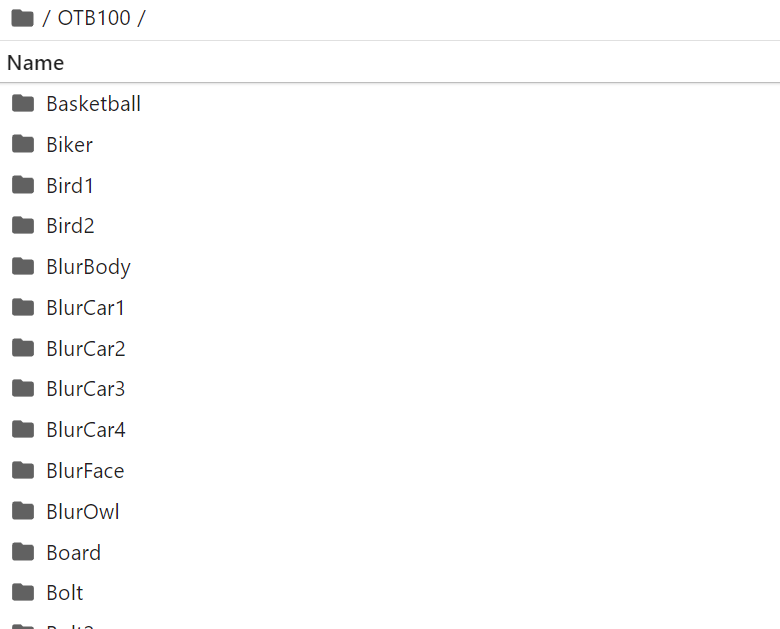
cat datasets.html | grep '\.zip' | sed -e 's/\.zip".\*/.zip/' | sed -e s'/.\*"//' >files.txt

cat files.txt | xargs -n 1 -P 8 -I {} wget -c "http://cvlab.hanyang.ac.kr/tracker\_benchmark/{}"

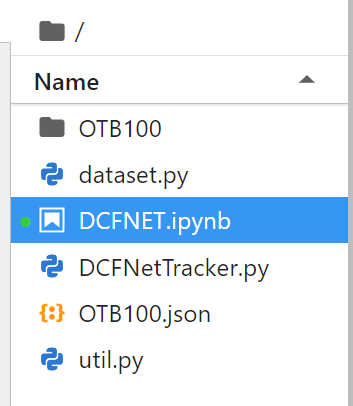
数据集解压

ls \*.zip | xargs -n 1 unzip

数据集配置完毕后，OTB100文件夹如下图将所示



将DCFnet文件夹下的文件上传，上传后文件结构如下所示



Dataset.py为OTB数据集处理类，用于训练时从数据集抽取整理好的数据

DCFNetTracker.py为DCFNET的跟踪器类

Util.py中为一些通用的方法和类

### 实验步骤

打开DCFNET.ipynb, 运行代码，导入Python库&模块并配置运行信息

#环境导入

import shutil

from os.path import join, isdir, isfile

from os import makedirs

import os

import json

import cv2

import glob

import ipywidgets as widgets

import torch

import torch.nn as nn

from torch.utils.data import dataloader

from torchvision import transforms

import numpy as np

import time

import numpy.random as npr

from torch.utils.data import Dataset, DataLoader

from dataset import OTB

from util import complex\_mulconj,complex\_mul

from util import gaussian\_shaped\_labels, AverageMeter, rect1\_2\_cxy\_wh, cxy\_wh\_2\_rect

from DCFNetTracker import DCFNetTracker

#创建模型检查点存储地址

#检查点都会存储在save文件夹下

save\_path = join("save")

if not isdir(save\_path):

makedirs(save\_path)

定义训练配置类

#########config类##########

class TrainConfig(object):

crop\_sz = 125

output\_sz = 125

lambda0 = 1e-4

padding = 2.0

output\_sigma\_factor = 0.1

output\_sigma = crop\_sz / (1 + padding) \* output\_sigma\_factor

y = gaussian\_shaped\_labels(output\_sigma, [output\_sz, output\_sz])

yf = torch.rfft(torch.Tensor(y).view(1, 1, output\_sz, output\_sz).cuda(), signal\_ndim=2)

input\_sz = 125

padding = 2.0

range = 20 # 抽取的两帧之间的最大距离

start\_epoch = 0 # 起始epoch

epochs = 5 # 训练进行的总epoch数

workers = 8 # DataLoader参数

batch\_size = 64 # DataLoader参数 每组包含的数据个数

lr = 0.01 # 优化器参数

momentum = 0.9 # 优化器参数

weight\_decay=5e-5 # 优化器参数

由于网络训练中往往需要许多参数，因此将其规整为一个config类是一个有效率的手段

定义DCFNet网络

我们所选择的为DCFNet网络。DCFNet网络利用两层的卷积生成特征来替代传统的HOG特征，运行逻辑依旧是类似于KCF的在线跟踪形式。其中建立dotrain函数方便进行训练，DCFNet的网络定义如下：

######################定义DCFNet网络################

class DCFNet(nn.Module):

def \_\_init\_\_(self, config=None):

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

#DCFNet feature

self.feature = nn.Sequential(

nn.Conv2d(3, 32, 3, padding=1), # in\_channel, out\_channel ,kennel\_size， padding

nn.ReLU(inplace=True),

nn.Conv2d(32, 32, 3, padding=1),

nn.LocalResponseNorm(size=5, alpha=0.0001, beta=0.75, k=1),

)

self.config = config

def forward(self, x):

x = self.feature(x) \* self.config.cos\_window

xf = torch.rfft(x, signal\_ndim=2)

kxzf = torch.sum(complex\_mulconj(xf, self.model\_zf), dim=1, keepdim=True)

response = torch.irfft(complex\_mul(kxzf, self.model\_alphaf), signal\_ndim=2)

return response

def update(self, z, lr=1.):

#z的shape为[1,3,125,125]

#[1,3,125,125] ->[1,32,125,125]

z = self.feature(z) \* self.config.cos\_window

#[1,3,125,125] ->[1,32,125,63,2] 傅里叶变换

zf = torch.rfft(z, signal\_ndim=2)

#kzzf = [1, 1, 125, 63, 1] 讲实部虚部和通道数都sum起来

kzzf = torch.sum(torch.sum(zf \*\* 2, dim=4, keepdim=True), dim=1, keepdim=True)

# alphaf = [1, 1, 125, 63, 2]

alphaf = self.config.yf / (kzzf + self.config.lambda0)

if lr > 0.99:

self.model\_alphaf = alphaf

self.model\_zf = zf

else:

self.model\_alphaf = (1 - lr) \* self.model\_alphaf.data + lr \* alphaf.data

self.model\_zf = (1 - lr) \* self.model\_zf.data + lr \* zf.data

# 用于训练

def dotrain(self, z, x):

z = self.feature(z)

x = self.feature(x)

zf = torch.rfft(z, signal\_ndim=2)

xf = torch.rfft(x, signal\_ndim=2)

kzzf = torch.sum(torch.sum(zf \*\* 2, dim=4, keepdim=True), dim=1, keepdim=True)

kxzf = torch.sum(complex\_mulconj(xf, zf), dim=1, keepdim=True)

alphaf = self.config.yf.to(device=z.device) / (kzzf + self.config.lambda0) # very Ugly

response = torch.irfft(complex\_mul(kxzf, alphaf), signal\_ndim=2)

return response

#读取参数

def load\_param(self, path='param.pth'):

checkpoint = torch.load(path)

if 'state\_dict' in checkpoint.keys(): # from training result

state\_dict = checkpoint['state\_dict']

if 'module' in list(state\_dict.keys())[0]: # train with nn.DataParallel

from collections import OrderedDict

new\_state\_dict = OrderedDict()

for k, v in state\_dict.items():

name = k[7:] # remove `module.`

new\_state\_dict[name] = v

self.load\_state\_dict(new\_state\_dict)

else:

self.load\_state\_dict(state\_dict)

else:

self.feature.load\_state\_dict(checkpoint)

定义训练以及验证相关的方法

############定义训练以及验证相关的方法###################

#调整学习率

def adjust\_learning\_rate(optimizer, epoch):

lr = np.logspace(-2, -5, num=config.epochs)[epoch]

for param\_group in optimizer.param\_groups:

param\_group['lr'] = lr

#存储检查点

def save\_checkpoint(state, is\_best, epoch, filename=save\_path):

filename = join(filename, 'checkpoint\_ep\_'+str(epoch)+'.pth')

torch.save(state, filename)

if is\_best:

shutil.copyfile(filename, join(save\_path, 'model\_best.pth'))

#训练函数

def train(train\_loader, model, criterion, optimizer, epoch, target):

#用于计算平均值

batch\_time = AverageMeter()

losses = AverageMeter()

# 切换到训练模式

model.train()

end = time.time()

for i, (template, search) in enumerate(train\_loader):

template = template.cuda(non\_blocking=True)

search = search.cuda(non\_blocking=True)

# compute output

output = model.dotrain(template, search)

# 计算loss

loss = criterion(output, target)/template.size(0)

losses.update(loss.item())

# 进行梯度下降

optimizer.zero\_grad()

loss.backward()

optimizer.step()

# 计算运行时间

batch\_time.update(time.time() - end)

end = time.time()

if i % 30 == 0:

print('Train: [{0}][{1}/{2}]\t'

'Time {batch\_time.val:.3f} \t'

'Loss {loss.val:.4f} (loss.avg:{loss.avg:.4f})\t'.format(

epoch, i, len(train\_loader), batch\_time=batch\_time, loss=losses))

#验证函数

def validate(val\_loader, model, criterion, target):

batch\_time = AverageMeter()

losses = AverageMeter()

model.eval()

with torch.no\_grad():

end = time.time()

for i, (template, search) in enumerate(val\_loader):

# 获取template和search

template = template.cuda(non\_blocking=True)

search = search.cuda(non\_blocking=True)

# 得到输出

output = model.dotrain(template, search)

loss = criterion(output, target)/(config.batch\_size \* gpu\_num)

# 计算损失

losses.update(loss.item())

# 计算运行时间

batch\_time.update(time.time() - end)

end = time.time()

if i % 30 == 0:

print('Test: [{0}/{1}]\t'

'Time {batch\_time.val:.3f} \t'

'Loss {loss.val:.4f} (loss.avg: {loss.avg:.4f})\t'.format(

i, len(val\_loader), batch\_time=batch\_time, loss=losses))

print(' \* Loss {loss.val:.4f} ({loss.avg:.4f})'.format(loss=losses))

return losses.avg

#计算两个boundingbox的中心点偏移，bbox均为ltwh模式

def compute\_center\_error(bbox1,bbox2):

cen1 = np.array([bbox1[0]+bbox1[2]-1/2,bbox1[1]+bbox1[3]-1/2])

cen2 = np.array([bbox2[0]+bbox2[2]-1/2,bbox2[1]+bbox2[3]-1/2])

errors = np.sqrt(np.sum(np.power(cen1 - cen2, 2)))

return errors

#计算两个boundingbox的iou bbox均为ltwh模式

def compute\_iou(bbox1,bbox2):

l1, t1, r1, d1 = bbox1[0], bbox1[1], bbox1[0]+bbox1[2], bbox1[1]+bbox1[3]

l2, t2, r2, d2 = bbox2[0], bbox2[1], bbox2[0]+bbox2[2], bbox2[1]+bbox2[3]

l = max(l1,l2)

t = max(t1,t2)

r = min(r1,r2)

d = min(d1,d2)

innerarea = (r - l)\*(d - t)

if (r - l) <= 0 or (d - t)<=0:

return 0.0

outarea1 = bbox1[2] \* bbox1[3]

outarea2 = bbox2[2] \* bbox2[3]

return innerarea/(outarea1+outarea2-innerarea)

#储存dict函数

def jsonsave(filename,file):

with open(filename, 'w') as f\_obj:

json.dump(file, f\_obj)

参数设置

##### 定义一些参数以及实例化DCFnet网络和config类

##################参数设置#####################

#生成config

config = TrainConfig()

#设置best\_loss

best\_loss = 1e6

#建立模型

model = DCFNet(config=config)

model.cuda()

#计算gpu数量

gpu\_num = torch.cuda.device\_count()

if gpu\_num > 1:

model = torch.nn.DataParallel(model, list(range(gpu\_num))).cuda()

定义损失函数及优化器

在进行定义之前，先简单介绍损失函数及优化器的概念。

损失函数：又叫目标函数，用于衡量预测值与实际值差异的程度。深度学习通过不停地迭代来缩小损失函数的值。定义一个好的损失函数，可以有效提高模型的性能。

优化器：用于最小化损失函数，从而在训练过程中改进模型。

定义了损失函数后，可以得到损失函数关于权重的梯度。梯度用于指示优化器优化权重的方向，以提高模型性能。

。

##################定义损失函数及优化器###########

#loss函数

criterion = nn.MSELoss(reduction = 'sum').cuda()

#优化器

optimizer = torch.optim.SGD(model.parameters(), config.lr,momentum=config.momentum,weight\_decay=config.weight\_decay)

开始训练

#######开始训练##########

if \_\_name\_\_ == "\_\_main\_\_":

#创建训练和验证数据集

train\_dataset = OTB(root\_path="", train=True, range=config.range)

val\_dataset = OTB(root\_path="", train=False, range=config.range)

train\_loader = torch.utils.data.DataLoader(

train\_dataset, batch\_size=config.batch\_size\*gpu\_num,num\_workers=config.workers,pin\_memory=True)

val\_loader = torch.utils.data.DataLoader(

val\_dataset, batch\_size=config.batch\_size\*gpu\_num, num\_workers=config.workers,pin\_memory=True)

target = torch.Tensor(config.y).cuda().unsqueeze(0).unsqueeze(0).repeat(config.batch\_size \* gpu\_num, 1, 1,

1) # for training

#开始训练

for epoch in range(config.start\_epoch, config.epochs):

#调整学习率

adjust\_learning\_rate(optimizer, epoch)

#训练

train(train\_loader, model, criterion, optimizer, epoch, target)

#验证

loss = validate(val\_loader, model, criterion, target)

# 储存当前的检查点以及记录最佳检查点

is\_best = loss < best\_loss

best\_loss = min(best\_loss, loss)

save\_checkpoint({

'epoch': epoch + 1,

'state\_dict': model.state\_dict(),

'best\_loss': best\_loss,

'optimizer': optimizer.state\_dict(),

}, is\_best,epoch)

训练过程中会打印loss值，类似下图。loss值会波动，但总体来说loss值会逐步减小，精度逐步提高。每个人运行的loss值有一定随机性，不一定完全相同。

在OTB数据集选取部分视频用于评估DCFnet，计算iou与中心点距离误差

###########定义TrackerConfig类####################

class TrackerConfig(object):

crop\_sz = 125

lambda0 = 1e-4

padding = 2

output\_sigma\_factor = 0.1

interp\_factor = 0.01

num\_scale = 3

scale\_step = 1.0275

scale\_factor = scale\_step \*\* (np.arange(num\_scale) - num\_scale / 2)

min\_scale\_factor = 0.2

max\_scale\_factor = 5

scale\_penalty = 0.9925

scale\_penalties = scale\_penalty \*\* (np.abs((np.arange(num\_scale) - num\_scale / 2)))

net\_input\_size = [crop\_sz, crop\_sz]

net\_average\_image = np.array([104, 117, 123]).reshape(-1, 1, 1).astype(np.float32)

output\_sigma = crop\_sz / (1 + padding) \* output\_sigma\_factor

y = gaussian\_shaped\_labels(output\_sigma, net\_input\_size)

yf = torch.rfft(torch.Tensor(y).view(1, 1, crop\_sz, crop\_sz).cuda(), signal\_ndim=2)

cos\_window = torch.Tensor(np.outer(np.hanning(crop\_sz), np.hanning(crop\_sz))).cuda()

##########评估模型#############

#创建用于运行的DCFnet

trackerconfig = TrackerConfig()

net = DCFNet(config=trackerconfig)

#创建traker，导入检查点

traker = DCFNetTracker("save/model\_best.pth",net,trackerconfig)

evaluate\_save = dict()

# 用于存储每个视频的iou和center\_error平均值

iou\_means = []

center\_error\_means = []

dataset = "OTB100"

root\_path = os.path.join('')

json\_path = os.path.join(root\_path, dataset + '.json')

annos = json.load(open(json\_path, 'r'))

videos = sorted(annos.keys())

#选取六个视频用于评估

video =['blurCar1','basketball','skater2','sylvester','trellis','freeman3']

for name in video:

# 视频名

video\_path\_name = annos[name]['name']

# 图像名

imgs\_name = annos[name]['image\_files']

# groundtruth

imgs\_gt = annos[name]['gt\_rect']

# 初始groudtruth

init\_rect = annos[name]['init\_rect']

image\_files =[os.path.join(root\_path, "OTB100", video\_path\_name, "img", i) for i in imgs\_name]

first\_image = cv2.imread(image\_files[0])

imgbox = widgets.Image(format='jpg', height=first\_image.shape[0], width=first\_image.shape[1]) #创建图片展示套件

display(imgbox)

target\_pos, target\_sz = np.array([init\_rect[0]+(init\_rect[2]-1)//2,init\_rect[1]+(init\_rect[3]-1)//2]), np.array([init\_rect[2],init\_rect[3]])

# 初始化跟踪器

traker.init(first\_image, target\_pos, target\_sz)

# 存放每一帧的iou

video\_ious = []

# 存放每一帧的中心坐标差

video\_center\_errors = []

for num, path\_name in enumerate(image\_files[1:]):

im = cv2.imread(path\_name)

target\_pos, target\_sz = traker.track(im)

boundingbox = cxy\_wh\_2\_rect(target\_pos, target\_sz)

gt = imgs\_gt[num + 1]

# OTB部分gt有误，跳过

if imgs\_gt[num + 1][2] == 0 or imgs\_gt[num + 1][3] == 0:

continue

# 计算iou

iou = compute\_iou(boundingbox, gt)

# 计算center\_error

center\_error = compute\_center\_error(boundingbox, gt)

res = [int(l) for l in boundingbox]

# 跟踪框绘制

cv2.rectangle(im, (res[0], res[1]), (res[0] + res[2], res[1] + res[3]), (0, 255, 255), 3)

cv2.rectangle(im, (imgs\_gt[num+1][0], imgs\_gt[num+1][1]), (imgs\_gt[num+1][0] + imgs\_gt[num+1][2], imgs\_gt[num+1][1] + imgs\_gt[num+1][3]), (255, 0, 0), 3)

cv2.putText(im, str(iou), (40, 40), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 255), 2, cv2.LINE\_AA)

# 显示跟踪结果

imgbox.value = cv2.imencode('.jpg', im)[1].tobytes()

cv2.waitKey(1)

video\_ious.append(iou)

video\_center\_errors.append(center\_error)

# 计算avg

iou\_mean = np.mean(np.array(video\_ious))

center\_error\_mean = np.mean(np.array(video\_center\_errors))

# 存储

evaluate\_save[name] = {

"name": video\_path\_name,

"video\_ious": video\_ious,

"iou\_mean": iou\_mean,

"video\_center\_errors": video\_center\_errors,

"center\_error\_mean": center\_error\_mean

}

iou\_means.append(iou\_mean)

center\_error\_means.append(center\_error\_mean)

print(

"video\_name:{}, avg\_iou:{}%, avg\_center\_error:{}".format(video\_path\_name, (iou\_mean \* 100), center\_error\_mean))

total\_iou\_mean = np.mean(np.array(iou\_means))

total\_center\_error\_mean = np.mean(np.array(center\_error\_means))

print("total\_avg\_iou:{}%, total\_avg\_center\_error:{}".format((total\_iou\_mean \* 100), total\_center\_error\_mean))

#将结果存储起来

jsonsave("OTB\_evaluate\_save.json", evaluate\_save)

最终会输出网络在这六个视频的平均iou和中心误差值

1645514466(1)

绘制算法iou与中心点距离误差图

#绘制iou和中心点误差图

import matplotlib.pyplot as plt

from scipy.interpolate import make\_interp\_spline

annos = json.load(open("OTB\_evaluate\_save.json", 'r'))

#################绘制video\_ious#########################

al = []

al = np.array(al)

for i in annos:

al = np.append(al , annos[i]["video\_ious"])

al = np.ndarray.flatten(al)

X = np.linspace(0, 1, 21, endpoint=True)

Y = np.zeros([21])

plt.axis([0,1,0,1])

for i in range(21):

Y[i] = np.size(np.where(al>=X[i]))/np.size(al)

x\_smooth = np.linspace(X.min(),X.max(),300) #300 represents number of points to make between T.min and T.max

# power\_smooth = make\_interp\_spline(x\_suc,L,xnew)

y\_smooth = make\_interp\_spline(X,Y)(x\_smooth)

# 加网格线

plt.grid()

# 标题 横纵坐标名

plt.title(u"Success plots of OPE")

plt.xlabel(u"Overlap threshold")

plt.ylabel(u"Success rate")

plt.plot(x\_smooth,y\_smooth)

plt.show()

##############################绘制video\_center\_errors##########################

al = []

al = np.array(al)

for i in annos:

al = np.append(al , annos[i]["video\_center\_errors"])

al = np.ndarray.flatten(al)

X = np.linspace(0, 50, 51, endpoint=True)

Y = np.zeros([51])

plt.axis([0,50,0,1])

for i in range(51):

Y[i] = np.size(np.where(al<=X[i]))/np.size(al)

x\_smooth = np.linspace(X.min(),X.max(),300) #300 represents number of points to make between T.min and T.max

# power\_smooth = make\_interp\_spline(x\_suc,L,xnew)

y\_smooth = make\_interp\_spline(X,Y)(x\_smooth)

# 加网格线

plt.grid()

# 标题 横纵坐标名

plt.title(u"Precision plots of OPE")

plt.xlabel(u"Location error threshold")

plt.ylabel(u"Precision")

plt.plot(x\_smooth,y\_smooth)

plt.show()

## 思考题

1. 模型训练时为何需要每个循环减少学习率呢？
2. 请改进代码，让六个视频的平均iou高于70%或中心点距离误差低于15 （进阶）

（epochs改为10基本可以达到要求，验证后，亦可尝试其它的网络改进方法）

## 实验总结

本章提供了一个基于华为ModelArts平台的目标跟踪机器学习方法的实验。该实验演示了如何利用华为云ModelArts完成目标跟踪网络的训练。本章阐明了整个实验的运行流程，详细解释了如何下载配置数据，处理数据、如何定义、训练网络以及如何评估网络效果。实验结果表明DCFnet算法在提高了跟踪性能的后依旧保持着较快的推断速度。

# 附录

## 利用OBS进行OTB数据集的上传

下载提供的OTB100.zip

1645510774(1)

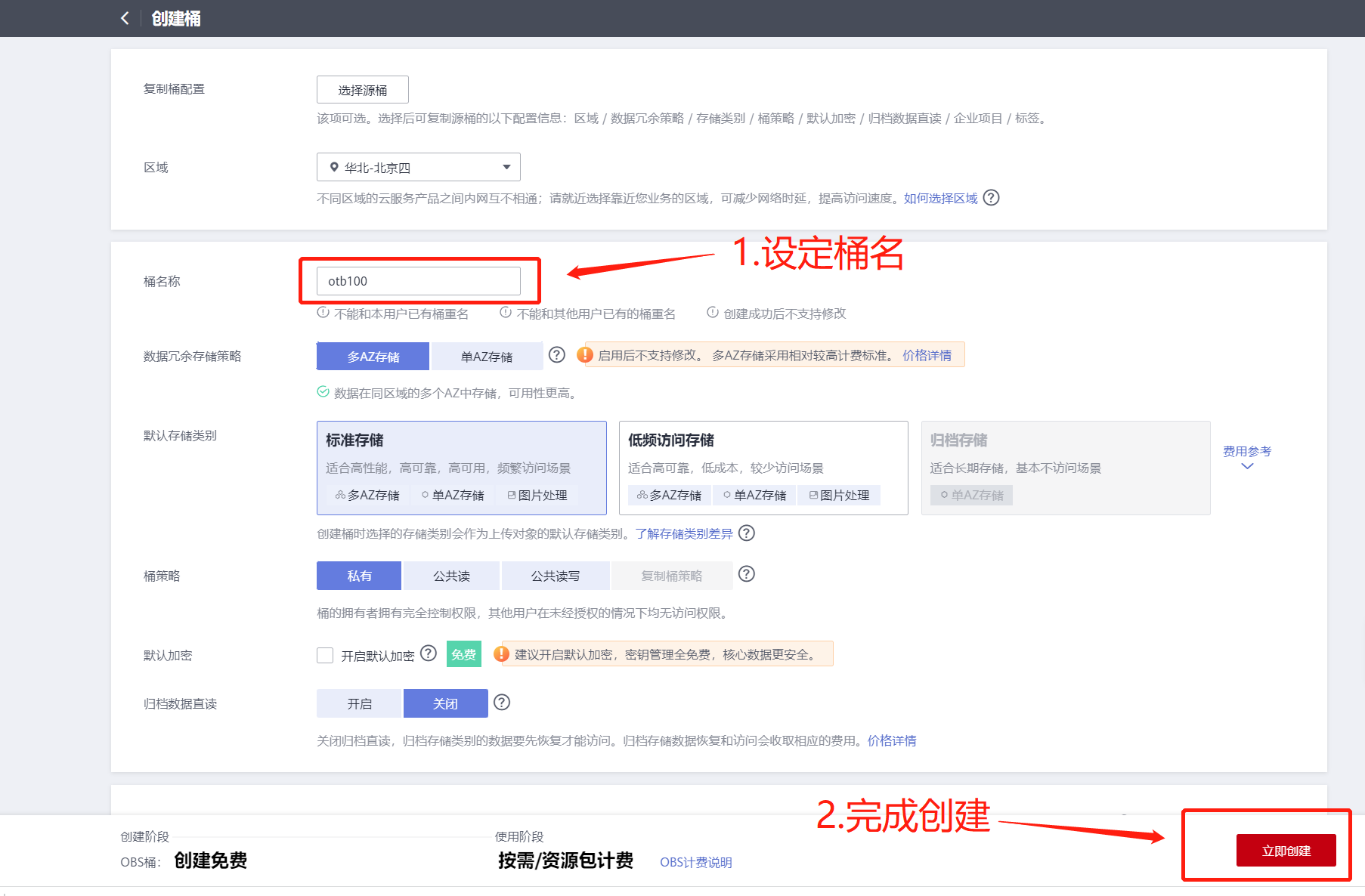
打开对象存储服务OBS

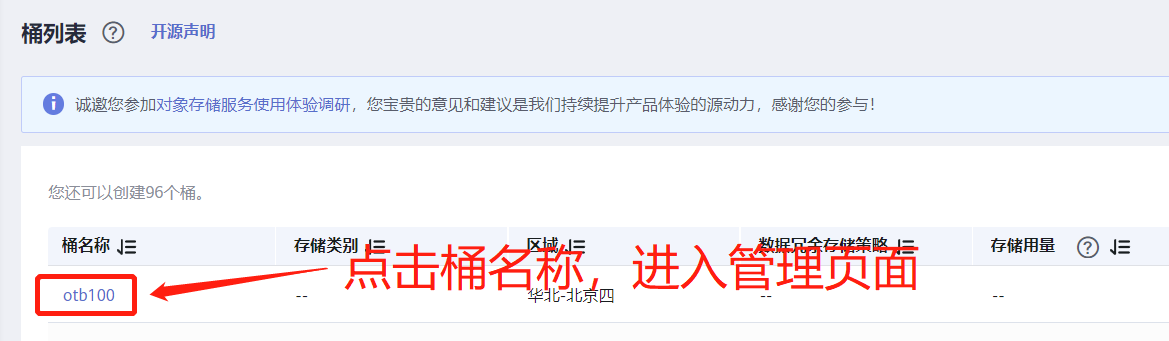


点击右上角创建桶

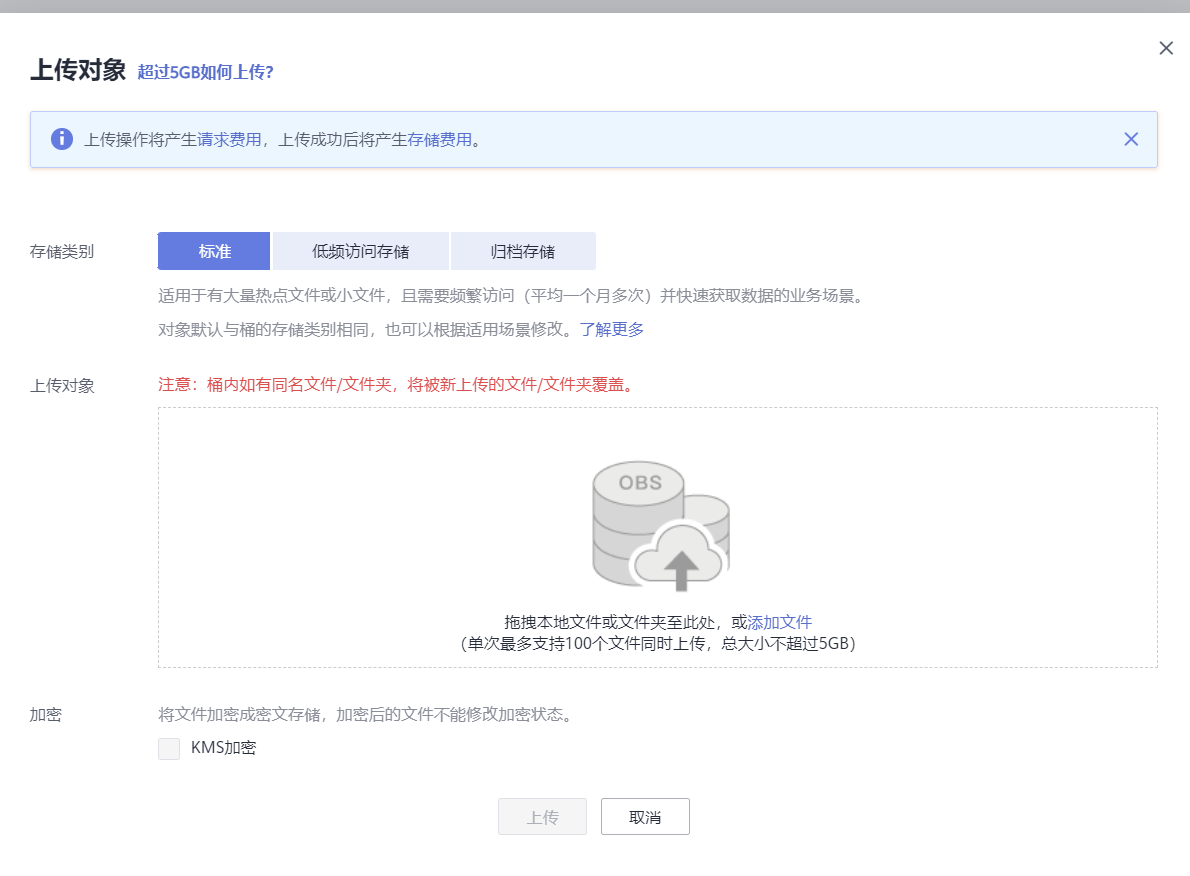


设定桶名并完成创建



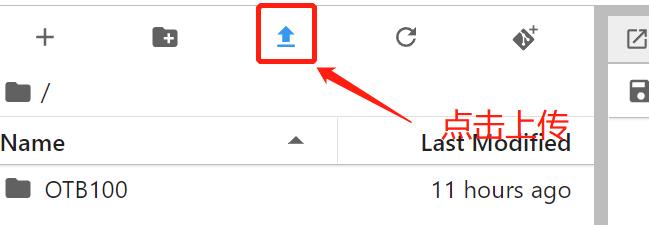
桶创建完成后，进入桶界面

将之前准备好的OTB100.zip文件上传



上传完成后如下所示



打开notebook，点击左上角的上传图标



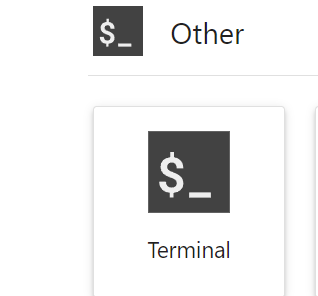


解压OTB100.zip

##### 点击左上角加号图标

##### 1645511916(1)

选择Terminal



输入指令，完成最终数据的解压：

进入工作路径

cd work

解压OTB100

unzip OTB100.zip

操作如下图：

