**处理测试集的方式**

假设有m个整图需要裁剪为不同尺寸的patch，可以按照以下两种方式处理数据

1. 将整图预先裁剪为patch，再存为hd5文件，在模型中读取若干个hd5文件加载patch数据。

2. 直接在模型中加载若干大图，在内存中裁剪大图为patch，然后在模型中读取patch数据

由于第1种方法在硬盘上生成大量小文件，测试读取时IO开销太大。推荐采用第2种方法。

**代码实例**

以下两种方法为雷达数据的测试集生成代码实例，均可作为测试集生成方法。

**方法1**

雷达数据为包含2个数据集的hd5文件，数据集分别为data和label。data的大小为(5,270,350)，label的大小为(10,270,350)。生成测试集时，同时裁剪data数据和label数据。

# -\*- coding: utf-8 -\*-

# !/usr/bin/env python

"""

@Author: wmj

@Date: 2021/5/19

@Desc: 测试Dataset，加载大图，在内存中生成patch

"""

from torch.utils import data

import torch

import h5py

import numpy as np

import time

class DataSet(data.Dataset):

def \_\_init\_\_(self, radar\_list, load\_file\_count, start\_index=0, test=False):

"""

radar\_list: 雷达数据的绝对路径列表

load\_file\_count: 一次加载的列表数据数量

start\_index=0: 起始索引

test=False: False为处理训练集,True为处理测试集

"""

self.radar\_list = radar\_list

self.test = test

self.radar\_data = [] # 雷达数据

self.target\_cla = [] # 分类标签

self.target\_reg = [] # 回归标签

# 处理整图

for i in range(start\_index, start\_index + load\_file\_count):

start\_time = time.time()

# 读取整图

with h5py.File(self.radar\_list[i]) as fhandle:

radar\_data = fhandle[u'data'][:].astype(np.float32)

label = fhandle[u'label'][:].astype(np.float32)

# 测试

if self.test:

# 整图的大小为270\*350

for m in range(0, 270 - 54 + 1):

for n in range(0, 350 - 54 + 1):

data1 = radar\_data[:, m:m + 54, n:n + 54] # 裁剪雷达数据 54\*54

data2 = label[:, m:m + 54, n:n + 54] # 裁剪雷达标签 54\*54

self.radar\_data.append(data1) # 将数据添加进列表

# 处理标签,以每张图的中心点的4个值的均值作为样本的真实标签

# l1: 30分钟的标签

# l2: 60分钟的标签

x = data2[4::5]

z = []

l\_1 = (x[0, 26, 26] + x[0, 26, 27] + x[0, 27, 26] + x[

0, 27, 27]) / 4.0 \* 85.0 - 10.0

l\_2 = (x[1, 26, 26] + x[1, 26, 27] + x[1, 27, 26] + x[

1, 27, 27]) / 4.0 \* 85.0 - 10.0

if l\_1 > 35:

l\_1 = 1

else:

l\_1 = 0

if l\_2 > 35:

l\_2 = 1

else:

l\_2 = 0

z.append([l\_1, l\_2])

self.target\_cla.append(z) # 将分类标签添加进列表

self.target\_reg.append(x[:, 3:51, 3:51]) # 将回归标签添加进列表

print('load the file use %.6f time' % (time.time() - start\_time))

# 将列表转成数组

self.radar\_data = np.array(self.radar\_data)

self.target\_cla = np.array(self.target\_cla)

self.target\_reg = np.array(self.target\_reg)

def \_\_getitem\_\_(self, index):

return torch.from\_numpy(self.radar\_data[index]).unsqueeze(0).float(), \

torch.from\_numpy(self.target\_cla[index:index + 1]).squeeze(), \

torch.tensor(self.target\_reg[index:index + 1], dtype=torch.float32).squeeze()

# 裁剪后小图的数量

def \_\_len\_\_(self):

return len(self.target\_cla)

if \_\_name\_\_ == '\_\_main\_\_':

# dataset list

with open('./dataset\_path/path\_train\_radar.txt', 'r') as fhandle:

radar\_dataset\_list = fhandle.read().split('\n')

radar\_dataset\_list.pop()

# 从路径列表加载1条数据

data\_set = DataSet(radar\_dataset\_list, 1, test=True)

# 输出裁剪后的小图的数量

print('data\_set num :%d' % len(data\_set))

train\_loader = data.DataLoader(data\_set, batch\_size=256) # 测试集

for ii, (input, label\_cla, label\_reg) in enumerate(train\_loader):

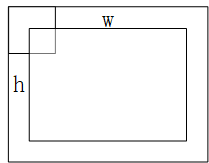
print(input.size())

print(label\_cla.size())

print(label\_reg.size())

**方法2**

假设有n张大图，每张大图能够生成m(m = h\*w)个patch，总的样本数为n\*m。Pytorch会为所有样本建立索引 [0, n\*m-1]，每次迭代会抽取batch\_size个索引。我们先根据索引计算(index // m)数据所在的大图，然后计算(index % m)数据是大图中的第几条数据，最后计算patch左上角的行、列值。通过切片取到一个完整的patch并返回。



# -\*- coding: utf-8 -\*-

"""

@Author: ZhangRui

@Date: 2021/5/19

@Desc: 测试Dataset，加载大图，在内存中生成patch

"""

import torch

import h5py

from torch.utils.data import Dataset, DataLoader

class VdrasDataset(Dataset):

def \_\_init\_\_(self, path):

self.temp = -1 # 记录加载进内存的文件

self.num = 200 \* 280 # 每张大图可以生成200\*280个patch

with open(path, 'r') as f:

self.samples = [line.strip() for line in f.readlines()] # 路径列表，每一条数据是一张大图的绝对路径

def \_\_len\_\_(self):

return len(self.samples) \* self.num # 大图的数量\*每张大图可以产生的patch的数量=总的样本数量

def \_\_getitem\_\_(self, item):

file\_no = item // self.num # 确定在哪个文件

self.index = item % self.num # 确定文件中的第几条数据

self.row = self.index // 280 # 确定path左上角所在行

self.col = self.index - self.row \* 280 # 确定patch左上角所在的列

#print("item:{}, file\_no:{}, row:{}, col:{}".format(item,file\_no,self.row,self.col))

if self.temp == file\_no:

# 说明文件已经在内存，直接取值返回

data = self.train\_data[:, :, self.row:self.row + 37, self.col:self.col + 37]

else: # 说明文件不在内存, 需要加载

self.temp = file\_no

with h5py.File(self.samples[file\_no], 'r') as f:

self.train\_data = f['data'][()]

# f.close() # with语句会自动关闭文件

#print(self.train\_data.shape)

data = self.train\_data[:, :, self.row:self.row + 37, self.col:self.col + 37]

return torch.from\_numpy(data).float() # 将numpy类型的数据转成tensor类型的数据，并指定为float类型

if \_\_name\_\_ == '\_\_main\_\_':

path = ""

dataset = VdrasDataset(path)

dataloader = DataLoader(dataset, batch\_size=1, shuffle=False, num\_workers=1, drop\_last=False)

i = 0

for item in dataloader:

assert item.shape == (1, 3, 120, 37, 37) # 断言，判断输出是否符合预期

i += 1

if i >= 2 \* 200 \* 280: # 作为测试，只读取了前两张大图

break