****广东海洋大学

**程序设计综合实践报告**

**(数据分析与软件开发方向)**

|  |
| --- |
| **基于SaCNN的Density Estimate实现** |

|  |  |
| --- | --- |
| 学生姓名 | 王精东 |
| 学 号 | 201711921328 |
| 班 级 | 数软1175 |
| 指导教师 | 石雄辉 |
| 成 绩 |  |
| 设计时间 | 2020.06.21——2020.07.12 |

教师评语：

# 问题简介

人群计数(crowd counting)和人群密度估计(pedestrian density estimate)是自人工智能兴起后的一个关注方向，该方向准确预估后人群数量后能够为公共安全提供可靠的参考。比如2015年年初35人在上海外滩的踩踏事件中失去了生命，对于踩踏事件的预警，该方向是颇为不错的选择。

该方向的研究主要是机器学习与深度学习两类，其中，机器学习多采用Adaboost集成学习方法将弱分类器转为强分类器，为了适应人头不同大小，多结合opencv提供的multi-scale detection一类的多尺度滑动窗口的方法[1,2]，但这种滑动窗口的缺陷是运算量大，多层循环导致detection的算法复杂度高，很难用于实时探测，并且由于滑窗算法本身易受滑动不能穷尽region of interesting而效果有所欠缺。

而深度学习方法则能够构造出复杂的模型，尤其是density estimate的模型能够得到较为精确的人数，而density map上的点同样能够得到每个人的位置信息，相比于大多数得到边框的算法，density map得到的是人头的估计中心，运用更方便。为了解决多尺度问题，方法是一种思路是采用多种大小的卷积核搭建卷积层，其中典型代表是Multi-column Neural Network(MCNN)[3],但对于该模型被认为多种卷积核的共存，尤其是大卷积核的存在，对于内存占用较大，而且多列卷积的方式使得最终模型容易存在大量冗余。故而Scale-adaptive Neural Network(SaCNN)[4]的提出针对这些问题进行了优化，该模型采用一种小卷积核(3x3)，但是在三个不同深度的位置保留了特征图，并分两批对特征图进行合并和处理，从而得到不同receptive field的特征图进而实现多尺度的detection。

# 数据预处理与探索性分析

为了保证模型能够对于任意场景的图像进行处理，如：不同大小的图片，不同时间不同日照的图片，模型对数据进行极少的数据预处理。但习惯上，对数据减去imagenet上的图像三通道均值使得图像数据。

另外，为了减少模型在predict阶段的计算规模，这里对较大图像进行放缩到最大边的大小为img\_side\_max，并排列成需要的数据排布。

代码如下：

VGG\_MEAN = [ 104.00698793, 116.66876762, 122.67891434]

img\_side\_max = 1200#177,79

def imgpreprocess(img):

if img.shape[0] > img\_side\_max or img.shape[1] > img\_side\_max:

img\_t = Image.fromarray(img)

if img.shape[0] == max(img.shape):

img\_t = img\_t.resize((int(img.shape[1]\*img\_side\_max/img.shape[0]),img\_side\_max))

img = np.array(img\_t)

else:

img\_t = img\_t.resize((img\_side\_max,int(img.shape[0]\*img\_side\_max/img.shape[1])))

img = np.array(img\_t)

img\_res = np.zeros(img.shape)

img\_res[:,:,0] = img[:,:,2]-VGG\_MEAN[0]

img\_res[:,:,1] = img[:,:,1]-VGG\_MEAN[1]

img\_res[:,:,2] = img[:,:,0]-VGG\_MEAN[2]

return np.expand\_dims(img\_res,axis = 0)

# 分类（预测）方法与处理结果

预测方法：SaCNN模型

对于一般的神经网络，可以如下表示一个神经元：

,

其中，是数据的第个输入，是激活函数，多采用relu函数，relu函数的解析式为

.

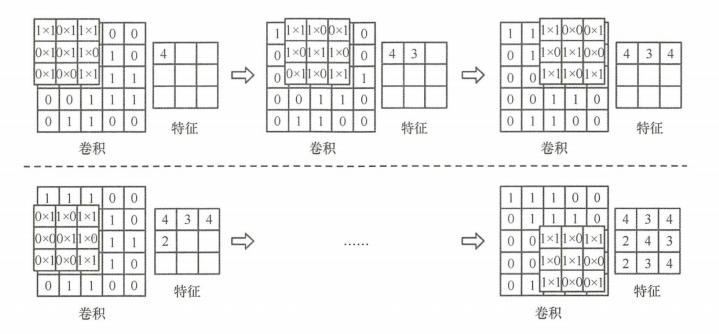
在卷积神经网络中，为了实现对空间特征的提取，借鉴图像处理的卷积，深度学习将其思想引入而有了如下卷积。另外，将卷积过程逆过来，则有反卷积操作。

图 1 卷积操作示意图[5]

激活函数的用法在这里便是对每一张特征图的每一个元素都通过一次激活函数得到相应的激活值。

为了减小计算规模以及为了方便得到更大的receptive field，又引入了如下的Pooling操作。

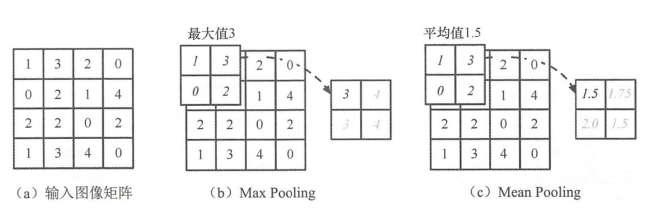


图 2 Pooling操作示意图

而仅采用这些操作时，而不采用全连接层时，则称为全卷积神经网络。这样的神经网络，输入与输出均为图像，能够方便实现图像之间的端到端的训练。

SaCNN的网络模型如下。

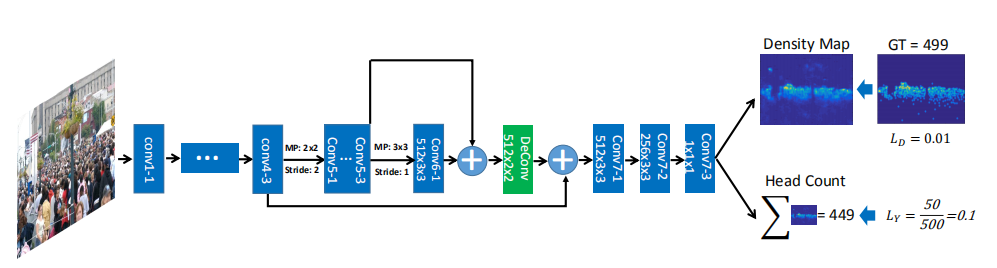


图 3 SaCNN网络结构

其中，backbone采用VGG16的一部分，直至conv4\_3层。

Loss function采用逐像素的残差平方和，最后除上batch size取平均()。当利用这个Loss训练出大致的density map效果后，以系数0.1乘上每张图的人头数误差平方和作为第二项Loss()，以此得到更好的效果并使得模型能够计算人头数。

训练网络时，同时起到数据增强的作用与减少计算规模，对输入图像进行切割，每张图切割成9张随机碎片，每张碎片是原图的1/4.本次实现简单起见，仅作简单4等分切割。

数据集采用上海科技大学收集的Crowd Counting数据集，部分展示如下：

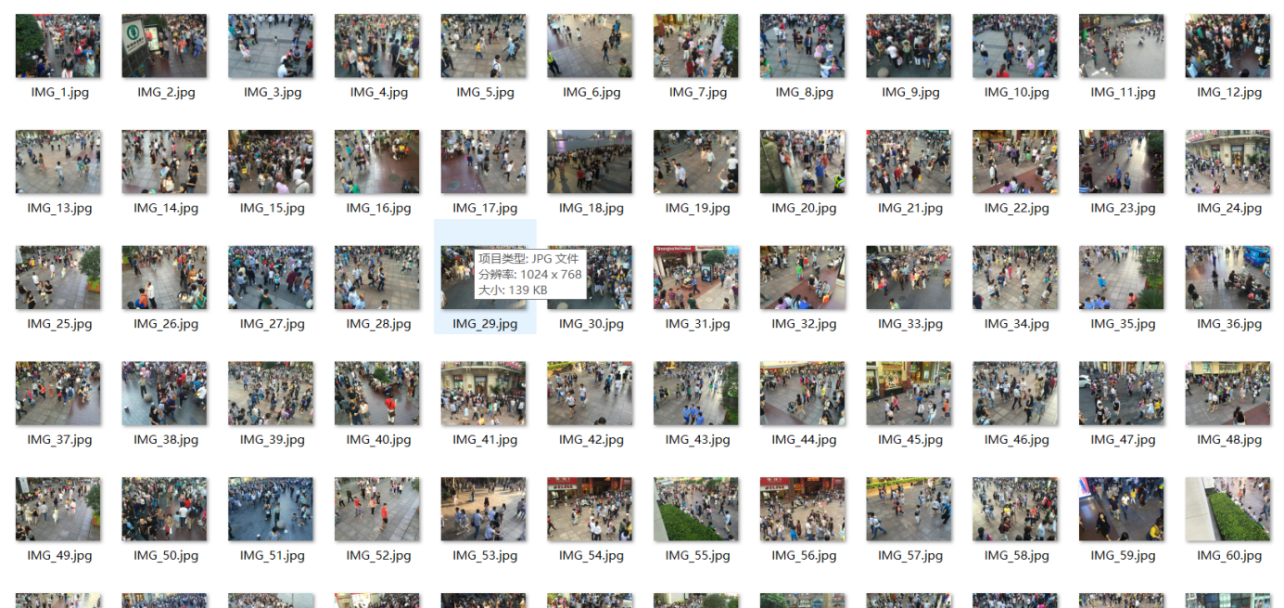


图 4 Crowd Counting数据集

代码结构如下：

Codes

|---------------data(数据集)

|---------------images（测试数据）

|---------------dmaps（生成的density map）

|---------------src

| |--------layers.py（网络框架）

| |--------sacnn.py（模型搭建）

| |--------utils.py（辅助文件操作）

|

|---------------vgg16.npy（VGG16的预训练参数）

|---------------train.py（基于tensorflow的模型训练）

|---------------SaCNN\_tf\_predict.py（基于tensorflow的模型预测）

|

|---------------two-scale\_model（另一种独立实现，但是欠拟合未完成训练）

| |-----src

| |-----vgg16.npy

| |-----train\_debug.py

| |-----grad\_check.py

|

|---------------caffe\_and\_tf

|-----SaCNN\_tf.py（利用sacnn\_tf.npy构建基于tensorflow的模型）

|-----sacnn\_caffe.py（基于caffe进行预测测试）

|-----deploy.prototxt（基于caffe的网络定义文件）

|-----list.txt（测试文件列表）

|-----predict\_caffe.py（基于caffe的预测模块）

|-----sacnn\_tf.npy（由SaCNN\_caffemodel2npy.py生成的npy文件）

|-----SaCNN\_caffemodel2npy.py（将caffemodel文件转为npy文件）

|-----ShanghaiTech\_part\_B.caffemodel（SaCNN的caffe预训练参数）

一些文件操作与图像操作的代码如下，命名为utils.py，并放置于src模块中。

import glob

import os

import random

import numpy as np

import tensorflow as tf

import sys

import cv2

#import src.layers as L

from PIL import Image

VGG\_MEAN = [103.939, 116.779, 123.68]

def img\_one2four(img,img\_type = 'data'):

'''对图像进行切割，一分为四'''

if img\_type == 'label':

img = np.expand\_dims(img,axis = [0,3])

img\_1 = img[:,:384,:512,:]#这里会变成[1,384,512,3]，仅对于上科大数据集

img\_2 = img[:,:384,512:,:]

img\_3 = img[:,384:,:512,:]

img\_4 = img[:,384:,512:,:]

if img\_type == 'label':

return [i[0,:,:,0] for i in [img\_1,img\_2,img\_3,img\_4]]

return [img\_1,img\_2,img\_3,img\_4]

def imgpreprocess(img):

#img = Image.fromarray(img.astype(np.uint8)).resize((512,384))

#img = np.array(img)

img\_res = np.zeros(img.shape)

img\_res[:,:,0] = img[:,:,2]-VGG\_MEAN[0]

img\_res[:,:,1] = img[:,:,1]-VGG\_MEAN[1]

img\_res[:,:,2] = img[:,:,0]-VGG\_MEAN[2]

return img\_res#.astype(np.float32)

def get\_density\_map\_gaussian(points, d\_map\_h, d\_map\_w): #后两个参数预估需要和最后的前馈输出结果做比较 ，从算法来看，应该是不用的

im\_density = np.zeros(shape=(d\_map\_h,d\_map\_w), dtype=np.float32)

if np.shape(points)[0] == 0: #如果输入的数据为空

sys.exit()

for i in range(np.shape(points)[0]): #遍历每一条记录

f\_sz = 15

sigma = 4

gaussian\_kernel = get\_gaussian\_kernel(f\_sz, f\_sz, sigma) #得到fsize，长宽为15的高斯核

x = min(d\_map\_w, max(1, np.abs(np.int32(np.floor(points[i, 0])))))

y = min(d\_map\_h, max(1, np.abs(np.int32(np.floor(points[i, 1])))))

if(x > d\_map\_w or y > d\_map\_h):

continue

x1 = x - np.int32(np.floor(f\_sz / 2))

y1 = y - np.int32(np.floor(f\_sz / 2))

x2 = x + np.int32(np.floor(f\_sz / 2))

y2 = y + np.int32(np.floor(f\_sz / 2))

dfx1 = 0

dfy1 = 0

dfx2 = 0

dfy2 = 0

change\_H = False

if(x1 < 1):

dfx1 = np.abs(x1)+1

x1 = 1

change\_H = True

if(y1 < 1):

dfy1 = np.abs(y1)+1

y1 = 1

change\_H = True

if(x2 > d\_map\_w):

dfx2 = x2 - d\_map\_w

x2 = d\_map\_w

change\_H = True

if(y2 > d\_map\_h):

dfy2 = y2 - d\_map\_h

y2 = d\_map\_h

change\_H = True

x1h = 1+dfx1

y1h = 1+dfy1

x2h = f\_sz - dfx2

y2h = f\_sz - dfy2

if (change\_H == True):

f\_sz\_y = np.double(y2h - y1h + 1)

f\_sz\_x = np.double(x2h - x1h + 1)

gaussian\_kernel = get\_gaussian\_kernel(f\_sz\_x, f\_sz\_y, sigma)

im\_density[y1-1:y2,x1-1:x2] = im\_density[y1-1:y2,x1-1:x2] + gaussian\_kernel

return im\_density

def get\_gaussian\_kernel(fs\_x, fs\_y, sigma):

gaussian\_kernel\_x = cv2.getGaussianKernel(ksize=np.int(fs\_x), sigma=sigma)

gaussian\_kernel\_y = cv2.getGaussianKernel(ksize=np.int(fs\_y), sigma=sigma)

gaussian\_kernel = gaussian\_kernel\_y \* gaussian\_kernel\_x.T

return gaussian\_kernel

def compute\_abs\_err(pred, gt):

return np.abs(np.sum(pred[:]) - np.sum(gt[:]))

def create\_session(log\_dir, session\_id):

folder\_path = os.path.join(log\_dir, 'session-'+str(session\_id))

if os.path.exists(folder\_path):

print ('Session already taken. It will create a different session id.')#所以最好每次删了logs文件夹

#sys.exit()

else:

os.makedirs(folder\_path)

return folder\_path

def get\_file\_id(filepath):

return os.path.splitext(os.path.basename(filepath))[0]

def get\_data\_list(data\_root, mode='train'):

if mode == 'train':

imagepath = os.path.join(data\_root, 'train\_data', 'images')

gtpath = os.path.join(data\_root, 'train\_data', 'ground\_truth')

elif mode == 'valid':

imagepath = os.path.join(data\_root, 'valid\_data', 'images')

gtpath = os.path.join(data\_root, 'valid\_data', 'ground\_truth')

else:

imagepath = os.path.join(data\_root, 'test\_data', 'images')

gtpath = os.path.join(data\_root, 'test\_data', 'ground\_truth')

image\_list = [file for file in glob.glob(os.path.join(imagepath,'\*.jpg'))]

gt\_list = []

for filepath in image\_list:

file\_id = get\_file\_id(filepath)

gt\_file\_path = os.path.join(gtpath, 'GT\_'+ file\_id + '.mat')

gt\_list.append(gt\_file\_path)

xy = list(zip(image\_list, gt\_list)) #列表中为元组

random.shuffle(xy) #会更新数据

s\_image\_list, s\_gt\_list = zip(\*xy) #zip(\*)可理解为解压

return s\_image\_list, s\_gt\_list

def reshape\_tensor(tensor,channel):

r\_tensor = np.reshape(tensor, newshape=(1, tensor.shape[0], tensor.shape[1], channel)) #

return r\_tensor

def save\_weights(graph, fpath):

sess = tf.get\_default\_session()

variables = graph.get\_collection("variables")

variable\_names = [v.name for v in variables]

kwargs = dict(zip(variable\_names, sess.run(variables)))

np.savez(fpath, \*\*kwargs)

def load\_weights(graph, fpath):

sess = tf.get\_default\_session()

variables = graph.get\_collection("variables")

data = np.load(fpath)

for v in variables:

if v.name not in data:

print("could not load data for variable='%s'" % v.name)

continue

print("assigning %s" % v.name)

sess.run(v.assign(data[v.name]))

def labelmap(img,loc):

img[loc.astype('int')] = 255

return img

def resize\_gt(train\_d\_map\_r):

'''resize ground truth'''

img = Image.fromarray(train\_d\_map\_r)

#img = img.resize(compute\_downsampling(512,384))

img = img.resize(compute\_downsampling(train\_d\_map\_r.shape[1],train\_d\_map\_r.shape[0]))

img = np.array(img)

real\_sum = train\_d\_map\_r.sum()

if real\_sum!=0:

factor = train\_d\_map\_r.sum()/img.sum()

else:

factor = 0

return img\*factor

def compute\_downsampling(h,w):

for i in range(3):

if h%2 == 0:

h = h/2

else:

h = np.ceil(h/2)

if w%2 == 0:

w = w/2

else:

w = np.ceil(w/2)

return int(h),int(w)

网络框架封装文件如下，命名为layers.py,同样放置于src模块中。

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

import tensorflow as tf

import numpy as np

vgg\_param = np.load('vgg16.npy',encoding = 'latin1',allow\_pickle = True).item()

def vgg\_conv(x,name,trainable = False):

with tf.variable\_scope(name):

if trainable :

gene\_fn = tf.Variable

else:

gene\_fn = tf.constant

return tf.nn.relu(tf.nn.bias\_add(

tf.nn.conv2d(x,gene\_fn(vgg\_param[name][0],dtype = tf.float32),

(1,1,1,1),padding = 'SAME',name = name),

gene\_fn(vgg\_param[name][1],dtype = tf.float32)))

def vgg\_pool(x,name):

with tf.variable\_scope(name):

return tf.nn.max\_pool(x,

ksize=[1,2, 2 , 1],

strides=[1, 2, 2, 1],

padding='SAME',

name=name)

def conv(input\_tensor, name, kw, kh, n\_out, dw=1, dh=1, activation\_fn=tf.nn.relu,trainable = True):

n\_in = input\_tensor.get\_shape()[-1].value

with tf.variable\_scope(name):

if trainable :

gene\_fn = tf.Variable

else:

gene\_fn = tf.constant

weights = gene\_fn(tf.truncated\_normal(shape=(kh, kw, n\_in, n\_out), mean = 0.0,stddev=0.1), dtype=tf.float32, name='weights')

biases = gene\_fn(tf.constant(0.0, shape=[n\_out]), dtype=tf.float32, name='biases')

conv = tf.nn.conv2d(input\_tensor, weights, (1, dh, dw, 1), padding='SAME')

if activation\_fn :

activation = activation\_fn(tf.nn.bias\_add(conv, biases))

else:

activation = tf.nn.bias\_add(conv,biases)

tf.summary.histogram("weights", weights)

return activation

def deconv(input\_tensor,name ,kw,kh,n\_out,out\_shape ,dw=2,dh=2,activation\_fn=tf.nn.leaky\_relu):

"这是反卷积"

n\_in = input\_tensor.get\_shape()[-1].value

#因为直接作为numpy数组来的话，似乎全都拿不到，这样的话，我至少自己指定最后一维，第一维他会自己拿到，但，，emm我怀疑会出问题

with tf.variable\_scope(name):

weights = tf.Variable(tf.truncated\_normal(shape = (kh,kw,n\_out,n\_in),stddev = 0.01),dtype = tf.float32,name = 'weights')

#biases = tf.Variable(tf.constant(0.0,shape = [n\_out]),dtype = tf.float32,name = 'biases')

deconv\_t = tf.nn.conv2d\_transpose(input\_tensor,weights,out\_shape ,strides = (1,dh,dw,1),padding = 'SAME' )

#conv2d\_transpose(value, filter, output\_shape, strides, padding="SAME",

# data\_format="NHWC", name=None)

return deconv\_t

def pool(input\_tensor, name, kh, kw, dh, dw):

return tf.nn.max\_pool(input\_tensor,

ksize=[1, kh, kw, 1],

strides=[1, dh, dw, 1],

padding='SAME',

name=name)

def mean\_pool(input\_tensor, name, kh, kw, dh, dw):

return tf.nn.avg\_pool(input\_tensor,

ksize=[1, kh, kw, 1],

strides=[1, dh, dw, 1],

padding='SAME',

name=name)

def loss(est, gt):

return 0\*tf.reduce\_mean(tf.pow((tf.reduce\_max(est,axis = [1,2,3])-tf.reduce\_max(gt,axis = [1,2,3])),2))+\

1\*tf.reduce\_mean(tf.reduce\_sum(tf.pow((est-gt),2),axis = [1,2,3]))+\

0.1\*tf.reduce\_mean(tf.pow(((tf.reduce\_sum(est,axis = [1,2,3])-\

tf.reduce\_sum(gt,axis = [1,2,3]))/(tf.reduce\_sum(gt,axis = [1,2,3])+1)),2))

模型搭建文件如下，命名为sacnn.py，放置于src模块中。

import tensorflow as tf

import src.layers as L

import numpy as np

csz = (1/8)#change size 的比例

#out\_art = 512

out\_art = 512

out1 = 64#16

out2 = 64#32

def backbone(input\_tensor):#based on VGG16

x = L.vgg\_conv(input\_tensor,'conv1\_1')

x = L.vgg\_conv(x,'conv1\_2')

x = L.vgg\_pool(x,'pool1\_1')

x = L.vgg\_conv(x,'conv2\_1')

x = L.vgg\_conv(x,'conv2\_2')

x = L.vgg\_pool(x,'pool2\_1')

x = L.vgg\_conv(x,'conv3\_1')

x = L.vgg\_conv(x,'conv3\_2')

x = L.vgg\_conv(x,'conv3\_3')

x = L.vgg\_pool(x,'pool3\_1')

x = L.vgg\_conv(x,'conv4\_1')#,trainable = True)

x = L.vgg\_conv(x,'conv4\_2')#,trainable = True)

x = L.vgg\_conv(x,'conv4\_3')#,trainable = True)

return x

def subscale\_1(x):

net = L.pool(x,name = 'pool4',kh=2,kw=2,dw=2,dh=2)

net = L.conv(net,name = 'conv5\_1',kh = 3,kw = 3,n\_out = out\_art)

net = L.conv(net,name = 'conv5\_2',kh = 3,kw = 3,n\_out = out\_art)

net = L.conv(net,name = 'conv5\_3',kh = 3,kw = 3,n\_out = out\_art)

return net

def subscale\_2(x):

x = L.pool(x,name = 'pool5',kh=3,kw=3,dw=1,dh=1)

net = L.conv(x,name = 'conv6\_1',kh = 3,kw = 3,n\_out = out\_art)

return net

def fuse\_layer(x1, x2):

x\_concat = tf.concat([x1, x2],axis=3)

return x\_concat

def build(input\_tensor):

net1 = backbone(input\_tensor)

net2 = subscale\_1(net1)

net3 = subscale\_2(net2)

fuse1 = fuse\_layer(net2,net3)

h3 = tf.shape(net1)[1]

w3 = tf.shape(net1)[2]

fuse1 = L.model\_deconv(fuse1,'conv\_concat1\_2x',

out\_shape = [1,h3,w3,1])#[1,?,?,1024]

fuse2 = fuse\_layer(net1,fuse1)

fuse2 = L.conv(fuse2,name = 'p\_conv1',kh = 3,kw = 3,n\_out = out\_art)

fuse2 = L.conv(fuse2,name = 'p\_conv2',kh = 3,kw = 3,n\_out = 256)

fuse2 = L.conv(fuse2,name = 'p\_conv3',kh = 1,kw = 1,n\_out = 1,activation\_fn = None)

return fuse2,fuse1#第二个返回项可以自定义，用于检查梯度

模型训练文件如下，命名为train.py,放置于data文件夹的同级目录下：

import tensorflow as tf

import src.tscnn as tscnn

import src.layers as L

import os

import src.utils as utils

import numpy as np

import matplotlib.image as mpimg

import scipy.io as sio

import time

import argparse

#import sys

#from PIL import Image

import pylab as plt

import keras.backend.tensorflow\_backend as KTF

#tf.device('/gpu:0')

# Global Constants. Define the number of images for training, validation and testing.

NUM\_TRAIN\_IMGS = 6000

NUM\_VAL\_IMGS = 590

NUM\_TEST\_IMGS = 587

MODEL\_SAVE\_PATH = './model/'

MODEL\_NAME = 'tscnn\_model'

batch\_size = 4

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

parser = argparse.ArgumentParser()

parser.add\_argument('--retrain', default=False, type=bool)

parser.add\_argument('--base\_model\_path', default=None, type=str)

parser.add\_argument('--log\_dir', default = './logs', type=str)

parser.add\_argument('--num\_epochs', default = 2000, type=int)

parser.add\_argument('--learning\_rate', default = 0.0001, type=float)

parser.add\_argument('--session\_id', default = 2, type=int)

parser.add\_argument('--data\_root', default='./data/comb\_dataset\_v3', type=str)

args = parser.parse\_args()

args.retrain = True

temp\_use\_path = os.path.join(args.log\_dir, 'session-'+str(args.session\_id))

if os.path.exists(temp\_use\_path):

session\_id\_list = [tempath.split('-')[-1] for tempath in os.listdir(args.log\_dir)]

session\_idl = [eval(comid) for comid in session\_id\_list]

session\_id\_t = max(session\_idl)

if args.retrain:

args.session\_id = session\_id\_t

#args.session\_id = 133

else:

args.session\_id = session\_id\_t+1

sess\_path = utils.create\_session(args.log\_dir, args.session\_id) # Create a session path based on the session id.

if args.retrain:

model\_list = os.listdir(sess\_path)

model\_list = [i for i in model\_list if '.npz' in i]

model\_list = [eval(i.split('.')[1]) for i in model\_list]

model\_final = max(model\_list)

args.base\_model\_path = sess\_path+f'/weights.{model\_final}.npz'

config = tf.ConfigProto()

config.gpu\_options.allow\_growth = True

G = tf.Graph()

with G.as\_default():

# Create image and density map placeholder

image\_place\_holder = tf.placeholder(tf.float32, shape=[None, None, None, 3]) #原本是1，沙雕东西

d\_map\_place\_holder = tf.placeholder(tf.float32, shape=[None, None, None, 1])

# Build all nodes of the network

d\_map\_est,net\_mid = tscnn.build(image\_place\_holder)

# Define the loss function.

with tf.variable\_scope('loss'):

euc\_loss = L.loss(d\_map\_est, d\_map\_place\_holder)

# Define the optimization algorithm

#optimizer = tf.train.GradientDescentOptimizer(args.learning\_rate)

#optimizer= tf.train.AdamOptimizer(0.001)

optimizer = tf.train.RMSPropOptimizer(0.000001)

# Training node.

train\_op = optimizer.minimize(euc\_loss)

train\_grad = tf.gradients(euc\_loss,image\_place\_holder)

train\_grad = tf.reduce\_mean(train\_grad)

# Initialize all the variables.

init = tf.group(tf.global\_variables\_initializer(), tf.local\_variables\_initializer())

# For summary

summary = tf.summary.merge\_all()

with tf.Session(graph=G,config=config) as sess:#坚决不能用relu，不然大多数脑袋都没了

KTF.set\_session(sess)

writer = tf.summary.FileWriter(os.path.join(sess\_path,'training\_logging'))

writer.add\_graph(sess.graph)

sess.run(init) #完成初始化

if args.retrain:

utils.load\_weights(G, args.base\_model\_path)

# Start the epochs

errs = []

for eph in range(args.num\_epochs):#训练的轮数

start\_train\_time = time.time()

# Get the list of train images.

train\_images\_list, train\_gts\_list = utils.get\_data\_list(args.data\_root, mode='train')

#total\_train\_loss = 0

# Loop through all the training images

train\_y,train\_x = [],[]

for img\_idx in range(len(train\_images\_list)):#每张图片 捞出来训练

# Load the image and ground truth

train\_image = np.asarray(mpimg.imread(train\_images\_list[img\_idx]), dtype=np.float32)

train\_image = utils.imgpreprocess(train\_image)

train\_d\_map = np.asarray(sio.loadmat(train\_gts\_list[img\_idx])['image\_info'][0][0][0][0][0], dtype=np.float32)

# Reshape the tensor before feeding it to the network #等价于np.expand\_dims

train\_image\_r = utils.reshape\_tensor(train\_image,3) #将代码修改为根据第二个参数进行设定第4维度

train\_d\_map\_r = utils.get\_density\_map\_gaussian(train\_d\_map,train\_image.shape[0],train\_image.shape[1]) #通过高斯卷积（模糊）得到密度图

train\_d\_map\_r = utils.resize\_gt(train\_d\_map\_r)#新增放缩算法

train\_d\_map\_r = utils.reshape\_tensor(train\_d\_map\_r,1)

if img\_idx%batch\_size!=0 or img\_idx == 0:

train\_x.append(train\_image\_r)

train\_y.append(train\_d\_map\_r)

continue

train\_x.append(train\_image\_r)

train\_y.append(train\_d\_map\_r)

train\_image\_r = np.concatenate(train\_x,axis = 0)

train\_d\_map\_r = np.concatenate(train\_y,axis = 0)

train\_x,train\_y = [],[]

#train\_d\_map\_r = utils.downsampling\_d\_map(train\_d\_map\_r,sess)

# Prepare feed\_dict

feed\_dict\_data = {

image\_place\_holder: train\_image\_r,

d\_map\_place\_holder: train\_d\_map\_r,

}

# Compute the loss for one image.

sess.run(train\_op,feed\_dict = feed\_dict\_data)

#输出loss使用的两个输出，查看形状

if img\_idx %200== 0:

d\_map\_view,loss\_per\_image = sess.run([d\_map\_est,euc\_loss],feed\_dict = feed\_dict\_data)

print('-------------------')

print(d\_map\_view.shape,d\_map\_view.dtype,d\_map\_view.min(),d\_map\_view.max(),d\_map\_view.mean())

errs.append(loss\_per\_image )

#grads.append(grad\_t)

plt.figure()

plt.imshow(d\_map\_view[0,:,:,0])

plt.colorbar()

plt.title(img\_idx)

d\_map\_true = train\_d\_map\_r

plt.figure()

plt.imshow(d\_map\_true[0,:,:,0])

plt.colorbar()

plt.title(img\_idx)

plt.figure()

if len(errs)<20:

plt.plot(errs)

else:

plt.plot(errs[15:])

plt.show()

print(f'Loss of {img\_idx} is :{loss\_per\_image}')

print('head count of d\_map\_view is :',d\_map\_view.sum())

# Save the weights as well as the summary

utils.save\_weights(G, os.path.join(sess\_path, "weights.%s" % (eph+1)))

summary\_str = sess.run(summary, feed\_dict=feed\_dict\_data)

writer.add\_summary(summary\_str, eph)

end\_train\_time = time.time()

train\_duration = end\_train\_time - start\_train\_time

# Compute the average training loss

#avg\_train\_loss = total\_train\_loss / len(train\_images\_list)

# Then we print the results for this epoch:

print("Epoch {} of {} took {:.3f}s".format(eph + 1, args.num\_epochs, train\_duration))

预测文件如下，命名为SaCNN\_tf\_predict.py。

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

import tensorflow as tf

import os

import numpy as np

import pylab as plt

import keras.backend.tensorflow\_backend as KTF

from src.sacnn import build

import time

from matplotlib import pyplot as plt

from PIL import Image

VGG\_MEAN = [103.939, 116.779, 123.68]

base\_model\_path = ‘./logs/session-133/training\_logging/weights.24.1.npy’

#VGG\_MEAN = [ 104.00698793, 116.66876762, 122.67891434]

img\_side\_max = 1200#177,79

def imgpreprocess(img):

if img.shape[0] > img\_side\_max or img.shape[1] > img\_side\_max:

img\_t = Image.fromarray(img)

if img.shape[0] == max(img.shape):

img\_t = img\_t.resize((int(img.shape[1]\*img\_side\_max/img.shape[0]),img\_side\_max))

img = np.array(img\_t)

else:

img\_t = img\_t.resize((img\_side\_max,int(img.shape[0]\*img\_side\_max/img.shape[1])))

img = np.array(img\_t)

img\_res = np.zeros(img.shape)

img\_res[:,:,0] = img[:,:,2]-VGG\_MEAN[0]

img\_res[:,:,1] = img[:,:,1]-VGG\_MEAN[1]

img\_res[:,:,2] = img[:,:,0]-VGG\_MEAN[2]

return np.expand\_dims(img\_res,axis = 0)

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

config = tf.ConfigProto()

config.gpu\_options.allow\_growth = True

input\_path = 'images/'

dmap\_path = 'dmaps/'

for i in [input\_path,dmap\_path]:

if not os.path.exists(i):

os.mkdir(i)

G = tf.Graph()

with G.as\_default():

image\_place\_holder = tf.placeholder(tf.float32, shape=[None, None, None, 3])

d\_map\_est,net\_mid = build(image\_place\_holder)

utils.load\_weights(G,base\_model\_path)

summary = tf.summary.merge\_all()

with tf.Session(graph=G,config=config) as sess:

KTF.set\_session(sess)

writer = tf.summary.FileWriter(os.path.join('logging'))

writer.add\_graph(sess.graph)

t\_old = time.time()

time.sleep(1)

while (time.time()-t\_old)>1:

t\_old = time.time()

imgs = os.listdir(input\_path)

res\_imgs = os.listdir(dmap\_path)

imgs = [input\_path+i for i in imgs if os.path.split(i)[-1] not in res\_imgs]

imgs\_net\_ls = [imgpreprocess(plt.imread(i)) for i in imgs]

for i in range(len(imgs\_net\_ls)):

imgs\_net = imgs\_net\_ls[i]

d\_map = sess.run(d\_map\_est,feed\_dict = {image\_place\_holder:imgs\_net})

res\_t = d\_map[0,:,:,0]

plt.imshow(res\_t)

plt.title(res\_t.sum())

plt.axis('off')

print(os.path.split(imgs[i])[-1])

plt.savefig(os.path.join(dmap\_path,os.path.split(imgs[i])[-1]))

np.save(os.path.join(dmap\_path,os.path.split(imgs[i])[-1]),res\_t)

plt.clf()

fig = plt.figure(figsize = (16,8))

ax = fig.add\_subplot(121)

ax.imshow(plt.imread(imgs[i]))

ax = fig.add\_subplot(122)

ax.imshow(res\_t)

plt.title(res\_t.sum())

plt.show()

由于显存不够进行模型训练，这里采用已有的caffe预训练参数进行效果展示，测试列表文件为list.txt。

D:\\Medium\\Codes\\SACNN\_application\\images/IMG\_8.jpg

D:\\Medium\\Codes\\SACNN\_application\\images/IMG\_9.jpg

网络定义文件为deploy.prototxt。

name: "SaCNN"

layer {

name: "data"

type: "ImageData"

top: "data"

top: "label"

image\_data\_param {

is\_color: true

source: "./list.txt"

batch\_size: 1

}

transform\_param {

}

}

layer {

bottom: "data"

top: "conv1\_1"

name: "conv1\_1"

type: "Convolution"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 64

pad: 1

kernel\_size: 3

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 0

}

}

}

layer {

bottom: "conv1\_1"

top: "conv1\_1"

name: "relu1\_1"

type: "ReLU"

}

layer {

bottom: "conv1\_1"

top: "conv1\_2"

name: "conv1\_2"

type: "Convolution"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 64

pad: 1

kernel\_size: 3

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 0

}

}

}

layer {

bottom: "conv1\_2"

top: "conv1\_2"

name: "relu1\_2"

type: "ReLU"

}

layer {

bottom: "conv1\_2"

top: "pool1"

name: "pool1"

type: "Pooling"

pooling\_param {

pool: MAX

kernel\_size: 2

stride: 2

}

}

layer {

bottom: "pool1"

top: "conv2\_1"

name: "conv2\_1"

type: "Convolution"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 128

pad: 1

kernel\_size: 3

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 0

}

}

}

layer {

bottom: "conv2\_1"

top: "conv2\_1"

name: "relu2\_1"

type: "ReLU"

}

layer {

bottom: "conv2\_1"

top: "conv2\_2"

name: "conv2\_2"

type: "Convolution"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 128

pad: 1

kernel\_size: 3

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 0

}

}

}

layer {

bottom: "conv2\_2"

top: "conv2\_2"

name: "relu2\_2"

type: "ReLU"

}

layer {

bottom: "conv2\_2"

top: "pool2"

name: "pool2"

type: "Pooling"

pooling\_param {

pool: MAX

kernel\_size: 2

stride: 2

}

}

layer {

bottom: "pool2"

top: "conv3\_1"

name: "conv3\_1"

type: "Convolution"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 256

pad: 1

kernel\_size: 3

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 0

}

}

}

layer {

bottom: "conv3\_1"

top: "conv3\_1"

name: "relu3\_1"

type: "ReLU"

}

layer {

bottom: "conv3\_1"

top: "conv3\_2"

name: "conv3\_2"

type: "Convolution"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 256

pad: 1

kernel\_size: 3

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 0

}

}

}

layer {

bottom: "conv3\_2"

top: "conv3\_2"

name: "relu3\_2"

type: "ReLU"

}

layer {

bottom: "conv3\_2"

top: "conv3\_3"

name: "conv3\_3"

type: "Convolution"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 256

pad: 1

kernel\_size: 3

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 0

}

}

}

layer {

bottom: "conv3\_3"

top: "conv3\_3"

name: "relu3\_3"

type: "ReLU"

}

layer {

bottom: "conv3\_3"

top: "pool3"

name: "pool3"

type: "Pooling"

pooling\_param {

pool: MAX

kernel\_size: 2

stride: 2

}

}

layer {

bottom: "pool3"

top: "conv4\_1"

name: "conv4\_1"

type: "Convolution"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 512

pad: 1

kernel\_size: 3

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 0

}

}

}

layer {

bottom: "conv4\_1"

top: "conv4\_1"

name: "relu4\_1"

type: "ReLU"

}

layer {

bottom: "conv4\_1"

top: "conv4\_2"

name: "conv4\_2"

type: "Convolution"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 512

pad: 1

kernel\_size: 3

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 0

}

}

}

layer {

bottom: "conv4\_2"

top: "conv4\_2"

name: "relu4\_2"

type: "ReLU"

}

layer {

bottom: "conv4\_2"

top: "conv4\_3"

name: "conv4\_3"

type: "Convolution"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output:512

pad: 1

kernel\_size: 3

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 0

}

}

}

layer {

bottom: "conv4\_3"

top: "conv4\_3"

name: "relu4\_3"

type: "ReLU"

}

layer {

bottom: "conv4\_3"

top: "pool4"

name: "pool4"

type: "Pooling"

pooling\_param {

pool: MAX

kernel\_size: 2

stride: 2

}

}

layer {

bottom: "pool4"

top: "conv5\_1"

name: "conv5\_1"

type: "Convolution"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 512

pad: 1

kernel\_size: 3

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 0

}

}

}

layer {

bottom: "conv5\_1"

top: "conv5\_1"

name: "relu5\_1"

type: "ReLU"

}

layer {

bottom: "conv5\_1"

top: "conv5\_2"

name: "conv5\_2"

type: "Convolution"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 512

pad: 1

kernel\_size: 3

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 0

}

}

}

layer {

bottom: "conv5\_2"

top: "conv5\_2"

name: "relu5\_2"

type: "ReLU"

}

layer {

bottom: "conv5\_2"

top: "conv5\_3"

name: "conv5\_3"

type: "Convolution"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 512

pad: 1

kernel\_size: 3

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 0

}

}

}

layer {

bottom: "conv5\_3"

top: "conv5\_3"

name: "relu5\_3"

type: "ReLU"

}

# add more conv

layer {

bottom: "conv5\_3"

top: "pool5"

name: "pool5"

type: "Pooling"

pooling\_param {

pool: MAX

kernel\_size: 3

stride: 1

pad: 1

}

}

layer {

bottom: "pool5"

top: "conv6\_1"

name: "conv6\_1"

type: "Convolution"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 512

pad: 1

kernel\_size: 3

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

}

}

}

layer {

bottom: "conv6\_1"

top: "conv6\_1"

name: "relu6\_1"

type: "ReLU"

}

layer {

name: "concat1"

bottom: "conv5\_3"

bottom: "conv6\_1"

top: "conv\_concat1"

type: "Concat"

concat\_param {

axis: 1

}

}

layer {

bottom: "conv\_concat1"

top: "conv\_concat1\_2x"

name: "conv\_concat1\_2x"

type: "Deconvolution"

convolution\_param {

kernel\_size: 2

stride: 2

num\_output: 512

group: 512

pad: 0

weight\_filler: {

type: "constant"

value: 1

}

bias\_term: false

}

param {

lr\_mult: 0

decay\_mult: 0

}

}

layer {

name: "concat"

bottom: "conv4\_3"

bottom: "conv\_concat1\_2x"

top: "conv\_concat"

type: "Concat"

concat\_param {

axis: 1

}

}

layer {

name: "p\_conv1"

type: "Convolution"

bottom: "conv\_concat"

top: "p\_conv1"

param {

lr\_mult: 1

}

param {

lr\_mult: 2

}

convolution\_param {

num\_output: 512

pad: 1

kernel\_size: 3

stride: 1

weight\_filler {

type: "gaussian"

std: 0.01000000

}

bias\_filler {

type: "constant"

value: 0

}

}

}

layer {

name: "relu\_p\_conv1"

type: "ReLU"

bottom: "p\_conv1"

top: "p\_conv1"

}

layer {

name: "p\_conv2"

type: "Convolution"

bottom: "p\_conv1"

top: "p\_conv2"

param {

lr\_mult: 1

}

param {

lr\_mult: 2

}

convolution\_param {

num\_output: 256

pad: 1

kernel\_size: 3

stride: 1

weight\_filler {

type: "gaussian"

std: 0.01000000

}

bias\_filler {

type: "constant"

value: 0

}

}

}

layer {

name: "relu\_p\_conv2"

type: "ReLU"

bottom: "p\_conv2"

top: "p\_conv2"

}

layer {

bottom: "p\_conv2"

top: "estdmap"

name: "p\_conv3"

type: "Convolution"

param {

lr\_mult: 10

decay\_mult: 1

}

param {

lr\_mult: 20

decay\_mult: 0

}

convolution\_param {

num\_output: 1

kernel\_size: 1

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 0

}

}

}

预测模块文件为predict\_caffe.py。

import numpy as np

import matplotlib.pyplot as plt

import caffe

import os

import sys

from PIL import Image

plt.rcParams['image.interpolation'] = 'nearest'

#plt.rcParams['image.cmap'] = 'gray'

caffe\_root = 'D:\\Medium\\Codes\\Caffe\_build\\caffe/'

sys.path.insert(0, caffe\_root + 'python')

if os.path.isfile('ShanghaiTech\_part\_B.caffemodel'):

print('CaffeNet found.')

else:

print('CaffeNet Not found.')

caffe.set\_mode\_cpu()

model\_def = 'deploy.prototxt'

model\_weights = 'ShanghaiTech\_part\_B.caffemodel'

net = caffe.Net(model\_def,model\_weights,caffe.TEST)

mu = np.load(caffe\_root + 'python/caffe/imagenet/ilsvrc\_2012\_mean.npy')

mu = mu.mean(1).mean(1)

print('mean-subtracted values:', zip('BGR', mu))

def data2image(data):

img = np.zeros((data.shape[2],data.shape[3],data.shape[1]))

img[:,:,2] = data[0,0,:,:]

img[:,:,1] = data[0,1,:,:]

img[:,:,0] = data[0,2,:,:]

return img

img\_side\_max = 1500

def imgadapt(img):

#try:

if img.shape[0] > img\_side\_max or img.shape[1] > img\_side\_max:

img\_t = Image.fromarray(img)

if img.shape[0] == max(img.shape):

img\_t = img\_t.resize((int(img.shape[1]\*img\_side\_max/img.shape[0]),img\_side\_max))

img = np.array(img\_t)

else:

img\_t = img\_t.resize((img\_side\_max,int(img.shape[0]\*img\_side\_max/img.shape[1])))

img = np.array(img\_t)

#except:

# print('Data type is ',img.dtype)

return img.astype(np.float32)

def predict(imgfile=r'D:\Medium\Codes\SACNN\SaCNN-CrowdCounting-Tencent\_Youtu-master\SaCNN-master\data\comb\_dataset\_v3\part\_B\_final\train\_data\images/IMG\_109.jpg'):

image = caffe.io.load\_image(imgfile)

image = imgadapt(image.astype(np.uint8))

print(image.shape)

transformer = caffe.io.Transformer({'data': (1,3,image.shape[0],image.shape[1])})

transformer.set\_transpose('data', (2,0,1))

transformer.set\_mean('data', mu)

transformer.set\_raw\_scale('data', 255)

transformer.set\_channel\_swap('data', (2,1,0))

net.blobs['data'].reshape(1,3,image.shape[0],image.shape[1])

transformed\_image = transformer.preprocess('data', image)

net.blobs['data'].data[...] = transformed\_image

print(imgfile,net.blobs['data'].data.shape)

out = net.forward()

print(out['estdmap'].shape)

output = out['estdmap'].data

return np.array(output)[0,0,:,:]

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

d\_map = predict()

plt.imshow(d\_map)

plt.show()

基于caffe框架的预测代码，sacnn\_caffe.py。

import os

from predict\_caffe import predict

from matplotlib import pyplot as plt

import time

from PIL import Image

import numpy as np

path = r'D:\Medium\Codes\SACNN\_application\images/'

fs = os.listdir(path)

fs = [path + i for i in fs ]

imgs = [plt.imread(i) for i in fs]

#imgs = [np.array(Image.fromarray(i).resize((768,1024))) for i in imgs ]

for i in range(len(fs)):

pred = predict(fs[i])

fig = plt.figure(figsize = (16,8))

ax = fig.add\_subplot(121)

ax.imshow(imgs[i])

ax = fig.add\_subplot(122)

ax.imshow(pred)

plt.title(pred.sum())

plt.show()

效果：

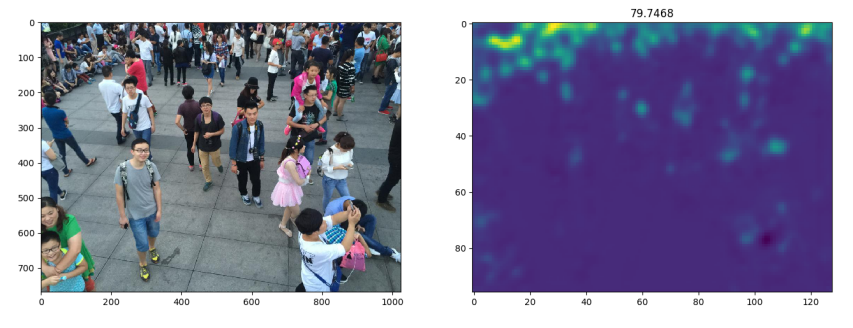


图 5 IMG\_8.jpg的效果示意图

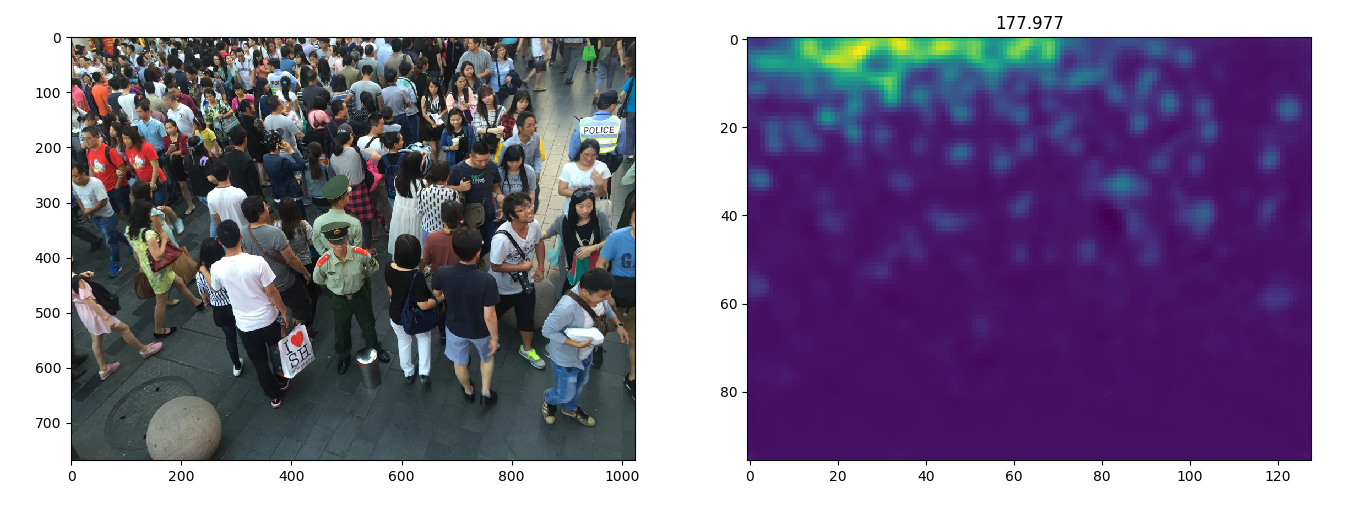


图 6 IMG\_9.jpg的效果示意图

# 总结

模型具有优异的效果，但是训练阶段对硬件要求过高。为了降低硬件要求，我尝试了将模型改小，但是改小后的two-scale model模型欠拟合严重，无法得到应有的效果，为了找到其中难以训练的问题，我编写了一个检查梯度的代码，这些将放置于附录中。总之，此次模型训练带来的极大困难，致使本次编写过程有十分多的不理想之处，十分抱歉。最后是，这个模型在caffe框架中依然不能达到实时的需求，但是在秒级别依然是能够完成运算的，依然能够满足应用需求，但caffe——cpu框架的内存占用过高，在移植时可以另行考虑依赖框架。故而附录也将给出我对于将caffe的参数文件转为npy文件以供tensorflow使用的尝试。

# 参考文献

[1] 刘辉, 朱闯, 张天永, 等. 一种基于头部特征的人头检测方法[J]. 光电子技术, 2014 (1): 21-25.

[2] 卢湖川, 张明修, 张继霞, 等. 一种有效的实时人群计数方法[J]. 计算机工程, 2008, 34(5): 222-224.

[3] Zhang Y, Zhou D, Chen S, et al. Single-image crowd counting via multi-column convolutional neural network[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 589-597.

[4] Zhang L, Shi M, Chen Q. Crowd counting via scale-adaptive convolutional neural network[C]//2018 IEEE Winter Conference on Applications of Computer Vision (WACV). IEEE, 2018: 1113-1121.

[5] 陈仲铭，彭凌西. 深度学习原理与实践[M]. 北京：人民邮电出版社，2018.

# 附录

Caffemodel文件转npy文件(SaCNN\_caffemodel2npy.py):

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

#此代码用于将caffemodel文件转为npy以便tensorflow运行

import numpy as np

import caffe

import os

import sys

caffe\_root = 'D:\\Medium\\Codes\\Caffe\_build\\caffe/'

sys.path.insert(0, caffe\_root + 'python')

if os.path.isfile('ShanghaiTech\_part\_B.caffemodel'):

print('CaffeNet found.')

else:

print('CaffeNet Not found.')

caffe.set\_mode\_cpu()

model\_def = 'deploy.prototxt'

model\_weights = 'ShanghaiTech\_part\_B.caffemodel'

net = caffe.Net(model\_def,model\_weights,caffe.TEST)

key\_use = net.params.keys()

SaCNN\_model = {}

for layer\_name in key\_use:

layer\_data\_caffe = net.params[layer\_name][0].data

layer\_data\_caffe = np.swapaxes(layer\_data\_caffe,0,2)#[n\_out, n\_in, h, w] caffe

layer\_data\_caffe = np.swapaxes(layer\_data\_caffe,1,3)#[kh, kw, n\_in, n\_out] tf

layer\_data\_caffe = np.swapaxes(layer\_data\_caffe,2,3)

if 'conv\_concat' in layer\_name:

layer\_data\_caffe = np.swapaxes(layer\_data\_caffe,2,3)

print(layer\_data\_caffe.shape)

SaCNN\_model[layer\_name] = [layer\_data\_caffe]

else:

SaCNN\_model[layer\_name] = [layer\_data\_caffe,net.params[layer\_name][1].data]

np.save('sacnn\_tf',SaCNN\_model)

print('Model have been generated.')

由npy文件导入生成tensorflow模型(layers.py):

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

import tensorflow as tf

import numpy as np

net\_model = np.load('sacnn\_tf.npy',allow\_pickle = True).item()

layer\_names = ['conv1\_1', 'conv1\_2', 'conv2\_1',

'conv2\_2', 'conv3\_1', 'conv3\_2',

'conv3\_3', 'conv4\_1', 'conv4\_2',

'conv4\_3', 'conv5\_1', 'conv5\_2',

'conv5\_3', 'conv6\_1', 'conv\_concat1\_2x',

'p\_conv1', 'p\_conv2', 'p\_conv3']

def model\_conv(x,name,trainable = False,activation = tf.nn.relu):

with tf.variable\_scope(name):

if trainable :

gene\_fn = tf.Variable

else:

gene\_fn = tf.constant

y = tf.nn.bias\_add(

tf.nn.conv2d(x,gene\_fn(net\_model[name][0],dtype = tf.float32),

(1,1,1,1),padding = 'SAME',name = name),

gene\_fn(net\_model[name][1],dtype = tf.float32))

if activation is not None:

print(name,'activate')

print(name,'sum is ',net\_model[name][1].sum(),net\_model[name][1].max())

return activation(y)

else:

return y

def model\_pool(x,name):

with tf.variable\_scope(name):

return tf.nn.max\_pool(x,

ksize=[1,2, 2 , 1],

strides=[1, 2, 2, 1],

padding='SAME',

name=name)

def model\_deconv(input\_tensor,name,out\_shape ,dw=2,dh=2):

with tf.variable\_scope(name):

deconv\_ls = []

weights = tf.constant(np.ones((2,2,1,2)),dtype = tf.float32)

for i in range(input\_tensor.shape[3]):

tensor\_t = input\_tensor[:,:,:,i\*2:(i+1)\*2]

deconv\_t = tf.nn.conv2d\_transpose(tensor\_t,weights,out\_shape ,strides = (1,dh,dw,1),padding = 'SAME' )

deconv\_ls.append(deconv\_t)

if (i+1)\*2==input\_tensor.shape[3]:

break

return tf.concat(deconv\_ls,axis = 3)

def pool(input\_tensor, name, kh, kw, dh, dw):

return tf.nn.max\_pool(input\_tensor,

ksize=[1, kh, kw, 1],

strides=[1, dh, dw, 1],

padding='SAME',

name=name)

def fuse\_layer(x1, x2):

x\_concat = tf.concat([x1, x2],axis=3)

return x\_concat

采用tensorflow进行预测(sacnn\_tf\_predict.py):

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

import tensorflow as tf

import os

import numpy as np

import pylab as plt

import keras.backend.tensorflow\_backend as KTF

from SaCNN\_tf import \*

import time

from matplotlib import pyplot as plt

from PIL import Image

VGG\_MEAN = [103.939, 116.779, 123.68]

#VGG\_MEAN = [ 104.00698793, 116.66876762, 122.67891434]

img\_side\_max = 1200#177,79

def imgpreprocess(img):

if img.shape[0] > img\_side\_max or img.shape[1] > img\_side\_max:

img\_t = Image.fromarray(img)

if img.shape[0] == max(img.shape):

img\_t = img\_t.resize((int(img.shape[1]\*img\_side\_max/img.shape[0]),img\_side\_max))

img = np.array(img\_t)

else:

img\_t = img\_t.resize((img\_side\_max,int(img.shape[0]\*img\_side\_max/img.shape[1])))

img = np.array(img\_t)

img\_res = np.zeros(img.shape)

img\_res[:,:,0] = img[:,:,2]-VGG\_MEAN[0]

img\_res[:,:,1] = img[:,:,1]-VGG\_MEAN[1]

img\_res[:,:,2] = img[:,:,0]-VGG\_MEAN[2]

return np.expand\_dims(img\_res,axis = 0)

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

config = tf.ConfigProto()

config.gpu\_options.allow\_growth = True

input\_path = 'images/'

dmap\_path = 'dmaps/'

for i in [input\_path,dmap\_path]:

if not os.path.exists(i):

os.mkdir(i)

G = tf.Graph()

with G.as\_default():

image\_place\_holder = tf.placeholder(tf.float32, shape=[None, None, None, 3])

d\_map\_est = build(image\_place\_holder)

summary = tf.summary.merge\_all()

with tf.Session(graph=G,config=config) as sess:

KTF.set\_session(sess)

writer = tf.summary.FileWriter(os.path.join('logging'))

writer.add\_graph(sess.graph)

t\_old = time.time()

time.sleep(1)

while (time.time()-t\_old)>1:

t\_old = time.time()

imgs = os.listdir(input\_path)

res\_imgs = os.listdir(dmap\_path)

imgs = [input\_path+i for i in imgs if os.path.split(i)[-1] not in res\_imgs]

imgs\_net\_ls = [imgpreprocess(plt.imread(i)) for i in imgs]

for i in range(len(imgs\_net\_ls)):

imgs\_net = imgs\_net\_ls[i]

d\_map = sess.run(d\_map\_est,feed\_dict = {image\_place\_holder:imgs\_net})

res\_t = d\_map[0,:,:,0]

plt.imshow(res\_t)

plt.title(res\_t.sum())

plt.axis('off')

print(os.path.split(imgs[i])[-1])

plt.savefig(os.path.join(dmap\_path,os.path.split(imgs[i])[-1]))

np.save(os.path.join(dmap\_path,os.path.split(imgs[i])[-1]),res\_t)

plt.clf()

fig = plt.figure(figsize = (16,8))

ax = fig.add\_subplot(121)

ax.imshow(plt.imread(imgs[i]))

ax = fig.add\_subplot(122)

ax.imshow(res\_t)

plt.title(res\_t.sum())

plt.show()

梯度检查代码，grad\_check.py:

import tensorflow as tf

import src.tscnn as tscnn

import src.layers as L

import os

import src.utils as utils

import numpy as np

import matplotlib.image as mpimg

import scipy.io as sio

import time

import argparse

#import sys

#from PIL import Image

import pylab as plt

import keras.backend.tensorflow\_backend as KTF

#tf.device('/gpu:0')

# Global Constants. Define the number of images for training, validation and testing.

NUM\_TRAIN\_IMGS = 6000

NUM\_VAL\_IMGS = 590

NUM\_TEST\_IMGS = 587

MODEL\_SAVE\_PATH = './model/'

MODEL\_NAME = 'tscnn\_model'

batch\_size = 4

def main(args):

args.retrain = True

temp\_use\_path = os.path.join(args.log\_dir, 'session-'+str(args.session\_id))

if os.path.exists(temp\_use\_path):

session\_id\_list = [tempath.split('-')[-1] for tempath in os.listdir(args.log\_dir)]

session\_idl = [eval(comid) for comid in session\_id\_list]

session\_id\_t = max(session\_idl)

if args.retrain:

args.session\_id = session\_id\_t

#args.session\_id = 133

else:

pass#args.session\_id = session\_id\_t+1

sess\_path = utils.create\_session(args.log\_dir, args.session\_id) # Create a session path based on the session id.

if args.retrain:

model\_list = os.listdir(sess\_path)

model\_list = [i for i in model\_list if '.npz' in i]

model\_list = [eval(i.split('.')[1]) for i in model\_list]

model\_final = max(model\_list)

args.base\_model\_path = sess\_path+f'/weights.{model\_final}.npz'

config = tf.ConfigProto()

config.gpu\_options.allow\_growth = True

G = tf.Graph()

with G.as\_default():

# Create image and density map placeholder

image\_place\_holder = tf.placeholder(tf.float32, shape=[None, None, None, 3]) #原本是1，沙雕东西

d\_map\_place\_holder = tf.placeholder(tf.float32, shape=[None, None, None, 1])

# Build all nodes of the network

d\_map\_est,net\_mid = tscnn.build(image\_place\_holder) #为什么这里要将channel设置为1，不是rgb么，难道不是3通道吗

# Define the loss function.

with tf.variable\_scope('loss'):

euc\_loss = L.loss(d\_map\_est, d\_map\_place\_holder)

# Define the optimization algorithm

#optimizer = tf.train.GradientDescentOptimizer(args.learning\_rate)

optimizer= tf.train.AdamOptimizer(0.01)

#optimizer = tf.train.RMSPropOptimizer(100)

# Training node.

#train\_op = optimizer.minimize(euc\_loss)

#train\_grad = tf.gradients(euc\_loss,d\_map\_est)

train\_grad = tf.gradients(euc\_loss,net\_mid)

train\_grad = tf.reduce\_mean(train\_grad)

# Initialize all the variables.

init = tf.group(tf.global\_variables\_initializer(), tf.local\_variables\_initializer())

# For summary

summary = tf.summary.merge\_all()

with tf.Session(graph=G,config=config) as sess:

KTF.set\_session(sess)

writer = tf.summary.FileWriter(os.path.join(sess\_path,'training\_logging'))

writer.add\_graph(sess.graph)

sess.run(init) #完成初始化

if args.retrain:

utils.load\_weights(G, args.base\_model\_path)

# Start the epochs

for eph in range(args.num\_epochs):#训练的轮数

grads = []

# Get the list of train images.

train\_images\_list, train\_gts\_list = utils.get\_data\_list(args.data\_root, mode='train')

#total\_train\_loss = 0

# Loop through all the training images

train\_y,train\_x = [],[]

for img\_idx in range(len(train\_images\_list)):#每张图片 捞出来训练

# Load the image and ground truth

train\_image = np.asarray(mpimg.imread(train\_images\_list[img\_idx]), dtype=np.float32) #卧槽，读图像时直接读成float32啊啊啊啊

train\_image = utils.imgpreprocess(train\_image)

train\_d\_map = np.asarray(sio.loadmat(train\_gts\_list[img\_idx])['image\_info'][0][0][0][0][0], dtype=np.float32) #为了得到个数据，我是没见过比这更有病的

# Reshape the tensor before feeding it to the network #等价于np.expand\_dims

train\_image\_r = utils.reshape\_tensor(train\_image,3) #将代码修改为根据第二个参数进行设定第4维度

train\_d\_map\_r = utils.get\_density\_map\_gaussian(train\_d\_map,train\_image.shape[0],train\_image.shape[1]) #通过高斯卷积（模糊）得到密度图

#print('-----------------\n',train\_d\_map\_r.shape,'-----------------------')

train\_d\_map\_r = utils.img\_one2four(train\_d\_map\_r,img\_type='label')

train\_d\_map\_r = [utils.resize\_gt(i) for i in train\_d\_map\_r]#新增放缩算法

train\_d\_map\_r = [utils.reshape\_tensor(i,1) for i in train\_d\_map\_r]

if img\_idx%batch\_size!=0 or img\_idx == 0:

#图像切割，one2four，只有训练时需要这样，其他时候没必要

#这里切割的实现将交由utils进行

#train\_image\_1 = train\_image\_r[:,:384,:512,:]#.reshape((1,384,512,3))#这里得到的shape其实是[1,384,512,3]

#train\_image\_2 = train\_image\_r[:,:384,512:,:]#.reshape((1,384,512,3))

train\_x+=utils.img\_one2four(train\_image\_r)

train\_y+=train\_d\_map\_r

continue

train\_x+=utils.img\_one2four(train\_image\_r)

train\_y+=train\_d\_map\_r

train\_image\_r = np.concatenate(train\_x,axis = 0)

train\_d\_map\_r = np.concatenate(train\_y,axis = 0)

train\_x,train\_y = [],[]

#train\_d\_map\_r = utils.downsampling\_d\_map(train\_d\_map\_r,sess)

# Prepare feed\_dict

feed\_dict\_data = {

image\_place\_holder: train\_image\_r,

d\_map\_place\_holder: train\_d\_map\_r,

}

#输出loss使用的两个输出，查看形状

if img\_idx %1== 0:

grad\_t,net\_output = sess.run([train\_grad,net\_mid],feed\_dict = feed\_dict\_data)

grads.append(grad\_t)

print(net\_output.min(),net\_output.max(),net\_output.mean())

plt.figure()

if len(grads)<20:

plt.plot(grads)

else:

plt.plot(grads[15:])

plt.title('gradients')

plt.figure()

plt.imshow(net\_output[0,:,:,0])

plt.title('net\_mid')

plt.colorbar()

plt.figure()

plt.imshow(train\_d\_map\_r[0,:,:,0])

plt.title('train\_d\_map')

plt.colorbar()

plt.show()

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

parser = argparse.ArgumentParser()

parser.add\_argument('--retrain', default=False, type=bool)

parser.add\_argument('--base\_model\_path', default=None, type=str)

parser.add\_argument('--log\_dir', default = './logs', type=str)

parser.add\_argument('--num\_epochs', default = 200, type=int)

parser.add\_argument('--learning\_rate', default = 0.0001, type=float)

parser.add\_argument('--session\_id', default = 2, type=int)

parser.add\_argument('--data\_root', default='./data/comb\_dataset\_v3', type=str)

args = parser.parse\_args()

died\_num = 0

while True:

main(args)

print(f'宕机{died\_num}次')

died\_num += 1