

```

1  #Data Pre-Processing

2  #ROI – Preprocessing Generate_ROI.py
3  import os
4  os.environ["CUDA_DEVICE_ORDER"] = "PCI_BUS_ID"
5  os.environ["CUDA_VISIBLE_DEVICES"] = "0,1"
6  import cv2
7  import numpy as np
8  import scipy.io as sio
9  import scipy.misc
10 from keras.preprocessing import image
11 from skimage.transform import rotate, resize
12 from skimage.measure import label, regionprops
13 from time import time
14 from mnet_utils import pro_process, BW_img, disc_crop
15 import matplotlib.pyplot as plt
16 from skimage.io import imsave
17
18
19
20
21 import Model_DiscSeg as DiscModel
22
23 #ROI_size_list = [400, 500, 600, 700, 800]
24 ROI_size_list = [700]
25 DiscROI_size = 700
26 DiscSeg_size = 640 # input size to the disc detection model
27
28
29 train_data_type = '.png'
30 mask_data_type = '.png'
31
32 Original_vali_img_path =
33 '/DATA/charlie/AWC/data_original/drishti/test/image/'
34 Original_Mask_img_path = '/DATA/charlie/AWC/data_original/drishti/test/mask/'
35
36 Image_save_path = '/DATA/charlie/AWC/CADA_Tutorial_Image/Target_Test/image/'
37 MaskImage_save_path =
38 '/DATA/charlie/AWC/CADA_Tutorial_Image/Target_Test/mask/'
39
40 if not os.path.exists(Image_save_path):
41     os.makedirs(Image_save_path)
42
43 if not os.path.exists(MaskImage_save_path):
44     os.makedirs(MaskImage_save_path)
45
46 is_polar_coordinate = False # in MICCAI version, this is false.
47 is_only_image = False #for target domain training images we do not have
48 masks, then it is True
49
50 file_train_list = [file for file in os.listdir(Original_vali_img_path) if
51 file.lower().endswith(train_data_type)]
52 print(str(len(file_train_list)))
53
54

```

```

55 DiscSeg_model = DiscModel.DeepModel(size_set=DiscSeg_size)
56 DiscSeg_model.load_weights('Model_DiscSeg_pretrain.h5')
57
58
59 for lineIdx in range(0, len(file_train_list)):
60
61     #####Generate mask ROIs
62     #####
63
64
65
66     temp_txt = [elt.strip() for elt in file_train_list[lineIdx].split(',')]
67     nameLen = len(temp_txt[0])
68     # print(' Processing Img: ' + temp_txt[0])
69     # load image
70     org_img = np.asarray(image.load_img(Original_vali_img_path +
71 temp_txt[0]))
72     # plt.imshow(org_img)
73     # plt.title('org_img')
74     # plt.show()
75
76     # Disc region detection by U-Net
77     temp_org_img = resize(org_img, (DiscSeg_size, DiscSeg_size, 3))
78     # plt.imshow(temp_org_img)
79     # plt.title('temp_org_img')
80     # plt.show()
81
82
83
84     if is_only_image == False:
85         org_mask = np.asarray(image.load_img(Original_Mask_img_path +
86 temp_txt[0][:nameLen - 4] +
87 mask_data_type))[:, :, 0]
88         # plt.imshow(org_mask)
89         # plt.title('org_mask')
90         # plt.show()
91
92         org_disc = org_mask < 255
93         # plt.imshow(org_disc)
94         # plt.title('org_disc')
95         # plt.show()
96
97         org_cup = org_mask == 0
98         # plt.imshow(org_cup)
99         # plt.title('org_cup')
100        # plt.show()
101
102
103
104        temp_org_mask = resize(org_mask, (DiscSeg_size, DiscSeg_size))
105        # plt.imshow(temp_org_mask)
106        # plt.title('temp_org_mask')
107        # plt.show()
108
109        temp_org_disc = resize(org_disc, (DiscSeg_size, DiscSeg_size))
110        # plt.imshow(temp_org_disc)
111        # plt.title('temp_org_disc')

```

```

112         # plt.show()
113
114         temp_org_cup = resize(org_cup, (DiscSeg_size, DiscSeg_size))
115         # plt.imshow(temp_org_cup)
116         # plt.title('temp_org_cup')
117         # plt.show()
118         org_disc_bw = BW_img(np.reshape(temp_org_disc, (DiscSeg_size,
119 DiscSeg_size)), 0.5)
120         org_cup_bw = BW_img(np.reshape(temp_org_cup, (DiscSeg_size,
121 DiscSeg_size)), 0.5)
122
123
124         temp_org_img = np.reshape(temp_org_img, (1,) + temp_org_img.shape) * 255
125
126         prob_10 = DiscSeg_model.predict([temp_org_img])
127
128         # plt.imshow(np.squeeze(np.clip(prob_10*255,0,255).astype('uint8'))))
129         # plt.title('temp_img')
130         # plt.show()
131
132         org_img_disc_map = BW_img(np.reshape(prob_10, (DiscSeg_size,
133 DiscSeg_size)), 0.5)
134         regions = regionprops(label(org_img_disc_map))
135
136         C_x = int(regions[0].centroid[0] * org_img.shape[0] / DiscSeg_size)
137         C_y = int(regions[0].centroid[1] * org_img.shape[1] / DiscSeg_size)
138
139         for disc_idx, DiscROI_size in enumerate(ROI_size_list):
140
141             org_img_disc_region, err_coord, crop_coord = disc_crop(org_img,
142 DiscROI_size, C_x, C_y)
143             # plt.imshow(org_img_disc_region)
144             # plt.title('org_img_disc_region')
145             # plt.show()
146
147             if is_only_image == False:
148                 org_mask_region, err_coord, crop_coord = disc_crop(org_mask,
149 DiscROI_size, C_x, C_y)
150                 org_disc_region, err_coord_disc, crop_coord_disc =
151 disc_crop(org_disc, DiscROI_size, C_x, C_y)
152                 # plt.imshow(org_disc_region)
153                 # plt.title('org_disc_region')
154                 # plt.show()
155
156                 org_cup_region, err_coord_cup, crop_coord_cup =
157 disc_crop(org_cup, DiscROI_size, C_x, C_y)
158                 # plt.imshow(org_cup_region)
159                 # plt.title('org_cup_region')
160                 # plt.show()
161
162                 ROI_mask_result = np.array(BW_img(org_disc_region, 0.5),
163 dtype=int) + np.array(BW_img(org_cup_region, 0.5),
164 dtype=int)
165                 # plt.imshow(ROI_mask_result)
166                 # plt.title('ROI_mask_result')
167                 # plt.show()
168

```

```

169
170     ROI_mask_result = (255.0 / ROI_mask_result.max() *
171 (ROI_mask_result - ROI_mask_result.min()).astype(np.uint8)
172     ROI_mask_result[ROI_mask_result == 255] = 200
173     ROI_mask_result[ROI_mask_result == 0] = 255
174     ROI_mask_result[ROI_mask_result == 200] = 0
175     ROI_mask_result[(ROI_mask_result > 0) & (ROI_mask_result < 255)]
176 = 128
177     # plt.imshow(ROI_mask_result)
178     # plt.title('ROI_mask_result')
179     # plt.show()
180
181     if is_polar_coordinate:
182
183         if is_only_image == False:
184             ROI_mask_result = rotate(cv2.linearPolar(ROI_mask_result,
185 (DiscROI_size / 2, DiscROI_size / 2), DiscROI_size / 2,
186                                     cv2.INTER_NEAREST +
187 cv2.WARP_FILL_OUTLIERS), -90)
188             # plt.imshow(ROI_mask_result)
189             # plt.title('ROI_mask_result')
190             # plt.show()
191
192
193             ROI_img_result = rotate(cv2.linearPolar(org_img_disc_region,
194 (DiscROI_size / 2, DiscROI_size / 2), DiscROI_size / 2,
195                                     cv2.INTER_NEAREST +
196 cv2.WARP_FILL_OUTLIERS), -90)
197         else:
198             ROI_img_result = org_img_disc_region
199
200
201         # filename_ROI_Img = '{}_{}_{}'.format(temp_txt[0][:nameLen - 4],
202 DiscROI_size, train_data_type)
203         filename_ROI_Img = '{}_{}'.format(temp_txt[0][:nameLen - 4], '.jpg')
204         imsave(Image_save_path + filename_ROI_Img, ROI_img_result)
205         if is_only_image == False:
206             filename_ROI_Mask = '{}_{}'.format(temp_txt[0][:nameLen - 4],
207 '.bmp')
208             imsave(MaskImage_save_path + filename_ROI_Mask, ROI_mask_result)
209
210         #####Generate mask ROI done
211         #####
212
213 #M-NET Utils - mnet_utils.py
214 #The code is modified from https://github.com/HzFu/MNet_DeepCDR
215 # -*- coding: utf-8 -*-
216
217 from __future__ import print_function
218
219 import os
220
221 import numpy as np
222 from PIL import Image
223 from scipy.ndimage import binary_fill_holes

```

```

224 from skimage.measure import label, regionprops
225 from tensorflow.python.keras import backend as K
226 from tensorflow.python.keras.preprocessing import image
227
228
229 def pro_process(temp_img, input_size):
230     img = np.asarray(temp_img*255).astype('uint8')
231     img = np.array(Image.fromarray(img).resize((input_size, input_size)))
232     return img
233
234
235 def train_loader(data_list, data_path, mask_path, input_size):
236     while 1:
237         for lineIdx, temp_txt in enumerate(data_list):
238             train_img = np.asarray(image.load_img(os.path.join(data_path,
239 temp_txt),
240                                                     target_size=(input_size,
241 input_size, 3))
242                                     ).astype('float32')
243             img_mask = np.asarray(
244                 image.load_img(os.path.join(mask_path, temp_txt),
245                                     target_size=(input_size, input_size, 3))
246                 ) / 255.0
247
248             train_img = np.reshape(train_img, (1,) + train_img.shape)
249             img_mask = np.reshape(img_mask, (1,) + img_mask.shape)
250             yield ([train_img], [img_mask, img_mask, img_mask, img_mask,
251 img_mask])
252
253
254 def BW_img(input, thresholding):
255     if input.max() > thresholding:
256         binary = input > thresholding
257     else:
258         binary = input > input.max() / 2.0
259
260     label_image = label(binary)
261     regions = regionprops(label_image)
262     area_list = [region.area for region in regions]
263     if area_list:
264         idx_max = np.argmax(area_list)
265         binary[label_image != idx_max + 1] = 0
266     return binary_fill_holes(np.asarray(binary).astype(int))
267
268
269 def dice_coef(y_true, y_pred):
270     smooth = 1.
271     y_true_f = K.flatten(y_true)
272     y_pred_f = K.flatten(y_pred)
273     intersection = K.sum(y_true_f * y_pred_f)
274     return 1- (2. * intersection + smooth) / (K.sum(y_true_f) +
275 K.sum(y_pred_f) + smooth)
276
277
278 def dice_coef2(y_true, y_pred):
279     score0 = dice_coef(y_true[:, :, :, 0], y_pred[:, :, :, 0])
280     score1 = dice_coef(y_true[:, :, :, 1], y_pred[:, :, :, 1])

```

```

281     score = 0.5 * score0 + 0.5 * score1
282
283     return score
284
285
286 def dice_coef_loss(y_true, y_pred):
287     return dice_coef2(y_true, y_pred)
288
289
290
291 def disc_crop(org_img, DiscROI_size, C_x, C_y):
292     tmp_size = int(DiscROI_size / 2);
293     if len(org_img.shape) == 2:
294         disc_region = np.zeros((DiscROI_size, DiscROI_size),
295 dtype=org_img.dtype)
296     else:
297         disc_region = np.zeros((DiscROI_size, DiscROI_size, 3),
298 dtype=org_img.dtype)
299
300     crop_coord = np.array([C_x - tmp_size, C_x + tmp_size, C_y - tmp_size,
301 C_y + tmp_size], dtype=int)
302     err_coord = [0, DiscROI_size, 0, DiscROI_size]
303
304     if crop_coord[0] < 0:
305         err_coord[0] = abs(crop_coord[0])
306         crop_coord[0] = 0
307
308     if crop_coord[2] < 0:
309         err_coord[2] = abs(crop_coord[2])
310         crop_coord[2] = 0
311
312     if crop_coord[1] > org_img.shape[0]:
313         err_coord[1] = err_coord[1] - (crop_coord[1] - org_img.shape[0])
314         crop_coord[1] = org_img.shape[0]
315
316     if crop_coord[3] > org_img.shape[1]:
317         err_coord[3] = err_coord[3] - (crop_coord[3] - org_img.shape[1])
318         crop_coord[3] = org_img.shape[1]
319     if len(org_img.shape) == 2:
320         disc_region[err_coord[0]:err_coord[1], err_coord[2]:err_coord[3]] =
321 org_img[crop_coord[0]:crop_coord[1],
322
323 crop_coord[2]:crop_coord[3]]
324     else:
325         disc_region[err_coord[0]:err_coord[1], err_coord[2]:err_coord[3], ] =
326 org_img[crop_coord[0]:crop_coord[1],
327
328 crop_coord[2]:crop_coord[3], ]
329
330     return disc_region, err_coord, crop_coord
331
332
333 def mk_dir(dir_path):
334     if not os.path.exists(dir_path):
335         os.makedirs(dir_path)
336     return dir_path
337

```

```

338
339 def files_with_ext(data_path, data_type):
340     file_list = [file for file in os.listdir(data_path) if
341 file.lower().endswith(data_type)]
342     print(len(file_list))
343     return file_list
344
345 #Disc Seg Model – Model_DiscSeg.py
346 #The code is modified from https://github.com/HzFu/MNet_DeepCDR
347 # -*- coding: utf-8 -*-
348
349 from __future__ import absolute_import
350 from __future__ import print_function
351
352 from tensorflow.python.keras.layers import (Input, concatenate, Conv2D,
353 MaxPooling2D,
354                                         Conv2DTranspose, UpSampling2D,
355 average)
356 from tensorflow.python.keras.models import Model
357
358
359 def DeepModel(size_set=640):
360     img_input = Input(shape=(size_set, size_set, 3))
361
362     conv1 = Conv2D(32, (3, 3), activation='relu', padding='same',
363 name='block1_conv1')(img_input)
364     conv1 = Conv2D(32, (3, 3), activation='relu', padding='same',
365 name='block1_conv2')(conv1)
366     pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
367
368     conv2 = Conv2D(64, (3, 3), activation='relu', padding='same',
369 name='block2_conv1')(pool1)
370     conv2 = Conv2D(64, (3, 3), activation='relu', padding='same',
371 name='block2_conv2')(conv2)
372     pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
373
374     conv3 = Conv2D(128, (3, 3), activation='relu', padding='same',
375 name='block3_conv1')(pool2)
376     conv3 = Conv2D(128, (3, 3), activation='relu', padding='same',
377 name='block3_conv2')(conv3)
378     pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)
379
380     conv4 = Conv2D(256, (3, 3), activation='relu', padding='same',
381 name='block4_conv1')(pool3)
382     conv4 = Conv2D(256, (3, 3), activation='relu', padding='same',
383 name='block4_conv2')(conv4)
384     pool4 = MaxPooling2D(pool_size=(2, 2))(conv4)
385
386     conv5 = Conv2D(512, (3, 3), activation='relu', padding='same',
387 name='block5_conv1')(pool4)
388     conv5 = Conv2D(512, (3, 3), activation='relu', padding='same',
389 name='block5_conv2')(conv5)
390
391     up6 = concatenate(
392         [Conv2DTranspose(256, (2, 2), strides=(2, 2), padding='same',

```

```

393     name='block6_dconv')(conv5), conv4],
394         axis=3)
395     conv6 = Conv2D(256, (3, 3), activation='relu', padding='same',
396     name='block6_conv1')(up6)
397     conv6 = Conv2D(256, (3, 3), activation='relu', padding='same',
398     name='block6_conv2')(conv6)
399
400     up7 = concatenate(
401         [Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same',
402     name='block7_dconv')(conv6), conv3],
403         axis=3)
404     conv7 = Conv2D(128, (3, 3), activation='relu', padding='same',
405     name='block7_conv1')(up7)
406     conv7 = Conv2D(128, (3, 3), activation='relu', padding='same',
407     name='block7_conv2')(conv7)
408
409     up8 = concatenate([Conv2DTranspose(64, (2, 2), strides=(2, 2),
410     padding='same', name='block8_dconv')(conv7), conv2],
411         axis=3)
412     conv8 = Conv2D(64, (3, 3), activation='relu', padding='same',
413     name='block8_conv1')(up8)
414     conv8 = Conv2D(64, (3, 3), activation='relu', padding='same',
415     name='block8_conv2')(conv8)
416
417     up9 = concatenate([Conv2DTranspose(32, (2, 2), strides=(2, 2),
418     padding='same', name='block9_dconv')(conv8), conv1],
419         axis=3)
420     conv9 = Conv2D(32, (3, 3), activation='relu', padding='same',
421     name='block9_conv1')(up9)
422     conv9 = Conv2D(32, (3, 3), activation='relu', padding='same',
423     name='block9_conv2')(conv9)
424
425     side6 = UpSampling2D(size=(8, 8))(conv6)
426     side7 = UpSampling2D(size=(4, 4))(conv7)
427     side8 = UpSampling2D(size=(2, 2))(conv8)
428     out6 = Conv2D(1, (1, 1), activation='sigmoid', name='side_6')(side6)
429     out7 = Conv2D(1, (1, 1), activation='sigmoid', name='side_7')(side7)
430     out8 = Conv2D(1, (1, 1), activation='sigmoid', name='side_8')(side8)
431     out9 = Conv2D(1, (1, 1), activation='sigmoid', name='side_9')(conv9)
432
433     out10 = average([out6, out7, out8, out9])
434     # out10 = Conv2D(1, (1, 1), activation='sigmoid', name='side_10')(out10)
435
436     return Model(inputs=[img_input], outputs=[out10])
437
438 #Data Loader
439
440 #Data Util – refuge.py
441
442 """Dataset setting and data loader for USPS.
443
444 Modified from
445 https://github.com/mingyuliutw/CoGAN/blob/master/cogan\_pytorch/src/dataset\_usps.py.
446
447 use the test data as target data

```



```

447 added by PengLiu 12/08/2018
448 """
449
450 import os
451 import numpy as np
452 import scipy.misc as m
453 from matplotlib.pyplot import imread
454 import cv2
455 import glob
456 import torch
457 import torch.utils.data as data
458
459
460 src_image_dir = '/DATA/charlie/AWC/CADA_Tutorial_Image/Source/image/'
461 src_mask_dir = '/DATA/charlie/AWC/CADA_Tutorial_Image/Source/mask/'
462
463 tgt_image_dir = '/DATA/charlie/AWC/CADA_Tutorial_Image/Target_Train/image/'
464
465 test_image_dir = '/DATA/charlie/AWC/CADA_Tutorial_Image/Target_Test/image/'
466 test_mask_dir = '/DATA/charlie/AWC/CADA_Tutorial_Image/Target_Test/mask/'
467
468
469 class REFUGE(data.Dataset):
470     """REFUGE Dataset.
471     Args:
472         root (string): Root directory of dataset where dataset file exist.
473         train (bool, optional): If True, resample from dataset randomly.
474         download (bool, optional): If true, downloads the dataset
475             from the internet and puts it in root directory.
476             If dataset is already downloaded, it is not downloaded again.
477         transform (callable, optional): A function/transform that takes in
478             an PIL image and returns a transformed version.
479             E.g, ``transforms.RandomCrop``
480     """
481
482     def __init__(
483         self,
484         train=True,
485         domain='REFUGE_SRC',
486         is_transform=False,
487         augmentations=None,
488         aug_for_target=None,
489         img_size=(400, 400),
490         max_iters=None
491     ):
492         self.train = train
493         self.is_transform = is_transform
494         self.augmentations = augmentations
495         self.aug_for_target = aug_for_target
496         self.dataset_size = None
497
498         self.domain = domain
499         if domain == 'REFUGE_SRC':
500             self.img_dir = src_image_dir
501             self.mask_dir = src_mask_dir
502         if domain == 'REFUGE_DST':
503             self.img_dir = tgt_image_dir

```

```

504     if domain == 'REFUGE_TEST':
505         self.img_dir = test_image_dir
506         self.mask_dir = test_mask_dir
507     self.class_map = {0: 2, 128: 1, 255: 0}
508     self.img_size = (
509         img_size if isinstance(img_size, tuple) else (img_size, img_size)
510     )
511
512     self._glob_img_files()
513     if not max_iters == None:
514         self.image_files = self.image_files * int(
515             np.ceil(float(max_iters) / len(self.image_files)))
516
517     def __getitem__(self, index):
518         """Get images and target for data loader.
519         Args:
520             index (int): Index
521         Returns:
522             tuple: (image, target) where target is index of the target class.
523         """
524         image_file = self.image_files[index]
525         img = imread(image_file)
526         img = np.array(img, dtype=np.uint8)
527
528         if self.domain != 'REFUGE_DST':
529             label_file = os.path.join(self.mask_dir,
530                                     os.path.basename(image_file)[:3] +
531                                     'bmp')
532             lbl_ori = imread(label_file)
533             lbl = lbl_ori.copy()
534             lbl = cv2.resize(lbl, (self.img_size[0],
535 self.img_size[1]), interpolation=cv2.INTER_NEAREST)
536             lbl = self.encode_segmap(np.array(lbl, dtype=np.uint8))
537
538         else:
539             lbl = np.zeros((self.img_size[0], self.img_size[1]),
540 dtype=np.uint8)
541
542         if self.augmentations is not None:
543
544             # data augmentation for student
545             aug = self.augmentations(image=img, mask=lbl)
546             img, lbl = aug['image'], aug['mask']
547
548             # data augmentation for teacher
549             aug = self.aug_for_target(image=img, mask=lbl)
550             img0, lbl0 = aug['image'], aug['mask']
551
552         else:
553             img0, lbl0 = img.copy(), lbl.copy()
554
555
556         if self.is_transform:
557             img, lbl = self.transform(img, lbl)
558             img0, lbl0 = self.transform(img0, lbl0)
559
560         return img, lbl, img0, lbl0, os.path.basename(image_file)[:4]

```

```

561
562 def __len__(self):
563     """Return size of dataset."""
564     return len(self.image_files)
565
566 def _glob_img_files(self):
567     """Check if dataset is download and in right place."""
568     if self.domain == 'REFUGE_SRC':
569         self.image_files = glob.glob(os.path.join(self.img_dir, '*.jpg'))
570     if self.domain == 'REFUGE_DST':
571         self.image_files = glob.glob(os.path.join(self.img_dir, '*.jpg'))
572     if self.domain == 'REFUGE_TEST':
573         self.image_files = glob.glob(os.path.join(self.img_dir, '*.jpg'))
574
575
576 def transform(self, img, lbl):
577     """transform
578     :param img:
579     :param lbl:
580     """
581     img = m.imresize(
582         img, (self.img_size[0], self.img_size[1])
583     ) # uint8 with RGB mode
584     img = img[:, :, ::-1] # RGB -> BGR
585     img = img.astype(np.float64)
586
587     img = img.transpose(2, 0, 1)
588
589     classes = np.unique(lbl)
590     lbl = lbl.astype(float)
591     lbl = cv2.resize(lbl, (self.img_size[0], self.img_size[1]),
592 interpolation=cv2.INTER_NEAREST)
593     lbl = lbl.astype(int)
594
595     if not np.all(classes == np.unique(lbl)):
596         print("WARN: resizing labels yielded fewer classes")
597
598     img = torch.from_numpy(img).float()
599     lbl = torch.from_numpy(lbl).long()
600
601     return img, lbl
602
603
604 def encode_segmap(self, mask):
605     # Put all void classes to zero
606     classes = np.unique(mask)
607     for each_class in classes:
608         assert each_class in self.class_map.keys()
609
610     for _validc in self.class_map.keys():
611         mask[mask == _validc] = self.class_map[_validc]
612     return mask

```

```

613 #Evaluation

614 #Evaluation Metric Calculation – evaluation_segmentation.py
615 import numpy as np
616 import matplotlib.pyplot as plt
617
618 from scipy import misc
619 from os import path, makedirs
620 import glob
621 from evaluation.file_management import
622 save_csv_mean_segmentation_performance, save_csv_segmentation_table
623 #from file_management import save_csv_mean_segmentation_performance,
624 save_csv_segmentation_table
625
626 EPS = 1e-7
627
628
629 def dice_coefficient(binary_segmentation, binary_gt_label):
630     """
631     Compute the Dice coefficient between two binary segmentation.
632     Dice coefficient is defined as here:
633     https://en.wikipedia.org/wiki/S%C3%B8rensen%E2%80%93Dice_coefficient
634     Input:
635         binary_segmentation: binary 2D numpy array representing the region of
636         interest as segmented by the algorithm
637         binary_gt_label: binary 2D numpy array representing the region of
638         interest as provided in the database
639     Output:
640         dice_value: Dice coefficient between the segmentation and the ground
641         truth
642     """
643
644     # turn all variables to booleans, just in case
645     binary_segmentation = np.asarray(binary_segmentation, dtype=np.bool)
646     binary_gt_label = np.asarray(binary_gt_label, dtype=np.bool)
647
648     # compute the intersection
649     intersection = np.logical_and(binary_segmentation, binary_gt_label)
650
651     # count the number of True pixels in the binary segmentation
652     segmentation_pixels = float(np.sum(binary_segmentation.flatten()))
653     # same for the ground truth
654     gt_label_pixels = float(np.sum(binary_gt_label.flatten()))
655     # same for the intersection
656     intersection = float(np.sum(intersection.flatten()))
657
658     # compute the Dice coefficient
659     dice_value = 2 * intersection / (segmentation_pixels + gt_label_pixels)
660
661     return dice_value
662
663
664 def vertical_diameter(binary_segmentation):
665     """
666     Get the vertical diameter from a binary segmentation.
667     The vertical diameter is defined as the "fattest" area of the

```

```

668 binary_segmentation parameter.
669     Input:
670         binary_segmentation: a boolean 2D numpy array representing a region
671 of interest.
672     Output:
673         diameter: the vertical diameter of the structure, defined as the
674 largest diameter between the upper and the lower interfaces
675     """
676
677     # turn the variable to boolean, just in case
678     binary_segmentation = np.asarray(binary_segmentation, dtype=np.bool)
679
680     # get the sum of the pixels in the vertical axis
681     vertical_axis_diameter = np.sum(binary_segmentation, axis=0)
682
683     # pick the maximum value
684     diameter = np.max(vertical_axis_diameter)
685
686     # return it
687     return float(diameter)
688
689
690 def vertical_cup_to_disc_ratio(segmentation):
691     """
692     Compute the vertical cup-to-disc ratio from a given labelling map.
693     The vertical cup to disc ratio is defined as here:
694     https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1722393/pdf/v082p01118.pdf
695     Input:
696         segmentation: binary 2D numpy array representing a segmentation, with
697 0: optic cup, 128: optic disc, 255: elsewhere.
698     Output:
699         cdr: vertical cup to disc ratio
700     """
701
702     # compute the cup diameter
703     cup_diameter = vertical_diameter(segmentation == 0)
704     # compute the disc diameter
705     disc_diameter = vertical_diameter(segmentation < 255)
706
707     return cup_diameter / (disc_diameter + EPS)
708
709
710 def absolute_error(predicted, reference):
711     """
712     Compute the absolute error between a predicted and a reference outcomes.
713     Input:
714         predicted: a float value representing a predicted outcome
715         reference: a float value representing the reference outcome
716     Output:
717         abs_err: the absolute difference between predicted and reference
718     """
719
720     return abs(predicted - reference)
721
722
723 def evaluate_binary_segmentation(segmentation, gt_label):
724     """

```

```

725     Compute the evaluation metrics of the REFUGE challenge by comparing the
726     segmentation with the ground truth
727     Input:
728         segmentation: binary 2D numpy array representing the segmentation,
729     with 0: optic cup, 128: optic disc, 255: elsewhere.
730         gt_label: binary 2D numpy array representing the ground truth
731     annotation, with the same format
732     Output:
733         cup_dice: Dice coefficient for the optic cup
734         disc_dice: Dice coefficient for the optic disc
735         cdr: absolute error between the vertical cup to disc ratio as
736     estimated from the segmentation vs. the gt_label, in pixels
737     """
738
739     # compute the Dice coefficient for the optic cup
740     # plt.imshow(segmentation==0)
741     # plt.title('cup binary_segmentation')
742     # plt.show()
743     # plt.imshow(gt_label==0)
744     # plt.title('cup binary_gt_label')
745     # plt.show()
746     cup_dice = dice_coefficient(segmentation == 0, gt_label == 0)
747     # compute the Dice coefficient for the optic disc
748     # plt.imshow(segmentation<255)
749     # plt.title('disc binary_segmentation')
750     # plt.show()
751     # plt.imshow(gt_label<255)
752     # plt.title('disc binary_gt_label')
753     # plt.show()
754     disc_dice = dice_coefficient(segmentation < 255, gt_label < 255)
755     # compute the absolute error between the cup to disc ratio estimated from
756     the segmentation vs. the gt label
757     cdr = absolute_error(vertical_cup_to_disc_ratio(segmentation),
758     vertical_cup_to_disc_ratio(gt_label))
759
760     return cup_dice, disc_dice, cdr
761
762
763 def generate_table_of_results(image_filenames, gt_folder, is_training=False):
764     """
765     Generates a table with image_filename, cup_dice, disc_dice and cdr values
766     Input:
767         image_filenames: a list of strings with the names of the images.
768         segmentation_folder: a string representing the full path to the
769     folder where the segmentation files are
770         gt_folder: a string representing the full path to the folder where
771     the ground truth annotation files are
772         is_training: a boolean value indicating if the evaluation is
773     performed on training data or not
774     Output:
775         image_filenames: same as the input parameter
776         cup_dices: a numpy array with the same length than the
777     image_filenames list, with the Dice coefficient for each optic cup
778         disc_dices: a numpy array with the same length than the
779     image_filenames list, with the Dice coefficient for each optic disc
780         ae_cdrs: a numpy array with the same length than the image_filenames
781     list, with the absolute error of the vertical cup to disc ratio

```

```

782     """
783
784     # initialize an array for the Dice coefficients of the optic cups
785     cup_dices = np.zeros(len(image_filenames), dtype=np.float)
786     # initialize an array for the Dice coefficients of the optic discs
787     disc_dices = np.zeros(len(image_filenames), dtype=np.float)
788     # initialize an array for the absolute errors of the vertical cup to disc
789 ratios
790     ae_cdrs = np.zeros(len(image_filenames), dtype=np.float)
791
792     # iterate for each image filename
793     for i in range(len(image_filenames)):
794
795         # read the segmentation
796         segmentation = misc.imread(image_filenames[i])
797         if False:
798             plt.imshow(segmentation)
799             plt.title('segmentation')
800             plt.show()
801         if len(segmentation.shape) > 2:
802             segmentation = segmentation[:, :, 0]
803         # read the gt
804         if is_training:
805             gt_filename = path.join(gt_folder, 'Glaucoma',
806 path.basename(image_filenames[i]))
807             if path.exists(gt_filename):
808                 gt_label = misc.imread(gt_filename)
809             else:
810                 gt_filename = path.join(gt_folder, 'Non-Glaucoma',
811 path.basename(image_filenames[i]))
812                 if path.exists(gt_filename):
813                     gt_label = misc.imread(gt_filename)
814                 else:
815                     raise ValueError(
816                         'Unable to find {} in your training folder. Make sure
817 that you have the folder organized as provided in our website.'.format(
818                             image_filenames[i]))
819             else:
820                 gt_filename = path.join(gt_folder,
821 path.basename(image_filenames[i]))
822                 if path.exists(gt_filename):
823                     gt_label = misc.imread(gt_filename)
824
825                 if False:
826                     plt.title('gt_label')
827                     plt.imshow(gt_label)
828                     plt.show()
829             else:
830                 raise ValueError(
831                     'Unable to find {} in your ground truth folder. If you
832 are using training data, make sure to use the parameter is_training in
833 True.'.format(
834                         image_filenames[i]))
835
836         # evaluate the results and assign to the corresponding row in the
837 table
838         cup_dices[i], disc_dices[i], ae_cdrs[i] =

```

```

839 evaluate_binary_segmentation(segmentation, gt_label)
840     print('cup_dices:', cup_dices[i], '---disc_dices:', disc_dices[i], '-
841 -ae_cdrs:', ae_cdrs[i])
842
843     # return the columns of the table
844     return image_filenames, cup_dices, disc_dices, ae_cdrs
845
846
847 def get_mean_values_from_table(cup_dices, disc_dices, ae_cdrs):
848     """
849     Compute the mean evaluation metrics for the segmentation task.
850     Input:
851         cup_dices: a numpy array with the same length than the
852         image_filenames list, with the Dice coefficient for each optic cup
853         disc_dices: a numpy array with the same length than the
854         image_filenames list, with the Dice coefficient for each optic disc
855         ae_cdrs: a numpy array with the same length than the image_filenames
856         list, with the absolute error of the vertical cup to disc ratio
857     Output:
858         mean_cup_dice: the mean Dice coefficient for the optic cups
859         mean_disc_dice: the mean Dice coefficient for the optic disc
860         mae_cdr: the mean absolute error for the vertical cup to disc ratio
861     """
862
863     # compute the mean values of each column
864     mean_cup_dice = np.mean(cup_dices)
865     mean_disc_dice = np.mean(disc_dices)
866     mae_cdr = np.mean(ae_cdrs)
867
868     return mean_cup_dice, mean_disc_dice, mae_cdr
869
870
871 def evaluate_segmentation_results(segmentation_folder, gt_folder,
872 output_path=None, export_table=False,
873 is_training=False):
874     """
875     Evaluate the segmentation results of a single submission
876     Input:
877         segmentation_folder: full path to the segmentation files
878         gt_folder: full path to the ground truth files
879         [output_path]: a folder where the results will be saved. If not
880         provided, the results are not saved
881         [export_table]: a boolean value indicating if the table will be
882         exported or not
883         [is_training]: a boolean value indicating if the evaluation is
884         performed on training data or not
885     Output:
886         mean_cup_dice: the mean Dice coefficient for the optic cups
887         mean_disc_dice: the mean Dice coefficient for the optic disc
888         mae_cdr: the mean absolute error for the vertical cup to disc ratio
889     """
890
891     # get all the image filenames
892     image_filenames = glob.glob(path.join(segmentation_folder, '*.bmp')) #
893     get_filenames(segmentation_folder, 'bmp')
894     if len(image_filenames) == 0:
895         print(

```



```

896         '** The segmentation folder does not include any bmp file. Check
897 the files extension and resubmit your results.')
898         raise ValueError()
899         # create output path if it does not exist
900         if not (output_path is None) and not (path.exists(output_path)):
901             makedirs(output_path)
902
903         # generate a table of results
904         _, cup_dices, disc_dices, ae_cdrs =
905 generate_table_of_results(image_filenames, gt_folder, is_training)
906         # if we need to save the table
907         if not (output_path is None) and (export_table):
908             # initialize the table filename
909             table_filename = path.join(output_path,
910 'evaluation_table_segmentation.csv')
911             # save the table
912             save_csv_segmentation_table(table_filename, image_filenames,
913 cup_dices, disc_dices, ae_cdrs)
914
915             # compute the mean values
916             mean_cup_dice, mean_disc_dice, mae_cdr =
917 get_mean_values_from_table(cup_dices, disc_dices, ae_cdrs)
918             # print the results on screen
919             print('Dice Optic Cup = {} \n Dice Optic Disc = {} \n MAE CDR =
920 {}'.format(str(mean_cup_dice), str(mean_disc_dice),
921 str(mae_cdr)))
922             # save the mean values in the output path
923             if not (output_path is None):
924                 # initialize the output filename
925                 output_filename = path.join(output_path,
926 'evaluation_segmentation.csv')
927                 # save the results
928                 save_csv_mean_segmentation_performance(output_filename,
929 mean_cup_dice, mean_disc_dice, mae_cdr)
930
931                 # return the average performance
932                 return mean_cup_dice, mean_disc_dice, mae_cdr
933
934
935 if __name__ == '__main__':
936     results_folder = '../.. /data/result_UNet120000_v22_refineNet/'
937     gt_folder = '../.. /data/valiMaskImage_save_path_460/'
938     output_path = results_folder
939     export_table = True
940     evaluate_segmentation_results(results_folder, gt_folder, output_path,
941 export_table)
942
943
944 #File-Management Utilities – file_management.py
945
946 import csv
947 import numpy as np
948
949 from scipy.io import savemat

```

```

950 from os import listdir, path, makedirs
951
952
953 def parse_boolean(input_string):
954     '''
955     Parse a string as a boolean
956     '''
957     return input_string.upper() == 'TRUE'
958
959
960 def get_filenames(path_to_files, extension):
961     '''
962     Get all the files on a given folder with the given extension
963
964     Input:
965         path_to_files: string to a path where the files are
966         [extension]: string representing the extension of the files
967     Output:
968         image_filenames: a list of strings with the filenames in the folder
969     '''
970
971     # initialize a list of image filenames
972     image_filenames = []
973     # add to this list only those filenames with the corresponding extension
974     for file in listdir(path_to_files):
975         if file.endswith('.' + extension):
976             image_filenames = image_filenames + [file]
977
978     return image_filenames
979
980
981 def read_csv_classification_results(csv_filename):
982     '''
983     Read a two-column CSV file that has the classification results inside.
984
985     Input:
986         csv_filename: full path and filename to a two column CSV file with
987     the classification results (image filename, score)
988     Output:
989         image_filenames: list of image filenames, as retrieved from the first
990     column of the CSV file
991         scores: numpy array of floats, as retrieved from the second column of
992     the CSV file
993     '''
994
995     # initialize the output variables
996     image_filenames = []
997     scores = []
998
999     # open the file
1000     with open(csv_filename, 'r') as csv_file:
1001         # initialize a reader
1002         csv_reader = csv.reader(csv_file)
1003         # ignore the first row, that only has the header
1004         next(csv_reader)
1005         # and now, iterate and fill the arrays
1006         for row in csv_reader:

```

```

1007         image_filenames = image_filenames + [row[0]]
1008         scores = scores + [float(row[1])]
1009
1010     # turn the list of scores into a numpy array
1011     scores = np.asarray(scores, dtype=np.float)
1012
1013     # return the image filenames and the scores
1014     return image_filenames, scores
1015
1016
1017 def sort_scores_by_filename(target_names, names_to_sort, values_to_sort):
1018     '''
1019     This function is intended to correct the ordering in the outputs, just in
1020     case...
1021
1022     Input:
1023         target_names: a list of names sorted in the order that we want
1024         names_to_sort: a list of names to sort
1025         values_to_sort: a numpy array of values to sort
1026     Output:
1027         sorted_values: same array than values_to_sort, but this time sorted
1028     :)
1029     '''
1030
1031     names_to_sort = [x.upper() for x in names_to_sort]
1032
1033     # initialize the array of sorted values
1034     sorted_values = np.zeros(values_to_sort.shape)
1035
1036     # iterate for each filename in the target names
1037     for i in range(len(target_names)):
1038         # assign the value to the correct position in the array
1039         sorted_values[i] =
1040 values_to_sort[names_to_sort.index(target_names[i].upper())]
1041
1042     # return the sorted values
1043     return sorted_values
1044
1045
1046 def sort_coordinates_by_filename(target_names, names_to_sort,
1047 values_to_sort):
1048     '''
1049     This function is intended to correct the ordering in the outputs, just in
1050     case...
1051
1052     Input:
1053         target_names: a list of names sorted in the order that we want
1054         names_to_sort: a list of names to sort
1055         values_to_sort: a numpy array of values to sort
1056     Output:
1057         sorted_values: same array than values_to_sort, but this time sorted
1058     :)
1059     '''
1060
1061     # initialize the array of sorted values
1062     sorted_values = np.zeros(values_to_sort.shape)
1063

```

```

1064     # iterate for each filename in the target names
1065     for i in range(len(target_names)):
1066         # assign the value to the correct position in the array
1067         sorted_values[i, :] =
1068 values_to_sort[names_to_sort.index(target_names[i])]
1069
1070     # return the sorted values
1071     return sorted_values
1072
1073
1074 def get_labels_from_training_data(gt_folder):
1075     '''
1076     Since the training data has two folder, "Glaucoma" and "Non-Glaucoma", we
1077 can use
1078     this function to generate an array of labels automatically, according to
1079 the image
1080     filenames
1081
1082     Input:
1083         gt_folder: path to the training folder, with "Glaucoma" and "Non-
1084 Glaucoma" folder inside
1085     Output:
1086         image_filenames: filenames in the gt folders
1087         labels: binary labels (0: healthy, 1:glaucomatous)
1088     '''
1089
1090     # prepare the folders to read
1091     glaucoma_folder = path.join(gt_folder, 'Glaucoma')
1092     non_glaucoma_folder = path.join(gt_folder, 'Non-Glaucoma')
1093
1094     # get all the filenames inside each folder
1095     glaucoma_filenames = get_filenames(glaucoma_folder, 'bmp')
1096     non_glaucoma_filenames = get_filenames(non_glaucoma_folder, 'bmp')
1097
1098     # concatenate them to generate the array of image filenames
1099     image_filenames = glaucoma_filenames + non_glaucoma_filenames
1100
1101     # generate the array of labels
1102     labels = np.zeros(len(image_filenames), dtype=np.bool)
1103     labels[0:len(glaucoma_filenames)] = True
1104
1105     return image_filenames, labels
1106
1107
1108 def save_roc_curve(filename, tpr, fpr, auc):
1109     '''
1110     Save the ROC curve values on a .mat file
1111
1112     Input:
1113         filename: output filename
1114         tpr: true positive rate
1115         fpr: false positive rate
1116         auc: area under the ROC curve
1117     '''
1118
1119     # save the current ROC curve as a .mat file for MATLAB
1120     savemat(filename, {'tpr': tpr, 'fpr': fpr, 'auc': auc})

```

```

1121
1122
1123 def save_csv_classification_performance(output_filename, auc,
1124 reference_sensitivity):
1125     '''
1126         Save the AUC and the reference sensitivity values in a CSV file
1127
1128         Input:
1129             output_filename: a string with the full path and the output file name
1130             (with .csv extension)
1131             auc: area under the ROC curve
1132             reference_sensitivity: sensitivity value for a given specificity
1133     '''
1134
1135     # open the file
1136     with open(output_filename, 'w') as csv_file:
1137         # initialize the writer
1138         my_writer = csv.writer(csv_file)
1139         # write the column names
1140         my_writer.writerow(['AUC', 'Sensitivity'])
1141         # write the values
1142         my_writer.writerow([str(auc), str(reference_sensitivity)])
1143
1144
1145 def save_csv_fovea_location_performance(output_filename, distance):
1146     '''
1147         Save the mean Euclidean distance on a CSV file
1148
1149         Input:
1150             output_filename: a string with the full path and the output file name
1151             (with .csv extension)
1152             distance: mean Euclidean distance
1153     '''
1154
1155     # open the file
1156     with open(output_filename, 'w') as csv_file:
1157         # initialize the writer
1158         my_writer = csv.writer(csv_file)
1159         # write the column names
1160         my_writer.writerow(['Mean Euclidean distance'])
1161         # write the values
1162         my_writer.writerow([str(distance)])
1163
1164
1165 def save_csv_segmentation_table(table_filename, image_filenames, cup_dices,
1166 disc_dices, ae_cdrs):
1167     '''
1168         Save the table of segmentation results as a CSV file.
1169
1170         Input:
1171             table_filename: a string with the full path and the table filename
1172             (with .csv extension)
1173             image_filenames: a list of strings with the names of the images
1174             cup_dices: a numpy array with the same length than the
1175             image_filenames list, with the Dice coefficient for each optic cup
1176             disc_dices: a numpy array with the same length than the
1177             image_filenames list, with the Dice coefficient for each optic disc

```

```

1178         ae_cdrs: a numpy array with the same length than the image_filenames
1179 list, with the absolute error of the vertical cup to disc ratio
1180     '''
1181
1182     # write the data
1183     with open(table_filename, 'w') as csv_file:
1184         # initialize the writer
1185         table_writer = csv.writer(csv_file)
1186         # write the column names
1187         table_writer.writerow(['Filename', 'Cup-Dice', 'Disc-Dice', 'AE-
1188 CDR'])
1189         # write each row
1190         for i in range(len(image_filenames)):
1191             table_writer.writerow([image_filenames[i], str(cup_dices[i]),
1192 str(disc_dices[i]), str(ae_cdrs[i])])
1193
1194
1195 def save_csv_fovea_location_table(table_filename, image_filenames,
1196 distances):
1197     '''
1198     Save the table of Euclidean distances results as a CSV file.
1199
1200     Input:
1201         table_filename: a string with the full path and the table filename
1202 (with .csv extension)
1203         image_filenames: a list of strings with the names of the images
1204         distances: a 1D numpy array with the Euclidean distances of the
1205 prediction, for each image
1206     '''
1207
1208     # write the data
1209     with open(table_filename, 'w') as csv_file:
1210         # initialize the writer
1211         table_writer = csv.writer(csv_file)
1212         # write the column names
1213         table_writer.writerow(['Filename', 'Euclidean distance'])
1214         # write each row
1215         for i in range(len(image_filenames)):
1216             table_writer.writerow([image_filenames[i], str(distances[i])])
1217
1218
1219 def save_csv_mean_segmentation_performance(output_filename, mean_cup_dice,
1220 mean_disc_dice, mae_cdrs):
1221     '''
1222     Save a CSV file with the mean performance
1223
1224     Input:
1225         output_filename: a string with the full path and the table filename
1226 (with .csv extension)
1227         mean_cup_dice: average Dice coefficient for the optic cups
1228         mean_disc_dice: average Dice coefficient for the optic discs
1229         mae_cdrs: mean absolute error of the vertical cup to disc ratios
1230     '''
1231
1232     # write the data
1233     with open(output_filename, 'w') as csv_file:
1234         # initialize the writer

```

```

1235         table_writer = csv.writer(csv_file)
1236         # write the column names
1237         table_writer.writerow(['Cup-Dice', 'Disc-Dice', 'AE-CDR'])
1238         # write each row
1239         table_writer.writerow([str(mean_cup_dice), str(mean_disc_dice),
1240 str(mae_cdrs)])
1241
1242
1243 def read_fovea_location_results(csv_filename):
1244     '''
1245         Read a CSV file with 3 columns: the first contains the filenames, and the
1246         second/third have
1247         the (x,y) coordinates, respectively.
1248
1249         Input:
1250             csv_filename: full path and filename to a three columns CSV file with
1251         the fovea location results (image filename, x, y)
1252         Output:
1253             image_filenames: list of image filenames, as retrieved from the first
1254         column of the CSV file
1255             coordinates: a 2D numpy array of coordinates
1256     '''
1257
1258     # initialize the output variables
1259     image_filenames = []
1260     coordinates = None
1261
1262     # open the file
1263     with open(csv_filename, 'r') as csv_file:
1264         # initialize a reader
1265         csv_reader = csv.reader(csv_file)
1266         # ignore the first row, that only has the header
1267         next(csv_reader)
1268         # and now, iterate and fill the arrays
1269         for row in csv_reader:
1270             # append the filename
1271             image_filenames = image_filenames + [row[0]]
1272             # append the coordinates
1273             current_coordinates = np.asarray(row[1:], dtype=np.float)
1274             if coordinates is None:
1275                 coordinates = current_coordinates
1276             else:
1277                 coordinates = np.vstack((coordinates, current_coordinates))
1278
1279     return image_filenames, coordinates
1280
1281
1282 import openpyxl
1283
1284
1285 def read_gt_fovea_location(xlsx_filename, is_training=False):
1286     '''
1287         Read a XLSX file with 3 columns: the first contains the filenames, and
1288         the second/third have
1289         the (x,y) coordinates, respectively.
1290
1291         Input:

```

```

1292         xlsx_filename: full path and filename to a three columns XLSX file
1293         with the fovea location results (image filename, x, y)
1294         [is_training]: boolean indicating if we are using training data or no
1295         Output:
1296         image_filenames: list of image filenames, as retrieved from the first
1297         column of the CSV file
1298         coordinates: a 2D numpy array of coordinates
1299         '''
1300
1301         # initialize the output variables
1302         image_filenames = []
1303         coordinates = None
1304
1305         # read the xlsx file
1306         book = openpyxl.load_workbook(xlsx_filename)
1307         current_sheet = book.active
1308
1309         # iterate for each row
1310         for row in current_sheet.iter_rows(min_row=2, min_col=1):
1311             # append the filename
1312             image_filenames = image_filenames + [row[1].value]
1313             # append the coordinates
1314             if is_training:
1315                 current_coordinates = np.asarray([float(row[2].value),
1316 float(row[3].value)], dtype=np.float)
1317             else:
1318                 current_coordinates = np.asarray([float(row[3].value),
1319 float(row[4].value)], dtype=np.float)
1320             if coordinates is None:
1321                 coordinates = current_coordinates
1322             else:
1323                 coordinates = np.vstack((coordinates, current_coordinates))
1324
1325         return image_filenames, coordinates
1326
1327
1328 import openpyxl
1329
1330
1331 def read_gt_labels(xlsx_filename):
1332     '''
1333     Read a XLSX file with 2 columns: the first contains the filenames, and
1334     the second/third have
1335     the binary label for glaucoma (1) / healthy (0).
1336
1337     Input:
1338     xlsx_filename: full path and filename to a three columns XLSX file
1339     with the fovea location results (image filename, x, y)
1340     Output:
1341     image_filenames: list of image filenames, as retrieved from the first
1342     column of the CSV file
1343     labels: a 2D numpy array of coordinates
1344     '''
1345
1346     # initialize the output variables
1347     image_filenames = []
1348     labels = None

```



```

1349
1350     # read the xlsx file
1351     book = openpyxl.load_workbook(xlsx_filename)
1352     current_sheet = book.active
1353
1354     # iterate for each row
1355     for row in current_sheet.iter_rows(min_row=2, min_col=1):
1356         # append the filename
1357         current_name = row[0].value[:-3] + '.jpg'
1358         image_filenames = image_filenames + [current_name]
1359         # append the coordinates
1360         current_label = row[1].value > 0
1361         if labels is None:
1362             labels = current_label
1363         else:
1364             labels = np.vstack((labels, current_label))
1365
1366     return image_filenames, labels
1367
1368
1369 import zipfile
1370
1371
1372 def unzip_submission(submission_file, output_folder):
1373     '''
1374     Unzip a .ZIP file with a submission to REFUGE from a team
1375
1376     Input:
1377         submission_file: full path and filename of the .zip file
1378         output_folder: folder where the output will be saved
1379     '''
1380
1381     # initialize the output folder
1382     if not path.exists(output_folder):
1383         makedirs(output_folder)
1384
1385     # open the zip file
1386     zip_ref = zipfile.ZipFile(submission_file, 'r')
1387     zip_ref.extractall(output_folder)
1388     zip_ref.close()
1389
1390
1391 def export_table_of_results(table_filename, team_names, segmentation_results,
1392 classification_results,
1393                             fovea_detection_results):
1394     '''
1395     Export a table of results (unsorted) as a CSV
1396
1397     Input:
1398         table_filename: filename of the CSV file with the table of results
1399         team_names: names of the teams evaluated
1400         segmentation_results: list of segmentation results
1401         classification_results: list of classification results
1402         fovea_detection_results: list of fovea detection results
1403     '''
1404
1405     # write the data

```

```

1406     with open(table_filename, 'w') as csv_file:
1407         # initialize the writer
1408         table_writer = csv.writer(csv_file)
1409         # write the column names
1410         table_writer.writerow(
1411             ['Team name', 'Mean optic cup Dice', 'Mean optic disc Dice', 'MAE
1412 cup to disc ratio', 'AUC',
1413             'Reference Sensitivity', 'Mean Euclidean distance'])
1414         # write each row
1415         for i in range(len(team_names)):
1416             # retrieve current results
1417             current_segmentation_results = segmentation_results[i]
1418             current_classification_results = classification_results[i]
1419             current_fovea_detection_results = fovea_detection_results[i]
1420             # write a row of results
1421             table_writer.writerow(
1422                 [team_names[i], str(current_segmentation_results[0]),
1423 str(current_segmentation_results[1]),
1424                 str(current_segmentation_results[2]),
1425                 str(current_classification_results[0]),
1426 str(current_classification_results[1]),
1427                 str(current_fovea_detection_results)])
1428
1429
1430 def export_ranking(table_filename, header, team_names, scores):
1431     '''
1432     Export the ranking
1433
1434     Input:
1435         table_filename: filename of the CSV file with the table of results
1436         header: list of strings with the header for the output file
1437         team_names: names of the teams evaluated
1438         scores: a numpy array with ranking information
1439     '''
1440
1441     scores = np.asarray(scores)
1442
1443     # write the data
1444     with open(table_filename, 'w') as csv_file:
1445         # initialize the writer
1446         table_writer = csv.writer(csv_file)
1447         # write the column names
1448         table_writer.writerow(header)
1449         # write each row
1450         for i in range(len(team_names)):
1451             # write a row of results
1452             if len(scores.shape) > 1:
1453                 table_writer.writerow([team_names[i]] + scores[i,
1454 :].tolist())
1455             else:
1456                 table_writer.writerow([team_names[i]] + [scores[i]])
1457
1458
1459 def read_table_of_results(table_filename):
1460     '''
1461     Read the table of results (unsorted) as a CSV
1462

```

```

1463     Input:
1464         table_filename: filename of the CSV file with the table of results
1465     Output:
1466         header: a list of strings with the name of the evaluation metrics
1467         teams: a list of strings with the name of the teams
1468         results: a numpy matrix of evaluation metrics
1469     '''
1470
1471     # open the file
1472     with open(table_filename, 'r') as csv_file:
1473         # initialize the reader
1474         csv_reader = csv.reader(csv_file)
1475         # get the first row
1476         header = next(csv_reader)[1:]
1477
1478         # initialize the list of teams
1479         teams = []
1480         # initialize a numpy matrix with all the other results
1481         results = None
1482         # and now, iterate and fill the arrays
1483         for row in csv_reader:
1484             # append the team name
1485             teams = teams + [row[0]]
1486             # append the results
1487             current_results = np.asarray(row[1:], dtype=np.float)
1488             if results is None:
1489                 results = current_results
1490             else:
1491                 results = np.vstack((results, current_results))
1492
1493     return header, teams, results
1494

```

```

1495 #Models

```

```

1496 #Discriminator Model – Discriminator.py

```

```

1497
1498 import torch.nn as nn
1499 import torch
1500 from models.spectral import SpectralNorm
1501 import numpy as np
1502 from models.self_attn import Self_Attn
1503
1504 class FCDiscriminator(nn.Module):
1505
1506     def __init__(self, num_classes, ndf = 64):
1507         super(FCDiscriminator, self).__init__()
1508
1509         self.conv1 = nn.Conv2d(num_classes, ndf, kernel_size=4, stride=2,
1510 padding=1)
1511         self.conv2 = nn.Conv2d(ndf, ndf*2, kernel_size=4, stride=2,
1512 padding=1)
1513         self.conv3 = nn.Conv2d(ndf*2, ndf*4, kernel_size=4, stride=2,
1514 padding=1)
1515         self.conv4 = nn.Conv2d(ndf*4, ndf*8, kernel_size=4, stride=2,

```

```

1516 padding=1)
1517     self.classifier = nn.Conv2d(ndf*8, 1, kernel_size=4, stride=2,
1518 padding=1)
1519
1520     self.leaky_relu = nn.LeakyReLU(negative_slope=0.2, inplace=True)
1521     #self.up_sample = nn.Upsample(scale_factor=32, mode='bilinear')
1522     #self.sigmoid = nn.Sigmoid()
1523     # self.attn1 = Self_Attn(256, 'relu')
1524     # self.attn2 = Self_Attn(512, 'relu')
1525
1526     def forward(self, x):
1527         x = self.conv1(x)
1528         x = self.leaky_relu(x)
1529         x = self.conv2(x)
1530         x = self.leaky_relu(x)
1531         x = self.conv3(x)
1532         x = self.leaky_relu(x)
1533         # x, p1 = self.attn1(x)
1534         x = self.conv4(x)
1535         x = self.leaky_relu(x)
1536         # x, p2 = self.attn2(x)
1537         x = self.classifier(x)
1538         #x = self.up_sample(x)
1539         #x = self.sigmoid(x)
1540
1541         return x
1542
1543     class Generator(nn.Module):
1544         """Generator."""
1545
1546         def __init__(self, batch_size, image_size=64, z_dim=100, conv_dim=64):
1547             super(Generator, self).__init__()
1548             self.imsz = image_size
1549             layer1 = []
1550             layer2 = []
1551             layer3 = []
1552             last = []
1553
1554             repeat_num = int(np.log2(self.imsz)) - 3
1555             mult = 2 ** repeat_num # 8
1556             layer1.append(SpectralNorm(nn.ConvTranspose2d(z_dim, conv_dim * mult,
1557 4)))
1558             layer1.append(nn.BatchNorm2d(conv_dim * mult))
1559             layer1.append(nn.ReLU())
1560
1561             curr_dim = conv_dim * mult
1562
1563             layer2.append(SpectralNorm(nn.ConvTranspose2d(curr_dim, int(curr_dim
1564 / 2), 4, 2, 1)))
1565             layer2.append(nn.BatchNorm2d(int(curr_dim / 2)))
1566             layer2.append(nn.ReLU())
1567
1568             curr_dim = int(curr_dim / 2)
1569
1570             layer3.append(SpectralNorm(nn.ConvTranspose2d(curr_dim, int(curr_dim
1571 / 2), 4, 2, 1)))
1572             layer3.append(nn.BatchNorm2d(int(curr_dim / 2)))

```

```

1573         layer3.append(nn.ReLU())
1574
1575         if self.imsize == 64:
1576             layer4 = []
1577             curr_dim = int(curr_dim / 2)
1578             layer4.append(SpectralNorm(nn.ConvTranspose2d(curr_dim,
1579 int(curr_dim / 2), 4, 2, 1)))
1580             layer4.append(nn.BatchNorm2d(int(curr_dim / 2)))
1581             layer4.append(nn.ReLU())
1582             self.l4 = nn.Sequential(*layer4)
1583             curr_dim = int(curr_dim / 2)
1584
1585         self.l1 = nn.Sequential(*layer1)
1586         self.l2 = nn.Sequential(*layer2)
1587         self.l3 = nn.Sequential(*layer3)
1588
1589         last.append(nn.ConvTranspose2d(curr_dim, 3, 4, 2, 1))
1590         last.append(nn.Tanh())
1591         self.last = nn.Sequential(*last)
1592
1593         self.attn1 = Self_Attn(128, 'relu')
1594         self.attn2 = Self_Attn(64, 'relu')
1595
1596     def forward(self, z):
1597         z = z.view(z.size(0), z.size(1), 1, 1)
1598         out = self.l1(z)
1599         out = self.l2(out)
1600         out = self.l3(out)
1601         out, p1 = self.attn1(out)
1602         out = self.l4(out)
1603         out, p2 = self.attn2(out)
1604         out = self.last(out)
1605
1606         return out, p1, p2
1607
1608
1609 class Discriminator(nn.Module):
1610     """Discriminator, Auxiliary Classifier."""
1611
1612     def __init__(self, image_size=64, conv_dim=64):
1613         super(Discriminator, self).__init__()
1614         self.imsize = image_size
1615         layer1 = []
1616         layer2 = []
1617         layer3 = []
1618         last = []
1619
1620         layer1.append(SpectralNorm(nn.Conv2d(3, conv_dim, 4, 2, 1)))
1621         layer1.append(nn.LeakyReLU(0.1))
1622
1623         curr_dim = conv_dim
1624
1625         layer2.append(SpectralNorm(nn.Conv2d(curr_dim, curr_dim * 2, 4, 2,
1626 1)))
1627         layer2.append(nn.LeakyReLU(0.1))
1628         curr_dim = curr_dim * 2
1629

```

```

1630         layer3.append(SpectralNorm(nn.Conv2d(curr_dim, curr_dim * 2, 4, 2,
1631 1)))
1632         layer3.append(nn.LeakyReLU(0.1))
1633         curr_dim = curr_dim * 2
1634
1635         if self.imsize == 64:
1636             layer4 = []
1637             layer4.append(SpectralNorm(nn.Conv2d(curr_dim, curr_dim * 2, 4,
1638 2, 1)))
1639             layer4.append(nn.LeakyReLU(0.1))
1640             self.l4 = nn.Sequential(*layer4)
1641             curr_dim = curr_dim * 2
1642             self.l1 = nn.Sequential(*layer1)
1643             self.l2 = nn.Sequential(*layer2)
1644             self.l3 = nn.Sequential(*layer3)
1645
1646         last.append(nn.Conv2d(curr_dim, 1, 4))
1647         self.last = nn.Sequential(*last)
1648
1649         self.attn1 = Self_Attn(256, 'relu')
1650         self.attn2 = Self_Attn(512, 'relu')
1651
1652     def forward(self, x):
1653         out = self.l1(x)
1654         out = self.l2(out)
1655         out = self.l3(out)
1656         out, p1 = self.attn1(out)
1657         out = self.l4(out)
1658         out, p2 = self.attn2(out)
1659         out = self.last(out)
1660
1661         return out.squeeze(), p1, p2

```

1662 #Exponential Moving Average – optim_weight_ema.py

```

1663 class WeightEMA (object):
1664     def __init__(self, params, src_params, alpha=0.999):
1665         self.params = list(params)
1666         self.src_params = list(src_params)
1667         self.alpha = alpha
1668
1669         for p, src_p in zip(self.params, self.src_params):
1670             p.data[:] = src_p.data[:]
1671
1672     def step(self):
1673         one_minus_alpha = 1.0 - self.alpha
1674         for p, src_p in zip(self.params, self.src_params):
1675             p.data.mul_(self.alpha)
1676             p.data.add_(src_p.data * one_minus_alpha)
1677

```

1678 #Self-Attention Utility – self_attn.py

```

1679 import torch.nn as nn
1680 import torch
1681 from models.spectral import SpectralNorm
1682 import numpy as np
1683

```

```

1684
1685 class Self_Attn(nn.Module):
1686     """ Self attention Layer """
1687
1688     def __init__(self, in_dim, activation):
1689         super(Self_Attn, self).__init__()
1690         self.chanel_in = in_dim
1691         self.activation = activation
1692
1693         self.query_conv = nn.Conv2d(in_channels=in_dim, out_channels=in_dim
1694 // 8, kernel_size=1)
1695         self.key_conv = nn.Conv2d(in_channels=in_dim, out_channels=in_dim //
1696 8, kernel_size=1)
1697         self.value_conv = nn.Conv2d(in_channels=in_dim, out_channels=in_dim,
1698 kernel_size=1)
1699         self.gamma = nn.Parameter(torch.zeros(1))
1700
1701         self.softmax = nn.Softmax(dim=-1) #
1702
1703     def forward(self, x):
1704         """
1705             inputs :
1706                 x : input feature maps ( B X C X W X H)
1707             returns :
1708                 out : self attention value + input feature
1709                 attention: B X N X N (N is Width*Height)
1710         """
1711         m_batchsize, C, width, height = x.size()
1712         proj_query = self.query_conv(x).view(m_batchsize, -1, width *
1713 height).permute(0, 2, 1) # B X CX(N)
1714         proj_key = self.key_conv(x).view(m_batchsize, -1, width * height) #
1715 B X C x (*W*H)
1716         energy = torch.bmm(proj_query, proj_key) # transpose check
1717         attention = self.softmax(energy) # BX (N) X (N)
1718         proj_value = self.value_conv(x).view(m_batchsize, -1, width * height)
1719 # B X C X N
1720
1721         out = torch.bmm(proj_value, attention.permute(0, 2, 1))
1722         out = out.view(m_batchsize, C, width, height)
1723
1724         out = self.gamma * out + x
1725         return out, attention
1726
1727 #Spectral Utilities – spectral.py
1728 import torch
1729 from torch.optim.optimizer import Optimizer, required
1730
1731 from torch.autograd import Variable
1732 import torch.nn.functional as F
1733 from torch import nn
1734 from torch import Tensor
1735 from torch.nn import Parameter
1736
1737 def l2normalize(v, eps=1e-12):
1738     return v / (v.norm() + eps)

```

```

1739
1740 class SpectralNorm(nn.Module):
1741     def __init__(self, module, name='weight', power_iterations=1):
1742         super(SpectralNorm, self).__init__()
1743         self.module = module
1744         self.name = name
1745         self.power_iterations = power_iterations
1746         if not self._made_params():
1747             self._make_params()
1748
1749     def _update_u_v(self):
1750         u = getattr(self.module, self.name + "_u")
1751         v = getattr(self.module, self.name + "_v")
1752         w = getattr(self.module, self.name + "_bar")
1753
1754         height = w.data.shape[0]
1755         for _ in range(self.power_iterations):
1756             v.data = l2normalize(torch.mv(torch.t(w.view(height,-1).data),
1757 u.data))
1758             u.data = l2normalize(torch.mv(w.view(height,-1).data, v.data))
1759
1760             # sigma = torch.dot(u.data, torch.mv(w.view(height,-1).data, v.data))
1761             sigma = u.dot(w.view(height, -1).mv(v))
1762             setattr(self.module, self.name, w / sigma.expand_as(w))
1763
1764     def _made_params(self):
1765         try:
1766             u = getattr(self.module, self.name + "_u")
1767             v = getattr(self.module, self.name + "_v")
1768             w = getattr(self.module, self.name + "_bar")
1769             return True
1770         except AttributeError:
1771             return False
1772
1773
1774     def _make_params(self):
1775         w = getattr(self.module, self.name)
1776
1777         height = w.data.shape[0]
1778         width = w.view(height, -1).data.shape[1]
1779
1780         u = Parameter(w.data.new(height).normal_(0, 1), requires_grad=False)
1781         v = Parameter(w.data.new(width).normal_(0, 1), requires_grad=False)
1782         u.data = l2normalize(u.data)
1783         v.data = l2normalize(v.data)
1784         w_bar = Parameter(w.data)
1785
1786         del self.module._parameters[self.name]
1787
1788         self.module.register_parameter(self.name + "_u", u)
1789         self.module.register_parameter(self.name + "_v", v)
1790         self.module.register_parameter(self.name + "_bar", w_bar)
1791
1792
1793     def forward(self, *args):
1794         self._update_u_v()
1795         return self.module.forward(*args)

```



```

1796 #U-NET Model – unet.py
1797 # full assembly of the sub-parts to form the complete net
1798
1799 # sub-parts of the U-Net model
1800
1801 import torch
1802 import torch.nn as nn
1803 import torch.nn.functional as F
1804
1805
1806 class UNet(nn.Module):
1807     def __init__(self, n_channels, n_classes):
1808         super(UNet, self).__init__()
1809         self.down1 = down(n_channels, 32)
1810         self.AdditionalInput1 = AdditionalInput(2, 3, 64)
1811         # self.single_conv1 = single_conv(3, 64)
1812
1813         self.x1_out = outconv(32, n_classes)
1814
1815         self.down2 = down(96, 64)
1816         self.AdditionalInput2 = AdditionalInput(4, 3, 128)
1817         # self.single_conv2 = single_conv(3, 128)
1818
1819         self.x2_out = outconv(64, n_classes)
1820
1821         self.down3 = down(192, 128)
1822         self.AdditionalInput3 = AdditionalInput(8, 3, 256)
1823         # self.single_conv3 = single_conv(3, 256)
1824
1825         self.x3_out = outconv(128, n_classes)
1826
1827         self.down4 = down(384, 256)
1828
1829         self.x4_out = outconv(256, n_classes)
1830
1831         self.down5 = down(256, 256)
1832
1833         self.conv5 = double_conv(256, 512)
1834
1835         self.up1 = up(512, 256)
1836         self.up2 = up(256, 128)
1837         self.up3 = up(128, 64)
1838         self.up4 = up(64, 32)
1839
1840         self.out6 = outconv(256, n_classes)
1841         self.out7 = outconv(128, n_classes)
1842         self.out8 = outconv(64, n_classes)
1843         self.out9 = outconv(32, n_classes)
1844
1845     def forward(self, x):
1846         # input1 = x      # 400x400
1847         input2 = self.AdditionalInput1(x) # 200x200
1848         input3 = self.AdditionalInput2(x) # 100x100
1849         input4 = self.AdditionalInput3(x) # 50x50
1850
1851         ###first level
1852         x1_conv, x1_downsample = self.down1(x) # 32x200x200

```

```

1853
1854     # x1_out = self.x1_out(x1_conv)
1855
1856     #####second level
1857     input2 = torch.cat([input2, x1_downsample], dim=1)
1858     x2_conv, x2_downsample = self.down2(input2) # 64x100x100
1859
1860     # x2_out = self.x2_out(x2_conv)
1861
1862     #####3rd level
1863     input3 = torch.cat([input3, x2_downsample], dim=1)
1864     x3_conv, x3_downsample = self.down3(input3) # 128x50x50
1865
1866     # x3_out = self.x3_out(x3_conv)
1867
1868     ### 4th level
1869     input4 = torch.cat([input4, x3_downsample], dim=1)
1870     x4_conv, x4_downsample = self.down4(input4) # 256x25x25
1871
1872     # x4_out = self.x4_out(x4_conv)
1873
1874     ### 5th level
1875     x5 = self.conv5(x4_downsample) # 512x25x25
1876
1877     ### -4th level
1878     x6 = self.up1(x5, x4_conv) # 256x50x50
1879     side6 = nn.Upsample(scale_factor=8, mode='bilinear',
1880 align_corners=True)(x6)
1881
1882     ### -3th level
1883     x7 = self.up2(x6, x3_conv) # 128x100x100
1884     side7 = nn.Upsample(scale_factor=4, mode='bilinear',
1885 align_corners=True)(x7)
1886
1887     x8 = self.up3(x7, x2_conv)
1888     side8 = nn.Upsample(scale_factor=2, mode='bilinear',
1889 align_corners=True)(x8)
1890
1891     x9 = self.up4(x8, x1_conv)
1892
1893     out6 = self.out6(side6)
1894     out7 = self.out7(side7)
1895     out8 = self.out8(side8)
1896     out9 = self.out9(x9)
1897
1898     my_list = [out6, out7, out8, out9]
1899     out10 = torch.mean(torch.stack(my_list), dim=0)
1900
1901     return out6, out7, out8, out9, out10
1902
1903
1904 class double_conv(nn.Module):
1905     '''(conv => BN => ReLU) * 2'''
1906
1907     def __init__(self, in_ch, out_ch):
1908         super(double_conv, self).__init__()
1909         self.conv = nn.Sequential(

```

```

1910         nn.Conv2d(in_ch, out_ch, 3, padding=1),
1911         nn.BatchNorm2d(out_ch),
1912         nn.ReLU(inplace=True),
1913         nn.Conv2d(out_ch, out_ch, 3, padding=1),
1914         nn.BatchNorm2d(out_ch),
1915         nn.ReLU(inplace=True)
1916     )
1917
1918     def forward(self, x):
1919         x = self.conv(x)
1920         return x
1921
1922
1923     class single_conv(nn.Module):
1924         def __init__(self, in_ch, out_ch):
1925             super(single_conv, self).__init__()
1926             self.conv = nn.Sequential(
1927                 nn.Conv2d(in_ch, out_ch, 3, padding=1),
1928                 nn.ReLU(inplace=True)
1929             )
1930
1931         def forward(self, x):
1932             x = self.conv(x)
1933             return x
1934
1935
1936     class AdditionalInput(nn.Module):
1937         def __init__(self, poolsize, in_ch, out_ch):
1938             super(AdditionalInput, self).__init__()
1939             self.AddiInput = nn.Sequential(
1940                 nn.AvgPool2d(poolsize),
1941                 nn.Conv2d(in_ch, out_ch, 3, padding=1)
1942             )
1943
1944         def forward(self, x):
1945             x = self.AddiInput(x)
1946             return x
1947
1948
1949     class inconv(nn.Module):
1950         def __init__(self, in_ch, out_ch):
1951             super(inconv, self).__init__()
1952             self.conv = double_conv(in_ch, out_ch)
1953
1954         def forward(self, x):
1955             x = self.conv(x)
1956             return x
1957
1958
1959     class down(nn.Module):
1960         def __init__(self, in_ch, out_ch):
1961             super(down, self).__init__()
1962             self.mpconv = double_conv(in_ch, out_ch)
1963             self.downsample = nn.MaxPool2d(2)
1964
1965         def forward(self, x):
1966             x_conv = self.mpconv(x)

```

```

1967         x_downsample = self.downsample(x_conv)
1968         return x_conv, x_downsample
1969
1970
1971 class up(nn.Module):
1972     def __init__(self, in_ch, out_ch, bilinear=False):
1973         super(up, self).__init__()
1974
1975         # would be a nice idea if the upsampling could be learned too,
1976         # but my machine do not have enough memory to handle all those
1977         weights
1978         self.conv = double_conv(in_ch, out_ch)
1979
1980         if bilinear:
1981             self.up = nn.Upsample(scale_factor=2, mode='bilinear',
1982 align_corners=True)
1983         else:
1984             # self.up = nn.ConvTranspose2d(in_ch//2, in_ch//2, 2, stride=2)
1985             self.up = nn.ConvTranspose2d(in_ch, in_ch // 2, 2, stride=2)
1986
1987     def forward(self, input_1, input_2):
1988         input_1 = self.up(input_1)
1989
1990         # input is CHW
1991         diffY = input_2.size()[2] - input_1.size()[2]
1992         diffX = input_2.size()[3] - input_1.size()[3]
1993
1994         input_1 = F.pad(input_1, (diffX // 2, diffX - diffX // 2,
1995                                diffY // 2, diffY - diffY // 2))
1996
1997         # for padding issues, see
1998         # https://github.com/HaiyongJiang/U-Net-Pytorch-Unstructured-
1999 Buggy/commit/0e854509c2cea854e247a9c615f175f76fbb2e3a
2000         # https://github.com/xiaopeng-liao/Pytorch-
2001 UNet/commit/8ebac70e633bac59fc22bb5195e513d5832fb3bd
2002
2003         x = torch.cat([input_2, input_1], dim=1)
2004         x = self.conv(x)
2005         return x
2006
2007
2008 class up5(nn.Module):
2009     def __init__(self, in_ch, out_ch, bilinear=True):
2010         super(up5, self).__init__()
2011
2012         # would be a nice idea if the upsampling could be learned too,
2013         # but my machine do not have enough memory to handle all those
2014         weights
2015         self.conv = double_conv(in_ch, out_ch)
2016
2017         if bilinear:
2018             self.up = nn.Upsample(scale_factor=2, mode='bilinear',
2019 align_corners=True)
2020         else:
2021             self.up = nn.ConvTranspose2d(in_ch // 2, in_ch // 2, 2, stride=2)
2022
2023     def forward(self, x):

```

```

2024         x = self.up5(x)
2025         return x
2026
2027
2028 class outconv(nn.Module):
2029     def __init__(self, in_ch, out_ch):
2030         super(outconv, self).__init__()
2031         self.conv = nn.Sequential(nn.Conv2d(in_ch, out_ch, 1),
2032                                     nn.Sigmoid()
2033                                     )
2034
2035     def forward(self, x):
2036         x = self.conv(x)
2037         return x
2038
2039
2040 def count_parameters(model):
2041     return sum(p.numel() for p in model.parameters() if p.requires_grad)
2042
2043
2044 if __name__ == '__main__':
2045     model = UNet(3, 3).cuda()
2046     input = torch.zeros((1, 3, 400, 400)).cuda()
2047     # m = nn.AvgPool2d(2).cuda()
2048     # input2 = m(input).cuda()
2049     # n = nn.Conv2d(3,256,3, padding=1).cuda()
2050     # input3 = n(input2)
2051     # print(input3.shape)
2052     x7, out6, _, _, _, output9 = model(input)
2053     hello = nn.Upsample(scale_factor=4, mode='bilinear',
2054 align_corners=True)(x7)
2055     convolution = nn.Conv2d(128, 3,1).cuda()
2056     hello2 = convolution(hello).cuda()
2057     print(x7.shape)
2058     print(hello.shape)
2059     print(out6.shape)
2060     # print(side7.shape)
2061
2062
2063 #U-NET Single Scale (CFEA) – unet_SingleScale.py
2064 1# full assembly of the sub-parts to form the complete net
2065
2066 # sub-parts of the U-Net model
2067
2068 import torch
2069 import torch.nn as nn
2070 import torch.nn.functional as F
2071
2072
2073
2074 class UNet(nn.Module):
2075     def __init__(self, n_channels, n_classes):
2076
2077         super(UNet, self).__init__()
2078         self.down1 = down(n_channels, 32)

```

```

2079     # self.AdditionalInput1 = AdditionalInput(2, 3, 64)
2080     # self.single_conv1 = single_conv(3, 64)
2081
2082     # self.x1_out = outconv(32, n_classes)
2083
2084     self.down2 = down(32, 64)
2085     # self.AdditionalInput2 = AdditionalInput(4, 3, 128)
2086     # self.single_conv2 = single_conv(3, 128)
2087
2088     # self.x2_out = outconv(64, n_classes)
2089
2090     self.down3 = down(64, 128)
2091     # self.AdditionalInput3 = AdditionalInput(8, 3, 256)
2092     # self.single_conv3 = single_conv(3, 256)
2093
2094     # self.x3_out = outconv(128, n_classes)
2095
2096     self.down4 = down(128, 256)
2097
2098     # self.x4_out = outconv(256, n_classes)
2099
2100     # self.down5 = down(256, 512)
2101
2102     self.conv5 = double_conv(256, 512)
2103
2104     self.up1 = up(512, 256)
2105     self.up2 = up(256, 128)
2106     self.up3 = up(128, 64)
2107     self.up4 = up(64, 32)
2108
2109     self.out6 = outconv(256, n_classes)
2110     self.out7 = outconv(128, n_classes)
2111     self.out8 = outconv(64, n_classes)
2112     self.out9 = outconv(32, n_classes)
2113
2114     def forward(self, x):
2115
2116         ###first level
2117         x1_conv, x1_downsample = self.down1(x)    #32x200x200
2118
2119
2120
2121         ###second level
2122
2123         x2_conv, x2_downsample = self.down2(x1_downsample) #64x100x100
2124
2125
2126
2127         ###3rd level
2128
2129         x3_conv, x3_downsample = self.down3(x2_downsample)    #128x50x50
2130
2131
2132
2133         ### 4th level
2134
2135         x4_conv, x4_downsample = self.down4(x3_downsample) #256x25x25

```

```

2136     ### 5th level
2137     x5 = self.conv5(x4_downsample) #512x25x25
2138
2139     ### -4th level
2140     x6 = self.up1(x5, x4_conv) #256x50x50
2141     side6 = nn.Upsample(scale_factor=8, mode='bilinear',
2142 align_corners=True)(x6) #512x400x400
2143
2144     ### -3th level
2145     x7 = self.up2(x6, x3_conv) #256x100x100
2146     side7 = nn.Upsample(scale_factor=4, mode='bilinear',
2147 align_corners=True)(x7)
2148
2149     x8 = self.up3(x7, x2_conv) #128x100x100
2150     side8 = nn.Upsample(scale_factor=2, mode='bilinear',
2151 align_corners=True)(x8)
2152
2153     x9 = self.up4(x8, x1_conv) #64x400x400
2154
2155     out6 = self.out6(side6)
2156     out7 = self.out7(side7)
2157     out8 = self.out8(side8)
2158     out9 = self.out9(x9)
2159
2160     #my_list=[out6, out7, out8, out9]
2161     my_list = [out6, out7, out8, out9]
2162     out10 = torch.mean(torch.stack(my_list), dim=0)
2163
2164     return out6, out7, out8, out9, out10
2165
2166 class double_conv(nn.Module):
2167     '''(conv => BN => ReLU) * 2'''
2168     def __init__(self, in_ch, out_ch):
2169         super(double_conv, self).__init__()
2170         self.conv = nn.Sequential(
2171             nn.Conv2d(in_ch, out_ch, 3, padding=1),
2172             nn.BatchNorm2d(out_ch),
2173             nn.ReLU(inplace=True),
2174             nn.Conv2d(out_ch, out_ch, 3, padding=1),
2175             nn.BatchNorm2d(out_ch),
2176             nn.ReLU(inplace=True)
2177         )
2178
2179     def forward(self, x):
2180         x = self.conv(x)
2181         return x
2182
2183
2184 class single_conv(nn.Module):
2185     def __init__(self, in_ch, out_ch):
2186         super(single_conv, self).__init__()
2187         self.conv = nn.Sequential(
2188             nn.Conv2d(in_ch, out_ch, 3, padding=1),
2189             nn.ReLU(inplace=True)
2190         )
2191
2192     def forward(self, x):

```

```

2193         x = self.conv(x)
2194         return x
2195
2196
2197 class AdditionalInput(nn.Module):
2198     def __init__(self, poolsize, in_ch, out_ch):
2199         super(AdditionalInput, self).__init__()
2200         self.AddiInput = nn.Sequential(
2201             nn.AvgPool2d(poolsize),
2202             nn.Conv2d(in_ch, out_ch, 3, padding=1)
2203         )
2204
2205     def forward(self, x):
2206         x = self.AddiInput(x)
2207         return x
2208
2209
2210 class inconv(nn.Module):
2211     def __init__(self, in_ch, out_ch):
2212         super(inconv, self).__init__()
2213         self.conv = double_conv(in_ch, out_ch)
2214
2215     def forward(self, x):
2216         x = self.conv(x)
2217         return x
2218
2219
2220 class down(nn.Module):
2221     def __init__(self, in_ch, out_ch):
2222         super(down, self).__init__()
2223         self.mpconv = double_conv(in_ch, out_ch)
2224         self.downsample = nn.MaxPool2d(2)
2225
2226     def forward(self, x):
2227         x_conv = self.mpconv(x)
2228         x_downsample = self.downsample(x_conv)
2229         return x_conv, x_downsample
2230
2231
2232 class up(nn.Module):
2233     def __init__(self, in_ch, out_ch, bilinear=False):
2234         super(up, self).__init__()
2235
2236         # would be a nice idea if the upsampling could be learned too,
2237         # but my machine do not have enough memory to handle all those
2238         weights
2239         self.conv = double_conv(in_ch, out_ch)
2240
2241         if bilinear:
2242             self.up = nn.Upsample(scale_factor=2, mode='bilinear',
2243 align_corners=True)
2244         else:
2245             # self.up = nn.ConvTranspose2d(in_ch//2, in_ch//2, 2, stride=2)
2246             self.up = nn.ConvTranspose2d(in_ch, in_ch // 2, 2, stride=2)
2247
2248     def forward(self, input_1, input_2):
2249         input_1 = self.up(input_1)

```



```

2250
2251     # input is CHW
2252     diffY = input_2.size()[2] - input_1.size()[2]
2253     diffX = input_2.size()[3] - input_1.size()[3]
2254
2255     input_1 = F.pad(input_1, (diffX // 2, diffX - diffX//2,
2256                             diffY // 2, diffY - diffY//2))
2257
2258
2259     x = torch.cat([input_2, input_1], dim=1)
2260     x = self.conv(x)
2261     return x
2262
2263
2264 class up5(nn.Module):
2265     def __init__(self, in_ch, out_ch, bilinear=True):
2266         super(up5, self).__init__()
2267
2268         # would be a nice idea if the upsampling could be learned too,
2269         # but my machine do not have enough memory to handle all those
2270         weights
2271         self.conv = double_conv(in_ch, out_ch)
2272
2273         if bilinear:
2274             self.up = nn.Upsample(scale_factor=2, mode='bilinear',
2275 align_corners=True)
2276         else:
2277             self.up = nn.ConvTranspose2d(in_ch // 2, in_ch // 2, 2, stride=2)
2278
2279     def forward(self, x):
2280         x = self.up5(x)
2281         return x
2282
2283
2284 class outconv(nn.Module):
2285     def __init__(self, in_ch, out_ch):
2286         super(outconv, self).__init__()
2287         self.conv = nn.Sequential(nn.Conv2d(in_ch, out_ch, 1),
2288                                   nn.Sigmoid()
2289                                   )
2290
2291     def forward(self, x):
2292         x = self.conv(x)
2293         return x
2294
2295
2296 def count_parameters(model):
2297     return sum(p.numel() for p in model.parameters() if p.requires_grad)
2298
2299
2300 if __name__ == '__main__':
2301     model = UNet(3, 3).cuda()
2302     input = torch.zeros((1, 3, 512, 512)).cuda()
2303     output = model(input)
2304     print(output.shape)
2305     print(count_parameters(model))

```

```

2306 #Training

2307 #Arguments – arguments.py
2308 import argparse
2309
2310 BATCH_SIZE = 1
2311 ITER_SIZE = 1
2312 NUM_WORKERS = 0
2313 INPUT_SIZE = '600, 600'
2314 TGT_SIZE = '500,500'
2315 LEARNING_RATE = 1e-4
2316 MOMENTUM = 0.9
2317 NUM_CLASSES = 3
2318 NUM_STEPS = 250000
2319 NUM_STEPS_STOP = 200000 # early stopping
2320 POWER = 0.9
2321 RESTORE_FROM = ''
2322 SAVE_NUM_IMAGES = 2
2323 SAVE_PRED_EVERY = 1000
2324 SNAPSHOT_DIR = '/home/charlietran/CADA_Tutorial/Model_Weights/Trial1/'
2325 TENSORBOARD_DIR =
2326 '/home/charlietran/CADA_Tutorial/tensorboard_directory/Trial1/'
2327 WEIGHT_DECAY = 0.0005
2328 TEACHER_ALPHA = 0.99
2329
2330 LEARNING_RATE_D = 2.5e-5
2331
2332 class_weights=[0.4,0.4,0.2]
2333
2334
2335 def get_arguments():
2336     """Parse all the arguments provided from the CLI.
2337     Returns:
2338         A list of parsed arguments.
2339     """
2340     parser = argparse.ArgumentParser(description="Domain Adaptation ")
2341     parser.add_argument("--batch-size", type=int, default=BATCH_SIZE,
2342                         help="Number of images sent to the network in one
2343 step.")
2344     parser.add_argument("--iter-size", type=int, default=ITER_SIZE,
2345                         help="Accumulate gradients for ITER_SIZE
2346 iterations.")
2347     parser.add_argument("--num-workers", type=int, default=NUM_WORKERS,
2348                         help="number of workers for multithread
2349 dataloading.")
2350     parser.add_argument("--input-size", type=str, default=INPUT_SIZE,
2351                         help="Comma-separated string with height and width of
2352 source images.")
2353     parser.add_argument("--tgt-size", type=str, default=TGT_SIZE,
2354                         help="Comma-separ ated string with height and width
2355 of target images.")
2356     parser.add_argument("--is-training", action="store_true",
2357                         help="Whether to updates the running means and
2358 variances during the training.")
2359     parser.add_argument("--learning-rate", type=float, default=LEARNING_RATE,
2360                         help="Base learning rate for training with polynomial

```

```

2361 decay.")
2362     parser.add_argument("--learning-rate-D", type=float,
2363 default=LEARNING_RATE_D,
2364     help="Base learning rate for discriminator.")
2365     parser.add_argument("--momentum", type=float, default=MOMENTUM,
2366     help="Momentum component of the optimiser.")
2367     parser.add_argument("--num-classes", type=int, default=NUM_CLASSES,
2368     help="Number of classes to predict (including
2369 background).")
2370     parser.add_argument("--num-steps", type=int, default=NUM_STEPS,
2371     help="Number of training steps.")
2372     parser.add_argument("--num-steps-stop", type=int, default=NUM_STEPS_STOP,
2373     help="Number of training steps for early stopping.")
2374     parser.add_argument("--power", type=float, default=POWER,
2375     help="Decay parameter to compute the learning rate.")
2376     parser.add_argument("--random-mirror", action="store_true",
2377     help="Whether to randomly mirror the inputs during
2378 the training.")
2379     parser.add_argument("--random-scale", action="store_true",
2380     help="Whether to randomly scale the inputs during the
2381 training.")
2382     parser.add_argument("--restore-from", type=str, default=RESTORE_FROM,
2383     help="Where restore model parameters from.")
2384     parser.add_argument("--save-num-images", type=int,
2385 default=SAVE_NUM_IMAGES,
2386     help="How many images to save.")
2387     parser.add_argument("--save-pred-every", type=int,
2388 default=SAVE_PRED_EVERY,
2389     help="Save summaries and checkpoint every often.")
2390     parser.add_argument("--snapshot-dir", type=str, default=SNAPSHOT_DIR,
2391     help="Where to save snapshots of the model.")
2392     parser.add_argument("--tensorboard-dir", type=str, default=
2393 TENSORBOARD_DIR, help="WHERE TO TENSORBOARD")
2394     parser.add_argument("--weight-decay", type=float, default=WEIGHT_DECAY,
2395     help="Regularisation parameter for L2-loss.")
2396     parser.add_argument("--gpu", type=int, default=0,
2397     help="choose gpu device.")
2398     parser.add_argument("--teacher_alpha", type=float, default=TEACHER_ALPHA,
2399     help="Teacher EMA alpha (decay)")
2400     parser.add_argument('--unsup_weight6', type=float, default= 1,
2401     help='unsupervised loss weight')
2402     parser.add_argument('--unsup_weight7', type=float, default= 1,
2403     help='unsupervised loss weight')
2404     parser.add_argument('--unsup_weight8', type=float, default= 1,
2405     help='unsupervised loss weight')
2406     parser.add_argument('--unsup_weight9', type=float, default= 1,
2407     help='unsupervised loss weight')
2408     parser.add_argument('--unsup_weight10', type=float, default=1,
2409     help='unsupervised loss weight')
2410     parser.add_argument("--lambda-adv-tgt6", type=float, default= 1,
2411     help="lambda_adv for adversarial training.")
2412     parser.add_argument("--lambda-adv-tgt7", type=float, default= 1,
2413     help="lambda_adv for adversarial training.")
2414     parser.add_argument("--lambda-adv-tgt8", type=float, default= 1,
2415     help="lambda_adv for adversarial training.")
2416     parser.add_argument("--lambda-adv-tgt9", type=float, default= 1,
2417     help="lambda_adv for adversarial training.")

```

```

2418     parser.add_argument("--lambda-adv-tgt10", type=float, default= 1,
2419                          help="lambda_adv for adversarial training.")
2420     parser.add_argument('--mse-weight6', type=float, default=1,
2421                          help='mse weight for discriminative training')
2422     parser.add_argument('--mse-weight7', type=float, default=1,
2423                          help='mse weight for discriminative training')
2424     parser.add_argument('--mse-weight8', type=float, default=1,
2425                          help='mse weight for discriminative training')
2426     parser.add_argument('--mse-weight9', type=float, default=1,
2427                          help='mse weight for discriminative training')
2428     parser.add_argument('--mse-weight10', type=float, default=1,
2429                          help='mse weight for discriminative training')
2430
2431
2432
2433     parser.add_argument("--class_weights", type=float, default=[0.4, 0.4,
2434 0.2],
2435                          help="segmentation pixel-wise class weights.")
2436
2437     parser.add_argument('--t', type=int, default=3, help='t for Recurrent
2438 step of R2U_Net or R2AttU_Net')
2439
2440     return parser.parse_args()
2441

```

```

2442 #Training Utilities – pytorch_utils.py
2443
2444 import torch.nn as nn
2445 import torch
2446 import torch.nn.functional as F
2447 from torch.autograd import Variable
2448
2449 def lr_poly(base_lr, iter, max_iter, power):
2450     return base_lr * ((1 - float(iter) / max_iter) ** (power))
2451
2452
2453 def adjust_learning_rate(optimizer, i_iter, args):
2454     lr = lr_poly(args.learning_rate, i_iter, args.num_steps, args.power)
2455     optimizer.param_groups[0]['lr'] = lr
2456     if len(optimizer.param_groups) > 1:
2457         optimizer.param_groups[1]['lr'] = lr * 10
2458
2459
2460 def adjust_learning_rate_D(optimizer, i_iter, args):
2461     lr = lr_poly(args.learning_rate_D, i_iter, args.num_steps, args.power)
2462     optimizer.param_groups[0]['lr'] = lr
2463     if len(optimizer.param_groups) > 1:
2464         optimizer.param_groups[1]['lr'] = lr * 10
2465
2466
2467 def calc_mse_loss(item1, item2, batch_size):
2468     criterion = nn.MSELoss(reduce=False)
2469     return criterion(item1, item2).sum() / batch_size
2470
2471
2472 def calc_l1_loss(item1, item2, batch_size, gpu):
2473     item2 = Variable(item2.float()).cuda(gpu)

```

```

2473     criterion = nn.L1Loss()
2474     return criterion(item1, item2).sum() / batch_size
2475
2476
2477 class LossMulti(nn.Module):
2478     def __init__(self, jaccard_weight=0, class_weights=None, num_classes=1):
2479         if class_weights is not None:
2480             self.nll_weight =
2481 class_weights#Variable(class_weights.float()).cuda()
2482         else:
2483             self.nll_weight = None
2484
2485         self.jaccard_weight = jaccard_weight
2486         self.num_classes = num_classes
2487
2488     def __call__(self, outputs, targets):
2489
2490         loss = (1 - self.jaccard_weight) * F.cross_entropy(outputs, targets,
2491 weight=self.nll_weight)
2492
2493         if self.jaccard_weight:
2494             eps = 1e-15
2495             outputs = F.softmax(outputs)
2496             for cls in range(self.num_classes):
2497                 jaccard_target = (targets == cls).float()
2498                 jaccard_output = outputs[:, cls]#.exp()
2499                 intersection = (jaccard_output * jaccard_target).sum()
2500
2501                 union = jaccard_output.sum() + jaccard_target.sum()
2502                 loss -= torch.log((intersection + eps) / (union -
2503 intersection + eps)) * self.jaccard_weight
2504             return loss
2505
2506
2507
2508 def Weighted_Jaccard_loss (label, pred, class_weights=None, gpu=0):
2509     """
2510     This function returns cross entropy loss for semantic segmentation
2511     """
2512     # out shape batch_size x channels x h x w -> batch_size x channels x h x
2513 w
2514     # label shape h x w x 1 x batch_size -> batch_size x 1 x h x w
2515     label = Variable(label.long()).cuda(gpu)
2516     if class_weights is not None and class_weights != 0:
2517         class_weights = torch.Tensor(class_weights)
2518         class_weights = Variable(class_weights).cuda(gpu)
2519         criterion = LossMulti(jaccard_weight=0.5,
2520 class_weights=class_weights,num_classes=3)#.cuda(gpu)
2521     else:
2522         criterion = LossMulti(jaccard_weight=0.5, num_classes=3) #
2523 .cuda(gpu)
2524     return criterion(pred, label)
2525
2526
2527
2528 def dice_loss(true, logits, eps=1e-7):
2529     """Computes the Sørensen-Dice loss.

```

```

2530 Note that PyTorch optimizers minimize a loss. In this
2531 case, we would like to maximize the dice loss so we
2532 return the negated dice loss.
2533 Args:
2534     true: a tensor of shape [B, 1, H, W].
2535     logits: a tensor of shape [B, C, H, W]. Corresponds to
2536             the raw output or logits of the model.
2537     eps: added to the denominator for numerical stability.
2538 Returns:
2539     dice_loss: the Sørensen-Dice loss.
2540     https://github.com/kevinzakka/pytorch-goodies/blob/master/losses.py
2541 """
2542 num_classes = logits.shape[1]
2543 if num_classes == 1:
2544     true_1_hot = torch.eye(num_classes + 1)[true.squeeze(1)]
2545     true_1_hot = true_1_hot.permute(0, 3, 1, 2).float()
2546     true_1_hot_f = true_1_hot[:, 0:1, :, :]
2547     true_1_hot_s = true_1_hot[:, 1:2, :, :]
2548     true_1_hot = torch.cat([true_1_hot_s, true_1_hot_f], dim=1)
2549     pos_prob = torch.sigmoid(logits)
2550     neg_prob = 1 - pos_prob
2551     probas = torch.cat([pos_prob, neg_prob], dim=1)
2552 else:
2553     true_1_hot = torch.eye(num_classes)[true.squeeze(1)]
2554     true_1_hot = true_1_hot.permute(0, 3, 1, 2).float()
2555     probas = F.softmax(logits, dim=1)
2556 true_1_hot = true_1_hot.type(logits.type())
2557 dims = (0,) + tuple(range(2, true.ndimension()))
2558 intersection = torch.sum(probas * true_1_hot, dims)
2559 cardinality = torch.sum(probas + true_1_hot, dims)
2560 dice_loss = (2. * intersection / (cardinality + eps)).mean()
2561 return (1 - dice_loss)

```

```

2562 #CADA Training Code – CADA.py
2563
2564 import os
2565 os.environ["CUDA_DEVICE_ORDER"] = "PCI_BUS_ID"
2566 os.environ["CUDA_VISIBLE_DEVICES"] = "3,4"
2567 import torch
2568 from torch.utils import data
2569 from torch.autograd import Variable
2570 import torch.optim as optim
2571 import torch.backends.cudnn as cudnn
2572 import torch.nn.functional as F
2573 from albumentations import (
2574     HorizontalFlip,
2575     VerticalFlip,
2576     Compose,
2577     Transpose,
2578     RandomRotate90,
2579     OneOf,
2580     CLAHE,
2581     RandomGamma,
2582     HueSaturationValue,
2583     IAAGaussianNoise, GaussNoise,
2584     RandomBrightnessContrast,

```

```

2585         IAASharpen, IAAEmboss
2586     )
2587
2588     from models.unet import UNet
2589     from models.discriminator import FCDiscriminator
2590     from dataset.refuge import REFUGE
2591     from pytorch_utils import (adjust_learning_rate, adjust_learning_rate_D,
2592                                calc_mse_loss, Weighted_Jaccard_loss, dice_loss)
2593     from models import optim_weight_ema
2594     from arguments import get_arguments
2595
2596     import tensorboard_logger as tb_logger
2597     import numpy as np
2598
2599     aug_student = Compose([
2600         OneOf([
2601             Transpose(p=0.5),
2602             HorizontalFlip(p=0.5),
2603             VerticalFlip(p=0.5),
2604             RandomRotate90(p=0.5)], p=0.2),
2605
2606         OneOf([
2607             IAAAdditiveGaussianNoise(p=0.5),
2608             GaussNoise(p=0.5),
2609         ], p=0.2),
2610
2611         OneOf([
2612             CLAHE(clip_limit=2),
2613             IAASharpen(p=0.5),
2614             IAAEmboss(p=0.5),
2615             RandomBrightnessContrast(p=0.5),
2616         ], p=0.2),
2617         HueSaturationValue(p=0.2),
2618         RandomGamma(p=0.2)])
2619
2620
2621
2622     aug_teacher = Compose([
2623
2624         OneOf([
2625             IAAAdditiveGaussianNoise(p=0.5),
2626             GaussNoise(p=0.5),
2627         ], p=0.2),
2628
2629         OneOf([
2630             CLAHE(clip_limit=2),
2631             IAASharpen(p=0.5),
2632             IAAEmboss(p=0.5),
2633             RandomBrightnessContrast(p=0.5),
2634         ], p=0.2),
2635         HueSaturationValue(p=0.2),
2636         RandomGamma(p=0.2)])
2637
2638     def main():
2639         """Create the model and start the training."""
2640         args = get_arguments()
2641

```

```

2642     cudnn.enabled = True
2643     n_discriminators = 5
2644     logger = tb_logger.Logger(logdir= args.tensorboard_dir, flush_secs=2)
2645     # create teacher & student
2646     student_net = UNet(3, n_classes=args.num_classes)
2647     # saved_state_dict = torch.load(args.restore_from)
2648     # print('The pretrained weights have been loaded', args.restore_from)
2649     # student_net.load_state_dict(saved_state_dict)
2650     teacher_net = UNet(3, n_classes=args.num_classes)
2651     # saved_state_dict = torch.load(args.restore_from)
2652     # teacher_net.load_state_dict(saved_state_dict)
2653     student_params = list(student_net.parameters())
2654
2655
2656     # teacher doesn't need gradient as it's just a EMA of the student
2657     teacher_params = list(teacher_net.parameters())
2658     for param in teacher_params:
2659         param.requires_grad = False
2660
2661     student_net.train()
2662     student_net.cuda(args.gpu)
2663     teacher_net.train()
2664     teacher_net.cuda(args.gpu)
2665
2666     cudnn.benchmark = True
2667     unsup_weights = [args.unsup_weight6, args.unsup_weight7,
2668 args.unsup_weight8,
2669                     args.unsup_weight9, args.unsup_weight10]
2670     lambda_adv_tgts = [args.lambda_adv_tgt6, args.lambda_adv_tgt7,
2671                       args.lambda_adv_tgt8, args.lambda_adv_tgt9,
2672                       args.lambda_adv_tgt10]
2673     mse_weights = [args.mse_weight6, args.mse_weight7, args.mse_weight8,
2674 args.mse_weight9,
2675                   args.mse_weight10]
2676
2677     # create a list of discriminators
2678     discriminators = []
2679     for dis_idx in range(n_discriminators):
2680         discriminators.append(FCDiscriminator(num_classes=args.num_classes))
2681         discriminators[dis_idx].train()
2682         discriminators[dis_idx].cuda(args.gpu)
2683
2684     if not os.path.exists(args.snapshot_dir):
2685         os.makedirs(args.snapshot_dir)
2686
2687     max_iters = args.num_steps * args.iter_size * args.batch_size
2688     src_set = REFUGE(True, domain='REFUGE_SRC', is_transform=True,
2689                     augmentations=aug_student, aug_for_target=aug_teacher,
2690 max_iters=max_iters)
2691     src_loader = data.DataLoader(src_set,
2692                                 batch_size=args.batch_size,
2693                                 shuffle=True,
2694                                 num_workers=args.num_workers,
2695                                 pin_memory=True)
2696
2697     src_loader_iter = enumerate(src_loader)
2698     tgt_set = REFUGE(True, domain='REFUGE_DST', is_transform=True,

```



```

2699         augmentations=aug_student, aug_for_target=aug_teacher,
2700         max_iters=max_iters)
2701     tgt_loader = data.DataLoader(tgt_set,
2702                                 batch_size=args.batch_size,
2703                                 shuffle=True,
2704                                 num_workers=args.num_workers,
2705                                 pin_memory=True)
2706
2707     tgt_loader_iter = enumerate(tgt_loader)
2708     student_optimizer = optim.SGD(student_params,
2709                                   lr=args.learning_rate,
2710                                   momentum=args.momentum,
2711                                   weight_decay=args.weight_decay)
2712     teacher_optimizer = optim_weight_ema.WeightEMA(
2713         teacher_params, student_params, alpha=args.teacher_alpha)
2714
2715     d_optimizers = []
2716     for idx in range(n_discriminators):
2717         optimizer = optim.Adam(discriminators[idx].parameters(),
2718                                lr=args.learning_rate_D,
2719                                betas=(0.9, 0.99))
2720         d_optimizers.append(optimizer)
2721
2722     calc_bce_loss = torch.nn.BCEWithLogitsLoss()
2723
2724     # labels for adversarial training
2725     source_label, tgt_label = 0, 1
2726     for i_iter in range(args.num_steps):
2727
2728         total_seg_loss = 0
2729         seg_loss_vals = [0] * n_discriminators
2730         adv_tgt_loss_vals = [0] * n_discriminators
2731         d_loss_vals = [0] * n_discriminators
2732         unsup_loss_vals = [0] * n_discriminators
2733
2734         for d_optimizer in d_optimizers:
2735             d_optimizer.zero_grad()
2736             adjust_learning_rate_D(d_optimizer, i_iter, args)
2737
2738         student_optimizer.zero_grad()
2739         adjust_learning_rate(student_optimizer, i_iter, args)
2740
2741         for sub_i in range(args.iter_size):
2742
2743             # ***** Optimize source network with segmentation loss
2744             *****
2745
2746             # As we don't change the discriminators, their parameters are
2747             fixed
2748             for discriminator in discriminators:
2749                 for param in discriminator.parameters():
2750                     param.requires_grad = False
2751
2752             _, src_batch = src_loader_iter.__next__()
2753             _, _, src_images, src_labels, _ = src_batch
2754             src_images = Variable(src_images).cuda(args.gpu)
2755
2756             # calculate the segmentation losses

```

```

2756         sup_preds = list(student_net(src_images))
2757         seg_losses, total_seg_loss = [], 0
2758         for idx, sup_pred in enumerate(sup_preds):
2759             sup_interp_pred = (sup_pred)
2760             # you also can use dice loss like: dice_loss(src_labels,
2761 sup_interp_pred)
2762             seg_loss = Weighted_Jaccard_loss(src_labels, sup_interp_pred,
2763 args.class_weights, args.gpu)
2764             seg_losses.append(seg_loss)
2765             total_seg_loss += seg_loss * unsup_weights[idx]
2766             seg_loss_vals[idx] += seg_loss.item() / args.iter_size
2767
2768         _, tgt_batch = tgt_loader_iter.__next__()
2769         tgt_images0, tgt_lbl0, tgt_images1, tgt_lbl1, _ = tgt_batch
2770         tgt_images0 = Variable(tgt_images0).cuda(args.gpu)
2771         tgt_images1 = Variable(tgt_images1).cuda(args.gpu)
2772
2773         # calculate ensemble losses
2774         stu_unsup_preds = list(student_net(tgt_images1))
2775         tea_unsup_preds = teacher_net(tgt_images0)
2776         total_mse_loss = 0
2777
2778
2779         for idx in range(n_discriminators):
2780             stu_unsup_probs = F.softmax(stu_unsup_preds[idx], dim=-1)
2781             tea_unsup_probs = F.softmax(tea_unsup_preds[idx], dim=-1)
2782
2783             unsup_loss = calc_mse_loss(stu_unsup_probs, tea_unsup_probs,
2784 args.batch_size)
2785             unsup_loss_vals[idx] += unsup_loss.item() / args.iter_size
2786             total_mse_loss += unsup_loss * mse_weights[idx]
2787
2788
2789         total_mse_loss = total_mse_loss / args.iter_size
2790
2791
2792         # As the requires_grad is set to False in the discriminator, the
2793         # gradients are only accumulated in the generator, the target
2794         # student network is optimized to make the outputs of target
2795 domain
2796         # images close to the outputs of source domain images
2797         stu_unsup_preds = list(student_net(tgt_images0))
2798         d_outs, total_adv_loss = [], 0
2799         for idx in range(n_discriminators):
2800             stu_unsup_interp_pred = (stu_unsup_preds[idx])
2801             d_outs.append(discriminators[idx](stu_unsup_interp_pred))
2802             label_size = d_outs[idx].data.size()
2803             labels = torch.FloatTensor(label_size).fill_(source_label)
2804             labels = Variable(labels).cuda(args.gpu)
2805             adv_tgt_loss = calc_bce_loss(d_outs[idx], labels)
2806             total_adv_loss += lambda_adv_tgts[idx] * adv_tgt_loss
2807             adv_tgt_loss_vals[idx] += adv_tgt_loss.item() /
2808 args.iter_size
2809
2810         total_adv_loss = total_adv_loss / args.iter_size
2811
2812

```

```

2813         # requires_grad is set to True in the discriminator, we only
2814         # accumulate gradients in the discriminators, the discriminators
2815         are
2816         # optimized to make true predictions
2817         d_losses = []
2818         for idx in range(n_discriminators):
2819             discriminator = discriminators[idx]
2820             for param in discriminator.parameters():
2821                 param.requires_grad = True
2822
2823             sup_preds[idx] = sup_preds[idx].detach()
2824             d_outs[idx] = discriminators[idx](sup_preds[idx])
2825
2826             label_size = d_outs[idx].data.size()
2827             labels = torch.FloatTensor(label_size).fill_(source_label)
2828             labels = Variable(labels).cuda(args.gpu)
2829
2830             d_losses.append(calc_bce_loss(d_outs[idx], labels))
2831             d_losses[idx] = d_losses[idx] / args.iter_size / 2
2832             d_losses[idx].backward()
2833             d_loss_vals[idx] += d_losses[idx].item()
2834
2835         for idx in range(n_discriminators):
2836             stu_unsup_preds[idx] = stu_unsup_preds[idx].detach()
2837             d_outs[idx] = discriminators[idx](stu_unsup_preds[idx])
2838
2839             label_size = d_outs[idx].data.size()
2840             labels = torch.FloatTensor(label_size).fill_(tgt_label)
2841             labels = Variable(labels).cuda(args.gpu)
2842
2843             d_losses[idx] = calc_bce_loss(d_outs[idx], labels)
2844             d_losses[idx] = d_losses[idx] / args.iter_size / 2
2845             d_losses[idx].backward()
2846             d_loss_vals[idx] += d_losses[idx].item()
2847
2848         for d_optimizer in d_optimizers:
2849             d_optimizer.step()
2850
2851
2852         total_loss = total_seg_loss + total_adv_loss + total_mse_loss
2853
2854         logger.log_value('total_seg_loss', total_seg_loss, i_iter)
2855         logger.log_value('adv_loss', total_adv_loss, i_iter)
2856         logger.log_value('mse_loss', total_mse_loss, i_iter)
2857         logger.log_value('target_loss', total_adv_loss + total_mse_loss,
2858 i_iter)
2859         logger.log_value('d_loss_0,', d_loss_vals[0], i_iter)
2860         logger.log_value('d_loss_1,', d_loss_vals[1], i_iter)
2861         logger.log_value('d_loss_2,', d_loss_vals[2], i_iter)
2862         logger.log_value('d_loss_3,', d_loss_vals[3], i_iter)
2863         logger.log_value('d_loss_4,', d_loss_vals[4], i_iter)
2864         logger.log_value('d_loss', np.mean(d_loss_vals[0] + d_loss_vals[1] +
2865 d_loss_vals[2] + d_loss_vals[3] + d_loss_vals[4]), i_iter)
2866
2867         total_loss.backward()
2868         student_optimizer.step()
2869         teacher_optimizer.step()

```

```

2870
2871
2872     log_str = 'iter = {0:7d}/{1:7d}'.format(i_iter, args.num_steps)
2873     log_str += ', total_seg_loss = {0:.3f} '.format(total_seg_loss)
2874     templ = 'seg_losses = [' + ', '.join(['%.2f'] * len(seg_loss_vals))
2875     log_str += templ % tuple(seg_loss_vals) + ']'
2876     templ = 'ens_losses = [' + ', '.join(['%.5f'] * len(unsup_loss_vals))
2877     log_str += templ % tuple(unsup_loss_vals) + ']'
2878     templ = 'adv_losses = [' + ', '.join(['%.2f'] *
2879 len(adv_tgt_loss_vals))
2880     log_str += templ % tuple(adv_tgt_loss_vals) + ']'
2881     templ = 'd_losses = [' + ', '.join(['%.2f'] * len(d_loss_vals))
2882     log_str += templ % tuple(d_loss_vals) + ']'
2883
2884     print(log_str)
2885     if i_iter >= args.num_steps_stop - 1:
2886         print('save model ...')
2887         filename = 'UNet' + str(args.num_steps_stop) +
2888 '_v18_weightedclass.pth'
2889         torch.save(teacher_net.cpu().state_dict(),
2890 os.path.join(args.snapshot_dir, filename))
2891         break
2892
2893     if i_iter % args.save_pred_every == 0 and i_iter != 0:
2894         print('taking snapshot ...')
2895         filename = 'UNet' + str(i_iter) + '_v18_weightedclass.pth'
2896         torch.save(teacher_net.cpu().state_dict(),
2897 os.path.join(args.snapshot_dir, filename))
2898         teacher_net.cuda(args.gpu)
2899
2900
2901 if __name__ == '__main__':
2902     main()
2903
2904 #Testing Code – predict.py
2905 import argparse
2906
2907 import numpy as np
2908
2909 from packaging import version
2910
2911 import os
2912 os.environ["CUDA_DEVICE_ORDER"]="PCI_BUS_ID"
2913 os.environ["CUDA_VISIBLE_DEVICES"]="2,3"
2914 from PIL import Image
2915 import matplotlib.pyplot as plt
2916 import cv2
2917 from skimage.transform import rotate
2918 import torch
2919 from torch.autograd import Variable
2920
2921 import torch.nn as nn
2922 from torch.utils import data
2923
2924 from models.unet import UNet
2925 from dataset.refuge import REFUGE

```

```

2925
2926 NUM_CLASSES = 3
2927 NUM_STEPS = 512 # Number of images in the validation set.
2928 RESTORE_FROM =
2929 '/home/charlietran/CADA_Tutorial/Model_Weights/Trial1/UNet1000_v18_weightedcl
2930 ass.pth'
2931 SAVE_PATH = '/home/charlietran/CADA_Tutorial/result/Trial1/'
2932 MODEL = 'Unet'
2933 BATCH_SIZE = 1
2934 is_polar = False #If need to transfer the image and labels to polar
2935 coordinates: MICCAI version is False
2936 ROI_size = 700 #ROI size
2937 from evaluation.evaluation_segmentation import *
2938
2939
2940 print(RESTORE_FROM)
2941
2942 palette=[
2943     255, 255, 255, # black background
2944     128, 128, 128, # index 1 is red
2945     0, 0, 0, # index 2 is yellow
2946     0, 0, 0 # index 3 is orange
2947 ]
2948
2949 zero_pad = 256 * 3 - len(palette)
2950 for i in range(zero_pad):
2951     palette.append(0)
2952
2953
2954 def colorize_mask(mask):
2955     # mask: numpy array of the mask
2956     new_mask = Image.fromarray(mask.astype(np.uint8)).convert('P')
2957     new_mask.putpalette(palette)
2958
2959     return new_mask
2960
2961 def get_arguments():
2962     """Parse all the arguments provided from the CLI.
2963     Returns:
2964         A list of parsed arguments.
2965     """
2966     parser = argparse.ArgumentParser(description="Unet Network")
2967     parser.add_argument("--model", type=str, default=MODEL,
2968                         help="Model Choice Unet.")
2969     parser.add_argument("--num-classes", type=int, default=NUM_CLASSES,
2970                         help="Number of classes to predict (including
2971 background).")
2972     parser.add_argument("--restore-from", type=str, default=RESTORE_FROM,
2973                         help="Where restore model parameters from.")
2974     parser.add_argument("--batch-size", type=int, default=BATCH_SIZE,
2975                         help="Number of images sent to the network in one
2976 step.")
2977     parser.add_argument("--gpu", type=int, default=0,
2978                         help="choose gpu device.")
2979     parser.add_argument("--save", type=str, default=SAVE_PATH,
2980                         help="Path to save result.")
2981     parser.add_argument("--is_polar", type=bool, default=False,

```

```

2982                                     help="If proceed images in polar coordinate. MICCAI
2983 version is false")
2984     parser.add_argument("--ROI_size", type=int, default=460,
2985                         help="Size of ROI.")
2986
2987     parser.add_argument('--t', type=int, default=3, help='t for Recurrent
2988 step of R2U_Net or R2AttU_Net')
2989
2990     return parser.parse_args()
2991
2992
2993 def main():
2994     """Create the model and start the evaluation process."""
2995
2996     args = get_arguments()
2997
2998     gpu0 = args.gpu
2999
3000     if not os.path.exists(args.save):
3001         os.makedirs(args.save)
3002
3003     model = UNet(3, n_classes=args.num_classes)
3004
3005     saved_state_dict = torch.load(args.restore_from)
3006     model.load_state_dict(saved_state_dict)
3007
3008     model.cuda(gpu0)
3009     model.train()
3010
3011     testloader = data.DataLoader(REFUGE(False, domain='REFUGE_TEST',
3012 is_transform=True),
3013                                 batch_size=args.batch_size,
3014 shuffle=False, pin_memory=True)
3015
3016
3017     if version.parse(torch.__version__) >= version.parse('0.4.0'):
3018         interp = nn.Upsample(size=(ROI_size, ROI_size), mode='bilinear',
3019 align_corners=True)
3020     else:
3021         interp = nn.Upsample(size=(ROI_size, ROI_size), mode='bilinear')
3022
3023     for index, batch in enumerate(testloader):
3024         if index % 100 == 0:
3025             print('%d processd' % index)
3026         image, label, _, _, name = batch
3027         if args.model == 'Unet':
3028             _, _, _, output2 = model(Variable(image,
3029 volatile=True).cuda(gpu0))
3030
3031             output = interp(output2).cpu().data.numpy()
3032
3033
3034     for idx, one_name in enumerate(name):
3035         pred = output[idx]
3036         pred = pred.transpose(1, 2, 0)
3037         pred = np.asarray(np.argmax(pred, axis=2), dtype=np.uint8)
3038         output_col = colorize_mask(pred)

```

```

3039
3040         print(output_col.size)
3041         one_name = one_name.split('/')[ -1]
3042         output_col = output_col.convert('L')
3043         output_col.save('%s/%s.bmp' % (args.save, one_name))
3044
3045
3046 if __name__ == '__main__':
3047     main()
3048     results_folder = SAVE_PATH
3049     gt_folder = '/DATA/charlie/AWC/CADA_Tutorial_Image/Target_Test/mask/'
3050     output_path = results_folder
3051     export_table = True
3052     evaluate_segmentation_results(results_folder, gt_folder, output_path,
3053 export_table)

3054 #YAML Packages – packages.yml
3055 name: cada_tutorial

3056 channels:

3057 - pytorch
3058 - anaconda
3059 - conda-forge
3060 - defaults

3061 dependencies:

3062 - _libgcc_mutex=0.1=main
3063 - alumentations=0.5.2=pyhd8ed1ab_0
3064 - attrs=20.3.0=pyhd3deb0d_0
3065 - backports=1.0=py_2
3066 - backports.functools_lru_cache=1.6.3=pyhd8ed1ab_0
3067 - blas=1.0=mkl
3068 - bleach=3.3.0=pyh44b312d_0
3069 - bzip2=1.0.8=h7b6447c_0
3070 - ca-certificates=2020.12.5=ha878542_0
3071 - cairo=1.16.0=hf32fb01_1
3072 - certifi=2020.12.5=py36h5fab9bb_1
3073 - cffi=1.14.5=py36h261ae71_0
3074 - cloudpickle=1.6.0=py_0
3075 - cudatoolkit=9.0=h13b8566_0

```

3076 - cudnn=7.1.2=cuda9.0_0
3077 - cycler=0.10.0=py36_0
3078 - cytoolz=0.11.0=py36h7b6447c_0
3079 - dask-core=2.30.0=py_0
3080 - decorator=4.4.2=py_0
3081 - defusedxml=0.7.1=pyhd8ed1ab_0
3082 - entrypoints=0.3=pyhd8ed1ab_1003
3083 - ffmpeg=4.0=hcdf2ecd_0
3084 - fontconfig=2.13.1=h6c09931_0
3085 - freeglut=3.0.0=hf484d3e_5
3086 - freetype=2.10.4=h5ab3b9f_0
3087 - geos=3.8.1=he1b5a44_0
3088 - glib=2.68.0=h36276a3_0
3089 - graphite2=1.3.14=h23475e2_0
3090 - harfbuzz=1.8.8=hffaf4a1_0
3091 - hdf5=1.10.2=hba1933b_1
3092 - icu=58.2=he6710b0_3
3093 - imageio=2.9.0=py_0
3094 - imgaug=0.4.0=py_1
3095 - importlib-metadata=3.8.0=py36h5fab9bb_0
3096 - intel-openmp=2020.2=254
3097 - ipykernel=5.5.0=py36he448a4c_1
3098 - ipython=5.8.0=py36_1
3099 - ipython_genutils=0.2.0=py_1
3100 - jasper=2.0.14=h07fcd6_1
3101 - jinja2=2.11.3=pyh44b312d_0
3102 - jpeg=9b=hbf39ab_1
3103 - jsonschema=3.2.0=pyhd8ed1ab_3
3104 - jupyter_client=6.1.12=pyhd8ed1ab_0
3105 - jupyter_core=4.7.1=py36h5fab9bb_0
3106 - kiwisolver=1.2.0=py36hfd86e86_0

3107 - lcms2=2.11=h396b838_0
3108 - ld_impl_linux-64=2.33.1=h53a641e_7
3109 - libffi=3.3=he6710b0_2
3110 - libgcc-ng=9.1.0=hdf63c60_0
3111 - libgfortran-ng=7.3.0=hdf63c60_0
3112 - libglu=9.0.0=hf484d3e_1
3113 - libopencv=3.4.2=hb342d67_1
3114 - libopus=1.3.1=h7b6447c_0
3115 - libpng=1.6.37=hbc83047_0
3116 - libsodium=1.0.18=h36c2ea0_1
3117 - libstdcxx-ng=9.1.0=hdf63c60_0
3118 - libtiff=4.1.0=h2733197_1
3119 - libuuid=1.0.3=h1bed415_2
3120 - libvpx=1.7.0=h439df22_0
3121 - libxcb=1.14=h7b6447c_0
3122 - libxml2=2.9.10=hb55368b_3
3123 - lz4-c=1.9.2=heb0550a_3
3124 - markupsafe=1.1.1=py36he6145b8_2
3125 - matplotlib-base=3.3.1=py36h817c723_0
3126 - mistune=0.8.4=py36h1d69622_1002
3127 - mkl=2020.2=256
3128 - mkl-service=2.3.0=py36he8ac12f_0
3129 - mkl_fft=1.3.0=py36h54f3939_0
3130 - mkl_random=1.1.1=py36h0573a6f_0
3131 - nbconvert=5.6.1=py36h9f0ad1d_1
3132 - nbformat=5.1.2=pyhd8ed1ab_1
3133 - ncurses=6.2=he6710b0_1
3134 - networkx=2.5=py_0
3135 - ninja=1.10.2=py36hff7bd54_0
3136 - notebook=5.7.10=py36h9f0ad1d_0
3137 - numpy-base=1.19.2=py36hfa32c7d_0

3138 - olefile=0.46=py36_0
3139 - opencv=3.4.2=py36h6fd60c2_1
3140 - openssl=1.1.1k=h27cfd23_0
3141 - packaging=20.9=pyh44b312d_0
3142 - pandoc=2.12=h7f98852_0
3143 - pandocfilters=1.4.2=py_1
3144 - pcre=8.44=he6710b0_0
3145 - pexpect=4.8.0=pyh9f0ad1d_2
3146 - pickleshare=0.7.5=py_1003
3147 - pillow=8.0.0=py36h9a89aac_0
3148 - pip=21.0.1=py36h06a4308_0
3149 - pixman=0.40.0=h7b6447c_0
3150 - prometheus_client=0.9.0=pyhd3deb0d_0
3151 - prompt_toolkit=1.0.15=py_1
3152 - ptyprocess=0.7.0=pyhd3deb0d_0
3153 - py-opencv=3.4.2=py36hb342d67_1
3154 - pycparser=2.20=py_2
3155 - pygments=2.8.1=pyhd8ed1ab_0
3156 - pyparsing=2.4.7=py_0
3157 - pyrsistent=0.17.3=py36h1d69622_1
3158 - python=3.6.13=hdb3f193_0
3159 - python-dateutil=2.8.1=py_0
3160 - python_abi=3.6=1_cp36m
3161 - pytorch=1.0.0=py3.6_cuda9.0.176_cudnn7.4.1_1
3162 - pywavelets=1.1.1=py36h7b6447c_2
3163 - pyzmq=19.0.2=py36h9947dbf_2
3164 - readline=8.1=h27cfd23_0
3165 - scikit-image=0.17.2=py36hdf5156a_0
3166 - send2trash=1.5.0=py_0
3167 - setuptools=52.0.0=py36h06a4308_0
3168 - shapely=1.7.1=py36h5d6357d_1

3169 - simplegeneric=0.8.1=py_1
3170 - six=1.15.0=py_0
3171 - sqlite=3.35.2=hdfb4753_0
3172 - terminado=0.9.3=py36h5fab9bb_0
3173 - testpath=0.4.4=py_0
3174 - tifffile=2020.10.1=py36hdd07704_2
3175 - tk=8.6.10=hbc83047_0
3176 - toolz=0.11.1=py_0
3177 - torchvision=0.2.1=py_2
3178 - tornado=6.0.4=py36h7b6447c_1
3179 - tqdm=4.59.0=pyhd3eb1b0_1
3180 - traitlets=4.3.3=py36h9f0ad1d_1
3181 - typing_extensions=3.7.4.3=py_0
3182 - wcwidth=0.2.5=pyh9f0ad1d_2
3183 - webencodings=0.5.1=py_1
3184 - wheel=0.36.2=pyhd3eb1b0_0
3185 - xz=5.2.5=h7b6447c_0
3186 - yaml=0.2.5=h7b6447c_0
3187 - zeromq=4.3.4=h2531618_0
3188 - zipp=3.4.1=pyhd8ed1ab_0
3189 - zlib=1.2.11=h7b6447c_3
3190 - zstd=1.4.4=h0b5b093_3
3191 - pip:
3192 - absl-py==0.12.0
3193 - astor==0.8.1
3194 - cached-property==1.5.2
3195 - chardet==4.0.0
3196 - config==0.5.0.post0
3197 - docopt==0.6.2
3198 - et-xmlfile==1.0.1
3199 - gast==0.4.0

3200 - google-pasta==0.2.0
3201 - grpcio==1.36.1
3202 - h5py==2.10.0
3203 - idna==2.10
3204 - keras==2.1.6
3205 - keras-applications==1.0.8
3206 - keras-preprocessing==1.1.2
3207 - markdown==3.3.4
3208 - numpy==1.19.5
3209 - openpyxl==3.0.7
3210 - pipreqs==0.4.10
3211 - protobuf==3.15.6
3212 - requests==2.25.1
3213 - scipy==1.2.2
3214 - tensorboard==1.12.2
3215 - tensorboard-logger==0.1.0
3216 - tensorflow-estimator==1.14.0
3217 - tensorflow-gpu==1.12.0
3218 - termcolor==1.1.0
3219 - urllib3==1.26.6
3220 - werkzeug==1.0.1
3221 - wrapt==1.12.1
3222 - yarg==0.1.9
3223 prefix: /home/charlietran/anaconda3/envs/da
3224