Weekly_Report_Alvin_20190131

Done List:

1. Write a label correction tool in python 3 for Machine-Aid Cognition (MAC).
2. Read the first 2 chapters of a book, ‘Clean Code’, given by Albert.

The Description of the first project in the list

> Background:

The labeled frames after the processing of MAC should be further ‘transformed’ to be the training materials of deep learning models.

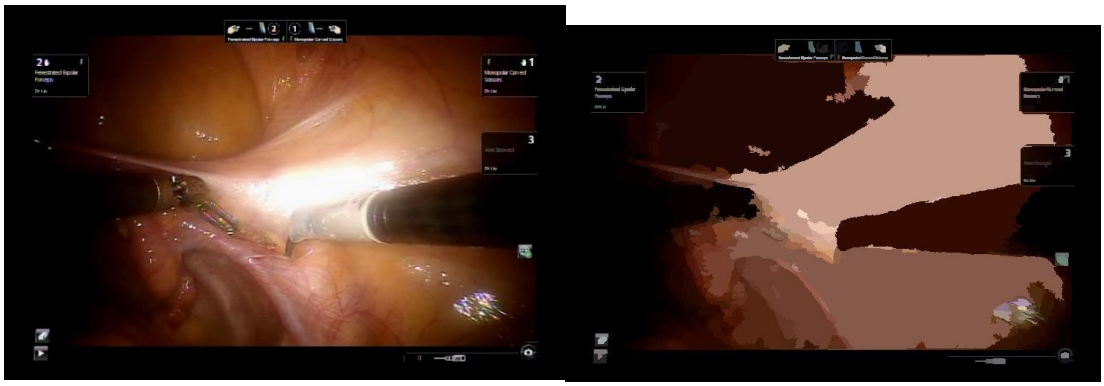
> Functionalities of this tool

1. Automatically read all the given labeled frames in the assigned directory.
2. Correct the labeling masks (.npy files) by the given mapping table (.json file).
3. Generate the corrected labeling masks (.npy files).
4. Overlay the new masks with the original frames correspondingly.

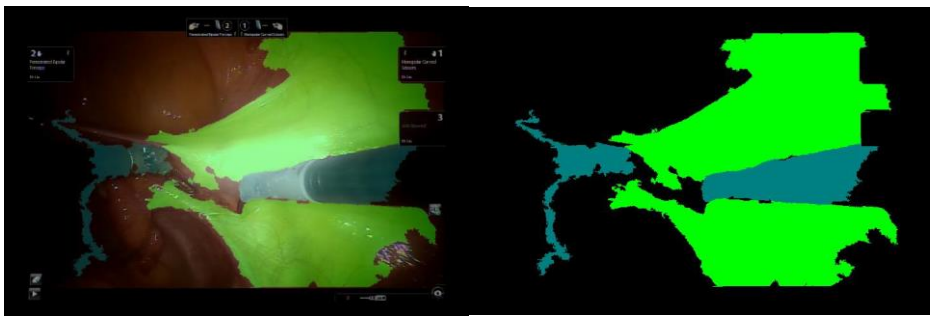
> Results

My code can run efficiently by averaging ~ 3.2s for doing the given 96 input files on my personal laptop.

Representative input files:



Corresponding results:



```
import json
import os
import fnmatch
import numpy as np
from PIL import Image
import matplotlib.cm as cm

##### Preface #####

# This tool is created by Alvin Pei Yan, Li, whose email addresses are Alvin.Li@acer.com and d05548014@ntu.edu.tw.
# Please contact to him, the author, if  any problems or requirments has been found or asked.
# 2019.01.29 for the 1st version.

##### Function Definitions #####

def  main():
    #
    # this is the main function handling all sub-functions together
    #
    print('Start.\n')

    global_IDtable = get_global_table_as_reference()
    namelist_local_table = get_a_list_of_local_tables_for_correction()
    do_correction_for_local_lablefiles(namelist_local_table, global_IDtable)

    print('\nDone.')

def  get_global_table_as_reference():
    #
    # get the correction table for correcting labels and return it as a global_IDtable
    # the name: label_to_id
    # the suffix: .json
    # the format: hash table
    #
    with open('label_to_id.json', encoding='utf-8-sig') as json_file:
        print('label_to_id.json has been read.')

        global_IDtable = json.load(json_file)
```

```
    return global_IDtable


def get_a_list_of_local_tables_for_correction():
    #
    # get a list of the local tables stored the wrong keys that should be corrected and return it as namelist_local_table
    # whereis? : 'labels/'
    # the name: DaVincixxxxxxx_y_z; x: Date, e.g. 20181213; y:a number with several digits; z: 0 typically
    # the suffix: .json
    # the format: hash table
    #
    os.chdir('labels/')

    namelist_local_table = []
    for file in os.listdir('.'):
        if fnmatch.fnmatch(file,'*.json'):
            print('{} has been found.'.format(file))


            namelist_local_table.append(str(file))

    os.chdir('..')
    return namelist_local_table


def do_correction_for_local_lablefiles(namelist_local_table, global_IDtable):
    #
    # do the label correction for all the label files in labels/ and store the corrected files in the subdir correct/
    #
    def submain():
        #
        # realize the functionality of 'do_correction_for_local_lablefiles'
        #
        os.chdir('labels/')
        os.mkdir('correct')

        for filename in namelist_local_table:
            local_label_filename = filename.replace(".json","")
            local_label_arr = np.load(local_label_filename + '.npy')
            with open(filename, encoding='utf-8-sig') as json_file:
                local_label_table = json.load(json_file)
```

```
                corrected_label_arr = do_for_single_npyfile(local_label_arr, local_label_filename, local_label_table,
global_IDtable)
                name_of_new_seg_im = generate_corrected_seg(corrected_label_arr, local_label_filename, global_IDtable)
                generate_overlayimage(name_of_new_seg_im, local_label_filename)

        os.chdir('..')

    def do_for_single_npyfile(local_label_arr, local_label_filename, local_label_table, global_IDtable):
        #
        # do the correction mapping for each local .npy label file
        #

        corrected_label_arr = np.zeros(local_label_arr.shape)

        for key in list(local_label_table.keys()):
            corrected_label_arr[local_label_arr == int(key)] = global_IDtable[local_label_table[key]]

        new_label_filename = local_label_filename.replace(".npy", "") + '_n'

        np.save('correct/' + new_label_filename, corrected_label_arr)

        print('{} has been saved.'.format(new_label_filename))

        return corrected_label_arr

    def generate_corrected_seg(corrected_label_arr, local_label_filename, global_IDtable):
        #
        # generate a corrected segamenation mask
        #

        #normalized_label_arr = cm.nipy_spectral(corrected_label_arr)
        #normalized_label_arr *= 255.0/normalized_label_arr.max()

        #new_seg_im = Image.fromarray(np.uint8(normalized_label_arr))

        color_dic = { 'Black' : [0, 0, 0],
                      'White' : [255, 255, 255],
                      'Medium Gray' : [128, 128, 128],
                      'Aqua' : [0, 128, 128],
```

```
                    'Navy Blue' : [0, 0, 128],
                    'Green' : [0, 255, 0],
                    'Orange' : [255, 165, 0],
                    'Yellow' : [255, 255, 0],
                    'Maroon' : [128, 0, 0]
                }

        #new_seg_im = Image.fromarray(corrected_label_arr).convert('RGB')

        frameshape = list(corrected_label_arr.shape)
        colored_new_seg_im = np.zeros((frameshape[0], frameshape[1], 3))

        stacked_for_boolean_masks = np.stack((corrected_label_arr,corrected_label_arr,corrected_label_arr), axis = 2)

        inform_Background = int(global_IDtable['Background'])
        inform_Peritoneum = int(global_IDtable['Peritoneum'])
        inform_Ovary = int(global_IDtable['Ovary'])
        inform_Uterus = int(global_IDtable['Uterus'])
        inform_Fallopian_Tube = int(global_IDtable['Fallopian_Tube'])
        inform_Ligament = int(global_IDtable['Ligament'])
        inform_Ureter = int(global_IDtable['Ureter'])
        inform_Artery = int(global_IDtable['Artery'])
        inform_Scapel = int(global_IDtable['Scapel'])

        Background = (stacked_for_boolean_masks == [inform_Background, inform_Background,
inform_Background]).all(axis = 2)
        Peritoneum = (stacked_for_boolean_masks == [inform_Peritoneum, inform_Peritoneum,
inform_Peritoneum]).all(axis = 2)
        Ovary = (stacked_for_boolean_masks == [inform_Ovary, inform_Ovary, inform_Ovary]).all(axis = 2)
        Uterus = (stacked_for_boolean_masks == [inform_Uterus, inform_Uterus, inform_Uterus]).all(axis = 2)
        Fallopian_Tube = (stacked_for_boolean_masks == [inform_Fallopian_Tube, inform_Fallopian_Tube,
inform_Fallopian_Tube]).all(axis = 2)
        Ligament = (stacked_for_boolean_masks == [inform_Ligament, inform_Ligament, inform_Ligament]).all(axis = 2)
        Ureter = (stacked_for_boolean_masks == [inform_Ureter, inform_Ureter, inform_Ureter]).all(axis = 2)
        Artery = (stacked_for_boolean_masks == [inform_Artery, inform_Artery, inform_Artery]).all(axis = 2)
        Scapel = (stacked_for_boolean_masks == [inform_Scapel, inform_Scapel, inform_Scapel]).all(axis = 2)

        colored_new_seg_im[Background] = color_dic['Black']
        colored_new_seg_im[Peritoneum] = color_dic['Maroon']
```

```
        colored_new_seg_im[Ovary] = color_dic['Medium Gray']
        colored_new_seg_im[Uterus] = color_dic['White']
        colored_new_seg_im[Fallopian_Tube] = color_dic['Navy Blue']
        colored_new_seg_im[Ligament] = color_dic['Green']
        colored_new_seg_im[Ureter] = color_dic['Orange']
        colored_new_seg_im[Artery] = color_dic['Yellow']
        colored_new_seg_im[Scapel] = color_dic['Aqua']

        name_of_new_seg_im = local_label_filename + '_n_seg.jpg'

        new_seg_im = Image.fromarray(np.uint8(colored_new_seg_im)).convert('RGB')

        new_seg_im.save('correct/' + name_of_new_seg_im)

        #new_seg_im.convert('RGB').save('correct/' + name_of_new_seg_im)

        print('{} has been saved.'.format(name_of_new_seg_im))

        return name_of_new_seg_im

def  generate_overlayimage(name_of_new_seg_im, local_label_filename):
        #
        # generate an image overlaying the original image with the corresponding mask
        #
        name_of_original_image = local_label_filename + '.jpg'

        original_im = Image.open(name_of_original_image)

        os.chdir('correct/')

        new_seg_im = Image.open(name_of_new_seg_im)
        original_im = original_im.convert('RGB')
        new_seg_im = new_seg_im.convert('RGB')

        overlay_image = Image.blend(original_im, new_seg_im, 0.4)      # 0.4 is the transparent rate.

        name_of_overlay_image = local_label_filename + '_n_overlay.jpg'
        overlay_image.convert('RGB').save(name_of_overlay_image)
```

```
            print('{} has been saved.'.format(name_of_overlay_image))

            os.chdir('..')

        submain()

##### Execution #####

import time
start = time. time()

main()

end = time. time()

duration = end - start

print('\nThis code runs so fast that only spends {} in second.'.format(duration))
```