# Excelsior v1.0

# by Phillip Stewart and Philip Tanwongprasert

# Excelsior converts scanned images of spreadsheets to excel spreadsheets

# Output is actually in .csv format, which may be imported by any

# spreadsheet program and many standard format readers.

# This script requires OpenCV and Tesseract for python

# This script may be imported as a library of functions for easier use

# of the OCR elements of tesseract and other functionality.

import numpy

import cv2

import cv2.cv as cv

import tesseract

import sys

# lower\_lim and upper\_lim are used for the range of cell sizes by area

# We found on different scans of spreadsheets, cells were usually about 25x100 pixels

# 800 and 5000 chosen for ease, the range should cover standard sized cells

# For higher res scans or large cells, these numbers may be modified.

lower\_lim = 800

upper\_lim = 5000

def prepare\_image(filename):

"""Reads an image and prepares it for OCR

Arguments:

filename (str) location of image file

Returns:

tuple (bw, threshold)

bw: black and white (binary) thresholded cv2 image

threshold: binary inverse thresholded cv2 image"""

image = cv2.imread(filename)

gray = cv2.cvtColor(image,cv2.COLOR\_BGR2GRAY)

\_,bw = cv2.threshold(gray,230,255,cv2.THRESH\_BINARY)

\_,threshold = cv2.threshold(gray,230,255,cv2.THRESH\_BINARY\_INV)

return bw, threshold

def find\_cells(image):

"""Finds the rectangle cells in an image

Arguments:

image (cv2Image) binary cv2 image

Returns:

cells (list of tuples) coordinates of cells in image"""

contours, \_ = cv2.findContours(image, cv2.RETR\_LIST, cv2.CHAIN\_APPROX\_SIMPLE)

cells = []

for contour in contours:

area = cv2.contourArea(contour)

if lower\_lim < area < upper\_lim:

# If cell sizes are similar to other contours or if cell sizes vary wildly,

# these lines will limit cells to contours that are box shaped.

# perim = cv2.arcLength(contour, True)

# points = cv2.approxPolyDP(contour, 0.02\*perim, True)

# if len(points) == 4:

# cells.append(cv2.boundingRect(contour))

cells.append(cv2.boundingRect(contour))

return (cells)

def start\_tess\_api():

"""Initializes a tesseract engine

Returns: tessapi (tesseract.API)"""

tess\_api = tesseract.TessBaseAPI()

tess\_api.SetOutputName("outputName");

tess\_api.Init(".","eng",tesseract.OEM\_DEFAULT)

tess\_api.SetPageSegMode(tesseract.PSM\_AUTO)

return tess\_api

def end\_tess\_api(tess\_api):

"""Closes the tesseract engine"""

tess\_api.End()

def ocr\_text(tess\_api, image):

"""Reads text from binary cv2 image

Arguments:

tess\_api (tesseract.API) an initialized tesseract engine

image (cv2Image) binary thresholded image for OCR

Returns:

img\_text (str) text extracted from image"""

tesseract.SetCvImage(image, tess\_api)

img\_text = tess\_api.GetUTF8Text()

return img\_text

def main(args):

infile = ''

outfile = ''

if len(args) != 3:

print ' Usage: $ ' + args[0] + ' <image input> <output csv>'

else:

infile = args[1]

outfile = args[2]

try:

bw\_image, prepped\_image = prepare\_image(infile)

except:

print 'Unable to read input file ' + args[1]

cells = find\_cells(prepped\_image)

cells.sort(key=lambda x: (x[1], x[0]))

t\_api = start\_tess\_api()

csv = []

curr\_pix\_row = cells[0][0]

for cell in cells:

x1, y1, x2, y2 = cell[0], cell[1], cell[0]+cell[2], cell[1]+cell[3]

cell\_image = bw\_image[y1:y2, x1:x2].copy()

comma = ','

if x1 > curr\_pix\_row:

curr\_pix\_row = x1

csv.append('\n')

comma = ''

csv.append(ocr\_text(t\_api, cell\_image) + comma)

end\_tess\_api(t\_api)

try:

with open(outfile, "w") as f:

f.write(','.join(csv))

except:

print 'Unable to write to output file ' + args[2]

#import protection:

# main() will not begin execution unless script is the main module.

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

main(sys.argv)