# author: Adrian Rosebrock

# website: http://www.pyimagesearch.com

# import the necessary packages

import numpy as np

import cv2

import sys

# import any special Python 2.7 packages

if sys.version\_info.major == 2:

from urllib import urlopen

# import any special Python 3 packages

elif sys.version\_info.major == 3:

from urllib.request import urlopen

def translate(image, x, y):

# define the translation matrix and perform the translation

M = np.float32([[1, 0, x], [0, 1, y]])

shifted = cv2.warpAffine(image, M, (image.shape[1], image.shape[0]))

# return the translated image

return shifted

def rotate(image, angle, center=None, scale=1.0):

# grab the dimensions of the image

(h, w) = image.shape[:2]

# if the center is None, initialize it as the center of

# the image

if center is None:

center = (w // 2, h // 2)

# perform the rotation

M = cv2.getRotationMatrix2D(center, angle, scale)

rotated = cv2.warpAffine(image, M, (w, h))

# return the rotated image

return rotated

def rotate\_bound(image, angle):

# grab the dimensions of the image and then determine the

# center

(h, w) = image.shape[:2]

(cX, cY) = (w / 2, h / 2)

# grab the rotation matrix (applying the negative of the

# angle to rotate clockwise), then grab the sine and cosine

# (i.e., the rotation components of the matrix)

M = cv2.getRotationMatrix2D((cX, cY), -angle, 1.0)

cos = np.abs(M[0, 0])

sin = np.abs(M[0, 1])

# compute the new bounding dimensions of the image

nW = int((h \* sin) + (w \* cos))

nH = int((h \* cos) + (w \* sin))

# adjust the rotation matrix to take into account translation

M[0, 2] += (nW / 2) - cX

M[1, 2] += (nH / 2) - cY

# perform the actual rotation and return the image

return cv2.warpAffine(image, M, (nW, nH))

def resize(image, width=None, height=None, inter=cv2.INTER\_AREA):

# initialize the dimensions of the image to be resized and

# grab the image size

dim = None

(h, w) = image.shape[:2]

# if both the width and height are None, then return the

# original image

if width is None and height is None:

return image

# check to see if the width is None

if width is None:

# calculate the ratio of the height and construct the

# dimensions

r = height / float(h)

dim = (int(w \* r), height)

# otherwise, the height is None

else:

# calculate the ratio of the width and construct the

# dimensions

r = width / float(w)

dim = (width, int(h \* r))

# resize the image

resized = cv2.resize(image, dim, interpolation=inter)

# return the resized image

return resized

def skeletonize(image, size, structuring=cv2.MORPH\_RECT):

# determine the area (i.e. total number of pixels in the image),

# initialize the output skeletonized image, and construct the

# morphological structuring element

area = image.shape[0] \* image.shape[1]

skeleton = np.zeros(image.shape, dtype="uint8")

elem = cv2.getStructuringElement(structuring, size)

# keep looping until the erosions remove all pixels from the

# image

while True:

# erode and dilate the image using the structuring element

eroded = cv2.erode(image, elem)

temp = cv2.dilate(eroded, elem)

# subtract the temporary image from the original, eroded

# image, then take the bitwise 'or' between the skeleton

# and the temporary image

temp = cv2.subtract(image, temp)

skeleton = cv2.bitwise\_or(skeleton, temp)

image = eroded.copy()

# if there are no more 'white' pixels in the image, then

# break from the loop

if area == area - cv2.countNonZero(image):

break

# return the skeletonized image

return skeleton

def opencv2matplotlib(image):

# OpenCV represents images in BGR order; however, Matplotlib

# expects the image in RGB order, so simply convert from BGR

# to RGB and return

return cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

def url\_to\_image(url, readFlag=cv2.IMREAD\_COLOR):

# download the image, convert it to a NumPy array, and then read

# it into OpenCV format

resp = urlopen(url)

image = np.asarray(bytearray(resp.read()), dtype="uint8")

image = cv2.imdecode(image, readFlag)

# return the image

return image

def auto\_canny(image, sigma=0.33):

# compute the median of the single channel pixel intensities

v = np.median(image)

# apply automatic Canny edge detection using the computed median

lower = int(max(0, (1.0 - sigma) \* v))

upper = int(min(255, (1.0 + sigma) \* v))

edged = cv2.Canny(image, lower, upper)

# return the edged image

return edged

def grab\_contours(cnts):

# if the length the contours tuple returned by cv2.findContours

# is '2' then we are using either OpenCV v2.4, v4-beta, or

# v4-official

if len(cnts) == 2:

cnts = cnts[0]

# if the length of the contours tuple is '3' then we are using

# either OpenCV v3, v4-pre, or v4-alpha

elif len(cnts) == 3:

cnts = cnts[1]

# otherwise OpenCV has changed their cv2.findContours return

# signature yet again and I have no idea WTH is going on

else:

raise Exception(("Contours tuple must have length 2 or 3, "

"otherwise OpenCV changed their cv2.findContours return "

"signature yet again. Refer to OpenCV's documentation "

"in that case"))

# return the actual contours array

return cnts

def is\_cv2(or\_better=False):

# grab the OpenCV major version number

major = get\_opencv\_major\_version()

# check to see if we are using \*at least\* OpenCV 2

if or\_better:

return major >= 2

# otherwise we want to check for \*strictly\* OpenCV 2

return major == 2

def is\_cv3(or\_better=False):

# grab the OpenCV major version number

major = get\_opencv\_major\_version()

# check to see if we are using \*at least\* OpenCV 3

if or\_better:

return major >= 3

# otherwise we want to check for \*strictly\* OpenCV 3

return major == 3

def is\_cv4(or\_better=False):

# grab the OpenCV major version number

major = get\_opencv\_major\_version()

# check to see if we are using \*at least\* OpenCV 4

if or\_better:

return major >= 4

# otherwise we want to check for \*strictly\* OpenCV 4

return major == 4

def get\_opencv\_major\_version(lib=None):

# if the supplied library is None, import OpenCV

if lib is None:

import cv2 as lib

# return the major version number

return int(lib.\_\_version\_\_.split(".")[0])

def check\_opencv\_version(major, lib=None):

# this function may be removed in a future release as we now

# use the get\_opencv\_major\_function to obtain the current OpenCV

# version and then perform the actual version check \*within\* the

# respective function

import warnings

message = """

The check\_opencv\_version function is deprecated and may be

removed in a future release. Use at your own risk.

"""

warnings.warn(message, DeprecationWarning, stacklevel=2)

# if the supplied library is None, import OpenCV

if lib is None:

import cv2 as lib

# return whether or not the current OpenCV version matches the

# major version number

return lib.\_\_version\_\_.startswith(major)

def build\_montages(image\_list, image\_shape, montage\_shape):

"""

---------------------------------------------------------------------------------------------

author: Kyle Hounslow

---------------------------------------------------------------------------------------------

Converts a list of single images into a list of 'montage' images of specified rows and columns.

A new montage image is started once rows and columns of montage image is filled.

Empty space of incomplete montage images are filled with black pixels

---------------------------------------------------------------------------------------------

:param image\_list: python list of input images

:param image\_shape: tuple, size each image will be resized to for display (width, height)

:param montage\_shape: tuple, shape of image montage (width, height)

:return: list of montage images in numpy array format

---------------------------------------------------------------------------------------------

example usage:

# load single image

img = cv2.imread('lena.jpg')

# duplicate image 25 times

num\_imgs = 25

img\_list = []

for i in xrange(num\_imgs):

img\_list.append(img)

# convert image list into a montage of 256x256 images tiled in a 5x5 montage

montages = make\_montages\_of\_images(img\_list, (256, 256), (5, 5))

# iterate through montages and display

for montage in montages:

cv2.imshow('montage image', montage)

cv2.waitKey(0)

----------------------------------------------------------------------------------------------

"""

if len(image\_shape) != 2:

raise Exception('image shape must be list or tuple of length 2 (rows, cols)')

if len(montage\_shape) != 2:

raise Exception('montage shape must be list or tuple of length 2 (rows, cols)')

image\_montages = []

# start with black canvas to draw images onto

montage\_image = np.zeros(shape=(image\_shape[1] \* (montage\_shape[1]), image\_shape[0] \* montage\_shape[0], 3),

dtype=np.uint8)

cursor\_pos = [0, 0]

start\_new\_img = False

for img in image\_list:

if type(img).\_\_module\_\_ != np.\_\_name\_\_:

raise Exception('input of type {} is not a valid numpy array'.format(type(img)))

start\_new\_img = False

img = cv2.resize(img, image\_shape)

# draw image to black canvas

montage\_image[cursor\_pos[1]:cursor\_pos[1] + image\_shape[1], cursor\_pos[0]:cursor\_pos[0] + image\_shape[0]] = img

cursor\_pos[0] += image\_shape[0] # increment cursor x position

if cursor\_pos[0] >= montage\_shape[0] \* image\_shape[0]:

cursor\_pos[1] += image\_shape[1] # increment cursor y position

cursor\_pos[0] = 0

if cursor\_pos[1] >= montage\_shape[1] \* image\_shape[1]:

cursor\_pos = [0, 0]

image\_montages.append(montage\_image)

# reset black canvas

montage\_image = np.zeros(shape=(image\_shape[1] \* (montage\_shape[1]), image\_shape[0] \* montage\_shape[0], 3),

dtype=np.uint8)

start\_new\_img = True

if start\_new\_img is False:

image\_montages.append(montage\_image) # add unfinished montage

return image\_montages

def adjust\_brightness\_contrast(image, brightness=0., contrast=0.):

"""

Adjust the brightness and/or contrast of an image

:param image: OpenCV BGR image

:param contrast: Float, contrast adjustment with 0 meaning no change

:param brightness: Float, brightness adjustment with 0 meaning no change

"""

beta = 0

# See the OpenCV docs for more info on the `beta` parameter to addWeighted

# https://docs.opencv.org/3.4.2/d2/de8/group\_\_core\_\_array.html#gafafb2513349db3bcff51f54ee5592a19

return cv2.addWeighted(image,

1 + float(contrast) / 100.,

image,

beta,

float(brightness))