photosbyjeremy / **img_qc_workshop**

Branch: master ▾   img_qc_workshop / img_qc / **img_qc.py**   Find file   Copy path

photosbyjeremy Add files via upload   1c140e1 on May 9

1 contributor

```
410 lines (304 sloc)   13.4 KB
```

```
# img_qc.py jeremy.moore@txstate.edu under the GNU GPL License:
# https://github.com/photosbyjeremy/img_qc_workshop/blob/master/LICENSE

### 2018-05 v0.90 ready for TCDL 2018 workshop: Introduction to Image
###               Processing with Python and Jupyter Notebooks
### 2018-02 v0.10 base code copied out of notebooks to import as functions

############################### TODO: ###############################
# 1. docstrings
# 2. logging
# 3. have separate sections for Pillow and OpenCV functions
#####################################################################

# === importing

# built--in
import logging
import os
from pathlib import Path

# 3rd party
import cv2
import img_qc.exiftool as exiftool
import ipywidgets as widgets
import numpy as np
import pandas as pd
from IPython.display import display
from PIL import Image
from scipy.spatial import distance as dist


# === functions in alphabetical order
def get_formatted_extension(from_extension):
    """-- Purpose --
    Return an extension with a period at the front

    -- Arguments --
    from_extension: file extension with or without a '.'

    -- Returns --
    formatted_extension: type=string; formatted extension"""

    if from_extension.startswith('.'):

        # currently we don't do anything
        formatted_extension = from_extension

    else:  # add a period
        formatted_extension = ('.' + from_extension)

    return formatted_extension


def get_rotated_cv_image(from_image, angle, center=None, scale=1.0):
    """-- Purpose --
    Rotate image with OpenCV

    -- Arguments --
    from_image: 2D numpy array
    angle: rotation angle in degrees; positive angle = clockwise direction

    -- Returns --
    rotated_image: 2D numpy array

    -- Source --
    https://www.pyimagesearch.com/2017/01/02/rotate-images-correctly-with-opencv-and-python/"""

    # get the image's dimensions and determine the center
    (height, width) = from_image.shape[:2]

    # if center is None then initialize the center of the image
    if center is None:
        (centerX, centerY) = (width // 2, height // 2)
    else:
        (centerX, centerY) = center

    # get the rotation matrix
    # -- NOTE: GET NEGATIVE ANGLE TO ROTATE CLOCKWISE
    matrix = cv2.getRotationMatrix2D((centerX, centerY), -angle, 1.0)

    # get sine and cosine
    # -- the rotation components of the matrix
    cosine = np.abs(matrix[0, 0])
    sin = np.abs(matrix[0, 1])

    # compute the new bounding dimensions of the image
    new_width = int((height * sin) + (width * cosine))
    new_height = int((height * cosine) + (width * sin))

    # adjust the rotation matrix for changes in height and width
    matrix[0, 2] += (new_width / 2) - centerX
    matrix[1, 2] += (new_height / 2) - centerY

    # rotate the image
    rotated_image = cv2.warpAffine(from_image, matrix, (new_width, new_height))

    # return the image
    return rotated_image


def check_directory_for_one_number_per_file(in_directory,
                                            with_extension,
                                            zeropad=4,
                                           ):
    """-- Purpose --
    Check that directory contains 1 tif per number in the form:
    <directory_name>_0001.tif
    <directory_name>_nnnn.tif

    -- Arguments --
    in_directory: the directory to check for one image per name
    with_extensions: takes a list of extensions; must be 1+
    zeropad: number of digits to zeropad

    -- Returns --
    TODO"""

    # get Path to directory
    directory_to_check = Path(in_directory)

    if directory_to_check.is_dir():

        # get directory name by indexing the last part of the Path before the name
        directory_name = directory_to_check.parts[-1]

        # Log the directory name
        logging.info(f'... checking {directory_name} . . .')

        # create list of all files in directory
        directory_contents_list = list(directory_to_check.glob('**/*'))

        # get formatted extension
        extension = get_formatted_extension(with_extension)

        # get list of everything NOT ending with extension
        # # except: Thumbs.db
        # #         .DS_Store
        unexpected_files_list = [x for x in directory_contents_list if
                                 not str(x).endswith(extension) and
                                 not str(x).endswith('Thumbs.db') and  # Windows file
                                 not str(x).endswith('.DS_Store')  # Mac file
                                ]

        # if we have ANYTHING unexpected
        if len(unexpected_files_list) > 0:
            # log a warning displaying the extension and the unexpected files list
            logging.warning(f"***UNEXPECTED NOT ENDING IN '{extension}': {unexpected_files_list}")
            # MINOR ERROR: extra files in the directory (not in extension list)
            # this could be where we move this directory into a new location
            # or create/add to a problems.txt for tracking

        # get a sorted list of all files that end in extension in the directory
        file_list = sorted([x for x in directory_contents_list if str(x).endswith(extension)])

        # get number of files in directory
        number_of_files = len(file_list)

        # check that each possible files is correctly numbered starting with 1
        for number in range(number_of_files):

            number += 1  # so we start counting files at 1 instead of 0

            # create file name in form <directory_name>_<zeropad><n>.suffix
            file_name_to_test = directory_name + '_' + str(number).zfill(zeropad) + extension

            # create file path
            file_path_to_test = Path(in_directory).joinpath(file_name_to_test)

            if file_path_to_test.is_file():
                # can process image futher here if necessary
                pass
            else:
                # log error
                logging.error(f'something BROKE: {file_path_to_test}')
                # MAJOR ERROR: extra/missing files in the folder with correct extension
                # this could be where we move this folder into a new location
                # or create/add to a problems.txt for tracking
    # else we error
    else:
        logging.error(f'NOT A DIRECTORY: {in_directory}')


def get_exiftool_metadata(in_directory, with_extension):
    """-- Purpose --
    load exiftool metadata using pyexiftool

    -- Arguments --
    directory_path: Path to directory of Document scans to load metadata
    with_extension: file extension
```

```python

    -- Returns --
    exiftool_metadata"""

    file_directory_path = Path(in_directory)

    extension = get_formatted_extension(with_extension)

    posix_file_list = list(file_directory_path.glob('**/*' + extension))

    sorted_file_list = sorted([str(x) for x in posix_file_list if x.exists()])

    with exiftool.ExifTool() as et:
        exiftool_metadata = et.get_metadata_batch(sorted_file_list)

    return exiftool_metadata


def get_images_df(in_directory, with_extension):
    """--- Purpose ---
    1. Get metadata for all files with_extension in_directory with ExifTool (including sub-folders)
    2. Return dataframe for further analysis

    --- Arguments ---
    in_directory: directory to start looking for images with_extension
    with_extension: image extension to search in_directory for
    --- Return ---
    images_dataframe: pandas dataframe including all images with_extension"""

    # get metadata using pyexiftool
    exiftool_metadata = get_exiftool_metadata(in_directory, with_extension)

    # convert exiftool metadata into pandas DataFrame
    images_df = pd.DataFrame(exiftool_metadata)

    logging.info(f'directory: {in_directory}')
    logging.info(f'number of images: {len(images_df)}')

    return images_df


def get_resized_cv_image(from_image, width=None, height=None, inter=cv2.INTER_AREA):
    # initialize the dimensions of the image to be resized and
    # grab the image size
    dimensions = None
    (h, w) = from_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 from_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)
        dimensions = (int(w * r), height)

    # otherwise, the height is None
    else:
        # calculate the ratio of the width and construct the
        # dimensions
        r = width / float(w)
        dimensions = (width, int(h * r))

    # resize the image
    resized = cv2.resize(from_image, dimensions, interpolation=inter)

    # return the resized image
    return resized


def get_image_resized_pillow(image, width=None, height=None, resample=Image.LANCZOS):
    # initialize the dimensions of the image to be resized and
    # grab the image size
    dimensions = None
    (starting_width, starting_height) = image.size

    # 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
        ratio = int(height) / float(starting_height)
        dimensions = (int(starting_width * ratio)), int(height)

    # otherwise, the height is None
    else:
        # calculate the ratio of the width and construct the
        # dimensions
        ratio = int(width) / float(starting_width)
        dimensions = (int(width), int(starting_height * ratio))

    # resize the image
    image_resized = image.resize(dimensions, resample=resample)

    # return the resized image
    return image_resized


def open_cv2_image(image_path):
    """--- Purpose ---
    TODO

    --- Arguments ---
    TODO

    --- Returns ---
    TODO"""

    # set filepath to string
    image_path = str(image_path)

    image = cv2.imread(image_path)
    return image


def order_points(pts):
    # sort the points based on their x-coordinates
    xSorted = pts[np.argsort(pts[:, 0]), :]

    # grab the left-most and right-most points from the sorted
    # x-coordinate points
    leftMost = xSorted[:2, :]
    rightMost = xSorted[2:, :]

    # now, sort the left-most coordinates according to their
    # y-coordinates so we can grab the top-left and bottom-left
    # points, respectively
    leftMost = leftMost[np.argsort(leftMost[:, 1]), :]
    (tl, bl) = leftMost

    # now that we have the top-left coordinate, use it as an
    # anchor to calculate the Euclidean distance between the
    # top-left and right-most points; by the Pythagorean
    # theorem, the point with the largest distance will be
    # our bottom-right point
    D = dist.cdist(tl[np.newaxis], rightMost, "euclidean")[0]
    (br, tr) = rightMost[np.argsort(D)[::-1], :]

    # return the coordinates in top-left, top-right,
    # bottom-right, and bottom-left order
    return np.array([tl, tr, br, bl], dtype="float32")


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, center=None, scale=1.0):
    # grab the dimensions of the image and then determine the
    # center
    (height, width) = image.shape[:2]

    # if the center is None, initialize it as the center of
    # the image
    if center is None:
        centerX = (w // 2)
        centerY = (h // 2)
    else:
        centerX, centerY = center

    # 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((centerX, centerY), -angle, scale)
    cos = np.abs(M[0, 0])
    sin = np.abs(M[0, 1])

    # compute the new bounding dimensions of the image
    width_new = int((height * sin) + (width * cos))
    height_new = int((height * cos) + (width * sin))

    # adjust the rotation matrix to take into account translation
    M[0, 2] += (width_new / 2) - centerX
    M[1, 2] += (height_new / 2) - centerY

    # perform the actual rotation and return the image
    return cv2.warpAffine(image, M, (width_new, height_new))


def sort_contours(contours, method="left-to-right"):
    # initialize the reverse flag and sort index
```

```python
    reverse = False
    i = 0

    # handle if we need to sort in reverse
    if method == "right-to-left" or method == "bottom-to-top":
        reverse = True

    # handle if we are sorting against the y-coordinate rather than
    # the x-coordinate of the bounding box
    if method == "top-to-bottom" or method == "bottom-to-top":
        i = 1

    # construct the list of bounding boxes and sort them from top to
    # bottom
    boundingBoxes = [cv2.boundingRect(contour) for contour in contours]
    (contours, boundingBoxes) = zip(*sorted(zip(contours, boundingBoxes),
                                        key=lambda b: b[1][i], reverse=reverse))

    # return the list of sorted contours and bounding boxes
    return (contours, boundingBoxes)
```