```
import cv2
import numpy as np
from PIL import Image
from numba import jit
import cv2

"""
This script takes the local standard deviation of an image and thresholds it,
removing spurious emission from the edge of sample holder and small flecks of
dirt on the sample chip
"""

def main(filename, box_size):
        img = cv2.imread(filename, 0)

        shape = np.shape(img)

        x_pix_max = shape[0]
        y_pix_max = shape[1]

        #box_size = 3 # default value of 3
        box_length = (2 * box_size) + 1

        std_dev_map = process(img, box_length, box_size, x_pix_max, y_pix_max)
        std_dev_map = np.asarray(std_dev_map)
        std_dev_map = np.uint8(std_dev_map)

        thresh = np.zeros((x_pix_max, y_pix_max))
        thresh_value = np.percentile(std_dev_map, 78)

        std_dev_map[std_dev_map >= thresh_value] = 255
        std_dev_map[std_dev_map < thresh_value] = 0

        std_dev_map = np.uint8(std_dev_map / 255)
        std_dev_img = Image.fromarray(np.uint8(std_dev_map * 255))

        filename_out = ('./images/std_dev_map_pre.png'     )
        std_dev_img.save(filename_out)

        mask = np.ones(std_dev_map.shape, dtype='uint8'   )

        # saving hard copy of thresh to avoid damage from findcontours
        thresh_perm = np.copy(std_dev_map)

        # find array of contours
        (cnts, hierarchy) = cv2.findContours(std_dev_map, cv2.RETR_CCOMP,\
                cv2.CHAIN_APPROX_SIMPLE)

        shape = np.shape(cnts)
        cnts_number = shape[0]

        for i in xrange(0, cnts_number):
                # if contour is bad draw to mask
                if (is_contour_bad(cnts, i, hierarchy) != 0):
                        cv2.drawContours(mask, [cnts[i]], -1, (0, 0, 0), -1)

        for i in xrange(0, cnts_number):
                # if contour is good revert mask back to original state over shape area
                if (is_contour_bad(cnts, i, hierarchy) == 0):
                        cv2.drawContours(mask, [cnts[i]], -1, (1, 1, 1), -1)

        std_dev_map = thresh_perm * mask

        std_dev_img = Image.fromarray(np.uint8(std_dev_map * 255))
        filename_out = ('./images/std_dev_map_post.png'     )
        std_dev_img.save(filename_out)
        print ('standard deviation map saved'      )

        return std_dev_map


@jit(nopython=True)
def process(img, box_length, box_size, x_pix_max, y_pix_max):

        std_dev_map = np.zeros((x_pix_max, y_pix_max))

        for x in xrange(0, x_pix_max):
```

```
                for y in xrange(0, y_pix_max):

                        box = np.zeros((box_length, box_length))

                        if ((x >= box_size) and (x < x_pix_max - box_size) and (y >= box_size) \
                                and (y < y_pix_max - box_size)):

                                for x_box in xrange(-box_size, box_size):
                                        for y_box in xrange(-box_size, box_size):
                                                box[x_box + box_size][y_box + box_size] = img[x + x_box][y +
y_box]

                                if (np.mean(box) != 0):
                                        std_dev_map[x][y] = (150 * np.std(box)) / np.mean(box)
                                else :
                                        std_dev_map[x][y] = 0

        return  std_dev_map


# returns 1 for bad contour and 0 for good contour
def is_contour_bad (cnts, i, hierarchy):
        area = cv2.contourArea(cnts[i])
        length = cv2.arcLength(cnts[i], True)
        extension = 0

        shape = np.shape(cnts)
        cnts_number = shape[0]

        area = cv2.contourArea(cnts[i])

#     if(area >= 10000):
#         print('inloop')
#         for i_h in xrange(0, cnts_number):
#             if((i == hierarchy[0][i_h][3]) & (i != i_h)):
#                 area = area - cv2.contourArea(cnts[i_h])

        if (area != 0):
                extension = ((length / 4) ** 2) / area

        if ((length >= 700) & (area >= 8000)):
                return  1
        elif (extension >= 10):
                return  1
        elif (area <= 50):
                return  1
        else :
                return  0
```