description code, step by step

Hi friends, thank you for your attention, hope this description helps you to work with this code easier.

1- ok, at first lest see the main methods.

(if __name__ ==‘__main__’: , 1 , the last part of code)

```
if __name__ =='__main__':
    time = []
    max_word_height = 140
    borders = [489, 560, 477, 630] # borders = [top, right, down, left]
```

max_word_height : this is height of thickest line in documents.
it will use in aaa for aaa
borders : as it mentioned this is border of image.
it will use in aaa for aaa

(if __name__ ==‘__main__’: , 2 , the last part of code)

```
    for i in range(1,2):
        e1 = cv2.getTickCount()
        file_name = 'image'+str(i)+'.bmp'
        img_source = cv2.imread(file_name)
```

In this part make image source format, in my personal work space, I have images behind my code, if your images are in different directory you would need to change it during code.

e1 : is just for time counting

2 - now I want you to look again to how this code is going to work:

We have two sort of noises:
a) near line noises
for these noises we should be very careful in order not to damage words.

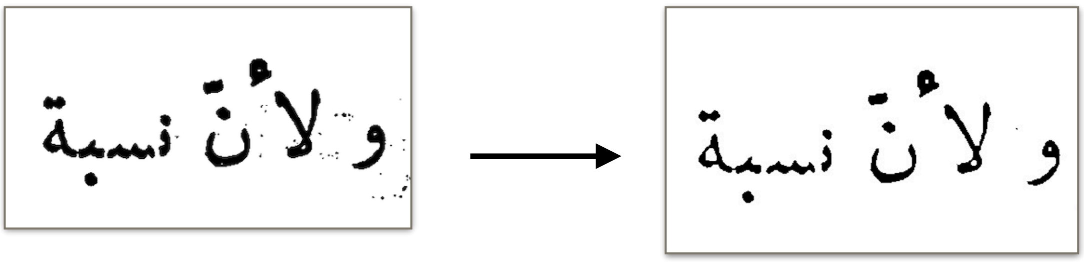
Attention: in this function, we make a mask image called pattern and then, 'and' it (bitwise_and or bitwise_or if image is in-versed) with source image, in order to save the originality of image

(if __name__ ==‘__main__’: , 3 , the last part of code)

```
blur_size = (3, 3)
denoise_erode_kernel = (3, 3)
contour_min_area = 100
min_noise_h = 150
max_noise_w = 5

denoised_by_contours = denoise_by_contours(img_source,img_source,file_name,
                                           blur_size=blur_size,

denoise_erode_kernel=denoise_erode_kernel,
                                           borders=borders,
                                           contour_min_area=contour_min_area,
                                           min_noise_h=min_noise_h,
                                           max_noise_w=max_noise_w)

cv2.imwrite(file_name + '-1-denoise_by_contours.jpg', denoised_by_contours)
```

I partitioned noises to 3 main groupes:

1- some very small noises:
for example in 1080x720 image of document, 3x3 dots. Removing them with blur and then a light closing at first, helps us calculate faster then remove them all with contours.(the blue size at last of function will be added as erosion(dilation in code, duo to image is not in-versed))



codes:

(denoise_by_contours: ,1, definition seegment)

```
blur = cv2.blur(gray_img, blur_size)
_, thresh = cv2.threshold(blur, 127, 255, cv2.THRESH_BINARY)
gray_img = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE,
kernel=np.ones(denoise_erode_kernel, np.uint8))
cv2.imwrite( file_name +'-1-1-first_sort_of_noises.jpg', gray_img)
```

2- some bigger noises that they are smaller than dots:
These noises will be removed with contour area, you should add your small dots area with negative bias, for example if area is 120, use 90.
Attention: in some mathematics documents this bias should be more(attention to = or - and etc.)

codes:

(denoise_by_contours: ,2, definition seegment)

Mask image {

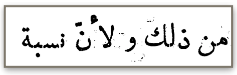
```
contour, _ = cv2.findContours(gray_img, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
new_img = np.zeros((img.shape[0], img.shape[1]), np.uint8)
new_img.fill(255)
for cnt in contour:
    x, y, w, h = cv2.boundingRect(cnt)
    epsilon = 1.23456789e-14
    approx = cv2.approxPolyDP(cnt, epsilon, True)

    if w > img.shape[1]-borders[1]-borders[3] or h > img.shape[0]-borders[1] -
borders[3]:
        continue

    if cv2.contourArea(cnt) < contour_min_area:
        continue
```

For third part, below {

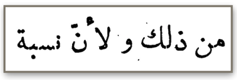
```
    if h > w and h > min_noise_h and w < max_noise_w:
        continue
```

3- and some long noises that they are not small, constant long and thing noises, we give then small width and long height and remove them with contour's boundingRect.
Attention: in some long parentheses.
codes:

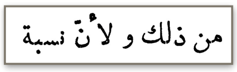
Above


من ذلك و لأنّ نسبة

SOURCE IMAGE


من ذلك و لأنّ نسبة

STEP 1(BLUR AND CLOSING)

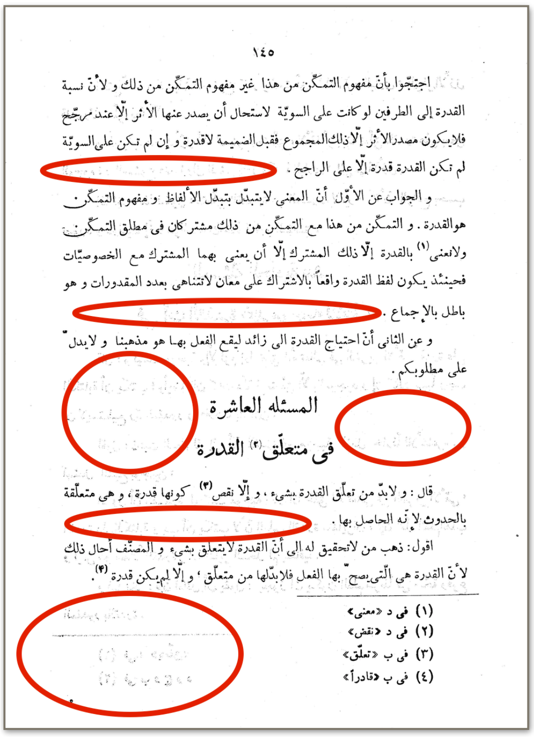

من ذلك و لأنّ نسبة

STEP 2 AND 3

b) not near noises:

These noises are not close to texts like theses:
And some other areas like these in image,
These areas are some areas that they are
Non-text areas.

Attention: in this function, we make a mask image called pattern and then, 'and' it (bitwise_and or bitwise_or if image is in-versed) with source image, in order to save the originality of image


١٤٥

اجتجّوا بأنّ مفهوم التمكّن من هذا غير مفهوم التمكّن من ذلك و لأنّ نسبة القدرة إلى الطرفين لو كانت على السويّة لاستحال أن يصدر عنها الأثر إلّا عند مرجّح فلايكون مصدرالأثر إلّا ذلك المجموع فقبل الضميمة لاقدرة و إن لم تكن على السويّة لم تكن القدرة قدرة إلّا على الراجح .

و الجواب عن الأوّل أنّ المعنى لايتبدّل بتبدّل الألفاظ و مفهوم التمكّن هوالقدرة . و التمكّن من هذا مع التمكّن من ذلك مشتركان فى مطلق التمكّن ولانعنى(١) بالقدرة إلّا ذلك المشترك إلّا أن يعنى بهما المشترك مع الخصوصيّات فحينئذ يكون لفظ القدرة واقعاً بالاشتراك على معان لاتتناهى بعدد المقدورات و هو باطل بالإجماع .

و عن الثانى أنّ احتياج القدرة الى زائد ليقع الفعل بها هو مذهبنا و لايدلّ على مطلوبكم .

المسئله العاشرة

فى متعلّق(٢) القدرة

قال : و لابدّ من تعلّق القدرة بشىء ، و إلّا نقص(٣) كونها قدرة ، و هى متعلّقة بالحدوث لإنّه الحاصل بها .

اقول: ذهب من لاتحقيق له الى أنّ القدرة لايتعلق بشىء و المصنّف أحال ذلك لأنّ القدرة هى الّتى يصحّ بها الفعل فلابدّلها من متعلّق ، و إلّا لم يكن قدرة(٤).

(١) فى د «معنى»
(٢) فى د «نقض»
(٣) فى ب «تعلّق»
(٤) فى ب «قادراً»

codes:

(if __name__ ==‘__main__’: , 4 , the last part of code)

```
first_kernel_erod = (5, 5)
first_kernel_dilate = (15, 15)
pixels_per_slice = 15
block_thresh = 20
border_pixel = 5
min_contour_area_minimizes_img = 3
last_kernel_dilate = (10, 10)

removed_wasted_round_area , pattern =
remove_wasted_round_area(denoised_by_contours,file_name,borders,

first_kernel_erod=first_kernel_erod,

first_kernel_dilate=first_kernel_dilate,

pixels_per_slice=pixels_per_slice,

block_thresh=pixels_per_slice,

border_pixel=border_pixel,

min_contour_area_minimizes_img=min_contour_area_minimizes_img,

last_kernel_dilate=last_kernel_dilate
```

```
                                                               )

cv2.imwrite(file_name + '-2-removed_wasted_round_area_pattern.jpg', pattern)
cv2.imwrite(file_name + '-2-removed_wasted_round_area_img.jpg',
removed_wasted_round_area)
```

For this problem (detecting Non-text area) we do this solution (after making this solution, I searched and found it's name, as I found it is 'pixel density' methodology)

Preprocessing :
For making image read for this methodology, do these:

1- make borders write
2- bitwise_not to be faster
3- erode it in order to connected noises be disconnected
4- dilate it more than erosion so that texts will be repaired and more strong

codes:

(denoise_by_contours: ,1, definition seegment)

```
img[img.shape[0] - borders[2]:,:]=255
img[:,img.shape[1] - borders[1]:]=255
img[:borders[0],:]=255
img[:,0:borders[3]] = 255

gray_env = cv2.bitwise_not(img)
gray_env_erod = cv2.erode(gray_env, kernel=np.ones(first_kernel_erod, np.uint8),
iterations=1)
# cv2.imwrite(file_name + 'area_erode.jpg', gray_env_erod)
gray_env_dilate = cv2.dilate(gray_env_erod, kernel=np.ones(first_kernel_dilate,
                  np.uint8), iterations=1)
# cv2.imwrite(file_name + 'area_dilate.jpg', gray_env_dilate)
cv2.imwrite(file_name +'-2-0-find_wasted_round_area_in_documents-
gray_env_dilate_.jpg', img)
```

First: slice image to pixels, for example in 1080x720 image, in code pixels_per_slice=30 (in each axis), so that we will have 36*24=60 slices
pixels_per_slice: if thicker, higher and for thinner, less.

codes:

(denoise_by_contours: ,2, definition seegment)

```
poly = np.zeros((int(gray_env_dilate.shape[0] / pixels_per_slice),
int(gray_env_dilate.shape[1] / pixels_per_slice), 1), np.uint8)
poly.fill(0)
pices = (int(gray_env_dilate.shape[0] / pixels_per_slice),
int(gray_env_dilate.shape[1] / pixels_per_slice))
for y in range(pices[0]):
    for x in range(pices[1]):
        poly[y, x] = np.mean(gray_env_dilate[(y * pixels_per_slice):((y + 1) *
pixels_per_slice), (x * pixels_per_slice):((x + 1) * pixels_per_slice)])
cv2.imwrite('poly_temp_not_threshed.jpg',poly)
```

Second: in-verse image (not image) and then, make a now image with 36x24 pixel and each pixel color is mean of 30x30 pixels (each pixel is 0 if not text and 255 if is text or black noise in image. Attention image is in-verse)

Our image has higher resolution so, this image have more slices.

image in last step:

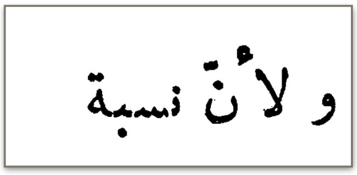
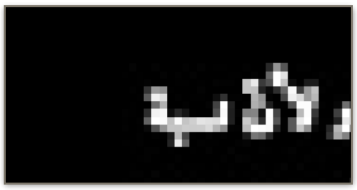


image in this step:



Third: with thresh will detect main part of image(thresh is declared in block_thresh=20)

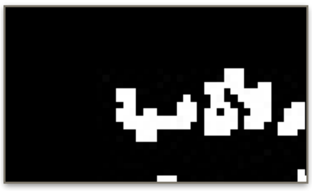
codes:

(denoise_by_contours: ,3, definition seegment)

```
_, poly = cv2.threshold(poly, block_thresh, 255, cv2.THRESH_BINARY)
cv2.imwrite('poly_temp_after_threshed.jpg', poly)
```

image in last step:

image in this step:

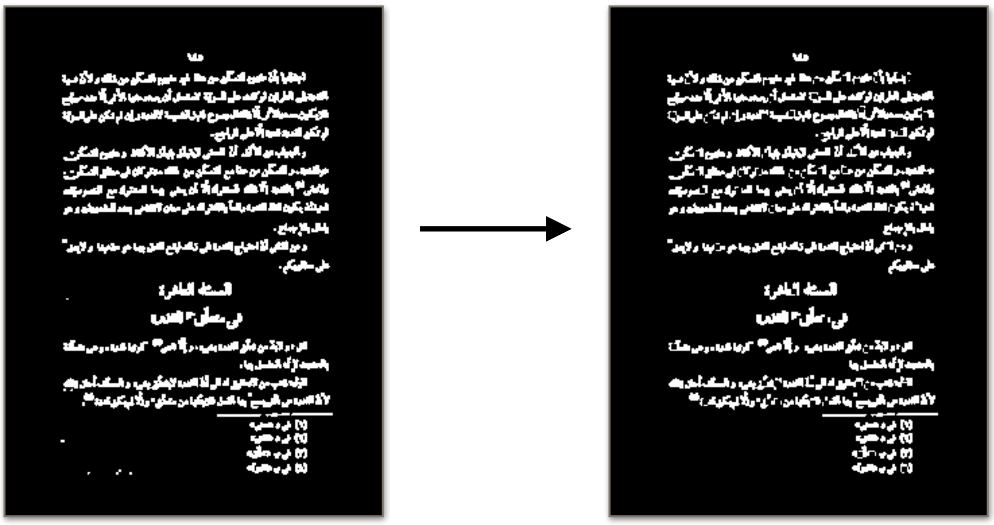


Fourth: now we should remove some still-stayed noises, with an other contour we can do it:
min_contour_area_minimizes_img= 0 if this document is not very non-text noisy
1 if this document is a bit non-text noisy
2 or 3 or …. if this document is very non-text noisy

codes:

(denoise_by_contours: ,4, definition seegment)

```
contours, _ = cv2.findContours(poly, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

for cnt in contours:
    if cv2.contourArea(cnt) < min_contour_area_minimizes_img:
        x, y, w, h = cv2.boundingRect(cnt)
        cv2.rectangle(poly, (x, y), (x + w, y + h), 0, -1)

cv2.imwrite(file_name +'-2-3-find_wasted_round_area_in_documents-poly-
contoured_.jpg', poly)
```

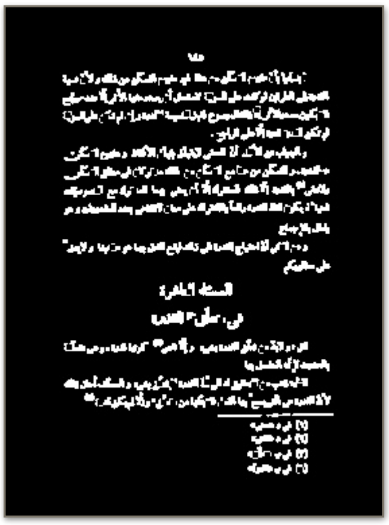


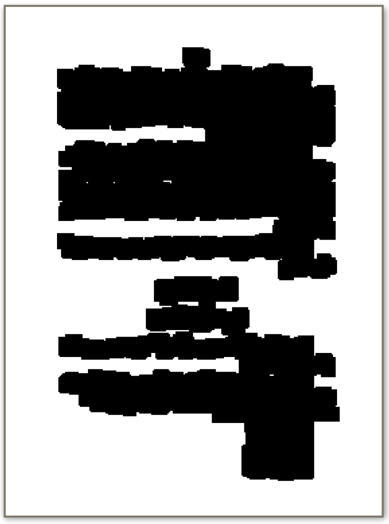
Fifth: if we just convert this image to original image, we may have some losses also duo to we have just denoised near lines with more care than Non-text, and moreover with this way that we have passes we can have text area in connected components, with dilation(also dilation in code) will find its connected component:

codes:

(denoise_by_contours: ,5, definition seegment)

```
poly = cv2.dilate(poly, kernel=np.ones(last_kernel_dilate, np.uint8), iterations=1)
poly3 = np.zeros((int(gray_env_dilate.shape[0]), int(gray_env_dilate.shape[1]), 1),
np.uint8)
poly3.fill(0)
for y in range(0, pices[0]):
    for x in range(0, pices[1]):
        poly3[(y * pixels_per_slice):((y + 1) * pixels_per_slice), (x *
pixels_per_slice):((x + 1) * pixels_per_slice)] = poly[y, x]
```



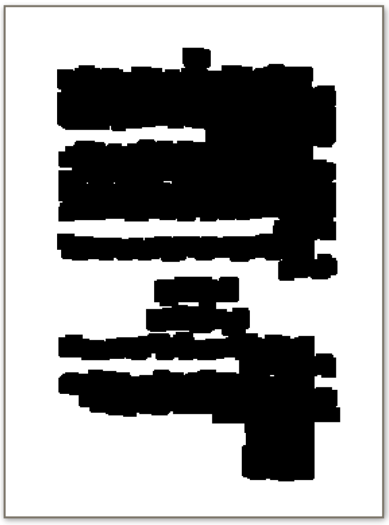


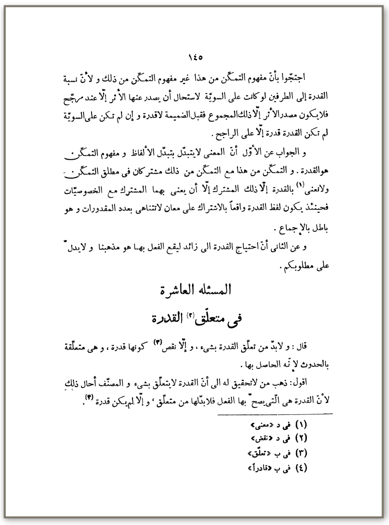
Sixth: then with ‘and’ing (bitwise_or in code duo to image is in-versed) we finish our denoising in both two steps.

codes:

(denoise_by_contours: ,6, definition seegment)

```
no_waisted_area = cv2.bitwise_not(poly3)
no_waisted_area_on_source = cv2.bitwise_or(img, no_waisted_area)
```





١٤٥

اجتجّوا بأنّ مفهوم التمكّن من هذا غير مفهوم التمكّن من ذلك و لأنّ نسبة القدرة إلى الطرفين لو كانت على السويّة لاستحال أن يصدر عنها الأثر إلّا عند مرجّح فلايكون مصدرالأثر إلّا ذلك المجموع فقبل الضميمة لاقدرة و إن لم تكن على السويّة لم تكن القدرة قدرة إلّا على الراجح .

و الجواب عن الأوّل أنّ المعنى لايتبدّل بتبدّل الألفاظ و مفهوم التمكّن هوالقدرة . و التمكّن من هذا مع التمكّن من ذلك مشتركان فى مطلق التمكّن ولانعنى(١) بالقدرة إلّا ذلك المشترك إلّا أن يعنى بهما المشترك مع الخصوصيّات فحينئذ يكون لفظ القدرة واقعاً بالاشتراك على معان لاتتناهى بعدد المقدورات و هو باطل بالإجماع .

و عن الثانى أنّ احتياج القدرة الى زائد ليقع الفعل بها هو مذهبنا و لايدلّ على مطلوبكم .

المسئله العاشرة

فى متعلّق(٢) القدرة

قال : و لابدّ من تعلّق القدرة بشىء ، و إلّا نقص(٣) كونها قدرة ، و هى متعلّقة بالحدوث لأنّه الحاصل بها .

اقول: ذهب من لاتحقيق له الى أنّ القدرة لايتعلّق بشىء و المصنّف أحال ذلك لأنّ القدرة هى الّتى يصحّ بها الفعل فلابدّلها من متعلّق ، و إلّا لم يكن قدرة(٤).

(١) فى د «معنى»
(٢) فى د «نقض»
(٣) فى ب «تعلّق»
(٤) فى ب «قادراً»

3- after denoising we need to do some other post-processing:

Attention: in spite of step 2, in this step we do not try to use make a mask, because there is no need to bitwise_and with main image, because

a) repairing images:
images in documents are damaging in denoising, we need to re-put them back.

codes:

(if __name__ ==‘__main__’: , 5 , the last part of code)

```
image_shape = [500,500]
        returned_images_img = move_images(removed_wasted_round_area , img_source, 
borders, image_shape)
        cv2.imwrite(file_name + '-5-returned_images.jpg', returned_images_img)
```

in the code, first we thresh contours by being inside the borders and images are bigger then image_shape and then replace them with source images.

codes:

(move_images: , 1, definition seegment)

```
def move_images(img, img_source , borders , image_shape): # borders = [top, right, 
down, left] image_shape = [min_w, min_h]

    img_source_gray = cv2.cvtColor(img_source, cv2.COLOR_BGR2GRAY)
    output_base_image = cv2.bitwise_not(img_source_gray)
    contours, _ = cv2.findContours(output_base_image, cv2.RETR_TREE, 
cv2.CHAIN_APPROX_SIMPLE)

    # the image or words must be between 10% and 90% of main image
    max_y = img.shape[0] - borders[2]
    max_x = img.shape[1] - borders[1]
    min_y = borders[0]
    min_x = borders[3]
    min_w = image_shape[0]
    min_h = image_shape[1]


    # for any contour in main image , check if it is big enough to ba an image or 
word
    for contour in contours:
        x, y, w, h = cv2.boundingRect(contour)  # position of contour
        if y < max_y and y > min_y and x < max_x and x > min_x and w > min_w and h 
> min_h:
            img[y:y + h, x:x + w] = img_source_gray[y:y + h, x:x + w]

    cv2.imwrite(file_name +'-5-move_images.jpg', img)
    return img
```

b) correcting rotation:
For correcting rotation we first need to find main line and then by using average of main lines degree find the rotation degree. Images are decreasing accuracy, so at first we need to remove images and then find main lines.

codes:

(if __name__ ==‘__main__’: , 6 , the last part of code)

```
corrected_rotation = correct_rotation(returned_images_img, max_word_height)
cv2.imwrite(file_name + '-6-corrected_rotation.jpg', corrected_rotation)
```

max_word_height: is our thresh for removing contours

codes:

(correct_rotation: , 1, definition seegment)

```
def correct_rotation(img,max_word_height):# max_word_height = 140

    env_img = cv2.bitwise_not(img)
    contours, _ = cv2.findContours(env_img, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    for contour in contours:
        x, y, w, h = cv2.boundingRect(contour)  # position of contour
        if h < max_word_height* 1.5:
            continue
        env_img[y:y + h, x:x + w] = 0

    img = cv2.bitwise_not(env_img)
    edges = cv2.Canny(img, 50, 150, apertureSize=3)
    lines = cv2.HoughLines(edges, 1, np.pi / 180, 200)
    num = 0
    sum = 0

    # if there is no line
    try:
        for i in lines:
            for rho, theta in i:
                if np.degrees(theta) > 45 and np.degrees(theta) < 135:
                    sum += np.degrees(theta)
                    num += 1

        rows, cols = img.shape[0], img.shape[1]
        if num != 0:
            M = cv2.getRotationMatrix2D((cols / 2, rows / 2), (sum / num) - 90, 1)
            img = cv2.warpAffine(img, M, (cols, rows))
            theta_radian = np.radians((sum / num) - 90)
            y = int(np.sin(theta_radian) / np.cos(theta_radian) * img.shape[0])
            img[0:abs(y), :] = 255
            img[img.shape[0] - abs(y):, :] = 255
            x = int(np.sin(theta_radian) / np.cos(theta_radian) * img.shape[1])
            img[:, 0:abs(y)] = 255
            img[:, img.shape[1] - abs(y):] = 255

    except:
        pass

    return img
```

End code description.

Some other questions:

- why we do that? Why we don't use neural networks?
for using neural networks we needed more datasets then English duo to curves and distributed dots in Persian and Arabic alphabet

- what is next plane?
as my next step, by help of one my friends and colleague, Mss. Nafarieh, we are trying to collect datasets for a neural network that do the same thing with images.