به نام خدا

گزارش پروژه بینایی کامپیوتر – 99521199 و 99521415

توضیحات به صورت کامل در کد آمده اما برای شرح کوتاه میتوانیم به این اشاره کنیم که ابتدا یک عکس را به عنوان ورودی به تابع main میدهیم و آنجا پس از اینکه با روش گفته شده اسکن و کراب شد چک میکنیم که آیا این کارت از جنس کارت ملیست و یا کارت بانکی است.

برای این چک کردن بررسی میکنیم که آیا کارت تصویر خورشید پایین سمت چپ را دارد یا نه.

def is\_national\_card(image):

    sun = image[700:1000 , 0:300]

    sun\_template = cv2.imread('sun.png', 0)

    # template matching

    res = cv2.matchTemplate(sun, sun\_template, cv2.TM\_CCOEFF\_NORMED)

    threshold = 0.5

    # show hit map

    loc = np.where(res >= threshold)

    if len(loc[0]) > 0:

        return True

    else:

        return False

وقتی فهمیدیم که کارت از چه جنسی است باید برای هرکدام به طور خاص اطلاعات خواسته شده را استخراج کنیم.

اولین حالت بررسی کارت ملیست.

def detect\_national\_card(image):

    templates = generate\_templates('Fa\_dataset')

    image = image[200:650, 800:1250]

    clustered\_rectangles, matched\_image = fa\_extract\_segments(image)

    for cluster in clustered\_rectangles:

        phrase = template\_matching(image, matched\_image, cluster, templates)

        if "/" in phrase:

            print("expir\_date: ", phrase)

            continue

        elif len(phrase) == 10:

            national\_id = phrase

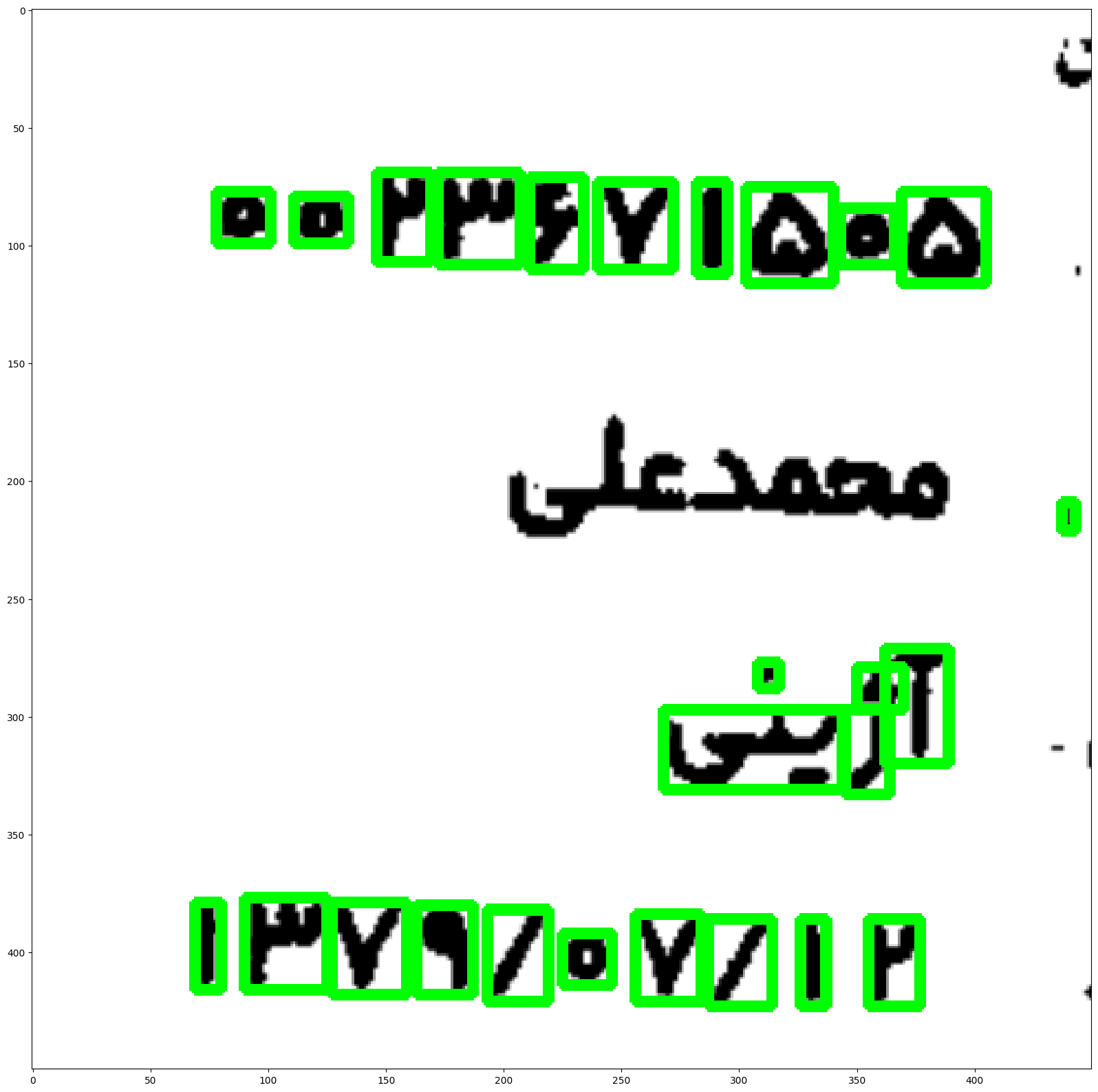
            print("national\_id: ", national\_id)

دیتاست 'Fa\_dataset' دیتاست ما برای اعداد فارسی است که درون آن اعداد 0 تا 9 به علاوه / برای استفاده در تمپلیت مچینگ قرار گرفته شده. ابتدا دیتاست خوانده شده و در templates قرار میگیرد و سپس عکسی که اسکن و کات کردیم را طوری برش میدهیم که فقط قسمت های شماره ملی و تاریخ انقضا مشخص باشد و بعد آن را به تابع fa\_extract\_segment میهیم تا نواحی مهم اسخراج بشع و بعد از اون اون نواحی رو بررسی میکنیم که آیا با تمپلیت هایی که داریم تشابهی کشف میه و عددی پیدا میشه یا نه.اگر / داشت یعنی تاریخ انقضاست و اگر نداشت یعتی شماره ملیه.

ورودی نمونه:



خروجی:





حال به بررسی کد برای حالت کارت بانکی میرسیم:

def detect\_credit\_card(image):

    templates = generate\_templates('dataset')

    clustered\_rectangles, matched\_image = extract\_segments(image)

    for cluster in clustered\_rectangles:

        phrase = template\_matching(image, matched\_image, cluster, templates)

        if "/" in phrase:

            slash\_index = phrase.index("/")

            expir\_date = phrase[slash\_index - 2: slash\_index + 3]

            if len(phrase)>=7 and slash\_index >= 4:

                print("expir\_date: ", expir\_date)

        elif len(phrase) == 16:

            card\_number = phrase

            print("card\_number: ", card\_number)

فولدر dataset شامل پند نمونه عدد با فونت های مختلف انگلیسی است که برای تمپلیت مچینگ استفاده میکنیم.

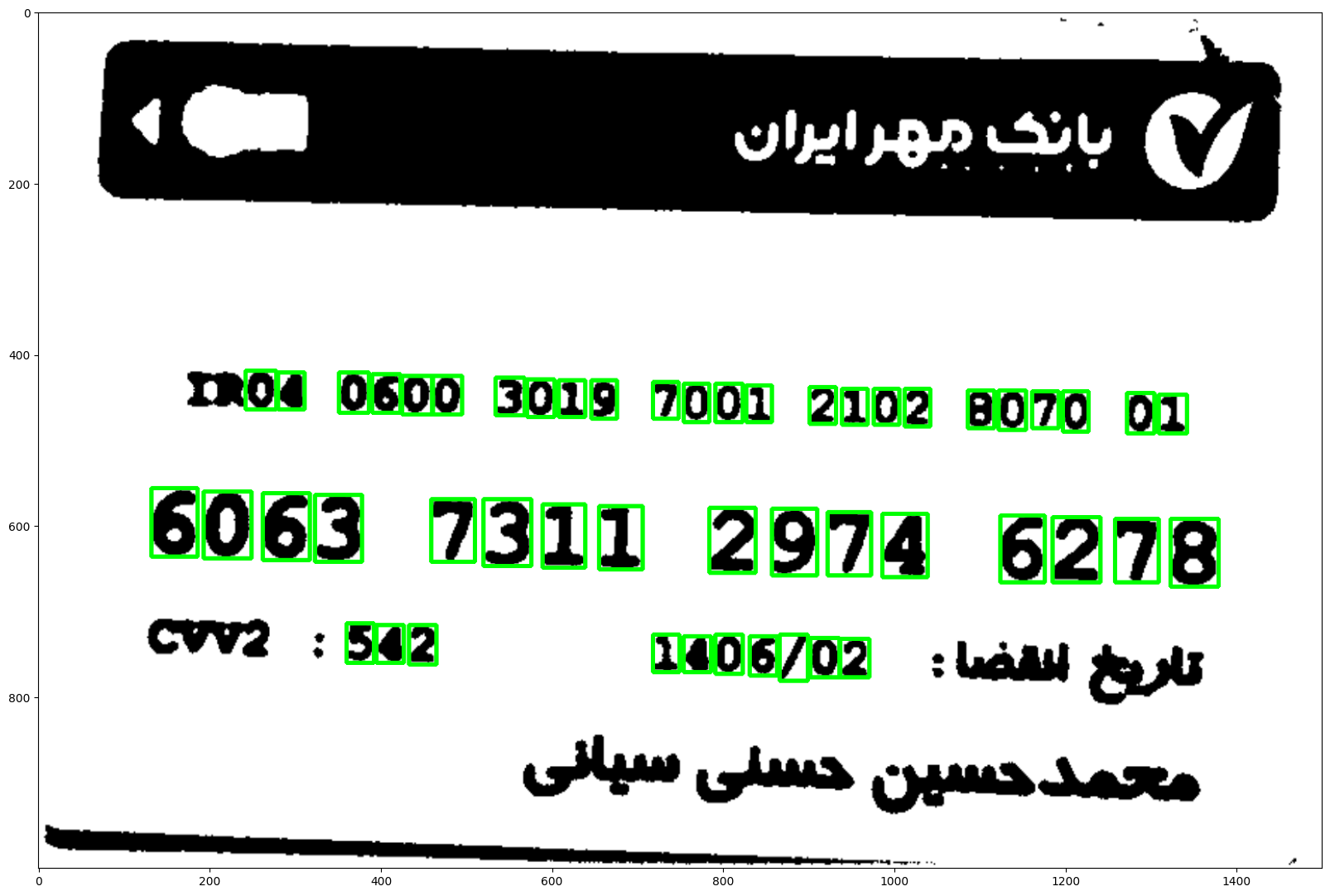
سپس مانند قبل نواحی مهم را استخراج میکنیم و برای هر کدام چک میکنیم که آیا مچ پیدا میشه یا نه

در صورتی که پیدا کرد و 16 رقم بود شماره کارت هست و اگر / داشت تاریخ انقضا.

نمونه ورودی:



خروجی:

این نمونه خروجی ما برای دو حالت خواسته شده است.

حال به توضیح توابع استفاده شده در هر بخش می پردازیم:

1. Reorder :
2. def reorder(poly):
3. # Find the index of the point with the lowest y-coordinate
4. p\_lowest = np.argmax(poly[:, 1])
5. # Check if there are two points with the same lowest y-coordinate
6. if np.count\_nonzero(poly[:, 1] == poly[p\_lowest, 1]) == 2:
7. # Find the index of the point with the lowest sum of x and y coordinates
8. p0\_index = np.argmin(np.sum(poly, axis=1))
9. # Determine the indices of the other points in clockwise order
10. p1\_index = (p0\_index + 1) % 4
11. p2\_index = (p0\_index + 2) % 4
12. p3\_index = (p0\_index + 3) % 4
13. # Reorder the points and return the new polygon
14. return poly[[p1\_index, p0\_index, p3\_index, p2\_index]]
15. else:
16. # Find the indices of the points to the right and left of the lowest point
17. p\_lowest\_right = (p\_lowest - 1) % 4
18. p\_lowest\_left = (p\_lowest + 1) % 4
19. # Calculate the angle between the lowest point and the point to the right
20. angle = np.arctan2(poly[p\_lowest\_right][1] - poly[p\_lowest][1], poly[p\_lowest\_right][0] - poly[p\_lowest][0])
21. # Check if the angle is negative
22. if angle < 0:
23. print(angle, poly[p\_lowest], poly[p\_lowest\_right])
24. # Check if the angle is greater than 45 degrees (pi/4 radians)
25. if np.degrees(angle) > 45:
26. # Determine the indices of the points in counterclockwise order
27. p2\_index = p\_lowest
28. p1\_index = (p2\_index - 1) % 4
29. p0\_index = (p2\_index - 2) % 4
30. p3\_index = (p2\_index + 1) % 4
31. return poly[[p1\_index, p0\_index, p3\_index, p2\_index]]
32. else:
33. # Determine the indices of the points in counterclockwise order
34. p3\_index = p\_lowest
35. p0\_index = (p3\_index + 1) % 4
36. p1\_index = (p3\_index + 2) % 4
37. p2\_index = (p3\_index + 3) % 4
38. return poly[[p1\_index, p0\_index, p3\_index, p2\_index]]

تابع `reorder` به عنوان ورودی یک چندضلعی محدب (polygon) به شکل یک آرایه دو بعدی دریافت می‌کند و آن را مرتب می‌کند. فرض می‌شود که چندضلعی ورودی دارای چهار راس است.

ابتدا، این تابع نقطه‌ای را پیدا می‌کند که کمترین مختصات `y` را دارد. برای این کار، اندیس نقطه با بیشترین مقدار `y` را محاسبه می‌کند و آن را در متغیر `p\_lowest` ذخیره می‌کند.

سپس، بررسی می‌شود که آیا دو نقطه با کمترین مختصات `y` دارای مقدار یکسان هستند یا خیر. اگر دو نقطه دارای مختصات یکسان باشند، این به معنی وجود دو نقطه با کمترین مختصات `y` است. در این صورت، تابع محاسباتی را انجام می‌دهد تا نقطه‌ای که مجموع مختصات `x` و `y` کمتری داشته باشد را پیدا کند. سپس، نقاط را به ترتیب مخالف عقربه‌های ساعت مرتب می‌کند و چندضلعی جدید را برمی‌گرداند.

در غیر این صورت )هنگامی که دو نقطه با کمترین مختصات `y` مختصات یکسان ندارند(، ابتدا اندیس نقطه‌ای که به سمت راست نقطه کمترین مختصات `y` قرار دارد را محاسبه می‌کند و به ترتیب مقدار عقربه‌های ساعت از نقطه کمترین مختصات `y` به نقطه‌ای که به سمت راست آن قرار دارد را محاسبه می‌کند. سپس، تابع زاویه محاسبه می‌کند که آن را به عنوان زاویه بین خطی که از نقطه کمترین مختصات `y` به سمت راست آن کشیده می‌شود و خط افقی در نقطه کمترین مختصات `y` در نظر می‌گیرد.

سپس، بررسی می‌شود که آیا زاویه به دست آمده منفی است یا خیر. اگر زاویه منفی باشد، این به معنی وجود یک زاویه بیشتر از ۱۸۰ درجه است. در این صورت، تابع نقاط را به ترتیب مقدار عقربه‌های مخالف عقربه‌های ساعت مرتب می‌کند و چندضلعی جدید را برمی‌گرداند.

در غیر این صورت (زاویه کمتر یا مساوی ۱۸۰ درجه)، تابع نقاط را به ترتیب مقدار عقربه‌های عقربه‌های ساعت مرتب می‌کند و چندضلعی جدید را برمی‌گرداند.

در هر دو حالت، چندضلعی جدید که نقاط را به ترتیب صحیح مرتب کرده است را برمی‌گرداند.

2. reflect 101

def Reflect101(img,filter\_size):

    image = np.pad(img, pad\_width=((filter\_size//2, filter\_size//2), (filter\_size//2, filter\_size//2)), mode='reflect', reflect\_type='odd')

    return image

کد مانند یکی از تمرین ها

2. Averaging\_Blurring

def Averaging\_Blurring(img, filter\_size):

    image = Reflect101(img, filter\_size)

    result = np.zeros((img.shape))

    avg\_filter = np.ones((filter\_size , filter\_size))/ (filter\_size \* filter\_size)

    for i in range(img.shape[0]):

      for j in range(img.shape[1]):

        result[i,j] = np.convolve(image[i :i + filter\_size , j : j + filter\_size].flatten(), avg\_filter.flatten() , mode='valid')

    return result

مانند یکی از تمرینها

3. distance

def calculate\_distance(point1, point2):

    x1, y1 = point1

    x2, y2 = point2

    distance = math.sqrt((x2 - x1)\*\*2 + (y2 - y1)\*\*2)

    return distance

مانند یکی از تمرینها

4. scanning:

def scanner(img , kernel\_size ,t\_lower , t\_higher , output\_name):

  preprocessed\_image = preprocess\_image(img, kernel\_size, t\_lower, t\_higher)

  card\_contour = find\_card\_contour(preprocessed\_image)

  perspective\_transformed = perspective\_transform(img, card\_contour, rectangle, flag\_super\_no\_rectangle)

  plt.imshow(perspective\_transformed , cmap = 'gray')

  cv2.imwrite(output\_name, perspective\_transformed)

اول با تابع نوشته شده تصویر را preprocess میکنیم ( gray , blur , canny , dilate) سپس گوشه ها و ضالع هارا در میارییم و بعد presprctive transform را پیاده سازی میکنیم (مانند یکی از تمرین ها)

5.make templates :

def make\_templates(template):

    list\_candid\_temps = []

    # Define the desired number of resize steps

    resize\_steps = 10

    # Calculate the resize ratios for both width and height

    width\_ratios = np.linspace(1, 0.4, resize\_steps)

    height\_ratios = np.linspace(1, 0.4, resize\_steps)

    # Iterate over the resize ratios

    for width\_ratio in width\_ratios:

        for height\_ratio in height\_ratios:

            if abs(width\_ratio - height\_ratio) == 0:

                # Resize the template

                resized\_template = cv2.resize(template, (int(template.shape[1] \* width\_ratio), int(template.shape[0] \* height\_ratio)))

                # erosion

                kernel = np.ones((5,5),np.uint8)

                erosion = cv2.erode(resized\_template,kernel,iterations = 1)

                # Rotate the resized template

                for angle in range(-5, 5):

                    rotated\_template = cv2.warpAffine(resized\_template, cv2.getRotationMatrix2D((resized\_template.shape[1] / 2, resized\_template.shape[0] / 2), angle, 1), (resized\_template.shape[1], resized\_template.shape[0]))

                    # Add the rotated template to the list of candidates

                    list\_candid\_temps.append(rotated\_template)

                # Rotate the resized template

                for angle in range(-5, 5):

                    rotated\_template = cv2.warpAffine(erosion, cv2.getRotationMatrix2D((erosion.shape[1] / 2, erosion.shape[0] / 2), angle, 1), (erosion.shape[1], erosion.shape[0]))

                    # Add the rotated template to the list of candidates

                    list\_candid\_temps.append(rotated\_template)

    return list\_candid\_temps

تابع `make\_templates` یک قالب (template) تصویر را به عنوان ورودی دریافت می‌کند و قالب‌های جدید را ایجاد می‌کند. در این تابع، ابتدا یک لیست خالی به نام `list\_candid\_temps` تعریف می‌شود که قالب‌های جدید را در آن ذخیره می‌کنیم.

سپس، تعداد مراحل تغییر اندازه را تعیین می‌کنیم. در اینجا، تعداد مراحل تغییر اندازه را برابر با ۱۰ قرار داده‌ایم.

سپس، با استفاده از توابع `linspace` در numpy، ضرایب تغییر اندازه را برای عرض و ارتفاع محاسبه می‌کنیم. این ضرایب به صورت یک آرایه توزیع شده از ۱ تا ۰.۴ هستند و نشان دهنده میزان تغییر اندازه قالب نسبت به اندازه اولیه است.

سپس، با استفاده از دو حلقه `for` به ترتیب بر روی ضرایب تغییر اندازه عرض و ارتفاع، عملیات تغییر اندازه و چرخش را انجام می‌دهیم. در هر مرحله، قالب را با استفاده از تابع `cv2.resize` تغییر اندازه می‌دهیم و سپس با استفاده از تابع `cv2.warpAffine` آن را به اندازه تغییر یافته چرخانده و در لیست `list\_candid\_temps` قرار می‌دهیم.

قبل از افزودن قالب چرخانده به لیست `list\_candid\_temps`، از تابع `cv2.erode` برای اعمال عملیات فرسایش (erosion) بر روی قالب تغییر یافته استفاده می‌کنیم. این عملیات می‌تواند به کاهش نویز و تقویت حواشی قالب کمک کند. سپس، قالب چرخانده و فرسایش یافته را به لیست `list\_candid\_temps` اضافه می‌کنیم.

در نهایت، لیست `list\_candid\_temps` که حاوی قالب‌های جدید است را برمی‌گردانیم.