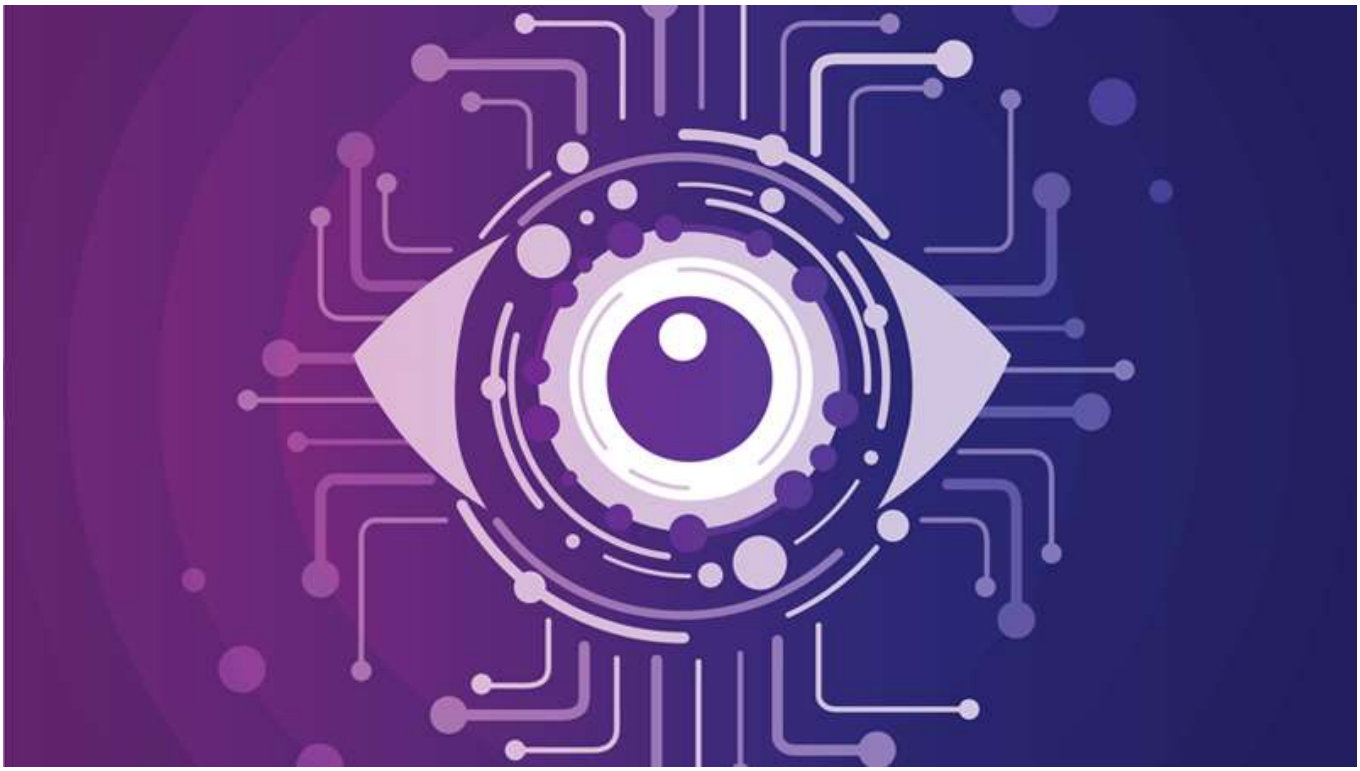
بسم الله الرحمن الرحیم



پارسا نوروزی(۹۸۵۲۲۱۰۳)

محمد عرفان زارع زردینی(۹۸۴۱۱۴۳۲)

پروژه درس بینایی کامپیوتر

استاد راهنما: استاد محمدی

برای پیاده سازی این پروژه از چندین تابع به صورت سری استفاده شد. در برنامه نوشته شده ما با استفاده از روش template matching، به تشخیص کارت های بانکی و ملی ازهم و در نهایت بررسی خواسته های موردنیاز پرداخته ایم. در کد نوشته شده تلاش شده با استفاده از functional کردن برنامه و استفاده بهینه و به جا از توابع مورد نیاز، به کاهش زمان و افزایش سرعت برنامه کمک شود. همچنین در برنماه سعی شده است تا هر بخش و فیچر به قسمت های جزئی تر تقسیم شود تا حجم تابع ها کم تر و بهینه تر شود.

کارکرد و هدف برنامه شامل یکسری توابع مشخص است.

این توابع چند بخش اصلی دارد که عبارتند از:

۱) جداسازی کارت از عکس
۲) تنظیم زایه ی کارت (Rotation) و تشخیص نوع کارت
۳) خواندن اعداد روی کارت و مشخص کردن محدوده ی اعداد روی کارت

کتابخانه های مورد استفاده در طول این پروژه به شرح زیر میباشدند:

```
import cv2

import matplotlib.pyplot as plt

import numpy as np

import os

from skimage import exposure
```

توابع استفاده شده در طول پروژه:

1)Resize_card:

```
def resize_card(image, corners):
    width = int(np.sqrt(np.sum((corners[1] - corners[0]) ** 2)))
    height = int(np.sqrt(np.sum((corners[3] - corners[0]) ** 2)))

    target_corners = np.array([[0, 0], [width, 0], [width, height], [0, height]], dtype=np.float32)

    transformation_matrix = cv2.getPerspectiveTransform(corners, target_corners)

    resized_image = cv2.warpPerspective(image, transformation_matrix, (width, height))
    if height>width:
        return cv2.rotate(resized_image,cv2.ROTATE_90_CLOCKWISE)
    return resized_image
```

این تابع به عنوان ورودی تصویر و آرایه ای از نقاط را میگیرد (چهار نقطه ی گوشه ی کارت) و بعد نقاط را روی گوشه های تصویر map کرده و اگر تصویر عمودی باشد آنرا ۹۰ درجه میچرخاند. در نهایت هم تصویر جدید که فقط متشکل از کارت و به صورت افقی است را برمیگرداند.

2) get4Corners:

این تابع برای مشخص نمودن ۴ نقطه نهایی که شامل مقادیر۴ گوشه عکس(بالا راست،بالاچپ، پایین راست، پایین چپ) است. بدین گونه که ورودی مختصات ۴ نقطه روی تصویر اولیه است که مشخص نشده است که هر نقطه نمایانگر کدام قسمت و نشانی از کارت می باشد. سپس در تابع ابتدا ما مختصات نقطه وسط کارت را پیدا و هرنقطه را با آن مقایسه می کنیم تا در نهایت ۴ مقدار مربوط به گوشه پر شود. اما در این مسیر چالشی که وجود دارد، در تصاویری است که کارت به شکل ناهمگون یا کج قرار گرفته یا دارای زاویه است. برای اینکار می آییم تا زمانی که همه ۴ گوشه مقداردهی نشده اند، نقاط کناری را با هم جابجا و مقدار دهی می نماییم تا در نهایت همه گوشه ها یافته شوند. این بررسی را براساس تعداد عنصر موجود در متغیر ها انجام می دهیم و براساس اینکه کدام متغیر درحال بررسی است ،عناصر داخل آن را با مقایسه و یکی از آن ها را براساس آن مقایسه به متغیر دیگر نسبت می دهیم.

```python
def get4Corners(sorted):
    p1 = sorted[0][0]
    p2 = sorted[1][0]
    p3 = sorted[2][0]
    p4 = sorted[3][0]
    centroid = ((p1[0] + p2[0] + p3[0] + p4[0]) / 4, (p1[1] + p2[1] + p3[1] + p4[1]) / 4)

    top_left = []
    top_right = []
    bottom_left = []
    bottom_right = []

    for point in [p1, p2, p3, p4]:
        if point[0] < centroid[0] and point[1] < centroid[1]:
            top_left.append(point)
        elif point[0] > centroid[0] and point[1] < centroid[1]:
            top_right.append(point)
        elif point[0] < centroid[0] and point[1] > centroid[1]:
            bottom_left.append(point)
        else:
            bottom_right.append(point)

    while len(top_left) < 1 or len(top_right) < 1 or len(bottom_right) < 1 or len(bottom_left) < 1 :
        if len(top_left) > 1:
            topleft0= top_left[0]
            topleft1= top_left[1]
            top_left.clear()

            if topleft0[0] < topleft1[0]:
                top_left.append(topleft0)
                top_right.append(topleft1)
            else:
                top_left.append(topleft1)
                top_right.append(topleft0)

        if len(bottom_right) > 1:
            bottomright0= bottom_right[0]
            bottomright1= bottom_right[1]
            bottom_right.clear()

            if bottomright0[0] < bottomright1[0]:
                bottom_left.append(bottomright0)
                bottom_right.append(bottomright1)
            else:
                bottom_left.append(bottomright1)
                bottom_right.append(bottomright0)

        if len(top_right) > 1:
            topright0= top_right[0]
            topright1= top_right[1]
            top_right.clear()

            if topright0[1] < topright1[1]:
                top_right.append(topright0)
                bottom_right.append(topright1)
            else:
                top_right.append(topright1)
                bottom_right.append(topright0)


        if len(bottom_left) > 1 :
            bottomleft0= bottom_left[0]
            bottomleft1= bottom_left[1]
            botton_left.clear()
            if bottomleft0[1] < bottomleft1[1]:
                top_left.append(bottomleft0)
                bottom_left.append(bottomleft1)
            else:
                top_left.append(bottomleft1)
                botton_left.append(bottomleft0)

    return [top_left,top_right,bottom_left,bottom_right]
```

3) extract_card:

```
def extract_card(image_path):
    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blurred = cv2.medianBlur(gray, 35)
    blurred = cv2.GaussianBlur(blurred, (7, 7), 10)
    average_intensity = np.mean(gray)
    thresh = 0
    if average_intensity >= 150:
        thresh =average_intensity - 80
    elif average_intensity >= 140:
        thresh = average_intensity - 50
    elif average_intensity >= 110:
        thresh = average_intensity - 20
    elif average_intensity >= 80:
        thresh = average_intensity + 11
    elif average_intensity >= 65:
        thresh = average_intensity + 40
    elif average_intensity >=50:
        thresh = average_intensity + 25
    elif average_intensity >=20:
        thresh = average_intensity + 15
    else:
        thresh = average_intensity - 5
    _, threshold = cv2.threshold(blurred, thresh, 255, cv2.THRESH_BINARY_INV)
    edges = cv2.Canny(threshold, 15, 30)
    contours, _ = cv2.findContours(edges.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
    contours = sorted(contours, key=cv2.contourArea, reverse=True)
    largest_contour = contours[0]
    epsilon = 0.05 * cv2.arcLength(largest_contour, True)
    approx = cv2.approxPolyDP(largest_contour, epsilon, True)
    sorted4 = get4Corners(approx)

    top_left = sorted4[0]
    top_right = sorted4[1]
    bottom_left = sorted4[2]
    bottom_right = sorted4[3]
    corners = np.array([top_left ,top_right, bottom_right, bottom_left], dtype=np.float32)
    resized_image = resize_card(image, corners)
    resized_image = cv2.resize(resized_image,(600,350))
    # new_path = 'cards/' + image_path.split('/')[1]
    # cv2.imwrite(new_path,resized_image)
    return resized_image
```

این تابع در ورودی مسیر خواندن یک تصویر را میگیرد.

سپس تصویر را میخواند، آنرا خاکستری میکند، روی آن MedianBlur میزند بعد روی نتیجه GaussianBlur میزند. سپس بر اساس میانگین روشنایی تصویر یک حد تعریف کرده و آن حد را در متغیر thresh نگهداری میکند و از آن برای باینری کردن تصویر با تابع آماده cv.threshold استفاده میکند. نتیجه trheshold را با canny لبه یابی میکند. بعد روی لبه ها findContours میزند سپس کانتور ها را بر اساس محیط آنها مرتب کرده و بزرگ ترین کانتور را که مشخص کننده ی محیط دور کارت است جدا میکند. سپس با approxPolyDP چهار نقطه ی اصلی کانتور را که معین کننده ی شکل کانتور هستند جدا میکند. پس از آن به کمک تابع get4Corners که در بالا توضیح داده شد معلوم میشود که این چهار نقطه هر کدام مربوط به کدام گوشه های مستطیل هستند. بعد با فرستادن گوشه ها و عکس به تابع resize_card که بالا تر توضیح داده شد تصویر کارت استخراج میشود و نهایتا هم ابعاد کارت به ۶۰۰ در ۳۵۰ تبدیل میشود و کارت نهایی برگردانده میشود.

4) adjust_rotation_bank:

```python
def adjust_rotation_bank(src):
    filter = np.array([[-1, -1, -1], [-1, 9, -1], [-1, -1, -1]])
    grayimg = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)
    grayimg = cv2.filter2D(grayimg,-1,filter)
    refimg = cv2.imread('templates/cards/bankTemplate.jpg')
    grayref = cv2.cvtColor(refimg, cv2.COLOR_BGR2GRAY)
    grayref=cv2.filter2D(grayref,-1,filter)
    grayref = cv2.resize(grayref,(59,38))
    grayscale_8bit = cv2.convertScaleAbs(grayref)
    refimg = cv2.cvtColor(grayscale_8bit, cv2.COLOR_GRAY2BGR)
    refimgReverse = cv2.rotate(refimg,cv2.ROTATE_180)
    color_matched = cv2.cvtColor(grayimg[:150,:300], cv2.COLOR_GRAY2BGR)
    color_matched_reverse = cv2.cvtColor(grayimg[150:,300:], cv2.COLOR_GRAY2BGR)
    result = cv2.matchTemplate( color_matched, refimg, cv2.TM_CCOEFF)
    reverseResult = cv2.matchTemplate( color_matched_reverse, refimgReverse, cv2.TM_CCOEFF)
    finalRes = src
    result = np.max(result)
    reverseResult = np.max(reverseResult)
    if result < reverseResult:
        finalRes = cv2.rotate(src, cv2.ROTATE_180)
    return finalRes
```

در این تابع تصویر کارت جدا شده به عنوان ورودی داده میشود.

روی تصویر فیلتر sharp کننده زده میشود، خاکستری میشود و بعد سه کاناله میشود. همچنین لوگوی مشترک روی تمام کارت های بانکی هم به عنوان template خوانده شده، خاکستری شده، sharp شده و بعد سه کاناله میشود. سپس یکبار روی تصویر صاف با template صاف template matching زده میشود و یکبار هم با تصویر ۱۸۰ درجه چرخیده و template ای که ۱۸۰ درجه چرخیده template matching زده میشود.

اگر نمره ی شباهت تصاویر صاف بیشتر بود یعنی تصویر از اول صاف بوده و همان تصویر را برمیگردانیم. اما اگر نمره ی شباهت تصاویر برعکس شده بیشتر باشد یعنی تصویر برعکس بوده و اول ۱۸۰ درجه تصویر را میچرخانیم و سپس برمیگردانیم.

5) adjust_rotation_id:

```
def adjust_rotation_id(src):
    grayimg = cv2.cvtColor(src.copy(), cv2.COLOR_BGR2GRAY)
    templateCardPic = cv2.imread('templates/cards/templateCard.jpg')
    templateCard = cv2.resize(templateCardPic,(600,350))
    refimg = templateCard
    grayref = cv2.cvtColor(refimg, cv2.COLOR_BGR2GRAY)
    refimgReverse = cv2.rotate(refimg,cv2.ROTATE_180)
    matched_image = exposure.match_histograms(grayimg, grayref)
    grayscale_8bit = cv2.convertScaleAbs(matched_image)
    color_matched = cv2.cvtColor(grayscale_8bit, cv2.COLOR_GRAY2BGR)
    result = cv2.matchTemplate( color_matched[:,220:], refimg[:,220:], cv2.TM_CCOEFF_NORMED)
    reverseResult = cv2.matchTemplate( color_matched[:,0:380], refimgReverse[:,0:380], cv2.TM_CCOEFF_NORMED)
    label = "Bank Card"
    finalRes = src
    if result > 0.22 or reverseResult > 0.22:
        label = "ID Card"
        if result < reverseResult:
            finalRes = cv2.rotate(src, cv2.ROTATE_180)
    else:
        finalRes = adjust_rotation_bank(src)

    return [finalRes, label]
```

در تابع نوشته شده، به بررسی این نکته میپردازیم که تصویر کارت بریده شده آیا درست است یا نیاز به دوران ۱۸۰ درجه دارد.همچنین به این نکته پرداخته می شود که تصویر ورودی آیا کارت ملی است یا خیر، که در صورت عدم شباهت داشتن template با عکس ورودی به اندازه محدوده مشخص شده(شباهت کمتر از ۰.۲۲) تشخیص داده می شود که تصویر کارت ورودی مربوط به کارت بانکی است . در تابع ابتدا تصویر ورودی را خاکستری می نماییم. همچنین template مدنظر کارت ملی را خوانده تا جلوتر با عکس ورودی مقایسه شود. سپس عکس

template را به سایز عکس کارت ها که ۳۵۰*۶۰۰ است می بریم تا مقایسه به درستی انجام شود. حال عکس template را خاکستری کرده ، یک عکس جدید با دوران ۱۸۰ درجه از آن می سازیم تا تطابق آن را با عکس ورودی بررسی کنیم و در صورت تطابق عکس را دوران دهیم. حال با استفاده از histogram_matching به همسان سازی طیف رنگی و نزدیکتر نمودن روشنایی تصویر ورودی خاکستری شده نسبت به تصویر الگو می پردازیم. حال روی تصویر خروجی مرحله قبل دستور زیر را اعمال کرده تا به فرمت ۸بیتی در آمده و درنهایت دوباره تصویر را ۳کاناله می نماییم تا بتوانیم با تصویر الگو(template اصلی) مقایسه نماییم. حال برای مقایسه قسمت های مشترک کارت ملی که شمال نوار های نام و نام خانوادگی است را جدا میکنیم و به تابع میدهیم. همین کار را برای فرمت برعکس هم مقایسه می نماییم. در نهایت بر اساس محدوده شباهت ، نوع کارت را مشخص کرده ایم. اگر بانکی بود که تابع مدنظر آن را صدا میزنیم. در صورتی هم که کارت ملی بود، لیبل مدنظر را مشخص نموده و اگر نیاز به چرخش داشت، آن را براساس مقایسه threshold شان میچرخانیم.

در نهایت عکس نهایی و label تصویر را برمی گردانیم.

6) Find_numbers:

```
def find_numbers(source_image, template, threshold, number, bank):
    if number==6 and bank==1:
        filter = np.array([[-1, -1, -1], [-1, 9, -1], [-1, -1, -1]])
        grayimg = cv2.cvtColor(source_image, cv2.COLOR_BGR2GRAY)
        grayimg = cv2.filter2D(grayimg,-1,filter)
        grayimg = cv2.cvtColor(grayimg, cv2.COLOR_GRAY2BGR)
        grayimg_T = cv2.cvtColor(template, cv2.COLOR_BGR2GRAY)
        grayimg_T = cv2.filter2D(grayimg_T,-1,filter)
        grayimg_T = cv2.cvtColor(grayimg_T, cv2.COLOR_GRAY2BGR)
    else:
        grayimg = cv2.cvtColor(source_image, cv2.COLOR_BGR2GRAY)
        grayimg = cv2.cvtColor(grayimg, cv2.COLOR_GRAY2BGR)
        grayimg_T = cv2.cvtColor(template, cv2.COLOR_BGR2GRAY)
        grayimg_T = cv2.cvtColor(grayimg_T, cv2.COLOR_GRAY2BGR)

    result = cv2.matchTemplate(np.array(grayimg) , grayimg_T, cv2.TM_CCOEFF_NORMED)
    locations = np.where(result >= threshold)
    finals = []
    for loc in zip(*locations[::-1]):
        already_added = False
        for final in finals:
            if abs(final[0][0]-loc[0]) < 5 :
                already_added = True
                break
        if already_added:
            continue
        else:
            finals.append([loc,number])
    return finals
```

این تابع مخصوص پیدا کردن تمام نقاط مشابه با یک رقم در تصویر است. ورودی های این تابع به شرح زیر هستند:

Sourc_image:تصویر وروری

Template:تصویر نمونه برای یافتن رقم مورد نظر

Threshold :حد شباهت مورد نظر برای پذیرفتن یک تصویر به عنوان آن عدد

Number :رقم مورد نظر برای استخراج

Bank:این پارامتر مخصوص کارت های بانکی است که مشخص میکند از کدام الگوی بانکی برای این رقم در حال استفاده هستیم

کارکرد این تابع به این صورت است که اگر رقم ۶ و بانک الگو ۱ باشد ابتدا یک فیلتر sharp کننده روی تصویر و الگو میزند. و در غیر این صورت فقط تصویر را خاکستری کرده و بعد سه کاناله میکند. سپس روی تصویر و الگو template matching میزنیم و نقاطی را که از threshold بیشتر یا برابر شباهت دارند را به عنوان کاندیدای آن رقم نگه میداریم. بعد داخل یک حلقه نقاطی را که یک عدد با فاصله ی چند پیکسلی چندین بار تشخیص داده شده و واحد میکنیم و نهایتا یک آرایه برمیگردانیم که هر عضوش یک مختصات بالا چپ دارد و یک مقدار عددی رقم([loc,number]).

7)**find_id_outter:**

```
def find_id_outter(src,number):
    gray = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)
    cgray = cv2.cvtColor(gray,cv2.COLOR_GRAY2BGR)
    path = 'templates/numbers/persian/'+str(number)+'.jpg'
    template = cv2.imread(path)
    threshold = 0
    if number == 1:
        threshold = 0.82
    elif number == 9:
        threshold = 0.85
    elif number ==3 or number ==8:
        threshold = 0.8
    elif number == 7 :
        threshold = 0.8
    elif number == 2:
        threshold = 0.85
    elif number == 0:
        threshold = 0.8
    else:
        threshold = 0.7
    result = find_numbers(cgray, template, threshold, number, None)
    return result
```

در این تابع تصویر کارت ملی و رقم مورد جستجو به عنوان ورودی داده میشوند. تصویر خاکستری و سپس سه کاناله میشود، تصویر الگوی رقم خوانده شده و بعد متناسب با رقم داده شده مقدار trheshold را معین میکنیم. نهایتا هم تصویر مبدا، الگوی رقم، trhreshold، رقم و مقدار None را به عنوان Bank به تابع find_number میدهیم و همان result ای که میدهد را برمیگردانیم.

8) **extract_ID_number:**

```
def extract_ID_number(img):
    pic = img[80:110,320:476]
    arr=[]
    for i in range(0,10):
        res =  find_id_outter(pic,i)
        arr.extend(res)
    output=sorted(arr,key=lambda x:x[0][0])
    code_number = ""
    for item in output:
        code_number+=str(item[1])
    return code_number
```

در تابع مدنظر ابتدا تصویر کارت ملی را ورودی را میگیریم و سپس محدوده شماره ملی را جدا نموده (با توجه به این نکته که در همه کارت های ملی، شماره ملی در این محدوده مشخص است) سپس روی تصویر بررسی انجام میدهیم و اعداد ۰ تا۹ را مقایسه می کنیم. تابع برای یافتن هر رقم در محدوده تصویر مشخص شده را صدا زده و مختصات و نوع رقم را ذخیره نموده ایم. حال مقادیر ذخیره شده را براساس مختصات طولی مرتب کرده و در نهایت بعد دوم آرایه که عدد داخل آن مختصات است را در رشته ای به ترتیب ریخته و برمیگردانیم.

9)**find_bank_outter:**

```
def find_bank_outter(src,number):
    gray = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)
    cgray = cv2.cvtColor(gray,cv2.COLOR_GRAY2BGR)
    types = [[0.8, 0.75, 0.7, 0.75, 0.75, 0.75],
             [0.8, 0.8, 0.7, 0.7, 0.7, 0.75,0.7],
             [0.7, 0.65, 0.67, 0.6, 0.7, 0.6],
             [0.8, 0.85, 0.7, 0.7, 0.7, 0.7,0.7, 0.7],
             [0.75, 0.7, 0.7, 0.7, 0.7, 0.7],
             [0.8, 0.82, 0.8, 0.75, 0.75, 0.7, 0.73, 0.73],
             [0.8, 0.75, 0.75, 0.75, 0.75, 0.8, 0.8, 0.7, 0.8, 0.75],
             [0.8, 0.75, 0.7, 0.75, 0.75],
             [0.7, 0.82, 0.8, 0.8, 0.8, 0.85, 0.7],
             [0.75,0.82,0.7,0.75, 0.75, 0.7, 0.7, 0.6]]
    final_result=[]
    for i in range(len(types[number])):
        bank = i
        path = 'templates/numbers/english/'+str(number)+'/lock/'+str(bank)+'.jpg'
        template = cv2.imread(path)
        threshold = types[number][bank]
        result = find_numbers(cgray, template, threshold, number, bank)
        if len(result) > len(final_result):
            final_result = result
    return final_result
```

در این تابع تصویر کارت بانکی و رقم مورد جستجو به عنوان ورودی داده میشوند. تصویر مبدا، خاکستری و سپس سه کاناله میشود، آرایه ی دو بعدی threshold به ازای رقم و نوع الگوی بانکی مقدار دهی میشود(سطر های مختلف ارقام متفاوت و ستون های متقاوت الگوی بانکی های متفاوت هستند) و بع تصویر الگوی رقم خوانده میشود. نهایتا هم تصویر مبدا، الگوی رقم، trhreshold، رقم و مقدار آن عددی بانک الگو را به عنوان Bank به تابع find_number میدهیم و همان result ای که میدهد را برمیگردانیم.

10) **extract_Bank_number:**

```
def extract_Bank_number(img):
    pic = img[:,:]
    arr=[]
    for i in range(0,10):
        res =  find_bank_outter(pic,i)
        arr.extend(res)
    output=sorted(arr,key=lambda x:x[0][0])
    top_left = output[0][0]
    top_left = [top_left[0]+20, top_left[1]+120]
    bottom_right = output[len(output)-1][0]
    bottom_right = [bottom_right[0]+65, bottom_right[1]+165]
    code_number = ""
    for item in output:
        code_number+=str(item[1])
    return [code_number, [top_left,bottom_right]]
```

در این تابع تصویر کارت بانکی به عنوان ورودی گرفته میشود. بعد از رقم ۰ تا ۹ یک حلقه زده میشود و در هر بار با فرستادن تصویر مبدا و رقم مورد نظر به تابع find_bank_outter آرایه تصاویر بدست آمده ی آن رقم به همراه مکان هایشان برمیگردد. و در ادامه داخل حلقه آن نتیجه به آرایه ی اصلی arr اضافه میشود تا در پایان حلقه همه ی اعداد خوانده شده از هر رقم را در آن داشته باشیم. سپس در بیرون حلقه ابتدا arr را بر حسب مقدار x ای که لوکیشن هر رقم دارد مرتب میکنیم تا ارقام از چپ به راست مرتب شده و عدد کارت قابل خواندن باشد:

```
output=sorted(arr,key=lambda x:x[0][0])
```

بعد محل عضو اول و آخر را با کمی جابجایی جدا میکنیم تا بتوانیم محدوده دور اعداد را با مستطیل مشخص کنیم. و بعد با یک حلقه روی آرایه ی مرتب شده ارقام را به ترتیب خوانده و به متغیر code_number اضافه میکنیم.

نهایتا عدد خوانده شده را به همراه دو نقطه ی قطری بالاچپ و پایین راست محدوده ی اعداد برمیگردانیم:

```
return [code_number, [top_left,bottom_right]]
```

11)**predict**

```
def predict(path):
    resized_image = extract_card(path)
    [rotated_image, label] = adjust_rotation_id(resized_image)
    title = ""
    if label == "ID Card":
        cv2.rectangle(rotated_image,(320,80), (475,110), (255,0,0),1)
        code = extract_ID_number(rotated_image)
        title = label + "/ Persian Numbers / " + code
    else:
        [code, [top_left, bottom_right]] = extract_Bank_number(rotated_image[125:290,30:580])
        cv2.rectangle(rotated_image, top_left, bottom_right, (0,255,0),2)
        code = ' '.join([code[i:i+4] for i in range(0,len(code), 4)])
        title = label + "/ English Numbers / " + code

    plt.imshow(cv2.cvtColor(rotated_image, cv2.COLOR_BGR2RGB))
    plt.title(title), plt.xticks([]), plt.yticks([])
    plt.show()

    return title
```

در تابع نوشته شده،در واقع از تمام توابع پیشین برای آماده سازی خروجی مدنظر و بررسی نکات مورد نیاز استفاده می شود .بدین گونه که ابتدا آدرس تصویر اولیه به تابع extract_card ارسال می شود، آنگاه برای بررسی صاف بودن یا نیاز به دوران داشتن به علاوه مشخص شدن نوع کارت و label آن، به تابع adjust_rotation_id فرستاده می شود. حال براساس نوع کارت و label مشخص شده، به بررسی حالات میپردازیم. اگر کارتمان ملی بود، با توجه به یکسان بودن محدوده شماره ملی، دور آن محدوده را با تابع cv2.rectangle مستطیل میکشیم. سپس کدملی را با تابع extract_ID_number مشخص می نماییم. و درنهایت متن مورد نیاز برای آن عکس را براساس این نکته که اعداد روی کارت ملی فارسی اند ، مشخص میکنیم. در صورتی هم که label مان نشان دهنده کارت بانکی بود، نقاط بالا چپ و پایین راست به همراه شماره کارت را با تابع extract_Bank_number بدست آورده و سپس دور آن محدوده را مستطیل میکشیم. درنهایت هم برای مشخص تر شدن شماره کارت ، آنرا ۴ تا ۴ جدا می نماییم و ذخیره میکنیم. عنوان عکس را هم براساس این نکته که اعداد کارت بانکی انگلیسی اند مینویسیم. در نهایت تصویر را خروجی و نمایش میدهیم.

نهایتا هم سلول آخر فایل نوتبوک برای خواندن تمام فایل های داخل یک دایرکتوری و فرستادن آن به تابع predict استفاده میشود تا تمام داده های ذخیره شده برای تست پردازش شده و بررسی شوند:

```
directory = "inputs/"
for filename in os.listdir(directory):
    image_path = os.path.join(directory, filename)
    prediction = predict(image_path)
    print(prediction)
```

طبق بررسی نتایج اجرا شده روی ۴۸ تصویر مشاهده میشود که تمام تصاویر به درستی جداسازی شده و محدوده و اعداد آنها هم به درستی تشخیص داده میشوند: