|  |
| --- |
| import cv2 |
|  | import numpy as np |
|  |  |
|  |  |
|  | class PreProcessor: |
|  | def \_\_init\_\_(self, width, height, inter=cv2.INTER\_AREA): |
|  | self.width = width |
|  | self.height = height |
|  | self.inter = inter |
|  |  |
|  | def preprocess(self, input\_image): |
|  | try: |
|  | # if in train time with 100 epoch and size 50 get bad result remove all this line und just resize , convert all dataset befor train network |
|  | # input\_image = cv2.imread(image\_paths) |
|  | # Type = image\_paths.split(os.path.sep)[-2] |
|  | image\_blur = cv2.GaussianBlur(input\_image, (7, 7), 0) |
|  | image\_blur\_hsv = cv2.cvtColor(image\_blur, cv2.COLOR\_RGB2HSV) |
|  | min\_red = np.array([80, 60, 140]) |
|  | max\_red = np.array([255, 255, 255]) |
|  | image\_red1 = cv2.inRange(image\_blur\_hsv, min\_red, max\_red) |
|  | big\_contour, mask = self.find\_biggest\_contour(image\_red1) |
|  | (x, y), radius = cv2.minEnclosingCircle(big\_contour) |
|  | center = (int(x), int(y)) |
|  | radius = int(radius) |
|  | imCircle = input\_image.copy() |
|  | cv2.circle(imCircle, center, radius, (0, 255, 0), 1) |
|  | height, width, channels = imCircle.shape |
|  | border = [0, 0, 0, 0] |
|  | if center[0] + radius > width: |
|  | extera = (center[0] + radius) - width |
|  | border[3] = extera + 1 |
|  |  |
|  | if (center[0] - radius < 0): |
|  | extera = width - (center[0] + radius) |
|  | border[2] = extera + 1 |
|  |  |
|  | if center[1] + radius > height: |
|  | extera = (center[1] + radius) - height |
|  | border[1] = extera + 1 |
|  |  |
|  | if center[1] + radius < 0: |
|  | extera = height - (center[1] + radius) |
|  | border[0] = extera + 1 |
|  |  |
|  | y = center[1] - radius |
|  | if y < 0: |
|  | y = 0 |
|  | y2 = center[1] + radius |
|  | x = center[0] - radius |
|  | if x < 0: |
|  | x = 0 |
|  |  |
|  | x2 = center[0] + radius |
|  |  |
|  | cropped\_image = input\_image[y:y2, x:x2] |
|  |  |
|  | return cv2.resize(cropped\_image, (self.width, self.height), |
|  | interpolation=self.inter) |
|  | except Exception as a: |
|  | print("preprocessor", a) |
|  |  |
|  | def find\_biggest\_contour(self, image): |
|  | image = image.copy() |
|  | s, contours, hierarchy = cv2.findContours(image, cv2.RETR\_LIST, cv2.CHAIN\_APPROX\_SIMPLE) |
|  | biggest\_contour = max(contours, key=cv2.contourArea) |
|  | mask = np.zeros(image.shape, np.uint8) |
|  | cv2.drawContours(mask, [biggest\_contour], -1, 255, -1) |
|  | return biggest\_contour, mask |
|  |  |
|  | def overlay\_mask(self, mask, image): |
|  | rgb\_mask = cv2.cvtColor(mask, cv2.COLOR\_GRAY2RGB) |
|  | img = cv2.addWeighted(rgb\_mask, 0.5, image, 0.5, 0) |
|  | # show(img) |