به نام خداوند بخشاینده مهربان

تشخیص پلاک خودرو

دانشجویان: راضیه عسگری، مبینا علی بهرامی

رشته: شبکه های کامپیوتری

استاد راهنما: جناب آقای مجتبی فر

درس مربوطه: پردازش تصویر

بهار 1404

مقدمه

تشخیص پلاک خودرو یکی از کاربردهای مهم بینایی ماشین است که در بسیاری از حوزه‌ها از جمله نظارت ترافیکی، کنترل دسترسی و امنیت استفاده می‌شود. این کد به شما این امکان را می‌دهد که پلاک‌های خودرو را شناسایی کرده و متن آن‌ها را استخراج کنید. این فرآیند شامل مراحل مختلفی از جمله پردازش تصویر، شناسایی لبه‌ها، تشخیص کانتورها و در نهایت استخراج متن از تصویر پلاک است.

پیش‌نیازها

برای اجرای این کد، به برخی از نرم‌افزارها و کتابخانه‌ها نیاز دارید:

- Python 3.x: زبان برنامه‌نویسی که این کد در آن نوشته شده است.

- کتابخانه‌های مورد نیاز:

- OpenCV: کتابخانه‌ای برای پردازش تصویر که قابلیت‌های متعددی را ارائه می‌دهد.

- pytesseract: این کتابخانه یک رابط برای Tesseract OCR است که می‌تواند متن را از تصاویر استخراج کند.

- imutils: کتابخانه‌ای برای تسهیل کار با OpenCV.

- نصب Tesseract OCR: برای استفاده از Tesseract، باید آن را بر روی سیستم خود نصب کنید

توضیحات کد

1. وارد کردن کتابخانه‌ها

python

```
import cv2
import pytesseract
import imutils
import os
```

- cv2: این کتابخانه اصلی برای پردازش تصویر است و ابزارهای متنوعی برای کار با تصاویر و ویدیوها فراهم می‌کند.

- pytesseract: این کتابخانه به شما امکان می‌دهد تا از قابلیت‌های Tesseract OCR برای شناسایی متن در تصاویر استفاده کنید.

- imutils: این کتابخانه شامل توابع کمکی است که کار با OpenCV را آسان‌تر می‌کند، مانند تغییر اندازه تصاویر.

- os: برای کار با سیستم فایل و مدیریت مسیرها و پوشه‌ها استفاده می‌شود.

2. تنظیم مسیر Tesseract (اختیاری)

- اگر ویندوز استفاده می‌کنی و Tesseract نصب داری، این خط رو از حالت کامنت در بیار:
- pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'
  - این خط برای تعیین مسیر نصب Tesseract در ویندوز است. اگر Tesseract را نصب کرده‌اید، باید مسیر صحیح را وارد کنید تا کد بتواند به آن دسترسی پیدا کند.

3. تابع preprocess_image(image)

```
def preprocess_image(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blur = cv2.bilateralFilter(gray, 11, 17, 17)
    edged = cv2.Canny(blur, 30, 200)
```

1. تبدیل تصویر به رنگ خاکستری

```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

- cv2.cvtColor: این تابع برای تبدیل تصویر از فضای رنگی BGR (Blue, Green, Red) به فضای رنگی خاکستری (Gray) استفاده می‌شود.
- هدف: تبدیل به خاکستری به کاهش پیچیدگی تصویر کمک می‌کند و باعث می‌شود که پردازش‌های بعدی (مانند شناسایی لبه‌ها) بهتر و سریع‌تر انجام شوند. در تصاویر رنگی، اطلاعات رنگی ممکن است باعث اختلال در شناسایی ویژگی‌ها شود.

2. اعمال فیلتر دوطرفه

```
blur = cv2.bilateralFilter(gray, 11, 17, 17)
```

- cv2.bilateralFilter: این تابع برای کاهش نویز تصویر با حفظ لبه‌ها استفاده می‌شود. فیلتر دوطرفه به این دلیل نامیده می‌شود که به هر پیکسل به‌طور جداگانه بر اساس شدت و رنگ پیکسل‌های اطراف آن وزن می‌دهد.

پارامترها:

  - 11: اندازه هسته فیلتر. هرچه این مقدار بزرگ‌تر باشد، تأثیر فیلتر بیشتر خواهد بود.
  - 17: انحراف استاندارد در فضای شدت رنگ.
  - 17: انحراف استاندارد در فضای مختصات (فضای فضایی).

– هدف: این فیلتر به کاهش نویز و حفظ لبه‌ها کمک می‌کند. این کار باعث می‌شود که شناسایی لبه‌ها در مرحله بعدی دقیق‌تر باشد.

3. شناسایی لبه‌ها با استفاده از الگوریتم Canny

```
edged = cv2.Canny(blur, 30, 200)
```

- cv2.Canny: این تابع برای شناسایی لبه‌ها در تصویر استفاده می‌شود. الگوریتم Canny یکی از معروف‌ترین و مؤثرترین الگوریتم‌ها برای شناسایی لبه‌ها است.

پارامترها:

- 30: حد پایین برای شناسایی لبه‌ها. پیکسل‌هایی که شدت آن‌ها بیشتر از این مقدار باشد به عنوان لبه شناسایی می‌شوند.
- 200: حد بالا برای شناسایی لبه‌ها. این مقدار به عنوان مرز نهایی برای شناسایی لبه‌ها استفاده می‌شود.

– هدف: با استفاده از این الگوریتم، لبه‌های موجود در تصویر شناسایی می‌شوند و تصویر نهایی که فقط شامل لبه‌ها است به متغیر edged اختصاص داده می‌شود.

4. تابع find_plate_contour(edged, image_shape)

```
def find_plate_contour(edged, image_shape):
    cnts = cv2.findContours(edged.copy(), cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)
    height, width = image_shape[:2]

    for c in sorted(cnts, key=cv2.contourArea, reverse=True):
        peri = cv2.arcLength(c, True)
        approx = cv2.approxPolyDP(c, 0.018 * peri, True)

        if len(approx) == 4:
            x, y, w, h = cv2.boundingRect(approx)
            aspect_ratio = w / float(h)
            area = w * h
            image_area = width * height
```

```
if 2 < aspect_ratio < 6 and (0.01 * image_area) < area < (0.15 * image_area):
return approx
return None
```

تحلیل تابع find_plate_contour

– پیدا کردن کانتورها: با استفاده از cv2.findContours، کانتورهای موجود در تصویر لبه‌ها شناسایی می‌شوند. این تابع کانتورهایی که در تصویر وجود دارند را به صورت لیستی برمی‌گرداند.

– فیلتر کردن کانتورها: در این مرحله، کانتورها بر اساس ابعاد و نسبت‌های آن‌ها فیلتر می‌شوند. فقط کانتوری که به شکل مستطیل نزدیک‌تر است و نسبت ابعادی معقولی دارد، انتخاب می‌شود.

```
python def find_plate_contour(edged, image_shape): cnts = cv2.findContours(edged.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE) cnts = imutils.grab_contours(cnts) height, width = image_shape[:2]

for c in sorted(cnts, key=cv2.contourArea, reverse=True):
peri = cv2.arcLength(c, True)
approx = cv2.approxPolyDP(c, 0.018 * peri, True)

if len(approx) == 4:
x, y, w, h = cv2.boundingRect(approx)
aspect_ratio = w / float(h)
area = w * h
image_area = width * height

if 2 < aspect_ratio < 6 and (0.01 * image_area) < area < (0.15 * image_area):
return approx
return None
```

- **ورودی‌های تابع**

- edged: تصویر ورودی که حاوی لبه‌ها است. این تصویر معمولاً از یک مرحله پیش‌پردازش به دست می‌آید.
- image_shape: ابعاد تصویر ورودی که به صورت (height, width, channels) است. در اینجا نیاز به ارتفاع و عرض تصویر داریم.

- **پیدا کردن کانتورها**

```
python cnts = cv2.findContours(edged.copy(), cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
```

- cv2.findContours : این تابع کانتورها را از تصویر لبه‌ها شناسایی می‌کند. دو پارامتر cv2.RETR_TREE و cv2.CHAIN_APPROX_SIMPLE به ترتیب برای بازیابی ساختار درختی کانتورها و کاهش تعداد نقاط کانتور به کار می‌روند.

- **گرفتن کانتورها**

```
python cnts = imutils.grab_contours(cnts)
```

- imutils.grab_contours :این تابع برای سازگاری با نسخه‌های مختلف OpenCV استفاده می‌شود و یک لیست از کانتورها را برمی‌گرداند.

- **استخراج ابعاد تصویر**

```
python height, width = image_shape[:2]
```

- استخراج ارتفاع و عرض: ابعاد تصویر (ارتفاع و عرض) از شکل تصویر استخراج می‌شود تا در مراحل بعدی از آن استفاده شود.

- **پردازش کانتورها**

```
python for c in sorted(cnts, key=cv2.contourArea, reverse=True)
```

- مرتب‌سازی کانتورها: کانتورها بر اساس مساحت آن‌ها به ترتیب نزولی مرتب می‌شوند. این کار به این دلیل است که معمولاً پلاک‌ها بزرگ‌تر از سایر اشیاء در تصویر هستند.

➢ **محاسبه طول محیط کانتور**

python peri = cv2.arcLength(c, True)

- cv2.arcLength : این تابع طول محیط کانتور را محاسبه می‌کند. پارامتر True به این معنی است که کانتور به صورت بسته در نظر گرفته می‌شود.

➢ **تقریب‌سازی کانتور**

python approx = cv2.approxPolyDP(c, 0.018 * peri, True)

- cv2.approxPolyDP : این تابع کانتور را به یک چندضلعی تقریب می‌زند. پارامتر 0.018 * peri میزان دقت تقریب را تعیین می‌کند. یعنی اگر فاصله نقاط کانتور از چندضلعی بیشتر از این مقدار باشد، آن نقاط حذف می‌شوند.

➢ **بررسی شرایط پلاک**

python if len(approx) == 4

- تعداد گوشه‌ها: بررسی می‌شود که آیا تعداد گوشه‌های تقریب‌زده شده برابر با ۴ است یا خیر. پلاک‌ها معمولاً مستطیل شکل هستند و به همین دلیل دارای ۴ گوشه هستند.

➢ **محاسبه ابعاد و نسبت ابعاد**

– محاسبه ابعاد و نسبت‌ها: با استفاده از ابعاد کانتور، نسبت ابعاد و مساحت آن محاسبه می‌شود تا اطمینان حاصل شود که کانتور مربوط به پلاک خودرو است.

python x, y, w, h = cv2.boundingRect(approx) aspect_ratio = w /

float(h) area = w * h image_area = width * height

- cv2.boundingRect : این تابع یک مستطیل حداقلی را که کانتور را در بر می‌گیرد محاسبه می‌کند.
- نسبت ابعاد: نسبت عرض به ارتفاع محاسبه می‌شود. این نسبت برای شناسایی پلاک‌ها مهم است.
- محیط کانتور: مساحت مستطیل حداقلی محاسبه می‌شود.
- محیط تصویر: مساحت کل تصویر نیز محاسبه می‌شود.

تحلیل شرط

python if 2 < aspect_ratio < 6 and (0.01 * image_area) < area < (0.15 * image_area): return approx

توضیحات:

❖ **شرط نسبت ابعاد:**

- 2 < aspect_ratio < 6 : این شرط بررسی می‌کند که نسبت عرض به ارتفاع کانتور (پلاک) بین ۲ و ۶ باشد.
- نسبت ابعاد :
  - اگر نسبت ابعاد کمتر از ۲ باشد، شکل ممکن است به صورت مستطیل باریک یا حتی به صورت خطی باشد که نمی‌تواند پلاک باشد.
  - اگر نسبت ابعاد بیشتر از ۶ باشد، شکل به احتمال زیاد بسیار عریض است و نمی‌تواند پلاک خودرو باشد.

❖ **شرط محیط:**

- (0.01 * image_area) < area < (0.15 * image_area): این شرط بررسی می‌کند که محیط کانتور بین ٪۱ تا ٪۱۵ از مساحت کل تصویر باشد.
- مساحت :
  - اگر محیط کانتور کمتر از ٪۱ از مساحت کل تصویر باشد، ممکن است کانتور ناشی از نویز یا اشیاء کوچک در تصویر باشد.
  - اگر محیط کانتور بیشتر از ٪۱۵ از مساحت کل تصویر باشد، احتمالاً کانتور یک شیء بزرگ‌تر (مانند یک خودرو) یا پس‌زمینه است و نمی‌تواند پلاک خودرو باشد.

نتیجه شرط

اگر هر دو شرط برقرار باشند، کانتوری که نمایانگر پلاک خودرو است به عنوان خروجی تابع بازگردانده می‌شود. این شرایط به کاهش تعداد کانتورهای نادرست کمک می‌کند و دقت شناسایی پلاک را افزایش می‌دهد.

اثر حذف شرط 2 < aspect_ratio < 6

اگر شرط 2 < aspect_ratio < 6 حذف شود:

1. **شناسایی کانتورها:**
   - ممکن است کانتورها با نسبت ابعاد نامناسب (کمتر از ۲ یا بیشتر از ۶) به عنوان پلاک شناسایی شوند. به عنوان مثال، کانتوری با نسبت ابعاد ۱ (مستطیل باریک) یا ۷ (مستطیل عریض) ممکن است به عنوان پلاک شناسایی شود.
2. **افزایش اشتباهات شناسایی:**
   - احتمال شناسایی اشیاء غیر مرتبط به عنوان پلاک افزایش می‌یابد. این ممکن است شامل اشیاء مختلف در تصویر یا حتی نویزهای تصویر باشد که به دلیل نسبت‌های ابعادی نامناسب شناسایی می‌شوند.
3. **کاهش دقت:**
   - بدون این شرط، دقت کلی الگوریتم در شناسایی پلاک‌ها کاهش می‌یابد، زیرا ممکن است الگوریتم به اشیاء غیر مرتبط توجه کند و نتایج نادرستی تولید کند.

5. تابع ocr_plate(plate_img)

```
def ocr_plate(plate_img):
    gray = cv2.cvtColor(plate_img, cv2.COLOR_BGR2GRAY)
    _, thresh = cv2.threshold(gray, 150, 255, cv2.THRESH_BINARY)
    config = r'--oem 3 --psm 8 -c tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'
    text = pytesseract.image_to_string(thresh, config=config)
```

```
return text.strip()
```

– آستانه‌گذاری تصویر: این مرحله شامل تبدیل تصویر به سیاه و سفید است تا زمینه و متن بهتر تفکیک شوند.

cv2.threshold : این تابع برای تبدیل تصویر به حالت دو رنگ (سیاه و سفید) استفاده می‌شود. این کار به تفکیک بهتر متن از پس‌زمینه کمک می‌کند.

- **پارامترهای تابع:**
  - gray : تصویر ورودی که به صورت خاکستری (Gray) است.
  - 150 : مقدار آستانه، که تعیین می‌کند چه پیکسل‌هایی باید به رنگ سفید (255) و چه پیکسل‌هایی باید به رنگ سیاه (0) تغییر یابند. اگر مقدار شدت روشنایی یک پیکسل بیشتر از 150 باشد، به رنگ سفید تغییر می‌کند و در غیر این صورت به رنگ سیاه.
  - 255 : مقدار حداکثری که برای پیکسل‌های سفید تعیین می‌شود.
  - cv2.THRESH_BINARY : نوع آستانه‌گذاری. در اینجا، نوع BINARY به این معنی است که پیکسل‌ها یا به رنگ سفید (255) یا به رنگ سیاه (0) تبدیل می‌شوند.
- **خروجی:**
  - متغیری که مقدار بازگشتی اول (ارزیابی کیفیت آستانه‌گذاری) را نگه می‌دارد، اما در اینجا استفاده نمی‌شود.
  - thresh:تصویر پردازش‌شده که فقط شامل دو رنگ (سیاه و سفید) است و برای تشخیص متن مناسب‌تر است.

– پیکربندی Tesseract: تنظیماتی برای شناسایی متن، مانند محدود کردن کاراکترهای مجاز به حروف و اعداد به کار می‌رود. این کار به بهبود دقت شناسایی کمک می‌کند.

- Config : این متغیر شامل تنظیماتی است که برای Tesseract استفاده می‌شود تا دقت شناسایی متن افزایش یابد.
- **پارامترهای پیکربندی:**
  - --oem 3 :این گزینه به Tesseract می‌گوید که از مد OCR (OCR Engine Mode) 3استفاده کند. این حالت می‌تواند از هر دو موتور (مدل‌های قدیمی و جدید) استفاده کند که معمولاً دقت بهتری دارد.

- --psm 8 : این گزینه به Tesseract می‌گوید که تصویر ورودی یک خط افقی از متن است . PSMمخفف Page Segmentation Mode است و عدد 8 نشان‌دهنده این است که Tesseractباید به‌عنوان یک خط از متن به تصویر نگاه کند.
- -c tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789: این گزینه به Tesseract می‌گوید که فقط کاراکترهای موجود در این لیست را شناسایی کند. با محدود کردن کاراکترها به حروف بزرگ و اعداد، دقت شناسایی افزایش می‌یابد و احتمال شناسایی کاراکترهای ناخواسته کاهش می‌یابد.

– استخراج متن: با استفاده از pytesseract، متن موجود در تصویر پلاک استخراج می‌شود و به صورت رشته‌ای برمی‌گردد.

6. تابع process_images_in_folder(folder_path)

```
def process_images_in_folder(folder_path):
    if not os.path.exists(folder_path):
        print(f"❌ Folder not found: {folder_path}")
        return

    for filename in os.listdir(folder_path):
        if filename.lower().endswith(('.png', '.jpg', '.jpeg')):
            img_path = os.path.join(folder_path, filename)
            image = cv2.imread(img_path)
            if image is None:
                print(f"❌ Cannot load image: {filename}")
                continue
```

```
resized = imutils.resize(image, width=600)
edged = preprocess_image(resized)
plate_contour = find_plate_contour(edged, resized.shape)

if plate_contour is not None:
x, y, w, h = cv2.boundingRect(plate_contour)
cv2.rectangle(resized, (x, y), (x+w, y+h), (0, 255, 0), 2)
plate_img = resized[y:y+h, x:x+w]
text = ocr_plate(plate_img)

print(f"✅ {filename}: Detected Plate Text: {text if text else 'No text detected'}")
cv2.imshow("Detected Plate", resized)
cv2.waitKey(0)
else:
print(f"🚫 {filename}: No plate detected.")

cv2.destroyAllWindows()
```

– بررسی وجود پوشه: ابتدا بررسی می‌شود که آیا پوشه‌ای که تصاویر در آن قرار دارد وجود دارد یا خیر. در صورت عدم وجود، یک پیام خطا چاپ می‌شود.

– پردازش تصاویر: برای هر تصویر در پوشه، تصویر خوانده می‌شود و در صورت امکان پردازش می‌شود. اگر تصویر بارگذاری نشود، پیام خطایی نمایش داده می‌شود.

- **resized = imutils.resize(image, width=600)**

توضیحات:

- imutils.resize : این تابع از کتابخانه imutils برای تغییر اندازه تصویر استفاده می‌شود. در اینجا، عرض تصویر به 600 پیکسل تنظیم می‌شود.
- مزایای تغییر اندازه:
  - کاهش زمان پردازش: با کوچک‌تر کردن ابعاد تصویر، زمان پردازش کاهش می‌یابد.
  - بهبود دقت: در برخی موارد، اندازه‌های بزرگ ممکن است باعث مشکلاتی مانند نویز اضافی یا بارگذاری بیش از حد داده شوند. تغییر اندازه به یک مقدار استاندارد می‌تواند دقت پردازش را بهبود بخشد.
- **پردازش تصویر برای شناسایی لبه‌ها**

python edged = preprocess_image(resized)

توضیحات:

- preprocess_image(resized) : این تابع که قبلاً تعریف شده است، تصویر ورودی (تصویر تغییر اندازه داده شده) را پردازش می‌کند و لبه‌ها را شناسایی می‌کند.
- خروجی: تصویر پردازش‌شده که فقط شامل لبه‌ها است، به متغیر edged اختصاص می‌یابد. این تصویر به عنوان ورودی برای شناسایی کانتورها استفاده می‌شود.
- پیدا کردن کانتور پلاک خودرو

- **python plate_contour = find_plate_contour(edged, resized.shape)**

توضیحات:

- find_plate_contour(edged, resized.shape) : این تابع کانتورهای موجود در تصویر لبه‌ها را پیدا می‌کند و تلاش می‌کند تا کانتوری که به شکل پلاک خودرو نزدیک‌تر است را شناسایی کند.
- خروجی: اگر کانتوری که به شرایط پلاک نزدیک است پیدا شود، به متغیر plate_contour اختصاص داده می‌شود. در غیر این صورت، مقدار آن None خواهد بود.

- **بررسی وجود کانتور پلاک**

python if plate_contour is not None

توضیحات:

- در اینجا بررسی می‌شود که آیا کانتوری برای پلاک پیدا شده است یا خیر. اگر کانتوری پیدا نشود، کد به بخش بعدی نمی‌رود.

- **محاسبه موقعیت و ابعاد کانتور**

python x, y, w, h = cv2.boundingRect(plate_contour)

توضیحات:

- cv2.boundingRect : این تابع یک مستطیل حداقلی را که کانتور را در بر می‌گیرد محاسبه می‌کند و مختصات آن(x) ، (yو ابعاد آن (عرض و ارتفاع) را به دست می‌آورد.

- **رسم مستطیل روی تصویر**

python cv2.rectangle(resized, (x, y), (x+w, y+h), (0, 255, 0), 2)

توضیحات:

- cv2.rectangle : این تابع یک مستطیل روی تصویر رسم می‌کند. در اینجا، مستطیل به رنگ سبز (RGB: 0, 255, 0)و با ضخامت 2 پیکسل رسم می‌شود تا پلاک شناسایی شده را نشان دهد.

- شناسایی پلاک: اگر پلاک شناسایی شود، تصویر پلاک با کادر سبز نمایش داده می‌شود و متن استخراج شده چاپ می‌شود. در غیر این صورت، پیام عدم شناسایی پلاک نمایش داده می‌شود.

- cv2.rectangle : این تابع یک مستطیل روی تصویر رسم می‌کند. در اینجا، مستطیل به رنگ سبز (RGB: 0, 255, 0)و با ضخامت 2 پیکسل رسم می‌شود تا پلاک شناسایی شده را نشان دهد.
- **plate_img = resized[y:y+h, x:x+w]**

توضیحات:

ایجاد تصویر پلاک: با استفاده از مختصات(x) ، (yو ابعاد(w) ، (h، ناحیه‌ای از تصویر که شامل پلاک خودرو است، استخراج می‌شود و به متغیر plate_img اختصاص داده می‌شود.

- **text = ocr_plate(plate_img)**

توضیحات:

ocr_plate(plate_img) : این تابع تصویر پلاک را به عنوان ورودی می‌گیرد و متن موجود در آن را با استفاده از Tesseract استخراج می‌کند. نتیجه به متغیر text اختصاص داده می‌شود.

7. بخش اصلی (if __name__ == "__main__":)

```
if __name__ == "__main__":
    script_dir = os.path.dirname(os.path.abspath(__file__))
    folder = os.path.join(script_dir, "data")
    print(f"📂 Looking for images in: {folder}")
    process_images_in_folder(folder)
```

- تنظیم مسیر پوشه: در اینجا مسیر پوشه‌ای که تصاویر در آن قرار دارند مشخص می‌شود و تابع پردازش تصاویر فراخوانی می‌شود.

نحوه استفاده

برای استفاده از این کد، مراحل زیر را دنبال کنید:

1. نصب Tesseract: Tesseract را بر روی سیستم خود نصب کنید و مسیر آن را در کد مشخص کنید (برای ویندوز).

2. قرار دادن تصاویر: تصاویر را در پوشه‌ای به نام data در کنار فایل اسکریپت قرار دهید. این تصاویر باید شامل پلاک‌های خودرو باشند.

3. اجرا کردن اسکریپت: اسکریپت را اجرا کنید. پس از اجرا، تصاویر پردازش می‌شوند و نتایج در کنسول چاپ می‌شوند.

نکات و توصیه‌ها

– کیفیت تصاویر: کیفیت تصاویر ورودی تأثیر زیادی بر دقت شناسایی پلاک دارد. سعی کنید از تصاویری با وضوح بالا استفاده کنید.

– تنظیمات پیش‌پردازش: ممکن است نیاز باشد پارامترهای مربوط به پیش‌پردازش تصاویر را بسته به شرایط نوری و کیفیت تصویر تنظیم کنید.

– تست با تصاویر مختلف: برای اطمینان از عملکرد صحیح الگوریتم، بهتر است کد را با انواع مختلف تصاویر پلاک آزمایش کنید.

نتیجه‌گیری

این کد به شما این امکان را می‌دهد که به سادگی پلاک خودروها را شناسایی و متن آن‌ها را استخراج کنید. با استفاده از ترکیب OpenCV وTesseract ، فرآیند تشخیص پلاک به طور خودکار انجام می‌شود. این کد می‌تواند در پروژه‌های مختلفی مانند نظارت بر ترافیک و کنترل دسترسی به کار رود.