

Advancing Parking Systems: A Performance Comparison of MobileNet and Canny in License Plate Detection

Suryani^{*1}, Husain², Faizal³

^{1,2,3}Dipa Makassar University, Jl. Perintis Kemerdekaan No.KM.9, Tamalanrea Indah, Kec. Tamalanrea, Kota Makassar

e-mail: ^{*1}suryani187@undipa.ac.id, ²husain@undipa.ac.id, ³F41241@undipa.ac.id

Abstract

Rapid advancements in technology, particularly in computer science, have driven progress in image processing, which plays a crucial role in daily life. This research focuses on object recognition through vehicle license plate detection, utilizing an image database to address human errors in recording vehicle numbers that can slow down parking system services. An automated system is proposed to enhance parking management, although challenges in accurately segmenting plates remain. Two segmentation methods are compared: the MobileNet architecture and the Canny algorithm. This study aims to evaluate the segmentation accuracy between the two methods. Canny for its edge detection capabilities that reduce noise, and MobileNet for its effectiveness as a deep learning-based approach. The system is implemented using Python, JavaScript, HTML, and CSS to modernize vehicle license plate segmentation. The results show that MobileNet significantly outperforms the Canny algorithm, achieving a lower Character Error Rate (CER) of 18.8%, compared to Canny's 50.96%, across 13 tested license plate samples. This finding demonstrates that MobileNet offers a more reliable and accurate approach for segmenting vehicle license plates, thereby contributing to the development of a more efficient and automated license plate recognition system.

Keywords— Segmentation, MobileNet, Algorithm Canny, CER Error, Vehicle License Plate

1. INTRODUCTION

The rapid development of technology in the scope of computer science has a significant impact on various aspects of daily life, particularly in image processing techniques. Image processing is a general term for various techniques used to manipulate and modify images in different methods [1]. Image processing has become an increasingly relevant field, particularly in applications related to object recognition, which is now widely used in security systems, traffic management, and business process automation. Image segmentation is one of the methods in digital image processing used to distinguish objects in an input image; in other words, it is the process of separating objects from their background [2]. Image segmentation plays a crucial role in enhancing the accuracy and efficiency of object recognition systems.

One practical application of this technique is vehicle license plate detection. In this context, human errors in recording vehicle numbers can lead to delays in parking services, making it necessary to have an automated solution that can improve system efficiency. By utilizing a picture database, this research aims to develop an automatic vehicle license plate recognition system so that it can minimize recording errors and improve parking management.

However, the challenge faced is achieving accurate vehicle license plate segmentation. This can be influenced by field conditions and the limitations of the algorithms. This research compares two segmentation methods. The first is the MobileNet architecture, and the second is the Canny algorithm to evaluate segmentation accuracy in the vehicle license plate detection method. The MobileNet architecture is a type of Convolutional Neural Network (CNN) that

addresses the need for processing large amounts of data by dividing convolution into depthwise convolution and pointwise convolution [3]. On the other hand, the Canny algorithm is a technique capable of detecting object edges in an image, which is susceptible to noise. [4]. One way to reduce the effects of noise in an image is by using Gaussian blur [5], [6], [7]. Gaussian blur is a Gaussian processing technique used to blur an image by applying a Gaussian function, often employed to model image or signal degradation [8], [9].

Also, the Character Error Rate (CER) is obtained from the Optical Character Recognition (OCR) process. This is a core process in vehicle license plate recognition. OCR is another application of computer vision used to recognize characters in images and convert them into strings or text [10].

The programming languages used include Python, JavaScript, HTML, and CSS. This system is designed to modernize the vehicle license plate segmentation process, enabling automated scanning and information retrieval without manual input. This research is expected to provide insights into which method is more effective for vehicle license plate segmentation. As well as contribute to the development of better plate recognition systems in the future.

The novelty of this study lies in the direct performance comparison between a traditional edge-based algorithm (Canny) and a lightweight deep learning model (MobileNet) within the context of license plate segmentation under real-world conditions. While previous studies often focus solely on recognition accuracy, this research emphasizes segmentation quality using Character Error Rate (CER) as a performance indicator, which is rarely used in segmentation evaluation. Furthermore, the integration of MobileNet in a web-based implementation for real-time vehicle license plate segmentation demonstrates a practical approach that bridges deep learning models with accessible front-end technologies.

This research is expected to provide insights into which method is more effective for vehicle license plate segmentation, contributing to the development of more accurate and efficient automatic license plate recognition (ALPR) systems in dynamic and real-world environments.

2. RESEARCH METHODS

This research uses an experimental approach to evaluate the effectiveness of two segmentation methods in vehicle license plate detection: MobileNet architecture and the Canny algorithm. The steps of the research method undertaken are as follows:

2.1. Data Collection

Images of vehicle license plates were collected to build a representative database. A total of one hundred eighty (180) images were captured under varying lighting conditions and distances to ensure diversity in the dataset. These images represent various scenarios such as close-range, medium-range, and angled shots.

2.2. Preprocessing

The collected images were then processed to enhance quality before segmentation was performed [11]. The preprocessing pipeline included:

1. Conversion to grayscale to simplify data and reduce computational complexity.
2. Noise reduction using bilateral filters to preserve edges while smoothing other areas.
3. Blurring with Gaussian kernels (3×3 , 5×5 , 7×7) to minimize noise and prepare the image for effective edge detection and segmentation.
4. Normalization and resizing of all images to a uniform size of 320×320 pixels to be compatible with the input layer of MobileNet.

Vehicle license plate images undergo several processes before the license plate numbers are recognized using OCR. During the preprocessing stage, image segmentation of the plates will also be conducted.

The preprocessing steps begin by converting the image to grayscale. After that, denoising [12], the image is through the bilateral filter (blurring).

2.3. Vehicle License Plate Segmentation

MobileNet Architecture

This method uses a pre-trained convolutional neural network to detect features on vehicle plates.

1. A pre-trained MobileNet model (MobileNetV2) was fine-tuned using transfer learning techniques.
2. The dataset was split into 80% for training and 20% for testing (i.e., 144 training images, 36 testing images).
3. Data augmentation techniques such as rotation, zooming, and flipping were applied to enrich the training dataset and reduce overfitting.
4. The model was trained using the Adam optimizer, with categorical cross-entropy loss, over 50 epochs and a batch size of 16.
5. The training process was conducted using TensorFlow/Keras in Python.
6. The output of the model was a segmented license plate region used as input for OCR processing.

Canny Algorithm

This method applies edge detection to extract the plate area from the vehicle image.

1. The Canny method involved applying Gaussian blurring followed by edge detection using predefined thresholds (e.g., lower=50, upper=150).
2. The result was a binary edge map used to isolate and segment the license plate area before feeding it into the OCR system.

2.4. Data Analysis dan Validation

After the segmentation process, an analysis is conducted to determine which method is more effective for vehicle license plate segmentation. The comparison is based on the validation error rates obtained from each method. The accuracy of both methods is evaluated using Optical Character Recognition (OCR) techniques to read vehicle license plate numbers. OCR was applied using the Tesseract OCR engine. The output text was compared against the ground truth license plate numbers. The Character Error Rate (CER) is measured for each method by comparing the OCR results against the original data.

CER was chosen because it measures how closely the recognized text matches the original plate number, making it suitable for evaluating segmentation quality in OCR applications. A lower CER value indicates better performance.

2.5. Segmentation Results dan CER Validation

The segmentations from MobileNet and Canny were tested on the same dataset to compare the performance of each algorithm. CER is an indicator for evaluating the accuracy of OCR results; the lower the CER value, the better the performance of the OCR.

3. RESULT AND DISCUSSION

3.1 Data Collection

Images of vehicle plates were collected to build a representative database, as shown in Figure 1. The data was initially stored in the internal memory of an OPPO A12 camera and then transferred to the internal storage of an Asus X441MA PC.



Figure 1. Sampling

3.2 Preprocessing

Preprocessing is a crucial step in the image segmentation pipeline, as it significantly influences the performance of the segmentation algorithms that follow. In this study, two separate preprocessing workflows are implemented: one tailored for the Canny edge detection algorithm, and another optimized for the MobileNet deep learning model. Each approach has specific requirements in terms of input format and quality, thus necessitating different preprocessing strategies.

3.2.1 Canny Algorithm

The preprocessing steps for the Canny-based segmentation method are designed to enhance image clarity, suppress noise, and emphasize object boundaries, thereby improving the accuracy of edge detection. Since the Canny algorithm is highly sensitive to variations in pixel intensity and noise artifacts, preprocessing is applied before segmentation to ensure that only meaningful edges are preserved.

1) RGB to Gray

The first step in the Canny preprocessing pipeline involves converting the original color image (in RGB or BGR format) into grayscale. This conversion is essential because the Canny algorithm operates on single-channel images, where edge detection is performed based on intensity gradients rather than color information. In this study, the conversion is performed using the OpenCV-Python library, specifically the `cv2.cvtColor()` function with the `cv2.COLOR_BGR2GRAY` argument. This method computes a weighted sum of the red, green, and blue channels to produce a single grayscale image, where each pixel value represents intensity on a scale from 0 (black) to 255 (white).

As shown in Figure 2, the left image represents the original vehicle photograph in color, while the right image demonstrates the result of the grayscale conversion. This step simplifies the image structure and reduces computational complexity, preparing it for further noise reduction and edge detection processes. The grayscale image is preferred because it allows for more efficient processing and ensures that edge detection focuses solely on variations in brightness where edges are most likely to occur without being distracted by color information that may be irrelevant or misleading in this context.



Figure 2. RGB to Gray

2) Denoise

In the denoising process, unwanted noise in the image is reduced [13]. Noise refers to avoided variations in pixel intensity that can obscure details, diminish clarity, and affect image quality [14], [15], [16]. In the image below, the denoising was performed using a bilateral filter from the OpenCV-Python library. The left image shows the original vehicle image, while the right image displays the result after denoising.

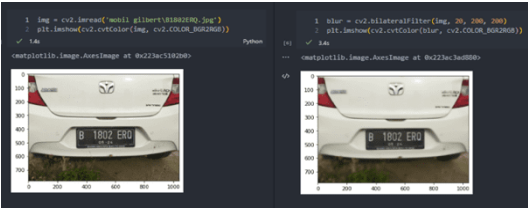


Figure 3. Denoise

3.2.1 MobileNet

Preprocessing is a critical preparatory step in the deep learning pipeline, particularly when deploying convolutional neural networks such as MobileNet for image segmentation tasks. The quality, consistency, and format of the input data directly affect the performance and generalization capabilities of the model. In this study, preprocessing for MobileNet segmentation involves resizing the input images to a standardized shape, which ensures compatibility with the architecture and facilitates efficient model inference.

1) Resize

The image resizing stage aims to change the dimensions of the image [17], [18]. The primary preprocessing step in the MobileNet workflow is resizing the input image to a fixed dimension of 320×320 pixels. This step is essential for two reasons:

- 3.4 Model Compatibility: MobileNet, like many convolutional neural networks, requires input images to have a consistent size. The network's architecture—including convolutional layers, pooling layers, and fully connected layers—is designed to process fixed-size inputs. Resizing ensures that all images conform to the expected input shape of the model.
- 3.5 Computational Efficiency: The choice of 320×320 pixels represents a balance between preserving sufficient detail in the license plate area and maintaining low computational cost. Larger images may contain more visual information, but they significantly increase the computational burden and memory requirements during training and inference. Conversely, images that are too small may result in loss of critical features, negatively impacting segmentation accuracy. The 320×320 resolution was selected experimentally to optimize both segmentation accuracy and processing speed.

The resizing process was implemented using the OpenCV-Python library (`cv2.resize()` function), which performs interpolation to scale the image while attempting to retain structural integrity. Importantly, aspect ratio distortion is minimized by applying padding where necessary, or cropping uniformly around the region of interest. As illustrated in Figure 4, the left image represents the original vehicle image, while the right image shows the result after resizing. This transformation standardizes the input data, ensuring that every image fed into the MobileNet model has consistent dimensions and pixel distribution characteristics.

By standardizing the input size, the MobileNet model is better able to focus on learning discriminative features relevant to license plate localization, such as edge patterns, text shapes, and spatial arrangement. This preprocessing step directly contributes to the model's ability to generalize well across various vehicle types, plate formats, and image conditions.

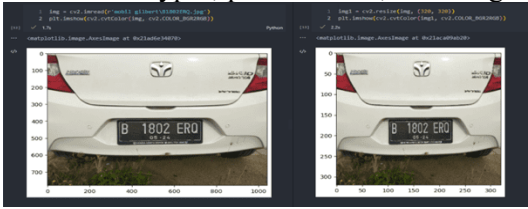


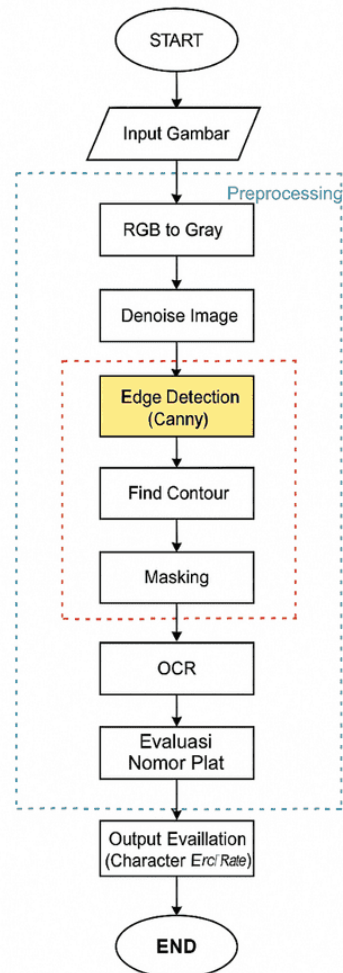
Figure 4. Resize

2) Splitting Data

From a total of 180 images, the dataset is divided into training data and testing data, with a ratio of 70% for training and 30% for testing. The split data will be used in the process of developing the MobileNet model.

3.3 Vehicle License Plate Segmentation

Canny Segmentation:



MobileNet Segmentation:

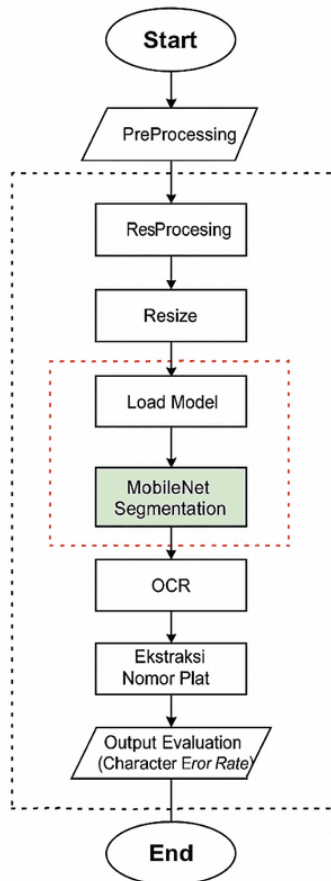


Figure 5. Segmentation

The segmentation process is initiated by using the Canny algorithm for edge detection, followed by the Find Contour and Masking processes. These two steps aim to identify rectangular patterns on the vehicle plates and separate them from other unnecessary parts of the image.

In MobileNet segmentation, the model will be loaded after the image is resized. The MobileNet model was previously created through a training process using the input dataset; once completed, the model was saved, as shown in Figure 6. This MobileNet model will perform segmentation on the images. After segmentation, whether using Canny or MobileNet, the next step will involve character recognition with OCR. Following this, a model training process is conducted, where the split data will be learned to produce a model capable of recognizing the objects to be detected in the images (Feature Learning Process).

```

6. Train the model

1 TRAINING_SCRIPT = os.path.join(pathlib.PYTHON_PATH, "research", "object_detection", "model_main_tf2.py")

2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Figure 6. Script for training a Model

3.4 Data Analysis dan Validation

3.4.1 Canny Segmentation

In the image segmentation process for vehicle plates using the Canny method, several steps are taken after preprocessing the images, including edge detection using the Canny method itself, followed by the Find Contour and Masking processes to locate the pixels of the vehicle license plate within the image and separate it from the rest of the picture.

1) Edge Detection

The edge detection process is performed using the Canny method from the OpenCV library, aiming to detect edges in the image. In Figure 7, the left image shows the original picture, which is then transformed to reveal the edge patterns on the vehicle, as seen in the right image.

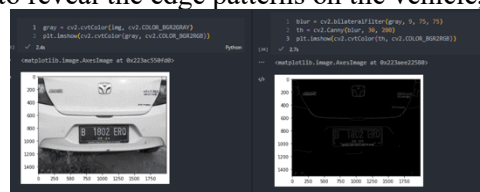


Figure 7. Edge Detection

The Canny algorithm is applied after noise reduction on the image using a noise reduction filter, specifically the `cv2.bilateralFilter`. Following this, the Canny algorithm performs edge detection through a process known as "Hysteresis Thresholding." The essence of Hysteresis Thresholding is to determine whether all detected edges in the image will be considered "real edges" or not [19]. In this context, two threshold values are required: `minVal` and `maxVal`, as shown in Figure 8.

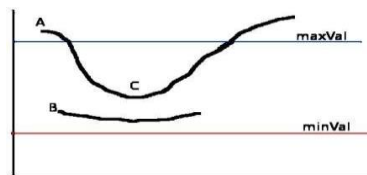
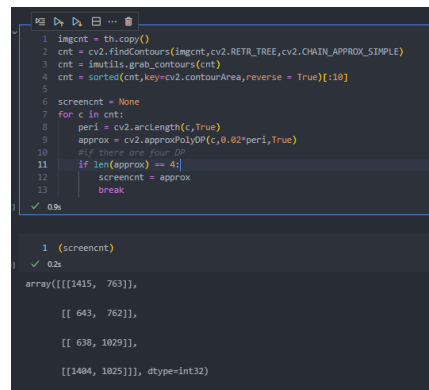


Figure 8. Hysteresis Thresholding

Edges with gradient intensity values more significant than the maximal value will be considered edges. Those below `minVal` will certainly not be classified as "edges" and will be discarded. The pixels that fall between these two thresholds will be classified as either "edges" or not based on their connectivity. If they are connected to a "definite edge" pixel, they will be regarded as part of the "edge." Otherwise, they will also be discarded.

2) Edge Detection

After the edge detection process on the image, the next step is the Find Contour process, which searches for the boundary edges of the vehicle plate. Typically identified in a rectangular shape. The outcome of the Find Contour process is four-pixel position points in (x, y) format from the vehicle image, as illustrated in Figure 9.



```

1 imgcnt = th.copy()
2 cnt = cv2.findContours(imgcnt,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
3 cnt = itertools.chain(*cnt)
4 cnt = sorted(cnt,key=cv2.contourArea,reverse = True)[:10]
5
6 screncnt = None
7 for c in cnt:
8     peri = cv2.arcLength(c,True)
9     approx = cv2.approxPolyDP(c,0.02*peri,True)
10     if len(approx) == 4:
11         screncnt = approx
12         break
13
14 ✓ 0.9s
15
16 1 (screncnt)
17 ✓ 0.2s
18 array([[[1415, 763]],
19        [[ 643, 762]],
20        [[ 638, 1829]],
21        [[1484, 1825]]], dtype=int32)

```

Figure 9. Find Contour

The contour searching process begins by retrieving all contours in the image using the `cv2.RETR_TREE` function. These contours are then sorted, and the ten largest contours, which represent closed contour areas such as rectangles and triangles, are selected.

Next, a loop is performed on the ten largest contours. During this iteration, the perimeter of each contour is calculated using the `cv2.arcLength` function, facilitating the contour shape simplification process with the `cv2.approxPolyDP` function, which simplifies the angles and points of a contour. After these two processes, it can be determined whether a contour has four corners, an indicator of the object to be segmented (the vehicle plate). If a contour meeting this criterion is found, the loop is terminated, and an output array containing the pixel positions (x, y) of the contour points or corners in the image (the location of the vehicle plate) is generated, as shown in Figure 9.

After identifying the boundary coordinates (contour) of the vehicle plate, the next step is to separate this image from the overall picture. This process is known as masking, where all pixels outside the identified contour are assigned a different pixel value, as seen in Figure 10. This is accomplished using the Numpy library function `np.zeros`, which assigns a pixel value of "0" to all pixels in the image except for the area to be separated, where a pixel value of "0" signifies the area that will be removed. After this process, the plate image can be effectively isolated from the entire image.



```

1 # Masking the image
2 mask = np.zeros(gray.shape,np.uint8)
3
4 new_image = cv2.drawContours(mask,[screncnt],0,255,-1)
5 new_image = cv2.bitwise_and(img,new_image)
6 plt.imshow(cv2.cvtColor(new_image, cv2.COLOR_BGR2RGB))
7
8 ✓ 0.5s
9
10 matplotlib.image.AxesImage at 8c223af3b6b0
11
12 ✓ 0.5s
13
14 matplotlib.image.AxesImage at 8c223af3f1fab0
15
16 ✓ 0.5s
17
18 matplotlib.image.AxesImage at 8c223af3f1fab0
19
20 ✓ 0.5s
21
22 matplotlib.image.AxesImage at 8c223af3f1fab0
23
24 ✓ 0.5s
25
26 matplotlib.image.AxesImage at 8c223af3f1fab0
27
28 ✓ 0.5s
29
30 matplotlib.image.AxesImage at 8c223af3f1fab0
31
32 ✓ 0.5s
33
34 matplotlib.image.AxesImage at 8c223af3f1fab0
35
36 ✓ 0.5s
37
38 matplotlib.image.AxesImage at 8c223af3f1fab0
39
40 ✓ 0.5s
41
42 matplotlib.image.AxesImage at 8c223af3f1fab0
43
44 ✓ 0.5s
45
46 matplotlib.image.AxesImage at 8c223af3f1fab0
47
48 ✓ 0.5s
49
50 matplotlib.image.AxesImage at 8c223af3f1fab0
51
52 ✓ 0.5s
53
54 matplotlib.image.AxesImage at 8c223af3f1fab0
55
56 ✓ 0.5s
57
58 matplotlib.image.AxesImage at 8c223af3f1fab0
59
60 ✓ 0.5s
61
62 matplotlib.image.AxesImage at 8c223af3f1fab0
63
64 ✓ 0.5s
65
66 matplotlib.image.AxesImage at 8c223af3f1fab0
67
68 ✓ 0.5s
69
70 matplotlib.image.AxesImage at 8c223af3f1fab0
71
72 ✓ 0.5s
73
74 matplotlib.image.AxesImage at 8c223af3f1fab0
75
76 ✓ 0.5s
77
78 matplotlib.image.AxesImage at 8c223af3f1fab0
79
80 ✓ 0.5s
81
82 matplotlib.image.AxesImage at 8c223af3f1fab0
83
84 ✓ 0.5s
85
86 matplotlib.image.AxesImage at 8c223af3f1fab0
87
88 ✓ 0.5s
89
90 matplotlib.image.AxesImage at 8c223af3f1fab0
91
92 ✓ 0.5s
93
94 matplotlib.image.AxesImage at 8c223af3f1fab0
95
96 ✓ 0.5s
97
98 matplotlib.image.AxesImage at 8c223af3f1fab0
99
100 ✓ 0.5s
101
102 matplotlib.image.AxesImage at 8c223af3f1fab0
103
104 ✓ 0.5s
105
106 matplotlib.image.AxesImage at 8c223af3f1fab0
107
108 ✓ 0.5s
109
110 matplotlib.image.AxesImage at 8c223af3f1fab0
111
112 ✓ 0.5s
113
114 matplotlib.image.AxesImage at 8c223af3f1fab0
115
116 ✓ 0.5s
117
118 matplotlib.image.AxesImage at 8c223af3f1fab0
119
120 ✓ 0.5s
121
122 matplotlib.image.AxesImage at 8c223af3f1fab0
123
124 ✓ 0.5s
125
126 matplotlib.image.AxesImage at 8c223af3f1fab0
127
128 ✓ 0.5s
129
130 matplotlib.image.AxesImage at 8c223af3f1fab0
131
132 ✓ 0.5s
133
134 matplotlib.image.AxesImage at 8c223af3f1fab0
135
136 ✓ 0.5s
137
138 matplotlib.image.AxesImage at 8c223af3f1fab0
139
140 ✓ 0.5s
141
142 matplotlib.image.AxesImage at 8c223af3f1fab0
143
144 ✓ 0.5s
145
146 matplotlib.image.AxesImage at 8c223af3f1fab0
147
148 ✓ 0.5s
149
150 matplotlib.image.AxesImage at 8c223af3f1fab0
151
152 ✓ 0.5s
153
154 matplotlib.image.AxesImage at 8c223af3f1fab0
155
156 ✓ 0.5s
157
158 matplotlib.image.AxesImage at 8c223af3f1fab0
159
160 ✓ 0.5s
161
162 matplotlib.image.AxesImage at 8c223af3f1fab0
163
164 ✓ 0.5s
165
166 matplotlib.image.AxesImage at 8c223af3f1fab0
167
168 ✓ 0.5s
169
170 matplotlib.image.AxesImage at 8c223af3f1fab0
171
172 ✓ 0.5s
173
174 matplotlib.image.AxesImage at 8c223af3f1fab0
175
176 ✓ 0.5s
177
178 matplotlib.image.AxesImage at 8c223af3f1fab0
179
180 ✓ 0.5s
181
182 matplotlib.image.AxesImage at 8c223af3f1fab0
183
184 ✓ 0.5s
185
186 matplotlib.image.AxesImage at 8c223af3f1fab0
187
188 ✓ 0.5s
189
190 matplotlib.image.AxesImage at 8c223af3f1fab0
191
192 ✓ 0.5s
193
194 matplotlib.image.AxesImage at 8c223af3f1fab0
195
196 ✓ 0.5s
197
198 matplotlib.image.AxesImage at 8c223af3f1fab0
199
200 ✓ 0.5s
201
202 matplotlib.image.AxesImage at 8c223af3f1fab0
203
204 ✓ 0.5s
205
206 matplotlib.image.AxesImage at 8c223af3f1fab0
207
208 ✓ 0.5s
209
210 matplotlib.image.AxesImage at 8c223af3f1fab0
211
212 ✓ 0.5s
213
214 matplotlib.image.AxesImage at 8c223af3f1fab0
215
216 ✓ 0.5s
217
218 matplotlib.image.AxesImage at 8c223af3f1fab0
219
220 ✓ 0.5s
221
222 matplotlib.image.AxesImage at 8c223af3f1fab0
223
224 ✓ 0.5s
225
226 matplotlib.image.AxesImage at 8c223af3f1fab0
227
228 ✓ 0.5s
229
230 matplotlib.image.AxesImage at 8c223af3f1fab0
231
232 ✓ 0.5s
233
234 matplotlib.image.AxesImage at 8c223af3f1fab0
235
236 ✓ 0.5s
237
238 matplotlib.image.AxesImage at 8c223af3f1fab0
239
240 ✓ 0.5s
241
242 matplotlib.image.AxesImage at 8c223af3f1fab0
243
244 ✓ 0.5s
245
246 matplotlib.image.AxesImage at 8c223af3f1fab0
247
248 ✓ 0.5s
249
250 matplotlib.image.AxesImage at 8c223af3f1fab0
251
252 ✓ 0.5s
253
254 matplotlib.image.AxesImage at 8c223af3f1fab0
255
256 ✓ 0.5s
257
258 matplotlib.image.AxesImage at 8c223af3f1fab0
259
260 ✓ 0.5s
261
262 matplotlib.image.AxesImage at 8c223af3f1fab0
263
264 ✓ 0.5s
265
266 matplotlib.image.AxesImage at 8c223af3f1fab0
267
268 ✓ 0.5s
269
270 matplotlib.image.AxesImage at 8c223af3f1fab0
271
272 ✓ 0.5s
273
274 matplotlib.image.AxesImage at 8c223af3f1fab0
275
276 ✓ 0.5s
277
278 matplotlib.image.AxesImage at 8c223af3f1fab0
279
280 ✓ 0.5s
281
282 matplotlib.image.AxesImage at 8c223af3f1fab0
283
284 ✓ 0.5s
285
286 matplotlib.image.AxesImage at 8c223af3f1fab0
287
288 ✓ 0.5s
289
290 matplotlib.image.AxesImage at 8c223af3f1fab0
291
292 ✓ 0.5s
293
294 matplotlib.image.AxesImage at 8c223af3f1fab0
295
296 ✓ 0.5s
297
298 matplotlib.image.AxesImage at 8c223af3f1fab0
299
300 ✓ 0.5s
301
302 matplotlib.image.AxesImage at 8c223af3f1fab0
303
304 ✓ 0.5s
305
306 matplotlib.image.AxesImage at 8c223af3f1fab0
307
308 ✓ 0.5s
309
310 matplotlib.image.AxesImage at 8c223af3f1fab0
311
312 ✓ 0.5s
313
314 matplotlib.image.AxesImage at 8c223af3f1fab0
315
316 ✓ 0.5s
317
318 matplotlib.image.AxesImage at 8c223af3f1fab0
319
320 ✓ 0.5s
321
322 matplotlib.image.AxesImage at 8c223af3f1fab0
323
324 ✓ 0.5s
325
326 matplotlib.image.AxesImage at 8c223af3f1fab0
327
328 ✓ 0.5s
329
330 matplotlib.image.AxesImage at 8c223af3f1fab0
331
332 ✓ 0.5s
333
334 matplotlib.image.AxesImage at 8c223af3f1fab0
335
336 ✓ 0.5s
337
338 matplotlib.image.AxesImage at 8c223af3f1fab0
339
340 ✓ 0.5s
341
342 matplotlib.image.AxesImage at 8c223af3f1fab0
343
344 ✓ 0.5s
345
346 matplotlib.image.AxesImage at 8c223af3f1fab0
347
348 ✓ 0.5s
349
350 matplotlib.image.AxesImage at 8c223af3f1fab0
351
352 ✓ 0.5s
353
354 matplotlib.image.AxesImage at 8c223af3f1fab0
355
356 ✓ 0.5s
357
358 matplotlib.image.AxesImage at 8c223af3f1fab0
359
360 ✓ 0.5s
361
362 matplotlib.image.AxesImage at 8c223af3f1fab0
363
364 ✓ 0.5s
365
366 matplotlib.image.AxesImage at 8c223af3f1fab0
367
368 ✓ 0.5s
369
370 matplotlib.image.AxesImage at 8c223af3f1fab0
371
372 ✓ 0.5s
373
374 matplotlib.image.AxesImage at 8c223af3f1fab0
375
376 ✓ 0.5s
377
378 matplotlib.image.AxesImage at 8c223af3f1fab0
379
380 ✓ 0.5s
381
382 matplotlib.image.AxesImage at 8c223af3f1fab0
383
384 ✓ 0.5s
385
386 matplotlib.image.AxesImage at 8c223af3f1fab0
387
388 ✓ 0.5s
389
390 matplotlib.image.AxesImage at 8c223af3f1fab0
391
392 ✓ 0.5s
393
394 matplotlib.image.AxesImage at 8c223af3f1fab0
395
396 ✓ 0.5s
397
398 matplotlib.image.AxesImage at 8c223af3f1fab0
399
400 ✓ 0.5s
401
402 matplotlib.image.AxesImage at 8c223af3f1fab0
403
404 ✓ 0.5s
405
406 matplotlib.image.AxesImage at 8c223af3f1fab0
407
408 ✓ 0.5s
409
410 matplotlib.image.AxesImage at 8c223af3f1fab0
411
412 ✓ 0.5s
413
414 matplotlib.image.AxesImage at 8c223af3f1fab0
415
416 ✓ 0.5s
417
418 matplotlib.image.AxesImage at 8c223af3f1fab0
419
420 ✓ 0.5s
421
422 matplotlib.image.AxesImage at 8c223af3f1fab0
423
424 ✓ 0.5s
425
426 matplotlib.image.AxesImage at 8c223af3f1fab0
427
428 ✓ 0.5s
429
430 matplotlib.image.AxesImage at 8c223af3f1fab0
431
432 ✓ 0.5s
433
434 matplotlib.image.AxesImage at 8c223af3f1fab0
435
436 ✓ 0.5s
437
438 matplotlib.image.AxesImage at 8c223af3f1fab0
439
440 ✓ 0.5s
441
442 matplotlib.image.AxesImage at 8c223af3f1fab0
443
444 ✓ 0.5s
445
446 matplotlib.image.AxesImage at 8c223af3f1fab0
447
448 ✓ 0.5s
449
450 matplotlib.image.AxesImage at 8c223af3f1fab0
451
452 ✓ 0.5s
453
454 matplotlib.image.AxesImage at 8c223af3f1fab0
455
456 ✓ 0.5s
457
458 matplotlib.image.AxesImage at 8c223af3f1fab0
459
460 ✓ 0.5s
461
462 matplotlib.image.AxesImage at 8c223af3f1fab0
463
464 ✓ 0.5s
465
466 matplotlib.image.AxesImage at 8c223af3f1fab0
467
468 ✓ 0.5s
469
470 matplotlib.image.AxesImage at 8c223af3f1fab0
471
472 ✓ 0.5s
473
474 matplotlib.image.AxesImage at 8c223af3f1fab0
475
476 ✓ 0.5s
477
478 matplotlib.image.AxesImage at 8c223af3f1fab0
479
480 ✓ 0.5s
481
482 matplotlib.image.AxesImage at 8c223af3f1fab0
483
484 ✓ 0.5s
485
486 matplotlib.image.AxesImage at 8c223af3f1fab0
487
488 ✓ 0.5s
489
490 matplotlib.image.AxesImage at 8c223af3f1fab0
491
492 ✓ 0.5s
493
494 matplotlib.image.AxesImage at 8c223af3f1fab0
495
496 ✓ 0.5s
497
498 matplotlib.image.AxesImage at 8c223af3f1fab0
499
500 ✓ 0.5s
501
502 matplotlib.image.AxesImage at 8c223af3f1fab0
503
504 ✓ 0.5s
505
506 matplotlib.image.AxesImage at 8c223af3f1fab0
507
508 ✓ 0.5s
509
510 matplotlib.image.AxesImage at 8c223af3f1fab0
511
512 ✓ 0.5s
513
514 matplotlib.image.AxesImage at 8c223af3f1fab0
515
516 ✓ 0.5s
517
518 matplotlib.image.AxesImage at 8c223af3f1fab0
519
520 ✓ 0.5s
521
522 matplotlib.image.AxesImage at 8c223af3f1fab0
523
524 ✓ 0.5s
525
526 matplotlib.image.AxesImage at 8c223af3f1fab0
527
528 ✓ 0.5s
529
530 matplotlib.image.AxesImage at 8c223af3f1fab0
531
532 ✓ 0.5s
533
534 matplotlib.image.AxesImage at 8c223af3f1fab0
535
536 ✓ 0.5s
537
538 matplotlib.image.AxesImage at 8c223af3f1fab0
539
540 ✓ 0.5s
541
542 matplotlib.image.AxesImage at 8c223af3f1fab0
543
544 ✓ 0.5s
545
546 matplotlib.image.AxesImage at 8c223af3f1fab0
547
548 ✓ 0.5s
549
550 matplotlib.image.AxesImage at 8c223af3f1fab0
551
552 ✓ 0.5s
553
554 matplotlib.image.AxesImage at 8c223af3f1fab0
555
556 ✓ 0.5s
557
558 matplotlib.image.AxesImage at 8c223af3f1fab0
559
560 ✓ 0.5s
561
562 matplotlib.image.AxesImage at 8c223af3f1fab0
563
564 ✓ 0.5s
565
566 matplotlib.image.AxesImage at 8c223af3f1fab0
567
568 ✓ 0.5s
569
570 matplotlib.image.AxesImage at 8c223af3f1fab0
571
572 ✓ 0.5s
573
574 matplotlib.image.AxesImage at 8c223af3f1fab0
575
576 ✓ 0.5s
577
578 matplotlib.image.AxesImage at 8c223af3f1fab0
579
580 ✓ 0.5s
581
582 matplotlib.image.AxesImage at 8c223af3f1fab0
583
584 ✓ 0.5s
585
586 matplotlib.image.AxesImage at 8c223af3f1fab0
587
588 ✓ 0.5s
589
590 matplotlib.image.AxesImage at 8c223af3f1fab0
591
592 ✓ 0.5s
593
594 matplotlib.image.AxesImage at 8c223af3f1fab0
595
596 ✓ 0.5s
597
598 matplotlib.image.AxesImage at 8c223af3f1fab0
599
600 ✓ 0.5s
601
602 matplotlib.image.AxesImage at 8c223af3f1fab0
603
604 ✓ 0.5s
605
606 matplotlib.image.AxesImage at 8c223af3f1fab0
607
608 ✓ 0.5s
609
610 matplotlib.image.AxesImage at 8c223af3f1fab0
611
612 ✓ 0.5s
613
614 matplotlib.image.AxesImage at 8c223af3f1fab0
615
616 ✓ 0.5s
617
618 matplotlib.image.AxesImage at 8c223af3f1fab0
619
620 ✓ 0.5s
621
622 matplotlib.image.AxesImage at 8c223af3f1fab0
623
624 ✓ 0.5s
625
626 matplotlib.image.AxesImage at 8c223af3f1fab0
627
628 ✓ 0.5s
629
630 matplotlib.image.AxesImage at 8c223af3f1fab0
631
632 ✓ 0.5s
633
634 matplotlib.image.AxesImage at 8c223af3f1fab0
635
636 ✓ 0.5s
637
638 matplotlib.image.AxesImage at 8c223af3f1fab0
639
640 ✓ 0.5s
641
642 matplotlib.image.AxesImage at 8c223af3f1fab0
643
644 ✓ 0.5s
645
646 matplotlib.image.AxesImage at 8c223af3f1fab0
647
648 ✓ 0.5s
649
650 matplotlib.image.AxesImage at 8c223af3f1fab0
651
652 ✓ 0.5s
653
654 matplotlib.image.AxesImage at 8c223af3f1fab0
655
656 ✓ 0.5s
657
658 matplotlib.image.AxesImage at 8c223af3f1fab0
659
660 ✓ 0.5s
661
662 matplotlib.image.AxesImage at 8c223af3f1fab0
663
664 ✓ 0.5s
665
666 matplotlib.image.AxesImage at 8c223af3f1fab0
667
668 ✓ 0.5s
669
670 matplotlib.image.AxesImage at 8c223af3f1fab0
671
672 ✓ 0.5s
673
674 matplotlib.image.AxesImage at 8c223af3f1fab0
675
676 ✓ 0.5s
677
678 matplotlib.image.AxesImage at 8c223af3f1fab0
679
680 ✓ 0.5s
681
682 matplotlib.image.AxesImage at 8c223af3f1fab0
683
684 ✓ 0.5s
685
686 matplotlib.image.AxesImage at 8c223af3f1fab0
687
688 ✓ 0.5s
689
690 matplotlib.image.AxesImage at 8c223af3f1fab0
691
692 ✓ 0.5s
693
694 matplotlib.image.AxesImage at 8c223af3f1fab0
695
696 ✓ 0.5s
697
698 matplotlib.image.AxesImage at 8c223af3f1fab0
699
700 ✓ 0.5s
701
702 matplotlib.image.AxesImage at 8c223af3f1fab0
703
704 ✓ 0.5s
705
706 matplotlib.image.AxesImage at 8c223af3f1fab0
707
708 ✓ 0.5s
709
710 matplotlib.image.AxesImage at 8c223af3f1fab0
711
712 ✓ 0.5s
713
714 matplotlib.image.AxesImage at 8c223af3f1fab0
715
716 ✓ 0.5s
717
718 matplotlib.image.AxesImage at 8c223af3f1fab0
719
720 ✓ 0.5s
721
722 matplotlib.image.AxesImage at 8c223af3f1fab0
723
724 ✓ 0.5s
725
726 matplotlib.image.AxesImage at 8c223af3f1fab0
727
728 ✓ 0.5s
729
730 matplotlib.image.AxesImage at 8c223af3f1fab0
731
732 ✓ 0.5s
733
734 matplotlib.image.AxesImage at 8c223af3f1fab0
735
736 ✓ 0.5s
737
738 matplotlib.image.AxesImage at 8c223af3f1fab0
739
740 ✓ 0.5s
741
742 matplotlib.image.AxesImage at 8c223af3f1fab0
743
744 ✓ 0.5s
745
746 matplotlib.image.AxesImage at 8c223af3f1fab0
747
748 ✓ 0.5s
749
750 matplotlib.image.AxesImage at 8c223af3f1fab0
751
752 ✓ 0.5s
753
754 matplotlib.image.AxesImage at 8c223af3f1fab0
755
756 ✓ 0.5s
757
758 matplotlib.image.AxesImage at 8c223af3f1fab0
759
760 ✓ 0.5s
761
762 matplotlib.image.AxesImage at 8c223af3f1fab0
763
764 ✓ 0.5s
765
766 matplotlib.image.AxesImage at 8c223af3f1fab0
767
768 ✓ 0.5s
769
770 matplotlib.image.AxesImage at 8c223af3f1fab0
771
772 ✓ 0.5s
773
774 matplotlib.image.AxesImage at 8c223af3f1fab0
775
776 ✓ 0.5s
777
778 matplotlib.image.AxesImage at 8c223af3f1fab0
779
780 ✓ 0.5s
781
782 matplotlib.image.AxesImage at 8c223af3f1fab0
783
784 ✓ 0.5s
785
786 matplotlib.image.AxesImage at 8c223af3f1fab0
787
788 ✓ 0.5s
789
790 matplotlib.image.AxesImage at 8c223af3f1fab0
791
792 ✓ 0.5s
793
794 matplotlib.image.AxesImage at 8c223af3f1fab0
795
796 ✓ 0.5s
797
798 matplotlib.image.AxesImage at 8c223af3f1fab0
799
800 ✓ 0.5s
801
802 matplotlib.image.AxesImage at 8c223af3f1fab0
803
804 ✓ 0.5s
805
806 matplotlib.image.AxesImage at 8c223af3f1fab0
807
808 ✓ 0.5s
809
810 matplotlib.image.AxesImage at 8c223af3f1fab0
811
812 ✓ 0.5s
813
814 matplotlib.image.AxesImage at 8c223af3f1fab0
815
816 ✓ 0.5s
817
818 matplotlib.image.AxesImage at 8c223af3f1fab0
819
820 ✓ 0.5s
821
822 matplotlib.image.AxesImage at 8c223af3f1fab0
823
824 ✓ 0.5s
825
826 matplotlib.image.AxesImage at 8c223af3f1fab0
827
828 ✓ 0.5s
829
830 matplotlib.image.AxesImage at 8c223af3f1fab0
831
832 ✓ 0.5s
833
834 matplotlib.image.AxesImage at 8c223af3f1fab0
835
836 ✓ 0.5s
837
838 matplotlib.image.AxesImage at 8c223af3f1fab0
839
840 ✓ 0.5s
841
842 matplotlib.image.AxesImage at 8c223af3f1fab0
843
844 ✓ 0.5s
845
846 matplotlib.image.AxesImage at 8c223af3f1fab0
847
848 ✓ 0.5s
849
850 matplotlib.image.AxesImage at 8c223af3f1fab0
851
852 ✓ 0.5s
853
854 matplotlib.image.AxesImage at 8c223af3f1fab0
855
856 ✓ 0.5s
857
858 matplotlib.image.AxesImage at 8c223af3f1fab0
859
860 ✓ 0.5s
861
862 matplotlib.image.AxesImage at 8c223af3f1fab0
863
864 ✓ 0.5s
865
866 matplotlib.image.AxesImage at 8c223af3f1fab0
867
868 ✓ 0.5s
869
870 matplotlib.image.AxesImage at 8c223af3f1fab0
871
872 ✓ 0.5s
873
874 matplotlib.image.AxesImage at 8c223af3f1fab0
875
876 ✓ 0.5s
877
878 matplotlib.image.AxesImage at 8c223af3f1fab0
879
880 ✓ 0.5s
881
882 matplotlib.image.AxesImage at 8c223af3f1fab0
883
884 ✓ 0.5s
885
886 matplotlib.image.AxesImage at 8c223af3f1fab0
887
888 ✓ 0.5s
889
890 matplotlib.image.AxesImage at 8c223af3f1fab0
891
892 ✓ 0.5s
893
894 matplotlib.image.AxesImage at 8c223af3f1fab0
895
896 ✓ 0.5s
897
898 matplotlib.image.AxesImage at 8c223af3f1fab0
899
900 ✓ 0.5s
901
902 matplotlib.image.AxesImage at 8c223af3f1fab0
903
904 ✓ 0.5s
905
906 matplotlib.image.AxesImage at 8c223af3f1fab0
907
908 ✓ 0.5s
909
910 matplotlib.image.AxesImage at 8c223af3f1fab0
911
912 ✓ 0.5s
913
914 matplotlib.image.AxesImage at 8c223af3f1fab0
915
916 ✓ 0.5s
917
918 matplotlib.image.AxesImage at 8c223af3f1fab0
919
920 ✓ 0.5s
921
922 matplotlib.image.AxesImage at 8c223af3f1fab0
923
924 ✓ 0.5s
925
926 matplotlib.image.AxesImage at 8c223af3f1fab0
927
928 ✓ 0.5s
929
930 matplotlib.image.AxesImage at 8c223af3f1fab0
931
932 ✓ 0.5s
933
934 matplotlib.image.AxesImage at 8c223af3f1fab0
935
936 ✓ 0.5s
937
938 matplotlib.image.AxesImage at 8c223af3f1fab0
939
940 ✓ 0.5s
941
942 matplotlib.image.AxesImage at 8c223af3f1fab0
943
944 ✓ 0.5s
945
946 matplotlib.image.AxesImage at 8c223af3f1fab0
947
948 ✓ 0.5s
949
950 matplotlib.image.AxesImage at 8c223af3f1fab0
951
952 ✓ 0.5s
953
954 matplotlib.image.AxesImage at 8c223af3f1fab0
955
956 ✓ 0.5s
957
958 matplotlib.image.AxesImage at 8c223af3f1fab0
959
960 ✓ 0.5s
961
962 matplotlib.image.AxesImage at 8c223af3f1fab0
963
964 ✓ 0.5s
965
966 matplotlib.image.AxesImage at 8c223af3f1fab0
967
968 ✓ 0.5s
969
970 matplotlib.image.AxesImage at 8c223af3f1fab0
971
972 ✓ 0.5s
973
974 matplotlib.image.AxesImage at 8c223af3f1fab0
975
976 ✓ 0.5s
977
978 matplotlib.image.AxesImage at 8c223af3f1fab0
979
980 ✓ 0.5s
981
982 matplotlib.image.AxesImage at 8c223af3f1fab0
983
984 ✓ 0.5s
985
986 matplotlib.image.AxesImage at 8c223af3f1fab0
987
988 ✓ 0.5s
989
990 matplotlib.image.AxesImage at 8c223af3f1fab0
991
992 ✓ 0.5s
993
994 matplotlib.image.AxesImage at 8c223af3f1fab0
995
996 ✓ 0.5s
997
998 matplotlib.image.AxesImage at 8c223af3f1fab0
999
1000 ✓ 0.5s
1001
1002 matplotlib.image.AxesImage at 8c223af3f1fab0
1003
1004 ✓ 0.5s
1005
1006 matplotlib.image.AxesImage at 8c223af3f1fab0
1007
1008 ✓ 0.5s
1009
1010 matplotlib.image.AxesImage at 8c223af3f1fab0
1011
1012 ✓ 0.5s
1013
1014 matplotlib.image.AxesImage at 8c223af3f1fab0
1015
1016 ✓ 0.5s
1017
1018 matplotlib.image.AxesImage at 8c223af3f1fab0
1019
1020 ✓ 0.5s
1021
1022 matplotlib.image.AxesImage at 8c223af3f1fab0
1023
1024 ✓ 0.5s
1025
1026 matplotlib.image.AxesImage at 8c223af3f1fab0
1027
1028 ✓ 0.5s
1029
1030 matplotlib.image.AxesImage at 8c223af3f1fab0
1031
1032 ✓ 0.5s
1033
1034 matplotlib.image.AxesImage at 8c223af3f1fab0
1035
1036 ✓ 0.5s
1037
1038 matplotlib.image.AxesImage at 8c223af3f1fab0
1039
1040 ✓ 0.5s
1041
1042 matplotlib.image.AxesImage at 8c223af3f1fab0
1043
1044 ✓ 0.5s
1045
1046 matplotlib.image.AxesImage at 8c223af3f1fab0
1047
1048 ✓ 0.5s
1049
1050 matplotlib.image.AxesImage at 8c223af3f1fab0
1051
1052 ✓ 0.5s
1053
1054 matplotlib.image.AxesImage at 8c223af3f1fab0
1055
1056 ✓ 0.5s
1057
1058 matplotlib.image.AxesImage at 8c223af3f1fab0
1059
1060 ✓ 0.5s
1061
1062 matplotlib.image.AxesImage at 8c223af3f1fab0
1063
1064 ✓ 0.5s
1065
1066 matplotlib.image.AxesImage at 8c223af3f1fab0
1067
1068 ✓ 0.5s
1069
1070 matplotlib.image.AxesImage at 8c223af3f1fab0
1071
1072 ✓ 0.5s
1073
1074 matplotlib.image.AxesImage at 8c223af3f1fab0
1075
1076 ✓ 0.5s
1077
1078 matplotlib.image.AxesImage at 8c223af3f1fab0
1079
1080 ✓ 0.5s
1081
1082 matplotlib.image.AxesImage at 8c223af3f1fab0
1083
1084 ✓ 0.5s
1085
1086 matplotlib.image.AxesImage at 8c223af3f1fab0
1087
1088 ✓ 0.5s
1089
1090 matplotlib.image.AxesImage at 8c223af3f1fab0
1091
1092 ✓ 0.5s
1093
1094 matplotlib.image.AxesImage at 8c223af3f1fab0
1095
1096 ✓ 0.5s
1097
1098 matplotlib.image.AxesImage at 8c223af3f1fab0
1099
1100 ✓ 0.5s
1101
1102 matplotlib.image.AxesImage at 8c223af3f1fab0
1103
1104 ✓ 0.5s
1105
1106 matplotlib.image.AxesImage at 8c223af3f1fab0
1107
1108 ✓ 0.5s
1109
1110 matplotlib.image.AxesImage at 8c223af3f1fab0
1111
1112 ✓ 0.5s
1113
1114 matplotlib.image.AxesImage at 8c223af3f1fab0
1115
1116 ✓ 0.5s
1117
1118 matplotlib.image.AxesImage at 8c223af3f1fab0
1119
1120 ✓ 0.5s
1121
1122 matplotlib.image.AxesImage at 8c223af3f1fab0
1123
1124 ✓ 0.5s
1125
1126 matplotlib.image.AxesImage at 8c223af3f1fab0
1127
1128 ✓ 0.5s
1129
1130 matplotlib.image.AxesImage at 8c223af3f1fab0
1131
1132 ✓ 0.5s
1133
1134 matplotlib.image.Axes
```

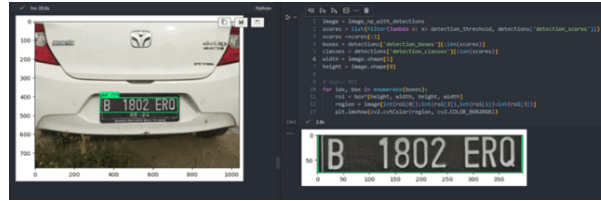



Figure 11. MobileNet Segmentation

1) Optical Character Recognition (OCR)

OCR (Optical Character Recognition) is the technique of recognizing text in images or digital documents and converting it into editable text [20]. For this, we used Google Tesseract, an open-source library developed by Google [21] that enables text recognition and extraction from the images.

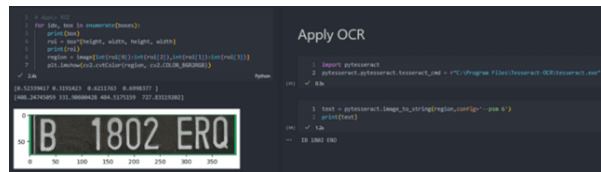


Figure 12. OCR Results of MobileNet Segmentation

Figure 12 presents the segmentation results using MobileNet on the left, followed by the OCR process with Google Tesseract. The final output, displayed on the right, identifies the vehicle license plate number.



Figure 13. OCR Results of Canny Algorithm Segmentation

On the left side of Figure 13, we see the segmentation results from the Canny algorithm, followed by the OCR process with Google Tesseract. The final output identifies the vehicle license plate number, shown on the right.

2) CER Validation

After obtaining the vehicle license plate number through OCR, the next step is to validate the accuracy of the result using CER (Character Error Rate). CER quantifies the error rate by comparing two text strings [22]: the original license plate and the OCR result.

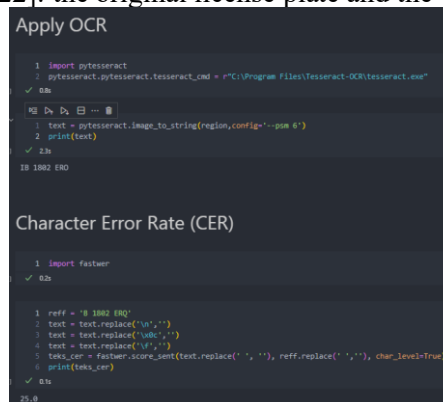


Figure 14. CER of MobileNet Segmentation

Figure 14 displays the OCR results of the vehicle license plate using MobileNet (IB 1802 ERO). Subsequently, the error was calculated using the Character Error Rate (CER) to compare:

OCRResult = IB 1802 ERO

Original Plate = B 1802 ERQ (reference variable)

The CER validation resulted in an error rate of 25%, calculated using the following formula:

$$CER = \frac{S+D+I}{N} \quad (1)$$

Where:

S = Number of Substitutions

D = Number of Deletions

I = Number of Insertions

N = Total number of characters in the original text

CER Calculation

In this case:

OCR result: IB 1802 ERO

Reference: B 1802 ERQ

Comparing both strings:

1 substitution: "I" instead of the missing character before "B"

1 insertion: Extra "O" at the end instead of "Q"

So:

S=1

D=0

I=1

N=8 (Total characters in the reference text "B 1802 ERQ")

Applying the values to Equation (1):

$$CER = \frac{1 + 0 + 1}{8}$$

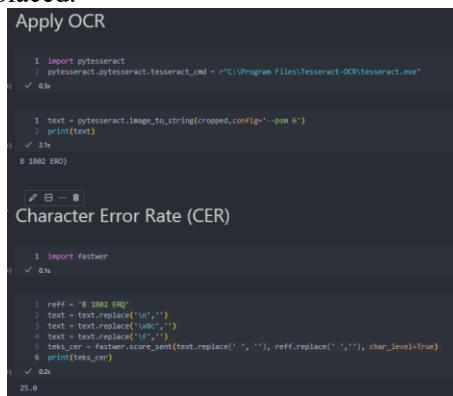
$$CER = \frac{2}{8}$$

$$CER = 0,25$$

$$CER = 25\%$$

Conclusion:

The Character Error Rate (CER) for the OCR output is 25%, indicating that 2 out of 8 characters were either incorrect or misplaced.



```

Apply OCR

1 import pytesseract
2 pytesseract.pytesseract.tesseract_cmd = r"C:\Program Files\Tesseract-OCR\tesseract.exe"
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
237
```

3.5 OCR Results and CER Validation

Table 1 presents the OCR results and error validation using CER, based on vehicle license plate segmentation performed by both the MobileNet and Canny algorithms. This table illustrates each algorithm's performance in recognizing characters on vehicle plates.

OCR results can be influenced by various factors, such as image quality, lighting conditions, font type on the vehicle plate, and background complexity. Consequently, analyzing the results and comparing the performance of both algorithms is vital for selecting the most appropriate OCR algorithm for specific applications.

Table 1. Segmentation Results and CER Validation

Number	Original License Plate	OCR MobileNet	OCR Canny	CER MobileNet	CER Canny
1	B 1802 ERQ	IB 1802 ERO	[B 1802 ERO	25%	25%
2	BE 1760 YQ	BE 1760 YQ	Undetected	0%	100%
3	BE 1876 ET	BE 1876 EI	Undetected	12,5%	100%
4	D 1358 AHG	ID 1358 AHG	Undetected	12.5%	100%
5	DD 1137 KQ	DD 1137 KQ	DU 1137 KO	0%	25%
6	DD 1137 LP	DD 1137 LP	Undetected	0%	100%
7	DD 1223 LB	0D 1223" LB	DD 1223 LB	25%	0%
8	DD 1949 KW	UU 19g49 KW)	OD 1949 KW	50%	12.5%
9	DD 1951 MH	'DD 1957 Mb	Undetected	37.5%	100%
10	L 1696 RR	L 1696 RR	L 1696 RR	0%	0%
11	DD 1552 XAI	DD 1752 YAI	Undetected	33.3%	100%
12	DD 1571 SQ	nn 1571 SQI	DD 1571 SQ	37.5%	0%
13	DD 1691 XAB	ND 1691 XAB	DD 1691 XAB	11,1%	0%
Total				18.8%	50.96%

The observed performance gap between the Canny algorithm and the MobileNet model can be attributed to fundamental differences in their underlying segmentation strategies. Canny is a traditional edge detection method that relies solely on pixel intensity gradients to identify object boundaries. Although this approach is computationally efficient and relatively straightforward, it is highly sensitive to image noise, lighting variations, and geometric distortions common challenges in real-world scenarios. Even with preprocessing techniques such as denoising and Gaussian blurring, the Canny algorithm often struggles to accurately isolate license plate regions when faced with complex backgrounds or low-quality input images. Its effectiveness largely depends on carefully tuned threshold parameters and the clarity of the plate's contour. As a result, the segmentation output may be incomplete or imprecise, adversely affecting subsequent OCR accuracy and increasing the Character Error Rate (CER).

In contrast, MobileNet adopts a deep learning-based approach that leverages convolutional neural networks to automatically extract and learn relevant features from labeled training data. Unlike Canny, it does not depend on fixed rules or manually defined parameters. This enables MobileNet to generalize effectively across a wide range of input conditions, including blurred images, inconsistent lighting, and oblique camera angles. Its architecture, based on depthwise separable convolutions, strikes a balance between computational efficiency and representational capacity. Furthermore, the model produces a consistent bounding box around the license plate area, ensuring that only the relevant region is passed to the OCR engine, thereby reducing noise interference.

Another critical advantage of MobileNet is its capacity for high-level feature learning, which allows the model to recognize abstract patterns such as character alignment, spacing, and structural consistency typical of license plates. These capabilities are beyond the scope of classical algorithms like Canny. As a result, even under suboptimal conditions, MobileNet is more likely to yield usable segmentation results that support more accurate character recognition, as reflected by its lower CER values in this study.

In summary, MobileNet's superior performance is attributed to its data-driven learning mechanism, robustness to input variability, and consistent localization of the region of interest.

These findings reinforce the suitability of deep learning-based segmentation techniques for real-world automatic license plate recognition systems, where image inconsistencies are unavoidable.

4. CONCLUSION

The results of the OCR and CER validation clearly show that MobileNet segmentation performs significantly better than traditional methods. With a Character Error Rate (CER) of just 18.8%, compared to 50.96% for the Canny method, MobileNet proves to be far more effective in accurately segmenting vehicle license plates. This level of accuracy is crucial for improving the performance and reliability of automated license plate recognition systems, particularly in real-world applications like smart parking and traffic monitoring. Compared to other popular convolutional neural network (CNN) architectures such as LeNet, AlexNet, VGG, GoogLeNet, and ResNet, MobileNet stands out by offering a strong balance between accuracy and efficiency. This makes it especially suitable for use in resource-limited environments like embedded systems or mobile devices, where computing power is limited. While deeper models like VGG and ResNet can deliver higher accuracy in some contexts, they often require much greater computational resources, which may not be practical for real-time or lightweight systems. Looking ahead, further research can focus on integrating MobileNet into real-time processing systems and evaluating its performance under varied or large-scale conditions. Exploring hybrid models—such as those that combine multiple CNNs or incorporate attention mechanisms could help boost performance even further. Additionally, investigating more recent lightweight models like EfficientNet or architectures based on transformers may open new possibilities for enhancing the speed and accuracy of automated vehicle recognition technologies.

REFERENCES

- [1] M. Idris et al., *Pengolahan Citra: Teori dan Implementasi*. Yayasan Kita Menulis, 2023.
- [2] S. Saifullah, D. B. Prasetyo, R. Dreżewski, and F. A. Dwiyanto, “Palm oil maturity classification using K-nearest neighbors based on RGB and L* a* b color extraction,” *Procedia Comput Sci*, vol. 225, pp. 3011–3020, 2023.
- [3] H. Hendriyana and Y. H. Maulana, “Identification of types of wood using convolutional neural network with MobileNet architecture,” *Jurnal RESTI (Rekayasa Sistem Dan Teknologi Informasi)*, vol. 4, no. 1, pp. 70–76, 2020.
- [4] J. S. T. Purni and R. Vedhapriyavadhana, “EOSA-Net: A deep learning framework for enhanced multi-class skin cancer classification using optimized convolutional neural networks,” *Journal of King Saud University-Computer and Information Sciences*, vol. 36, no. 3, p. 102007, 2024.
- [5] D. Zhang, G. Sun, Z. Yang, and J. Yu, “A high-density gamma white spots-Gaussian mixture noise removal method for neutron images denoising based on Swin Transformer UNet and Monte Carlo calculation,” *Nuclear Engineering and Technology*, vol. 56, no. 2, pp. 715–727, 2024.
- [6] H. Huang, Y. Wang, G. Yuan, and X. Li, “A Gaussian Noise-Based Algorithm for Enhancing Backdoor Attacks,” *Computers, Materials & Continua*, vol. 80, no. 1, 2024.
- [7] S. Boruah, A. Dehloo, P. Gupta, M. R. Prusty, and A. Balasundaram, “Gaussian blur masked resnet2. 0 architecture for diabetic retinopathy detection,” *Computers, Materials and Continua*, vol. 75, no. 1, pp. 927–942, 2023.
- [8] J. S. T. Purni and R. Vedhapriyavadhana, “EOSA-Net: A deep learning framework for enhanced multi-class skin cancer classification using optimized convolutional neural networks,” *Journal of King Saud University-Computer and Information Sciences*, vol. 36, no. 3, p. 102007, 2024.
- [9] R. Muthana and A. N. Alshareefi, “Techniques in de-blurring image,” in *Journal of physics: conference series*, IOP Publishing, 2020, p. 012115.

-
- [10] P. Phoenix, R. Sudaryono, and D. Suhartono, "Classifying promotion images using optical character recognition and Naïve Bayes classifier," *Procedia Comput Sci*, vol. 179, pp. 498–506, 2021.
 - [11] N. Z. Munantri, H. Sofyan, and M. Y. Florestiyanto, "Aplikasi Pengolahan Citra Digital Untuk Identifikasi Umur Pohon," *Telematika: Jurnal Informatika dan Teknologi Informasi*, vol. 16, no. 2, pp. 97–104, 2020.
 - [12] M. Idris et al., *Pengolahan Citra: Teori dan Implementasi*. Yayasan Kita Menulis, 2023.
 - [13] A. Y. Rahman, "Noise Reduction in RTL-SDR using Least Mean Square and Recursive Least Square," *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*, vol. 4, no. 2, pp. 286–295, 2020.
 - [14] F. Sugandi, "IMPLEMENTASI METODE GAUSSIAN FILTERING DALAM MENGURANGI NOISE PADA PENGOLAHAN CITRA DIGITAL," *International Research on Big-Data and Computer Technology: I-Robot*, vol. 7, no. 2, pp. 21–26, 2023.
 - [15] B. Goyal, A. Dogra, S. Agrawal, B. S. Sohi, and A. Sharma, "Image denoising review: From classical to state-of-the-art approaches," *Information fusion*, vol. 55, pp. 220–244, 2020.
 - [16] D. F. Ramadan, M. D. Sulistiyo, and A. F. Ihsan, "Noisy Image Filtering Methods for an Improved Sundanese Script Handwriting Classification," in *2023 International Conference on Data Science and Its Applications (ICoDSA)*, IEEE, 2023, pp. 454–459.
 - [17] P. A. Buana, A. N. Putri, and S. Adinugroho, "Optimalisasi Penggunaan Image Stitching dan Seam Carving dalam Pengembangan Tur Virtual Responsif," *Building of Informatics, Technology and Science (BITS)*, vol. 6, no. 1, pp. 520–532, 2024.
 - [18] O. V. Putra, A. Musthafa, M. Nur, and M. Rido, "Classification of Calligraphy Writing Types Using Convolutional Neural Network Method (CNN)," *Procedia of Engineering and Life Science*, vol. 2, 2021.
 - [19] R. D. C. Utomo, B. Kanata, and A. Zainuddin, "Analisis Pergeseran Garis Pantai Di Wilayah Pesisir Kabupaten Lombok Utara Dengan Metode Weighted Normalized Difference Water Index (Wndwi) Dan Deteksi Tepi Canny Pada Citra Landsat 8," *DIELEKTRIKA*, vol. 9, no. 2, pp. 130–140, 2022.
 - [20] M. D. U. Okta, S. Aulia, and B. Burhanuddin, "Pengenal Pola Berbasis OCR untuk Pengambilan Data Bursa Saham," *Jurnal Rekayasa ElektriKA*, vol. 17, no. 2, 2021.
 - [21] K. A. Nugraha, "Penerapan Optical Character Recognition untuk Pengenal Variasi Teks pada Media Presentasi Pembelajaran," *Jurnal Buana Informatika*, vol. 15, no. 01, pp. 69–78, 2024.
 - [22] T. Gelar and A. Nanda, "Eksplorasi Pengembangan Korpus Pembicaraan Spontan pada Video Instruksional Pertanian Perkotaan," *Jurnal of Software Engineering Information and Communication Technology*, vol. 3, no. 1, pp. 111–120, 2022.
-