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

Suryani*[1], Husain[2], Faizal[3]
[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**, [2]husain@undipa.ac.id, [3]F41241@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.

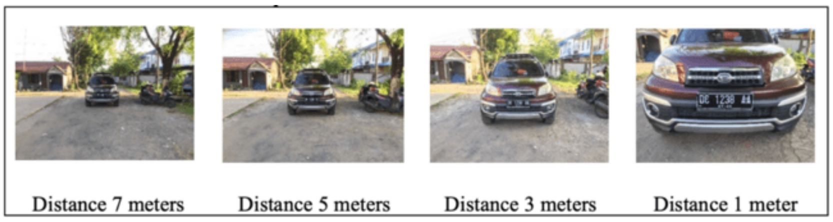


Figure 1. Sampling

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.

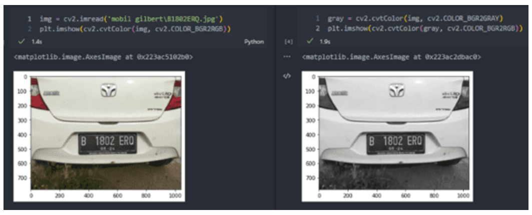
3.2.1 Canny Algorithm

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.



```
img = cv2.imread('mobil gilbert\B1802ERQ.jpg')
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x223ac5102b0>
```

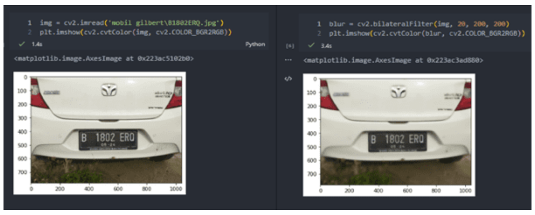
```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
plt.imshow(cv2.cvtColor(gray, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x223ac2dbac0>
```

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.



```
img = cv2.imread('mobil gilbert\B1802ERQ.jpg')
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x223ac5102b0>
```

```
blur = cv2.bilateralFilter(img, 20, 200, 200)
plt.imshow(cv2.cvtColor(blur, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x223ac3ad880>
```

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

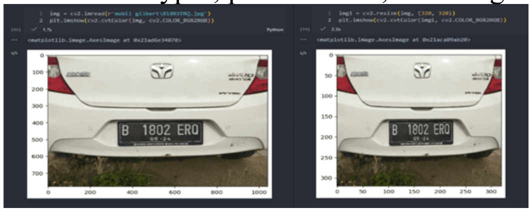
The image resizing stage aims to change the dimensions of the image [17], [18]. The primary preprocessing step in the MobileNet workflow is resizing the input image to a fixed dimension of 320 × 320 pixels. This step is essential for two reasons:

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.



```
img = cv2.imread(r'mobil gilbert\B1802ERQ.jpg')
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x21ad6e34070>
```

```
img1 = cv2.resize(img, (320, 320))
plt.imshow(cv2.cvtColor(img1, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x21aca09ab20>
```

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

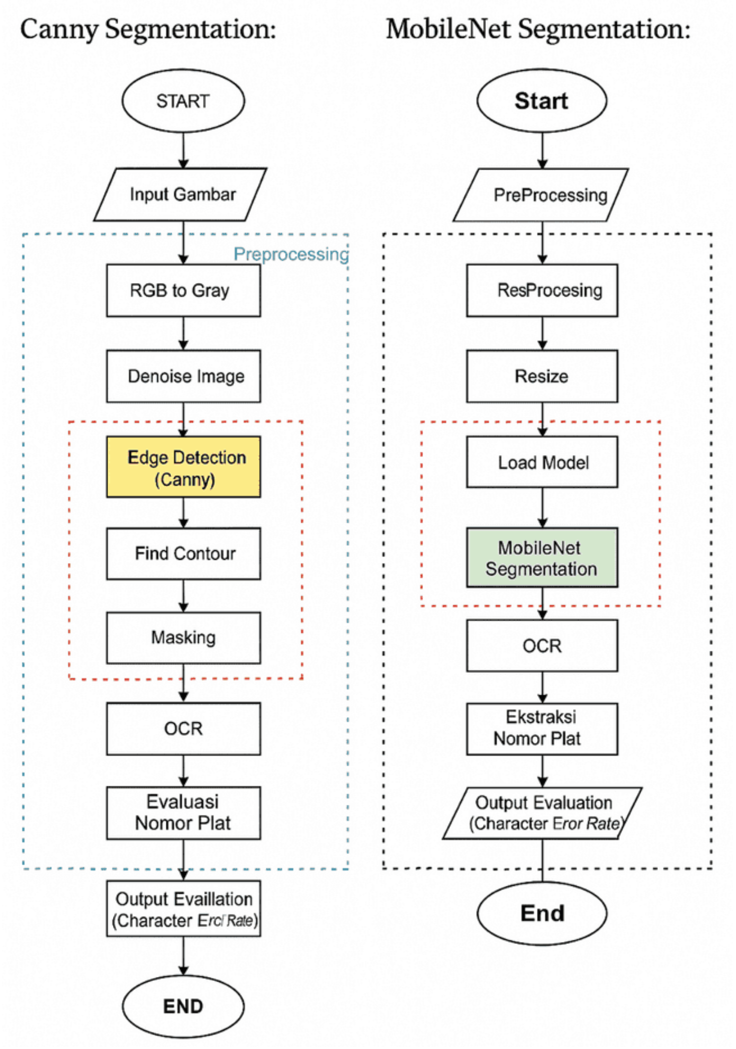
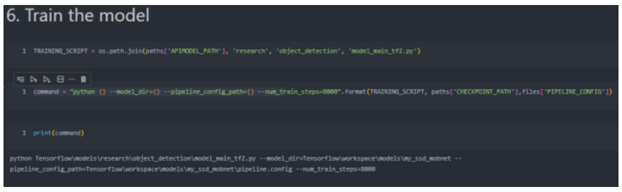


Figure 5. Segmentation

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

```
TRAINING_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection', 'model_main_tf2.py')
```

```
command = "python {} --model_dir={} --pipeline_config_path={} --num_train_steps=8000".format(TRAINING_SCRIPT, paths['CHECKPOINT_PATH'],files['PIPELINE_CONFIG'])
```

```
print(command)
```

```
python Tensorflow\models\research\object_detection\model_main_tf2.py --model_dir=Tensorflow\workspace\models\my_ssd_mobnet --
pipeline_config_path=Tensorflow\workspace\models\my_ssd_mobnet\pipeline.config --num_train_steps=8000
```

Figure 6. Script for training a Model

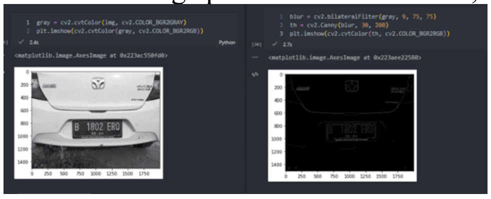
3.4 *Data Analysis dan Validation*

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.



```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
plt.imshow(cv2.cvtColor(gray, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x223ac550fd0>
```

```
blur = cv2.bilateralFilter(gray, 9, 75, 75)
th = cv2.Canny(blur, 30, 200)
plt.imshow(cv2.cvtColor(th, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x223aee22580>
```

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.

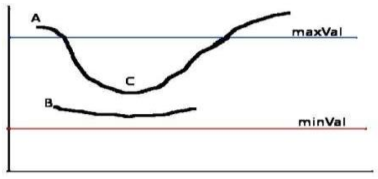
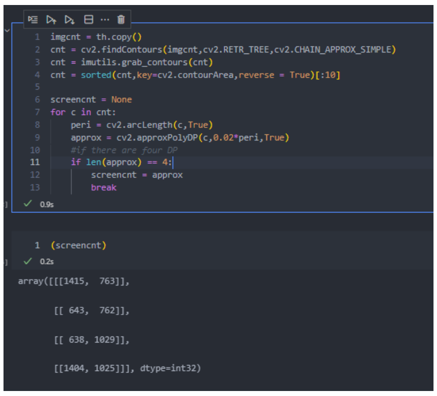


Figure 8. Hysteresis Tresholding

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.



```python
imgcnt = th.copy()
cnt = cv2.findContours(imgcnt,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
cnt = imutils.grab_contours(cnt)
cnt = sorted(cnt,key=cv2.contourArea,reverse = True)[:10]

screencnt = None
for c in cnt:
    peri = cv2.arcLength(c,True)
    approx = cv2.approxPolyDP(c,0.02*peri,True)
    #if there are four DP
    if len(approx) == 4:
        screencnt = approx
        break
```

```
(screencnt)
```

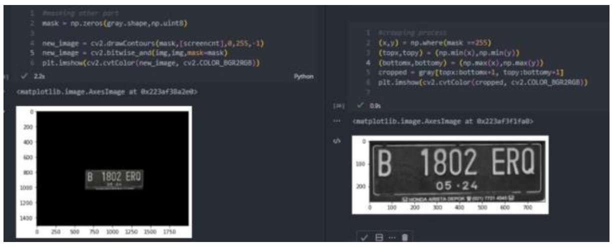
```
array([[[1415,  763]],

       [[ 643,  762]],

       [[ 638, 1029]],

       [[1404, 1025]]], 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.



```python
#masking other part
mask = np.zeros(gray.shape,np.uint8)

new_image = cv2.drawContours(mask,[screencnt],0,255,-1)
new_image = cv2.bitwise_and(img,img,mask=mask)
plt.imshow(cv2.cvtColor(new_image, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x223af38a2e0>
```

```python
#cropping process
(x,y) = np.where(mask ==255)
(topx,topy) = (np.min(x),np.min(y))
(bottomx,bottomy) = (np.max(x),np.max(y))
cropped = gray[topx:bottomx+1, topy:bottomy+1]
plt.imshow(cv2.cvtColor(cropped, cv2.COLOR_BGR2RGB))
```

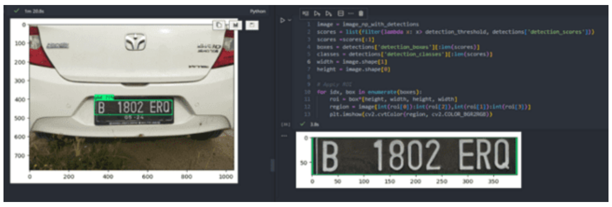
```
<matplotlib.image.AxesImage at 0x223af3f1fa0>
```

Figure 10. Masking

3.4.2 MobileNet Segmentation

In the MobileNet segmentation phase, the process starts by loading the vehicle image and detecting the vehicle plate's position using the pre-trained MobileNet model. Unlike the Canny algorithm, MobileNet segmentation is more efficient, as it uses a trained model to identify the vehicle license plate based on learned patterns, marking it with a bounding box (the green box in Figure 11).

Once the MobileNet model locates the vehicle plate, the image is cropped by extracting the pixel positions within the bounding box. Consequently, only the area identified as the vehicle license plate is retained from the entire vehicle image. The final result is shown in Figure 11.

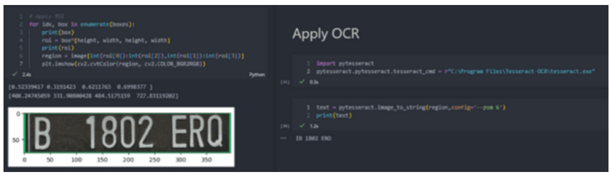


```
image = image_np_with_detections
scores = list(filter(lambda x: x> detection_threshold, detections['detection_scores']))
scores =scores[:1]
boxes = detections['detection_boxes'][:len(scores)]
classes = detections['detection_classes'][:len(scores)]
width = image.shape[1]
height = image.shape[0]

# Apply ROI
for idx, box in enumerate(boxes):
    roi = box*[height, width, height, width]
    region = image[int(roi[0]):int(roi[2]),int(roi[1]):int(roi[3])]
    plt.imshow(cv2.cvtColor(region, cv2.COLOR_BGR2RGB))
```

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.



```
# Apply ROI
for idx, box in enumerate(boxes):
    print(box)
    roi = box*[height, width, height, width]
    print(roi)
    region = image[int(roi[0]):int(roi[2]),int(roi[1]):int(roi[3])]
    plt.imshow(cv2.cvtColor(region, cv2.COLOR_BGR2RGB))
```

```
[0.52339417 0.3191423  0.6211763  0.6998377 ]
[408.24745059 331.98880428 484.5175159  727.83119202]
```

Apply OCR

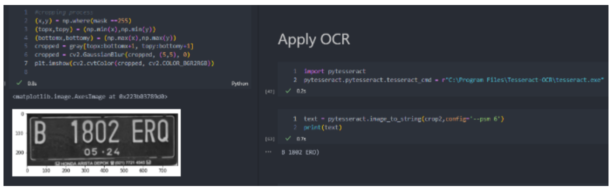
```
import pytesseract
pytesseract.pytesseract.tesseract_cmd = r"C:\Program Files\Tesseract-OCR\tesseract.exe"
```

```
text = pytesseract.image_to_string(region,config='--psm 6')
print(text)
```

```
IB 1802 ERO
```

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.



```
#cropping process
(x,y) = np.where(mask ==255)
(topx,topy) = (np.min(x),np.min(y))
(bottomx,bottomy) = (np.max(x),np.max(y))
cropped = gray[topx:bottomx+1, topy:bottomy+1]
cropped = cv2.GaussianBlur(cropped, (5,5), 0)
plt.imshow(cv2.cvtColor(cropped, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x223b03789d0>
```

Apply OCR

```
import pytesseract
pytesseract.pytesseract.tesseract_cmd = r"C:\Program Files\Tesseract-OCR\tesseract.exe"
```

```
text = pytesseract.image_to_string(crop2,config='--psm 6')
print(text)
```

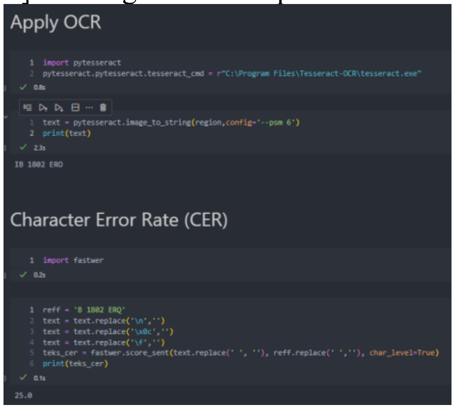
```
B 1802 ERO)
```

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.



Apply OCR

```
import pytesseract
pytesseract.pytesseract.tesseract_cmd = r"C:\Program Files\Tesseract-OCR\tesseract.exe"
```

```
text = pytesseract.image_to_string(region,config='--psm 6')
print(text)
```

```
IB 1802 ERO
```

Character Error Rate (CER)

```
import fastwer
```

```
reff = 'B 1802 ERQ'
text = text.replace('\n','')
text = text.replace('\x0c','')
text = text.replace('\f','')
teks_cer = fastwer.score_sent(text.replace(' ', ''), reff.replace(' ',''), char_level=True)
print(teks_cer)
```

```
25.0
```

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:

$$CER = \frac{S+D+I}{N} \tag{1}$$

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):

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

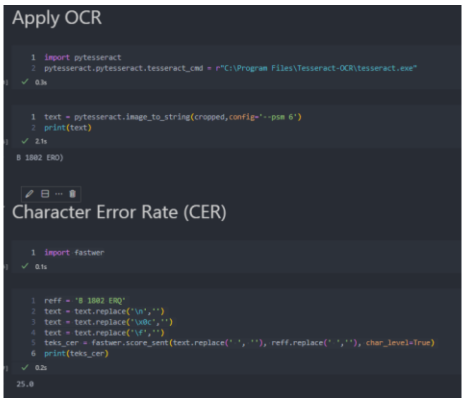
$$CER = \frac{2}{8}$$

$$CER = 0,25$$

$$CER = 25\%$$

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

```
import pytesseract
pytesseract.pytesseract.tesseract_cmd = r"C:\Program Files\Tesseract-OCR\tesseract.exe"
```

✓ 0.3s

```
text = pytesseract.image_to_string(cropped,config='--psm 6')
print(text)
```

✓ 2.1s

```
B 1802 ERO)
```

Character Error Rate (CER)

```
import fastwer
```

✓ 0.1s

```
reff = 'B 1802 ERQ'
text = text.replace('\n','')
text = text.replace('\x0c','')
text = text.replace('\f','')
teks_cer = fastwer.score_sent(text.replace(' ', ''), reff.replace(' ',''), char_level=True)
print(teks_cer)
```

✓ 0.2s

```
25.0
```

Figure 15. CER for Canny Algorithm Segmentation

Figure 15 presents the OCR results of the vehicle license plate using the Canny algorithm (B 1802 ERO). The error calculation using CER compares the following:

OCR Result = B 1802 ERO

Original Plate = B 1802 ERQ (reference variable)

The CER validation indicates an error rate of 25%, calculated using formula 1.

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, "Pengenalan 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 Pengenalan 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.