Computer Vision Project

License Plate Detection

Group Members

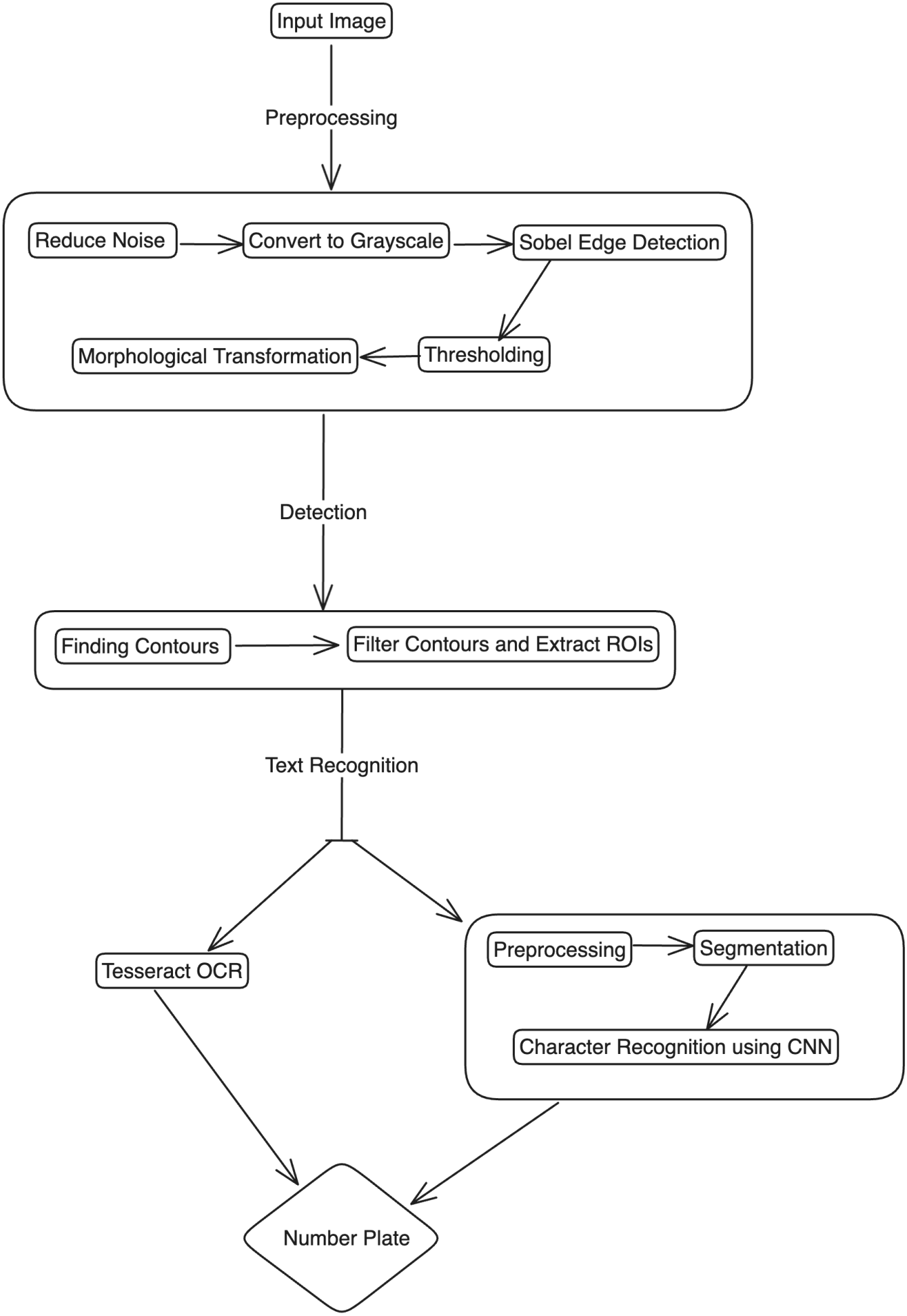
- Jiya Kumawat B21CS036
- Himanshu Gupta B21CS034
- Arnav Singhal B21CS014

Problem Statement

> The security personnel at our campus have to manually verify the details of each vehicle entering and exiting the premises. This process is tedious, time-consuming, and requires them to work in harsh outdoor conditions, such as scorching sunlight. All the vehicles entries are made in a offline documents, which are prone to physical damage. The objective of this project is to develop an automated number plate detection system that can efficiently identify and record the license plate numbers of vehicles passing through the campus gates. This system aims to reduce the workload on security guards, streamline the vehicle entry/exit process as well as remove the use of offline documents and manual entries. Such a system can also help reduce manual errors that may arise due to bad handwriting, erroneous reading or hearing of license plate numbers.
> The main benefits over traditional system are:-
>
> - Improved Efficiency: The system automates the license plate reading process, significantly reducing processing time and eliminating queues at entry points.
> - Enhanced Accuracy: By eliminating human error in data entry, the detection system ensures accurate identification of vehicles and promotes robust security.
> - Reduced Reliance on Manpower: The system frees up security personnel from manual verification tasks, allowing them to focus on more strategic security measures.
> - Improved Working Conditions: Security guards are no longer required to stand outdoors in harsh weather, leading to improved working conditions and potentially higher alertness.

Methodology

> The functionalities can be broken down into a series of distinct stages:


Input Image
Preprocessing
Reduce Noise
Convert to Grayscale
Sobel Edge Detection
Thresholding
Morphological Transformation
Detection
Finding Contours
Filter Contours and Extract ROIs
Text Recognition
Tesseract OCR
Preprocessing
Segmentation
Character Recognition using CNN
Number Plate

Before, we discuss preprocessing of the image, the following functions used in the detection are explained:-

Preprocessing

`clean2_plate`: This function is designed to clean a license plate image by performing various image processing operations to isolate the characters on the plate.

```
def clean2_plate(plate):
        gray_img = cv2.cvtColor(plate, cv2.COLOR_BGR2GRAY)
        _, thresh = cv2.threshold(gray_img, 110, 255, cv2.THRESH_BINARY)
        if cv2.waitKey(0) & 0xff == ord('q'):
            pass
        num_contours, _ = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)

        if num_contours:
            contour_area = [cv2.contourArea(c) for c in num_contours]
            max_cntr_index = np.argmax(contour_area)
            max_cnt = num_contours[max_cntr_index]
            max_cntArea = contour_area[max_cntr_index]
            x, y, w, h = cv2.boundingRect(max_cnt)

            if not ratioCheck(max_cntArea, w, h):
                return plate, None

            final_img = thresh[y:y+h, x:x+w]
            return final_img, [x, y, w, h]

        else:
            return plate, None
```

Input Parameters:

- `plate`: A NumPy array representing the input license plate image in BGR color format.

Output Parameters:

- `final_img`: A NumPy array representing the cleaned and isolated characters on the license plate in binary format.
  - `[x, y, w, h]`: A list containing the coordinates and dimensions of the bounding box around the isolated characters.

Theory:

1. Convert the input plate image from BGR color space to grayscale using OpenCV's `cv2.cvtColor()` function.
2. Apply a binary thresholding operation to the grayscale image using `cv2.threshold()`. This operation converts the grayscale image into a binary image where pixels with intensity greater than 110 are set to 255 (white) and pixels with intensity less than or equal to 110 are set to 0 (black).
3. Find contours in the thresholded image using `cv2.findContours()`. Contours are the outlines of objects in an image.
4. If contours are found:
    - Calculate the area of each contour using `cv2.contourArea()`.
    - Find the contour with the maximum area.
    - Extract the bounding box coordinates (x, y, width, height) of the maximum contour using `cv2.boundingRect()`.
    - Check if the aspect ratio of the bounding box satisfies a certain condition using the `ratioCheck()` function.
    - If the condition is satisfied, extract the region of interest (ROI) from the thresholded image using the bounding box coordinates.
    - Return the extracted ROI and its bounding box coordinates.
5. If no contours are found, return the original plate image and None.

`ratioCheck`: The `ratioCheck` function is used to evaluate whether a given bounding box satisfies certain criteria based on its area and aspect ratio.

```
def ratioCheck(area, width, height):
        ratio = float(width) / float(height)
        if ratio < 1:
            ratio = 1 / ratio
        if (area < 1063.62 or area > 73862.5) or (ratio < 3 or ratio > 6):
            return False
        return True
```

Input Parameters:

- `area`: The area of the bounding box.
- `width`: The width of the bounding box.
- `height`: The height of the bounding box.

Output Parameters:

- Returns `True` if the bounding box meets the specified criteria, otherwise `False`.

Theory:

1. Calculate the aspect ratio of the bounding box by dividing its width by its height. This is done to ensure that the aspect ratio is always greater than or equal to 1.
2. If the aspect ratio is less than 1, invert it to ensure a consistent comparison.
3. Check if the area of the bounding box falls within a specified range (1063.62 to 73862.5) and if the aspect ratio falls within another specified range (3 to 6).
4. Return True if both conditions are met, indicating that the bounding box is suitable for further processing. Otherwise, return False.

Note: The specified area and aspect ratio ranges are chosen based on the expected dimensions and proportions of license plate characters. These values may need adjustment based on the specific characteristics of the license plates being processed.

`isMaxWhite`: The `isMaxWhite` function is designed to determine whether the mean pixel intensity of a given image is sufficiently high, indicating that the image contains mostly white pixels.

```
def isMaxWhite(plate):
        avg = np.mean(plate)
        if avg >= 115:
            return True
        else:
            return False
```

Input Parameters:

- `plate`: A NumPy array representing the input image.

Output Parameters:

- Returns `True` if the mean pixel intensity of the input image is greater than or equal to 115, indicating a predominantly white image. Otherwise, returns `False`.

Theory:

1. Calculate the mean pixel intensity of the input image using NumPy's `np.mean()` function.
2. Check if the mean intensity is greater than or equal to 115.

3. If the mean intensity is greater than or equal to 115, return True, indicating that the image contains mostly white pixels.
4. If the mean intensity is less than 115, return False, indicating that the image does not contain predominantly white pixels.

Note: The threshold value of 115 for the mean intensity is chosen based on empirical observations of typical pixel intensities in images containing white objects. Adjustments may be necessary depending on the specific characteristics of the input images.

`ratio_and_rotation`: The `ratio_and_rotation` function is designed to determine whether a given rectangle (specified by its position, dimensions, and rotation angle) satisfies certain criteria related to its aspect ratio and rotation angle.

```
def ratio_and_rotation(rect):
        (x, y), (width, height), rect_angle = rect

        if width > height:
            angle = -rect_angle
        else:
            angle = 90 + rect_angle

        if angle > 180:
            return False

        if height == 0 or width == 0:
            return False

        area = height * width
        if not ratioCheck(area, width, height):
            return False
        else:
            return True
```

Input Parameters:

- `rect`: A tuple containing information about the rectangle, including:
  - `(x, y)`: The coordinates of the top-left corner of the rectangle.
  - `(width, height)`: The dimensions (width and height) of the rectangle.
  - `rect_angle`: The rotation angle of the rectangle.

Output Parameters:

- Returns `True` if the rectangle meets the specified criteria, otherwise `False`.

Theory:

1. Extract the components of the input rectangle tuple: `(x, y), (width, height), and rect_angle`.
2. Determine the angle of rotation required to make the rectangle upright. If the width is greater than the height, the angle is set to the negative of the rectangle's angle; otherwise, it is set to `90` degrees plus the rectangle's angle.
3. Check if the calculated angle is greater than 180 degrees. If so, return `False`, as it indicates an invalid rotation angle.
4. Check if the height or width of the rectangle is zero. If either dimension is zero, return `False`, as it indicates an invalid rectangle.
5. Calculate the area of the rectangle using the formula: `area = height * width`.
6. Check if the rectangle satisfies the aspect ratio criteria by calling the `ratioCheck` function with the `area`, `width`, and `height` as parameters. If the aspect ratio criteria are not met, return `False`; otherwise, return `True`.

`number_plate_detection`: The `number_plate_detection` function is responsible for detecting license plates in an input image using various image processing techniques and returning relevant information about the detected plates.

Input Parameters:

- `img`: A NumPy array representing the input image in BGR color format.

Output Parameters:

- `cnt`: The contour of the detected license plate.
- `img_with_contours`: The input image with overlaid contours.
- `img`: The original input image.

Theory:

1. The function begins by creating a copy of the input image to work with, stored in img_with_contours. This copy will be used to visualize the detected contours.
2. Several image processing operations are performed on the input image sequentially:
    1. Gaussian blur is applied to reduce noise in the image.
    2. The image is converted to grayscale.
    3. Sobel edge detection is used to highlight edges in the image.

    4. Thresholding is applied to create a binary image using Otsu's method.
    5. Morphological operations (closing) are performed to further enhance the contours in the image.
3. Contours are then found in the processed image using the cv2.findContours function.
4. For each contour found:
    1. The minimum bounding rectangle (min_rect) is calculated using cv2.minAreaRect.
    2. The ratio_and_rotation function is called to check if the bounding rectangle meets certain criteria related to aspect ratio and rotation.
    3. If the criteria are met, the bounding rectangle coordinates are used to extract the region of interest (plate_img) from the original image.
    4. The isMaxWhite function is called to check if the extracted region contains mostly white pixels, which is a common characteristic of license plates.
    5. If the region passes the white pixel check, the contour, img_with_contours, and the original image are returned.
5. If no valid license plate is found, None is returned.

Results

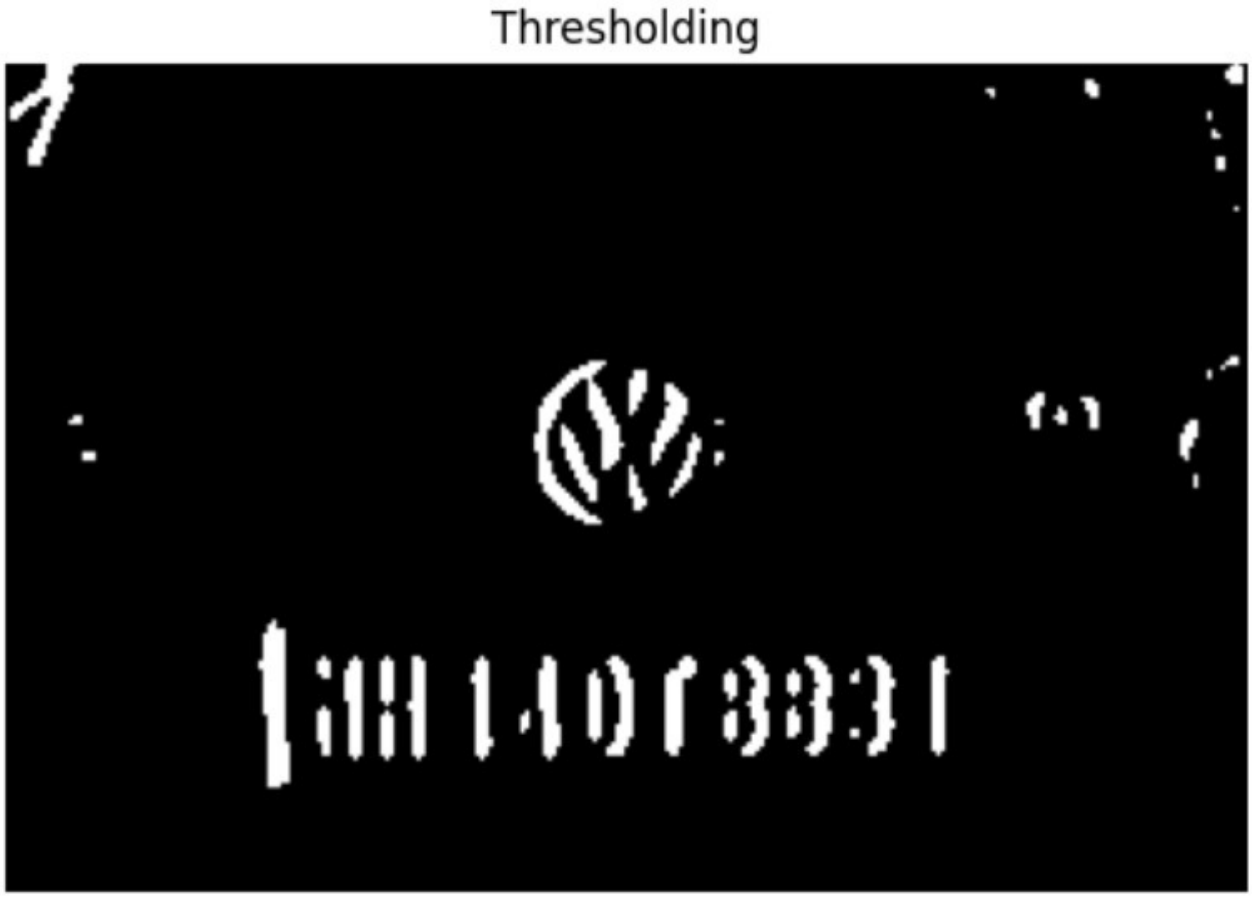
1. Original Image:

2. Blurred Image:

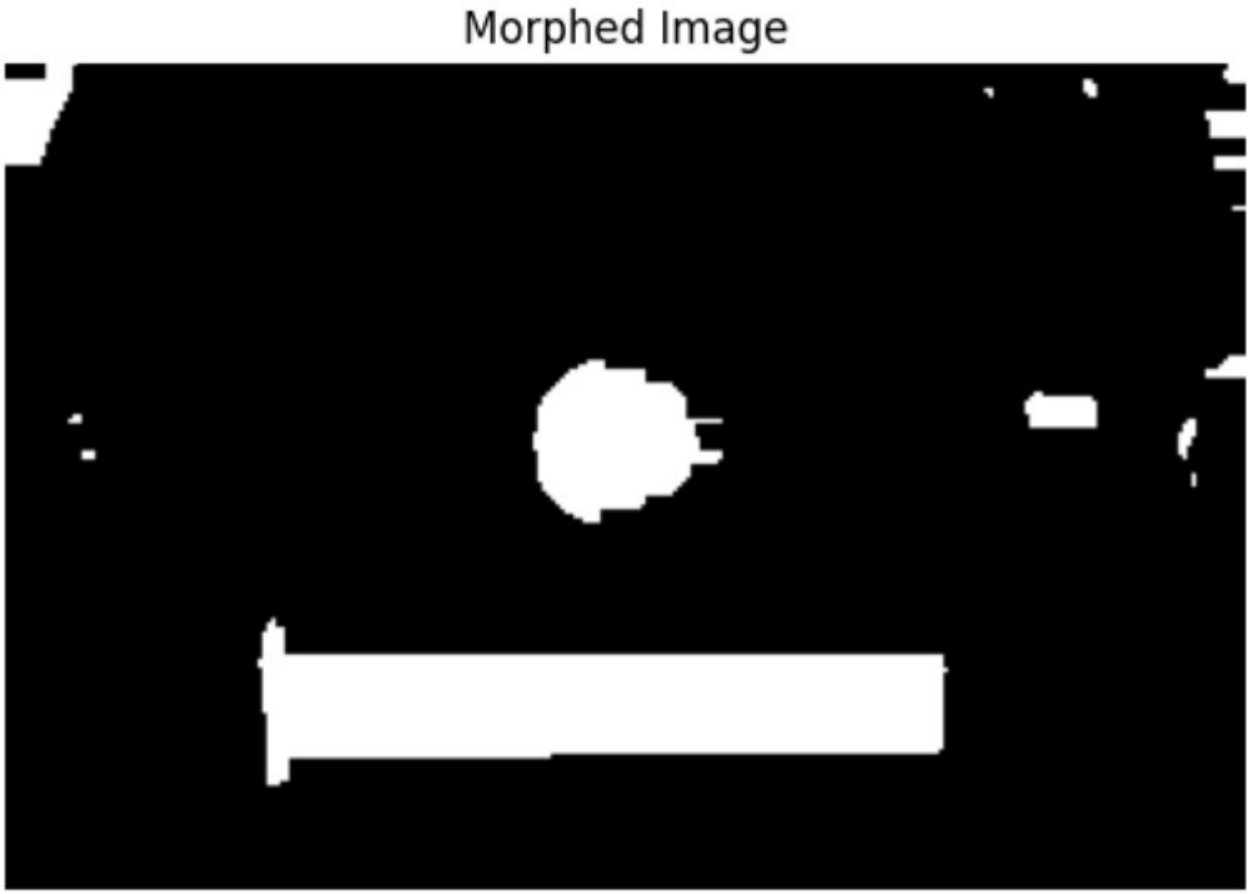
3. Grayscale Image:

4. Sobel Edge Detected Image:

5. Thresheld Image:



6. Morphed Image (Close Morphing):



7. Image with plate:

8. Detected Plate:

So, with this we have extracted the number plate from the image of a car.

Recognition

Now, that we have extracted the license plate, we need to clean it and recognise the text. To do this we have initially we used Tesseract OCR. For cleaning we have used the clean2_plate function explained earlier.

Next, we simply pass the cleaned image to the inbuilt image_to_string function for text recognition.

Results

```
Detected License Plate Number is: HH14078831
```

Since, we also learned CNN, we next tried to implement OCR by ourselves by creating a neural network. The current cleaned plate required further preprocessing for CNN-based recognition. The following functions are used to do that:-

---

`find_contours` : The find_contours function is designed to extract and process potential characters from an input binary image, such as a license plate segmentation output.

```
def find_contours(dimensions, img) :

    # Find all contours in the image
    cntrs, _ = cv2.findContours(img.copy(), cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

    # Retrieve potential dimensions
    lower_width = dimensions[0]
    upper_width = dimensions[1]
    lower_height = dimensions[2]
    upper_height = dimensions[3]

    # Check largest 5 or  15 contours for license plate or character
respectively
    cntrs = sorted(cntrs, key=cv2.contourArea, reverse=True)[:15]

    ii = cv2.imread('contour.jpg')

    x_cntr_list = []
    target_contours = []
    img_res = []
    for cntr in cntrs :
        # detects contour in binary image and returns the coordinates of
rectangle enclosing it
        intX, intY, intWidth, intHeight = cv2.boundingRect(cntr)

        # checking the dimensions of the contour to filter out the
characters by contour's size
```

```
        if intWidth > lower_width and intWidth < upper_width and intHeight >
lower_height and intHeight < upper_height :
            x_cntr_list.append(intX) #stores the x coordinate of the
character's contour, to used later for indexing the contours

            char_copy = np.zeros((44,24))
            # extracting each character using the enclosing rectangle's
coordinates.
            char = img[intY:intY+intHeight, intX:intX+intWidth]
            char = cv2.resize(char, (20, 40))

            cv2.rectangle(ii, (intX,intY), (intWidth+intX, intY+intHeight),
(50,21,200), 2)
            plt.imshow(ii, cmap='gray')
            plt.title('Predict Segments')

            # Make result formatted for classification: invert colors
            char = cv2.subtract(255, char)

            # Resize the image to 24x44 with black border
            char_copy[2:42, 2:22] = char
            char_copy[0:2, :] = 0
            char_copy[:, 0:2] = 0
            char_copy[42:44, :] = 0
            char_copy[:, 22:24] = 0

            img_res.append(char_copy) # List that stores the character's
binary image (unsorted)

    # Return characters on ascending order with respect to the x-coordinate
(most-left character first)

    plt.show()
    # arbitrary function that stores sorted list of character indeces
    indices = sorted(range(len(x_cntr_list)), key=lambda k: x_cntr_list[k])
    img_res_copy = []
    for idx in indices:
        img_res_copy.append(img_res[idx])# stores character images according
to their index
    img_res = np.array(img_res_copy)

    return img_res
```

Input Parameters:

- `dimensions`: A list containing four integer values representing the lower and upper bounds for character width and height.
- `img`: A NumPy array representing the binary input image containing potential character contours.

Output Parameters:

- `img_res:` A NumPy array containing the extracted character images sorted in ascending order based on their x-coordinate (most-left character first).

Theory:

1. The function begins by finding all contours in the input binary image using cv2.findContours with the RETR_TREE retrieval mode, which retrieves all contours and reconstructs the hierarchy.
2. It then extracts the lower and upper bounds for character width and height from the dimensions list.
3. The function sorts the contours based on their area in descending order and retains only the largest 15 contours.
4. It initializes lists to store the x-coordinates of the contours (x_cntr_list), the extracted character images (img_res), and a copy of the input image (ii) for visualization purposes.
5. It iterates through each contour and calculates the bounding rectangle enclosing it using cv2.boundingRect.
6. For each contour, it checks if the dimensions of the bounding rectangle fall within the specified lower and upper bounds for character width and height. If so, it proceeds to extract and process the character.
7. It resizes the extracted character to a standard size (20x40) and inverts its colors.
8. It creates a black-bordered image of size 24x44 and places the resized character in the center.
9. It appends the processed character image to the img_res list.
10. After processing all contours, it sorts the character images in img_res based on their x-coordinates using the indices list.
11. It returns the sorted character images as a NumPy array.

`segment_characters`: The segment_characters function aims to preprocess a cropped license plate image and segment individual characters from it based on contours.

```
def segment_characters(image) :
```

```
    # Preprocess cropped license plate image
    img_lp = cv2.resize(image, (333, 75))
    img_gray_lp = cv2.cvtColor(img_lp, cv2.COLOR_BGR2GRAY)
    _, img_binary_lp = cv2.threshold(img_gray_lp, 200, 255,
cv2.THRESH_BINARY+cv2.THRESH_OTSU)
    img_binary_lp = cv2.erode(img_binary_lp, (3,3))
    img_binary_lp = cv2.dilate(img_binary_lp, (3,3))

    LP_WIDTH = img_binary_lp.shape[0]
    LP_HEIGHT = img_binary_lp.shape[1]

    # Make borders white
    img_binary_lp[0:3,:] = 255
    img_binary_lp[:,0:3] = 255
    img_binary_lp[72:75,:] = 255
    img_binary_lp[:,330:333] = 255

    # Estimations of character contours sizes of cropped license plates
    dimensions = [LP_WIDTH/6,
                  LP_WIDTH/2,
                  LP_HEIGHT/10,
                  2*LP_HEIGHT/3]
    plt.imshow(img_binary_lp, cmap='gray')
    plt.title('Contour')
    plt.show()

    # Get contours within cropped license plate
    char_list = find_contours(dimensions, img_binary_lp)

    return char_list, img_binary_lp
```

Input Parameters:

- `image:` A NumPy array representing the cropped license plate image in BGR color format.

Output Parameters:

- `char_list:` A NumPy array containing the segmented character images.
- `img_binary_lp:` A NumPy array representing the binary image after preprocessing.

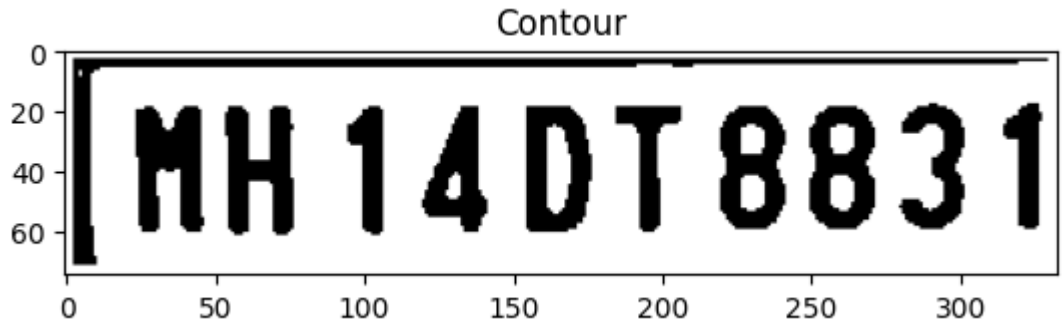
Theory:

1. The function starts by resizing the input license plate image to a fixed size (333x75) using cv2.resize.

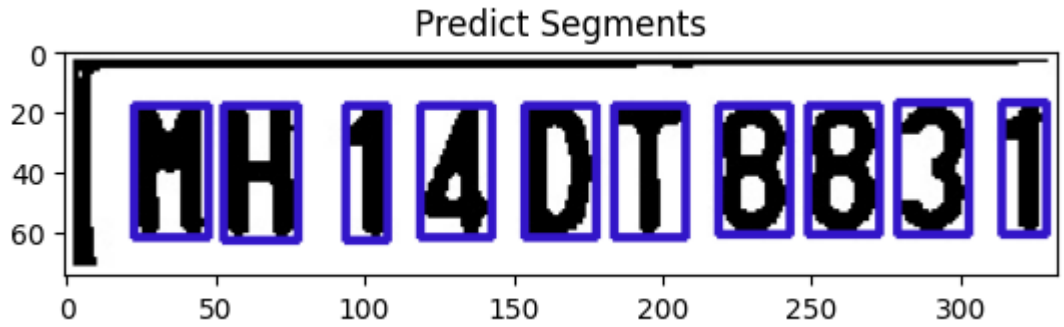
2. It converts the resized image to grayscale (img_gray_lp) and applies Otsu's thresholding method to binarize it using cv2.threshold. The threshold value is set to 200.
3. Morphological operations (erosion and dilation) are applied to the binary image using cv2.erode and cv2.dilate to remove noise and smoothen the contours.
4. Borders of the binary image are made white to eliminate potential noise or artifacts.
5. Estimations of character contours sizes are defined based on the dimensions of the cropped license plate.
6. The binary image is visualized using plt.imshow and plt.show for debugging purposes.
7. The find_contours function is called to extract potential character contours from the preprocessed binary image.
8. The function returns the segmented character images (char_list) and the preprocessed binary image (img_binary_lp).

Results

1. Preprocessed:



2. Predicted Segments:



3. char is list of segmented character images:

Now, that we have segmented the characters, we will use CNN to recognise characters. Here, is the overview:

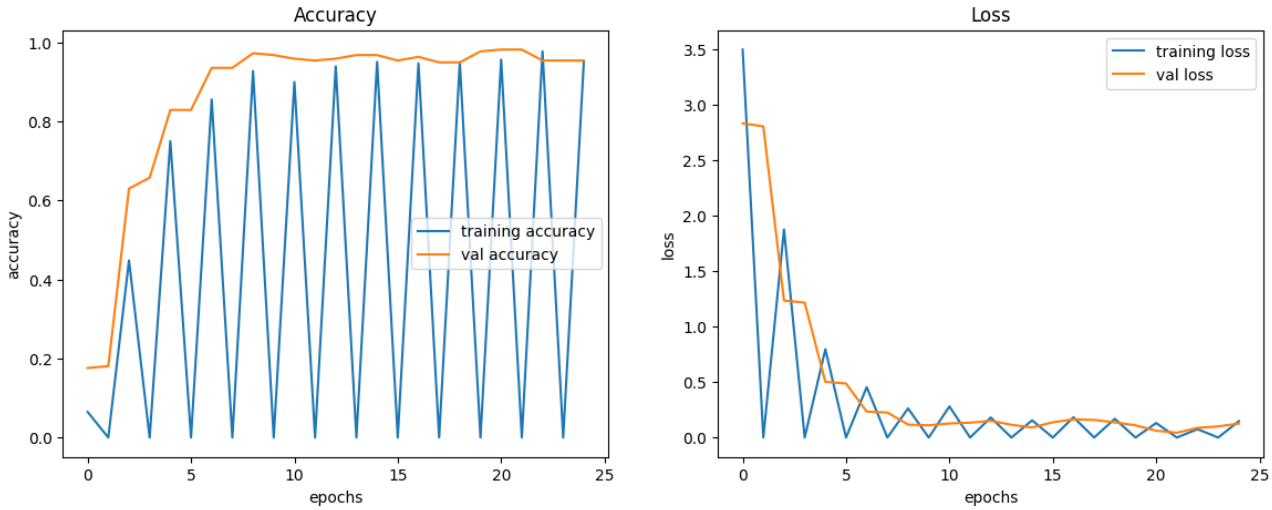
4. **Data Preparation:**
    - The `train_datagen` object is created with an ImageDataGenerator instance that includes various data augmentation techniques such as rescaling and random shifts in width and height.
    - Training and validation data directories are specified, and ImageDataGenerator instances are initialized for both directories using the `flow_from_directory` method. This method generates batches of augmented/normalized data from image files in the directory, with images resized to 28x28 pixels and a batch size of 1.
5. **Model Architecture:**
    - The model architecture is defined using the Sequential API in Keras.
    - It consists of multiple convolutional layers (`Conv2D`) with different filter sizes, activation functions (ReLU), and padding modes (same).
    - Max-pooling layers (`MaxPooling2D`) are inserted to reduce spatial dimensions and capture the most important features.
    - Dropout regularization is applied to reduce overfitting by randomly dropping a fraction of the units during training.
    - The fully connected layers (`Dense`) at the end of the model perform classification, with a softmax activation function to output probability distributions over the classes.
6. **Model Compilation and Training:**
    - The model is compiled with the Adam optimizer, a popular choice for deep learning models, and the sparse categorical cross-entropy loss function, suitable for multi-class classification problems with integer labels.
    - The `fit` method is used to train the model on the training data generator, specifying the number of training steps per epoch, validation data, and the number of epochs.
7. **Model Evaluation:**
    - Matplotlib is used to create plots of training and validation accuracy and loss over epochs.
    - Two subplots are generated: one for accuracy and one for loss. These plots help monitor the training progress and identify potential issues such as overfitting or underfitting.
8. **Model Checkpointing:**
    - A directory for storing checkpoints (`checkpoint_dir`) is created if it doesn't exist.
    - The model weights are saved to a checkpoint file (`my_checkpoint.weights.h5`) using the `save_weights` method. This allows the model to be loaded later and used for predictions or further training without retraining from scratch.
9. **Model Loading and Prediction:**
    - Another instance of the model (`loaded_model`) is created with the same architecture as the trained model.

- The saved weights are loaded into this model using the `load_weights` method, restoring the model's parameters to the state they were in when the weights were saved.
- The `show_results` function preprocesses character images, predicts characters using the loaded model, and returns the predicted license plate number.

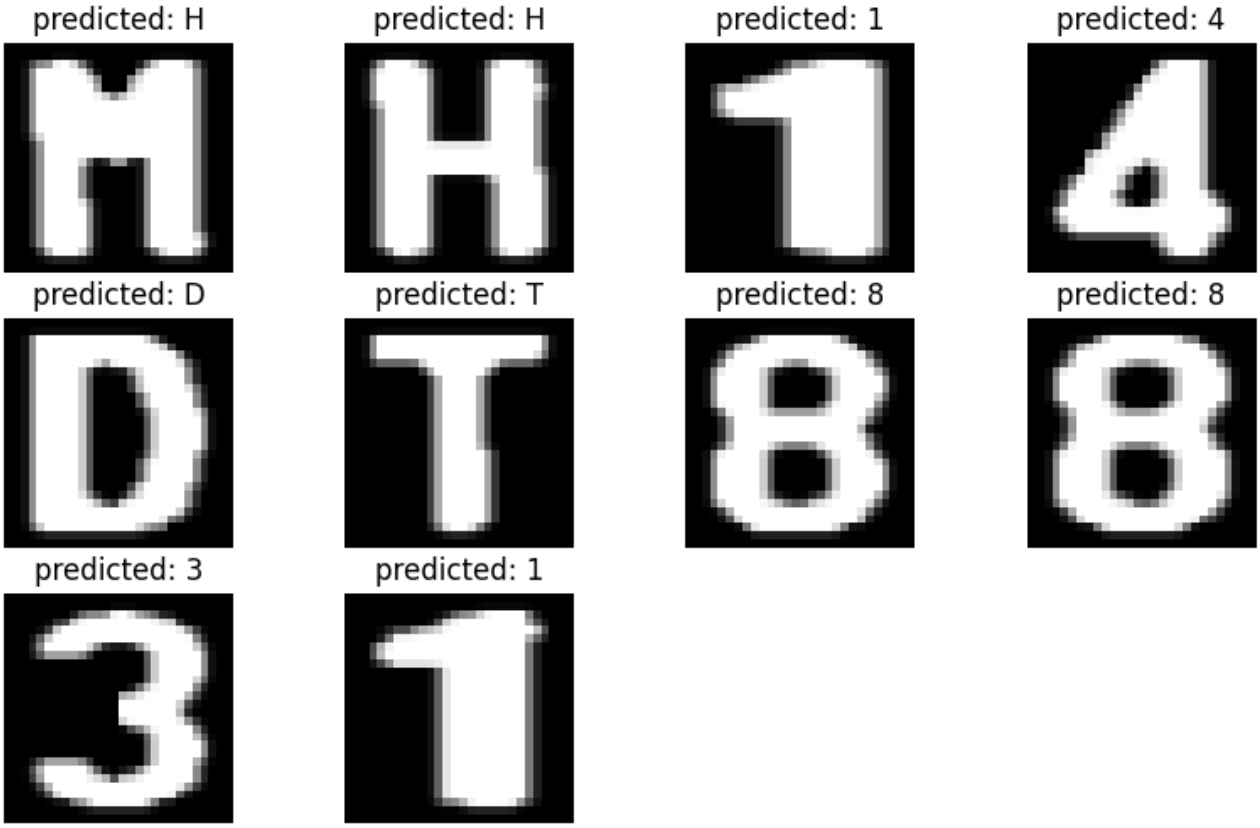
10. **Displaying Results:**
    - The predicted license plate number and corresponding character images are displayed using matplotlib.
    - For each character image, the predicted character is overlaid on top of the image for visual inspection.

Results

1. Accuracy Loss:



2. Recognition:



References:

1. https://github.com/symisc/sod/tree/master
2. https://sod.pixlab.io/articles/license-plate-detection.html
3. https://medium.com/@wongsirikuln/non-deep-learning-license-plate-detection-d2a62e54d555
4. https://www.iosrjournals.org/iosr-jce/papers/Conf.17014-2017/Volume-1/6.%2028-33.pdf?id=7557
5. https://docs.opencv.org/4.x/