License Plates Detection and Recognition
Final Report

Karolina Bogacka, Marek Brynda, Mateusz Wójcik

January 2022

1 Task description

The main purpose of the task is to create a license plates detector. In order to do that, we have decided to split the work into three sub-tasks and divide them between members of the group. These tasks are:

- recognizing the plate area,
- extracting letters and numbers from the plate,
- classifying each of the letters and numbers.

2 Dataset

Unfortunately, large and robust datasets with license number plates are difficult to obtain [8], because:

- these datasets contain sensitive and personal information
- companies do not share these datasets as proprietary information

Our training and testing datasets will be an aggregate dataset, consisting of three smaller image datasets and a set of additional labels. These smaller datasets are:

- AOLP Law Enforcement : `http://aolpr.ntust.edu.tw/lab/`
- SSIG SegPlate Database: `http://www.ssig.dcc.ufmg.br/ssig-segplate-database/`
- Cars dataset : `https://ai.stanford.edu/~jkrause/cars/car_dataset.html`

Data format: CSV
We have additionally managed to obtain a dataset with presegmented licence plate characters detection `https://www.kaggle.com/aladdinss/license-plate-digits-classification-dataset`. Unfortunately, due to the format of the data, we have not been able to use it to improve the performance of any task parts except the last. Additionally,

3 License Plate detection

The first part of the task was the detection of the license plates. To perform it we used 2 approaches:

1. looking for letters and digits,
2. looking for rectangles.

3.1 Looking for the digits

The first of those sub-tasks was done by using the pretrained EAST network [1] specializing in detecting text.

- Firstly we are using the EAST network to detect the regions with possible text occurrences.
- The next step is to join regions if they are similar in height and have similar y coordinates -there could be a plate divided into 2 parts.
- Then we are choosing only regions which meet the condition $w > h$.

Then we save the regions and will use them later in the second part.

3.2 Looking for the rectangles

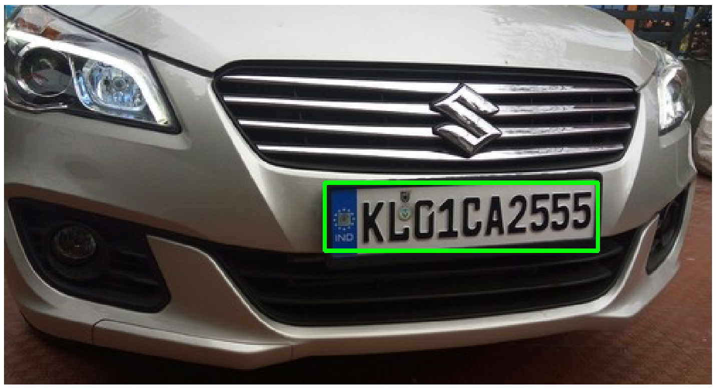

Figure 1: Example image with marked license plate

The second of those sub-tasks was done by using the given steps inspired by post [2]:

- The image was converted to the gray-scale,
- one of 3 versions of the bilateral filter was used,
- The rectangular structures of ratio 13x5 were created and used for removing noise - with blackhat morphology (the size was chosen based on the common size of license plates). We assumed that the background of a license plate is light compared to letters.

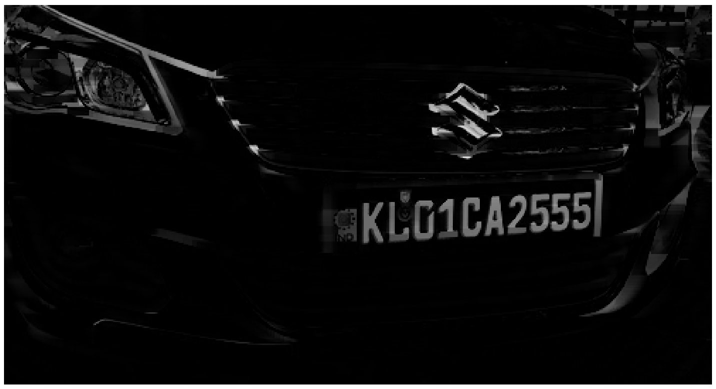

Figure 2: The grayscale image after performing blackhat morphology

- The next step was using the small kernel size with closing morphology performing binary threshold using Otsu's method to reveal light parts which may contain license plate.

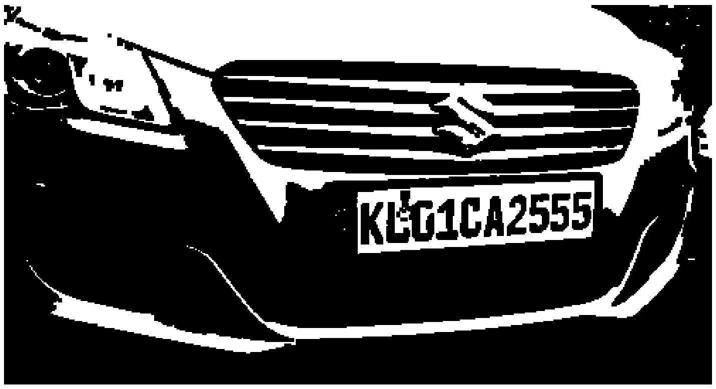

Figure 3: The grayscale image after performing closing morphology

- We compute the Scharr gradient representation in the x-direction of our blackhat image and rescaled it to range [0, 255].

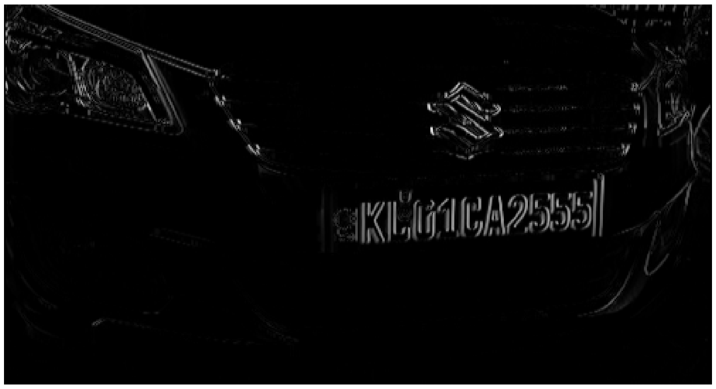

Figure 4: The Scharr gradient in the x-direction after rescaling

- Then the closing morphology (with binary threshold using Otsu's method) was used but this time at x-direction gradient with some gaussian blure to close the regions with close visible vertical lines.

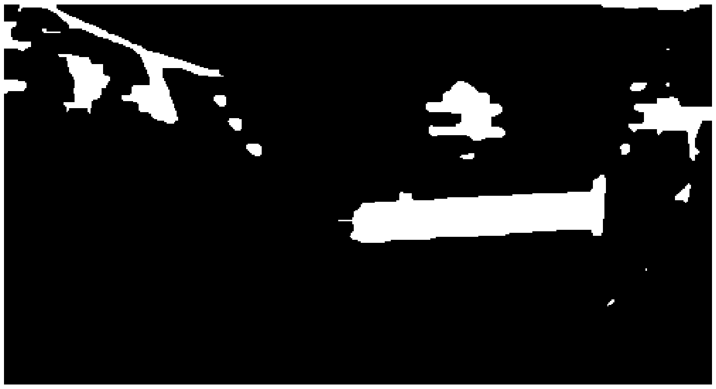

Figure 5: Closing morphology after bluring the x-directional gradient

- The series of of erosions (to get rid of small regionas) and dilations (to extend bigger one) were used in an attempt to smooth a bit the image and get rid of small areas of white spaces.

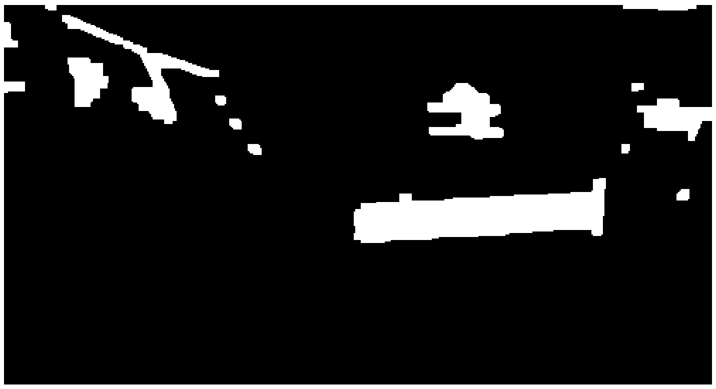
Figure 6: Closing morphology after bluring the x-directional gradient

- Next step was performing bitewise and between smoothed white regions Figure 6 and light parts Figure 3 which may contain the license plate.

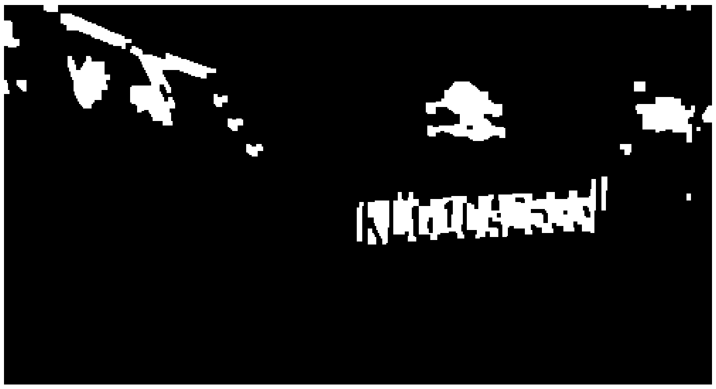
Figure 7: Image after bitewise operation

- The next step was finding the contours in the image, finding the bounding rectangle and checking if the ratio of the rectangles sides is at least $h \times 1.5 < w$ and if so then we are assuming it can be the license plate.

Now we can go back to the list of regions with text created in the Section 3.1. Now hawing the possible rectangles and regions with text we iterate over them and check if are there some regions that appear in both lists. If so then we are joining them and assuming that the license plate can be there, if not then we use list combined list of regions of rectangles and regions with text.

Finally, we have the list of potential places which are rectangles with text. We now again check if some of the regions are overlapping and get the final list of the possible regions.

The next step is:

- to get the chosen region,
- converts to the grayscale,
- rescale it to make it 3 times bigger,
- perform sharpening operation,
- convert to the white and black,
- and having only two values cut only region which at the first time the color changes in x and y-axis.

Now we moved to the next step, choosing the best of them - to get the answer we performed the comparison of histograms for the sample plate and each from the list. Knowing the final best region with license plate we are returning it with 3 color channels in original size and the score how much the grayscale histogram differs from the example one in Figure 8.

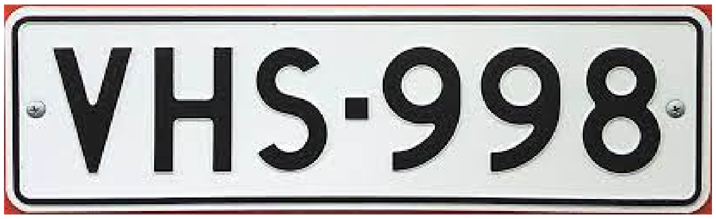

Figure 8: The example license plate

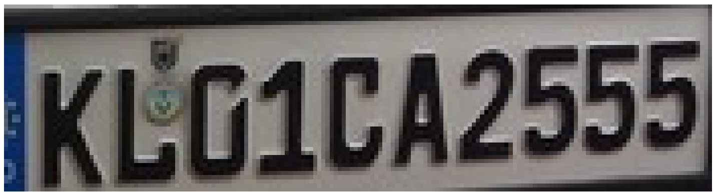

Figure 9: The extracted licence plate from example image

3.3 Choosing the best candidate

The steps from Section 3.2 are performed 3 times for different blurring techniques on greyscale image and now we are choosing the best one from the list of 3, choosing the one with the smallest difference of grayscale histogram of colors from the example license plate. We are assuming that the result with the smallest difference is the license plate.

4 Segmentation of Characters

After segmentation of a plate, there are a lot of possible approaches to extract and classify the letters. One way is to use Tesseract OCR (Optical Character Recognition) which is an open source text recognition engine based on LSTMs. Since the engine may be used as a ready-to-go API, we will try to implement our own, more basic and elementary methods and compare the results with the Tesseract-based approach.

4.1 Colorspace and Thresholding

After using the results of the previous part, a cropped license plate is transformed to split the characters. First, the colorspace of the image is changed and the thresholding is applied. In this step, we have tested two different approaches. In the first one, we changed the RGB color space to the greyscale. Next, the Gaussian blur with 5×5 kernel was applied.

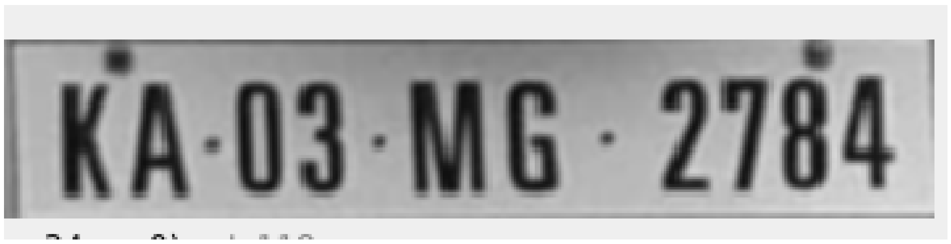

Figure 10: Greyscale and Gaussian Blur

Then, the threshold limit is calculated based on the mean of the intensities in the image. The boundary value is the mean shifted by 75% of the difference between 255 and the mean. Also, the inversed image will be used in the future, since there were different colors of characters and the background.

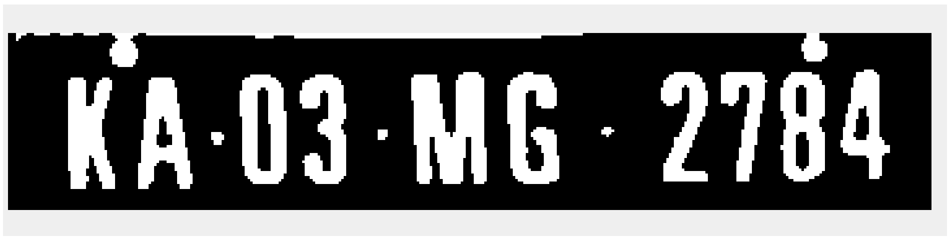

Figure 11: Threshold using the mean

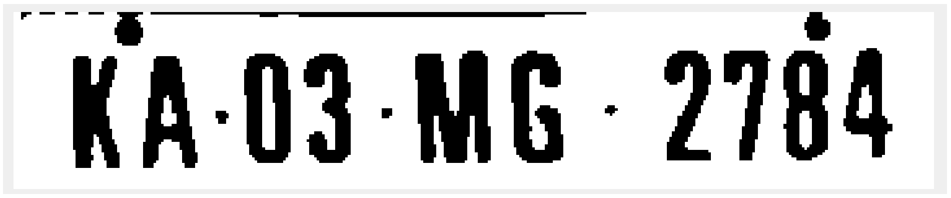

Figure 12: Inversed threshold using the mean

In the second approach, we first extract the Value channel from the HSV color space, because this helps extract dark regions from a light background (or the other way round) and obtain better results than for grayscale. Then the adaptive threshold (local gaussian threshold) is applied.

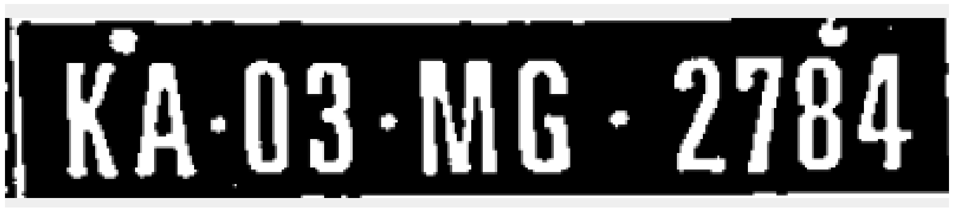

Figure 13: Adaptive threshold based on the Value channel of the HSV color space

In both approaches, we have also tested resizing and sharpening. However, in these cases, plates required more complicated kernel operations to eliminate unnecessary lines, edges, and blobs in the image. To summarize, we have chosen the second approach without resizing and sharpening as a final method.

4.2 Kernel Morphological Transformations

The next step includes applying kernel operations on the binary image. We have checked a lot of different possibilities, which highly depend on the particular license plate. The quality of the image, the spaces between the characters, and different fonts influenced the results significantly. We decided to first apply erosion, as sometimes the characters were very close to each other. Also, some of the blobs disappeared from the image. Then, we apply opening to clear the image even more and finally, we use dilation to emphasize the shape of the characters.

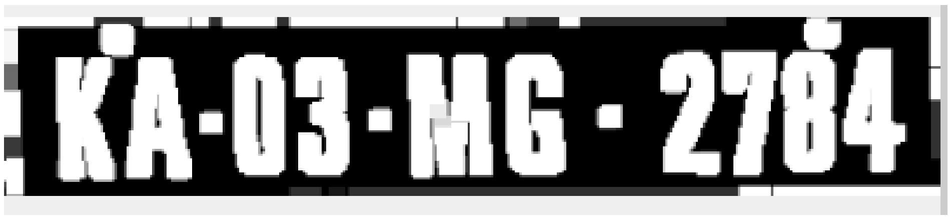

Figure 14: Erosion → Opening → Dilation

Sometimes, morphological transformations were not even necessary for the segmentation of the characters (as we can see in the presented example).

4.3 Finding Contours

Finally, the characters needed to be extracted and cropped from the image of the license plate. To do so, we use the OpenCV function *find contours*. As the arguments, we choose *RETR EXTERNAL* flag to keep only the extreme outer contours (child contours are left behind). Also, we use the *CHAIN APPROX SIMPLE* flag to save memory, as we need only 4 vertices off the character.

During the selection of the obtained contours, we use the fact that characters are usually higher than wider. Based on the UK license plate, the ratio between the height and the width is approximately 8 : 5. With a proper approximation, we use this ratio to exclude non-character shapes. Also, we only choose contours whose height makes up at least 40% of the height of the license plate. This allows to save the proper rectangles and resize the letters to a consistent size. We also add the small black margin to the characters.

Based on the number of found characters (which depends on the chosen binary plate image), characters are cropped from the license plate and ready for the classification.

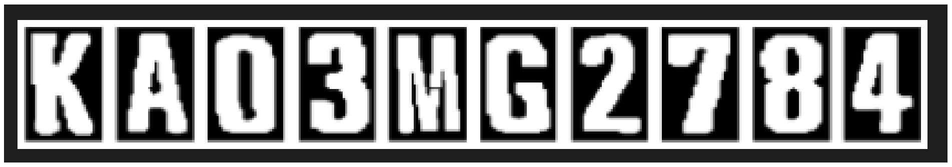

Figure 15: Cropped characters

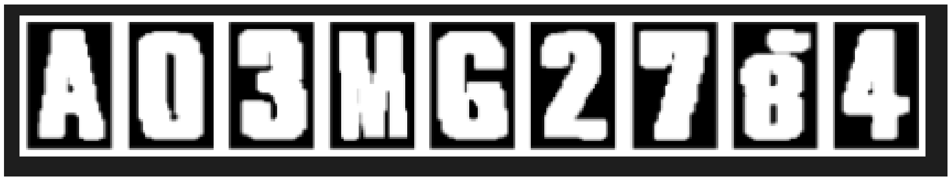

Figure 16: Cropped characters with morphological transformations

4.4 Positive Examples

We want to present some of the correct segmentation examples.

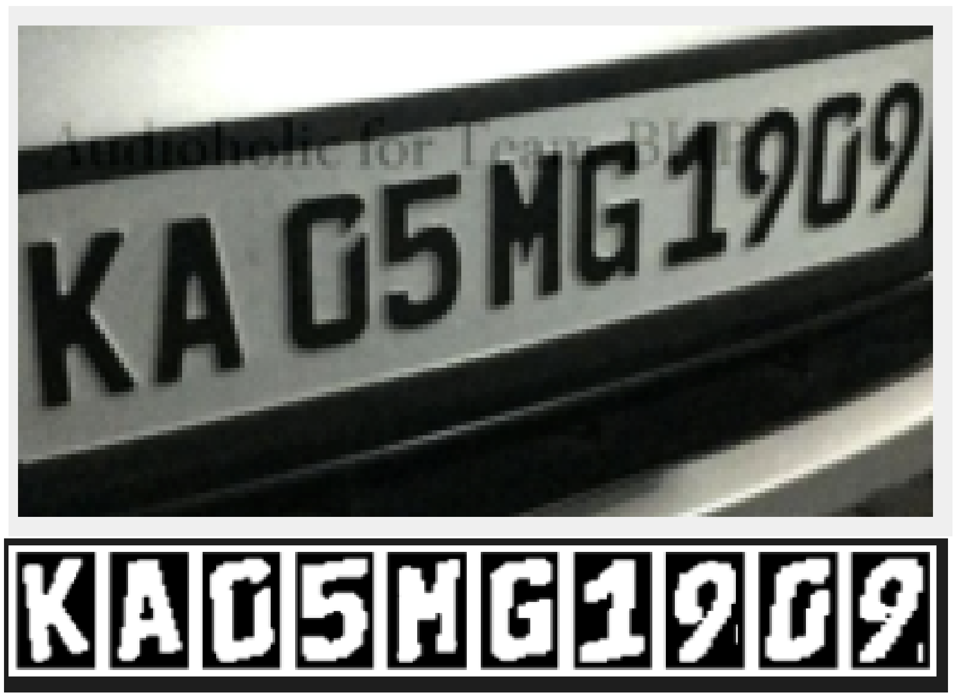

Figure 17: Perfect example, rotated license plate

In the next example, one of the contours consisting 3 numbers was additionally selected. It may be caused by the edge of the plate.

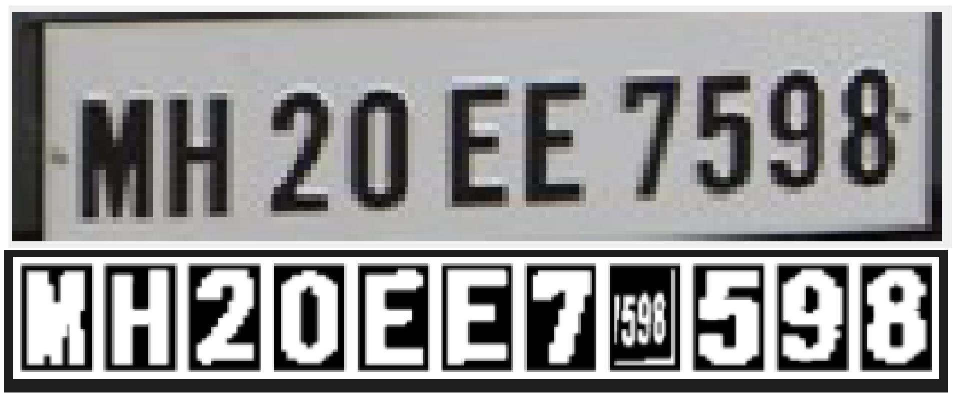

Figure 18: One additional triple character

Sometimes, the method had problems with finding narrower characters, for example "1" or "I".

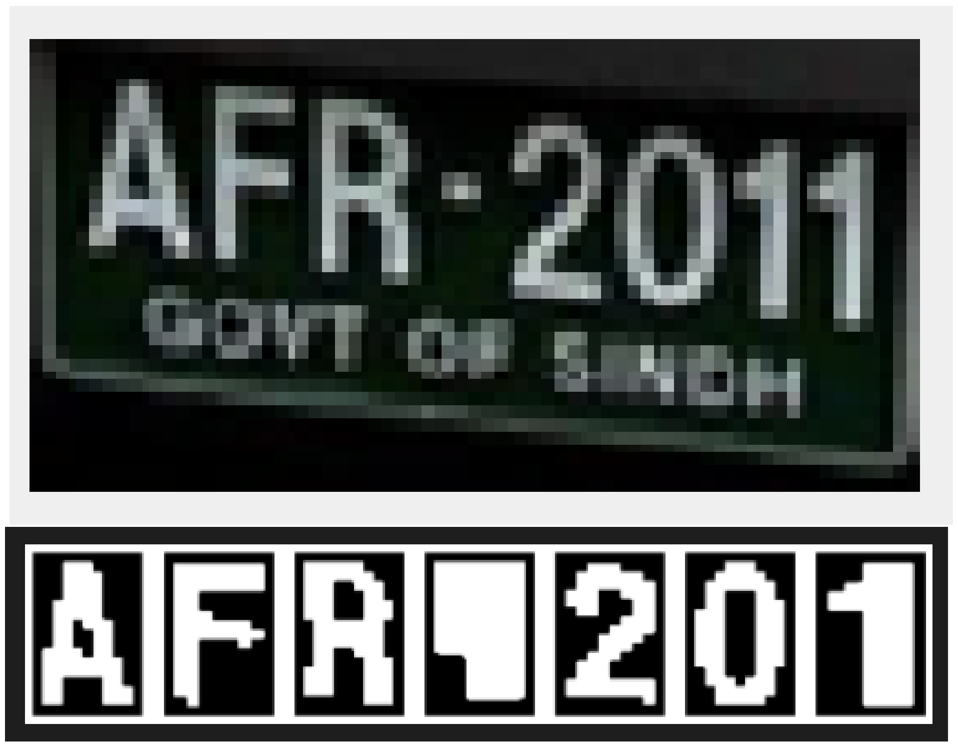

Figure 19: Green background plate

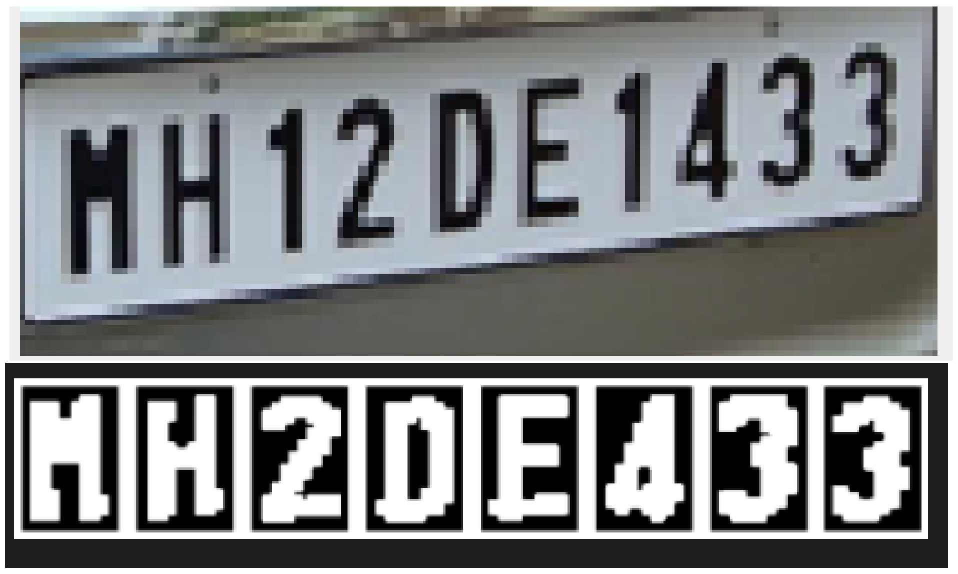

Figure 20: Lack of 1's

4.5 Negative Examples

There are also examples when the chosen approach did not give good results. In the following examples also some other parts of the plate were included.

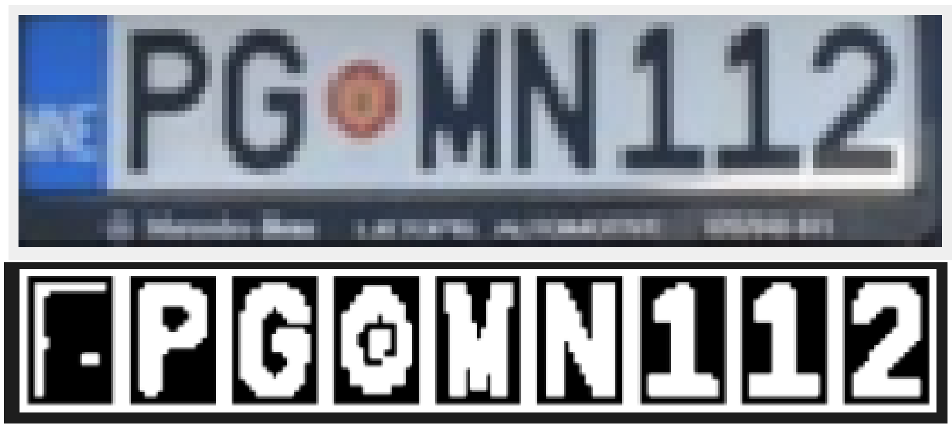

Figure 21: Other objects in the plate

In the following plate, the characters were cropped correctly. Unfortunately, also some other parts of the plate were chosen as characters.

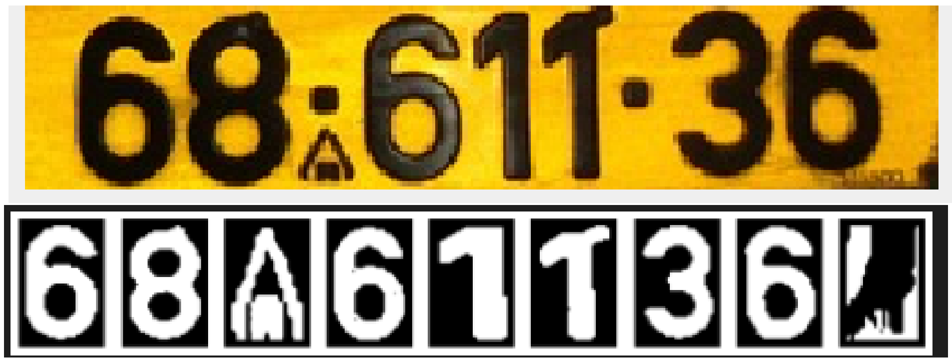

Figure 22: Yellow background plate

The quality of some plates was very low, so that the segmentation of the characters was not possible with used methods.

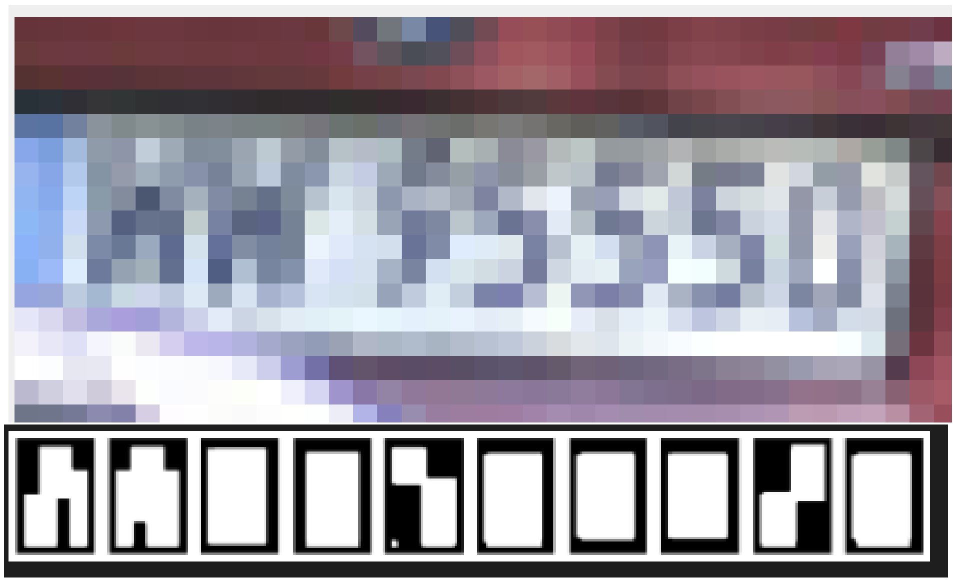

Figure 23: Low quality plate

5 Classification of each of the characters

In order to classify all digits and letters present in extracted images we're going to first preprocess them by employing a form of image binarization. [2] The process of binarization refers to reducing all pixel values in an image either to a value of 0 or 255. In the case of our already extracted digits it should lead to minimal loss of information, since licence plates already consist of black and white letters. Using it, we aimed to improve the efficiency of our classifier down the line [3]. We've analyzed a few binarization techniques, with a preset threshold value and a truncating threshold, and decided to follow with the Otsu binarization due to its flexibility. It allowed us to choose the most appropriate threshold value for each image in the dataset. We have saved the original preprocessed images locally as grayscale images, such as not to needlessly repeat this time consuming process.

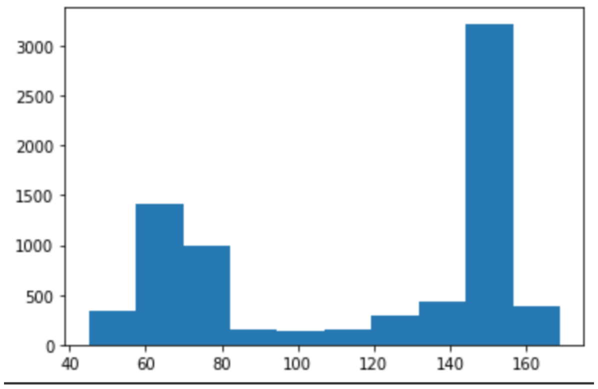

Figure 24: The histogram of pixel values in an image from the CNN License Plate Dataset. We have decided that the value of 100 is the most appropriate for thresholding and used it as default for Otsu method.

As a final step, we've implemented a classifier that assigns our extracted letter or digit a correct label. To do so, we've uses a common algorithm, in this context of k-Nearest Neighbors, which divides the data into k classes based on distance metrics. [7] We have predefined the number of neighbours to be equal to the number of categories present in data and resized the images to a common size, that would be the most similar to the data from our base dataset coming from custom segmentation (in order to possibly maximize the results from this pipeline). The model, as trained solely on the additional, Belgian license plates dataset achieved the test accuracy, as tested on 25% of their dataset, of around 95.4%. The model used for digit classification was later stored in the project repository as a pickle file under the name of ”$my_dumped_classifier.pkl$”, in order to preserve the results of model training and properly deal with a large dataset such as the CNN License Plate Dataset.

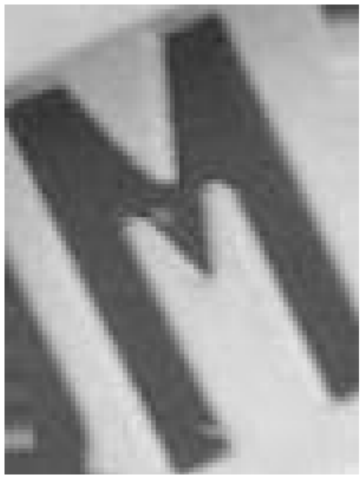

Figure 25: The image of M as placed in the Belgian CNN License Plate Dataset. One can see that the medium part of the letter stops in the middle of its height in this dataset, differing from the examples in our dataset.

Unfortunately, the data in our additional dataset differed from our base dataset not only in its lower quality and often size, but also the inclusion of, for example, different forms. Due to this fact, in some cases the model has received lower than expected results by mistaking the data with similarly shaped characters and letters. To mitigate this problem, we have dilated the letters (to make shapes be leaner and more pronounced) using OpenCV's $cv2.dilate$[16] function, in order to increase the white space on the picture. We have also inverted the colors, so that these images would be similar to training images when it comes to coloring space, and resized them according to the input from custom segmentation. In order to make a meaningful comparison we've an existing tool, we have used a python implementation of Tesseract, an open source OCR engine[17]. Possible improvement of the method may be achieved using Convolutional Neural Networks.

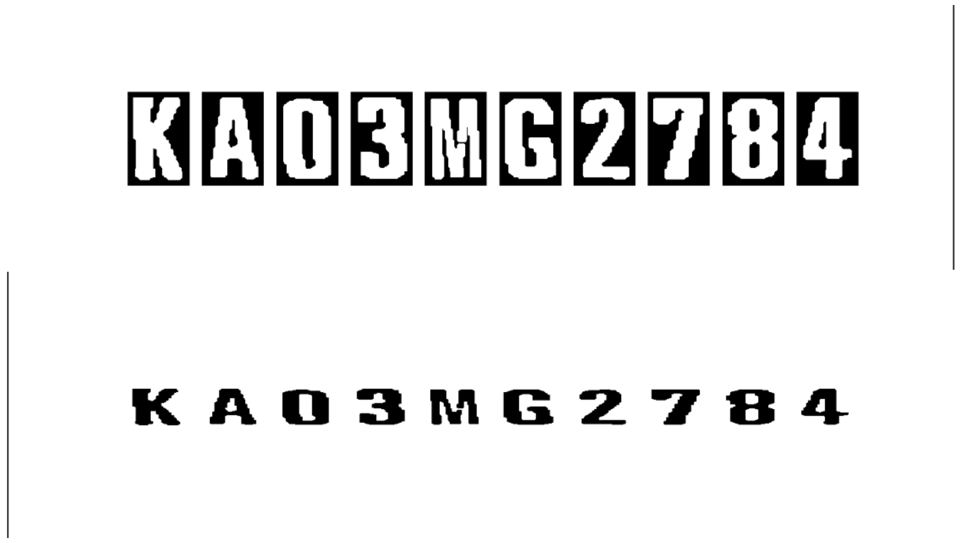

Figure 26: The license plates as segmented and later preprocessed for prediction. The Tesseract result for this example was $COF - WE)2789$, which is very different not only in terms of recognized characters, but also spacing. Concurrently, the KNN result was $KAB3N6Z181$. We can see that the falsely detected characters are often of similar shape, like G and 6 or M and N

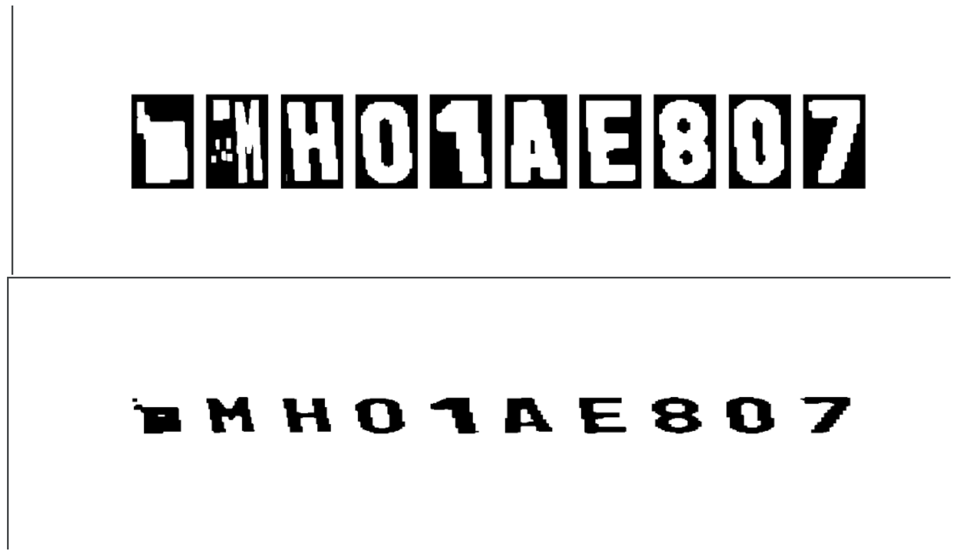

Figure 27: Here, the first character is practically unrecognizable. The Tesseract classifies it as $MAOTAESOT7$, with an additional character of T inserted at the end. Concurrently, the KNN classifier recognized it as $AMH01AE887$, which is a result very close to the actual case.

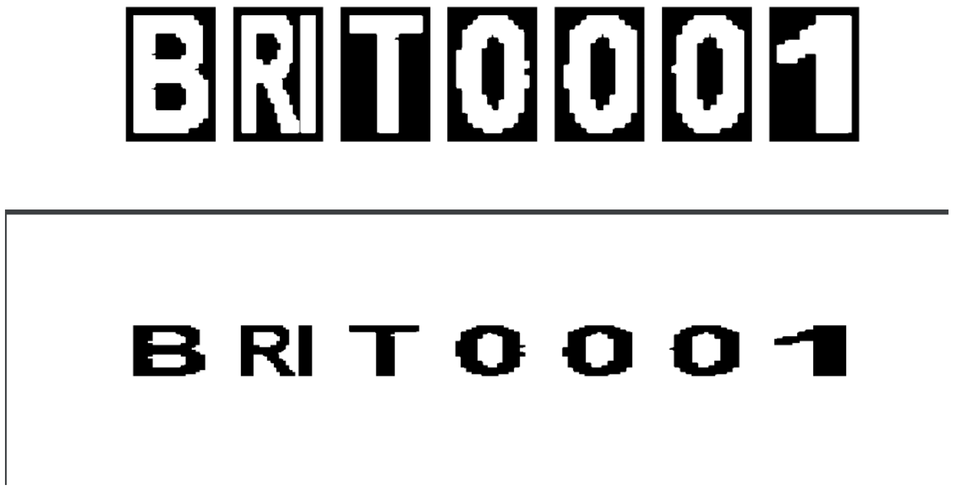

Figure 28: Previous examples demonstrated the possible advantages of segmentation before classification. Here, we have a tougher case, with an additional character I being very close to the letter R. Tesseract recognized the text as $BRilJOOT$, which is close in terms of correct placement of R and i. KNN, on the other hand, correctly classifies R with the whole text being $8RtDDD1$, struggling with differing the D shape from the 0 shape

References

[1] Xinyu Zhou, Cong Yao, He Wen, Yuzhi Wang, Shuchang Zhou, Weiran He, and Jiajun Liang. East: An efficient and accurate scene text detector, 2017.

[2] Adrian Rosebrock. OpenCV: Automatic License/Number Plate Recognition (ANPR) with Python. `https://www.pyimagesearch.com/2020/09/21/opencv-automatic-license-number-plate-recognition-anpr-with-python/`.

References

[1] Towards Data Science: Scanned digits recognition using K-Nearest-Neighbors. `https://towardsdatascience.com/scanned-digits-recognition-using-k-nearest-neighbor-k-nn-d1a1528f0dea`

[2] Craft Of Coding: Image Binarization (1): Introduction `https://craftofcoding.wordpress.com/2017/02/13/image-binarization-1-introduction/`

[3] Deep AI: Binarization Definition `https://deepai.org/machine-learning-glossary-and-terms/binarization`

[4] Analytics Vidhya: Image simplification through binarization. `https://medium.com/analytics-vidhya/image-simplification-through-binarization-in-opencv-1292d91cae12`

[5] Halina Kwaśnicka and Bartosz Wawrzyniak *License plate localization and recognition in camera pictures* `https://www.ii.pwr.edu.pl/~kwasnicka/download/kwasnickawawrzyniak.pdf`

[6] Nguyen Thai-Nghe and Nguyen Chi-Ngon *An Approach for Building an Intelligent Parking Support System*

[7] Lanna Dharma Printed Character Recognition using k-Nearest Neighbor and Conditional Random Fields `https://www.scitepress.org/papers/2012/41128/41128.pdf`

[8] OpenCV: Automatic License/Number Plate Recognition (ANPR) with Python https://www.pyimagesearch.com/2020/09/21/opencv-automatic-license-number-plate-recognition-anpr-with-python/

[9] Build an OCR System from Scratch in Python https://medium.com/lumenore/build-an-ocr-system-from-scratch-in-python-69c08e78de2

[10] Quang Nguyen, Detect and Recognize Vehicle's License Plate with Machine Learning and Python — Part 2: Plate character segmentation with OpenCV https://medium.com/@quangnhatnguyenle/detect-and-recognize-vehicles-license-plate-with-machine-learning-and-python-part-2-plate-de644de9849f

[11] Segmenting characters from license plates https://customers.pyimagesearch.com/lesson-sample-segmenting-characters-from-license-plates/

[12] Image binarization in OpenCV and Numpy https://note.nkmk.me/en/python-numpy-opencv-image-binarization/

[13] Thresholding in OpenCV https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html

[14] Thresholding in OpenCV2 https://www.pyimagesearch.com/2021/04/28/opencv-thresholding-cv2-threshold/

[15] K-Nearest-Neighbours for image classification https://www.pyimagesearch.com/2016/08/08/k-nn-classifier-for-image-classification/

[16] OpenCV - Erosion and Dilation https://www.geeksforgeeks.org/erosion-dilation-images-using-opencv-python/

[17] Tesseract Repository https://github.com/tesseract-ocr/tesseract