	Instytut Informatyki Politechniki Śląskiej Zespół Mikroinformatyki i Teorii Automatów Cyfrowych			
		Subject	Group	Section
2017/2018	**SSI**	**BIAI**	**PSI**	**1**
Name:	**Jakub**	**Leading subject:**	**GB**	
Surname:	**Czachor**			

Final Raport

Topic of the project:

Registration Number Recognizer

Date: **dd/mm/yyyy**	30/09/2018

Table of Contents

Topic, problem examination and main assumptions of the program

The project's topic is an application which searches for a car license plate on a photo and then recognises it's registration number (at the end user can copy the registration number as a string from generated textbox).

An application provides user graphic interface with loaded image preview, separated license plate, information about loaded image resolution, few additional windows with succeeding stages of photo editing, textbox with recognized registration number and buttons to start teaching the program and start editing the image.

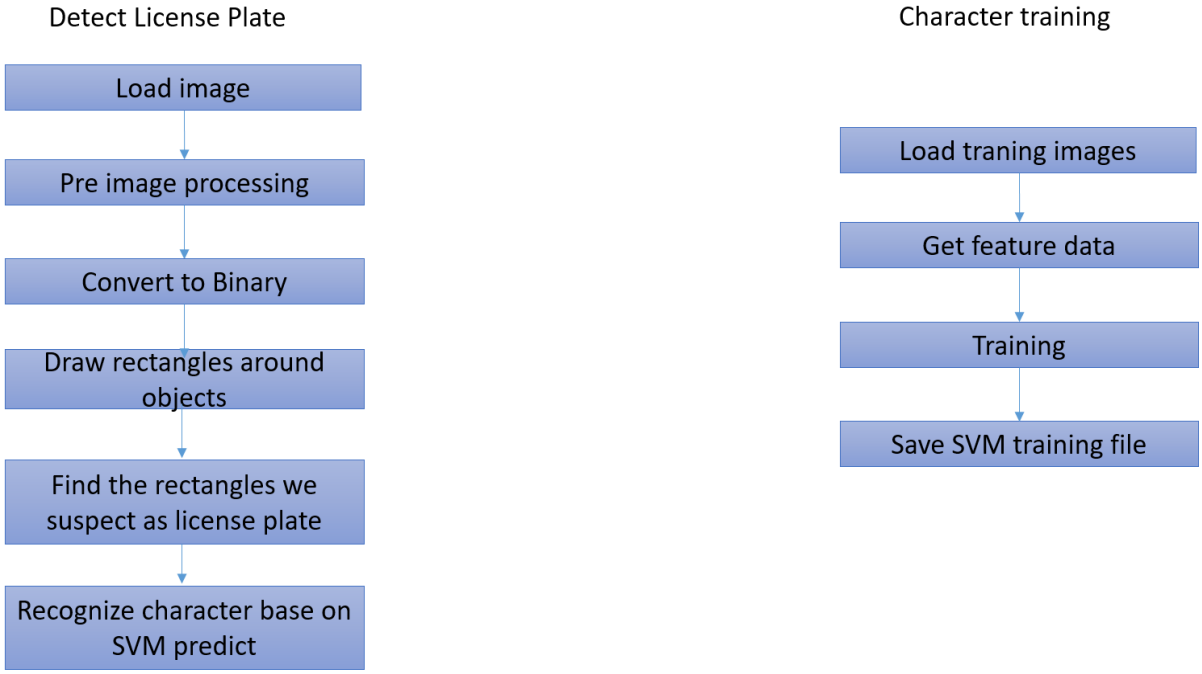
To detect licence plate and registration number, application uses openCV framework which is an open source computer vision and machine learning software library, free for academic and commercial use. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms.

Recognizing licence plate's registration number is done by using openCV Support Vector Machine which is discriminative classifier formally defined by a separating hyperlane. In other words, given laveled training data (supervised learning), the algorithm outputs an optimal hyperlane which categorizes new examples.

User's graphic interface is made with Windows Forms framework which gives access to Windows system components such as buttons, forms, labels, textboxes etc. in the form of dragging and dropping further components.

The source code is written in C++ language in Visual Studio 15 IDE.

Architecture and algorithms

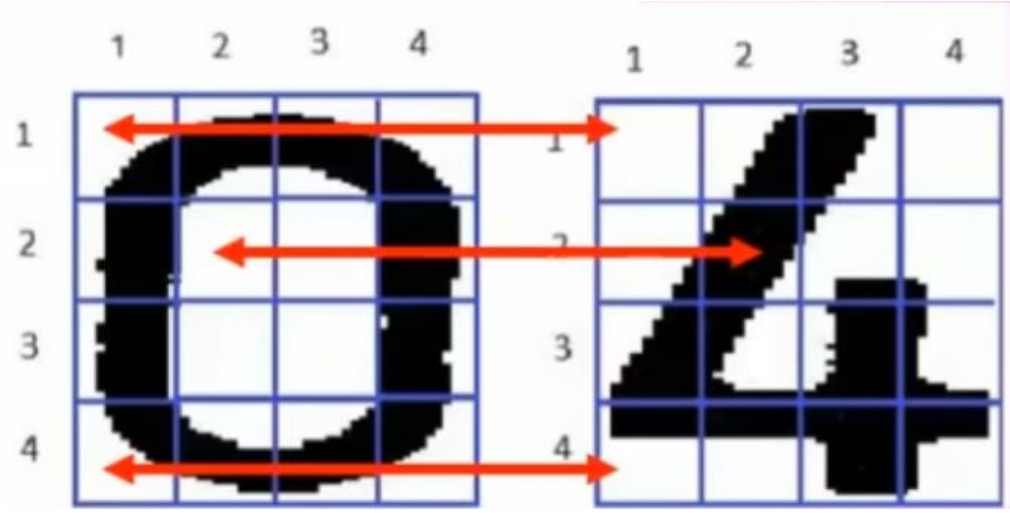


Picture 1. Structure of the program.

To detect characters in the license plate, there was necessity to find and calculate feature for each sample of each alphanumeric character. I've noticed that darkness level of individual sections of the photo can define uniqueness feature to determine differences between characters and numbers.

What my program does to determine feature, is editing every sample to its bitmap form, resizing to 40x40 pixels, dividing resized sample into 16 equal regions and counting black pixels on each part. After that, number of black pixels on the part of a sample is divided by number of all black pixels on a sample. After counting feature for each of 16 parts of the sample, I'm calculating another 16 features by summing given values in random order (set by me), then I'm saving it one by one in the container, which represents features of one sample of any character as a primitive, multidimensional points in the hyperplane. After calculating all the features for all samples, the values determined like this become a feature that (in this case) describes the distribution of all the black pixels in this alphanumeric character.

When all features of all characters are determined, the collected data container is passed to Support Vector Machine, where multidimensional support vectors are calculated. Support vectors are used as classifiers to distinguish between characters.



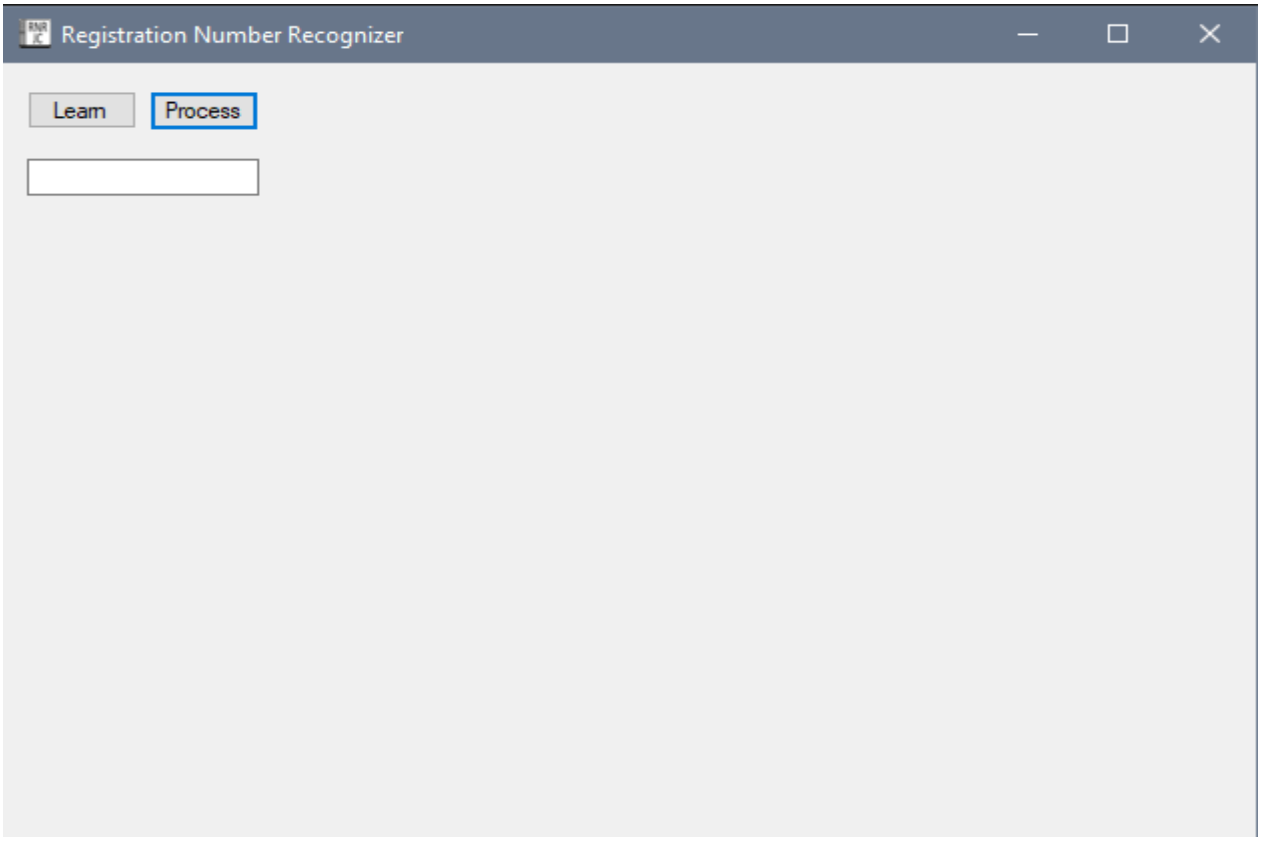
Picture 2. Simplified principle of looking for differences between two characters.

In my program, I'm using 36 classes of alphanumeric characters (0-9, A-Z), 10 samples each, where one sample is described by 32 features. Total training time for this set is about 4 minutes. I've tried set of handwritten characters (400 samples each), but after work results, with uppercase straight letters on license plate, were very poor. It is possible, that some of the morphological transformations are able to remove noises from handwritten characters.

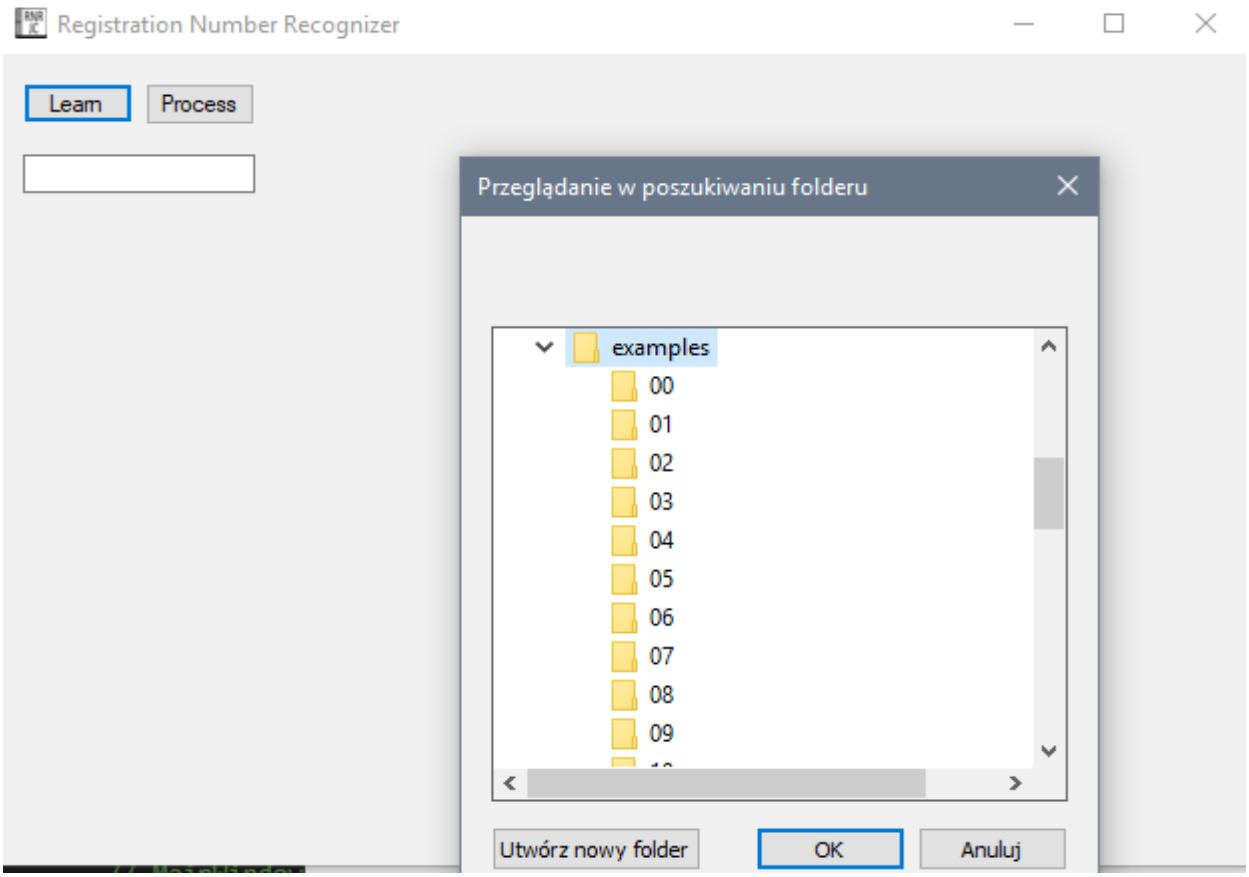
To recognize license plate, application edits image by changing its resolution (if it is over 3000pixels width and 4000pixels height), converting image to shades of grey and then to binary, noises created that way are removed by morphological transformations such as erode (pixels neighbour to more black than white pixels become black too) and dilate (pixels neighbour to more white than black pixels become white too). After that, picture is ready to search contours on it. Contours are filtered in a way that if they have the same proportions of sides as license plate (520mm/114mm) or are not bigger than half of the photo width and height or are not smaller than 120x40 pixels then they may provide us to find real license plate so we start searching for characters inside it, so again we search for contours inside this separated fragment of the picture and contours that are bigger than height divided by 2 and width divided by 7 (because I assumed that on the Polish license plate there are always 7 characters) of the fragment of the picture divided by 2. If it matches, it's possible that we have correct license plate. After gathering necessary data contours, program sends fragments of license plate outlined by them to the function which recognises letters. These function counts features of each character like in training part of the program, loads SVM file with all possible support vectors and then starts recognising characters using SVM's function – predict. Depending on the order of the transmitted data to training SVM, recognized character is returned as an integer value which is like ID. In my case (as I passed data in alphabetic order), I'm treating ID as an offset which I add to constant value to get ASCII code at the end, which I convert back to number or character and send it to textbox.

Training

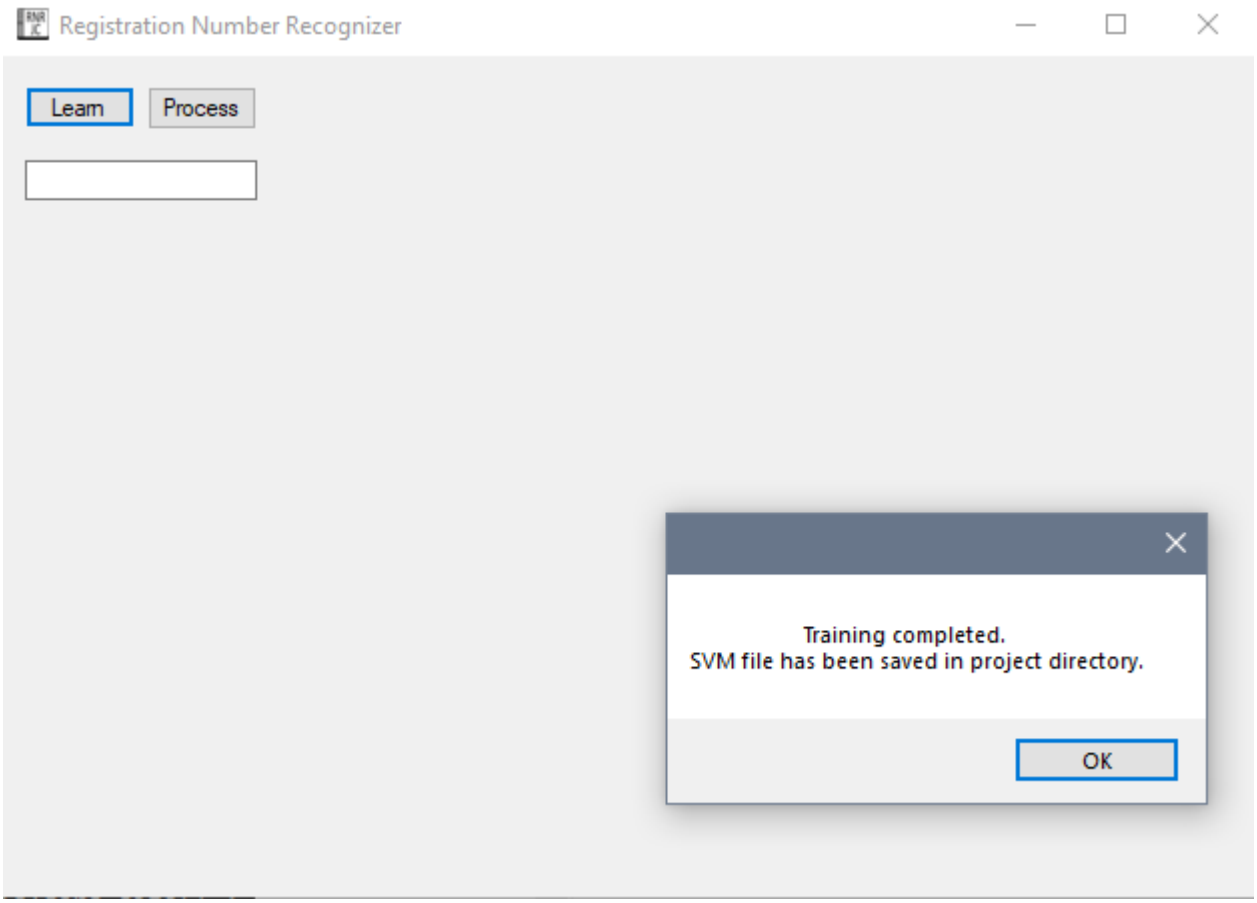
For training purposes I've extracted manually letters from license plates and put them in sets of 10 samples per class in training data folder.
To use them, user must follow these steps:



Picture 3. Run program.



Picture 4. Click Learn button and select location of training examples.



Picture 5. Training data has been saved in project directory (default folder is SVM).

Example part of training data looks like this:

```
%YAML:1.0
---
opencv_ml_svm:
   format: 3
   svmType: C_SVC
   kernel:
      type: RBF
      gamma: 5.0625000000000009e-01
   C: 6.2500000000000000e+01
   term_criteria: { iterations:100 }
   var_count: 64
   class_count: 36
   class_labels: !!opencv-matrix
      rows: 36
      cols: 1
      dt: i
      data: [ 0, 1, 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 ]
   sv_total: 271
   support_vectors:
      - [ 0., 1.91387553e-02, 2.87081338e-02, 2.39234455e-02,
          2.55183410e-02, 3.18979248e-02, 1.43540669e-02, 0.,
          1.27591705e-02, 3.98724079e-02, 3.34928222e-02, 3.18979248e-02,
          3.18979248e-02, 3.18979248e-02, 3.82775106e-02, 1.11642741e-02,
          2.71132383e-02, 1.11642741e-02, 0., 0., 0., 0., 1.43540669e-02,
          2.71132383e-02, 3.18979248e-02, 0., 0., 0., 0., 0., 0.,
```

3.18979248e-02, 3.18979248e-02, 0., 0., 0., 0., 0.,
1.59489631e-03, 3.18979248e-02, 2.71132383e-02, 9.56937764e-03,
0., 0., 0., 0., 1.43540669e-02, 2.71132383e-02, 1.11642741e-02,
3.66826169e-02, 2.87081338e-02, 2.39234455e-02, 3.03030312e-02,
3.18979248e-02, 3.82775106e-02, 7.97448121e-03, 0.,
1.43540669e-02, 3.03030312e-02, 2.39234455e-02, 2.39234455e-02,
2.39234455e-02, 1.27591705e-02, 0.]
- [0., 7.16845877e-03, 1.79211479e-02, 2.50896066e-02,
2.68817209e-02, 2.68817209e-02, 1.25448033e-02, 0.,
5.37634408e-03, 4.12186384e-02, 3.58422957e-02, 3.58422957e-02,
3.58422957e-02, 3.58422957e-02, 4.12186384e-02, 5.37634408e-03,
1.79211479e-02, 2.15053763e-02, 0., 0., 0., 0., 1.61290318e-02,
2.15053763e-02, 3.04659493e-02, 3.58422939e-03, 0., 0., 0., 0.,
5.37634408e-03, 3.04659493e-02, 3.04659493e-02, 3.58422939e-03,
0., 0., 0., 0., 7.16845877e-03, 3.04659493e-02, 1.97132621e-02,
1.61290318e-02, 0., 0., 0., 0., 2.50896066e-02, 1.79211479e-02,
8.96057393e-03, 4.48028669e-02, 3.58422957e-02, 4.12186384e-02,
4.48028669e-02, 4.48028669e-02, 3.94265242e-02, 1.79211469e-03,
3.58422939e-03, 7.16845877e-03, 1.97132621e-02, 1.79211479e-02,
1.79211479e-02, 1.79211479e-02, 3.58422939e-03, 0.]
decision_functions:
-
sv_count: 5
rho: -7.7129583783021359e-03
alpha: [1.6648814222845878e+01, 6.2500000000000000e+01,
1.4185594393366499e+01, -3.0834408616212375e+01,
-6.2500000000000000e+01]
index: [6, 7, 9, 13, 14]
-
sv_count: 7
rho: -3.0005289097143180e-02
alpha: [4.1814150802379920e+01, 1.6387057832073999e+01,
5.8018452491980959e+01, -3.6936880462974074e+01,
-4.5058544375909740e+01, -7.7113074093852045e+00,
-2.6512928878165884e+01]
index: [6, 7, 9, 16, 18, 19, 20]

At the beginning there are informations about set options:

- **svm_type** - type of a SVM formulation. Possible values are**: CvSVM::C_SVC**, CvSVM::NU_SVC, CvSVM::ONE_CLASS, CvSVM::EPS_SVR, CvSVM::NU_SVR**;**
- **kernel_type** - type of a SVM kernel. Possible values are: CvSVM::LINEAR, CvSVM::POLY, **CvSVM::RBF**, CvSVM::SIGMOID**;**
- **gamma -** parameter γ of a kernel function (POLY / RBF / SIGMOID);
- **Cvalue -** parameter c of a SVM optimization problem (C_SVC / EPS_SVR / NU_SVR);
- **term_crit** - termination criteria of the iterative SVM training procedure which solves a partial case of constrained quadratic optimization problem.

External specification

- Starting application:

  To run application, user may use executable file located in program directory or run it by itself using cmd command: program\directory> GUI + CALCULATIONS.exe. Application doesn't require any command argument.

– Input data:

Application supports common extensions such as jpg, png, and many others. User may locate pictures when program is working. There are no constrains about resolution, file size, color palette etc. but those parameters may affect the processing time. To prevent that, when resolution is too high, program changes it to optimal. In addition, original image files will not be modified. Moreover, user does not have to configure application at all – everything is automated.

Internal specification

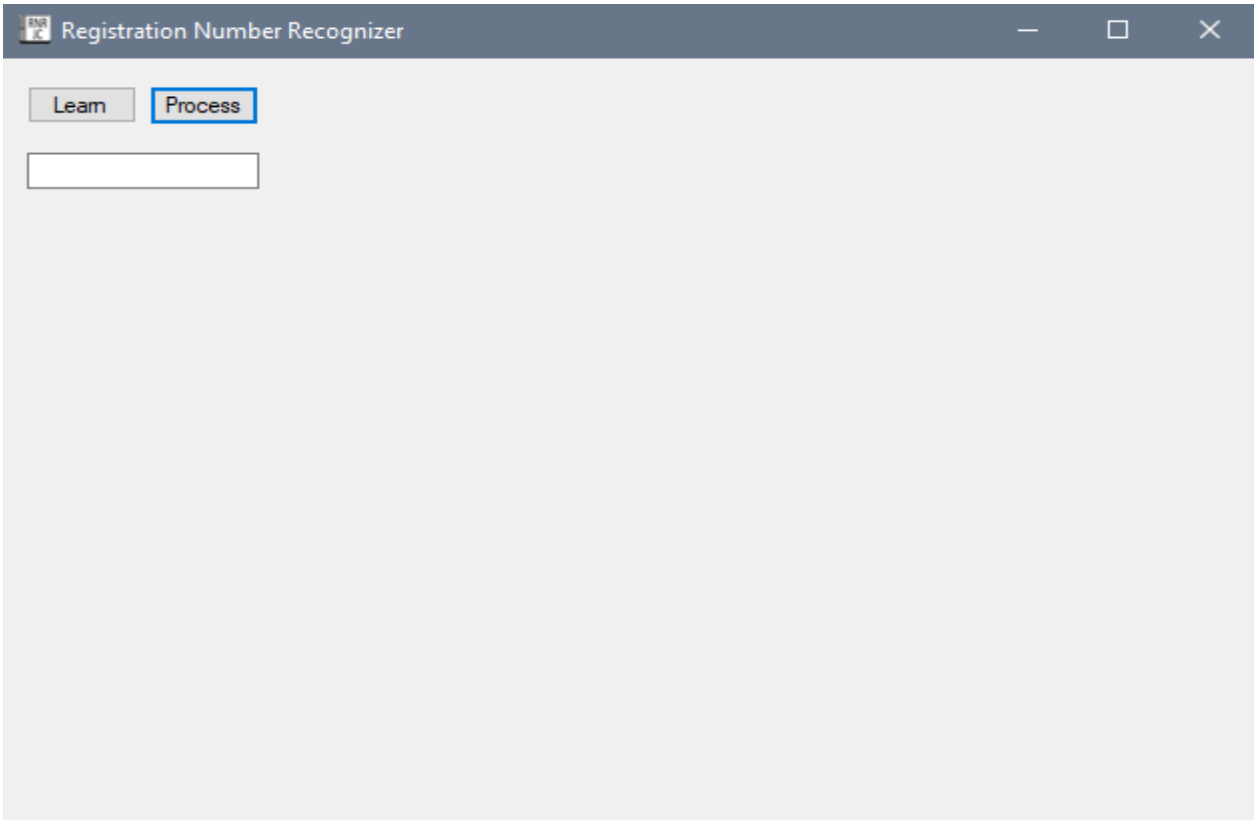
1. Common variables:
    a. `classes – global, constant, integer variable which represents number of different classes (characters 0-9, A-Z);`
    b. `samples – global, constant, integer variable which represents number of different samples inside a class folder;`
    c. `numberOfFeature – global, constant, integer variable which represents number of different, alphanumeric character's features to calculate;`
    d. `src – variable which represents loaded picture showed in GUI as a matrix;`
    e. `mat2bmp – object of class Matrix2Bitmap which provides method converting matrix image to bitmap;`

2. GUI variables:
    a. ptbSource – object of PictureBox class which provides area to display loaded picture in GUI;
    b. btnProcess – object of Button class which activates operation of processing;
    c. btnLearn – object of Button class which activates machine training;
    d. textbox – object of TextBox class to paste recognized registration number (where you can copy it as a string of characters);
    e. pictureBox1 – object of PictureBox class to display binary image of license plate;
    f. pircuteBox2 – object of PictureBox class to display image of license plate with recognized characters on it;
    g. label1 – object of Label class to display resolution of the loaded picture;

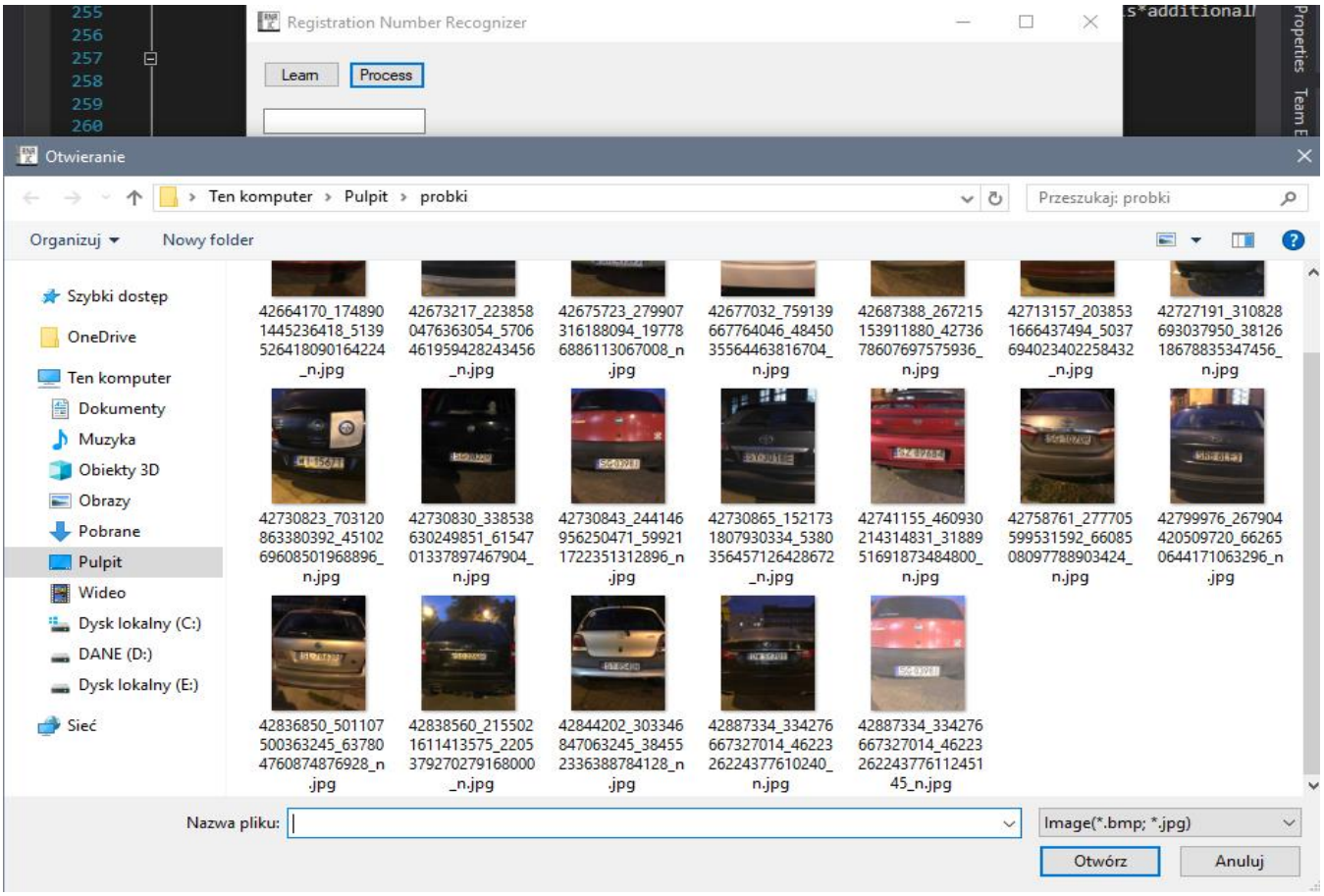
3. Important methods:
    a. getContentPaths - function which gets a path to the training data root directory and returns vector of individual paths to each folder containing samples;
    b. countPixels – function which gets part of a sample photo as a matrix as an argument and returns number of black pixels on it;
    c. calculateFeature – function which gets loaded picture matrix as an argument and returns vector of calculated features;
    d. countSamples – function which counts samples inside root directory;
    e. findContours – function which saves returns contours found on a passed as an argument binary image (loaded or found plate), described by points in a vector of vectors of points;
    f. convertToBinary – function which converts 3-canal image to grey and then to binary (and returns its copy);
    g. training – function that uses calcuateFeature for all classes, then trains and generates SVM file;
    h. characterRecognition – function that loads SVM file, calculate features for passed as an argument, matrix image of possible character and after retrieving id, calculates ASCII code of a character, which returns at the end;
    i. recognize – function which uses other functions to operate on data. This function returns true if license plate and characters were recognized or false if not;

User manual

Application provides simple graphic interface. At the beginning, program waits for user to click Learn or Process button. In this section I will describe “Process” function. “Learn” option is described in “Training” section. There is no limit for user to use the “Process” option.



Picture 6. Run program and click Process button.



Picture 7. Select any photo and click Open.

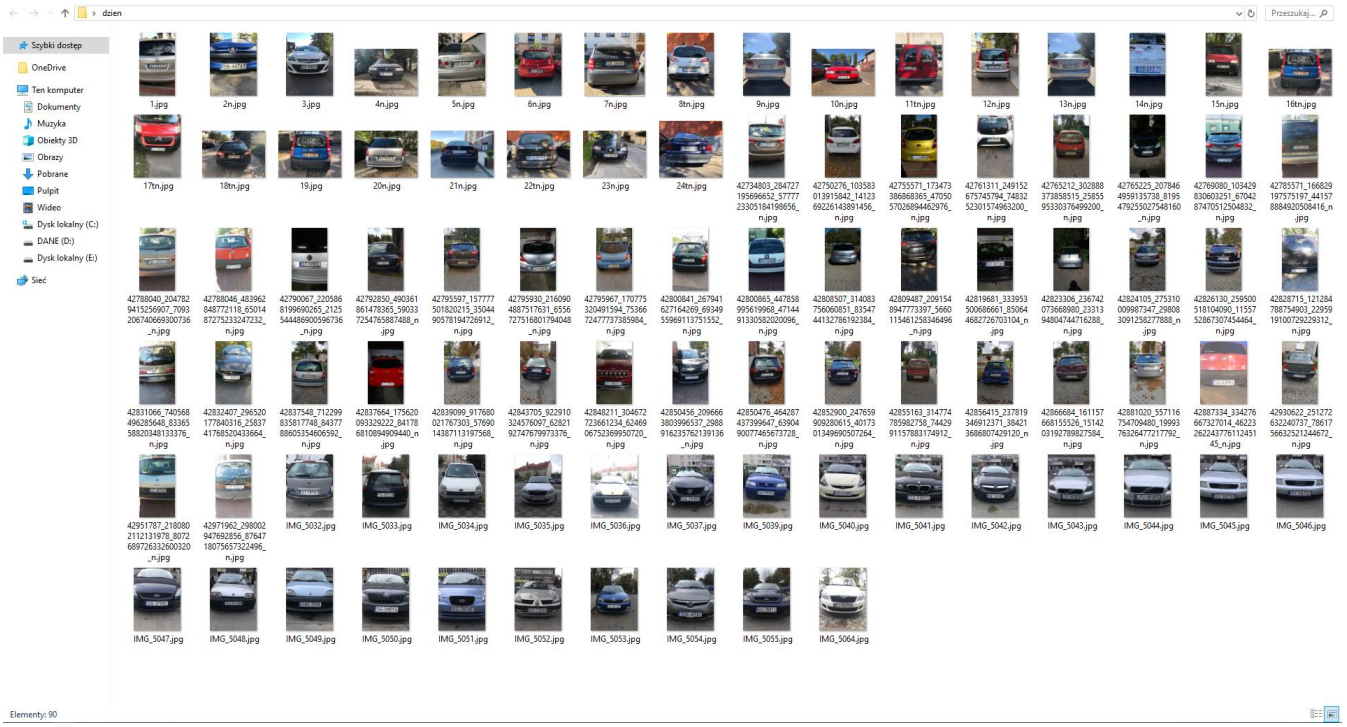
Picture 8. Program returns images and recognized plate. Additional windows are only to let user see image processing at the different stages.

There is no limit for user to use the “Process” option.

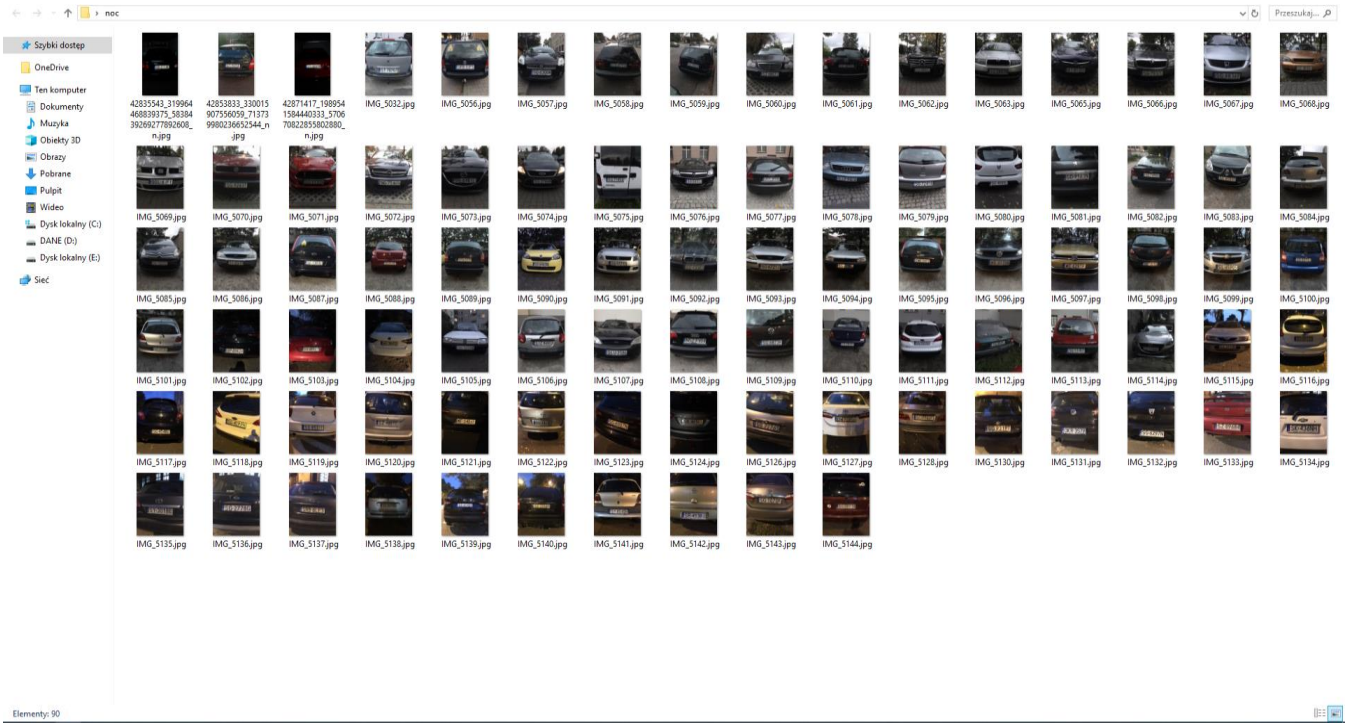
Testing

For testing, I made 90 photos during the day (high brightness) and 90 photos at night (low light) and decided that I’ll make 3 different tests to measure percentage recognition to find answers to questions such as:

1) Which conditions are better for recognizing: low light or high brightness?
2) What impact on the results has the use of 32 features or 16 features?
3) What are and what impact on the results has the use of different methods to find contours (CV_RETR_EXTERNAL, CV_RETR_LIST, CV_RETR_CCOMP, CV_RETR_TREE)

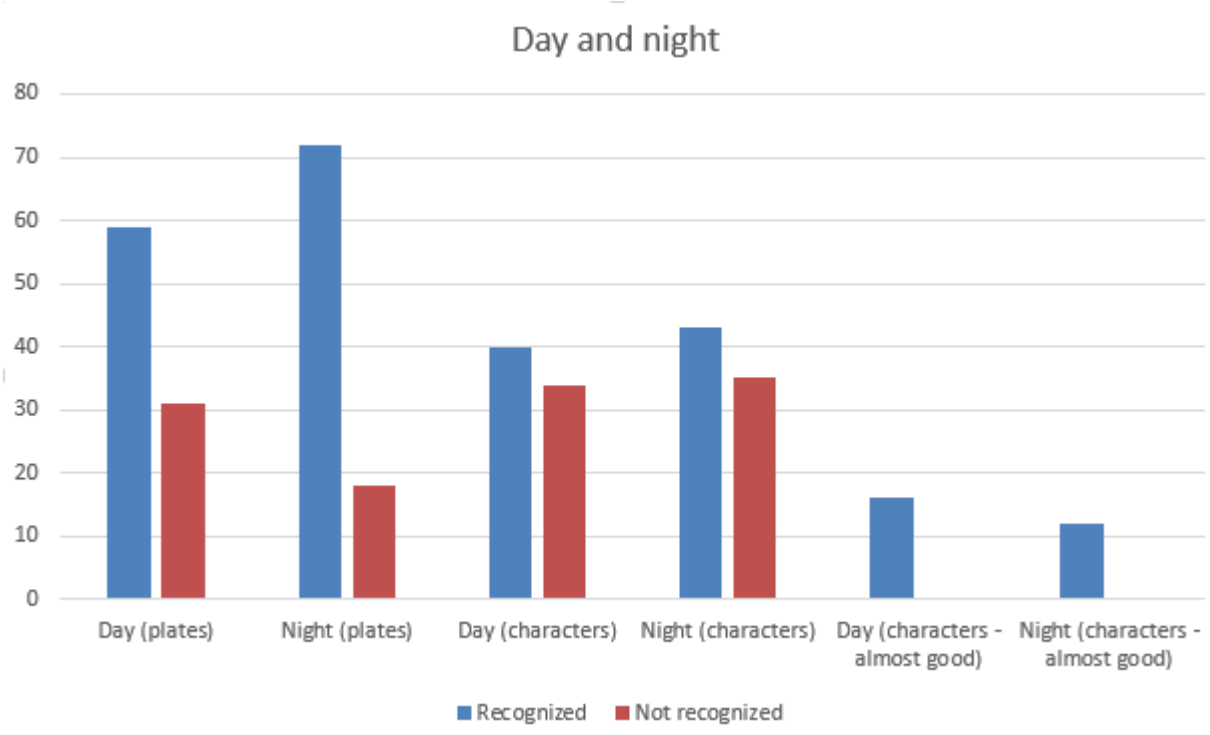


Picture 9. Gallery of test photos taken during the daylight.

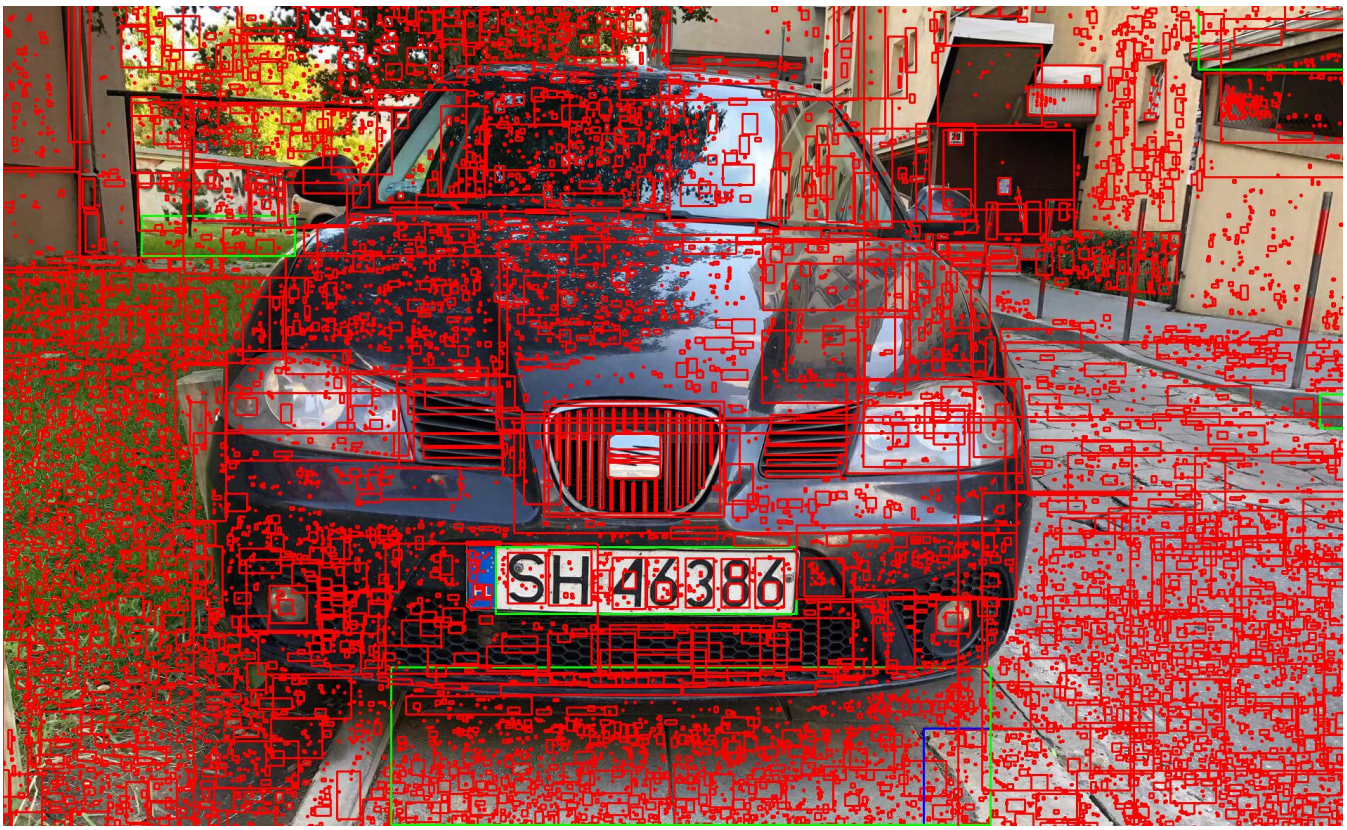


Picture 10. Gallery of test photos taken during the night.

Test No. 1



Picture 11. Results diagram.

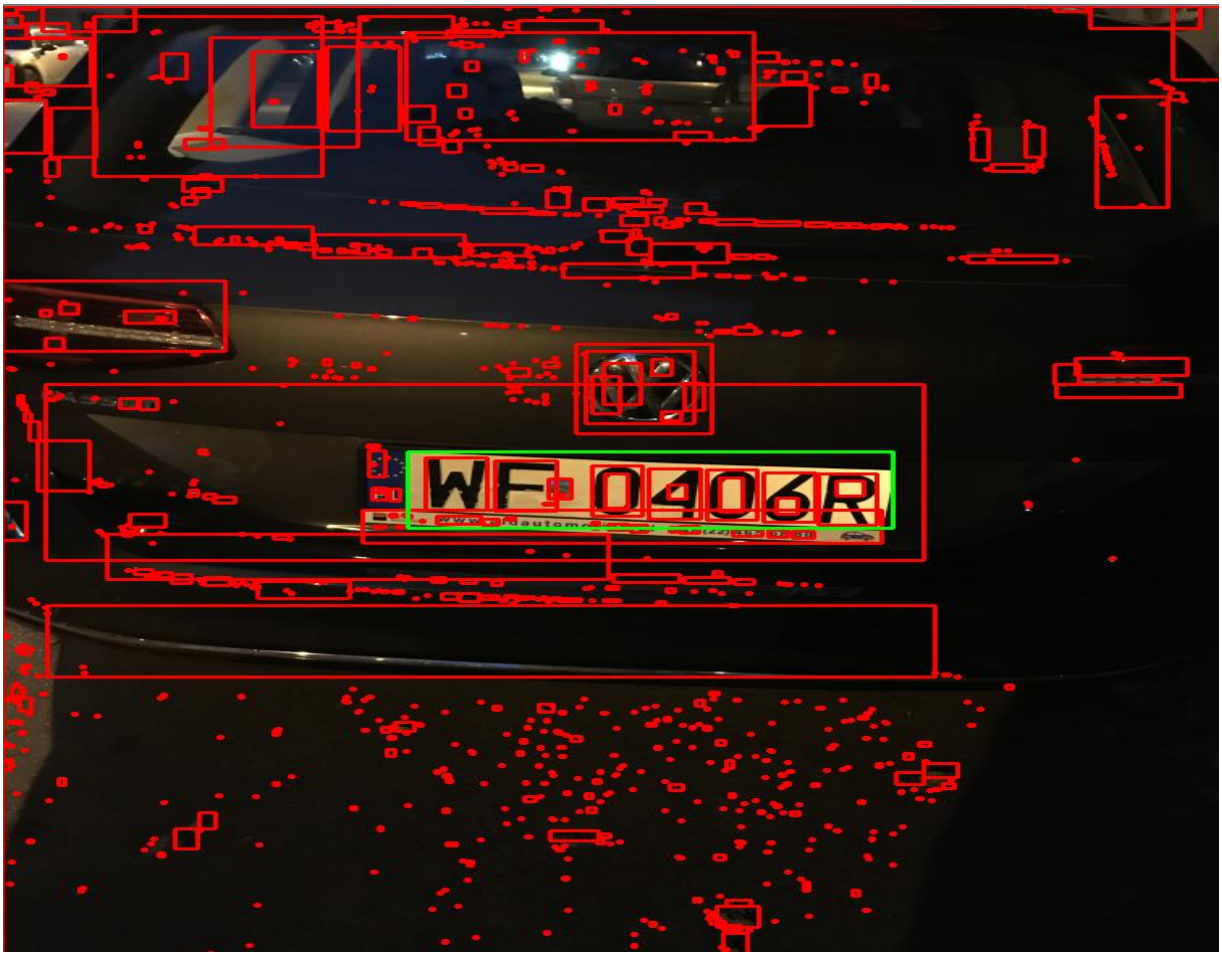


Picture 12. The picture taken during the day has tons of noises caused by reflecting light rays.

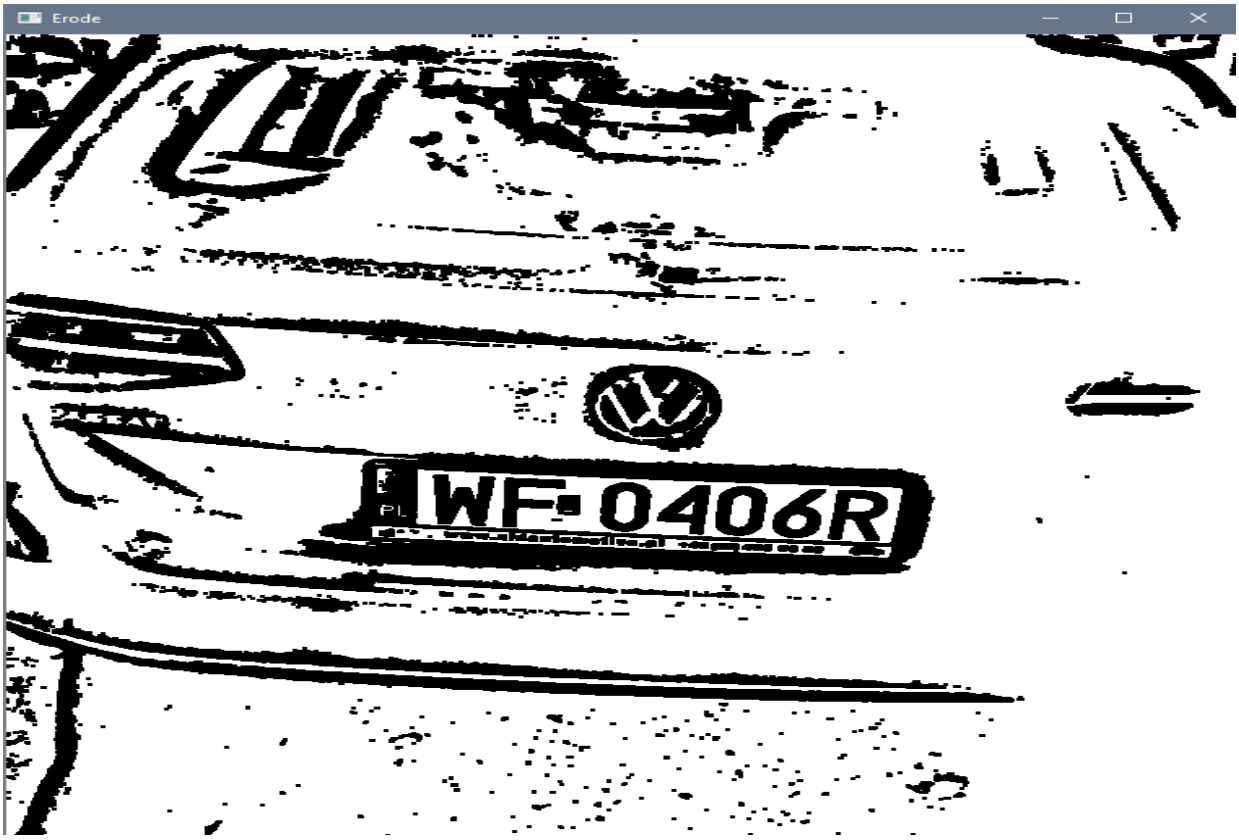
Picture 13. Binary picture taken during the day after erode effect.

Picture 14. Binary picture taken during the day after dilate effect.

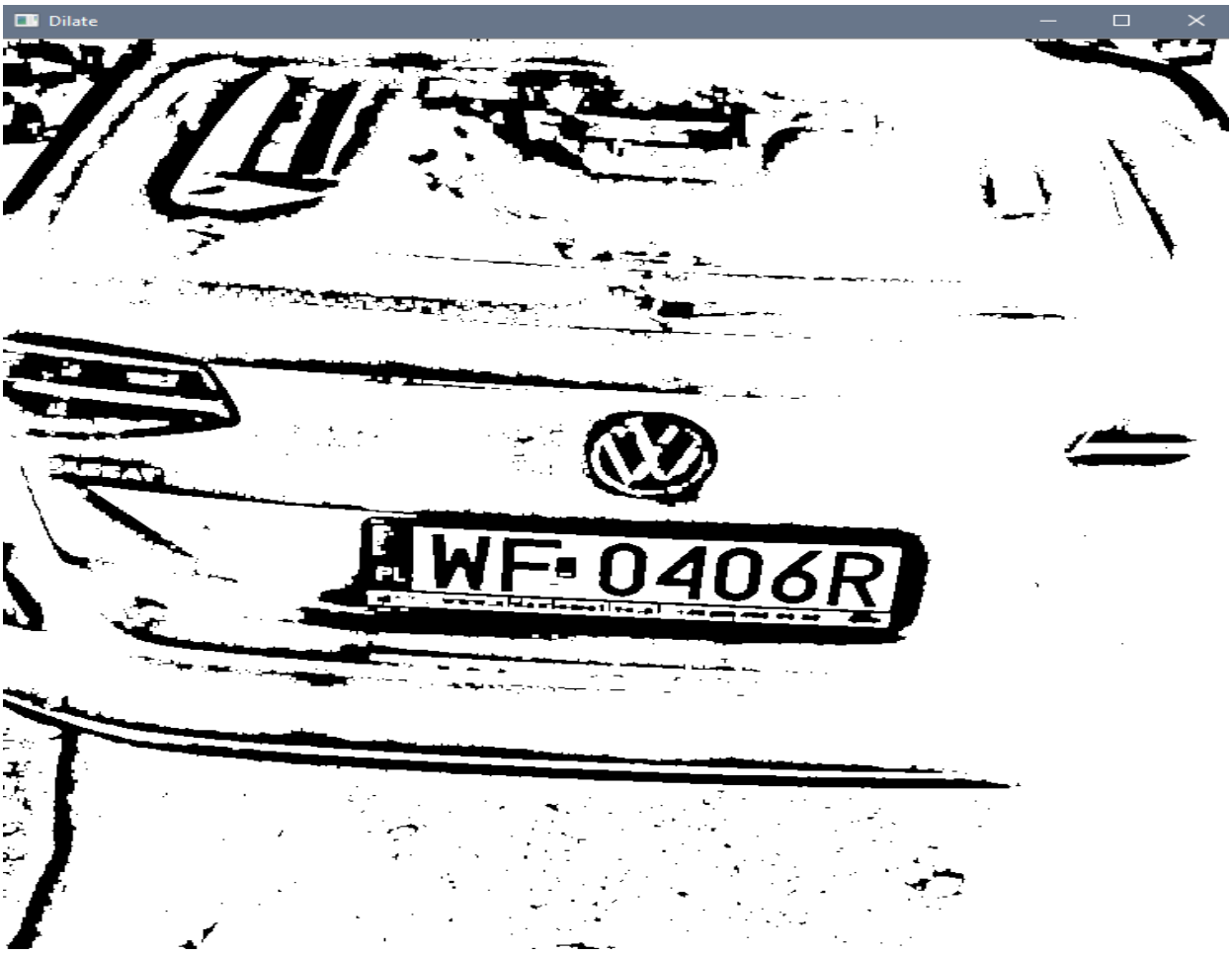
As we can see, during the daylight there is a lot of going on in the picture, so it is not surprising that the processing of such a photo takes almost 30 seconds. Meanwhile processing time of image down below (night photo) took 0.8s.



Picture 15. Contours with marked license plate on the night photo.

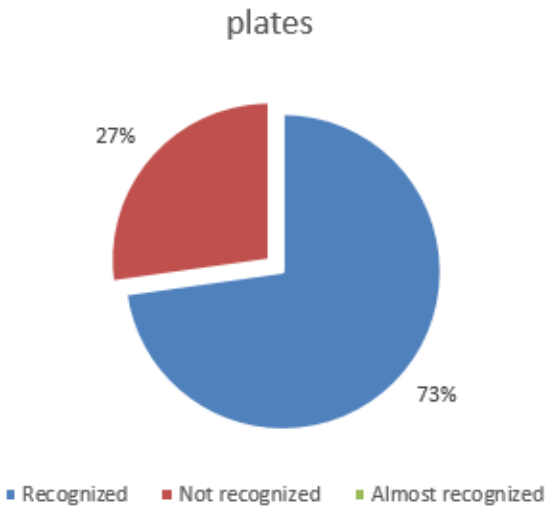


Picture 16. Binary picture taken during the night after erode effect.

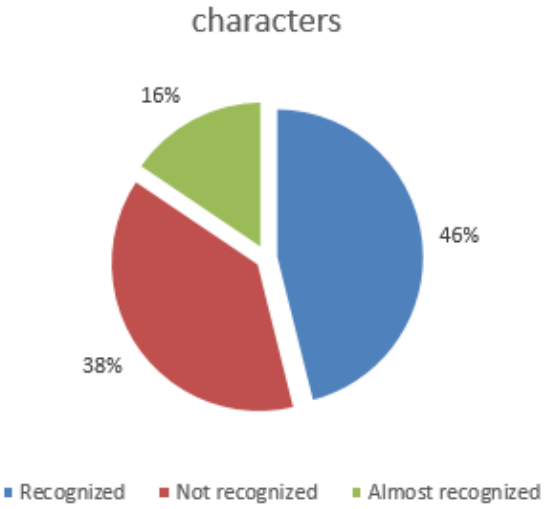


Picture 17. Binary picture taken during the night after dilate effect.

As we can see, daylight photos has some problems too, but it’s not like they are worse, it just depends on situation. The other causes of this result may be too high or too low resolution of an image or bad angle from where the photo was shot.

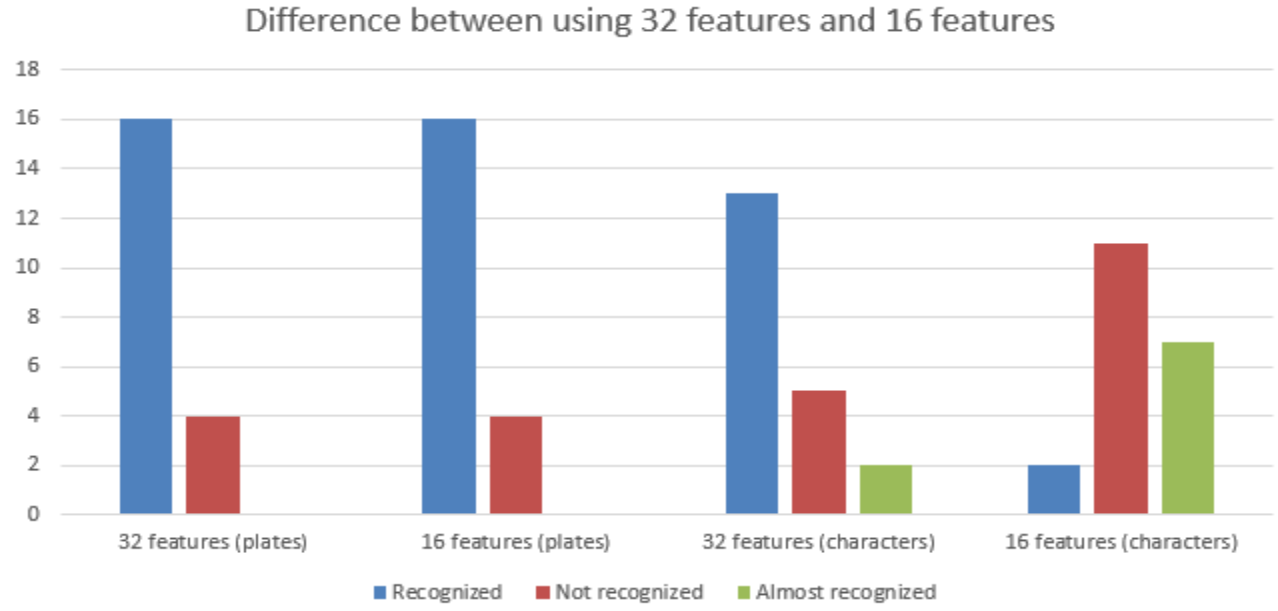


Picture 18. The total result of the number plate detection.



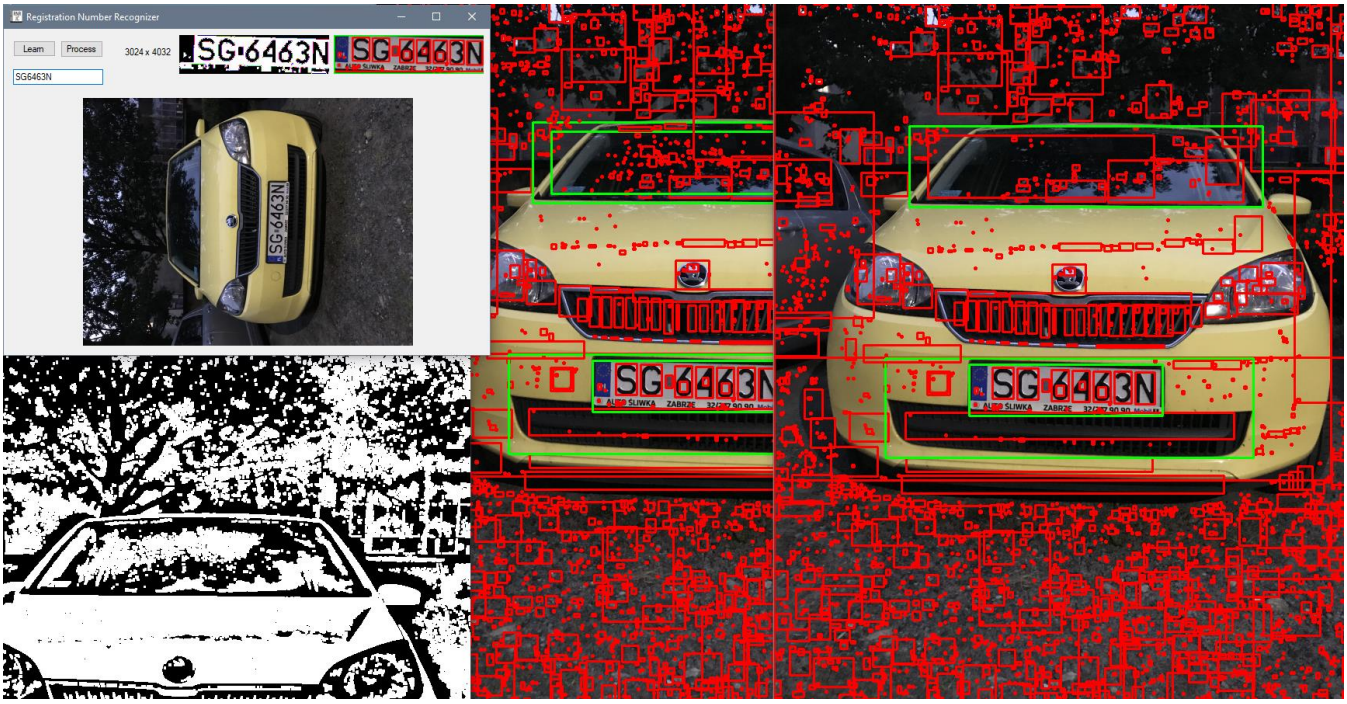
Picture 19. Total result of the characters recognition. Almost recognized characters set means that 1 or 2 letters was wrong in a way that they looked similar to the right ones.

Test No. 2

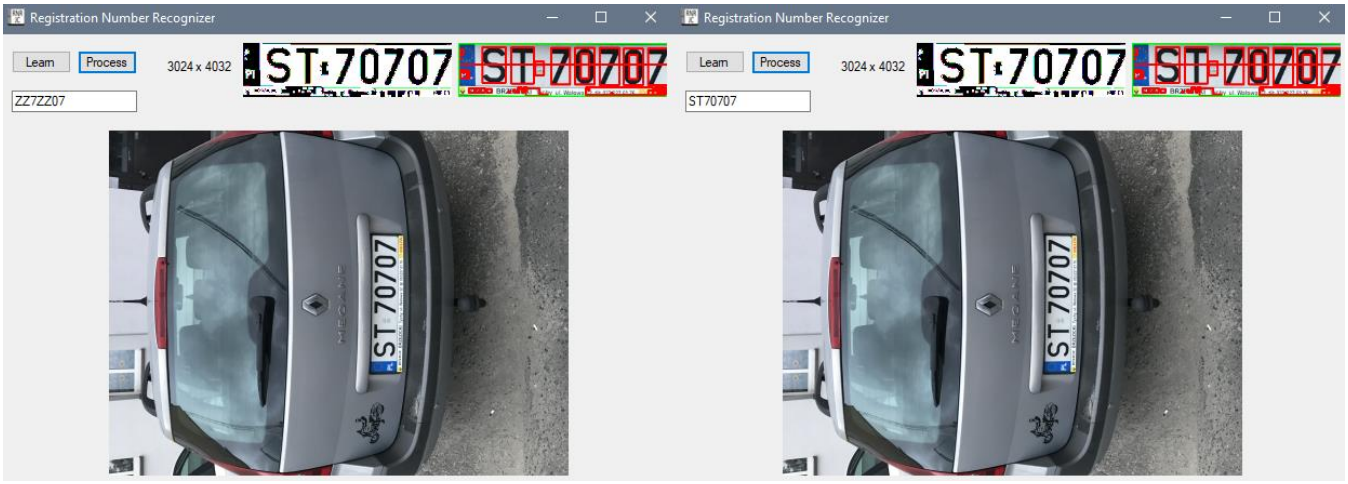


Picture 20. Results of using 32 features mode and 16 features mode on set of 20 pictures.

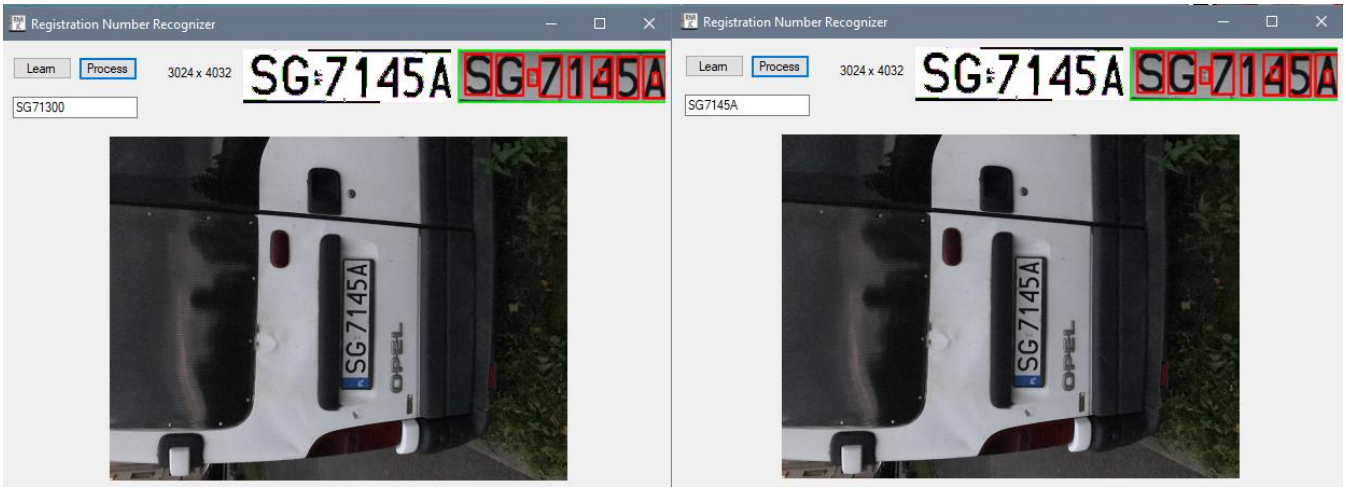
Picture 21. Recognizing characters described using 16 features.



Picture 22. Recognizing characters described using 32 features.



Picture 23. Recognizing characters described using 16 features (left) and 32 features (right).



Picture 24. Recognizing characters described using 16 features (left) and 32 features (right).

Tests prove that if we cut half of features describing characters, it may cause glitches. The more special features we have, the more accurate the recognition becomes.

Test No. 3

In OpenCV there are four contour retrieval models that are distinguished:

- **CV_RETR_EXTERNAL** retrieves only the extreme outer contours
- **CV_RETR_LIST** retrieves all of the contours without establishing any hierarchical relationships.
- **CV_RETR_CCOMP** retrieves all of the contours and organizes them into a two-level hierarchy. At the top level, there are external boundaries of the components. At the second level, there are boundaries of the holes. If there is another contour inside a hole of a connected component, it is still put at the top level.
- **CV_RETR_TREE** retrieves all of the contours and reconstructs a full hierarchy of nested contours.

Picture 25. Retrieving contours using method (from left) CV_RETR_EXTERNAL, CV_RETR_LIST, CV_RETR_CCOMP and CV_RETR_TREE.

Picture 26. Retrieving contours using method (from left) CV_RETR_EXTERNAL, CV_RETR_LIST, CV_RETR_CCOMP and CV_RETR_TREE.

Picture 27. Retrieving contours using method (from left) CV_RETR_EXTERNAL, CV_RETR_LIST, CV_RETR_CCOMP and CV_RETR_TREE.

Picture 28. Retrieving contours using method (from left) CV_RETR_EXTERNAL, CV_RETR_LIST, CV_RETR_CCOMP and CV_RETR_TREE.

Each method works only on binary picture (upper photos are original images merged with retrieved contours), but even they all seem similar, every is used in different way:

- CV_RETR_EXTERNAL after searching retrieves only extreme outer contours. It may be used to determine big shape, for example whole cars ignoring the mirrors, windows etc.
- CV_RETR_LIST searches for every contour on a photo and puts them in the list. This mode should be used when hierarchy of contours is needn't.

- CV_RETR_CCOMP retrieves all the contours and organizes them into a two-level hierarchy. It may be used to express two layers for example making one layer from a green screen and second from a person in front of it.
- CV_RETR_TREE is used to retrieve all the contours with information which contour (layer) was on top of another. It's an advanced method used to express depth in the photo.

When hierarchy isn't specified, CV_RETR_CCOMP and CV_RETR_TREE works just like CV_RETR_LIST but without putting contour in the list, that's why last tree photos looks the same and license plate was recognized with its number.

Conclusions

Image analyse is very complex and time-consuming process that needs high computing power. Nowadays, there are a lot of frameworks that makes that process easier. One of the most important is OpenCV that supports multi-language such as C++, Java and Python. It provides features such as gesture recognition, motion understanding, object identification, motion tracking and many more.

These tools allows us to train computer to detect any object we want. The only things we need are good training dataset and time. It is advised to have more than 360 samples to teach, but more doesn't mean better because resolution matters a lot.

In my project, biggest problem was to find out how to process image, what modifications would cause best results, how to remove noises, find out proper algorithm to calculate samples' features, debug program with over several thousands of possible plates and finding out why it doesn't work.

After all I feel like OpenCV framework has huge potential and capabilities of live video editing caught my attention. In nearest future I'm going to check TensorFlow framework too which is open source software library for high performance numerical computation. It has strong support for machine learning, deep learning and excellent multi-threading support that significantly reduces calculation time.

Bibliography

- OpenCV documentation
- Opencv.org/about.html
- Chars74k (training data set): ee.surrey.ac.uk/CVSSP/demos/chars74k/
- docs.opencv.org/2.4/doc/tutorials
- licenseplatemania.com/landenframes/engeland_fr.htm

GitHub Repository

Project can be downloaded from following GitHub repository:

https://github.com/jakczo/First-Neural-Network