Automatic number-plate recognition with optical character re-cognition (OCR) on images to read vehicle registration plates.

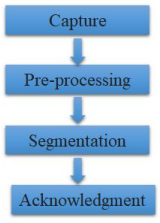
Juan Camilo Ordoñez Arbelaez, Julian Gonzalez Cifuentes
Valle Del Cauca.,
Universidad del valle
762022
orarjuan@hotmail.com, jugoci1106@gmail.com

Abstract- The following document describes the implementation of the abstraction of the characters of the image of a car plate, using for the learning the algorithm "vector support machine (SVM)", resulting in the ability to recognize each letter of the entered image. In this project report we explore the methods to detect number plate in a frame using machine learning methods. We first use some basic image processing techniques (which uses some properties of number plate like white background etc.) to filter out possible objects for number plate and then use trained model to detect number plates among them.

Keywords: License plate, Computer Vision, Pattern Recognition, Python, OCR

I. Introduction

Artificial vision systems Artificial vision is a branch of artificial intelligence whose purpose is to design computer systems capable of "understanding" the elements and characteristics of a scene or image of the real world.
These systems allow extracting information - numerical and symbolic - from the recognition of objects and structures present in the image. To achieve this, they carry out four main activities:



Acquisition of images, through a photographic camera or other sensor.

Treatment of images to highlight possible regions of interest and eliminate non-relevant information.

Ailing of the regions of interest for further analysis and interpretation

Identification and classification of objects and structures in the regions of interest.

Figure 1 - Artificial vision system

ALPR systems

Automatic license plate recognition systems - ALPR (Automatic License Plate Recognition) - are a particular case of artificial vision systems. These systems are designed to "read" the contents of vehicle license plates from the images captured by a camera, and are used mainly in surveillance and traffic control devices.

In these systems the recognition of the images of the numbers and letters of the license plates can be implemented by means of different machine learning techniques, the most common being the vector support machines (SVM). Even some systems choose to use commercial optical character recognition (OCR) tools for this purpose.

II. Prior and related work

Classification of images

The recognition or classification of images consists of assigning an image a label of a defined set of categories according to their characteristics.

Although it seems a relatively trivial problem from our perspective, it is one of the most important challenges faced by artificial vision systems. Factors such as scale, lighting conditions, deformations or the partial concealment of objects make the classification of images a complex task, to which a great effort has been devoted to develop sophisticated techniques of pattern recognition, which do not always produce the expected results.

From the point of view of machine learning, the classification of images is a supervised learning problem, in which the classification algorithms generate a model from a previously categorized dataset or set of images. The model obtained is subsequently used to classify new images.

For these algorithms, the images are three-dimensional matrices whose dimensions are width, height and depth of color, the content of each position of the matrix being a numerical value that represents the intensity of color of each pixel of the digital image.

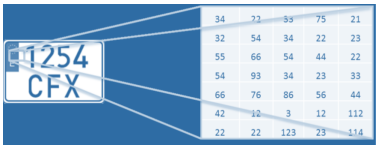


Figure 2- Digital image expressed as a matrix (the values of the pixels are fictitious)

Classification systems based on vector support machines (SVM) were the ones that offered the best results in image recognition and classification problems.

The main drawback of these systems is that they require a previous process of extracting the relevant characteristics of the images, almost always designed to measure the problem to be solved, for which sophisticated techniques of object and pattern detection are used - gradient histograms, and in which some intuition and knowledge of the input data is usually necessary.

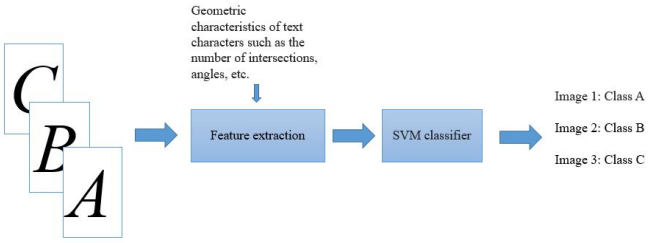


Figure 3 - Artificial vision system based on an SVM

The SVM classifier algorithm is trained from the extracted characteristics for a subset of the training dataset, so the effectiveness of the generated model depends on how representative these are.

The most negative aspect of these systems is that they do not really learn the characteristics or relevant attributes of each category, since these arrive to them predefined in the extraction stage. In addition, these systems are extremely sensitive to variations in scale, lighting, perspective, etc. who can present the images.

Description of your algorithm or system

Support Vector Machine (SVM)

• SVM is a supervised learning algorithm for classification
• SVM constructs a hyperplane or set of hyperplanes in a space of high or infinite dimension that differentiates the classes of patterns
• The SVM algorithm is based on finding the hyperplane that gives the greatest minimum distance to training example:

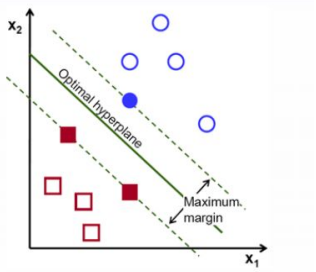


Figure 4 -optimal hyperplane for classification

• For SVM, you do not need all the training data only those that are close to the opposite group are sufficient (vector support)
• Any hyperplane can be written as the set of points x satisfactory:

$$\boldsymbol{w} \cdot \boldsymbol{x} + b_0 = 0$$

where w is a weighted vector, x is the pattern vector and b0 is the bias

• The minimum distance from the support vector to the limit decision (hyperplane) is given by 1 ‖ w ‖ . The margin is twice this distance, and we need to maximize this margin

Technologies used

Python

Python is a general purpose, multiplatform and multi-paradigm interpreted programming language (imperative, object-oriented and functional). It is a strongly typed language, that is, it is not possible to assign a variable of a different type to the initial one without an explicit conversion, this dynamic type being in turn: the type of the variables is determined in execution time in function of the assigned value.

OpenCV

OpenCV is an artificial vision library developed by Intel and currently published under BSD license, which has more than five hundred algorithms optimized to perform the main tasks of artificial vision, such as image processing, feature detection or object recognition. The library also has different machine learning algorithms such as vector support machines (SVM), Naïve Bayes or KNN among others.

It is written in C ++, is multiplatform and has interfaces to work with languages such as Java or Python. The large number of algorithms available, its speed of execution and the extensive community of users available, make OpenCV an indispensable tool for developing artificial vision systems based on open source software.

Image processing

The first process that is executed once the image of a license plate is introduced is the segmentation of characters, whose objective is to detect those regions of the image that contain the numbers and letters that the neuronal network classifier will later try to identify.
The Python module that runs this process is based on OpenCV, and uses the following image processing techniques:

Grayscale conversion

The color information is not necessary for the proposed problem and its elimination facilitates the detection of the characters of the license plate. For this reason, the first transformation that is applied to the image is a conversion to gray scale, or what is the same, to convert the image into a matrix in which each value represents the gray level of the corresponding pixel. This transformation is carried out using the cvtColor function of OpenCV.

Gaussian defocus

The second transformation that is made to the image is to apply a smoothing effect to eliminate the noise. Specifically, a Gaussian blur filter is applied, which consists in mixing the gray levels slightly between adjacent pixels, which results in an image with softer edges in which small details are lost, similar to what happens in the unfocused photographs. This transformation is done through the GaussianBlur function of OpenCV.

Thresholding

Thresholding or "threshold value technique" is a method that converts an image in black and white by setting a value from which all the pixels that exceed it become a binary color (white or black), and the rest in the opposite. In the case of vehicle registration this transformation allows obtaining an image in which the contours of the characters are clearly defined, facilitating the process of segmentation or isolation of the areas that contain them.
In the proposed system, this transformation is carried out using the

adaptiveThreshold function of OpenCV, in which the threshold value is calculated for different regions of the image, offering good results even if there are variations of illumination in the image.

Contour detection

Contours or edges are curves that join sets of contiguous pixels that have the same color or intensity. These curves allow to locate the borders of the objects in the image, and their detection is fundamental for an artificial vision system to recognize or detect shapes in an image.
In the proposed system, this detection is the last stage of pre-processing of the license plate image, and it is carried out by means of the functions findContours and boundingRect of OpenCV. The first one takes as input the binary image of the previous phase and generates a series of vectors of points for each detected contour. The second one is used to draw a rectangle that surrounds the contour, which will facilitate the subsequent extraction of the regions of the image that contain the numbers and letters of the license plate.
In the following illustration you can check the result of the four techniques applied to the image of a license plate:

Figure 5 - Treatment of the input image

Svm training

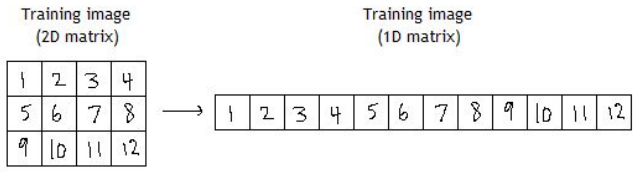
first you have to construct the training matrix for the SVM. This matrix is specified as follows: each row of the matrix corresponds to one image, and each element in that row corresponds to one feature of the class -- in this case, the color of the pixel at a certain point. Since your images are 2D, you will need to convert them to a 1D matrix. The length of each row will be the area of the images (note that the images must be the same size).

Let's say you wanted to train the SVM on 5 different images, and each image was 4x3 pixels. First you would have to initialize the training matrix. The number of rows in the matrix would be 5, and the number of columns would be the area of the image, 4*3 = 12.

```
int num_files = 5; int img_area = 4*3; Mat training_mat(num_files,img_area,CV_32FC1);
```

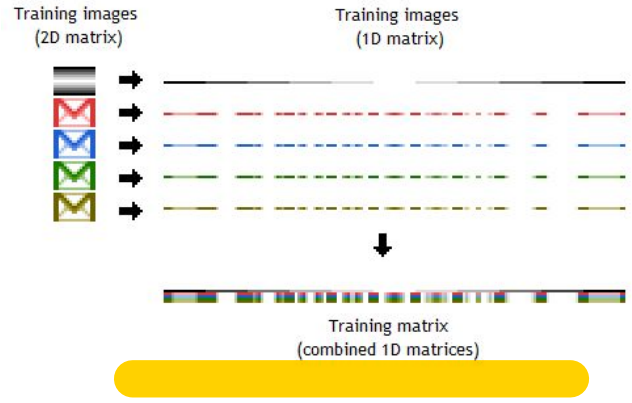
Ideally, num_files and img_area wouldn't be hardcoded, but obtained from looping through a directory and counting the number of images and taking the actual area of an image.

The next step is to "fill in" the rows of training_mat with the data from each image. Below is an example of how this mapping would work for one row.



I've numbered each element of the image matrix with where it should go in the corresponding row in the training matrix. For example, if that were the third image, this would be the third row in the training matrix.

You would have to loop through each image and set the value in the output matrix accordingly. Here's an example for multiple images:



Do this for every training image (remembering to increment file_num). After this, you should have your training matrix set up properly to pass into the SVM functions. The rest of the steps should be very similar to examples online:

```
Mat img_mat = imread(imgname,0); // I used 0 for greyscale int ii = 0; // Current column in training_mat for (int i = 0; i<img_mat.rows; i++) { for (int j = 0; j < img_mat.cols; j++) { training_mat.at<float>(file_num,ii++) = img_mat.at<uchar>(i,j); } }
```

I've had to deal with this recently, and here's what I ended up doing to get SVM to work for images.

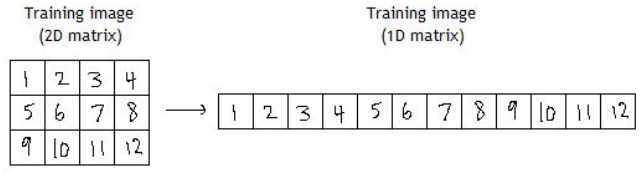
To train your SVM on a set of images, first you have to construct the training matrix for the SVM. This matrix is specified as follows: each row of the matrix corresponds to one image, and each element in that row corresponds to one feature of the class -- in this case, the color of the pixel at a certain point. Since your images are 2D, you will need to convert them to a 1D matrix. The length of each row will be the area of the images (note that the images must be the same size).

Let's say you wanted to train the SVM on 5 different images, and each image was 4x3 pixels. First you would have to initialize the training matrix. The number of rows in the matrix would be 5, and the number of columns would be the area of the image, 4*3 = 12.

```
int num_files = 5; int img_area = 4*3; Mat training_mat(num_files,img_area,CV_32FC1);
```

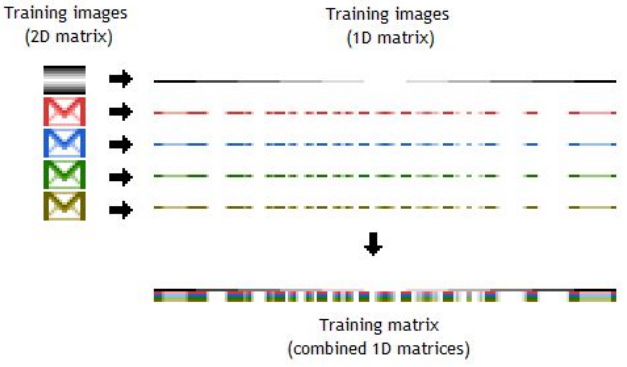
Ideally, num_files and img_area wouldn't be hardcoded, but obtained from looping through a directory and counting the number of images and taking the actual area of an image.

The next step is to "fill in" the rows of training_mat with the data from each image. Below is an example of how this mapping would work for one row.



I've numbered each element of the image matrix with where it should go in the corresponding row in the training matrix. For example, if that were the third image, this would be the third row in the training matrix.

You would have to loop through each image and set the value in the output matrix accordingly. Here's an example for multiple images:



As for how you would do this in code, you could use reshape(), but I've had issues with that due to matrices not being continuous. In my experience I've done something like this:

```
Mat img_mat = imread(imgname,0); // I used 0 for greyscale int ii = 0; // Current column in training_mat for (int i = 0; i<img_mat.rows; i++) { for (int j = 0; j < img_mat.cols; j++) { training_mat.at<float>(file_num,ii++) = img_mat.at<uchar>(i,j); } }
```

Do this for every training image (remembering to increment file_num). After this, you should have your training matrix set up properly to pass into the SVM functions. The rest of the steps should be very similar to examples online.

Note that while doing this, you also have to set up labels for each training image. So for example if you were classifying eyes and non-eyes based on images, you would need to specify which row in the training matrix corresponds to an eye and a non-eye. This is specified as a 1D matrix, where each element in the 1D matrix corresponds to each row in the 2D matrix. Pick values for each class (e.g., -1 for non-eye and 1 for eye) and set them in the labels matrix.

```
Mat labels(num_files,1,CV_32FC1);
```

So if the 3rd element in this labels matrix were -1, it means the 3rd row in the training matrix is in the "non-eye" class. You can set these values in the loop where you evaluate each image. One thing you could do is to sort the training data into separate directories for each class, and loop through the images in each directory, and set the labels based on the directory.

The next thing to do is set up your SVM parameters. These values will vary based on your project, but basically you would declare a CvSVMParams object and set the values:

```
CvSVMParams params;

params.svm_type = CvSVM::C_SVC;

params.kernel_type = CvSVM::POLY;

params.gamma = 3; // ...etc
```

There are several examples online on how to set these parameters, like in the link you posted in the question.

Next, you create a CvSVM object and train it based on your data!

```
CvSVM svm;

svm.train(training_mat, labels, Mat(), Mat(), params);
```

Depending on how much data you have, this could take a long time. After it's done training, however, you can save the trained SVM so you don't have to retrain it every time.

```
svm.save("svm_filename");// saving svm.load

("svm_filename"); // loading
```

To test your images using the trained SVM, simply read an image, convert it to a 1D matrix, and pass that in to svm.predict():

```
svm.predict(img_mat_1d);
```

It will return a value based on what you set as your labels (e.g., -1 or 1, based on my eye/non-eye example above). Alternatively, if you want to test more than one image at a time, you can create a matrix that has the same format as the training matrix defined earlier and pass that in as the argument. The return value will be different, though.

IV. Experimental results

TRAINING CHARACTERS

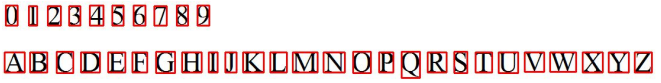
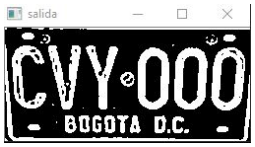


Image trainig _number.png

Test 1

ORIGINAL IMAGE



THRESHOLDING OF THE IMAGE

CHARACTER RECOGNITION

OUTPUT BY CONSOLE

Test 2

THRESHOLDING OF THE IMAGE

CHARACTER RECOGNITION

OUTPUT BY CONSOLE

As you can see, in test 1 the program works correctly in in quality images in test 2 not detects the registration correctly by containing irregular brightness and the frame of the plate.

V. Conclusions

The objective of the project was to develop a software with a manageable graphical interface, which would automatically detect license plates from car images and extract the number with a text format readable by the computer. This objective has been met, although as you can see in the previous section with the examples, the program has its limitations. These limitations come largely from the beginning of the synoy project that they study a lot about C ++ and that they learn the operation of the openCV library and because of ignorance of gastronomic alternatives, although the correct ones for the ignorance of the alternatives have not been found. Better, an example is the way to detect the registration, which has greatly limited the correct operation of the program.
By that now I will propose some of the future possibilities that in my opinion would be necessary to improve the program.

In the first place, the detection of enrollment should improve the detection of enrollment to make it more functional.
Secondly, the correction of the inclination of the license plates, which allows us that the images could be from several angles.

VI. Bibliography

OpenCV – Python Tutorials. Alexander Mordvintsev, AbidK.Revision.
https://opencv-python-tutroals.readthedocs.io/en/latest/index.html
Simple Digit Recognition OCR in OpenCV-Python. Abid Rahman.
http://stackoverflow.com/questions/9413216/simple-digit-recognition-ocr-in-opencv-python
Wikipedia.org. http://wikipedia.org
https://docs.google.com/document/d/1GTs_sQMohxSuK1cpZnlg062ST7wx7PdijLiwBdh7JGk/edit?ts=5b2d65b5