Moratello Matteo 1205720 06/13/2019
Master Degree in Computer Engineering
A.Y. 2018/19 Course of Computer Vision

Project Report

AUTOMATIC LICENCE PLATE READING

Automatic licence plate reading require some intermediate processes in order to:

- Identify in the original image a window that contain the license plate (*);
- Identify the licence plate in the window found;
- Segment the licence plate and then extract all the digits;
- Use a model of neural network in order to classify the digits found, at this point we have a string of the number of the licence plate.

(*) I used this approach in order to deal with all image of the dataset, in fact a focus on a window permit to extract licence plates well in images with very different size without change algorithm parameters.

1.1 - Identification of the license plate location using the sliding window approach

The use of a sliding window is used in the program as a preprocessing of the image in order to detect subsequently the contours of the licence plate.
In order to detect the location of the licence plate in the best way, together with the sliding window a ORB extractor has been used.
The ORB extractor is used instead of SIFT because it is faster even if less precise, but since the images can also be very large, the time to calculate the descriptors can be very high and must be avoided especially for real-time systems.
For first, the descriptors of a sample of a licence plate are extracted with ORB, then, once the window size is chosen, the latter is scrolled in y and x direction and the features are extracted with ORB.
Using a matcher (BFMatcher in this case), all matches between descriptors of the window and the sample licence plate are computed.
In order to choose the best window that best cointain the licence plate, the average of distances of the matches is computed for each window. The best window will have the minimum mean of distances.
Considering the first image of the dataset (1.jpg), in the figure below are shown the matches between the sample plate and the first one window captured. It can be noticed that there are many false positives, in fact, the average of the distances of the correspondences is 93.6635 so it will certainly be discarded.

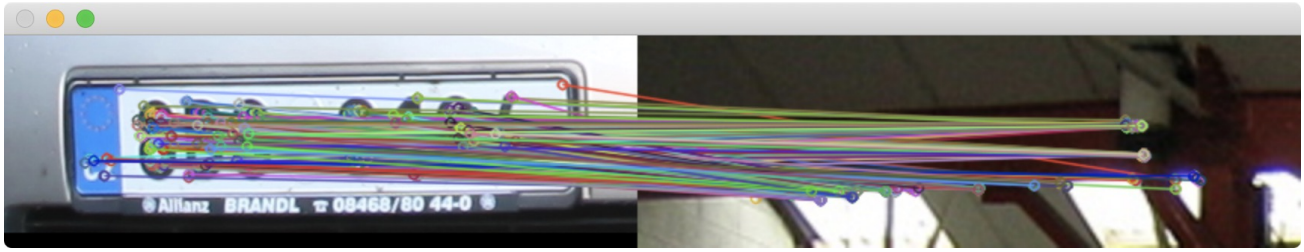


Figure 1: Matches between descriptors of the sample plate (on the left) and the first one window (on top-left of the image 1.jpg)

The next figure show instead the window that best contain the licence plate, in fact it's average of distances is 63.7841 , the minimum one between all the averages.

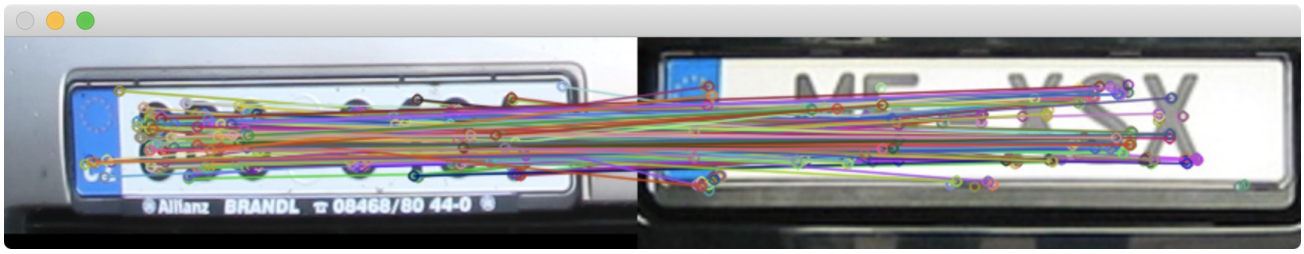


Figure 2: Matches between descriptors of the sample plate (on the left) and the best window found w.r.t. matches distances

Below are shown two examples of windows that best contain the license plate for first two images of the dataset.

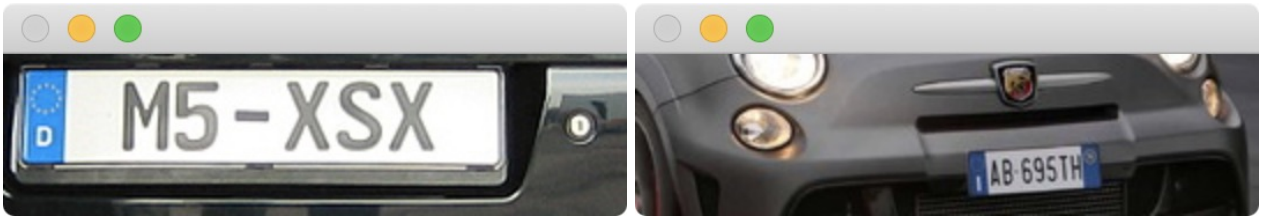


Figure 3: Best windows found for the first two images of the dataset. The best window is saved by adding some padding to the right to make sure you have taken the whole plate well.

It can be noted that for two very different images in dimensions such as the first two, obviously the regions containing the targe will also be very different, this is the reason why the sliding window approach was used, in the image on the right (Fiat abarth) it was very complex to find the licence plate in the whole image because the dimensions were very small.

1.2 - Extract the license plate from the window that contain it

In order to extract the licence plate from the window, a first process to apply on the window is Canny algorithm and then extract all edges from the image.
Considering the window found in image 1.jpg, the following image shows the edges found with the Canny algorithm.

The next step is to find the contours of the edges found with the opencv findContours() function and then, for each contour found, a quadrilateral that contains it is constructed with the boundingRect() function.
At this point, the quadrilateral that best approximates the shape of a licence plate will be selected.
A first constraint is applied to the area of the quadrilateral which must be sufficiently large (I chose, rect.area() $>$ 1000) and two other constraints on the width/height ratio of the quadrilateral. This in fact must be within a certain range (I chose, 2.7 $<=$ rect.width/rect.height $<=$ 5.5).
After applying all the constraints to the various contours found, the Figure 4 shows the remaining quadrilaterals that are colored green.

Figure 4: All rectangles found after applying constraints

Among all the rectangles found, the one with the smallest area is chosen and which will preferably contain only the characters of the plate. The final licence plate is shown in figure 5.

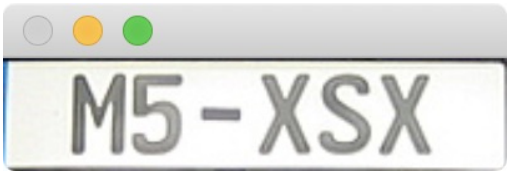
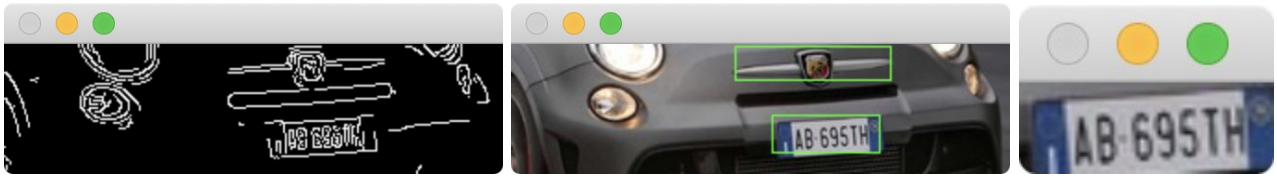


Figure 5: Licence plate found

Considering briefly the image 2.jpg (in which the plate is a bit more challenging to find), the process just described with the images is shown below and it can be seen that the licence plate is extracted correctly.



1.3 - Extract digits from licence plate image

In order to segment the licence plate, for first the image is converted in grayscale and then applied a threshold to the pixels intensities. The threshold transforms the image into a binary image, that is, with only the intensity values 0 or 255. The figures below show the result of the process just described.

The next step is to find the contours of the digits with the opencv findContours() function and for each contour found a quadrilateral is constructed that contains it, with the boundingRect() function. In order to discard shapes that aren't characters, some constraints are applied to quadrilaterals found. A first constraint is applied to the area of the quadrilateral which must be sufficiently large (in this case, rect.area() $>$ 50) and two other constraints on the width/height ratio of the quadrilateral. This in fact must be within a certain range (in this case, 1.4 <= rect.width/rect.height <= 4.6).
Another constraint applied in the selection of possible characters, is to discard the contours that contain more than 77% of white pixels (intensity 255) on the total number of pixels.
The use of this constraint was necessary because in some digits/characters (like 8 or B), the "holes" inside are identified as possible characters of the licence plate.
The figure 6 shows a possible case where this problem occurs.

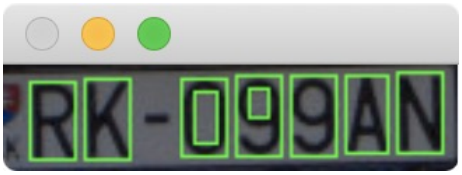
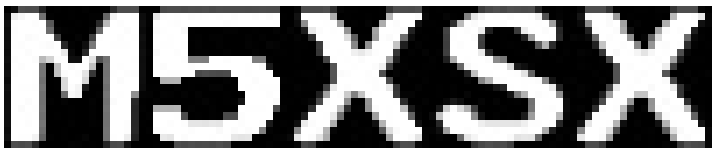


Figure 6: In this case the hole in digit 0 and in digit 9 are detected as chars

Figure 7 shows two results where all the characters are correctly segmented. The chars are then sorted according to their position in the x direction and saved in a vector of Mat.

Figure 7: Characters of the licence plate are correctly segmented

To get the correct input of the neural network, each character is resized to a size of 20x20 and the image has been made negative (1 - intensity of the pixel, for each pixel in the image).
For each plate, the images of the chars are concatenated in order to form a single image with the number of the licence plate and then saved in the folder "licence_plates".
Figure below shows the image of characters found in image 1.jpg.



The three processes seen so far are performed using methods of the PlateDetector class. Process 1.1 is contained in the **findPlateLocationInImage()** method, the process 1.2 in the **findBoundingOfPlate()** method and finally the process 1.3 in the **findCharsInPlate()** method. All the first part is write in c++ language.

2 - Use of a Neural Network to classify digits found

In order to classify digits found it is been used a neural network using python Deep Learning library Keras.
The biggest problem in using neural networks is to find a fairly large and qualitative dataset. To classify license plate characters, the MNIST dataset cannot be used because it uses handwritten digits/characters.
For the classification i used a dataset found on the net that includes characters and digits with different fonts (source: https://archive.ics.uci.edu/ml/datasets/Character+Font+Images).
The dataset ARIAL.csv is a csv file that contains:

- Label: integer of the ascii code
- Features: grayscale intensity of each pixels (400 features from 20x20 image)

Only the images relating to numbers or capital letters were extracted from the dataset, taking the number of classes to 36.

2.1 - Architecture of the network

The architecture of the network used is the one that best perform with MNIST dataset, in fact the images between the datasets are very similar.

- 2D convolution layer (number of filters: 18, size of each filter: 5x5)
  Spatial convolution over images
- MaxPooling2D
  It is used for reduce the dimension and allow for assumptions to be made about features contained in the sub-regions binned. It reduces the computational cost by reducing the number of parameters to learn.
- 2D convolution layer (number of filters: 26, size of each filter: 5x5)
- MaxPooling2D
- 2D convolution layer (number of filters: 24, size of each filter: 5x5)
- MaxPooling2D
- Flatten
- Dense (Fully connected layer)
- Dense (softmax)

Final layer returns the probability for each class that the input image belongs to that class.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input0 (InputLayer)          (None, 20, 20, 1)         0
_________________________________________________________________
conv1 (Conv2D)               (None, 20, 20, 18)        468
_________________________________________________________________
pool1 (MaxPooling2D)         (None, 10, 10, 18)        0
_________________________________________________________________
conv2 (Conv2D)               (None, 10, 10, 26)        11726
_________________________________________________________________
pool2 (MaxPooling2D)         (None, 5, 5, 26)          0
_________________________________________________________________
conv3 (Conv2D)               (None, 5, 5, 24)          15624
_________________________________________________________________
pool3 (MaxPooling2D)         (None, 2, 2, 24)          0
_________________________________________________________________
flatten (Flatten)            (None, 96)                0
_________________________________________________________________
fc1 (Dense)                  (None, 144)               13968
_________________________________________________________________
fc2 (Dense)                  (None, 36)                5220
=================================================================
Total params: 47,006
Trainable params: 47,006
Non-trainable params: 0
```

Figure 8: Summary of the Neural Network model

2.1 - Training of the Neural Network

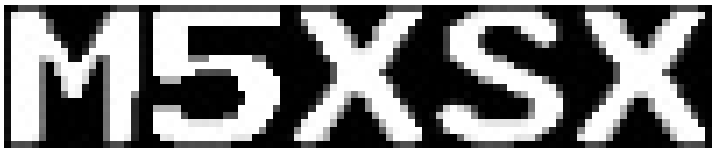
The image dataset composed of 8181 images was divided as follows:
- 7362 training set (90%)
- 819 validation set (10%)

The neural network has been trained for 8 epochs, achieving an estimation error of 0.01 and training error of 0.0019.

Neural Network training is performed by the python program “training_model.py”.

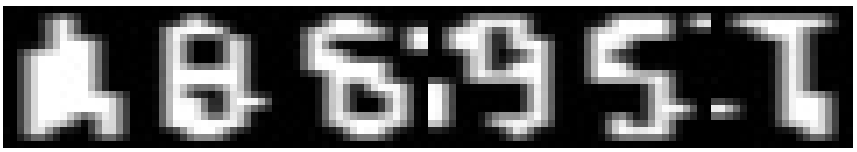
2.2 - Classification

In order to obtain a string of the number of the licence plate each single char extracted, from the concatenated image, is give in input to the neural network then all predicted characters are concatenated to form the final string. Example of correct classification is shown below (image 1.jpg).



Predict Labels: M5XSX
True Labels: M5XSX

In the image 2.jpg the characters found was very small and not very detailed (with also some outliers), in fact there are many misclassified.



Predict Labels: AVY351
True Labels: AB695T

In general, the network obtained 91% accuracy on the total of inputs (41 correct classify over 45). Chars classification is performed by the python program “classify_chars.py”

3 - Some results

The c++ program load the txt file that contains the strings of the chars of all licence plates and show:
- The original image with a green bounding box that highlights the licence plate found;
- The number of the plate found in green. Some results are shown below.

Figure 9: Result of image 7.jpg (the number of the licence plate is also print in output)

Figure 10: Result of image 1.jpg

Figure 11: Result of image 4.jpg