REPORT ON LICENSE PLATE RECOGNITION

PRATESI LEONARDO MATRICOLA 1214388

In this report I will explain the process in the making of the license plate algorithm in C++ and Python.

All the process was liberally inspired by the book “Mastering OpenCV”.

All the code and image used are available on GitHub at https://github.com/leopra/LicensePlateRecognition

The process can be divided in 3 STEPS:

* License Plate Recognition
* Single Digits Segmentation
* Character Recogniton

I will show the process on this partcular image of a BMW M5 back. The process of tuning the parameters was evaluated on the six given images.

In the coding process no C++ classes or headers where used to simplify the refactoring.

At first I apply a Gaussian Blur to reduce the noise of the image with a 5by5 kernel (other sizes were tested but this perfomed better). Then a Sobel filter is applied to emphasize vertical lines as a plate by having digits should have a lot of them.

A threshold filter is then applied so a rectangular dilate operator with size 20x3 can be executed with the purpose of linking together the vertical lines of the Sobel algorithm representing the plate’s characters.

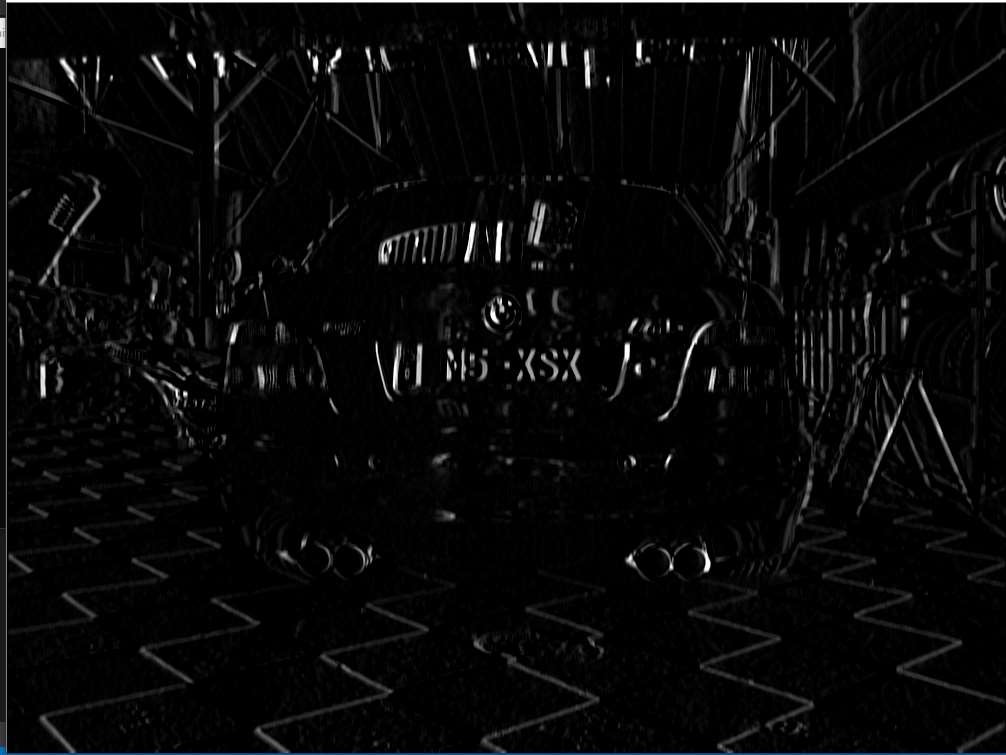


Figura 1 Sobel filter



Figura 2 threshold filter



Figura 3 Dilate filter

The function findContours is then applied.



Figura 4 find contours befoure checking for sizes



Figura 4.1 contours after checking for sizes

Then all the contours that cannot be contained in a rectangle in a range of area and radius (it was considered the plate to be a rectangle of measures 520x110).

This method was not enough to recognize the plates as many wrong results are obtained. By

checking the book “Mastering Opencv” a new method was suggested. By exploiting the fact that a plate is made of a white background linked alltogether I applied a floodfill algoritm by starting from 5 random points near the weighted center of the contours from the previous step.

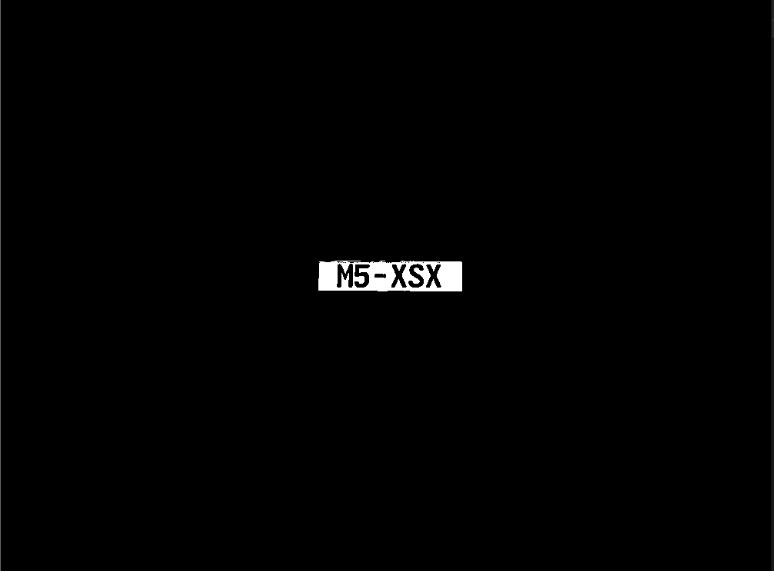


Figura 5 right mask found by floodfilling

Again findcontours() is applied ad the result not respectiong some kind of size are discarded.

The final results ( should be only one but the algorith is not perfect) are cutted to a standard resolution (33x140 as the next step only needs 28x28 characters) and saved in the licenseplate folder.

To better test this step I found a dataset online containing about 200 images of car plates.

https://platerecognizer.com/number-plate-datasets/

# SINGLE DIGITS SEGMENTATION

In the next step I explain the way of obtaining the characters and storing them.

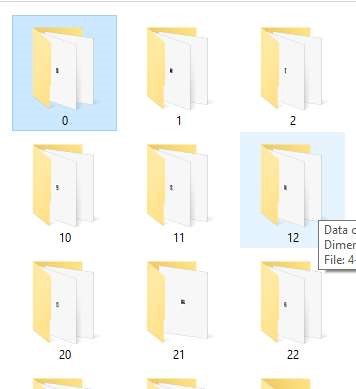
The single license plate is threshholded and then the external contours are searched. Even this time the size is checked to be the same ratio as the dimension of a license plate character.

The characters are then cutted, a white border is added to better match the images in the dataset of EMNIST (explained in the next step) and all the numbers and letters of the same

plate are stored in the same folder. This step returns almost perfect result if the plate in the previuos step was correctly found.



*Way of storing data*





# CHARACTER RECOGNITION

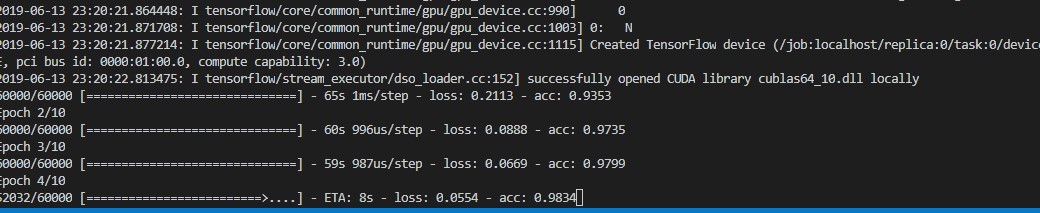
The recognition of the characters in the license plate is made by applying a neural network to the single images of the characters.

At first a neural network was trained on the MNIST dataset to see the accuracy in finding the digits in the plate and performed ok.

The neural network then was trained by considering only the classes of DIGITS and

UPPERSCORE LETTERS of the EMNIST dataset to also cover the letters in the license plate (the letters I, O, Q, U were not considered as european plates do not have these letters).

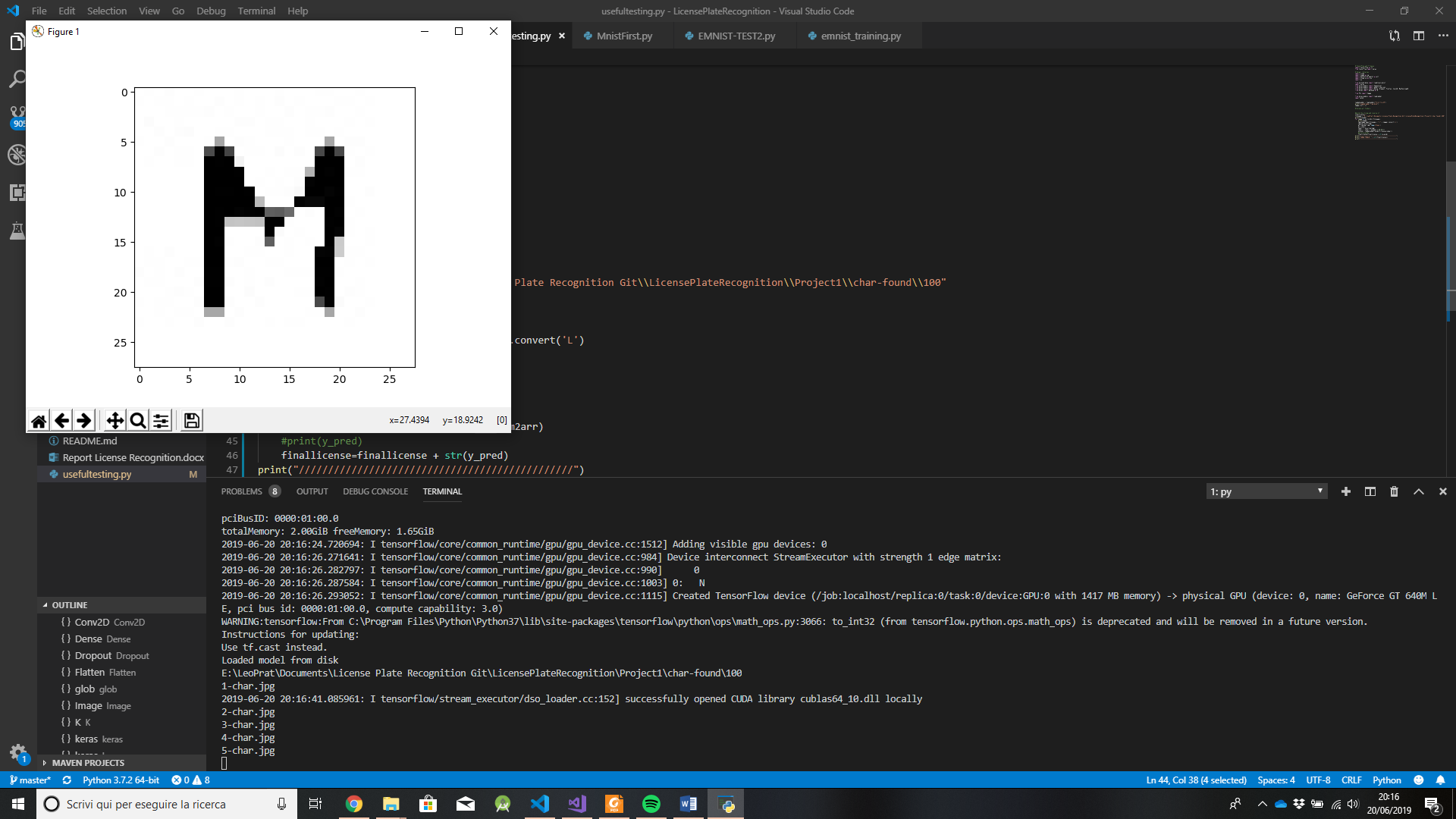
A lot of time was spent in the installation of the drivers needed to run the training on CPU as the GPU available was too old and so tensorflow-gpu had to be installed from source files.

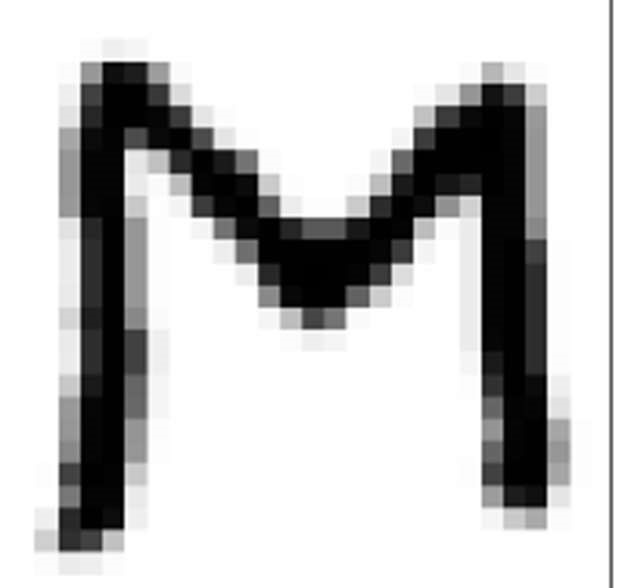
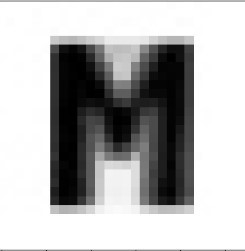


After the training unfortunately the results proved to be kind of disappointing,. The digits were correctly classified most of the time, but most of the letters were never recognized correctly (except A and B).

As can be seen down here the M of the license plate (center) is very different from the M of the EMNIST dataset (left). And on the right there is the M after the smallest erode possible was applied, and still the neural network was failing the recognition.

To sum up the results obtained: the process finds the license plate and the characters, is able to classify the digits but suffers in the classification of letters.





*Some examples of M letter in: eminst dataset, initial license processing and after erode operator.*

After my failure in delivering the project I decided to try template matching for character recognition with some results so bad I don’t even know if I shoud add it in the report.

So i created a folder with one example of each possible character and I applied a template matching function that returned the result with the highest similarity factor. (only the letter M was recognized).

