SAPIENZA UNIVERSITY - ROMA

Students: Alessio Borgi, Martina Doku, Giuseppina Iannotti

Reference email:

borgi.*****@studenti.uniroma1.it**

doku.*****@studenti.uniroma1.it,**

iannotti.*****@studenti.uniroma1.it**

DELIVERY DATE: July 2022 the 4th

Applied Computer Science and Artificial Intelligence

AI LAB PROJECT REPORT

"AUTOMATIC LICENSE PLATE RECOGNITION"

Table of Contents

INTRODUCTION

The project models a program for **Automatic License Plate Recognition(ALPR)** using an artificial intelligence approach with an Alphanumeric dataset, a **Convolutional Neural Network(CNN)** trained on it, completed by a database and a web application.

The problem we approached could be of interest in many real-life applications, we focused on a 'Company's Security' orientation. We tried to model the recognition as if:

1. The program was run by a company that wants to identify employees that are entering a gate.
2. The image acquisition comes from a camera on the gate that point to th front part of entering cars
3. The employees are registered in a database with their cars, so that decision about entrance allowing are made taking into account whether the car is or not of an employee

Nowadays plate recognition is getting more and more used in different fields such as:

1) On-road security: for speed control and tracking the amount of vehicles;

2) Highways payment, parking lots payment;

3) Security (private and not).

There are different hardware and software on markets that are selling products for that kind of recognition, (main Competitor is the famous **Google Tesseract**), but we tried to create on our own a valid alternative for competing with the great Giant :)

METHOD

To reach our result and detect the plates from our images/videos, we perform the following steps:

1. **TRAIN CNN MODEL ON OUR ALPHABET DATASET**
   A first preliminary step we execute is to train our Convolutional Neural Network. We defined our own model of CNN using:
   a. 2 convolutional layers
   b. 3 fully connected layers
   c. Mish as an activation function
   d. Cross Entropy as a Loss Function

   We trained our model on a maximum of 100 epochs and imported fixed weights (from a previously trained model) to start from a greater accuracy. We sampled in a balanced way taking 70% of samples from each class and using the remaining 30% for testing. We reached a maximum accuracy of 95.628%.

   THE DATASET:
   The Dataset we are using is a collection of labeled characters and numbers, for a total of 36 classes (10 numbers and 26 upper case characters) and 1016 samples for each class. The class is made of 28x28 pixel .png images, in black and white, stored inside directories with the label name.

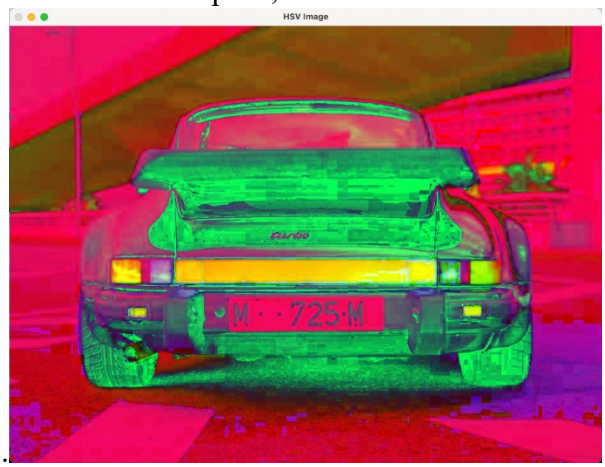
   (Note that, in this step, we had also initially tried with the EMNIST Dataset, but we opted for this other choice, due to the handwritten samples, and for the huge dimension).

2. **IMPORT THE CNN MODEL WE ARE GOING TO USE**
   The training is, of course, not executed each time we run the program to recognize a License Plate, we use import our model, previously saved in a 'Model' directory. We stored different accuracy-level models, in the final version of the program we use a model with about 91% of accuracy, since it performs better than higher accuracy models (due to overfitting).

3. **GET OUR IMAGE OR VIDEO FOR PLATE RECOGNITION**
   We have modeled three option of plate recognition:
   a. From **Image**
   b. From **Pre-Registered Video**
   c. From **Live-Video**

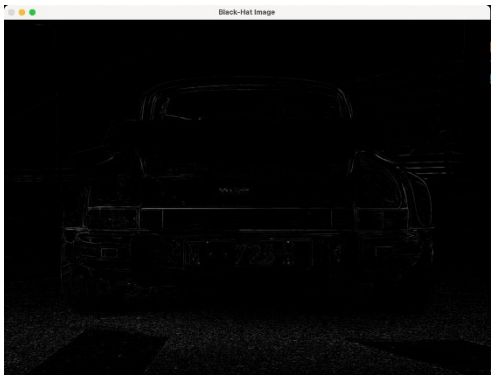
4. **GET POSSIBLE PLATES FROM IMAGE**
   a. **PRE-PROCESS THE IMAGE**
      We undergo the following list of transformations to the image to make the recognition easier:
      i. **TURN IMAGE GRAYSCALE**
         From the Original Image, that is in BGR Color Space, we first convert it to

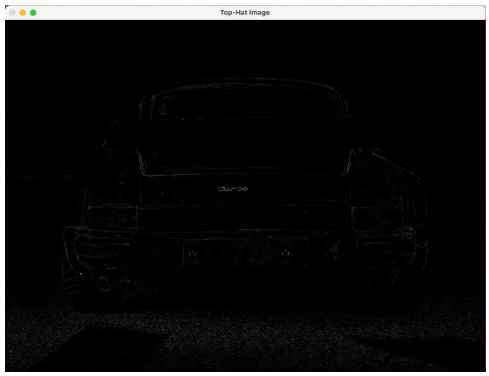
the HSV color Space, then we use the V values to obtain the grayscale image



ii. **ENHANCING THE IMAGE CONTRASTS**

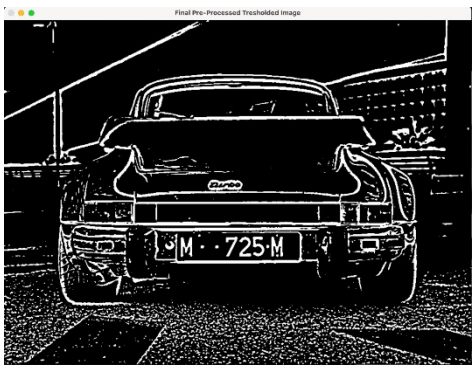
We apply two morphological transformation that remove noise and enhance the contrast: Top Hat Transformation, Black Hat Transformation





iii. **DE-NOISING**

We apply two final transformations to the image: **Gaussian Blur** and **Adaptive Thresholding,** that, w.r.t. the **Simple Thresholding** (in which for every pixel, the same threshold value is applied, in this case the algorithm determines the threshold for a pixel based on a small region(**ROIs**) around it so it performs better when an image has different lighting conditions in different areas.



b. **FIND CANDIDATE CHARCATERS (DIMENSION CHECK)**

Check roughly if the dimensions of the **Candidate Character** we've find out is

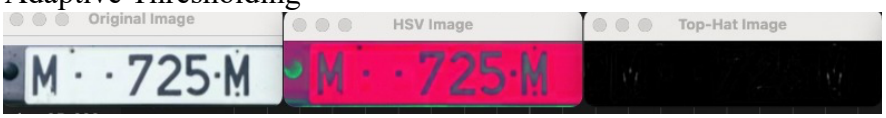
"acceptable". It represents a sort of 'First Screening' that is performed on a certain contour to see if it could be a character. The parameters we are using have been found empirically and they are such that far-away writings or character that are not in a standard plate format are immediately rejected.

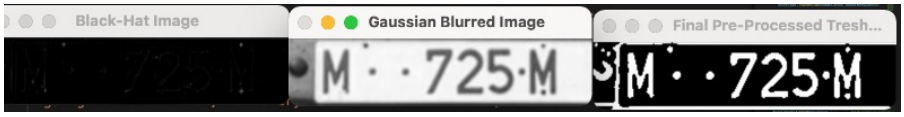
c. **CREATE A MATRIX OF COMPATIBLE CHARACTERS (ALLIGNEMENT CHECK)**
The reasoning here is to calculate for every character in the Big List of detected Characters, if it is a match with all the others. We define a **Match** to be a Character that is near to another and on the same horizontal line. For this reason, we are going to compute the **Pythagorean Distance** and the **Angle** between the two Characters. If they respect certain Threshold that we have imposed empirically, we decide to append the Candidate matching Character to the Candidate Plate List. In the end, we obtain a matrix with list of 'Candidate License Plates'.

d. **FIND PLATE PARAMETERS (SHAPE, PIECE OF IMAGE….)**
We start with the instantiation of an object of the Candidate License Blate class that we previously defined. Consequently, we first sort out the list of characters, depending on their distance. We then compute the COM of the License Plate, its width, and its height. The former is computed by simply doing the subtraction between the outermost and innermost Character, whilst for the latter we apply the following reasoning. First we sum up all the height of the characters and we divide it by the number of characters present in the License Plate, in such a way to be able to compute the Average Height between Characters.
Note that to both Height and Width we apply a multiplication for expanding the Rectangle due to the **Padding** of the License Plates (otherwise we have a rectangle that is exactly cropping the characters).
Finally we compute the correction angle we have to apply to the plate in order to have a perfectly aligned and straight rectangle. We can, in the end, return the Cropped Image of the License Plate.

5. FIND THE FINAL LICENSE PLATE

a. **PRE-PROCESS PLATE IMAGE**
Apply the preprocessing steps to the plate cropped image to further remove noise in that order:
-Grayscale Transformation
-Top-Hat and Black-Hat Transformation
-Gaussian Blur
-Adaptive Thresholding



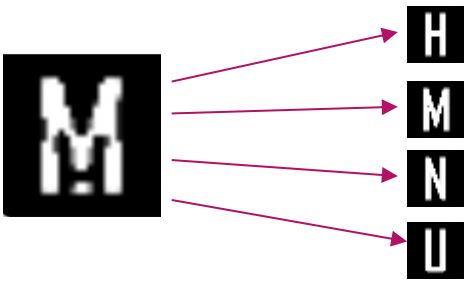


b. **FIND POSSIBLE CHARACTERS**
We retrieve the finals characters from the adjusted image, checking if proportions are correct for an admissible character.

c. **FIND ALLIGNED CHARACTERS**
Check if all the character retrieved are on the same line, using the function for trigonometric and Pythagorean distance of the COMs.

d. **FIND THE PLATE WITH MORE VALID CHARACTERS**
The previous steps are performed for all the possible plates, now that the actual number of characters is computed, the final plate is decided counting the one with the greatest number. This decision assumes that, once we are focusing on plate-like piece of images there won't be combination of character longer than our actual plate (also considering that the photo/and video acquisition contains mainly the car inside the frame)

e. **GETS THE PLATE 'STRING' VALUE (USING CNN)**
Given the chosen plate, each character identified by our **CNN** model, imported from the Model directory, that uses the weights obtained from training on our Alpha-Numeric Dataset to classify each character



6. SHOW PLATE WITH RECOGNISED PLATE NUMBER AND CAR INFOS

a. **FIND THE PLATE IN OUR DATA BASE**
Once each character has been recognized it is put into a string that constitutes the plate string.
We then perform a connection to a **Database** that contains the information about the CAR (plates associated with model, color, and so on), EMPLOYEEs (their corresponding plate, name, surname and so on.) PRESENCES (id, time, plate and so on for each access).
The string obtained from the plate characters is fetched inside our database plates

and information about the car (if available) are retrieved.

In addition, if the plate has been recognized a new '**access**' is uploaded into our database.

b. **RETRIEVE INFORMATIONS AND PUT IT ON IMAGE**
At the end of the process the program can show two types of images:

First case: The Plate has been Recognized
An image of the car with the plate bound by a green rectangle, the PLATE detected value and employee's info is shown on the screen.

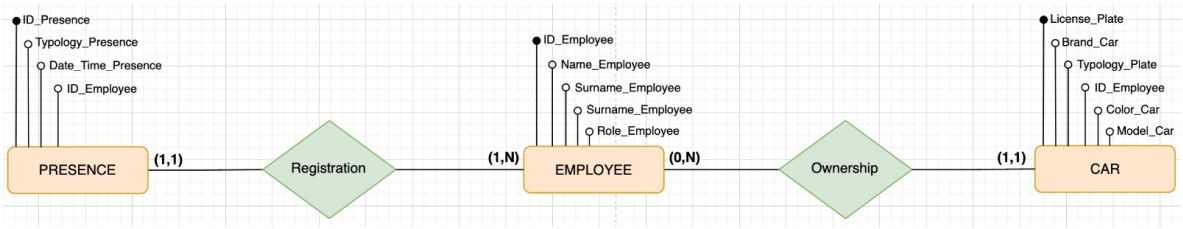
Second case: the plate has NOT been recognized
An image of the car with the plate bound by a red rectangle, the 'NOT RECOGNISED' strig and no employee's info, is shown on the screen.

EXTRA: DATABASE AND WEB APPLICATION

The project includes two extra parts that make it more prone to an eventual real-life use:
1) a **Database** that stores car information.
2) a **Web Application** to show that information and the user view.

DATABASE

The database is built on **PgAdmin** using **PostgreSQL** and is composed of three tables structured with the following ER schema:



The table Car and Employee are used from our project main file to retrieve information to be displayed on the final produced image and to recognize whether the plate is valid or not, while the Presence table contains instances representing different accesses.

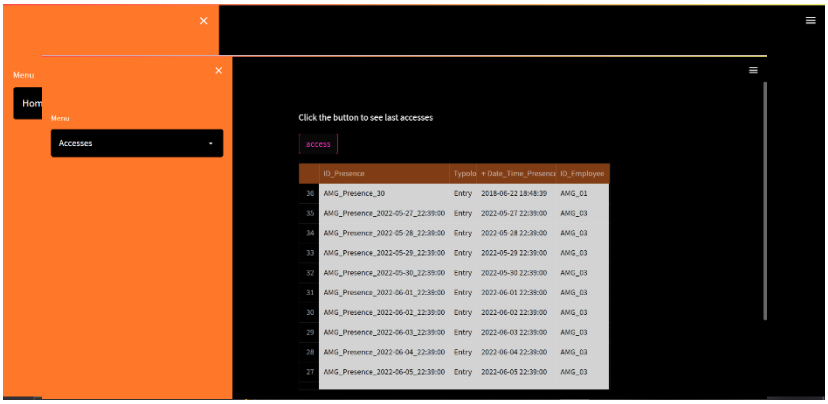
The database also implements a trigger that is then used to notify the web application after each update on the Presence table.

(We will provide you with the PgAdmin DB Snapshot, in such a way that you will be able to use the entire program).

WEB APP

We created a Web Application using the **Streamlit** python library , together with psycopg2 library (to communicate with our database), to show the data related to our project and to show the result of the execution of our program in Realtime.

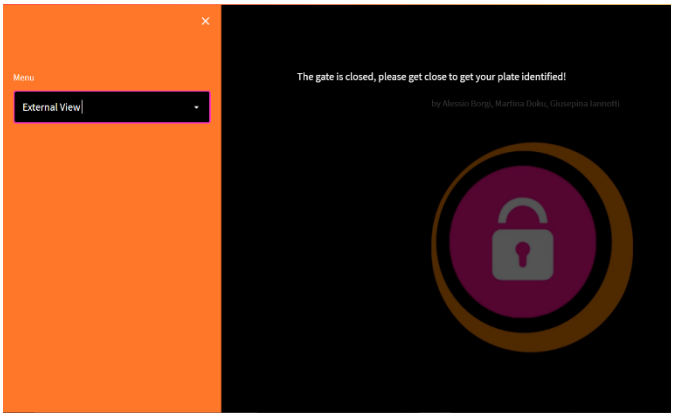
The web page can be run by typing **'streamlit run app/main.py'** in the terminal
The web app is composed of 4 pages, an hoe page, an 'Access' page, a 'Valid Plates' and 'User View' page.

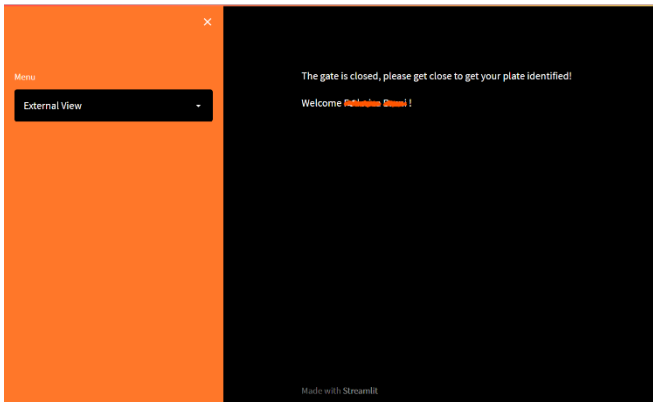


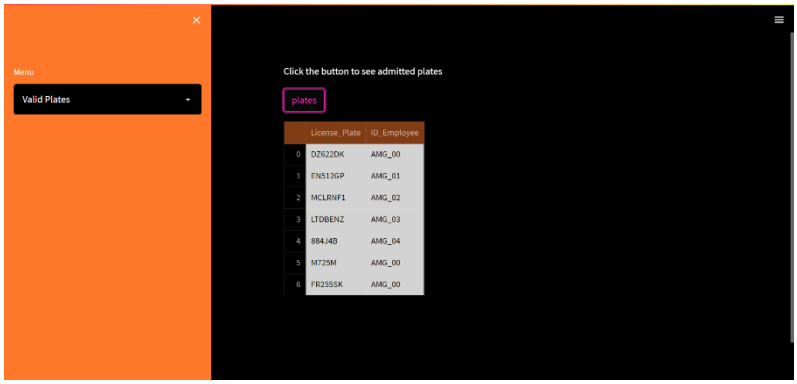
The Access Page shows the list of 'presence' table elements and can be refreshed with a button to get last updates, so that after each successful execution of the program a new presence is shown.

The Valid plates page shows the plates that are stored inside the database and that when recognized by the program would end in a successful insertion in the database.

The last page shows the view that the user has from ‘outside the gate’, it performs polling on the database and displays a welcome string whenever a new presence is registered, together with the name and surname of he recognized employee.







PROJECT PROBLEMS, CONCLUSION AND FINAL TAKEAWAYS

We modeled our project so that it could fit a real-life scenario of a company's security system. We made choices during the implementation that fit at best that idea.

An example is the choice of the dataset: there are several databases that store samples of numbers and letters, that are currently greatly used. One of the most famous ones is the '**EMNIST'** dataset. In the first version of our program, we tried to use that Dataset, but we faced the following problems:
1) Image Adjustments Needed: images were rotated in the dataset and had to be transformed to be used for classification. Thar required an additional computation step.
2) Unnecessarily High Number of Samples: The dataset contained an unfeasible number of samples: 814,255 characters. 62 unbalanced classes, that could make training too long and heavy.
3) Unused Classes: the dataset contained a class (and the relative high number of samples) for each lower letter of the alphabet, that do never appear in plates
4)Frequent Misclassification: the EMNIST dataset contains images of handwritten characters that can be often very different from the ones that we find on plates, that lead to a great percentage of error.
For those reasons we decided to exclude the EMNIST dataset and use a more 'problem-adequate' one.

Another challenge we faced was to choose the AI approach. We started with a **KNN** implementation that performed classification between flattened versions of both dataset and image character. The misclassification was however very high and very often it did not recognize the characters at all. We decide to switch to a more performing model: the **CNN**.

Another challenge we faced is **Portability** between devices. We worked on the project form three different devices, and we had to change small parts of the code to allow the correct execution on each device:
1) **Paths Composition**: When performing on windows devices the path had to be written using '\' separators, while MACOS devices use '/'.

2) **Database Access Data:** data are retrieved from a database stored on the device, and each database is locked with a different password that is chosen by the user on the PgAdmin application, and it must be specified at the beginning of the project

Note that the **Most Difficult** part of the Project was to Build-up, by our own, the Dataset. It was really a hard work, since we needed first to have a map between labels and images, then we had to do the manual split in test and train set, maintaining a balanced division.

However, at the very end, this **Project Succesfully Reached the Objectives we wanted to achieve**, recognizing, with a very great accuracy, car license plates from a photo/video. It does also constitute a good basis for a possible real-life usage.

We would like to thank you the Professor for some small advices provided during the project (usage of the Mish Activation function and some other tricks), but, most importantly, for having given the opportunity to put in practice what we have learned during the course, and for having created so much interest in the amazing World of Computer Vision!

Best,

Alessio Borgi Martina Doku Giuseppina Iannotti