Project – Expert systems

**Smart Parking**

**Introduction**

Searching for a parking spot has become a daily habit for most people everywhere. It is usually a time-consuming and frustrating process that many citizens have to go through. Besides the fact that parking is an everyday inconvenience, it also has a major impact on the pollution of the planet.

After naming the various problems that arise due to the search for parking spaces, only one question remains: How can this problem be solved? The solution is Smart Parking, but what does this term stand for?

**Definition**

Smart Parking is a parking solution that can include smart parking sensors in the ground, cameras or counting sensors. These devices are usually built into parking spaces or positioned next to them to detect whether parking spaces are free or occupied. This is done by collecting data in real time. The data is then transmitted to a smart parking mobile app or website, which communicates availability to its users. Some companies also provide other information in the app, such as prices and parking locations. This gives you the opportunity to explore every available parking option.

**Current status**

Smart Parking System provides certain features that help people understand what the products can do for them and how they will be able to achieve their goals. Each solution has at least one mandatory function, which is required to achieve the solution's core purpose, and several additional functions, which are features that can be added to provide additional benefits.

Funcții obligatorii:

· monitoring of parking spaces: products that control occupancy and length of stay in parking lots

· informing the driver about parking possibilities: products that inform the customer about free parking spaces

Funcții suplimentare:

· payment for parking: products that allow payment for the service

· parking space reservation: products that allow the reservation of a parking space

· guiding the driver to free parking spaces: products that show the way to free parking spaces

· customer car parking: products that automatically park the customer's car

**Ideas for future implementation**

Senzori de parcare inteligenți:

· The idea is simple: installing sensors throughout the city that communicate with an app to help drivers find free parking spaces.

Valet parking :

· In some airports like Dusseldorf, Germany, drivers can have the pleasure of experiencing the latest parking technology and having their car picked up and parked by robotic valets like Stan, which is made by Stanley Robotics, a parking harbinger future

Fully automated garages :

· In Boulder, Colorado, a fully automated garage is being installed where cars can be scanned with lasers and parked by robotic valets. They are then transported by an automated trolley to multi-level storage racks. This means that this garage can hold up to 4 times more cars than normal.

Multi-level parking :

· Smart parking systems must integrate multi-level parking so that multiple cars can be accommodated in a single parking lot. These include parking ideas such as lifts and sensors to guide your car and can contribute to some of the largest car parks in the world.

Asistenți personali virtuali:

· virtual assistants that guide drivers to their destinations may be able to guide them to the nearest free parking space using real-time data.

**Examples of smart parking**

VoicePark

· Being a key player in the parking sector has always been a priority for VoicePark. The first step was to develop an innovative smartphone application that improves and supports urban mobility. VoicePark was the first to direct step-by-step vehicles to open on-street parking spaces in real time during testing with the San Francisco Municipal Transportation Agency (SFMTA). The team was able to reduce the time it took cars to find a parking spot in San Francisco from 12.5 minutes to 45 seconds using proprietary algorithms.

Yazamtec

· Parking Hero Matrix is a blockchain-based solution for parking management, monetization and enforcement. The technology makes it possible to incorporate any parking space – on-street, off-street, private, commercial or individual – into a global distributed ledger.

Smart City System

· The Smart City system produces intelligent parking sensors that wirelessly transmit the current status of parking space occupancy. Their easy to use software allows you to view parking information and can also be easily connected to your current system. Parking sensors from smart city systems are completely wireless, simple to install and immediately operational.

Parquetry

· Any type of parking space, whether indoor or outdoor, permanent or temporary, continuous or irregular, in good or bad weather, can be searched in real time using Parquery. Parquery is a tried and true partner for end users looking for a smart parking solution to manage their spaces. Furthermore, it serves distributors, system integrators and manufacturers in the parking industry who are looking for a technology partner to expand their portfolio.

**Our project idea**

The system proposed by us in the first phase contains two barriers at the entrance to the parking lot (one for entry and one for exit), on which two cameras are implemented, with the help of which we will retrieve the information about the cars that want to enter or leave the parking lot. When a car arrives in front of the barrier, the person in the car will collect a parking ticket from an automatic machine located near the barrier. When the person in the car picks up the ticket, the camera mounted on the barrier will take the information about the car's registration number and send it to a database. How does he get this information? This is the first part that we do with the help of the Python programming language and PaddleOCR (a program that will take a picture of the car when the ticket is taken and that will look for the car's registration number in the image and transform into a text/string that will be saved in the database). After which the person will receive the ticket that will contain the registration number of the car, the time at which he entered and the time at which he should leave in order not to exceed the free period. Our system has implemented a program that allows the customer/person to stay in the parking lot for free for one hour, everything that goes beyond this hour will be billed.

Below is the example of the ticket as it would look like and we could also put a QR code that will give him the possibility to pay online from a website implemented by us (we make the work of the client easier).

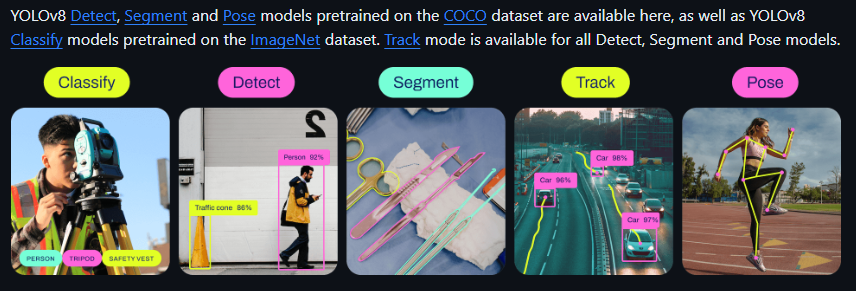


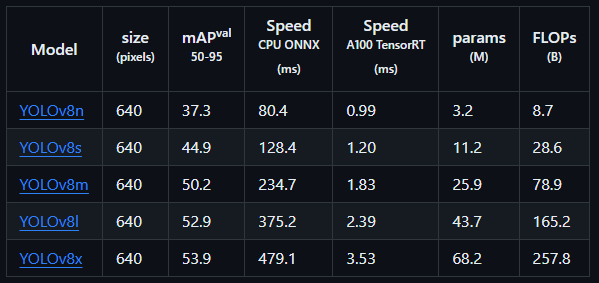
Also, when leaving the parking lot, a camera is mounted on the exit barrier that will detect the registration number of the car and if it is seen in the date base as paid for or still in the free period, then the barrier will open automatically. But if the respective registration number is unpaid in the database (it is possible that the customer forgot to pay), then he will either be able to pay online with the QR code or he will be able to pay at a machine that will be placed at the exit of the parking lot for cases like these and thus you will be able to leave the parking lot.

**YOLO V8 (** [**https://github.com/ultralytics/ultralytics**](https://github.com/ultralytics/ultralytics) **)**

YOLOv8 (You Only Look Once version 8) represents the technological peak in the field of effective object detection in images and video. Developed as an evolution of previous YOLO models, this algorithm provides outstanding performance, efficiency and accuracy in the process of identifying objects in complex scenes.

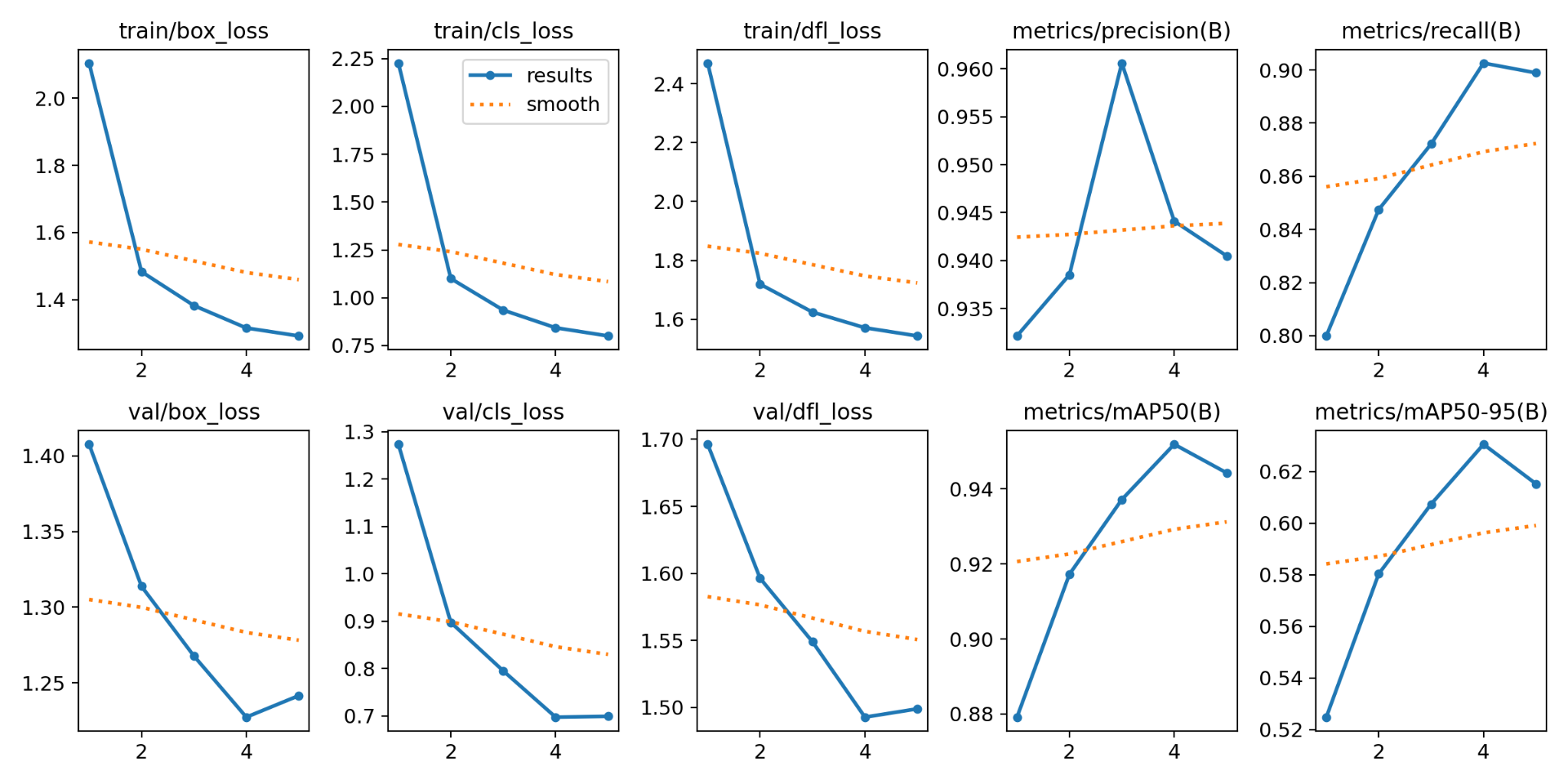
The fundamental principle of YOLO is the ability to detect and classify multiple objects in a single iteration without requiring multiple image analyses. YOLOv8 improves on this approach, bringing increased speed and improved accuracy. The algorithm divides the image into a grid and associates to each grid cells that predict the presence and characteristics of objects.



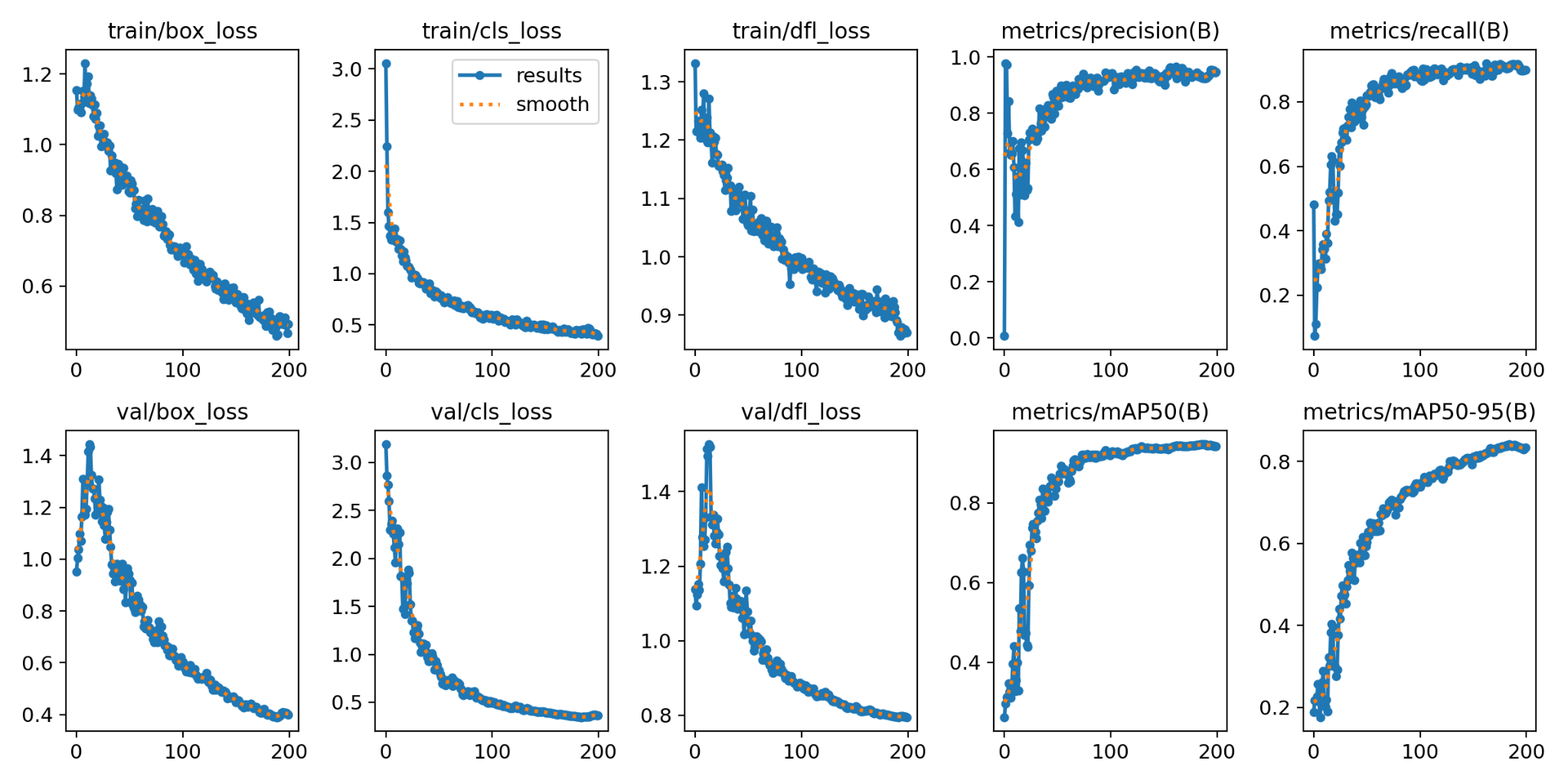


YoloV8 offers the possibility of using models such as Yolov8n, Yolov8s, Yolov8m, Yolov8l, Yolov8x. These pre-trained models are able to distinguish several objects such as people, cars, animals, structures such as buildings or bridges, but do not detect registration numbers.

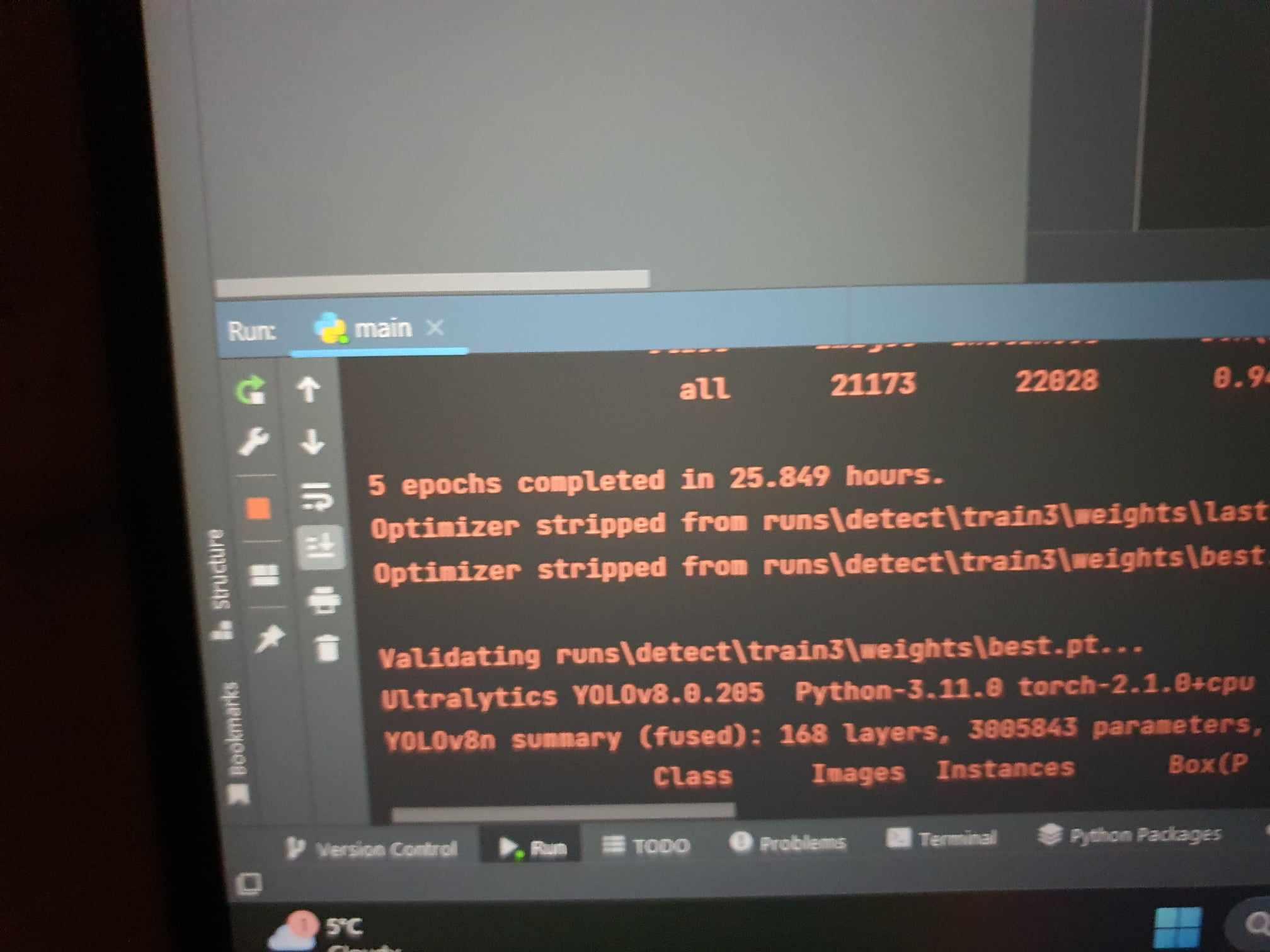
We tried to train our own license plate detection model, but this operation requires a lot of processing power, which we did not have. The model involved following 5 cycles through which the program had to go, i.e. I had a folder with approximately 20,000 pictures of vehicles in different positions where the registration numbers were visible and the program used for training looked for the position of the registration number in each picture the respective picture and save in a different folder a txt file (with the same name of the picture), the coordinates of the registration number. After the program went through those 5 cycles, the data regarding the analysis of the model created by us were saved in another file and at the same time a file with the extension py in which the created model is saved. But due to the fact that the model we trained was not so efficient and did not detect registration numbers so well, we found an already trained model on the Internet with 200 cycles, which worked well for our project and this model was the final variate on which I included in the project.

The figure below shows the results obtained by the model we created: 

In the figure below you can see the results obtained for the final model used with 200 cycles:



In this figure you can see how long it took to train our model:



The code we used to create our model:

from ultralytics import YOLO

# Load a model

model = YOLO ( "yolov8n.yaml" ) # build a new model from scratch

# Use the model

model . train ( data = "config.yaml" , epochs = 5 ) # train the model

The part of the final code where I used the model and Yolov8:

model\_path = bone \_ path . join ( 'models/license\_plate\_detector.pt' )

license\_plate\_detector = YOLO ( model\_path ) # load a custom model

**PaddleOCR:**

It is an optical character recognition (OCR) library and toolkit developed by PaddlePaddle, an open-source deep learning platform.

**Optical Character Recognition (OCR)** : The main purpose of PaddleOCR is to perform OCR, that is, to be able to extract text from scanned images or documents. This is very important for our project.

**Multi-language support:** PaddleOCR provides support for text recognition in multiple languages, allowing users to apply the technology in global contexts.

**PaddlePaddle Model:** PaddleOCR uses the PaddlePaddle deep learning platform. This includes pre-trained models for specific tasks, making implementation and use easier for developers.

**Flexibilitate și configurabilitate:**PaddleOCR can be configured to meet the specific needs of different applications. Users can adjust the parameters to get better results depending on the type of images they are working with.

**Updates and Community Support:** The PaddlePaddle platform and the PaddleOCR project have active community support, and updates and improvements may become available as the project evolves.

ocher = PaddleOCR ( use\_angle\_cls = True , lang = 'en' )

result = ocher \_ ocr ( license\_plate\_crop\_threshold , det = True , cls = True )

forum idx in the range ( len ( result )):

res = result [ idx ]

forum LINE in the res :

print ( line )

**firebat**

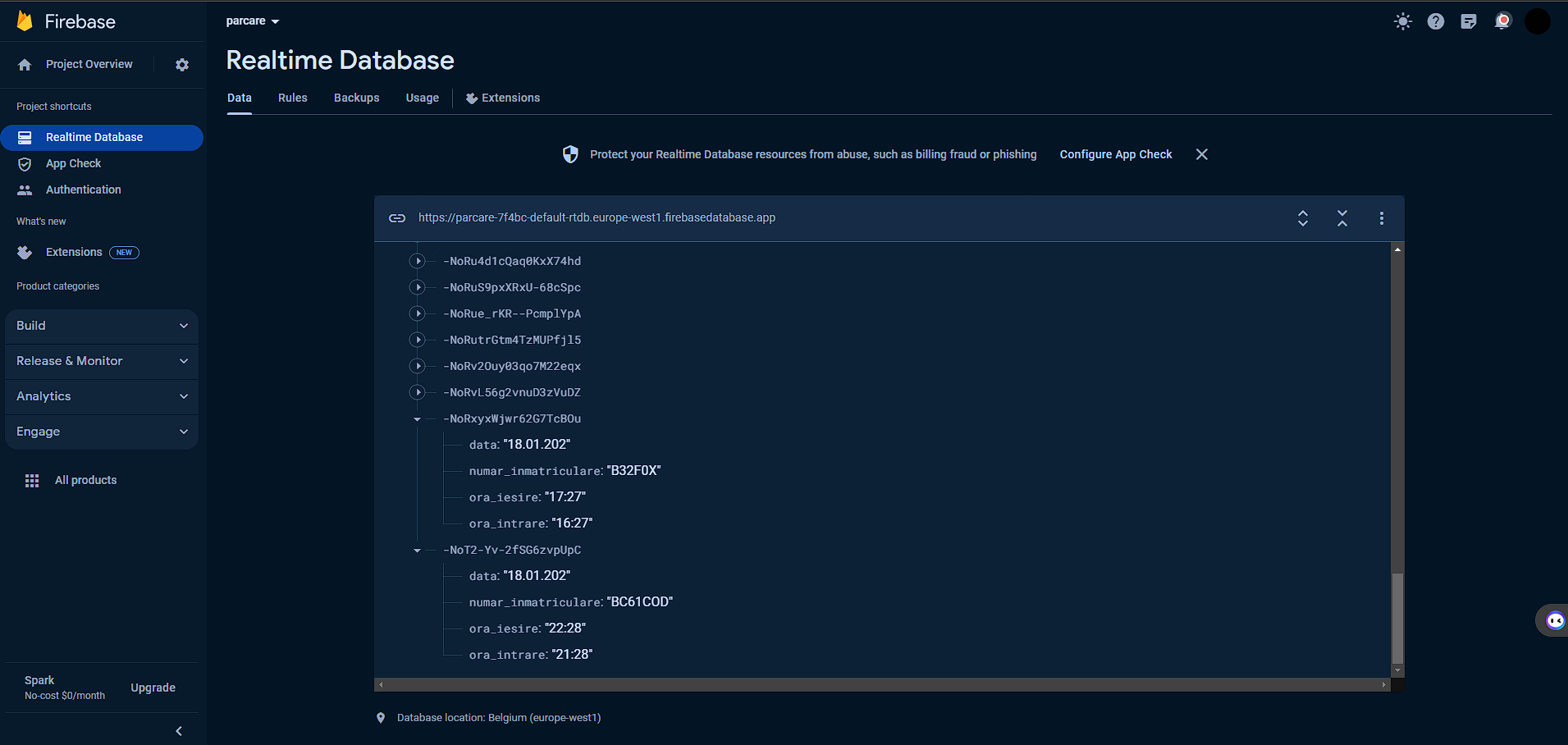
Firebase is a platform developed by Google that provides a suite of services and tools for the rapid development of mobile and web applications. It provides a wide range of functionality such as real-time database, authentication, hosting, messaging functionality, analytics and more.

We used Firebase in this project to save the following data:

- the registration number of the car;

- the day he entered the parking lot;

- the time at which he entered the parking lot;

- the time at which they must leave the parking lot.

The part of the final code where I used Firebase:

forum elements in the the list :

if element [ - 1 ] >= max :

MAX = element [ - 1 ]

# print(element[0])

variables = str ( license\_complies\_format ( element [ 0 ].replace( " " , "" ).upper()))

variable = str ( extract\_letters\_and\_numbers ( variables ))

print ( variable )

date\_to\_upload = {

'registration\_number' : variable ,

'entry\_time' : str ( timestamp [ 9 : 11 ]) + ":" + str ( timestamp [ 11 : 13 ]),

'output\_time' : str (( int ( timestamp [ 9 : 11 ]) + 1 )) + ":" + str ( timestamp [ 11 : 13 ]),

'date' : str (( int ( timestamp [ 6 : 8 ]))) + "." + str ( timestamp [ 4 : 6 ]) + "." + str ( timestamp [ 0 : 3 ])

}

resultat = FBconn . post ( "/MyTestData" , data\_to\_upload )

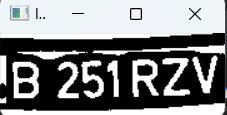
**CV2**

**OpenCV** ( [*Open Source*](https://ro.wikipedia.org/wiki/Software_cu_surs%C4%83_deschis%C4%83) *Computer Vision* ) is a computer function library specialized in real-time computer vision.

The use of this library in our code is to process the image that we obtain from the registration number model.

In order for the image to be as clear as possible so that when we apply PaddleOCR we get the best possible result, I applied the following filters:

1. We crop the image according to the coordinates obtained from the trained model
2. We apply a gray color to the image so that later we can apply another filter
3. We apply a blurring filter in such a way as to eliminate noise from the image
4. We apply an AdaptiveThresholt for the image in such a way that we can obtain an image in alt and black through which PaddleOCR will be able to extract the text (that is, the text with the parking number will be white and the background will be black)



new\_x1 = int ( x1 + (( x2 - x1 ) \* 0.085 ))

new\_x2 = int ( x2 - (( x2 - x1 ) \* 0.05 ))

new\_y1 = int ( y1 + (( y2 - y1 ) \* 0.04 ))

new\_y2 = int ( y2 - (( y2 - y1 ) \* 0.04 ))

license\_plate\_crop = image [ new\_y1 : new\_y2 , new\_x1 : new\_x2 ,:]

license\_plate\_crop\_gray = cv2 .cvtColor( license\_plate\_crop , cv2 .COLOR\_BGR2GRAY)

bfilter = cv2 .bilateralFilter( license\_plate\_crop\_gray , 11 , 17 , 17 ) # Noise reduction

license\_plate\_crop\_threshold = cv2 .adaptiveThreshold( bfilter , 255 , cv2 .ADAPTIVE\_THRESH\_GAUSSIAN\_C, cv2 .THRESH\_BINARY\_INV, 41 , 4 )

cv2 .imshow( 'license\_plate\_crop\_treshold' , license\_plate\_crop\_treshold )

**Python code:**

import bone

from datetime import datetime

from ultralytics import YOLO

import cv2

import string

from paddleocr import PaddleOCR

from firebat import firebat

from PIL import Image , ImageDraw , ImageFont

import qrcode

Qr = qrcode .QRCode(

version = 3 , # controls the size of the QR Code

error\_correction = qrcode .constants.ERROR\_CORRECT\_L, # controls the error correction used for the QR Code

box\_size = 6 , # controls how many pixels each "box" of the QR code is

border = 1 , # controls how many boxes to use for the border

)

FBconn = firebase . FirebaseApplication ( "https://parcare-7f4bc-default-rtdb.europe-west1.firebasedatabase.app/" )

list = []

variables = ""

# Mapping dictionaries for character conversion

dict\_char\_to\_int = { 'O' : '0' ,

'Q' : '0' ,

'I' : '1' ,

'J' : '3' ,

'A' : '4' ,

'G' : '6' ,

'S' : '5' }

dict\_int\_to\_char = { '0' : 'O' ,

'1' : 'I' ,

'3' : 'J' ,

'4' : 'A' ,

'6' : 'G' ,

'5' : 'S' }

def extract\_letters\_and\_numbers ( input\_string ):

return '' . join ( char forum Chara in the input\_string if char .isalnum())

def license\_complies\_format ( text ):

"""

Check if the license plate text complies with the required format.

args:

text (str): License plate text.

Returns:

bool: True if the license plate complies with the format, False otherwise.

"""

modified\_text = []

if text [ 0 ] == 'B' \

and text [ 1 ] in thong \_ numbers \

and text [ 2 ] in thong \_ numbers \

and text [ 3 ] in thong \_ ascii\_uppercase \

and text [ 4 ] in thong \_ ascii\_uppercase \

and text [ 5 ] in thong \_ ascii\_uppercase :

if text [ 0 ] in thong \_ ascii\_uppercase hours text [ 0 ] in dict\_int\_to\_char . key ():

modified\_text . append ( text [ 0 ])

modified\_text . append ( '\_' )

if text [ 1 ] in thong \_ digits hours text [ 1 ] in dict\_char\_to\_int . key ():

modified\_text . append ( text [ 1 ])

if text [ 2 ] in thong \_ digits hours text [ 2 ] in dict\_char\_to\_int . key ():

modified\_text . append ( text [ 2 ])

modified\_text . append ( '\_' )

if text [ 3 ] in thong \_ ascii\_uppercase hours text [ 3 ] in dict\_char\_to\_int . key ():

modified\_text . append ( text [ 3 ])

if text [ 4 ] in thong \_ ascii\_uppercase hours text [ 4 ] in dict\_int\_to\_char . key ():

modified\_text . append ( text [ 4 ])

if text [ 5 ] in thong \_ ascii\_uppercase hours text [ 5 ] in dict\_int\_to\_char . key ():

modified\_text . append ( text [ 5 ])

elif text [ 0 ] == 'B' \

and text [ 1 ] in thong \_ digits \

and text [ 2 ] in thong \_ numbers \

and text [ 3 ] in thong \_ numbers \

and text [ 4 ] in thong \_ ascii\_uppercase \

and text [ 5 ] in thong \_ ascii\_uppercase \

and text [ 6 ] in thong \_ ascii\_uppercase :

if text [ 0 ] in thong \_ ascii\_uppercase :

modified\_text . append ( text [ 0 ])

modified\_text . append ( '\_' )

if text [ 1 ] in thong \_ digit :

modified\_text . append ( text [ 1 ])

if text [ 2 ] in thong \_ digit :

modified\_text . append ( text [ 2 ])

if text [ 3 ] in thong \_ digit :

modified\_text . append ( text [ 3 ])

modified\_text . append ( '\_' )

if text [ 4 ] in thong \_ ascii\_uppercase :

modified\_text . append ( text [ 4 ])

if text [ 5 ] in thong \_ ascii\_uppercase :

modified\_text . append ( text [ 5 ])

if text [ 6 ] in thong \_ ascii\_uppercase :

modified\_text . append ( text [ 6 ])

elif text [ 0 ] in thong \_ ascii\_uppercase \

and text [ 1 ] in thong \_ ascii\_uppercase \

and text [ 2 ] in thong \_ numbers \

and text [ 3 ] in thong \_ numbers \

and text [ 4 ] in thong \_ ascii\_uppercase \

and text [ 5 ] in thong \_ ascii\_uppercase \

and text [ 6 ] in thong \_ ascii\_uppercase :

if text [ 0 ] in thong \_ ascii\_uppercase hours text [ 0 ] in dict\_int\_to\_char . key ():

modified\_text . append ( text [ 0 ])

if text [ 1 ] in thong \_ ascii\_uppercase hours text [ 1 ] in dict\_int\_to\_char . key ():

modified\_text . append ( text [ 1 ])

modified\_text . append ( '\_' )

if text [ 2 ] in thong \_ digits hours text [ 2 ] in dict\_char\_to\_int . key ():

modified\_text . append ( text [ 2 ])

if text [ 3 ] in thong \_ digits hours text [ 3 ] in dict\_char\_to\_int . key ():

modified\_text . append ( text [ 3 ])

modified\_text . append ( '\_' )

if text [ 4 ] in thong \_ ascii\_uppercase hours text [ 4 ] in dict\_int\_to\_char . key ():

modified\_text . append ( text [ 4 ])

if text [ 5 ] in thong \_ ascii\_uppercase hours text [ 5 ] in dict\_int\_to\_char . key ():

modified\_text . append ( text [ 5 ])

if text [ 6 ] in thong \_ ascii\_uppercase hours text [ 6 ] in dict\_int\_to\_char . key ():

modified\_text . append ( text [ 6 ])

else :

return text

final\_text = '' . join ( modified\_text )

return final\_text

PHOTO\_DIR = bone \_ path . join ( '.' , 'photos' )

number\_palace = ""

# Read webcam

webcam = cv2 .VideoCapture( 0 ) # This assumes you have only one camera attached to the laptop

model\_path = bone \_ path . join ( 'models/license\_plate\_detector.pt' )

license\_plate\_detector = YOLO ( model\_path ) # load a custom model

threshold = 0.5

while true :

ret , frame = webcam .read()

# Display the frame

cv2 .imshow( 'Frame' , frame )

key = cv2 .waitKey( 40 ) & 0x FF

if key == ord ( 'q' ):

break

elif key == ord ( 's' ):

# Capture screenshot and save with current date and time

now = datetime . now ()

timestamp = now . strttime ( "%Y%m %d \_%H%M" )

screenshot\_path = bone \_ path . join ( PHOTO\_DIR , f 'screenshot\_ { timestamp } .png' )

print ( timestamp )

cv2 .imwrite( screenshot\_path , frame )

image = cv2 .imread( screenshot\_path )

try :

license\_plates = license\_plate\_detector ( image )[ 0 ]

forum license\_plate in the license\_plates .boxes.xyxy:

x1 , y1 , x2 , y2 = license\_plate

# crop the license plate

new\_x1 = int ( x1 + (( x2 - x1 ) \* 0.085 ))

new\_x2 = int ( x2 - (( x2 - x1 ) \* 0.05 ))

new\_y1 = int ( y1 + (( y2 - y1 ) \* 0.04 ))

new\_y2 = int ( y2 - (( y2 - y1 ) \* 0.04 ))

license\_plate\_crop = image [ new\_y1 : new\_y2 , new\_x1 : new\_x2 ,:]

license\_plate\_crop\_gray = cv2 .cvtColor( license\_plate\_crop , cv2 .COLOR\_BGR2GRAY)

bfilter = cv2 .bilateralFilter( license\_plate\_crop\_gray , 11 , 17 , 17 ) # Noise reduction

license\_plate\_crop\_threshold = cv2 .adaptiveThreshold( bfilter , 255 , cv2 .ADAPTIVE\_THRESH\_GAUSSIAN\_C, cv2 .THRESH\_BINARY\_INV, 41 , 4 )

cv2 .imshow( 'license\_plate\_crop\_treshold' , license\_plate\_crop\_treshold )

ocher = PaddleOCR ( use\_angle\_cls = True , lang = 'en' )

result = ocher \_ ocr ( license\_plate\_crop\_threshold , det = True , cls = True )

forum idx in the range ( len ( result )):

res = result [ idx ]

forum LINE in the res :

print ( line )

#print the ticket

print ( "\_\_\_\_GOOD DAY\_\_\_\_" )

print ( f " { timestamp [ 0 : 4 ] } / { timestamp [ 4 : 6 ] } / { timestamp [ 6 : 8 ] } " )

print ( "ENTER TIME:" )

print ( f " { timestamp [ 9 : 11 ] } : { timestamp [ 11 : 13 ] } " )

print ( "EXIT TIME:" )

print ( f " { int ( timestamp [ 9 : 11 ]) + 1 } : { timestamp [ 11 : 13 ] } " )

print ( "NUMBER PLATE:" )

variables = ""

list = []

forum idx in the range ( len ( result )):

res = result [ idx ]

forum LINE in the res :

the list . append ( line [ 1 ])

MAX = list [ 0 ][ - 1 ]

# print(type(max))

forum elements in the the list :

if element [ - 1 ] >= max :

MAX = element [ - 1 ]

# print(element[0])

variables = str ( license\_complies\_format ( element [ 0 ].replace( " " , "" ).upper()))

variable = str ( extract\_letters\_and\_numbers ( variables ))

print ( variable )

date\_to\_upload = {

'registration\_number' : variable ,

'entry\_time' : str ( timestamp [ 9 : 11 ]) + ":" + str ( timestamp [ 11 : 13 ]),

'output\_time' : str (( int ( timestamp [ 9 : 11 ]) + 1 )) + ":" + str ( timestamp [ 11 : 13 ]),

'date' : str (( int ( timestamp [ 6 : 8 ]))) + "." + str ( timestamp [ 4 : 6 ]) + "." + str ( timestamp [ 0 : 3 ])

}

resultat = FBconn . post ( "/MyTestData" , data\_to\_upload )

print ( result )

print ( "\_\_\_\_GOODBYE\_\_\_\_" )

ticket\_name = variable + "\_" + str (( int ( timestamp [ 6 : 8 ]))) + str ( timestamp [ 4 : 6 ]) + str ( timestamp [ 0 : 4 ]) + "\_" + str ( timestamp [ 9 : 11 ]) + str ( timestamp [ 11 : 13 ]) + "\_" + str (( int ( timestamp [ 9 : 11 ]) + 1 )) + str ( timestamp [ 11 : 13 ])

date = ticket\_name

qr .add\_data( date )

qr .make( fit = True )

# Create an image from the QR Code instance

img\_qr = qr .make\_image( fill\_color = "black" , back\_color = "white" )

# Create a blank image

image = Images . new ( "RGB" , ( 250 , 400 ), "white" )

draw = ImageDraw . draw \_ \_ \_

# Add text and details to the image

font\_size = 22

FONT = ImageFont . load\_default ()

font1 = ImageFont . truetype ( "C:/Users/PC/Desktop/proiectse2/fonts/Arial.ttf" , font\_size )

font2 = ImageFont . truetype ( "C:/Users/PC/Desktop/proiectse2/fonts/Arial.ttf" , font\_size - 7 )

draw \_ text (( 57 , 30 ), "Parking Ticket" , fill = "black" , font = font1 )

draw \_ text (( 15 , 75 ), "Registration number: " + str ( variable ), fill = "black" , font = font2 )

draw \_ text (( 15 , 100 ), "Date: " + str (( int ( timestamp [ 6 : 8 ]))) + "." + str ( timestamp [ 4 : 6 ]) + "." + str ( timestamp [ 0 : 4 ]), fill = "black" , font = font2 )

draw \_ text (( 15 , 125 ), "Entry time: " + str ( timestamp [ 9 : 11 ]) + ":" + str ( timestamp [ 11 : 13 ]), fill = "black" , font = font2 )

draw \_ text (( 15 , 150 ), "Exit time: " + str (( int ( timestamp [ 9 : 11 ]) + 1 )) + ":" + str ( timestamp [ 11 : 13 ]), fill = "black" , font = font2 )

image . paste ( img\_qr ,( 31 , 185 ))

# Save the image to a folder

image . save ( "C:/Users/PC/Desktop/proiectse2/ticket/" + ticket\_name + ".png" )

date = ""

print ( "Parking ticket image created and saved." )

exception Exception I would is :

print ( "something gone wrong:((((" )

print ( s )

#delete the screen from folder

if bone \_ path . exists ( screenshot\_path ):

bone \_ remove ( screenshot\_path )

webcam .release()

cv2 .destroyAllWindows()

**Code operation:**

The camera is used to take a picture of the car. Then with the help of YOLO and the registration number detection model, the license plate coordinates will be saved. After that we use those coordinates to apply a crop to the picture so that we can continue to read only the text that interests us, namely the registration number, with the help of PaddleOCR.

To be able to read the text properly with PaddleOCR, we apply filters and corrections to the image.

In order for the reading of the text to be as good as possible, the text must be black and not blur. We use an adaptive threshold to remove reflections from the image.

After retrieving the data from the final image, the data will be saved in the database, a prototype of the ticket will be made in the console and the actual ticket will be created.