

Ministerul Educației și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică și Microelectronică

Departamentul Ingineria Software și Automatică

**RAPORT**

**Lucrarea de laborator nr.6**

**la Inteligența Artificială**

*Tema: Detectarea de obiecte în imagini*

Grupa academică: TI-211  
A efectuat: Popa Cătălin

**A verificat: Mariana Rusu**

Chișinău 2024

YOLO (You Only Look Once) este o metodă populară de detectare a obiectelor în imagini și videoclipuri. A fost dezvoltată pentru a obține o detecție rapidă și precisă a obiectelor în timp real. Principiul de bază al YOLO este că imaginea este împărțită într-o grilă și pentru fiecare celulă a grilei, algoritmul încearcă să prezică regiunile de interes (bounding boxes) și clasele obiectelor conținute în acele regiuni. Acest lucru se face într-o singură trecere a rețelei neurale, ceea ce face ca YOLO să fie extrem de eficient în comparație cu alte metode care pot necesita mai multe etape de prelucrare.

**Avantajele cheie ale YOLO includ:**

* **Viteză**: YOLO este capabil să detecteze obiecte în timp real pe imagini și videoclipuri, datorită abilității sale de a face predicții rapid.
* **Precizie**: YOLO are o precizie ridicată în detectarea obiectelor, deoarece analizează întreaga imagine într-o singură trecere și încearcă să prezică bounding boxes și clasele obiectelor simultan.
* **Versatilitate**: YOLO este versatil și poate fi antrenat să detecteze o varietate largă de obiecte, inclusiv persoane, mașini, animale, obiecte domestice și multe altele.

Inițial vom instala librăria Ultralytics și apoi rulăm funcția checks(), pentru a verifica configurarea și setările. Ultralytics este o bibliotecă Python care oferă suport pentru antrenarea și evaluarea modelelor de detecție a obiectelor, inclusiv variantele YOLO.

!pip install ultralytics==8.0.196

from IPython import display

import ultralytics ultralytics.checks()

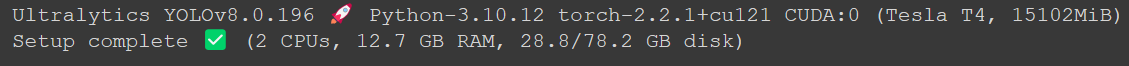


Figura 1 – Vizualizare configurări

Importăm YOLO, pentru a inițializa și manipula modele YOLO, pentru sarcina de detecție a obiectelor. La fel importăm funcțiile display și Image, care vor fi utilizate pentru afișarea imaginilor.

from ultralytics import YOLO

from IPython.display import display, Image

Apoi, creăm un directoriu nou, unde se va salva datasetul nostru cu imagini și folosind cd ne deplasăm în directoriul creat.

!mkdir {HOME}/datasets

%cd {HOME}/datasets

Instalăm biblioteca roboflow, care oferă funcționalități pentru lucrul cu seturi de date pentru detecția obiectelor, inclusiv încărcarea, descărcarea și prelucrarea acestora.

!pip install roboflow

Pentru a crea setul de date, am folosit Roboflow, unde a fost nevoit să evidențiăm in iamgini ce tip de vehicol este.

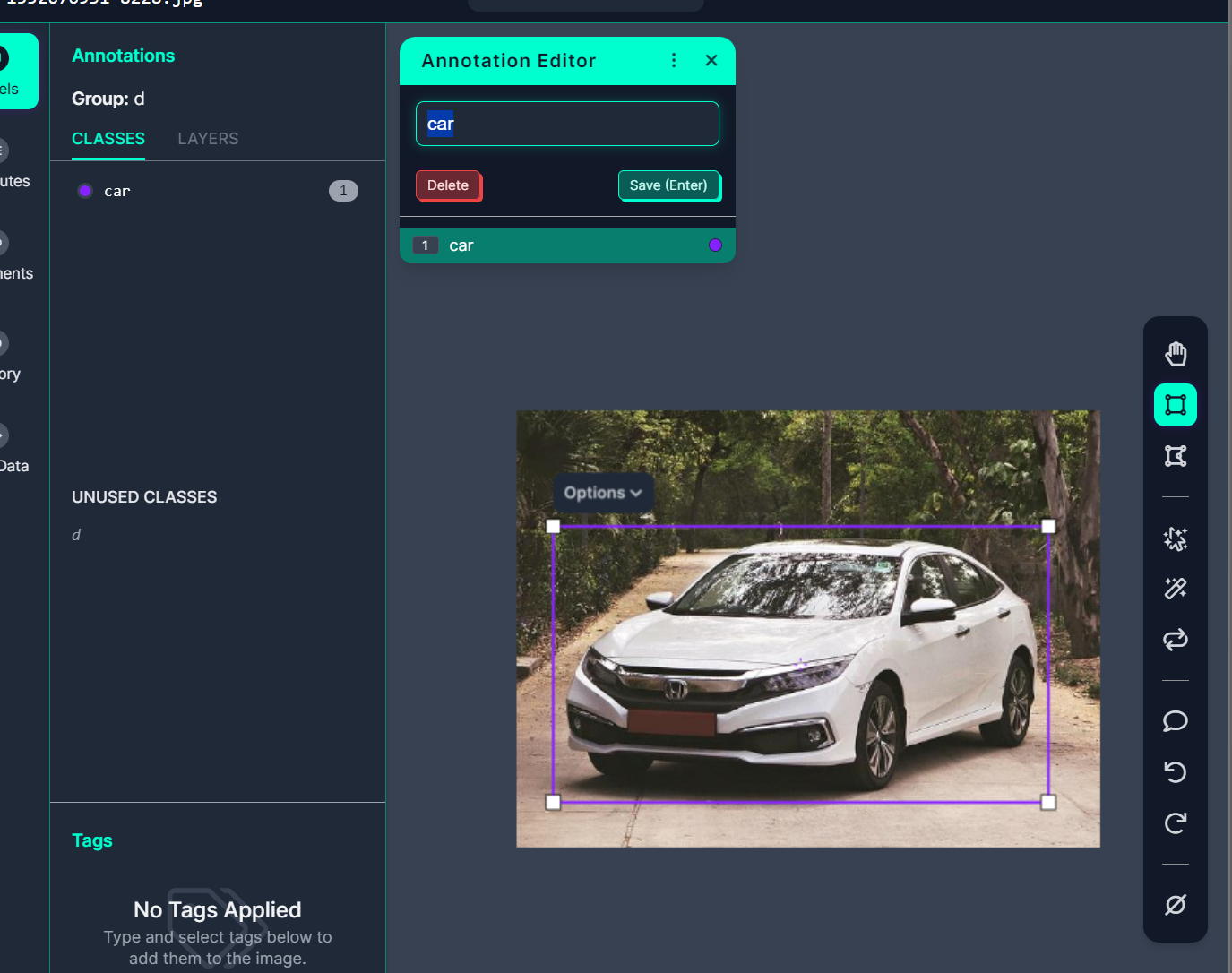


Figura 2 – Configurare set de date

Apoi preluăm datele din roboflow, care vor fi transferate în proiectul nostru.

from roboflow import Roboflow

rf = Roboflow(api\_key="wzH2qwv1bksSA4rBQj8q")

project = rf.workspace("catalin-rwg2n").project("lab6-c7zql")

version = project.version(2)

dataset = version.download("yolov8")

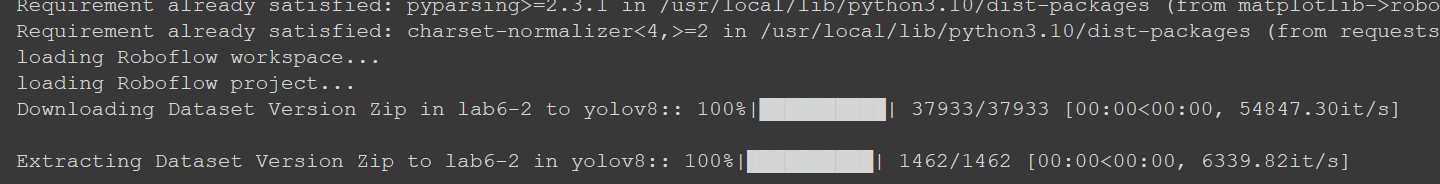


Figura 2 – Instalarea setului de date

În continuare antrenăm modelul YOLO pentru detecția vehiculelor folosind setul de date descărcat.

!yolo task=detect mode=train model=yolov8s.pt data={dataset.location}/data.yaml epochs=25 imgsz=800 plots=True

**Task=detekt** – specificăm că va avea loc detecția obiectelor

**Mode-train** – specificăm ca are loc antrenarea modelului

**Data** – specificăm calea către fișierul de configurare yaml, care conține informații despre setul de date utilizat pentru antrenare.

**Epochs=25** – specificăm numărul de epoci de antrenamente, adică de treceri complete ale setului de date de antrenamente.

**Imgsz=800** – specificăm dimensiunea imaginilor de intrare folosite în timpul antrenamentului (800x800).

**Plots=True** – specificăm să se afișeze grafice pentru monitorizarea procesului antrenamentului.

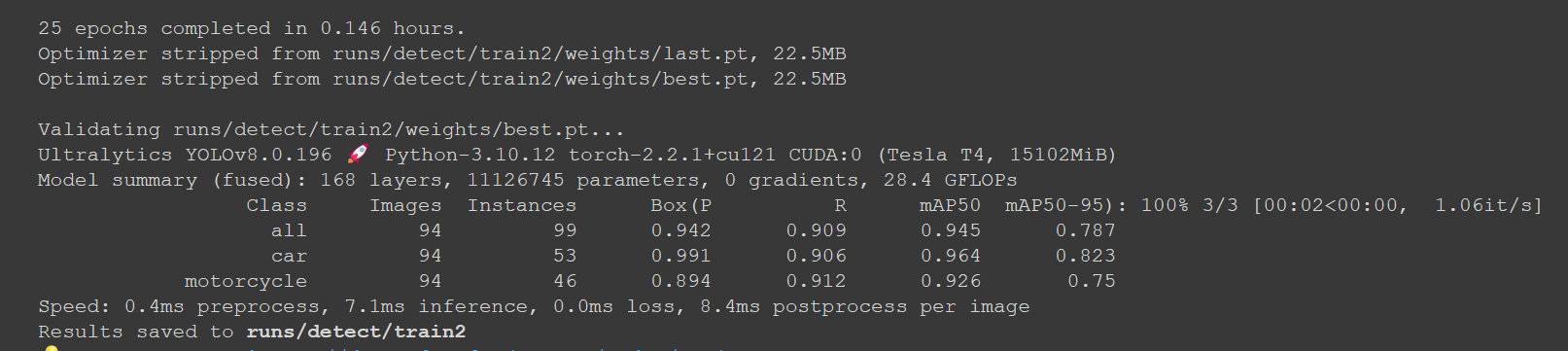


Figura 3- Afișarea rezultatului după antrenarea monelului cu 25 de epoci.

Observăm ca am obținut o precizie generală de 0.945. Precizia la mașini și motociclete este aproximativ simetrică.

Rezultatele obținute sunt salvate în directoriul runs/detect/train2.

În continuarea afișăm matricea de confuzie pentru a evalua performanța modelului antrenat în detecția vehiculelor.

Image(filename=f'{HOME}/runs/detect/train2/confusion\_matrix.png', width=600)

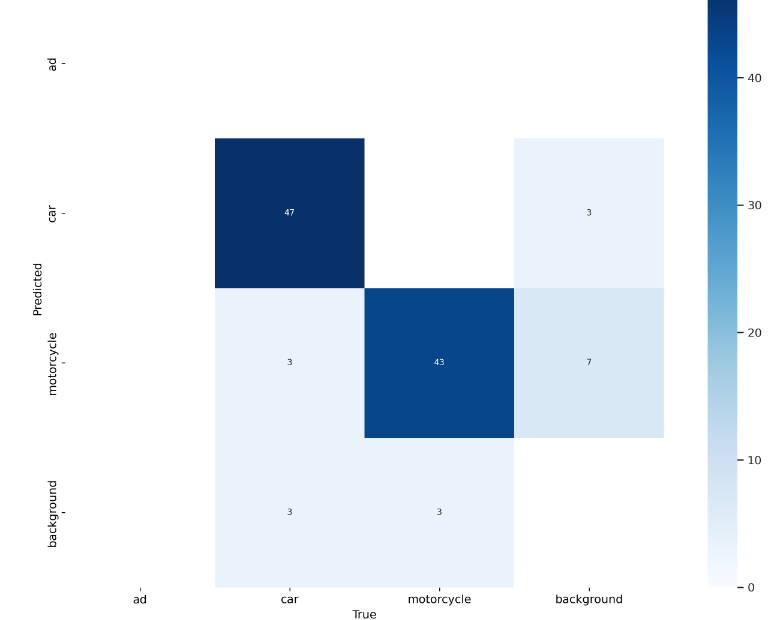


Figura 4- Matricea de confuzie

**Car** – modelul a prezis corect car de 47 de ori, dar a confundat cu motorcycle si backgorund de 3 ori.

**Motorcycle** – modelul a prezis corect motocycle de 43 de ori, dar a confundat cu background de 3 ori.

În continuare voi afișa o serie de grafice care arată evoluția diferitelor metrici de performanță și valori de pierdere (loss) ale modelului de învățare automată pe parcursul antrenamentului. Fiecare grafic urmărește o anumită metrică sau pierdere, comparând valoarea calculată direct (linia albastră continuă) cu o medie netezită (linia portocalie punctată) pentru a arăta tendințele generale.

Image(filename=f'{HOME}/runs/detect/train2/results.png', width=600)

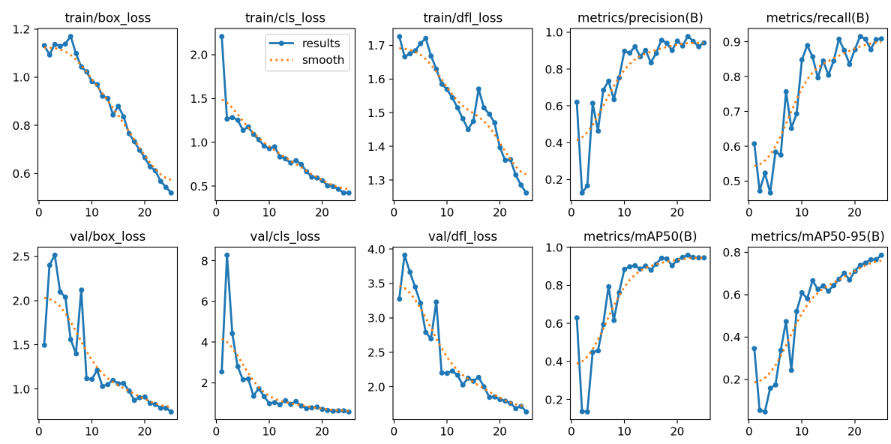


Figura 5 – Afișare metricilor de performanță

**train/box\_loss și val/box\_loss:** Arată pierderea legată de predicția bounding boxes (cadrele delimitatoare) pentru seturile de date de antrenament și validare, respectiv. Scăderea acestei valori indică o îmbunătățire a abilității modelului de a localiza corect obiectele.

**train/cls\_loss și val/cls\_loss:** Reprezintă pierderea asociată cu clasificarea corectă a obiectelor pentru seturile de date de antrenament și validare. O valoare în scădere arată că modelul devine mai bun în clasificarea corectă a tipurilor de obiecte.

**train/df\_loss și val/df\_loss:** Acestea indică o pierdere specifică modelului.

**metrics/precision(B) și metrics/recall(B):** Afișează evoluția preciziei și a ratei de recuperare (recall) pentru setul de antrenament.

**metrics/mAP50(B) și metrics/mAP50-95(B):** performanța generală a modelului.

Codul de mai jost, va realiza o vizualizare a modelului.

Image(filename=f'{HOME}/runs/detect/train2/val\_batch0\_pred.jpg', width=600)

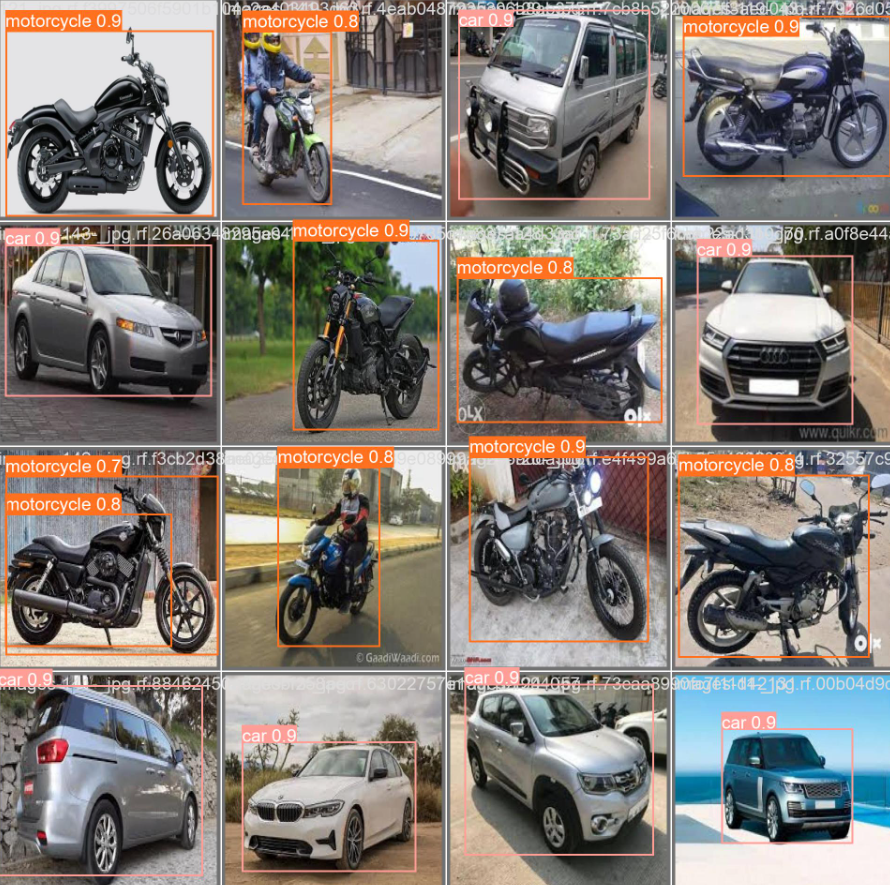


Figura 6 – Vizualizare model

În continuare am realizat validarea modelului. Aici se va realiza detecția obiectelor pe setul de date de validare. Specificăm calea către model și calea către fișierul care conține informații despre setul de date.

!yolo task=detect mode=val model={HOME}/runs/detect/train2/weights/best.pt data={dataset.location}/data.yaml

Apoi realizăm predicția obiectelor folosind modelul antrenat. Specificăm că are loc detectarea obiectelor și predicția obiectelor pe imagini de test. La fel specificăm pragul de confidență pentru detecție, adică doar obiectele cu o confidență mai mare vor fi raportate. În final,salvăm rezultatele predicției.

!yolo task=detect mode=predict model={HOME}/runs/detect/train2/weights/best.pt conf=0.25 source={dataset.location}/test/images save=True

În finaș afisăm rezultatele obținute.

import glob

from IPython.display import Image, display

for image\_path in glob.glob(f'{HOME}/runs/detect/predict/\*.jpg')[:4]:

      display(Image(filename=image\_path, width=600))

      print("\n")



Figura 7 – Afișare rezultat

**Concluzie**

Lucrarea de laborator prezintă procesul de antrenare și evaluare a unui model YOLO pentru detecția vehiculelor. Modelul antrenat a obținut o precizie generală de 0.945 pe setul de date de validare. Matricea de confuzie a evidențiat confuziile între clase și erorile de clasificare. Graficele de performanță au monitorizat evoluția metricilor de performanță și pierderilor în timpul antrenamentului. Rezultatele antrenamentului și predicția obiectelor au fost vizualizate, oferind o reprezentare vizuală a performanței modelului în detectarea vehiculelor.