5. Predicţia unei reţele neuronale de tip detector de obiecte

În acest laborator vom parcurge pașii necesari pentru a încărca un model folosind *torch*, vom utiliza *Git* pentru a descărca de pe un *repository* un *checkpoint* pentru un model care realizează detecția de obiecte în imagini și vom rula pasul de predicție pe un set de date. Pentru a ilustra aceste concepte, vom folosi modelul *YoloV5*, care este unul dintre cele mai bune detectoare de obiecte la ora actuală.

Așadar vom descărca un fișier cu ponderile preantrenate pentru un model de YoloV5, vom încărca acest model și vom realiza predicții folosind anumite imagini pentru a ilustra modul de funcționare al unui detector de obiecte.

5.1 Noțiuni teoretice

YoloV5 (`https://github.com/ultralytics/yolov5`) este unul dintre cele mai bune detectoare de obiecte, antrenat pe setul de date COCO (`https://cocodataset.org/#home`). Acesta vine în mai multe variante, respectiv yolov5s, yolov5m, yolov5l și yolov5x. Fiecare dintre aceste variații diferă de celelalte prin numărul de straturi din rețea, cel mai simplu model fiind yolov5s, iar cel mai complex yolov5x. Evident, pe măsură ce folosim un model mai complicat, crește precizia modelului, însă scade viteza de execuție.

5.2 Module folosite

1. *requests* - modul folosit pentru descărcarea modelului yolo.
2. *tqdm* - modul folosit pentru progress bars în consola Python.
3. *pyyaml* - modul folosit pentru parsare de fișiere yaml (markup language, similar cu XML sau Json).
4. *seaborn* - modul folosit pentru vizualizarea datelor, bazat pe matplotlib.

5.3 Aplicația completă

```
1 import torch
2 import cv2
3
4 # Definirea culorilor cu care vom desena detectiile pe imagini
5 colormap = [
```

```
6       (255, 0, 0),
7       (0, 128, 200),
8       (200, 128, 128),
9       (0, 0, 255)
10  ]
11
12  if __name__ == '__main__':
13      # Incarcarea modelului yolov5 folosind torch.hub
14      model = torch.hub.load('ultralytics/yolov5', 'yolov5s')
15
16      # Acest model suporta predictia folosind imagini pe care le poate↩
            descarca de pe internet
17      img = 'https://ultralytics.com/images/zidane.jpg'
18
19      # Predictia
20      results = model(img)
21
22      # Imaginea incarcata
23      image = results.imgs[0]
24
25      # Imaginile OpenCV sunt incarcate in BGR, iar imaginea incarcata ↩
            de pe internet este incarcata RGB.
26      # Deci trebuie realizata conversia RGB - BGR
27      image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
28
29      # Detectiile din yolo au mai multe proprietati. Cea care ne ↩
            intereseaza pe noi exte xyxy:
30      # Rezultatul YoLo este o lista de dreptunghiuri care ar trebui sa↩
            incadreze obiectele de interes
31      # din imagini. Aceste dreptunghiuri sunt reprezentate in mai ↩
            multe moduri:
32      # - xywh: coordonatele (x,y) ale centrului dreptunghiului, ↩
            respectiv latimea si inaltimea
33      # - xyxy: coordonatele (x, y) ale coltului din stanga sus al ↩
            dreptunghiului,
34      #   respectiv coordonatele (x, y) ale coltului din dreapta jos
35      #
36      # Pe langa aceste dreptunghiuri, avem de asemenea confidenta ↩
            fiecaruia (cat de <<sigur>> este
37      # detectorul ca respectivul bounding box face parte dintr-o ↩
```

```
        anumita clasa) si clasa fiecarui
38      # bounding box (din ce categorie face parte obiectul/dreptunghiul↩
            /bounding box detectat).
39      #
40      # Pe langa aceste liste de detectii (x1, y1, x2, y2, conf, cls), ↩
            in obiectul results avem
41      # si numele asociate fiecarei clase, in functie de cum a fost ↩
            antrenat modelul.
42      # Fiind antrenat pe COCO, acest model are 80 de clase.
43      for i, detection in enumerate(results.xyxy[0]):
44          # detection = tensor (x1,y1,x2,y2,conf,cls)
45          cv2.rectangle(image,
46                        (int(detection[0]), int(detection[1])), (int(↩
                              detection[2]), int(detection[3])),
47                        color=colormap[i % len(colormap)],
48                        thickness=2)
49          cv2.putText(image,
50                      results.names[int(detection[-1])],  # Printeaza ↩
                            numele clasei detectate
51                      (int(detection[0]), int(detection[1])),
52                      cv2.FONT_HERSHEY_SIMPLEX, 0.8, colormap[i % len(↩
                            colormap)])
53      cv2.imshow("Test", image)
54      cv2.waitKey()
55
56      # Afisarea statisticilor
57      results.print()
```

Fiind vorba doar despre încărcarea unui model, iar acest model de YoLo este *state of the art*, funcționalitățile necesare pentru încărcare, respectiv predicție sunt implementate deja în *torch*. Prin urmare, nu avem încă nevoie să implementăm arhitectura modelului sau metodologia de antrenare.

Pornind de la acest exemplu, să se realizeze:

1. Schimbarea imaginii pe care se realizează predicția cu una la alegere, aflată pe harddisk.

2. Implementarea funcționalității care realizează afișarea bounding boxes, însă folosind detecțiile **în formatul xywh**.

3. Schimbarea colorării bounding boxes, în funcție de clasă. De exemplu, persoanele vor fi încadrate folosind culoarea albastru, ș.a.m.d. Culorile pot fi alese aleator. Să

se ilustreze această funcționalitate, rulând inferența (detecția) pe imagini care conțin multiple instanțe de obiecte de interes.

4. Implementarea unei metode care primește ca parametru un path către un folder (care conține mai multe imagini) și realizează predicția pe imaginile aflate în acel folder. Predicțiile trebuie să fie desenate în imagini.