Détection d'objets en temps réel avec OpenCV (DNN) + SSD

Portfolio de projet

Technologies : Python, OpenCV (cv2.dnn), SSD MobileNet, COCO

Résumé

Ce projet implémente une **détection d'objets en direct via webcam** en s'appuyant sur le module `cv2.dnn` d'OpenCV. Le programme charge un modèle de type **SSD** (poids `.pb` et configuration `.pbtxt`) et les classes **COCO** (`coco.names`). À chaque image vidéo, le modèle retourne les objets détectés, leur **score de confiance** et leur **boîte englobante** affichés à l'écran.

1 Contexte et objectifs

1.1 Problématique

La vision par ordinateur permet d'automatiser l'identification d'objets dans une scène (personne, voiture, téléphone, etc.). L'objectif ici est de réaliser une solution **simple, portable et temps réel** à partir d'OpenCV, sans framework lourd.

1.2 Objectifs

— Détecter des objets en **temps réel** depuis une webcam.

— Afficher **labels**, **bounding boxes** et **confiance**.

— Garder une architecture de projet claire (poids/modèle/classes séparés).

2 Architecture du projet

2.1 Fichiers principaux

Fichier	Rôle
`main.py`	Script principal (capture webcam, inférence, affichage).
`coco.names`	Liste des classes (COCO) utilisées pour nommer les objets.
`ssd.pbtxt`	Configuration du réseau (graph/architecture).
`file.pb`	Poids du modèle (réseau entraîné).

TABLE 1 – Structure des fichiers du projet.

2.2 Pipeline de fonctionnement

1. Chargement des classes COCO (`coco.names`).
2. Chargement du modèle SSD via `cv2.dnn_DetectionModel`.
3. Paramétrage des entrées : taille, normalisation, moyenne, swap RB.

4. Capture vidéo avec `cv2.VideoCapture(0)`.

5. Pour chaque frame : **détection** → **dessin** → **affichage**.

3 Choix techniques

3.1 Pourquoi OpenCV DNN ?

— Facile à déployer (une seule dépendance principale).

— Compatible CPU (pas besoin de GPU pour une démo).

— API claire pour l'inférence (`detect`, `setInputSize`, etc.).

3.2 Pré-traitements appliqués

Le modèle est configuré avec :

— `InputSize = 320x320` : compromis vitesse/précision.

— `InputScale = 1/127.5` et `Mean = (127.5,127.5,127.5)` : normalisation.

— `SwapRB = True` : conversion BGR → RGB.

— Seuil de confiance : `thres = 0.45`.

4 Implémentation (extraits)

4.1 Chargement du modèle et des classes

Listing 1 – Chargement des classes COCO et du modèle SSD.

```
class_file = "coco.names"
config_path = "ssd.pbtxt"
weights_path = "file.pb"

with open(class_file, "rt") as f:
    class_names = f.read().rstrip("\n").split("\n")

net = cv2.dnn_DetectionModel(weights_path, config_path)
net.setInputSize(320, 320)
net.setInputScale(1.0 / 127.5)
net.setInputMean((127.5, 127.5, 127.5))
net.setInputSwapRB(True)
```

4.2 Boucle temps réel et affichage

Listing 2 – Inférence sur chaque frame et rendu.

```
cap = cv2.VideoCapture(0)

thres = 0.45
while True:
    success, frame = cap.read()
    if not success:
        break

```

```
    class_ids, confs, bbox = net.detect(frame, confThreshold=thres)

    if class_ids is not None and len(class_ids) > 0:
        for class_id, confidence, box in zip(class_ids.flatten(), confs.
            flatten(), bbox):
            cv2.rectangle(frame, box, (0, 255, 0), 2)

            label = class_names[class_id - 1].upper()
            conf_txt = f"{confidence * 100:.2f}%"

            cv2.putText(frame, label, (box[0] + 10, box[1] + 30),
                        cv2.FONT_HERSHEY_COMPLEX, 1, (0, 255, 0), 2)
            cv2.putText(frame, conf_txt, (box[0] + 10, box[1] + 65),
                        cv2.FONT_HERSHEY_COMPLEX, 0.8, (0, 255, 0), 2)

    cv2.imshow("Object Detection (OpenCV)", frame)
    if cv2.waitKey(1) & 0xFF == ord("q"):
        break
```

5 Résultats

5.1 Ce que l'application affiche

— Boîte englobante (rectangle) autour de l'objet détecté.

— Nom de la classe (en majuscules).

— Score de confiance en %.

5.2 Performances observables (qualitatives)

Les performances dépendent principalement de :

— la résolution caméra,

— la puissance CPU,

— le seuil de confiance (`0.45`) et la taille d'entrée (`320x320`).

À documenter dans ton portfolio : ajoute une capture d'écran et/ou une courte vidéo de démonstration, et note une estimation (FPS approximatif) sur ta machine.

6 Limites

— Sensibilité aux conditions de lumière et aux objets petits/lointains.

— Détections parfois instables (faux positifs / faux négatifs).

— Pas de suivi d'objets entre images (tracking) : chaque frame est indépendante.

7 Améliorations possibles

— Ajouter un **compteur FPS** et l'afficher à l'écran.

— Mettre en place un **tracking** (SORT/DeepSORT) pour stabiliser les boîtes.

— Optimiser l'inférence : réduire la résolution, activer backend/target (si disponible).

— Ajouter une fonctionnalité : sauvegarde vidéo, logs, ou export des détections.

8 Guide d'exécution

8.1 Prérequis

— Python 3.x

— OpenCV : `pip install opencv-python`

8.2 Lancer le projet

Place les fichiers `main.py`, `coco.names`, `ssd.pbtxt`, `file.pb` dans le même dossier, puis :

`python main.py`

Appuie sur `q` pour quitter.
