Rapport: Reconnaissance d'Objets en Temps Réel

Yassine ZAOUI

Introduction

Ce test vise à développer un système de détection d'objets en temps réel. L'objectif principal est de détecter et d'identifier un ensemble restreint d'objets courants (pour notre cas: une tasse, un téléphone ,une bouteille ,une fourchette et une cuillère) à partir du flux vidéo capturé par la caméra de l'ordinateur. Pour ce faire, nous déployons les bibliothèques **MediaPipe** pour la détection d'objets et **OpenCV** pour la capture, le traitement et l'affichage des flux vidéo.

Choix des Technologies

Le développement de ce projet repose principalement sur les technologies suivantes :

- **Python** : Nous utilisons Python comme langage de programmation en raison de sa simplicité, sa lisibilité, et la richesse de ses bibliothèques pour les applications en Intelligence artificielle..
- **OpenCV** : Bibliothèque de vision par ordinateur utilisée pour la capture vidéo, le traitement d'images, et l'affichage en temps réel.
- **MediaPipe** : bibliothèque de machine learning qui intègre des modèles pré-entraînés pour des tâches telles que la détection d'objets..
- **mediapipe.tasks** : Ce module spécifique de MediaPipe permet d'accéder à des fonctionnalités avancées pour personnaliser les tâches de vision par ordinateur.
  - **mediapipe.tasks.python** : Cette sous-bibliothèque fournit les options de base (`BaseOptions`) pour configurer les modèles MediaPipe à l'aide de fichiers TensorFlow Lite (`.tflite`).
  - **mediapipe.tasks.python.vision** : Cette sous-bibliothèque permet d'effectuer des tâches de vision par ordinateur, telles que la détection d'objets, via la classe `ObjectDetector`. Elle facilite l'utilisation des modèles de détection d'objets pour extraire les cadres englobants (bounding boxes) et les scores de probabilité des objets détectés.

Nous avons aussi utilisé les bibilothèques **NumPy** et **time** pour la manipulation des images en tant que tableau NumPy et pour calculer les temps de latence.

Classes d'objets détectés

Le but est de détecter un petit ensemble d'objets courants à l'instar d'une tasse ou un téléphone. Ainsi, nous choisissons les 5 classes suivantes : **une tasse, un téléphone, une bouteille, une fourchette et une cuillère**

Prétraitement des Données

- **Capture vidéo** : La capture vidéo est réalisée à l'aide de la fonction `cv2.VideoCapture(0)` d'OpenCV, qui utilise la caméra par défaut de l'ordinateur.
- **Conversion de format** : Avant d'être transmis au modèle MediaPipe, chaque frame capturé est converti du format BGR (utilisé par OpenCV) au format RGB (requis par MediaPipe pour le traitement).
- **Encapsulation de l'image** : Bien que chaque frame soit initialement un tableau NumPy, il doit être encapsulé en un objet `mp.Image` avec le format `sRGB` pour être compatible avec les modèles MediaPipe.
- **Détection en temps réel** : La détection des objets est effectuée à chaque frame, et les résultats sont superposés en temps réel sur la vidéo à l'aide de la fonction `cv2.rectangle()` pour dessiner des cadres autour des objets détectés.

Choix du modèle

La bibliothèque **MediaPipe** offre 3 modèles pré-entraînés pour la détection d'objets à partir d'un flux vidéo en temps réel, à savoir **EffectiveDet-Lite0**, **EffectiveDet-Lite2** et **MobileNetV2 SSD**. Le modèle **EffectiveDet-Lite0** est en général recommandé vu qu'il offre un bon compromis entre latence et justesse.
Nous avons essayé ces 3 modèles et effectivement le plus adéquat pour notre tâche est **EffectiveDet-Lite0**: en se basant sur plusierus vidéos .mp4 enregistrés à partir des essais, nous donnons Le tableau ci-contre qui illustre la performance des 3 modèles pour une quantification **int8**:

Modèle	**EfficientDet-lite0**	**EfficientDet-lite2**	**MobileNetV2 SSD**
Latence approximative	30-40 ms	100-110 ms	35-45 ms
Score de probabilité lors d'une détection	0.5-0.75	0.5-0.85	0.5-0.65

En fait, nos essais montrent qu'il est à la fois plus précis et légèrement plus rapide que **MobileNetV2 SSD**.De plus, bien qu'il soit moins précis que **EffectiveDet-Lite2**, il est 3 fois plus rapide. Ainsi, nous optons sûrement pour **EffectiveDet-Lite0**. Il faut souligner que nous nous sommes intéressé aux modèles quantifiés en **int8**, ce qui réduit la taille du modèle et accélère les inférences (réduit le temps de latence) tout en maintenant une précision raisonnable. La quantification **int8** permet d'optimiser le modèle pour des applications en temps réel sur des dispositifs à faible puissance, tels que les ordinateurs portables.

Implémentation

Le système fonctionne de la manière suivante :

1. **Chargement du modèle** : Le modèle **EffectiveDet-Lite0** est chargé à partir d'un fichier `.tflite` en utilisant les options de MediaPipe.

2. **Capture vidéo en continu** : Un flux vidéo est capturé en temps réel depuis la caméra de l'ordinateur.

3. **Détection d'objets** : Chaque frame capturé est transmis au modèle pour détecter les objets spécifiés.

4. **Annotation des résultats** : Les objets détectés sont entourés de cadres colorés avec leur nom et un score de confiance affiché sur la vidéo en direct. Le temps de latence est également affiché.

5. **Enregistrement vidéo** : Le flux vidéo annoté est enregistré dans un fichier `output_detection.mp4`.

Évaluation des Performances

- **Précision** : Le modèle pré-entraîné EffectiveDet-Lite0 a démontré une précision suffisante à un certain degrès pour détecter les objets d'intérêt dans un environnement contrôlé. En effet, nous remarquons que le score en général est compris entre 0.5 et 0.75. Ceci est satisfaisant bien que ce score soit influencé par la nature de l'arrière-plan et la position des objets à détecter ce qui donnent pour des cas bien particuliers des faux positifs (une partie d'une cuillière ou une fourchette est classé comme un téléphone).

- **Latence** : La latence moyenne observée pour chaque détection est d'environ 30-40 ms, ce qui donne une très bonne fluidité pour une détection en temps réel.

- **Efficacité** : Le système fonctionne de manière fluide sur une machine à ressources limité à l'instar d'un ordinateur portable.

Pistes d'amélioration

- Mettre en évidence une fonction pour ajouter artificiellement du bruit gaussien avec des niveaux de bruit différents, afin d'évaluer la robustesse du modèle de détection face à des données bruitées . Cela permettrait de tester la tolérance du modèle aux variations imprévues dans l'environnement réel (éclairage faible/intense, objets en mouvement rapide, conditions météorologiques, etc...).

- Adapter le modèle pré-entrainé **EffectiveDet-Lite0** en utilisant **MediaPipe Model Maker** pour personnaliser ce modèle pour la détection d'objets spécifiques (comme une tasse, un téléphone, une bouteille, une fourchette et une cuillière). MediaPipe Model Maker utilise l'apprentissage par transfert, ce qui permet de bénéficier de la logique et des connaissances déjà acquises par le modèle tout en ré-entrainant

uniquement les dernières couches. En supprimant ces couches et en les reconstruisant avec de nouvelles données, le modèle peut être adapté de manière plus ciblée. Il serait aussi possible de personnaliser davantage l'entraînement en ajustant les hyperparamètres comme le taux d'apprentissage, le nombre d'époques, et en appliquant des techniques d'augmentation de données (inclure des images bruitées aussi peut-être) pour améliorer la généralisation du modèle.

- Envisager d'explorer des modèles plus avancés pour une précision accrue, même si cela peut entraîner une latence plus importante. L'idée serait de tester si des modèles ou des architectures encore plus puissantes telles que **YOLOv4** ou **SSD** pourraient offrir une meilleure précision, même si cela induit une latence plus élevée. Ce compromis pourrait être acceptable si la latence reste suffisamment faible pour garantir une détection fluide et en temps réel.

Conclusion

Nous avons concçu un système efficace pourtant simple pour la détection de 5 classes d'objets en temps réel via Python et principalement ses bibilothèques MediaPipe et OpenCV. Le modèle EffectiveDet-Lite0 a été choisi pour son équilibre entre exigence ressources et performance en temps réel.