2024 l'odyssée de l'IA

améliorer la détection d'objets : intégration de techniques avancées

Brian Tristam Williams (Elektor)

Après avoir réussi à déployer le principe fondamental de la détection d'objets sur un Raspberry Pi sans tête (*headless*), nous poursuivons notre aventure en nous concentrant sur l'amélioration de cette technologie. Dans cet épisode nous abordons les améliorations techniques visant à accroître la précision et l'efficacité, ainsi que l'intégration de capteurs supplémentaires afin d'améliorer le fonctionnement dans diverses conditions environnementales.

Après avoir rencontré des difficultés pour détecter des objets et du texte sur de vieilles vidéos d'archives, j'ai dû chercher des moyens d'améliorer les algorithmes de détection.
Dans mon exemple, les vidéos entrelacées et de faible résolution sur lesquelles je travaillais ont probablement semé la confusion. Les différences entre les demi-trames alternées sont susceptibles de perturber le modèle de détection, conçu pour analyser une image (composée de deux demi-trames) (**figure 1**). Il est essentiel que j'examine le désentrelacement de l'ancienne vidéo capturée avant de l'intégrer dans le flux de travail.

Amélioration de la détection

Malgré les défis de niche que j'ai rencontrés, cela ne signifie pas que nous ne pouvons pas améliorer la détection avec quelques modifications. Voici ce que j'ai trouvé utile en travaillant avec une vidéo numérique non entrelacée provenant du Raspberry Pi Camera Module 3 :

Seuil dynamique : pour pallier les variations de l'éclairage et des distances, j'ai établi un seuil de confiance adaptatif. Cette méthode permet d'ajuster les niveaux de confiance de la détection en temps réel en fonction de la luminosité moyenne détectée par la caméra, grâce à un algorithme de calcul de luminosité simple :

```
def adjust_threshold(lux):
    base_threshold = 0.5  # base confidence level
    if lux < 50:  # low light conditions
        return base_threshold - 0.1
    elif lux > 500:  # very bright conditions
        return base_threshold + 0.1
    return base_threshold
```

Figure 1. La vidéo PAL et NTSC utilise une méthode entrelacée pour dessiner les lignes de balayage, ce qui affecte les images complètes en cas de mouvement.

Intégration du capteur de luminosité : En parlant d'éclairage, bien que la caméra puisse fonctionner comme un capteur de luminosité, mes essais passés d'ajuster les nombreux paramètres du module caméra pour obtenir des photos et vidéos « pafaits », je sais que la relation entre la luminosité du signal vidéo de sortie et la luminosité réelle à l'extérieur sera ténue – c'est juste une variable confondante de plus dont je ne veux pas gérer la frustration, et j'ai donc opté pour l'utilisation d'un capteur à cet effet.
J'ai choisi le capteur de lumière TLS2561 (**figure 3**) pour fournir des données en temps réel au Raspberry Pi, qui ajuste alors dynamiquement l'exposition de la caméra ainsi les paramètres de traitement. Cela permet d'assurer une qualité d'image optimale pour les algorithmes de détection dans des conditions d'éclairage fluctuantes :

```
lux = read_lux_sensor()
camera.set_exposure(calculate_exposure(lux))
```

Encore une fois, je n'ai pas eu à écrire ces fonctions moi-même, car il y existe une excellente bibliothèque *Adafruit-TSL2561* disponible [1]. L'installation est simple via le terminal :

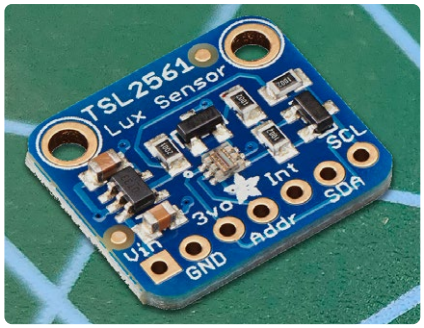

Figure 2. Le capteur de luminosité TSL2561 sur un module Adafruit.

```
sudo apt-get update
sudo apt-get install python3-smbus
pip3 install Adafruit-TSL2561
```

Ce code Python vous permet ensuite de lire et d'éditer les valeurs du capteur :

```
import time
from Adafruit_TSL2561 import TSL2561

# Initialize the sensor
tsl = TSL2561()

while True:
    lux = tsl.calculate_lux()
    print("Current Lux: ", lux)
    time.sleep(1)  # Delay for 1 second
```

De nos jours, les bibliothèques disponibles sur internet facilitent grandement les choses. À l'époque, il fallait consulter la fiche technique de chaque nouveau capteur acheté et créer ses propres fonctions ou sous-programmes.

Amélioration du traitement des images : En intégrant la fonction `cv2.GaussianBlur()` avant la détection d'objet, j'ai pu réduire l'impact du bruit et du grain dans les images vidéo à faible luminosité. J'ai trouvé cela particulièrement utile avec le Raspberry Pi Camera Module 3 NoIR (**figure 2**, à droite) que j'ai testé pour la surveillance nocturne. Ce modèle de caméra ne dispose pas de filtre infrarouge et peut donc capter bien les éclairages infrarouges. Cependant, son efficacité dépend toujours de la distance entre votre éclairage infrarouge et le sujet, et parfois les objets distants posent un problème.
L'utilisation de cette fonction a permis au modèle de traiter les images avec beaucoup moins de bruit, bien que cela ait affecté la netteté de l'image d'entrée. Cependant, lorsque vous essayez de détecter des objets à grande échelle sans avoir besoins de détails fins tels que la lecture de plaques d'immatriculation, la netteté des « humains » détectés importe peu au modèle – c'est à l'utilisateur de vérifier les détails du flux de la caméra lorsqu'une alerte est déclenchée. L'appel de la fonction se fait en une seule ligne de code :

```
frame = cv2.GaussianBlur(frame, (5, 5), 0)
```

Prise en charge de plusieurs modèles : Au début, j'ai opté pour un seul modèle après en avoir testé plusieurs, mais il était un peu frustrant de constater qu'aucun ne convenait à toutes les configurations et à tous les scénarios. Puis j'ai réalisé, lentement, qu'il était possible de basculer entre différents modèles à la volée.
L'intégration d'un modèle de détection secondaire spécialisé pour des cibles spécifiques améliore la précision. Par exemple, l'intégration d'un modèle YOLO rationalisé augmente la robustesse de la détection des véhicules et des piétons. Le passage d'un modèle à l'autre en fonction du contexte de la scène est géré comme dans cet exemple :

```
if scene == 'urban':
    model.load('yolo_city.tflite')
else:
    model.load('tensorflow_lite_default.tflite')
```

Réduction de la latence : Malheureusement, l'augmentation de la complexité de la détection a entrainé une augmentation de la latence. J'ai essayé d'implémenter le traitement asynchrone des modèles, ce qui a permis d'accélérer les choses en termes de chiffres bruts, même si l'amélioration n'était pas évidente à mes yeux. Néanmoins, toute amélioration est bienvenue. Voici donc comment l'essayer :

```
from concurrent.futures import ThreadPoolExecutor

with ThreadPoolExecutor() as executor:
    future = executor.submit(process_frame, frame)
    result = future.result()
```

Nous voyons ici l'utilité de la fonction `ThreadPoolExecutor()` du module `concurrent.futures`, qui fait partie de la bibliothèque standard de Python. Cette méthode permet de gérer l'exécution asynchrone de tâches dans des threads distincts, ce qui peut s'avérer particulièrement utile pour des applications comme le traitement vidéo en temps réel où la réactivité est essentielle.
Après la première ligne, où l'on intègre la fonction, les trois lignes suivantes font respectivement ce qui suit :

1. Créer une instance de `ThreadPoolExecutor`, qui gère un pool de threads pour l'exécution asynchrone des appels. L'instruction `with` garantit que les ressources de l'exécuteur sont correctement gérées, en fermant automatiquement l'exécuteur lorsqu'il n'est plus nécessaire.
2. La méthode `submit()` planifie l'exécution de la fonction `process_frame()` en lui passant `frame` comme argument. La fonction est exécutée dans un thread distinct géré par l'exécuteur. `submit()` renvoie un objet `future`, qui représente le résultat potentiel de l'appel de la fonction.
3. Bloquer le thread principal jusqu'à ce que la fonction `process_frame()` s'exécute et renvoie un résultat. La méthode `result()` récupère le résultat de 'exécution de la fonction. Si une exception est soulevée durant l'exécution, celle-ci sera soulevée à nouveau ici.

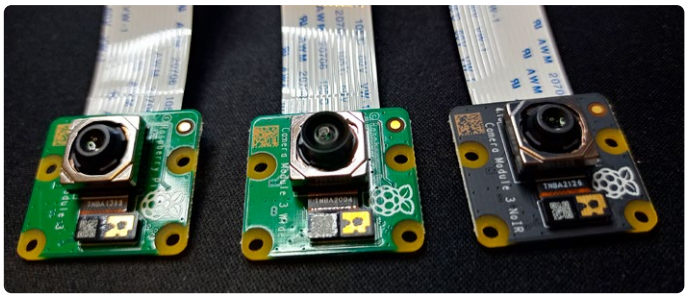

Figure 3. Outre la Camera Module 3 large et ordinaire, j'ai essayé la version NoIR pour sa capacité à bénéficier d'un éclairage infrarouge la nuit.

En déchargeant les tâches gourmandes en ressources sur des threads séparés, le thread principal du programme peut rester réactif. Ceci est particulièrement essentiel dans les applications graphiques ou en temps réel, où une latence élevée peut nuire à l'expérience de l'utilisateur.
Le déploiement de ce système au sein d'un réseau de surveillance domestique a prouvé sa capacité à distinguer une personne d'un chien ou d'un arbre. Toutefois, il n'atteint pas le niveau de sophistication requis pour la détection de visages comme le ferait un système de reconnaissance faciale.

Pour aller plus loin

Outre le désentrelacement et le prétraitement nécessaires pour avancer sur mon projet de traitement de vieilles vidéos, de nombreuses autres possibilités s'offrent avec TensorFlow Lite, et je ne sais pas laquelle sera la plus intéressante. N'hésitez donc pas à me contacter pour me faire part de toute piste que vous aimeriez explorer ensemble !
Voici quelques tâches que nous pouvons explorer :

- **Traitement d'images et de vidéos :** Je continue à travailler sur des applications de traitement avancé d'images et de vidéos, comme la segmentation d'images, où le modèle identifie différents objets dans l'image et les sépare de l'arrière-plan, ou le transfert de style, qui permet d'appliquer le style d'une image au contenu d'une autre.
- **Détection et classification d'objets en temps réel :** TensorFlow Lite exécute des modèles qui détectent et classifient des objets en temps réel, ce qui est utile pour des applications telles que les caméras de sécurité, le contrôle qualité automatisé et la surveillance de la faune. Des modèles pré-entraînés tels que MobileNet peuvent être adaptés et déployés sur des appareils pour identifier des objets efficacement.
- **Reconnaissance de la parole et traitement audio :** TensorFlow Lite gère des modèles de reconnaissance vocale, ce qui permet de créer des applications à commande vocale, de convertir la parole en texte et d'effectuer d'autres tâches de traitement audio. Pour moi, cela pourrait être un moyen de préserver des histoires oubliées des archive, mais cela permet également de développer des commandes mains libres ou des outils d'assistance pour personnes handicapées.
- **Traitement du langage naturel (NLP) :** Le framework prend en charge des modèles NLP légers qui peuvent effectuer des tâches telles que l'analyse des sentiments, la détection du langage, ou même alimenter de simples chatbots. C'est particulièrement utile dans les applications nécessitant une interaction et traitement des retours utilisateurs.
- **Reconnaissance des gestes :** En utilisant TensorFlow Lite, vous pouvez développer des systèmes qui comprennent et interprètent les gestes humains comme des commandes, permettant des installations interactives ou améliorant les interfaces utilisateur dans les appareils.
- **Détection d'anomalies :** TensorFlow Lite peut être utilisé pour la détection d'anomalies ou de valeurs aberrantes dans les données de séries temporelles, ce qui est précieux pour la maintenance prédictive dans les environnements industriels ou les systèmes de surveillance tels que la détection de battements cardiaques irréguliers dans les appareils de santé.
- **Estimation de la pose :** Le framework est capable d'exécuter des modèles d'estimation de la pose qui détectent les figures humaines et leurs postures dans les images ou les vidéos. Cela peut être appliqué à l'analyse sportive, aux applications de fitness et aux applications interactives avancées, comme démontré lors de notre récente visite au salon embedded world 2024.

Grâce aux améliorations apportées à TensorFlow Lite sur Raspberry Pi, nous avons réalisé des progrès significatifs en matière de détection d'objets et de texte, bien que les vidéos entrelacées et de faible résolution présentent toujours des défis. Ces améliorations pourraient aider les amateurs à gérer efficacement les applications en temps réel et sont particulièrement utiles pour des projets tels que la surveillance domestique et l'archivage de médias. Il reste encore beaucoup à explorer et à perfectionner, et j'ai hâte de voir comment nous pourrons adapter ces outils à nos besoins.

230181-G-04

Questions ou commentaires ?

Envoyez un courriel à l'auteur (brian.williams@elektor.com).

À propos de l'auteur

Brian Tristam Williams est fasciné par les ordinateurs et l'électronique depuis qu'il a eu son premier « micro-ordinateur » à l'âge de 10 ans. Son aventure avec le magazine Elektor a commencé lorsqu'il a acheté son premier numéro à 16 ans, et depuis lors, il suit le monde de l'électronique et de l'informatique, explorant et apprenant sans cesse. Il a commencé à travailler pour Elektor en 2010 et, aujourd'hui, il s'attache à suivre les dernières tendances technologiques, en se concentrant notamment sur l'IA et les ordinateurs monocartes tels que le Raspberry Pi.

Related Products

- **Raspberry Pi 5 (4 GB)**
  www.elektor.fr/20598
- **Raspberry Pi Camera Module 3**
  www.elektor.fr/20362
- **Raspberry Pi Camera Module 3 NoIR**
  www.elektor.fr/20363

LIEN

[1] Bibliothèque Adafruit TSL2561 pour Python sur le Raspberry Pi : https://github.com/adafruit/TSL2561-Arduino-Library