Vérificateur de l'uniforme - Yolo v8 👷

Application web Flask de vérification du port de l'uniforme des travailleurs (casque et gilet)

A - YOLO : YOu Look Once

1/ Le besoin du demandeur et le sujet d'inférence

Les caméras sont devenues un moyen important pour développer plusieurs applications en termes de surveillance et de sécurité.
Une entreprise qui propose des solutions technologiques en caméra de surveillance et de sécurité cherche à développer une application IA capable de détecter et de localiser la présence du port d'un casque/gilet de chantier ou non à partir d'une vidéo.
L'application à développer est une application Web, elle est capable d'activer la caméra du PC et affiche sur la vidéo le résultat de la détection.
Le besoin de cette entreprise est d'assurer les fonctionnalités suivantes :

- Détection et/ou localisation de la présence/absence du gilet de sécurité.
- Un message d'alerte à afficher sur la page web (en dehors de la vidéo) si une seule personne ne porte pas son casque ou son gilet.

Le modèle devra être précis mais léger, et devra être déployable dans un navigateur web.

2/ Les données : de la recherche à la création du dataset

Le projet a commencé par la recherche de datasets labellisés avec des labels en lien avec la détection d'objets à réaliser, notamment des personnes, des gilets de sécurité et des casques.

En premier lieu nous nous sommes rendus sur *Kaggle*, et avons pu relever que de larges datasets étaient disponibles, mais qu'ils ne contenaient pas les labels que l'on souhaitait.Il aurait fallu fusionner 2 larges datasets,et conjoindre le nommage des labels, en veillant à ne pas les intervertir.
Sous l'impulsion De Thibault, nous avons opté pour unir nos forces, et chacun scraper des données, les rassembler, nous répartir le labelling, et constituer notre propre custom dataset.

Scrapping

J'ai commencé par utiliser la bibliothèque python Beautifulsoup (bs4), mais au bout de quelques minutes, je me rends compte que je vais être limité par les anti-crawler qui repèrent l'activité de mon script. J'opte donc pour l'installation de Spyder, nettement plus puissant, notamment du fait de cette option :

```
# Obey robots.txt rules
ROBOTSTXT_OBEY = False
```

permettant d'avoir accès aux images que l'on souhaite pour entraîner le modèle.

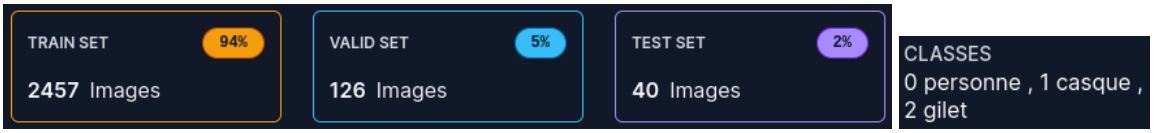
Voici les URL utilisées :

```
def start_requests(self):
# URLs pour depositphotos.com
for i in range(502, 1000):
url = f"https://depositphotos.com/fr/photos/workers-vest-helmet.html?offset={(i - 1) * 100 + 100}"
yield scrapy.Request(url=url, callback=self.parse)

# URLs pour fr.freepik.com
for i in range(502, 1000):
url = f"https://fr.freepik.com/search?format=search&people=include&query=workers%20vest%20helmet&type=photo&offset={(i -
1) * 100 + 100}"
yield scrapy.Request(url=url, callback=self.parse)
# URLs pour shutterstock.com
for i in range(502, 1000):
url = f"https://www.shutterstock.com/fr/search/workers-vest-helmet?page={i}"
yield scrapy.Request(url=url, callback=self.parse)
# URLs pour unsplash.com
for i in range(401, 1000):
```

```
url = f"https://unsplash.com/fr/s/photos/workers-vest-helmet?page={i}"
yield scrapy.Request(url=url, callback=self.parse)
```

Ensuite nous les avons uploadé sous Roboflow : cet outil de labelling fait partie de la suite Ultralytics, il est simple à manier, et nous a permis de labelliser nos images en groupe, et ainsi nous répartir la charge. Nous nous en sommes sorti avec un total de 2700 images labellisées avec les labels suivants : personne, casque et gilet.



3/ Le modèle et son entraînement: la puissance de YOLO

Le modèle choisi pour ce projet est basé sur l'architecture YOLO (You Only Look Once), connue pour sa rapidité et son efficacité dans la détection d'objets en temps réel. Après avoir préparé le dataset, il a été facile de l'intégrer dans le code grâce à l'intégration entre YOLO, Ultralytics et Roboflow.

```
!pip install roboflow --quiet
```

```
from roboflow import Roboflow
rf = Roboflow(api_key="MA_CLEF_API_ROBOFLOW")
project = rf.workspace("yolosafetygear").project("safety_gear_simplon")
dataset = project.version(3).download("yolov8")
```

B - Déployer un modèle

1/ État de l'art du déploiement de modèle : en majeur partie Streamlit/Gradio & Flask

Pour le déploiement de notre modèle de détection basé sur YOLO, nous avons examiné plusieurs options populaires. Streamlit et Gradio sont des choix courants en raison de leur facilité d'utilisation et de leur convivialité. Ces bibliothèques permettent de créer facilement des applications Web interactives pour présenter les modèles d'apprentissage automatique de manière conviviale.
Flask, en revanche, offre une flexibilité et un contrôle plus poussés, ce qui en fait un choix judicieux pour des applications plus complexes nécessitant une personnalisation approfondie.

2/ Déploiement sur Flask : avantage et inconvénients

Nous avons opté pour Flask en raison de sa polyvalence et de sa capacité à gérer efficacement les tâches plus avancées. Voici quelques avantages et inconvénients liés à notre choix de déploiement :

Avantages :

- Flexibilité et Personnalisation : Flask offre un contrôle total sur la structure de l'application. Cela nous a permis de personnaliser l'interface utilisateur et d'ajuster les fonctionnalités en fonction des besoins spécifiques de notre projet.
- Intégration facile : Flask peut être facilement intégré à d'autres technologies et services, ce qui facilite la mise en place de flux de travail complexes.

- Support de la communauté : Flask bénéficie d'une large base d'utilisateurs et d'une documentation exhaustive. Cela signifie que nous avons accès à un grand nombre de ressources pour résoudre d'éventuels problèmes et optimiser notre application.

Inconvénients :

- Complexité initiale : Comparé à des outils comme Streamlit, la mise en place initiale d'une application Flask peut sembler plus complexe, en particulier pour les utilisateurs moins expérimentés. Cependant, cette complexité initiale est compensée par la flexibilité qu'elle offre.
- Gestion de l'évolutivité : Alors que Flask est capable de gérer des applications complexes, la gestion de l'évolutivité peut devenir un défi si l'application devient extrêmement populaire et doit gérer un grand nombre de requêtes simultanées. Cependant, avec une architecture appropriée, ce défi peut être surmonté.

3/ Conclusion et improvement possibles

Ce projet permet de saisir tout le potentiel des modèles de Computer Vision, notamment YOLO qui permet de la détection en temps réel du port du casque/gilet de chantier à partir de vidéos, en utilisant un modèle YOLO efficace et un dataset personnalisé. Le déploiement sur Flask a offert la flexibilité nécessaire pour personnaliser notre application selon les exigences spécifiques de notre client, tout en garantissant une expérience utilisateur fluide.

Cependant, il y a toujours place à l'amélioration :

- Optimisation du Modèle : Bien que notre modèle soit précis et léger, il est toujours judicieux de chercher des moyens de l'optimiser davantage pour une utilisation plus efficace des ressources système, ce qui est crucial lorsqu'il s'agit de déploiement dans un navigateur web.
- Interface Utilisateur : Bien que fonctionnelle, l'interface utilisateur peut toujours être améliorée en termes de convivialité et de design. L'expérience utilisateur est essentielle pour garantir que l'application soit intuitive et facile à utiliser, même pour les utilisateurs non techniques.
- Sécurité : En matière de sécurité, il est crucial de s'assurer que l'application est protégée contre les menaces telles que l'injection SQL et les attaques par déni de service (DDoS). Des mesures de sécurité robustes doivent être mises en place pour protéger les données utilisateur et l'intégrité de l'application.

source : https://github.com/Artemis-IA/YOLOv8_Detection