

Rapport de Mini Projet IoT : Détection de Feu et de fumée

Etudiant : Mahdi SELMANI

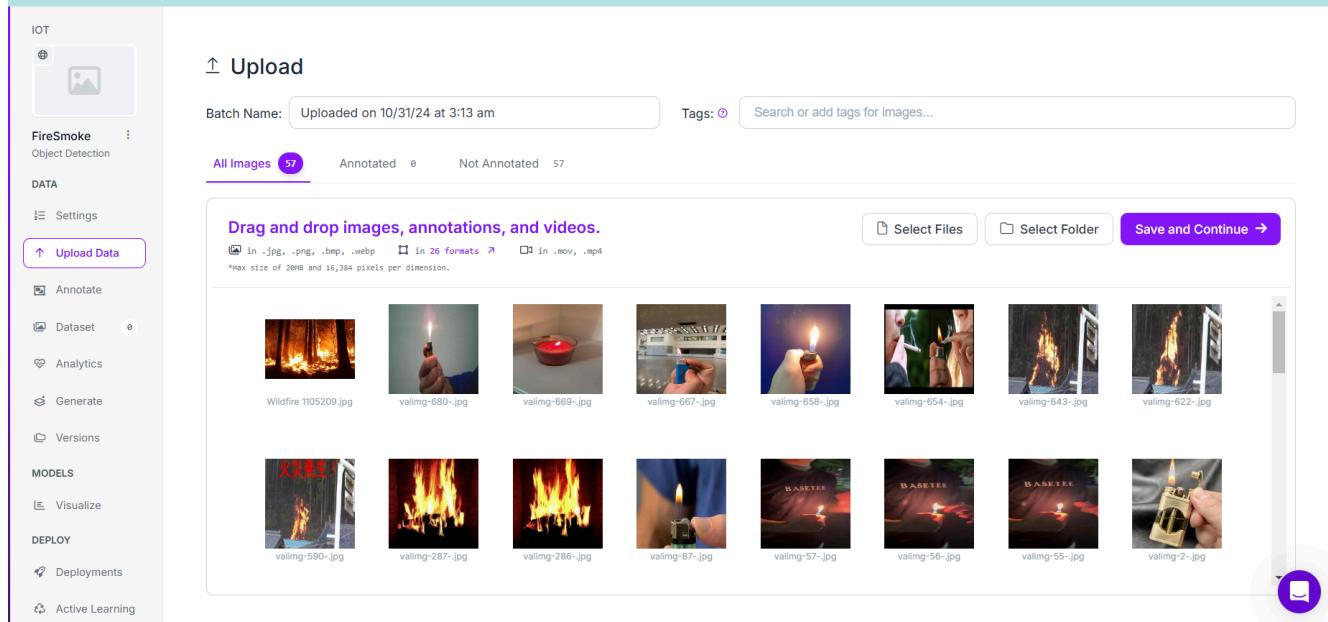
Configuration de l'environnement:

Installation de Python et Ultralytics

Note : J'ai également configuré une machine virtuelle pour simuler la carte Raspberry Pi. Cependant, j'ai rencontré des conflits de dépendances en raison de la dernière version de Raspberry Pi OS Desktop lancée en 2022. Je serais ravi de connaître vos suggestions d'alternatives, autres que QEMU.

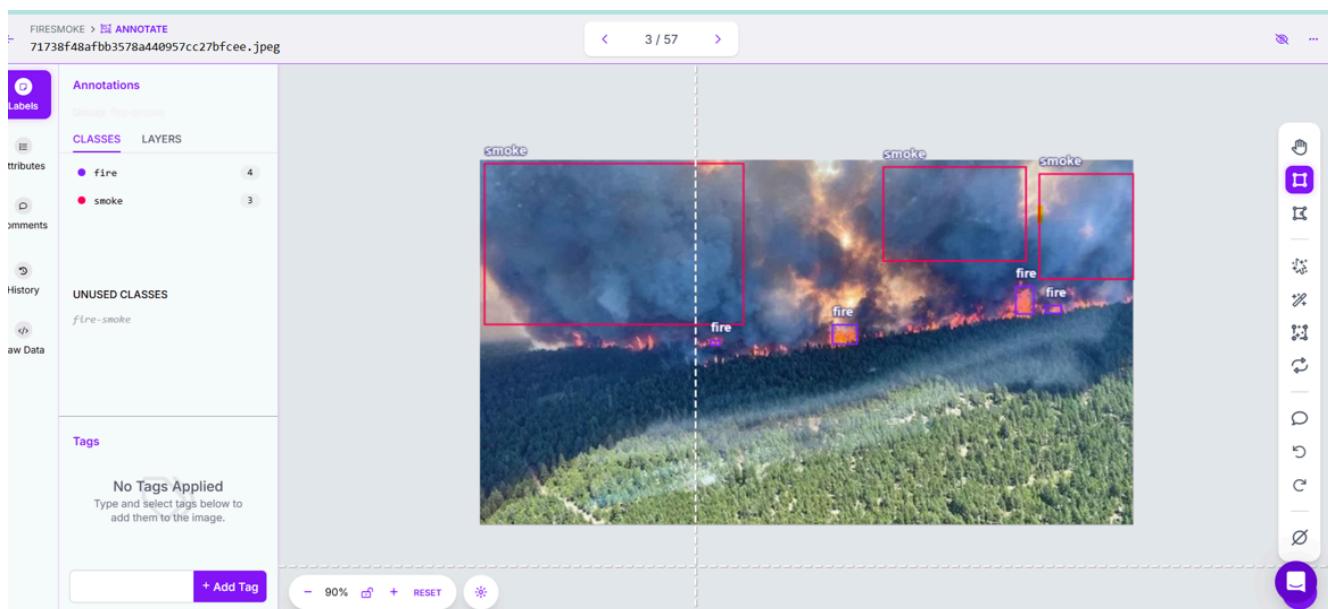
1ère Étape : Collecte d'Images

Dans cette première étape, j'ai collecté diverses images de feu et de fumée provenant de plusieurs sources (internet, images personnelles, etc.).



2ème Étape : Annotation des Images

Pour préparer le dataset, les images collectées ont été annotées afin de créer des boîtes englobantes autour des instances de feu ou de fumée. On a donc deux classes : 'FIRE' et 'SMOKE'



Outils Utilisés :

Roboflow : Une plateforme en ligne utilisée pour annoter facilement les images. Roboflow a permis de :

- Charger les images en masse.
- Dessiner des boîtes englobantes sur les zones contenant du feu ou fumée
- Exporter les annotations au format compatible avec YOLO.

3ème Étape : Entraînement du Modèle

J'ai choisi le modèle YOLOv8 pour entraîner notre système de détection de feu et de fumée.

The screenshot shows a code editor interface with several tabs open. The main tab contains the Python script 'train.py' which uses the 'ultralytics' library to train a YOLO model. The script imports YOLO, loads a pre-trained 'yolov8n.pt' model, and performs training with a configuration file 'data.yaml'. The output terminal shows the progress of the training process, including model summary, transferred items, and training metrics. The code editor has a dark theme with syntax highlighting for Python and JSON.

```
from ultralytics import YOLO

# chargement du modèle pré-entraîné de Yolo
model = YOLO("yolov8n.pt") # Load the pre-trained model

# entraînement du dataset de feu
results = model.train(data="data.yaml", epochs=20, imgsz=640, batch=16,
save=True)

print("Training completed!")
```

Script d'entraînement :

```
from ultralytics import YOLO

# chargement du modèle pré-entraîné de Yolo
model = YOLO("yolov8n.pt") # Load the pre-trained model

# entraînement du dataset de feu
results = model.train(data="data.yaml", epochs=20, imgsz=640, batch=16,
save=True)

print("Training completed!")
```

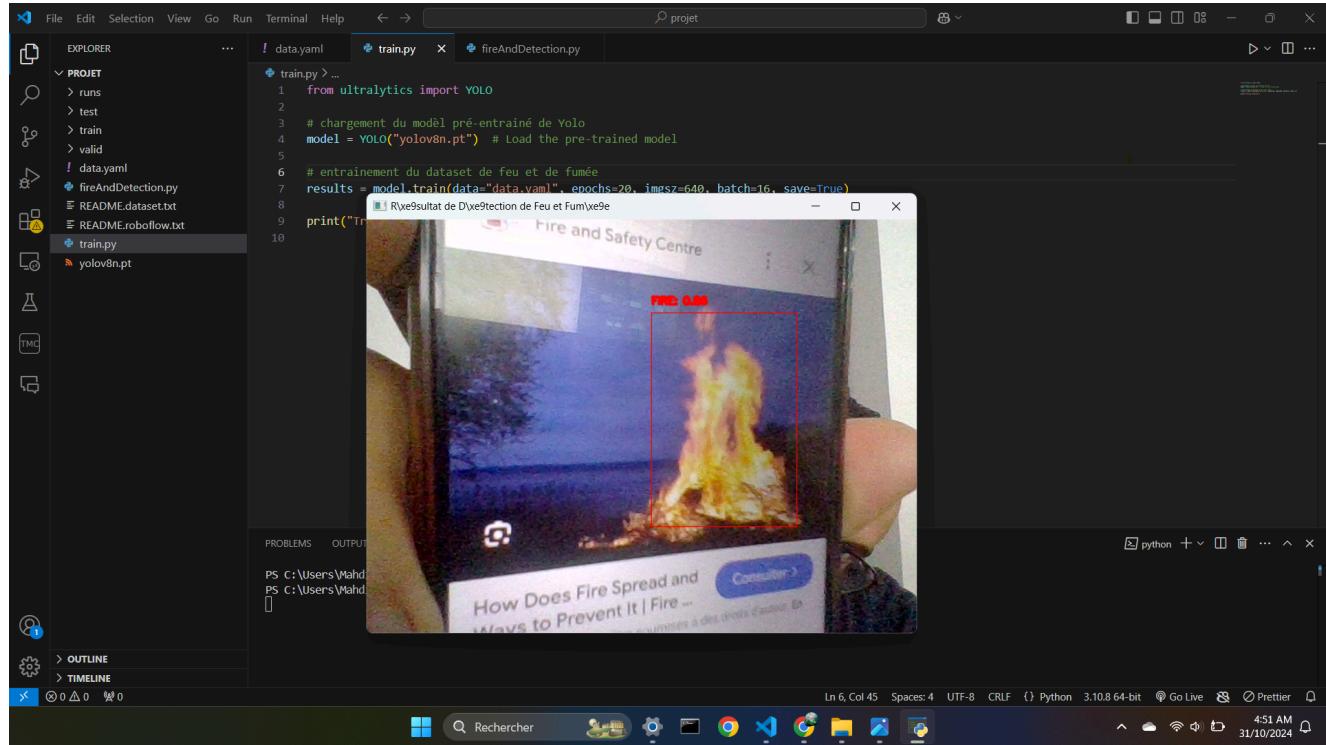
Durée d'entraînement : Environ 20 minutes sur un GPU NVIDIA GTX 3060.

4ème Étape : Évaluation et Tests

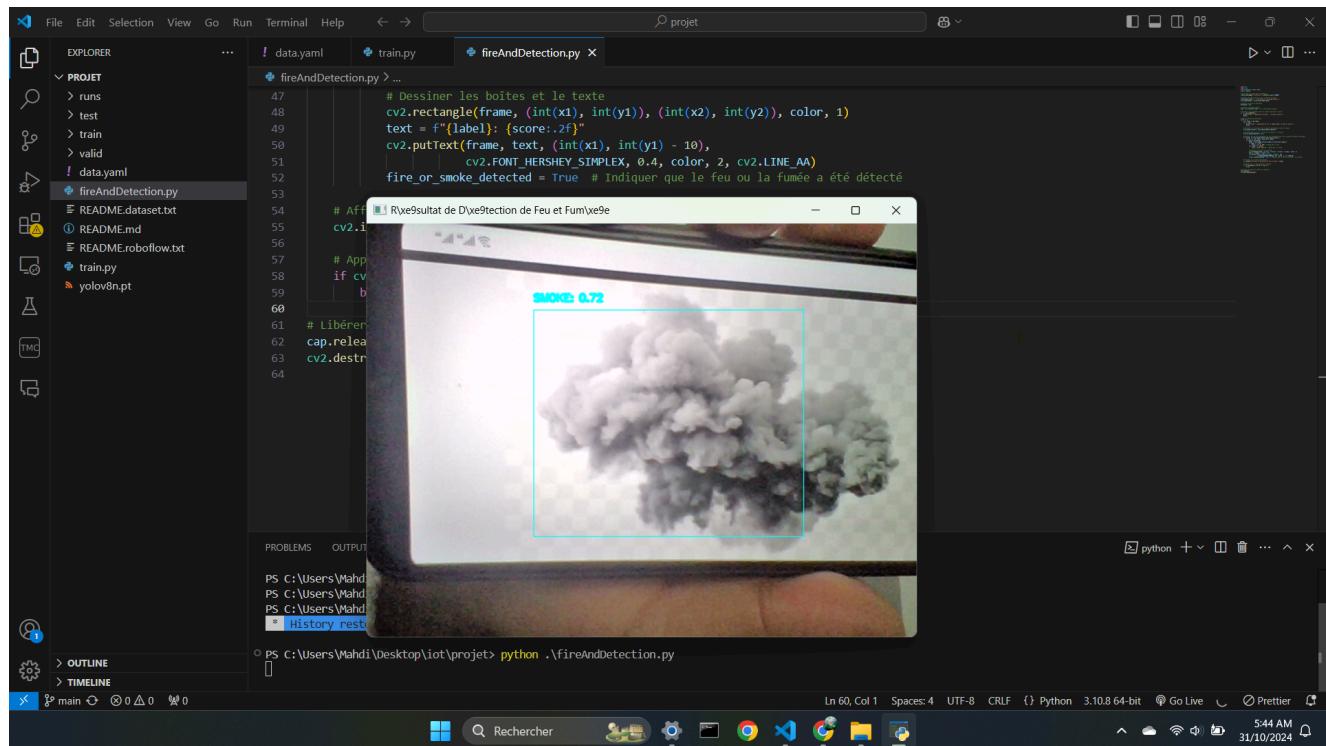
Après l'entraînement, le modèle a été testé afin d'évaluer sa performance. Les résultats ont montré une précision de détection de 80% avec un faible taux de faux positifs.

Les captures d'écran illustrent que le modèle détecte la présence de feu et de fumée sur une photo affichée sur mon téléphone.

Détection du feu :



Détection de la fumée :



Script du programme de détection de feu ou de fumée :

```
import os
import cv2
from ultralytics import YOLO
import logging

# Désactiver les logs pour ultralytics
logging.getLogger('ultralytics').setLevel(logging.ERROR)

# Charger le modèle entraîné pour le feu et la fumée
custom_model_path = './runs/detect/train/weights/best.pt'
fire_smoke_model = YOLO(custom_model_path)

# Définir le seuil de confiance
threshold = 0.6

# Utiliser la webcam locale
cap = cv2.VideoCapture(0) # 0 est la cam par défaut (Webcam locale)

# Vérifier si la connexion avec la caméra locale est réussie
if not cap.isOpened():
    print("Erreur : Impossible d'accéder à la caméra locale.")
    exit()

# Boucle continue de détection
while True:
    ret, frame = cap.read()
    if not ret:
        print("Erreur : Impossible de lire l'image depuis la caméra locale.")
        break

    # Effectuer la détection avec le modèle personnalisé (feu et fumée)
    fire_smoke_results = fire_smoke_model(frame)[0]

    # Variable pour suivre si le feu ou la fumée a été détecté
    fire_or_smoke_detected = False

    # Dessiner les boîtes englobantes et les étiquettes pour les résultats du
    # modèle feu/fumée
    for result in fire_smoke_results.boxes.data.tolist():
        x1, y1, x2, y2, score, class_id = result
        if score > threshold:
            label = fire_smoke_results.names[int(class_id)].upper()
            if label == "FIRE":
                color = (0, 0, 255) # Rouge pour le feu
```

```

        elif label == "SMOKE":
            color = (255, 255, 0) # Jaune pour la fumée

            # Dessiner les boîtes et le texte
            cv2.rectangle(frame, (int(x1), int(y1)), (int(x2), int(y2)), color,
1)
            text = f"{label}: {score:.2f}"
            cv2.putText(frame, text, (int(x1), int(y1) - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.4, color, 2, cv2.LINE_AA)
            fire_or_smoke_detected = True # Indiquer que le feu ou la fumée a
été détecté, on va utiliser cette variable pour la deuxième partie avec le
Broker MQTT

        # Afficher le résultat de détection
        cv2.imshow("Résultat de Détection de Feu et Fumée", frame)

        # Appuyer sur 'q' (quitter) pour quitter la boucle
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

# Libérer la caméra et fermer les fenêtres
cap.release()
cv2.destroyAllWindows()

```

Prochaines Étapes :

Mettre en place un broker MQTT qui permettra au script de publier des messages en cas de détection de feu ou de fumée.

Lien github vers le projet : <https://github.com/MahdiSelmani/iot>

Lien vers le DataSet Roboflow : <https://universe.roboflow.com/iot-7d4ka/firesmoke-jqkgo>