Projet MLOps

Nabilath NACHIROU LIAMIDI
Nathan CERISARA

Table des matières

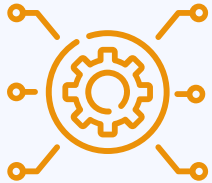
01

Description & architecture du projet

01.A - Architecture Générale

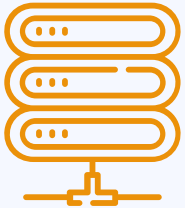
Python 3.12

Github



Docker Compose

-> 6 Services Docker



MLFlow

PostgreSQL + MinIO

Modèles YOLO

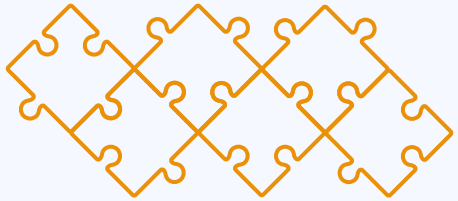
Ultralytics YOLOv11/26
(nano, small, medium)

Serving API & Frontend

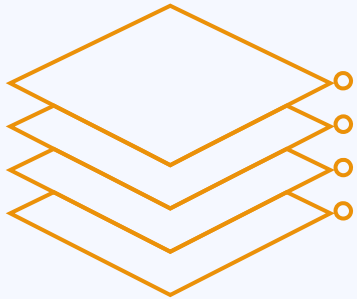
Nginx + FastAPI
+ App Web (Vanilla JS)

01.B - Services Docker

6 services docker:

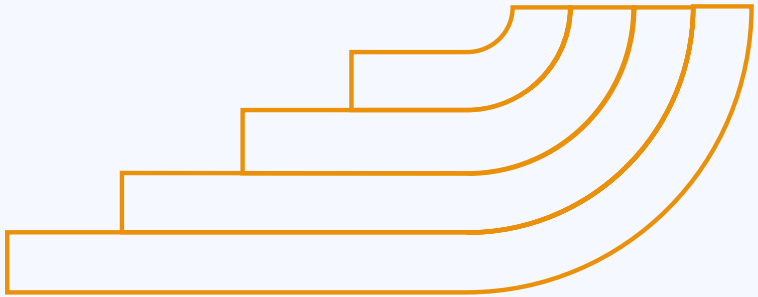


- **PostgreSQL** (Metadata store)
- **MinIO S3** (Model registry)
- **MLFlow** (Lié aux conteneurs **PostgreSQL** et **MinIO S3**)
- **Training pipeline** (version CUDA et non CUDA)
- **Backend Serving API** (serveur)
- **Frontend Web App** (page web)



01.C - MLFlow

- **Metadata store :** PostgreSQL
- **Model Registry :** MinIO S3
- Hyperparamètres (epochs, batch, lr)
- Artefacts (plots, configs, ...)
- Versioning automatique



01.D - Scripts Cross-Platform

- Scripts **bash (linux)** ou **bat/ps1 (Windows)**
- Servant à:
  - Lancer **un entraînement**
  - Lancer **le serving**
  - **Exporter/Importer** Metadatastore & Model Registry (en un *backup.zip*)

01.E - Bonnes pratiques

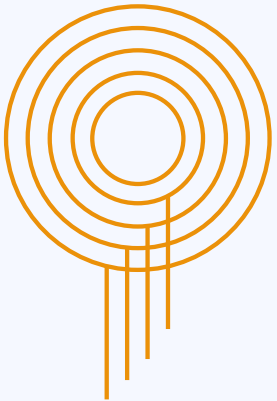
- Projet bien modulaire. Bonne utilisation de la PPO
- Type Hint complet
- **Pylint** & **Ruff** pour le ***linter*** dans vscode
- **Ruff** pour le ***formatage du code*** dans vscode
- Pas de **mypy** car problèmes avec les .venv dans vscode
- **Github Action :** CI avec Pylint (qualité de code de ***9.63/10***)
- Début d'implémentation des **tests unitaires**, mais pas le temps de faire beaucoup de coverage

02

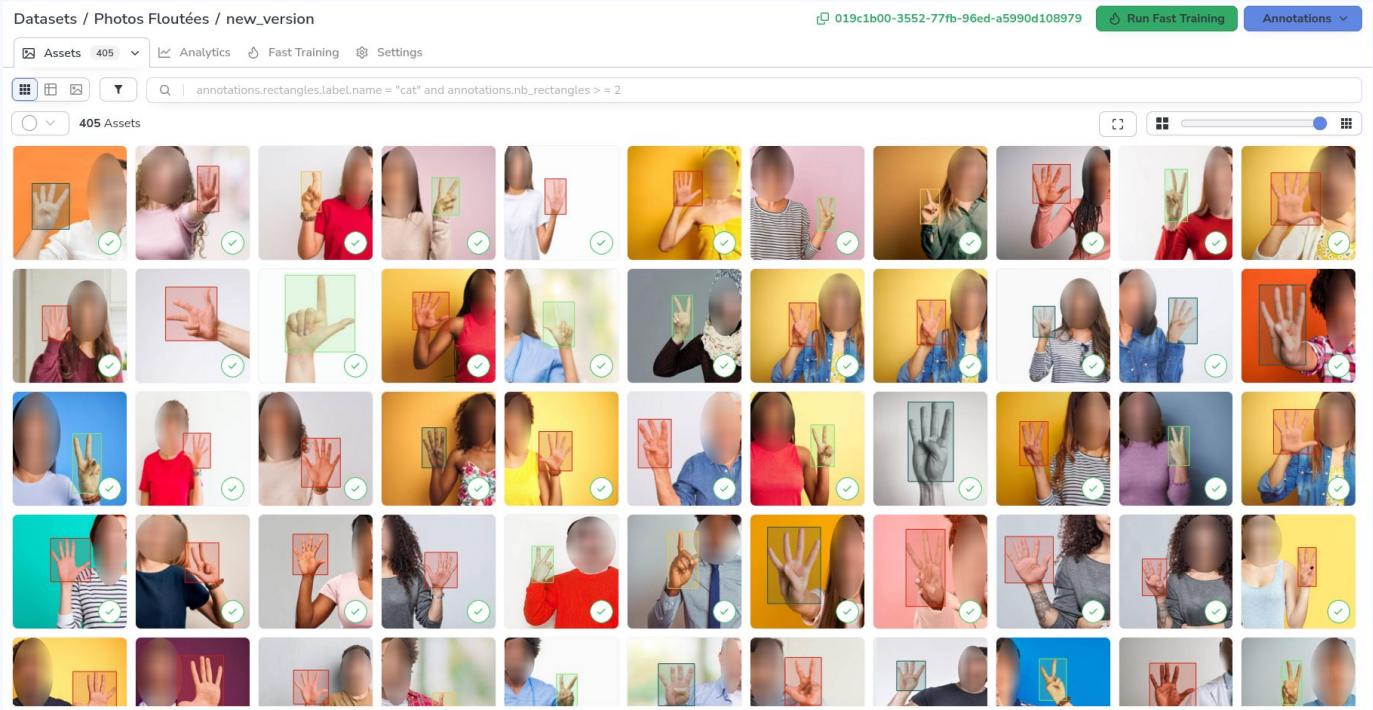
Données & Dataset

02.A - Préparation des données

- **405** images annotées : **~105** images personnelles, **~90** images dataset Kaggle, **~200** images Roboflow
- Floutage du visage avec **deface**
- **Picsellia** pour l'annotation
- **7 classes :** 0, 1, 2, 3, 4, 5, unknown
- Double validation des annotations



02.B - Picsellia



02.C - Répartition des classes

26 images - **0 doigt**

77 images - **1 doigt**

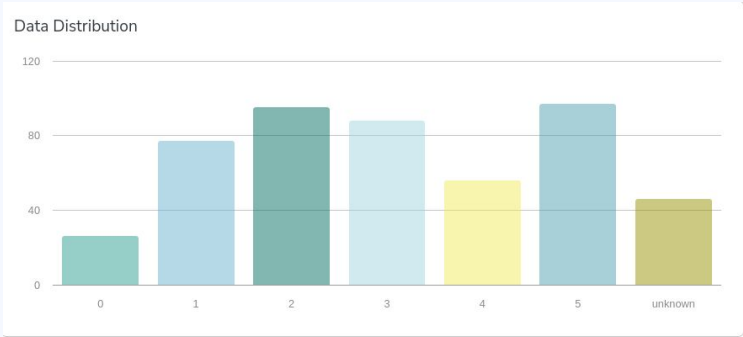
95 images - **2 doigts**

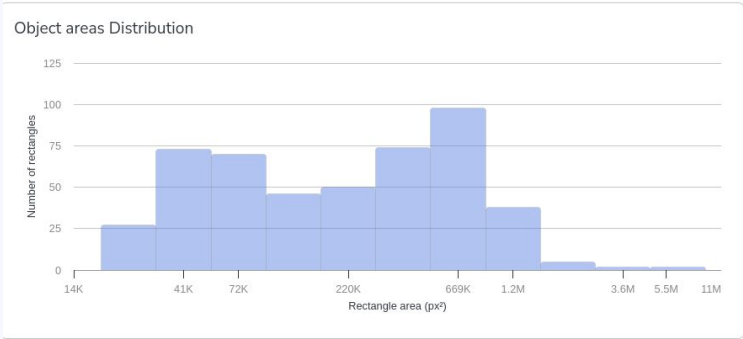
88 images - **3 doigts**

56 images - **4 doigts**

97 images - **5 doigts**

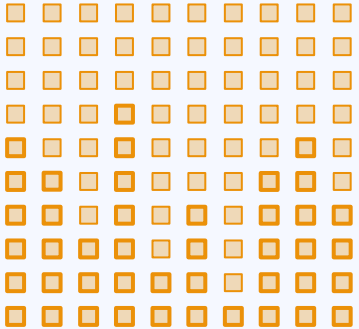
46 images - **Unknown**





02.C - Gestion des données

- Depuis **Picsellia SDK** en format **YOLO**
- Récupération de **315** images <- **Erreur**
  - Plus aucun exemple avec 0 doigts !
- **Split :** 70% train, 15% test, 15% validation



03

Entraînement & Déploiement

03.A - Pipeline

Picsellia API

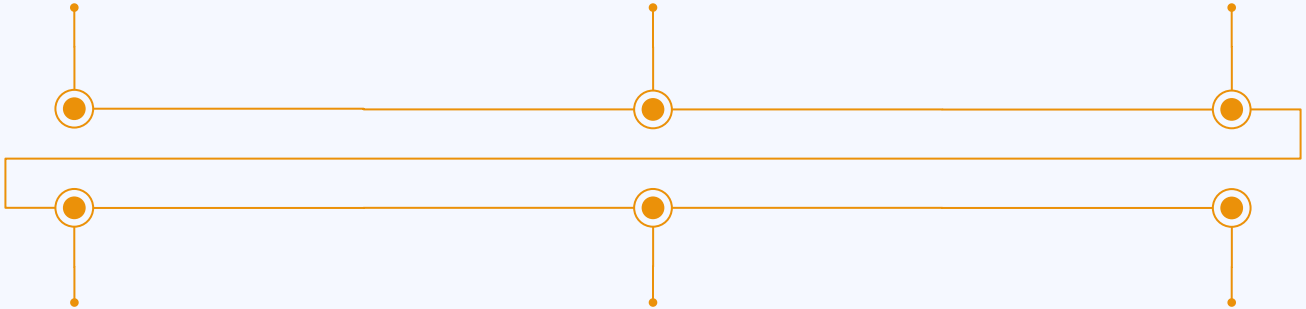
Data extraction

Manuellement sur Picsellia
Vérifications d'ultralytics YOLO

Data validation

YOLO format

Data preparation



Model training

Utilisation de ultralytics yolo.train

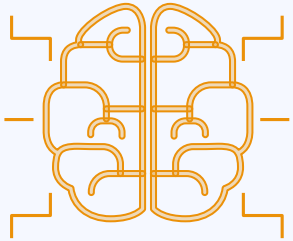
Model evaluation

Test avec ultralytics yolo

Model validation

On propose tous les modèles, l'utilisateur choisi

03.C - Entraînement



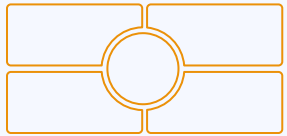
- **300** epochs, batch size **16**
- **Early stopping** avec patience de **30**
- **Optimiseur :** *AdamW*
- **Loss :** *box loss (boite), cls loss (classe), dfl loss (Features, pour différencier les exemples difficiles)*
- **Appareil d'entraînement :** pc InnovLab / CUDA

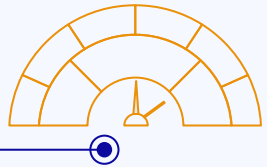
03.D - Évaluation & Validation

- **Jeu de Test** de **40 examples**
- **Métriques :** *mAP50-95*, *precision*, *recall*
- **Matrices de confusion** multi classes
- **Modèle-validation:** le champion serait ***yolo26m***, mais on préfère laisser le choix du modèle à l'utilisateur

03.E - API de Serving et WebApp

- **Serveur python docker** + **FastAPI**
- **Serveur simple** (1 client non multithreadé)
- Chargement **dynamique** des modèles
- Web Application **HTML/CSS/JS Vanilla** légère
- Communication via **HTTP POST**, Vidéo transmise par **Blob**



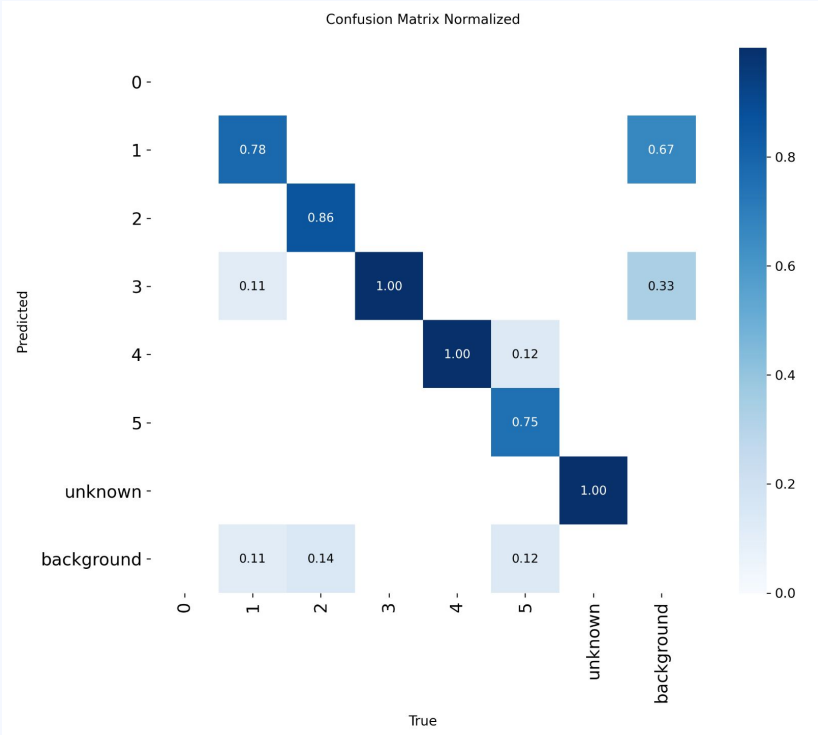


03.F - Preprocessing

- **Détection** et **floutage** de visage avec *OpenCV Haar Cascades*
- Configurable via **variable d'environnement** (ON/OFF)
- Preprocessing appliqué **avant l'inférence** de YOLO

03.J - Résultats

Modèle	Final mAP@50-95
Yolo26m	0.787
Yolo26s	0.771
Yolo11s	0.748
Yolo11n	0.739
Yolo26n	0.672



Matrice de confusion normalisée pour Yolo26m

Conclusion & Démo

Difficultés rencontrées

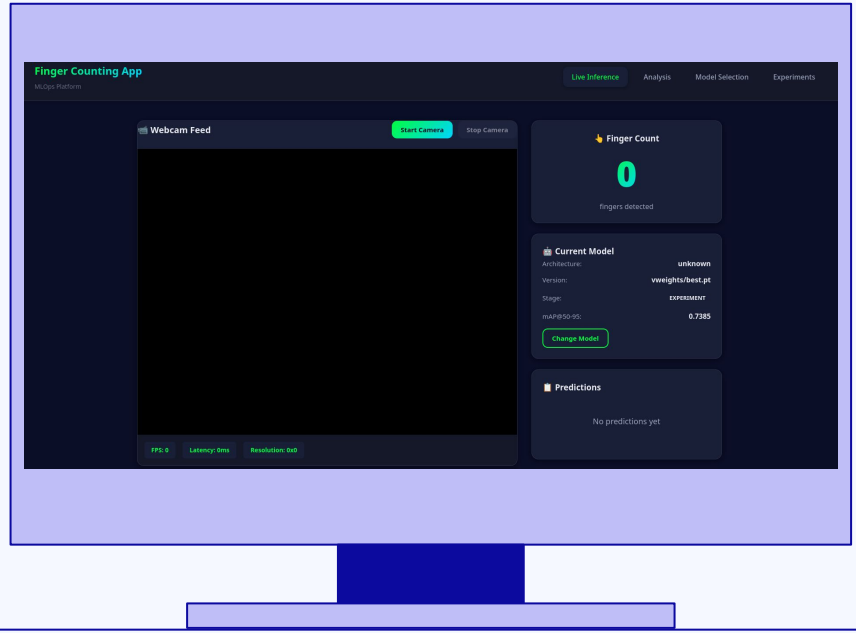
- Lourdeur de docker / Pytorch
- Inférence Lente / FPS non fluides
- Perte de données de Picsellia (~100 données perdues)
- Modèles pas trop performants

    -> Il faut travailler nos données

Conclusion

- Projet Intéressant mais calendrier serré 
- Bonnes pratiques de code utilisées 
- Pipeline MLOps fonctionnelle 
- Modèles améliorables 

Démo


Finger Counting App
MLOps Platform
Live Inference
Analysis
Model Selection
Experiments
Webcam Feed
Start Camera
Stop Camera
FPS: 0
Latency: 0ms
Resolution: 0x0
Finger Count
0
fingers detected
Current Model
Architecture:
unknown
Version:
vweights/best.pt
Stage:
EXPERIMENT
mAP@50-95:
0.7385
Change Model
Predictions
No predictions yet


Questions ?