Université de Strasbourg
TPS

Projet SHARP
(Smart Hand Automated Recognition Project)

L'objectif du projet SHARP est de développer un projet en Python capable d'utiliser la *Computer Vision* pour reconnaître automatiquement les différentes combinaisons des doigts d'une ou plusieurs mains, tout en appliquant les bonnes pratiques de développement ainsi que les composants MLOps abordés en cours.

1. Contraintes

Les contraintes à respecter pour le projet seront les suivantes :

- Projet à réaliser en groupe de 2 ou 3 personnes.
- Chaque groupe doit disposer, au minimum, d'un GPU NVIDIA ou d'une puce Apple M1 (ou supérieure). À défaut, il est possible d'utiliser les machines mises à disposition par TPS.
- Le code est versionné sur GitHub.
- Le projet est réalisé en Python 3.12.
- Le datalake, la gestion des datasets et l'annotation sont réalisés via un outil parmi ceux présentés en cours (Picsellia, Roboflow, Labelbox, etc.).
- L'entraînement des modèles est réalisé avec les modèles YOLO développés par Ultralytics (YOLO11 ou YOLO26).
- L'Experiment Tracking (Metadata Store) et le versioning des modèles (Model Registry) sont réalisés avec MLflow (possibilité d'héberger sa propre instance dans le cloud, ou d'utiliser une version locale via Docker), Weights & Biases, Roboflow ou Picsellia.
- Le déploiement des modèles (Serving) est réalisé dans un conteneur Docker qui récupère et charge le modèle YOLO au démarrage, puis expose une interface web affichant le flux webcam en temps réel sur lequel s'exécute le modèle (ex. localhost:XXXX).
- La dockerisation de la Pipeline de Training est optionnelle.

2. Contenu du projet 🚀

A. Pipeline de Training

Le projet SHARP met en œuvre un ensemble d'outils, de *pre-processing* et de *post-processing* pour développer une IA capable de détecter les doigts d'une ou plusieurs mains. La liste des combinaisons est définie et ne peut pas être modifiée :

- 0_doigt
- 1_doigt
- 2_doigts
- 3_doigts
- 4_doigts
- 5_doigts

On notera que si deux mains sont visibles sur une frame donnée, alors chaque main aura sa propre *bounding box* avec son numéro associé.

Concernant le développement de la Pipeline de Training, il conviendra d'appliquer les principes de la Pipeline ML abordés pendant les cours. Libre donc à chaque groupe de choisir la manière la plus efficace d'implémenter les différentes étapes de la Pipeline ML (fonctions, classes, etc.), tout en s'efforçant de :

- Veiller à ce que le code soit de bonne qualité.
- S'assurer que les responsabilités des différentes briques de la Pipeline ML soient correctement séparées.

Voici quelques informations supplémentaires concernant chaque partie de la Pipeline :

1. **Extraction** :
    - Utiliser le SDK fourni par le Datalake pour télécharger le dataset annoté. L'identifiant du dataset à récupérer devra être configurable, soit via un argument (utiliser *argparse*), soit via un fichier de configuration *config.py* (utiliser *dynaconf*, *python-decouple*, etc.).
    - Veiller à bien récupérer les images et les annotations associées.
2. **Validation** : effectuer les tests nécessaires comme vu en cours.

3. **Préparation** :

   - Convertir le dataset au format YOLO (uniquement si le dataset est exporté dans un format différent).
   - Si aucun split n'est fourni, générer aléatoirement les trois splits train / val / test avec les proportions 60% / 20% / 20%, en utilisant la seed 42. Selon le Datalake utilisé, il est aussi possible de définir les splits directement via l'interface web associée. Vous pouvez également fixer uniquement le split test puis expérimenter plusieurs répartitions pour train et val. Le choix et la gestion des splits restent à la discrétion de chaque groupe.
   - Si le fichier n'est pas présent lors de l'extraction, générer dynamiquement le fichier *config.yaml* (en utilisant la librairie *yaml*) attendu par Ultralytics, afin de lancer l'entraînement via le paramètre data de la méthode *train(...).*

4. **Training** :

   - Configurer et ajuster les hyper-paramètres afin d'aboutir aux meilleures performances. La configuration des augmentations est à la discrétion de chaque groupe. On pourra aussi utiliser le guide d'*Ultralytics* pour fine-tuner les hyper-paramètres, ou encore le guide pour les data augmentations.
   - L'architecture à utiliser est YOLO11 ou YOLO26. La version du modèle à utiliser (nano, small, medium, large ou XL) est à la discrétion de chaque groupe. Un minimum de 10 FPS est attendu lorsque le modèle tournera en live.
   - Les métriques (loss, accuracy, etc.), générées tout au long du training par Ultralytics, devront être loggées sur le Metadata Store choisi.

5. **Evaluation** :

   - Logger les métriques d'évaluation/artifacts générés par Ultralytics dans le Metadata Store.
   - Possibilité également de générer/logger ses propres métriques personnalisées si applicable.

6. **Validation** :

   - Une fois le training terminé, utiliser le SDK correspondant pour stocker le nouveau modèle entraîné dans le *Model Registry* (MLflow, Roboflow, Weights & Biases, Picsellia, etc.).

B. Application pour le Serving

Note : Cette partie peut être réalisée très rapidement grâce à une utilisation pertinente des LLM. N'hésitez pas à les utiliser (Claude, Gemini, ChatGPT, etc.), tout en prenant soin de documenter vos choix et ce que vous avez appris. N'hésitez pas non plus à challenger les réponses des IA afin de vous assurer qu'elles répondent correctement au sujet. Veillez enfin à conserver une codebase saine et minimale : évitez de sur-architecturer et ne rajoutez que ce qui est réellement nécessaire.

Cette seconde partie du projet prendra la forme d'une application web dockerisée chargée de servir le modèle via une API. L'application exposera également un dashboard (sur localhost) affichant le flux webcam en temps réel ainsi que le nombre de doigts visibles et détectés par le modèle. Les contraintes seront les suivantes :

- Le choix des technologies pour l'API (backend) et le frontend est à la discrétion de chaque groupe.
- Le nombre de conteneurs nécessaires au bon fonctionnement de l'application est également à la discrétion de chaque groupe.
- Le choix du framework frontend est aussi à la discrétion de chaque groupe.
- Le flux caméra doit être visible dans l'interface web, avec les prédictions superposées (*bounding boxes* et classes associées).
- À côté du flux, afficher un indicateur correspondant à la somme des doigts détectés, en additionnant tous les doigts de toutes les mains visibles à l'écran.

Bonus : Une page supplémentaire de Monitoring pourra également être développée afin d'afficher un ensemble de métriques de Monitoring (usage du CPU/GPU, throughput, nombre d'erreurs, nombre de prédictions, etc.).

Le schéma ci-dessous est donné à titre d'exemple afin d'illustrer ce qui est attendu. Il s'agit ici d'un exemple **minimal** (libre à vous de faire plus si vous avez envie) :

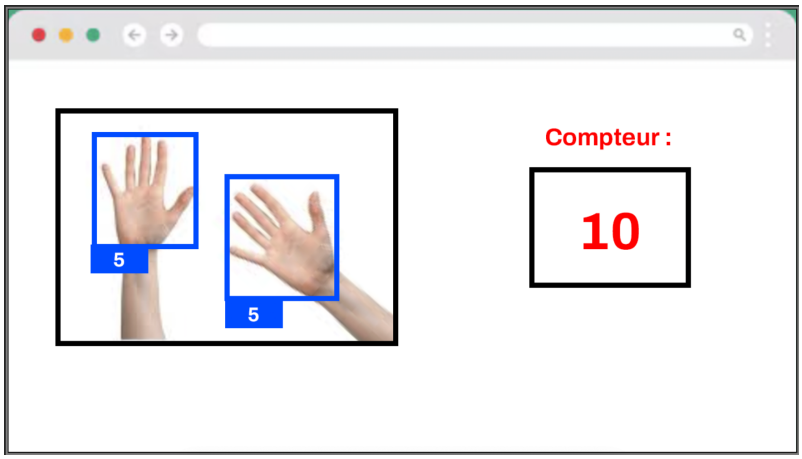

Figure 1 – Schéma récapitulatif des éléments graphiques attendus pour le Serving

3. Mode d'évaluation 🥇

L'évaluation portera sur les éléments suivants :

- Bonne utilisation des bibliothèques Ultralytics et de celles associées au Datalake choisi.
- Qualité du code Python : type hints, absence d'erreurs de lint, code formaté, présence de docstrings, présence d'un pre-commit-config, et code correctement structuré (réparti dans plusieurs fichiers).
- Bonne utilisation des outils choisis pour gérer le Datalake, le Metadata Store (Experiment Tracking) et le Model Registry.
- Présence d'une Pipeline de Training et d'une application de Serving (backend + frontend).
- README complet, contenant toutes les instructions nécessaires pour lancer la Pipeline de Training et l'application de Serving.
- Messages de commit explicites et conformes aux recommandations des Conventional Commits.
- Qualité de la présentation orale.

Bonus :

- Bonus 1 : mise en place d'une pipeline de CI exécutant des tests unitaires à chaque Pull Request.
- Bonus 2 : présence d'une page de Monitoring.

4. Date de rendu ⏳

Plusieurs échéances sont à retenir :

- Présentation des projets : mercredi 11 février au matin.
- Rendu final (GitHub) : mercredi 18 février à 20h.

Pour la restitution des projets, une présentation avec slides est attendue et pourra alors inclure :

- Des statistiques sur le dataset (nombre d'images collectées, annotées, répartition des splits, etc.). On pourra également mentionner ce qui était facile/difficile, ce que vous avez aimé/moins aimé avec l'outil d'annotation.
- Le choix des outils MLOps (Datalake, Metadata Store, Model Registry, Serving).
- Les meilleurs métriques atteintes pendant l'entrainement et l'évaluation du modèle.
- Votre choix pour la stratégie de data augmentation.
- Si déjà terminé, une démo live du projet (bonus). À défaut on pourra juste lancer le modèle avec la commande « *yolo predict model=path/to/model show=True source=0* ».
- Toute autre information jugée pertinente pour valoriser le travail réalisé.

Plusieurs scénarios seront mis en œuvre durant la démonstration. Une partie de la note dépendra donc de la robustesse du modèle face à ces situations (exemples : plus que deux mains visibles, différentes combinaisons de doigts et occlusions).

La notation se fait sur deux évaluations distinctes : une note pour le projet (GitHub) et une note pour la présentation. Chacune est notée sur 22 points : une base sur 20, plus 2 points de bonus. Pour le projet GitHub, les bonus sont attribués via la mise en place d'une CI/CD (+1 point) et la présence d'une page de Monitoring (+1 point). Pour la présentation, le bonus correspond à une démo live du Serving (+2 points). Au total, cela représente donc jusqu'à 4 points de bonus possibles (2 sur le projet + 2 sur la présentation). Parmi ces bonus, jusqu'à 2 points au-dessus de 20 pourront être reportés sur la note du partiel. Enfin, les coefficients sont les suivants : 0,5 pour le partiel, 0,3 pour le projet, et 0,2 pour la présentation.

Les coefficients sont de 0.5 pour le partiel, 0.3 pour le projet et 0.2 pour la présentation.