extractorAPI

Cahier des charges – Avril 2023

Fonctionnalités

L'API permet de déplacer l'annotation des images envoyées par les utilisateur-rice-s de EiDA sur le GPU : dans une logique de développement modulaire, le modèle entraîné n'est pas intégré directement à l'application, mais dans une API indépendante. Cette séparation du corps de l'application et du module générant les annotations permet d'éviter le ralentissement général de l'exécution de l'application sur le serveur Web : l'utilisation d'une API permet de profiter de la capacité de calcul du GPU lors de l'inférence, alors que l'application est développée sur un serveur différent. La communication en SSH entre le GPU et l'application EiDA n'étant pas satisfaisante du point de vue de la sécurité, le développement d'une API permet de gérer ces échanges par une voie sécurisée.
Du point de vue de l'*open source*, cette forme de développement permet la création d'un outil plus flexible, qui pourra être réutilisé par d'autres projets en fonction de leurs besoins et de leur propre architecture.
L'API reçoit une requête HTTP de la part de l'application EiDA lorsque l'utilisateur-rice envoie une nouvelle numérisation de son témoin dans la base de données via un formulaire. Le manifeste généré par l'application est envoyé à un *API endpoint* qui l'enregistre et en extrait les images, qui sont ensuite traitées par le modèle. Les annotations générées sont retournées à l'application en réponse à la requête, et peuvent ainsi être visualisées par l'utilisateur-rice.
L'objectif d'extractorAPI – par rapport à notre *workflow* actuel – est d'automatiser la détection des diagrammes par le modèle, pour que celle-ci puisse être faite à la demande de l'utilisateur-rice, sans intervention humaine, et que les annotations lui soient retournées automatiquement après envoi des numérisations.

Workflow

1. Envoi d'une numérisation par l'utilisateur-rice sur l'application EiDA ;
2. Création automatique d'un manifeste IIIF à partir des images ou du manifeste ;
3. Envoi du manifeste IIIF via requête HTTP POST à l'API *endpoint* ;
4. Appel de la fonction Python :
    a. Vérification et enregistrement du manifeste ;
    b. Enregistrement des images issues du manifeste pour leur traitement (IIIF downloader) ;
    c. Traitement des images par le modèle : création des annotations (fichiers .txt) ;
5. *Return* : envoi des annotations à l'application ;
6. Affichage des images annotées à l'utilisateur-rice.

L'architecture actuelle de l'application implique d'enregistrer deux fois les images : une première fois dans l'application EiDA pour générer le manifeste IIIF, puis une seconde fois sur le GPU pour lancer l'inférence. On peut envisager le déplacement du serveur Cantaloupe sur le GPU pour éviter la duplication de cette tâche.

Outils

L'API est développée avec Flask, plus simple et flexible que Django pour la construction d'une API légère, et le modèle est déployé à l'aide du module PyTorch.

Modèle

Le modèle utilisé a pour base docExtractor – appuyé sur le réseau de neurones U-Net –, un modèle entraîné sur des images synthétiques de manuscrits générées par l'algorithme SynDoc et sur ImageNet. Le script d'inférence utilisé est celui de YOLOv5.
Il est nécessaire de prendre en compte, dans le développement de l'API, que d'autres modèles pour la détection d'objets seront testés, et que sera conservé le modèle produisant les meilleurs résultats.
Le modèle final sera un modèle réentraîné, idéalement commun avec VHS. Il est cependant envisageable d'avoir un modèle par type de source (manuscrit ou imprimé), ce qui nécessiterait

Task queues

Pour éviter la surcharge de l'API et gérer la réception de multiples requêtes, il est nécessaire de mettre en place un *task manager* comme Celery, qui traite en tâche de fond les requêtes reçues, et permet d'imposer qu'elles soient traitées successivement.

Échange de données

Données envoyées

Le **manifeste IIIF** généré par EiDA.
S'il est décidé après le réentraînement de conserver un modèle pour chaque type de témoin (manuscrit ou imprimé), il est nécessaire d'envoyer une donnée sur le type de support pour choisir le modèle correspondant.

Données reçues

Un fichier d'**annotations au format .txt** par manifeste. Celui-ci contient, pour chaque image :

- le numéro de l'image généré par une fonction `enumerate()`,
- le nom du fichier image annoté,
- les coordonnées de la détection au format `x y width height`.

Par exemple, pour deux objets détectés sur la 21e image du manuscrit 22 :

```
21 ms22_0021.jpg
514 907 1700 1685
```

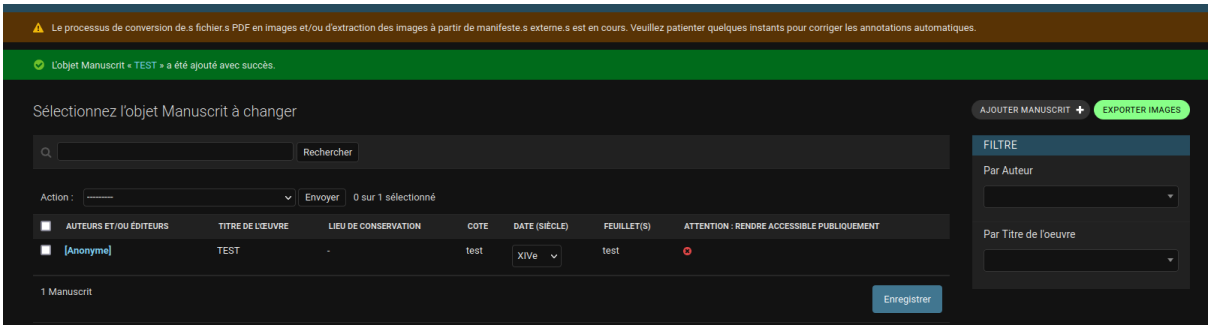
```
2424  831  1441  1423
```

Sécurité

Pour assurer la sécurité de l'API, une authentification avec un token a été envisagée. Pour plus de simplicité, l'API utilise un décorateur pour **restreindre les hôtes autorisés** à envoyer des requêtes aux *endpoints* : seuls l'hôte obspm.fr peut envoyer des requêtes pour l'inférence, évitant ainsi un risque de surcharge par des requêtes envoyées depuis un autre hôte.

Interactions

L'annotation des images est automatique après envoi de la numérisation ; il est cependant nécessaire de prendre en compte le temps de réponse de la requête du point de vue de l'utilisateur-rice.
L'application EiDA répond actuellement à cette problématique en affichant le message flash suivant : "Le processus de conversion de.s fichier.s PDF en images et/ou d'extraction des images à partir de manifeste.s externe.s est en cours. Veuillez patienter quelques instants pour corriger les annotations automatiques."
On peut envisager de désactiver les boutons de visualisation et d'édition du manifeste jusqu'à réception de la réponse à la requête.



Ressources

Utilisation de PyTorch avec Flask et Django

Deploying PyTorch in Python via a REST API with Flask
Using PyTorch Inside a Django App
Create a Machine Learning API With Django Rest Framework

Sécurité et *tokens*

Flask-JWT-Extended's Documentation
How To Authenticate Flask API Using JWT
Securing a Python Flask API with JWTs
How to Secure a Flask REST API with JSON Web Token?
RESTful Authentication with Flask

YOLOv5

Train Custom Data
PyTorch Hub
YOLOv5 PyTorch Tutorial

docExtractor

docExtractor: An off-the-shelf historical document element extraction

EfficientDet

EfficientDet: Scalable and Efficient Object Detection