**Projet Détection de Signature avec Python**

Le but de ce projet est de détecter la présence d’une signature sur plusieurs types de documents PDF.

Pour davantage d’information, se référer au rapport de stage *2022-23-stage5A-Roig-Lila\_PFE.pdf*, aux présentation PowerPoint ou aux commentaires et description des fonctions implémentées. Il est conseillé de lire le rapport de stage avant de reprendre le projet pour avoir une idée de la démarche globale.

**Mise en route du projet**

Pour faire fonctionner le projet, il vaut avoir au préalable installé Python sur sa machine (voir le OneNote [Data & Décisionnel Notebook](onenote:https://groupebpce.sharepoint.com/sites/DataDcisionnel/SiteAssets/Data%20&%20Décisionnel%20Notebook/) et la partie « A lire pour le prochain alternant »).

Puis créer un nouvel environnement conda spécifique à ce projet, par exemple intitulé *pdf*. Ensuite, installer dans cet environnement *pdf* :

* Jupyter Notebook (voir le OneNote)
* Les librairies listées dans le fichier *requirements.txt* avec la version donnée

L’environnement de travail est prêt.

**Description des répertoires & fichiers**

* Répertoire data : contient un petit nombre de PDF labellisés servant de données d’entrée pour les fonctions de ce projet. Ceci permet de tester avec un petit nombre de PDF les différents cas possibles.
* Répertoire brouillons : contient les anciens codes ayant permis d’implémenter la solution finale. On garde ces codes au cas où on souhaiterait y revenir. Cependant, ils ne sont plus utiles et on pourrait supprimer le répertoire brouillons.

* Notebook run\_detectSignature\_tesseract.ipynb : Notebook permettant de savoir comment utiliser les fonctions implémentés dans le répertoire utilities.
* Répertoire utilities : ensemble des fonctions python .py développées. Toutes les fonctions sont commentées et documentées. Pour savoir comment utiliser ces fonctions .py, il faut exécuter le notebook disponible (notebook run\_detectSignature\_tesseract.ipynb décrit plus bas).

Le répertoire utilities contient les fichiers suivants :

* + Répertoire poppler-23.01.0 : librairie installée dans le cadre de l’installation de la librairie python pdf2image (voir fichier requirements.txt pour l’installation). En effet, avec le pare-feu de l’entreprise, il a fallu rajouter ce répertoire pour faire fonctionner pdf2image.
  + Fichier set\_global.py : fichier permettant de définir les variables globales qui seront ensuite utilisées par tous les autres fichiers .py.
  + Fichier preprocessing.py : module contenant les fonctions pour effectuer le chargement, l’affichage et pré-traitement des images afin que celles-ci soient mieux lisibles par un OCR (Tesseract). Ces fonctions sont utiles pour les modules template.py et detectSignature.py.
  + Fichier template.py : module permettant de définir un PDF modèle pour la détection de signature. Ce modèle n’est à définir qu’une seule fois pour chaque type de PDF.

Ce module demande à l'utilisateur de choisir un PDF servant de modèle. L’utilisateur doit ensuite saisir une liste de mots clés « keyWords\_template » présents sur la page contenant la signature. Enfin, l’utilisateur doit dessiner sur la page du PDF modèle, les rectangles supposés contenir la signature. Ce module calcule la position des rectangles par rapport à la position des mots clés et stocke cette information pour la suite. Le module template.py crée les fichiers suivants en sortie :

* + - Fichier template\_shape.txt : taille du PDF modèle (utilisé dans le module signatureDetect.py)
    - Fichier template\_text.txt : texte de la page contenant la signature sur le PDF modèle (utilisé dans le module signatureDetect.py)
    - Fichier saved\_template.pickle : fichier contenant la position relative des rectangles par rapport à la position des mots clés sur le PDF modèle (utilisé dans le module signatureDetect.py)
  + Fichier signatureDetect.py : Ce module permet de vérifier la signature sur un PDF. Au préalable, l'utilisateur a dû définir un modèle avec le module template.py. Ainsi, la liste de mots clés « keyWords\_template », la taille « template\_shape.txt », le texte « template\_text.txt » du PDF modèle ainsi que la position relative des rectangles par rapport aux mots clés « saved\_template.pickle » ont été calculés avec le module template.py.

Pour chaque PDF à analyser du même type que le modèle, ce module redimensionne le PDF aux dimensions du modèle avec « template\_shape.txt » puis identifie la page contenant la signature avec « template\_text.txt ». Enfin, ce module place sur le PDF à analyser, les nouveaux rectangles supposés contenir la signature en fonction de la position du premier mot clé de « keyWords\_template » trouvé sur la page. Pour cela, il utilise la position relative des rectangles par rapport aux mots clés du modèle avec « saved\_template.pickle ».

Enfin, une fois les rectangles extraits, le pourcentage de pixels continus à l’intérieur des rectangles est calculé pour déterminer si le rectangle contient une signature.

Ce module produit en sortie un fichier de résultat au format .csv. Ce fichier contient les colonnes "file\_name" (pour le nom des fichiers PDF) et "result" (pour le résultat de la classification). Les différents résultats possibles sont : "signature", "no signature" ou "Page contenant la signature non détectée".

* + Répertoire signature\_detect : répertoire contenant les fichiers \_\_init\_\_.py, cropper.py et judger.py permettant de détecter si un rectangle contient une signature en calculant le pourcentage de pixels continus dans ce rectangle. Ces fonctions sont inspirées du package signature\_detect déjà implémenté (voir le fichier requirements.txt pour davantage d’informations)
  + Fichier run\_template.py : fichier permettant d’exécuter depuis le terminal de commande la fonction main du module template.py.
  + Fichier run\_detectSignature.py : fichier permettant d’exécuter depuis le terminal de commande la fonction main du module detectSignature.py.
  + Fichier run\_retrieveErrorFiles.py : Après avoir exécuté le code avec un premier modèle (en utilisant les modules template.py puis detectSignature.py), on obtient un fichier de résultat au format .csv contenant les différents résultats possibles. Le résultat "Page contenant la signature non détectée" peut se produire lorsqu’un PDF à analyser est différent du modèle utilisé. Dans ce cas, le PDF à analyser correspond peut-être à un autre modèle. Par exemple, le document EAI possède 3 versions différentes et donc requiert 3 modèles différents. On doit donc exécuter le code avec le premier modèle puis récupérer les PDF pour lesquels le résultat est "Page contenant la signature non détectée" et les soumettre au deuxième modèle. Puis récupérer les PDF pour lesquels le résultat est "Page contenant la signature non détectée" et les soumettre au troisième modèle. Ainsi, le module run\_retrieveErrorFiles.py prend en entrée le fichier .csv de résultat et crée un fichier .txt contenant le nom des PDF pour lesquels la mention "Page contenant la signature non détectée" apparaît.
* Fichier EAI\_pdfs\_round2.txt : Fichier contenant les noms des documents EAI à soumettre à un deuxième modèle. Obtenu avec l’exécution du fichier run\_retrieveErrorFiles.py. Le fichier round2.txt n’est plus utile à présent mais est conservé à titre d’exemple.
* Fichiers keyWords\_template\_bs.txt, keyWords\_template\_eai.txt, keyWords\_template\_lea.txt, keyWords\_template\_qcfqr.txt : fichiers contenant les mots clés « keyWords\_template » pour chaque type de document PDF (BS, EAI, LEA, QCFQR). Ces mots clés sont utilisés pour calculer la position des rectangles supposés contenir la signature. Ils sont utilisés en entrée des modules template.py et detectSignature.py.
* Documents PDF :
  + template\_bs\_type1.pdf : PDF servant de modèle pour le document BS
  + template\_eai\_type1.pdf, template\_eai\_type2.pdf, template\_eai\_type3.pdf : PDF servant de modèles pour le document EAI. Le EAI possède 3 versions différentes et requiert donc 3 modèles.
  + template\_lea.pdf : PDF servant de modèle pour le document LEA.
  + template\_qcfqr\_type1.pdf, template\_qcfqr\_type2.pdf : PDF servant de modèles pour le document QCFQR. Le QCFQR possède 2 versions différentes et requiert donc 2 modèles.