Année 2022-2023

ING3 ESME Sudria
Projet Data Science
Computer Vision pour la lecture automatique de factures
Encadrant : Wajd MESKINI - wajd.meskini@gmail.com
Keywords : Computer Vision, Text Extraction
Language/Outils : Python, Opencv, Tesseract

Introduction

On se place dans la perspective d'une boîte de conseil qui essaye de diversifier son offre d'extraction de texte à partir d'images. En particulier, le client cherche un système capable d'extraire les frais totaux à partir de factures.

Le client ne veut pas vous donner de données de facture pour des raisons réglementaires. En conséquence, nous allons devoir construire un système d'extraction de texte à partir d'images à partir de photos de reçus de paiement. Notre objectif sera donc de pouvoir extraire le total de factures à partir de reçus de paiement.

Ce projet comprend plusieurs phases :

- **Construction de l'environnement de travail** : Préparer l'environnement de code
- **Traitement d'image** : Préparer l'image à l'extraction de texte
- **Extraction de texte à partir d'image** : Extraire le montant total à payer à partir de la facture
- **Application du système à un jeu de données d'images (Bonus)** : Appliquer le système qu'on aura perfectionné sur une seule image sur un jeu de données complet
- **Déploiement sur une solution web (Bonus)** : Construire une application web pour faire la démonstration du bon fonctionnement process

Puisque notre boîte de conseil ne dispose pas encore d'un pôle d'expertise en Computer Vision, celle-ci a engagé un consultant pour épauler nos efforts. Il a ainsi fourni plusieurs codes de base ainsi que des travaux préliminaires.

Les codes et les données de test sont disponibles sur github dans un repo privé. Celui-ci sera partagé sur demande et est absolument nécessaire au succès du projet.

1 Construction de l'environnement de travail

Objectif : Préparer l'environnement de code

Notez bien que si vous utilisez une technologie de notebooks hébergés comme Google Colab, vous n'aurez par forcément à faire les étapes ci-dessous puisque la plupart des librairies et logiciels utilisés seront déjà installés par défaut.

1. Sur le github, prendre connaissance du fichier requirements.txt. Comment ferez-vous pour installer toutes ces librairies en une seule ligne avec pip en partant du requirements.txt ?
2. Prendre connaissance des librairies installées. A quoi pourraient-elles servir ?
3. Installer Tesseract dans votre ordinateur. Pour cela, téléchargez et installer Tesseract depuis le lien disponible dans la sitographie

4. Lors de l'installation, notez bien le chemin où vous avez installé le logiciel. Par défaut, ça devrait ressembler à ça :

   ```
   C:\Program Files\Tesseract-OCR\tesseract.exe
   ```

5. Lors de votre usage de tesseract dans le code, n'oubliez pas d'initialiser le chemin vers tesseract avec le chemin que vous avez noté dans l'étape suivante.

   ```
   pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR
   \tesseract.exe'
   ```

2 Traitement d'image

Objectif : Préparer une image à l'extraction de texte

Nous allons partir d'un exemple d'image avec pour objectif de tester tous les traitements dessus. A terme, il devrait être possible d'appliquer ces traitements sur les autres images et d'obtenir des résultats. Il faudra donc essayer de tester des paramètres pour s'assurer que le process puisse se généraliser à d'autres images.

2.1 Etapes préliminaires

1. Prenez connaissance du fichier imgutils.py ainsi que des fonctions qui y sont déclarées. Vous serez amenés à les utiliser par la suite.
2. Téléchargez et dézippez le fichier data.zip. Celui-ci contient un échantillon d'images dont vous aurez besoin dans la section 4. Votre objectif sera d'essayer de construire un process qui généralise à un échantillon de ces images là.
3. Lire et afficher l'image sample.jpg grâce à la classe Image de la librairie PIL et matplotlib.

Le consultant en Computer Vision nous informe à ce stade que la librairie Tesseract nécessite que l'image soit bien alignée horizontalement, et que le texte soit en noir sur background blanc avec le minimum de bruit. Sans ce nettoyage préalable, la librairie ne pourra pas lire le texte.

La stratégie que nous allons employer pour nettoyer l'image se fera en trois grandes étapes :

1. Nettoyer l'image pour n'avoir que des silhouettes
2. Extraire le contours du reçu de l'image
3. Changer la perspective de façon à avoir le reçu et le texte bien alignés horizontalement

2.2 Extraction des silhouettes à partir de l'image

Pour extraire les silhouettes de l'image, nous pourrons utiliser les étapes suivantes.

1. Lire l'image sous format opencv.

   Vous pourrez utiliser l'instruction imread
2. Rapetisser l'image. La détection de contours devient plus difficile sur des images plus grandes car les détails sont détectés par les algorithmes et rajoutent de la complexité au traitement.

   Vous pourrez utiliser la fonction opencv_resize de imgutils. Vous pourrez tester différents paramètres, mais vous pourrez commencer par un ratio égal à 500 / image.shape[0]
3. Convertir l'image en noir et blanc.

   Vous pourrez utiliser l'instruction cvtColor de opencv et vérifier si la modification a été faite en l'affichant avec plot_gray de imgutils

4. Rajoutez du flou sur l'image. Le flou permet de se débarrasser du bruit et des petits détails inutiles.
   Vous pourrez utiliser la fonction GaussianBlur de opencv avec un kernel de (3,3) et un sigma de 0. Celle-ci va rajouter du flou et est très utile pour se débarrasser d'un bruit gaussien sur une image
5. Sur l'image floutée, appliquez une dilation des pixels. La dilation permet d'étendre les pixels des objets de façons à ce qu'ils soient plus visibles contre le background.
   Vous pourrez d'abord construire un noyau rectangulaire (5, 5) grâce à la fonction getStructuringElement de opencv, puis appliquer la dilation sur l'image en passant le noyau ainsi construit en paramètre à dilate de opencv. Vous pourrez voir le résultat de votre traitement grâce à plot_gray
6. Nous allons maintenant détecter les extrémités des objets présents dans l'image grâce à la méthode de Canny Edge Detection
   Pour cela, utilisez la fonction Canny de opencv en passant l'image en paramètre, avec les paramètres suivants : (image, 100, 100, apertureSize=3). Vous pourrez en tester d'autres par la suite quand vous arriverez à la section 4

2.3 Extraction le contours du reçu de l'image

Une fois qu'on a les silhouettes de l'image, nous allons devoir essayer d'extraire les contours de l'image. Nous partirons du principe que le contours du reçu sera celui du rectangle le plus grand qu'on a réussi à détecter.

1. Commencez par extraire les contours de l'image
   Nous utiliserons pour cela la fonction findContours de opencv que nous appliquerons sur la silhouette avec les paramètres suivants :(silhouette, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE). Attention, la fonction retourne deux variables.
2. Ensuite, dessinez les contours sur l'image originale et affichez l'image avec les contours détectés
   Nous utiliserons pour cela la fonction drawContours, que nous appliquerons avec ces paramètres (image.copy(), contours, -1, (0,255,0), 3). POur l'affichage, nous utiliserons plot_rgb de imgutils appliquée à l'image avec contours. Ceci nous permettra de voir si nous avons détecté des contours.
3. Nous allons par la suite trier les contours par taille et récupérer les 10 contours les plus grands
   Pour cela, il faudra trier la liste des contours obtenue dans la question précédente en fonction de ce que retourne la fonction cv2.contourArea qui calcule la superficie des formes. Dessinez ensuite les 10 contours les plus larges grâce à drawContours et affichez le résultat grâce à plot_rgb.
4. Chacun des contours obtenus est avant tout un polygone de plusieurs points. Même s'il semble rectangulaire à l'oeil humain, il ne le sera pas en mémoire. Nous devons donc simplifier la forme des contours obtenus de façon à ce qu'ils soient dessinés par le minimum de points possibles
   Nous allons à cet effet utiliser la fonction approxPolyDP qui permet d'approximer le polygone le plus simple possible pour expliquer un contours. Puisque ce code est déjà prêt, il suffira d'exécuter la fonction get_receipt_contour sur les contours. Celle-ci va approximer chacun des contours disponibles puis en retourner celui qui a une forme rectangulaire (c'est à dire qui a 4 extrémités).
   Dessinez ce contours là sur l'image comme fait précédemment et affichez la. Si ce contour est celui du reçu, tout va bien. Sinon, vous ne pourrez pas continuer jusqu'aux étapes suivantes

2.4 Changer la perspective de l'image

Cette étape est à l'origine assez complexe et d'un point de vue théorique, il faut faire les étapes suivantes :

1. Convertir le contour en un tableau de coordonnées ordonnées de la façon suivante : haut-gauche, haut-droit, bas-droit, bas-gauche, correspondant aux coordonnées de chaque extrémité du rectangle.
2. On calcule ensuite les coordonnées d'arrivée de ce rectangle ainsi que la taille de la nouvelle image
3. On donne les coordonnées d'arrivée à cv2.getPerspectiveTransform pour construire une matrice de transformation

4. On utilise cv2.warpPerspective pour faire la transformation

En pratique, il suffira d'exécuter la ligne suivante et d'affiicher l'image transformée grâce à imshow :

```
wrap_perspective(original.copy(), contour_to_rect(receipt_contour))
```

Si tout va bien, on devrait se retrouver avec une image alignée horizontalement. A ce stade, il faudra faire une transformation noir et blanc pour simplifier l'image, et appliquer un "seuil" afin de supprimer le bruit (celui-ci s'appliquera selon l'intensité des pixels, les pixels les moins intenses sont considérés comme du bruit et supprimés). Pour cela, il suffira d'appliquer bw_scanner sur l'image.

3 Extraction de texte à partir d'image

Objectif : Extraire le montant total à payer à partir de la facture

Afin d'extraire le montant total à payer de la facture, il faudra suivre les étapes suivantes :

1. Utiliser pytesseract pour lire le texte à partir de l'image. Renseignez vous sur la documentation de la librairie pour obtenir les fonctions nécessaires
2. Récupérer les coordonnées des textes que vous avez extraits, puis dessinez des rectangles sur l'image. Ceci vous aidera à savoir quels segments ont été lus par Tesseract. Affichez l'image et les rectangles dessinés.

   Si vous trouvez la zone où il y a le montant total, c'est qu'à priori, vous avez réussi. Sinon, il faudra revoir le process de traitement d'image.
3. Le montant total est simplement le chiffre le plus grand. Utiliser des expressions régulières pour parcourir le texte et extraire ce qui ressemble à un montant, puis comparez ces chiffres. Le montant total devrait être le chiffre le plus important.

4 Application du système à un jeu de données d'images (Bonus)

Objectif : Appliquer le système qu'on aura perfectionné sur une seule image sur un jeu de données complet

1. Transformez tout le data process que vous avez construit en une succession de fonctions, de la lecture à l'extraction
2. Rajoutez des loggers pour tracer le progrès du process
3. Rajoutez des blocs de try/except pour capturer les exceptions. Ceci permettra de faire le suivi de votre application et de ne pas arrêter le traitement s'il y a une erreur.
4. Regardez les images présentes dans le fichier zippé et sélectionnez en un sous ensemble relativement homogène. Appliquez votre data process dessus et essayez d'altérer les paramètres de façon à être le plus générique possible et de capter la plus grande part de montants

5 Déploiement sur une solution web (Bonus)

Objectif : Déployer votre solution sur une solution web prête pour une démonstration de votre algorithme

1. Proposer une maquette votre solution
2. Développer une interface web qui intègre les fonctionnalités étudiées
3. Déployer votre solution

6 Sitographie

Lien vers Tesseract

https ://github.com/UB-Mannheim/tesseract/wiki

Lien vers le github (privé, il faudra créer des comptes github et demander l'accès

https ://github.com/atracordis/receipt-ocr-project/

Documentation d'opencv (en anglais)

https ://opencv24-python-tutorials.readthedocs.io/en/latest/