Ecole Spéciale de Mécanique et d'electricité

INGE4 Majeure Intelligence Artificielle

Projet Data Science

Computer Vision pour la lecture automatique de factures

Cyrian Crusson / Jérémie Degorce-Dumas / Marc Valat

2023-2024

Remerciements

Nous tenons tout d'abord à exprimer notre profonde gratitude à notre encadrant de projet, Wajd MESKINI, pour son accompagnement, sa patience et sa confiance tout au long de ce travail.
Ses conseils, son expertise et son soutien ont été d'une grande aide et nous ont grandement aidé pour l'aboutissement de ce projet.

Table des matières

1 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 nous 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 :** Appliquer le système qu'on aura perfectionné sur une seule image sur un jeu de données complet.
- **Déploiement sur une solution web :** 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 en Annexe.

2 Veille scientifique et technique

La Computer Vision ou Vision par Ordinateur est une technique d'intelligence artificielle qui consiste à analyser puis interpréter une image ou une vidéo de la même manière qu'un humain. Autrement dit, elles cherchent à donner à un ordinateur la capacité de voir.

Les algorithmes de Computer Vision utilisent majoritairement des réseaux de neurones, et notamment les CNN (Convolutional Neural Network).

2.1 Technologies et outils de traitement d'image

Le traitement d'image est une discipline de l'informatique et des mathématiques appliquées étudiant les images et leurs transformations, afin d'améliorer leur qualité et/ou d'en extraire de l'information. Traiter l'image, c'est lui rajouter du sens par un procédé automatique.

2.1.1 Tesseract OCR :

La reconnaissance optique de caractères (ROC, ou OCR pour l'anglais optical character recognition), désigne les procédés informatiques pour la traduction d'images de textes imprimés ou dactylographiés en fichiers de texte.
Un logiciel OCR utilise des algorithmes de reconnaissance de motifs pour comparer les images de texte, caractère par caractère, à sa base de données interne.

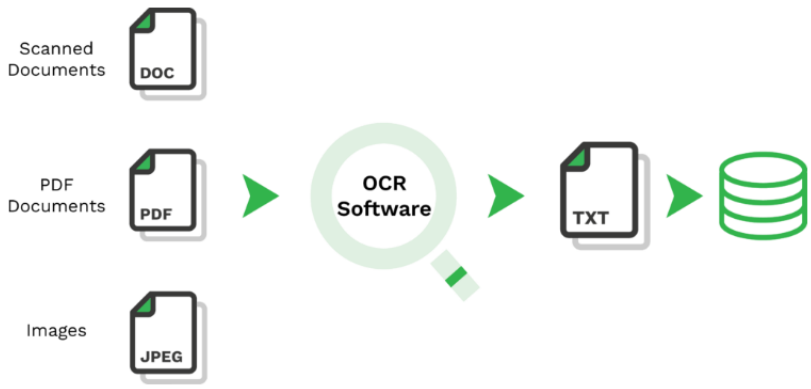


FIGURE 1 – Fonctionnement d'un logiciel OCR

Initialement développé par Hewlett-Packard, son développement a été repris par Google. À l'heure actuelle, il prend en charge la reconnaissance linguistique pour plus de 100 langues. L'une de ses grandes forces est qu'il est compatible avec de nombreux langages et cadres de programmation grâce à des wrappers tels que Pytesseract, également connu sous le nom de Python-Tesseract.

Voila comment Tesseract fonctionne : l'image d'entrée est d'abord soumise à un pré-traitement par le moteur OCR de Tesseract, souvent en association avec OpenCV, afin d'améliorer sa qualité pour des résultats précis. Ensuite, le moteur OCR de Tesseract, procède à l'extraction des données de l'image. Une fois le texte extrait, il peut être converti dans différents formats pris en charge par Tesseract, tels que PDF, texte brut, HTML, TSV et XML.

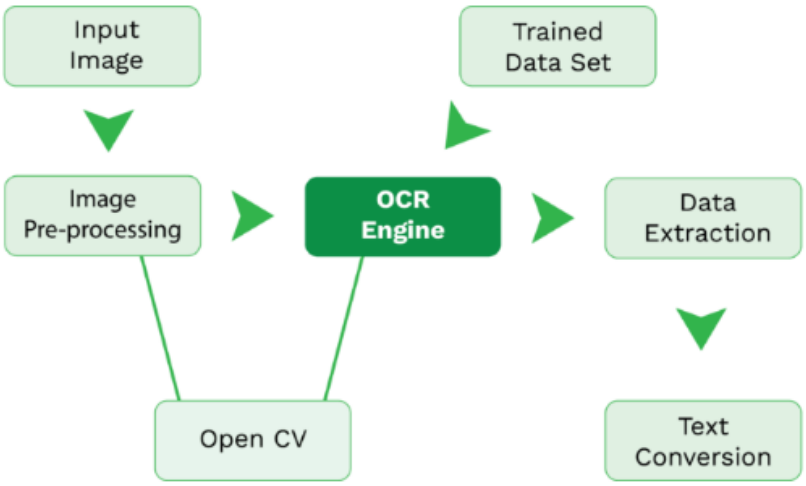


FIGURE 2 – Processus d'OCR de Tesseract

2.1.2 OpenCV (Open Source Computer Vision Library) :

OpenCV est une librairie open source, considérée comme l'outil standard pour la Computer Vision et le traitement d'images.
Développée et publiée en 2000 par Intel qui voulait faciliter le développement d'applications commerciales de Computer Vision, en fournissant une infrastructure commune et ouverte accessible à tous.

OpenCV permet egalement d'améliorer l'extraction de données des moteurs OCR tels que Tesseract :

- Détection d'objets – permet à la solution de détecter une grande variété d'objets
- Réseaux neuronaux profonds (DNN) – permet à la solution de classer les images
- Traitement de l'image – permet à la solution de mieux traiter les images d'entrée grâce à diverses techniques telles que la détection des bords, la manipulation des pixels, le redressement, etc.

Autrement dis sans OpenCV, Tesseract n'est pas aussi sophistiqué que ce que l'on pourrait attendre des solutions d'OCR actuelles, car beaucoup d'entre elles appliquent diverses technologies d'IA.

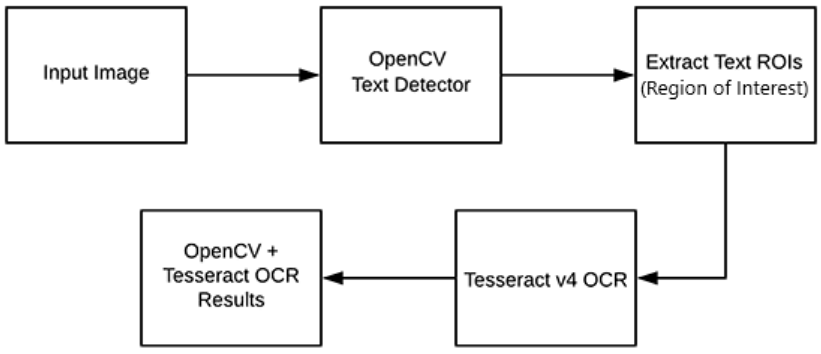


FIGURE 3 – OpenCV + Tesseract

2.2 Algorithmes de traitement d'images :

2.2.1 Détection de contours et segmentation d'images

La détection de contours et la segmentation d'images sont des étapes essentielles dans le traitement d'images, particulièrement lorsqu'il s'agit d'extraire des informations précises telles que les montants sur des reçus de paiement. Les algorithmes de détection de contours, comme l'algorithme de Canny, jouent un rôle crucial en identifiant les bords des objets dans une image. Cet algorithme fonctionne en détectant les variations de l'intensité des pixels, ce qui permet de repérer les transitions nettes entre différentes zones de l'image. En appliquant un filtre de Canny, on obtient une carte des contours qui met en évidence les structures significatives, facilitant ainsi la reconnaissance ultérieure des éléments importants tels que le texte ou les bordures des reçus.

La segmentation d'images, quant à elle, consiste à diviser une image en plusieurs segments ou régions distinctes. Chaque segment représente une partie de l'image ayant des caractéristiques similaires, ce qui simplifie l'analyse et le traitement. Pour les reçus de paiement, la segmentation permet d'isoler les différentes sections du document, comme les zones contenant du texte et celles vides. Cette étape est fondamentale pour focaliser les efforts de traitement sur les parties pertinentes de l'image, rendant le processus d'extraction de texte plus efficace et précis. Les techniques de segmentation peuvent inclure des méthodes basées sur les contours détectés, ainsi que des approches plus avancées utilisant des modèles de machine learning pour distinguer les différentes zones de manière automatique et adaptative.

2.2.2 Filtrage et nettoyage d'images

Le filtrage et le nettoyage d'images sont des processus essentiels pour préparer une image avant l'extraction de texte, assurant ainsi une meilleure reconnaissance par les algorithmes OCR. Les techniques de filtrage, comme l'application d'un flou gaussien, jouent un rôle important en réduisant le bruit et les petits détails indésirables dans l'image. Le flou gaussien fonctionne en lissant les variations d'intensité des pixels, ce qui aide à éliminer les artefacts qui pourraient perturber la détection des contours et la lecture du texte.

La dilatation des pixels est une autre technique couramment utilisée pour améliorer la qualité des images. En étendant les pixels des objets présents dans l'image, cette méthode rend les structures importantes plus visibles et distinctes par rapport à l'arrière-plan. Cela est particulièrement utile pour renforcer les bords et les contours détectés, facilitant ainsi la segmentation et l'analyse.

Après le filtrage initial, la transformation en noir et blanc est une étape cruciale pour simplifier l'image et rendre le texte plus lisible. En convertissant l'image en niveaux de gris, on peut appliquer des seuils locaux pour distinguer le texte du fond de manière plus efficace. Le seuil local est une technique qui ajuste dynamiquement le niveau de gris pour chaque région de l'image, en fonction des variations d'intensité locale. Cela permet de traiter des images avec des éclairages non uniformes ou des contrastes variables, améliorant ainsi la précision de l'OCR.

En combinant ces techniques de filtrage et de nettoyage, on obtient une image optimisée pour l'extraction de texte, où les éléments indésirables sont minimisés et les structures importantes sont mises en avant. Cela augmente la probabilité que l'OCR identifie correctement les caractères, aboutissant à une extraction plus fiable et précise des informations cruciales, telles que les montants totaux sur les reçus de paiement.

3 Méthodologie de résolution

3.1 Construction de l'environnement de travail

La première étape de notre projet va consistr à établir un environnement de travail adéquat pour le développement et l'exécution de notre code. Pour ce faire, un fichier requirements.txt qui énumère toutes les librairies nécessaires nous à été fournit en annexe.
Par la suite, on installera le logiciel Tesseract OCR, en veillant à configurer correctement son chemin d'accès dans notre code pour permettre une utilisation fluide de ses fonctionnalités d'extraction de texte.

3.2 Traitement d'images

L'objectif de cette partie est de préparer les images pour l'extraction de texte. Dans un premier temps nous travaillerons sur un exemple d'image afin de tester et affiner nos traitements, avec pour but de pouvoir les appliquer à d'autres images par la suite.

3.2.1 Etapes préliminaires

Une étude du fichier imgutils.py (fournit encore une fois en annexes) sera cruciale. Il repertorie les fonctions utiles pour le traitement d'images.
Ensuite, la lecture et l'affichage des images fournies, avec les différentes libraries, permettront de poser les bases pour les traitements plus complexes à venir.

3.2.2 Extraction des silhouettes à partir de l'image

Pour extraire les silhouettes, nous avons prevu de suivre une série d'étapes visant à simplifier l'image et à mettre en évidence ses contours principaux.
Dans un premier temps redimensionner l'image pour faciliter la détection des contours, puis la convertir en noir et blanc.
Dans un second temps, appliquer un flou gaussien pour réduire le bruit et utiliser une technique de dilatation des pixels pour rendre les objets plus visibles.
Enfin, utiliser la méthode Canny Edge Detection, pour bien isoler les silhouettes présentes.

3.2.3 Extraction le contours du reçu de l'image

Une fois les silhouettes extraites, notre objectif sera d'isoler le contour du reçu dans l'image. Pour ce faire, nous allons extraire tous les contours présents et on les dessinera sur l'image originale pour vérification.
Ensuite, on triera ces contours par taille et on gardera les plus grands, supposant que le plus grand rectangle représentera le reçu. Afin de simplifier ces contours en polygones facilement exploitables, on utilisera une méthode d'approximation. Cette étape permettra d'identifier précisément le contour rectangulaire du reçu.

3.2.4 Changer la perspective de l'image

Pour que le texte du reçu soit bien aligné horizontalement, il sera nécessaire de corriger la perspective de l'image.
Nous convertirons le contour détecté en coordonnées ordonnées, calculerons les nouvelles dimensions de l'image, et appliquerons une transformation de perspective. Une fois l'image correctement alignée, nous la convertirons en noir et blanc et appliquerons un seuil pour éliminer le bruit résiduel, rendant ainsi l'image prête pour l'extraction de texte.

3.3 Extraction de texte

L'objectif de cette étape sera d'extraire le montant total à payer à partir de la facture. Nous utiliserons la librairie PyTesseract pour lire le texte contenu dans l'image transformée.
Pour vérifier l'exactitude de l'extraction, nous dessinerons des rectangles autour des segments de texte identifiés par Tesseract.
Enfin, nous emploierons des expressions régulières pour identifier et extraire le montant total, en nous basant sur le chiffre le plus grand trouvé dans le texte (ou d'autres critères si necessaire).

3.4 Application à un jeu de données d'images

Par la suite nous appliquerons notre processus à l'ensemble d'images du dataset.
Nous transformerons notre flux de traitement en une série de fonctions et ajouterons des mécanismes de journalisation et de gestion des erreurs pour assurer le suivi et la résilience du processus. Nous appliquerons ensuite ces traitements à un sous-ensemble d'images, en ajustant les paramètres pour maximiser la généralisation et la précision de l'extraction des montants.

3.5 Déploiement sur une solution web

Pour rendre notre solution accessible et démontrable, nous déploierons notre système sur une plateforme web.
Nous utiliserons Streamlit, une plateforme open-source qui permet de transformer des scripts Python en applications web interactives et conviviales.

4 Développement de la solution

4.1 Installation des librairies et de Tesseract

Installation des libraries : Nous avons commencé par prendre connaissance du fichier requirements.txt disponible sur le GitHub qui nous a été fournit en annexe. Il répertorie toutes les libraries nécessaires pour bien débuter le projet. Pour installer toutes les librairies en une seule ligne, nous avons utilisé la commande suivante :

```
pip install -r requirements.txt
```

Vérification des librairies installées : Chaque librairie installée a été examinée pour comprendre son utilité dans le cadre du projet.

- NumPy (Numerical Python) est une bibliothèque fondamentale pour le calcul scientifique en Python. Elle offre un support pour les tableaux multidimensionnels (ou matrices) et une collection de fonctions mathématiques pour opérer sur ces tableaux de manière efficace.
- Pandas est une bibliothèque utilisée pour la manipulation et l'analyse de données. Elle fournit des structures de données de haut niveau, comme les DataFrames, et des outils pour travailler avec des données structurées (similaires aux tableaux en SQL ou aux feuilles de calcul Excel).
- Scikit-image est une bibliothèque pour le traitement d'images. Elle est basée sur NumPy et offre une large gamme d'algorithmes pour le filtrage, la segmentation, la mesure, la transformation et bien d'autres opérations sur les images.
- OpenCV (Open Source Computer Vision Library) est une bibliothèque de vision par ordinateur. Elle contient des fonctions pour la capture, le traitement et l'analyse d'images et de vidéos. Elle est largement utilisée pour des tâches telles que la reconnaissance faciale, la détection d'objets et la suivi de mouvement.
- Pillow est une bibliothèque de traitement d'images, qui est une continuation améliorée de la bibliothèque Python Imaging Library (PIL). Elle permet de créer, modifier et sauvegarder différents formats d'images, ainsi que de faire des opérations basiques comme le redimensionnement, la rotation et le filtrage.

— Pytesseract est un wrapper pour le moteur de reconnaissance optique de caractères (OCR) Tesseract. Il permet de lire et de convertir du texte à partir d'images en texte brut, rendant ainsi les images de texte lisibles et modifiables par des programmes.

Chacune de ces bibliothèques joue un rôle dans la manipulation, l'analyse et le traitement de données numériques et d'images en Python.

Installation de Tesseract : Tesseract est un logiciel OCR open source qui peut être utilisé pour extraire du texte à partir d'images. Il reconnnait plus de 100 langues et est compatible avec de nombreux langages de programmation et cadres de travail.
En plus d'être gratuit, il peut être intégré et couplé à des bibliothèques OCR Python, ce qui permet d'accéder à des avantages tels que la vision par ordinateur (Computer Vision – CV) en temps réel et à des fonctions de traitement d'image.
L'installation a été faite a partir du lien en annexe. Le chemin d'installation ressemble a ceci :

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

Il est important de garder en tete le chemin d'installation du logiciel car il est necessaire de le renseigner dans le code comme ceci :

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

4.2 Préparation et traitement des images

Dans cette section, nous détaillerons les étapes de préparation et de traitement d'images pour l'extraction de texte à partir de reçus de paiement. Le processus commence par la lecture de l'image, puis la préparation de celle-ci afin de rendre le texte plus lisible pour les algorithmes de reconnaissance optique de caractères (OCR).

4.2.1 Lecture et affichage de l'image

Nous commençons par lire et afficher l'image de base pour en comprendre la structure et les caractéristiques. Pour ce faire, nous utilisons la bibliothèque Pillow (PIL) pour charger l'image et Matplotlib pour l'afficher.

```
from PIL import Image
import matplotlib.pyplot as plt

# Lecture de l'image
im = Image.open("sample.jpg")

# Affichage de l'image
plt.imshow(im)
plt.show()
```

4.2.2 Extraction des silhouettes à partir de l'image

La première étape du traitement d'image consiste à simplifier l'image pour extraire les silhouettes des objets présents. Cela inclut le redimensionnement de l'image, la conversion en noir et blanc, l'application d'un flou gaussien, et l'utilisation de techniques de dilatation des pixels pour améliorer la visibilité des objets. Ces étapes permettent respectivement de :

— Réduire la taille de l'image pour faciliter le traitement.
— Transformer l'image en niveaux de gris pour simplifier les calculs.
— Réduire le bruit et les petits détails.
— Étendre les pixels pour rendre les contours plus visibles.

Puis pour detecter les contours on utilise l'algorithme de détection de contours de Canny pour isoler les silhouettes.

```
import cv2
from imgutils import opencv_resize, plot_gray

# Lecture de l'image avec OpenCV
imopencv = cv2.imread("sample.jpg")

# Redimensionnement de l'image
imresized = opencv_resize(imopencv, (500.0 / imopencv.shape[0]))

# Conversion en noir et blanc
imbw = cv2.cvtColor(imresized, cv2.COLOR_BGR2GRAY)

# Application du flou gaussien
imblur = cv2.GaussianBlur(imbw, (3, 3), 0)

# Dilatation des pixels
rectkernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))
dilation = cv2.dilate(imblur, rectkernel, iterations=1)

# Détection des contours
edge = cv2.Canny(dilation, 100, 100, apertureSize=3)

# Affichage de l'image avec contours
plot_gray(edge)
```

4.2.3 Extraction du contour du reçu

Une fois les contours détectés, nous devons extraire spécifiquement le contour du reçu. Pour ce faire, nous utilisons les techniques suivantes :

- **Extraction des contours :** Utiliser la fonction findContours d'OpenCV pour identifier tous les contours dans l'image.
- **Tri et sélection des contours :** Trier les contours par taille et sélectionner les plus grands, en supposant que le plus grand rectangle représente le reçu.
- **Approximation des contours :** Simplifier les contours en polygones pour faciliter leur manipulation.

```
# Extraction des contours
contours, hierarchy = cv2.findContours(edge, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
sorted_contours = sorted(contours, key=cv2.contourArea, reverse=True)
largest_contours = sorted_contours[:10]

# Dessiner les 10 contours les plus larges
draw_largest_contours = cv2.drawContours(imresized.copy(),
largest_contours, -1, (0, 255, 0), 3)

# Fonction pour obtenir le contour du reçu
def get_receipt_contour(contour):
    for c in contour:
        approx = approximate_contour(c)
        if len(approx) == 4:
            return approx
    return None

receipt_contour = get_receipt_contour(largest_contours)
if receipt_contour is not None:
    cv2.drawContours(imresized.copy(), [receipt_contour], -1,
    (0, 255, 0), 3)
    plot_rgb(imresized.copy())
```

4.2.4 Changement de perspective de l'image

Pour que le texte du reçu soit bien aligné horizontalement, il est nécessaire de corriger la perspective de l'image. Les étapes sont les suivantes :

- **Conversion du contour en coordonnées :** Convertir les points du contour en un tableau de coordonnées ordonnées.
- **Calcul des dimensions de la nouvelle image :** Déterminer les dimensions de l'image transformée.
- **Transformation de perspective :** Appliquer la transformation de perspective pour aligner le reçu.
- **Transformation noir et blanc et application d'un seuil :** Simplifier l'image pour l'extraction de texte.

4.3 Extraction de texte à partir d'image

L'objectif de cette étape est d'extraire le montant total à payer à partir de l'image du reçu transformée. Nous utiliserons la bibliothèque PyTesseract pour lire le texte de l'image et identifier le montant total.

4.3.1 Extraction initiale du texte

Utiliser PyTesseract pour extraire le texte de l'image transformée en utilisant la fonction 'image to string'.

```
text = pytesseract.image_to_string(scanned, lang='eng')
print("Texte extrait:", text)
```

4.3.2 Vérification de l'extraction de texte

Pour vérifier l'exactitude de l'extraction, nous dessinerons des rectangles autour des segments de texte identifiés par Tesseract en utilisant la fonction 'image to data'.

```
data = pytesseract.image_to_data(scanned,
output_type=pytesseract.Output.DICT)
n_boxes = len(data['level'])
for i in range(n_boxes):
    (x, y, w, h) = (data['left'][i], data['top'][i], data['width'][i],
    data['height'][i])
    cv2.rectangle(scanned, (x, y), (x + w, y + h), (0, 255, 0), 2)

plt.figure(figsize=(10, 10))
plt.imshow(scanned, cmap='gray')
plt.title("Text Detection")
plt.show()
```

4.3.3 Extraction du montant total

Pour extraire le montant total, nous utiliserons des expressions régulières pour rechercher les montants dans le texte extrait, puis identifier le plus grand montant trouvé.

```
import re

# Expression régulière pour détecter les montants
amount_pattern = re.compile(r'(\d+\.\d{2})\s?EUR')
matches = amount_pattern.finditer(text)

# Extraire et comparer les montants
amounts = []
for match in matches:
    amount = float(match.group(1))
    if amount < 10000:  # Filtrer les montants irréalistes
        amounts.append(amount)

if amounts:
    total_amount = max(amounts)
    print("Total Amount to Pay:", total_amount)
else:
    print("No valid amounts detected in the text.")
```

4.4 Application à un jeu de données d'images

4.4.1 Adaptation du processus à un jeu de données complet

Pour tester et valider l'efficacité de notre solution d'extraction de texte, nous avons appliqué notre pipeline de traitement à un jeu de données contenant plusieurs images de reçus. Les images utilisées sont nommées séquentiellement de 1132-receipt.jpg à 1198-receipt.jpg.

Liste des étapes réalisées pour l'adaptation :

— **Construction de la liste des chemins d'accès aux images :** Nous avons généré une liste de chemins d'accès pour chaque image du jeu de données en les ajoutant manuellement dans une liste Python. Cela nous permet de traiter chaque image séquentiellement.

```
image_paths = [
    './data/1132-receipt.jpg',
    './data/1133-receipt.jpg',
    './data/1136-receipt.jpg',
    './data/1138-receipt.jpg',
    './data/1140-receipt.jpg',
    './data/1145-receipt.jpg',
    './data/1146-receipt.jpg',
    './data/1147-receipt.jpg',
    './data/1148-receipt.jpg',
    './data/1151-receipt.jpg',
    './data/1153-receipt.jpg',
    './data/1155-receipt.jpg',
    './data/1158-receipt.jpg',
    './data/1163-receipt.jpg',
    './data/1164-receipt.jpg',
    './data/1175-receipt.jpg',
    './data/1183-receipt.jpg',
    './data/1188-receipt.jpg',
]
```

— **Création d'une fonction pour traiter plusieurs images :** Une fonction nommée process_multiple_images a été développée pour traiter chaque image dans la liste, appliquer le pipeline de traitement et extraire le texte. Les résultats sont stockés dans un dictionnaire pour une analyse ultérieure.

```
def process_multiple_images(image_paths):
    results = {}
    for image_path in image_paths:
        print(f"Processing {image_path}...")
        total = extract_text_from_receipt(image_path)
        results[os.path.basename(image_path)] = total
    return results
```

— **Traitement séquentiel des images :**
Chaque image est lue et traitée en appliquant les mêmes étapes de prétraitement, de détection des contours, de transformation de perspective, et d'extraction de texte via OCR.

— **Extraction du montant total :**
Pour chaque image, le texte extrait est analysé pour identifier et extraire la valeur monétaire associée au "total". Cela permet de vérifier l'efficacité du système sur un ensemble diversifié d'images de reçus.

4.4.2 Optimisation et généralisation du process

Comme pour le premier code (sur le sample.png) l'application de filtres adaptatifs et de transformations de perspective a été optimisée pour améliorer la qualité des images et la précision de l'OCR.

Un algorithme robuste pour détecter et corriger l'orientation des images a été mis en place pour s'assurer que les reçus sont toujours alignés verticalement avant l'extraction du texte.

Le code a été conçu de manière modulaire pour permettre une adaptation facile à d'autres types de documents ou d'autres ensembles de données.

Les méthodes utilisées pour l'extraction de texte sont suffisamment générales pour être appliquées à divers formats et dispositions de reçus.

L'application du processus à l'ensemble des images a permis de démontrer la robustesse et l'efficacité de notre solution. Voici un aperçu des résultats obtenus :

Ces résultats montrent que notre pipeline est capable de traiter un grand nombre d'images de reçus, d'extraire le texte et d'identifier les montants totaux avec une précision acceptable. Les images qui n'ont pas pu être traitées correctement ont été identifiées, ce qui permet une analyse ultérieure pour améliorer le pipeline.

4.5 Déploiement sur une solution web

4.5.1 Détails sur la conception et le développement de l'interface web

Pour la solution web nous avons opté, suivant les recommandations de notre encadrant, pour le framework open-source Streamlit, utilisant le langage de programmation Python.

Streamlit est un outil conçu pour aider les développeurs à créer rapidement des applications web interactives pour la visualisation de données. Son objectif principal est de simplifier et d'accélérer le développement de ces applications, en permettant aux utilisateurs de créer des interfaces utilisateurs élégantes pour des scripts Python avec peu de code supplémentaire.

Il est donc facile et rapide à prendre en main et mettre en place et à la capacité à rafraîchir automatiquement l'interface utilisateur à chaque fois que le code source est modifié ce qui rend la modification agréable et réactive.

En revanche la solution est idéale à petite échelle et ne permet pas une personnalisation très poussée, mais dans notre cas cela est parfait pour créer notre démonstration.

Le script original de traitement d'image est adapté pour fonctionner au sein de l'application Streamlit. Cela implique d'enlever toute visualisation directe avec Matplotib et de remplacer cette dernière par des composants Streamlit pour afficher des images et des résultats.

Étape 1 :Création de l'interface utilisateur avec Streamlit :
Le code est encapsulé dans des fonctions qui peuvent être appelées par l'interface Streamlit. Les résultats et les images sont affichés à l'aide de widgets comme « st.image » pour les images et « st.write » ou « st.markdown » pour le texte.

Étape 2 : Intégration des fonctionnalités de traitement d'image :
Utilisation de « st.columns » pour organiser l'affichage en plusieurs colonnes contenant les résultats des différentes fonctions de traitement d'image, rendant l'interface plus organisée et adaptée pour la démonstration.

Étape 3 : Interactivité avec le chargement de fichiers :
« st.file_uploader » permet aux utilisateurs de télécharger des images directement dans l'application, permettant aussi le « drag&drop », rendant le processus dynamique et interactif. L'image téléchargée est traitée en temps réel, affichant les étapes successives du traitement de l'image et les résultats de l'OCR.

4.5.2 Processus de déploiement et démonstration de la solution

Avec notre code Streamlit le déploiement est simple, une commande dans le terminal suffit :

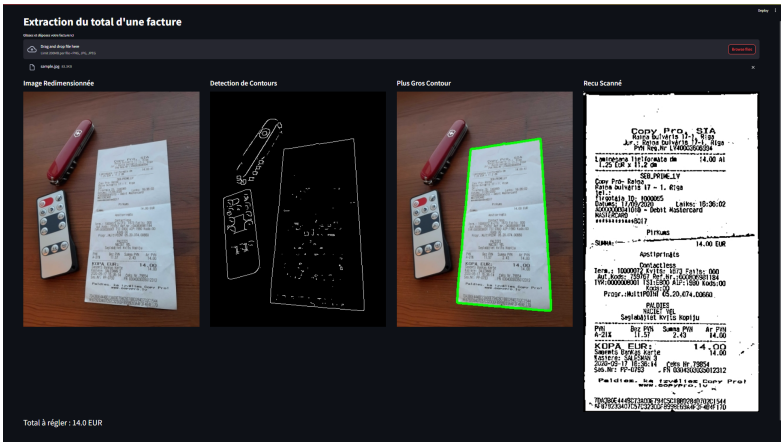
```
streamlit run nomDuFichier.py
```

Et nous obtenons le message suivant :

```
You can now view your Streamlit app in your browser
  Local URL: http://localhost:----
  Network URL: http://---.---.-.---:----
```

Étape 5 : Analyse et présentation des résultats :
Affichage du montant total détecté, sinon affichage d'un message indiquant l'absence de détection d'un montant valide.

Étape 6 : Gestion des erreurs et des cas particuliers :
Messages clairs sont affichés pour guider l'utilisateur en cas de problème ou de résultats inattendus.



5 Résultats obtenus et analyses

5.1 Résultats de l'extraction de texte sur les images tests

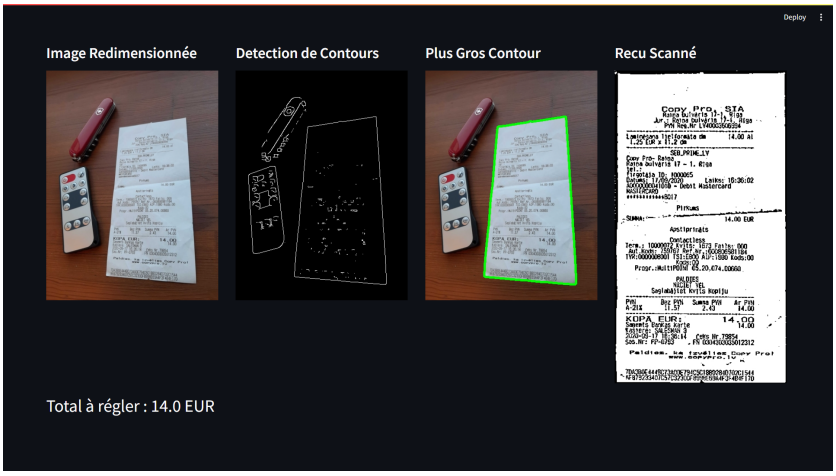


FIGURE 4 – Extraction de texte sur l'image "sample.png"

```
Results: {'1132-receipt.jpg': '61.80', '1133-receipt.jpg': '146.57', '1136-receipt.jpg': '13.13', '1138-receipt.jpg': '25.94', '1140-
receipt.jpg': '25.47', '1145-receipt.jpg': '65.14', '1146-receipt.jpg': '13.26', '1147-receipt.jpg': '13.00', '1148-receipt.jpg':
'13.07', '1151-receipt.jpg': 13.53, '1153-receipt.jpg': 313.49, '1155-receipt.jpg': 129.75, '1158-receipt.jpg': '780.45', '1163-
receipt.jpg': '41.32', '1164-receipt.jpg': 22.7, '1175-receipt.jpg': '7.56', '1183-receipt.jpg': '10.00', '1188-receipt.jpg': '26.96'}
```

FIGURE 5 – Extraction du montant sur les images du dataset

5.2 Analyse de la précision et de l'efficacité du système

Codes majeurs	Echantillon	Reussite	Taux de reussite
1	55	1	1,818181818
2	55	1	1,818181818
3	55	1	1,818181818
4	55	0	0
5	55	0	0
6	55	5	9,090909091
7	55	10	18,18181818
8	55	6	10,90909091
9	55	21	38,18181818
Potentiel facilement atteignable	55	40	72,72727273
Potentiel difficilement atteignable	55	53	96,36363636

FIGURE 6 – Premier tableau de performance de nos codes

5.3 Difficultés rencontrées et solutions apportées

Dans le développement de notre solution d'extraction de texte à partir d'images de reçus, plusieurs difficultés ont été rencontrées. Chacune de ces difficultés a été abordée avec des solutions spécifiques pour améliorer la robustesse et l'efficacité du système. Voici un aperçu des principaux défis et des mesures prises pour les surmonter :

5.3.1 Variabilité des formats de reçus

Problème : Les reçus peuvent varier considérablement en termes de format, de disposition, de police de caractères et de qualité d'impression. Cette variabilité complique l'extraction fiable des informations nécessaires, telles que le montant total.

Solution : Prétraitement adaptatif : Nous avons mis en œuvre des techniques de prétraitement adaptatives, telles que la binarisation adaptative et le filtrage bilatéral, pour améliorer la qualité des images indépendamment des variations de format et de qualité. Détection des contours et transformation de perspective : Nous avons utilisé des méthodes robustes pour détecter les contours et effectuer des transformations de perspective, garantissant que les reçus soient correctement alignés et redressés avant l'extraction de texte. Utilisation de Tesseract avec configurations spécifiques : Des configurations

spécifiques de Tesseract (par exemple, –oem 3 –psm 6) ont été utilisées pour optimiser la reconnaissance des caractères dans divers formats de reçus.

5.3.2 Problèmes d'orientation et d'alignement

Problème : Les images de reçus peuvent être capturées sous différents angles et orientations, ce qui complique l'extraction correcte du texte.

Solution : Détection et correction de l'orientation : Un algorithme de détection de l'orientation a été implémenté pour détecter l'angle d'inclinaison des reçus. Une correction de l'orientation a ensuite été appliquée pour s'assurer que les reçus soient verticalement alignés avant l'extraction de texte. Rotation automatique : La fonction orient_vertical a été utilisée pour ajuster automatiquement l'orientation des reçus en fonction de leurs dimensions, garantissant ainsi une meilleure alignement pour l'OCR.

5.3.3 Bruit et artéfacts dans les images

Problème : Les images de reçus peuvent contenir du bruit et des artefacts, rendant difficile l'extraction précise du texte.

Solution : Filtrage et lissage des images : Des techniques de filtrage, comme le flou gaussien et le filtre bilatéral, ont été appliquées pour réduire le bruit tout en conservant les bords des caractères. Dilatation et érosion : Des opérations de dilatation et d'érosion ont été utilisées pour améliorer la netteté des contours et réduire les artefacts. Binarisation adaptative : La binarisation adaptative a été utilisée pour convertir les images en noir et blanc, améliorant ainsi le contraste et facilitant l'extraction de texte par OCR.

5.3.4 Extraction incorrecte des montants

Problème : L'extraction du montant total des reçus peut être perturbée par des erreurs OCR ou par la présence de montants similaires (comme les sous-totaux et les taxes).

Solution : Filtrage des lignes de texte : Un filtrage des lignes de texte contenant le mot "total" a été mis en place pour se concentrer uniquement sur les montants pertinents. Les lignes contenant des termes comme "subtotal" ont été explicitement ignorées. Extraction des valeurs numériques : Des expressions régulières ont été utilisées pour identifier et extraire les valeurs

monétaires avec précision. La ligne contenant le montant total a été priorisée pour éviter les erreurs d'extraction dues à des montants similaires.

5.3.5 Performance et efficacité

Problème : Le traitement de nombreuses images de reçus peut être chronophage et exiger des ressources computationnelles importantes.

Solution : Traitement par lots : Le traitement des images a été optimisé pour fonctionner par lots, permettant ainsi une gestion plus efficace des ressources et une réduction du temps de traitement global. Optimisation du code : Le code a été optimisé pour réduire les redondances et améliorer l'efficacité des algorithmes utilisés. Des techniques de parallélisation pourraient également être envisagées pour de futures améliorations.

6 Conclusion

Ce rapport a détaillé les étapes de développement et de mise en œuvre d'un système de reconnaissance optique de caractères (OCR) destiné à extraire les montants totaux de reçus de paiement à partir d'images. Le projet s'est articulé autour de plusieurs phases clés, de la préparation de l'environnement de travail à l'application du système sur un jeu de données complet, en passant par le traitement des images et l'extraction de texte.

Dans un premier temps, nous avons configuré un environnement de travail adéquat, installant les librairies nécessaires et le logiciel Tesseract OCR, essentiel pour la reconnaissance de texte. La phase de traitement d'images a impliqué une série d'opérations visant à simplifier et améliorer la qualité des images pour maximiser la précision de l'extraction de texte. Cela incluait la conversion des images en niveaux de gris, l'application de flous pour réduire le bruit, et la correction de la perspective pour garantir un alignement optimal du texte.

L'extraction de texte à partir des images transformées a été réalisée en utilisant PyTesseract, qui a permis de lire et d'interpréter le texte présent dans les images de reçus. Nous avons également implémenté des techniques de vérification visuelle pour confirmer l'exactitude de l'extraction, en dessinant des rectangles autour des segments de texte détectés. Enfin, nous avons utilisé des expressions régulières pour identifier et extraire le montant total à payer, en nous assurant que les montants détectés étaient réalistes et pertinents.

L'élaboration d'un premier code sur une image sample nous a permit de prendre nos marque avec les différents outils du traitement d'image avec python.
Par la suite nous avons rencontré plusieurs difficultés en voulant applique notre code à un dataset de plusieurs images.
En surmontant ces défis, nous avons pu développer une solution robuste et adaptable pour l'extraction de texte à partir d'images de reçus. Chaque difficulté a été abordée de manière ciblée, en utilisant des techniques avancées de traitement d'image et des configurations optimisées pour l'OCR. Les solutions mises en œuvre ont permis d'améliorer considérablement la précision et l'efficacité de notre système, rendant notre pipeline d'extraction de texte performant face à une grande variété de formats de reçus et de conditions d'image.

Et enfin pour rendre notre solution accessible et démontrable, nous avons envisagé de la déployer sur une plateforme web interactive, utilisant Streamlit pour créer une interface conviviale et intuitive. Cette démarche permet non seulement de présenter les capacités de notre système, mais aussi de faciliter son utilisation par des utilisateurs non techniques.

7 Annexes

Lien vers le github du projet :
https ://github.com/CyrianCrPro/MasterProjectFACTURE

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

Documentation générale
https ://datascientest.com/
https ://pyimagesearch.com/
https ://fr.wikipedia.org/