Project report

Information extraction from scanned documents

1. Introduction

In a professional context where invoice processing represents a significant administrative burden, automating the extraction of essential data becomes a major efficiency issue. Our project is part of this approach, offering a comprehensive solution for automatically extracting key information from scanned invoices.

The main objective is to quickly process a large volume of documents to extract structured data such as invoice numbers, dates, customer identities, and monetary values (net amount, VAT, gross amount). This automation not only saves considerable time compared to manual processing, but also significantly reduces the risk of human error during data entry.

This report presents all the stages of the project, from document loading and pre- processing to data extraction and structuring, and the creation of an intuitive user interface. We will also detail the technical choices made, the difficulties encountered, and the solutions implemented.

2. Project architecture

The project was structured into three main modules, each corresponding to a key stage in the extraction process:

File structure

- **project.py**: Module responsible for steps 1 to 3 (PDF conversion, image preprocessing, OCR)
- **data_strct.py**: Module managing steps 4 and 5 (extraction of structured data, storage)
- **app.py**: Module for step 6 (web user interface with Streamlit)

Workflow

The processing pipeline follows a sequential logic:

1. **Input**: Documents in PDF or image format (JPG, PNG, TIFF)
2. **Conversion**: PDFs are converted to images (project.py)
3. **Preprocessing**: Image enhancement to optimize OCR (project.py)

4. **OCR**: Raw text extraction with EasyOCR (project.py)
5. **Structured extraction**: Use of regular expressions to identify relevant data (data_strct.py)
6. **Storage**: Saving to an SQLite database and exporting to multiple formats (data_strct.py)
7. **User interface**: Web application for interacting with the system (app.py)

Technologies used

- **Languages and frameworks**: Python, Streamlit
- **Image processing**: OpenCV (cv2)
- **OCR**: EasyOCR
- **Data manipulation**: Pandas, NumPy
- **Regular expressions**: Re
- **Database**: SQLite
- **PDF conversion**: pdf2image
- **Export formats**: CSV, JSON, HTML

3. Steps 1 to 3 – Conversion, Preprocessing, and OCR

This section details the first steps of the project, which consist of preparing the documents and extracting the plain text.

3.1 Loading and converting documents

The project.py module implements document detection and conversion with the following features:

- **Automatic file type detection**: The count_file_types() function analyzes the source directory and counts the files by extension to identify the formats to be processed.
- **PDF to image conversion**: The convert_pdfs_to_images() function uses the pdf2image library to convert each page of PDF documents into PNG images, allowing them to be further processed by OCR algorithms.

Python:

```
def convert_pdfs_to_images(folder_path, output_folder=None, dpi=200): """
  Convert each page of every PDF in folder_path to images.
  Saves images in output_folder or a subfolder 'pdf_images' by default. """
  folder = Path(folder_path)
  out_base = Path(output_folder) if output_folder else folder / 'pdf_images'
  out_base.mkdir(exist_ok=True)

  for pdf_file in folder.glob('*.pdf'):
    doc_name = pdf_file.stem
    pages = convert_from_path(str(pdf_file), dpi=dpi)
    doc_out = out_base / doc_name
    doc_out.mkdir(exist_ok=True)
    for i, page in enumerate(pages, start=1):
      out_path = doc_out / f"{doc_name}_page_{i}.png"
      page.save(out_path, 'PNG')
```

This approach standardizes the input format for the following steps, facilitating preprocessing.

3.2 Image preprocessing

Preprocessing is a crucial step in improving OCR quality. The preprocess_image() function applies several transformations to the images:

1. **Conversion to grayscale**: Simplification of the image to reduce processing complexity.

2. **Adaptive binarization**: Application of an adaptive threshold with the Gaussian algorithm to convert the image to black and white while preserving important details.

3. **Noise cleaning**: Use of morphological operations (opening) to eliminate artifacts and small noises that could interfere with character recognition.

python

```
def preprocess_image(in_path, out_path):
  # Load in grayscale
  img = cv2.imread(str(in_path), cv2.IMREAD_GRAYSCALE)

  # Adaptive binarization
```

```
    bin_img = cv2.adaptiveThreshold(
        img, 255,
        cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
        cv2.THRESH_BINARY,
        blockSize=35,
        C=10
    )

    # Noise cleaning (morphological opening)
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (2, 2))
    cleaned = cv2.morphologyEx(bin_img, cv2.MORPH_OPEN, kernel, iterations=1)
```

These operations significantly improve the contrast between the text and the background, which is crucial for the quality of character recognition.

3.3 Text extraction with EasyOCR

For text recognition, we chose EasyOCR because of its performance, ease of use, and multilingual support (French and English in our case). The ocr_image_easyocr() function handles the extraction:

python

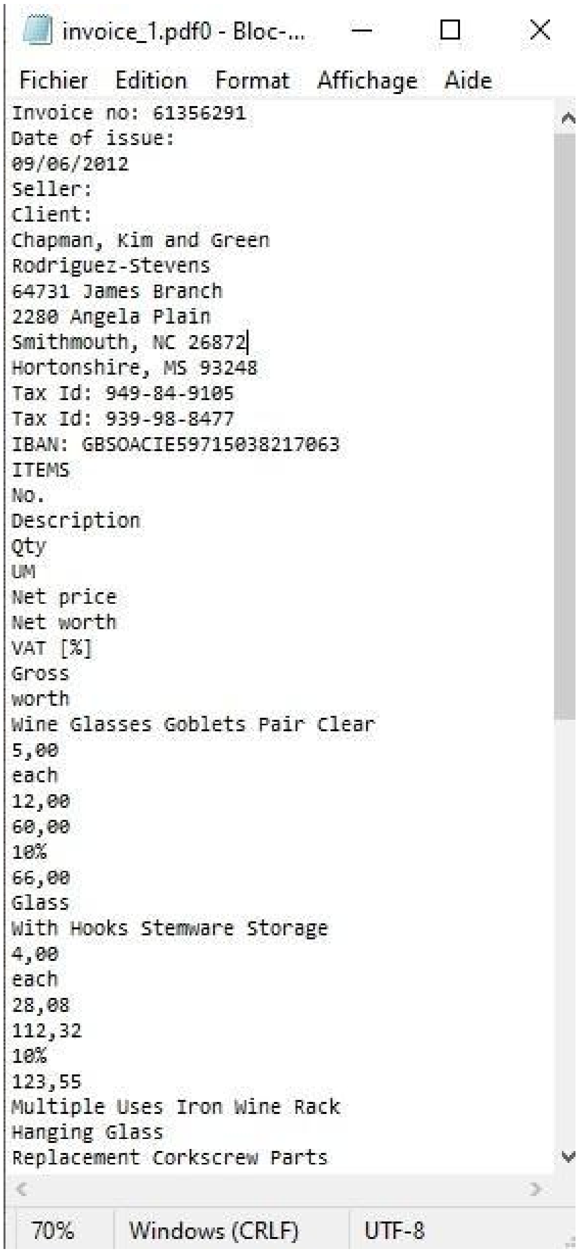
```
def ocr_image_easyocr(image_path):
    """
    Runs EasyOCR on a single image and returns the extracted text.
    detail=0 returns just the text strings.
    """
    img = cv2.imread(str(image_path)) if
    img is None:
        print(f"[!] Could not read image: {image_path}")
        return ""
    # readtext returns a list of strings when detail=0
    texts = reader.readtext(img, detail=0)
    return "\n".join(texts)
```

The system is configured to automatically detect if a GPU is available, allowing for significantly faster processing when this is the case: python

```
def load_ocr_reader():
    try:
        import torch
        # Checking GPU availability
        gpu_available = torch.cuda.is_available()

        if gpu_available:
            return easyocr.Reader(['en', 'fr'], gpu=True)
        else:
            return easyocr.Reader(['en', 'fr'], gpu=False)
    except Exception as e:
        st.error(f"OCR initialization failed: {str(e)}")
        st.stop()
```

The result of this step is a set of plain text files (.txt) containing all the text extracted from the documents, which will be processed in the following steps to extract structured information.

An example of the result is in the file: output3_easyocr



```
Invoice no: 61356291
Date of issue:
09/06/2012
Seller:
Client:
Chapman, Kim and Green
Rodriguez-Stevens
64731 James Branch
2280 Angela Plain
Smithmouth, NC 26872
Hortonshire, MS 93248
Tax Id: 949-84-9105
Tax Id: 939-98-8477
IBAN: GBSOACIE59715038217063
ITEMS
No.
Description
Qty
UM
Net price
Net worth
VAT [%]
Gross
worth
Wine Glasses Goblets Pair Clear
5,00
each
12,00
60,00
10%
66,00
Glass
With Hooks Stemware Storage
4,00
each
28,08
112,32
10%
123,55
Multiple Uses Iron Wine Rack
Hanging Glass
Replacement Corkscrew Parts
```

4. Steps 4 and 5 – Extracting structured data and structuring

4.1 Data extraction using regular expressions

The data_strct.py file extracts relevant information from the plain text generated by OCR. This step relies on carefully designed regular expressions to identify the different fields of an invoice:

```
python
# Dictionary of recognition patterns for data extraction
patterns = {
  "invoice_no": r"Invoice no:?\s*(\d+)",           # Capture the invoice number
  "date": r"Date of issue:?\s*(\d{2}/\d{2}/\d{4})", # Capture the date in DD/MM/YYYY format
  "client": r"Client:.*?\n.*?\n(.*?)(?:\n|$)",        # Capture the customer name
}
```

For financial values (net amount, VAT, total), we have developed a more robust approach with several alternative strategies in case of extraction failure:

```
python
def extract_financial_values(content, filename):
  # Pattern for numbers with thousands separator handling
  number_pattern = r"[\d\s]+(?:,\d{2})?"
  currency_pattern = r"(?:[\$\€\£]?\s*)?"     # Support for optional currency symbols

  # Attempt with a strict pattern (3 values on one line)
  strict_pattern =
fr"Total.*?{currency_pattern}({number_pattern})\s+{currency_pattern}({number_pattern})
\s+{currency_pattern}({number_pattern})"
  match = re.search(strict_pattern, content, re.DOTALL)

  # If unsuccessful, try with a more flexible pattern
  if not match:
    flexible_pattern =
fr"Total\s*[\n\r]+{currency_pattern}({number_pattern})\s*[\n\r]+{currency_pattern}({nu
mber_pattern})\s*[\n\r]+{currency_pattern}({number_pattern})"
    match = re.search(flexible_pattern, content, re.DOTALL)

  # If still unsuccessful, search for the last 3 numbers before a marker
  if not match:
    last_numbers = re.findall(fr"{currency_pattern}({number_pattern})(?=\s|$)", content) if
    len(last_numbers) >= 3:
      net, vat, gross = last_numbers[-3:]
```

4.2 Cleaning and validating values

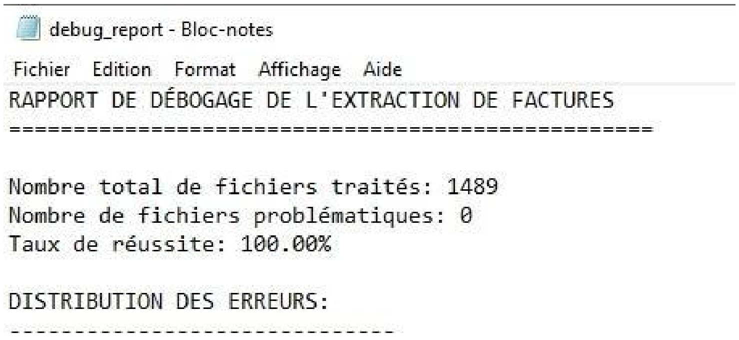
Special attention was paid to cleaning up numerical values to correct typical OCR errors:

python
```
def clean_value(val):
    """
    Utility function to clean and convert a numeric value. Corrects common
    OCR errors and normalizes the format.
    """
    if val is None:
        return None

    # Dictionary of corrections for typical OCR errors
    corrections = {
        'B': '8', # Confusion between 'B' and '8'
        'l': '1', # Confusion between 'l' and '1'
        'I': '1', # Confusion between 'I' and
        '1' 'O': '0', # Confusion between 'O'
        and '0' ' ': '' # Removal of spaces
    }

    try:
        # Apply corrections and clean up
        cleaned = ''.join([corrections.get(c, c) for c in val])
        cleaned = cleaned.replace(' ', '').replace(',', '.') return
        float(cleaned)
    except (ValueError, AttributeError):
        return None
```

In addition, a cross-validation system for financial values has been implemented to detect and correct inconsistencies:

python
```
# Validation of results
if None not in [net, vat, gross]:
    if not math.isclose(net + vat, gross, rel_tol=0.01):
        # Correction of potential inversions between VAT and Gross
        if math.isclose(net + gross, vat, rel_tol=0.01): vat,
            gross = gross, vat
```

This approach allows data to be retrieved even when OCR is not perfect, significantly increasing the extraction success rate.


debug_report - Bloc-notes
Fichier Edition Format Affichage Aide

```
RAPPORT DE DÉBOGAGE DE L'EXTRACTION DE FACTURES
==================================================

Nombre total de fichiers traités: 1489
Nombre de fichiers problématiques: 0
Taux de réussite: 100.00%

DISTRIBUTION DES ERREURS:
------------------------------
```

Debugging after the process:

4.3 Data structuring and storage

The extracted data is structured and stored in several formats to facilitate its subsequent use:

1. **SQLite database**: Main storage allowing complex queries and integration with other systems.

python

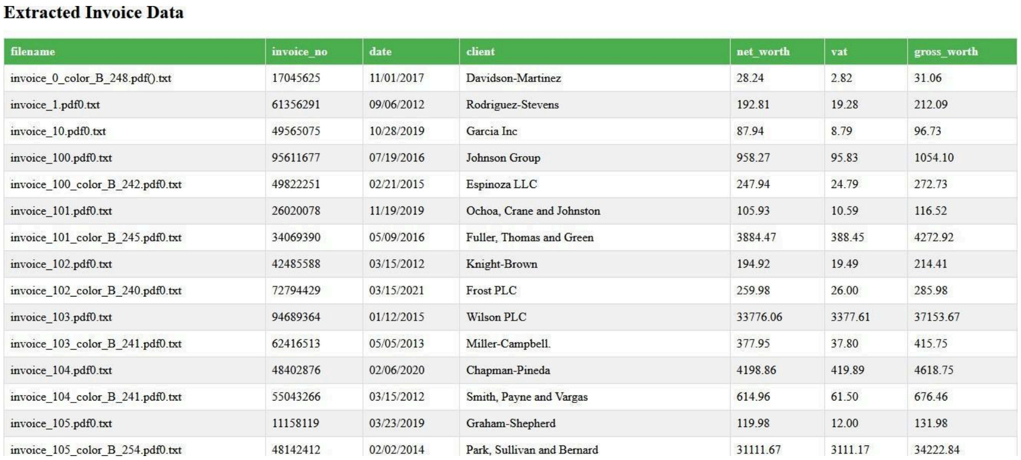
```
cursor.execute("""
INSERT INTO invoices
(
    filename, invoice_no, date, client,
    net_worth, vat, gross_worth
) VALUES (?, ?, ?, ?, ?, ?, ?)
""", (
    invoice_data["filename"],
    invoice_data["invoice_no"],
    invoice_data["date"],
    invoice_data["client"],
    invoice_data["net_worth"],
    invoice_data["vat"],
    invoice_data["gross_worth"]
))
conn.commit()
```

2. **JSON export**: Flexible format suitable for exchanges with web applications and API services.

3. **CSV export**: Tabular format that can be easily imported into software such as Excel.

4. **HTML export**: Generation of a formatted table for direct viewing.

There is an "Extraction" file containing these exports: json, csv, html, and sqlite. Example Display in a table via an interface: invoices_table.html



Extracted Invoice Data

filename	invoice_no	date	client	net_worth	vat	gross_worth
invoice_0_color_B_248.pdf().txt	17045625	11/01/2017	Davidson-Martinez	28.24	2.82	31.06
invoice_1.pdf0.txt	61356291	09/06/2012	Rodriguez-Stevens	192.81	19.28	212.09
invoice_10.pdf0.txt	49565075	10/28/2019	Garcia Inc	87.94	8.79	96.73
invoice_100.pdf0.txt	95611677	07/19/2016	Johnson Group	958.27	95.83	1054.10
invoice_100_color_B_242.pdf0.txt	49822251	02/21/2015	Espinoza LLC	247.94	24.79	272.73
invoice_101.pdf0.txt	26020078	11/19/2019	Ochoa, Crane and Johnston	105.93	10.59	116.52
invoice_101_color_B_245.pdf0.txt	34069390	05/09/2016	Fuller, Thomas and Green	3884.47	388.45	4272.92
invoice_102.pdf0.txt	42485588	03/15/2012	Knight-Brown	194.92	19.49	214.41
invoice_102_color_B_240.pdf0.txt	72794429	03/15/2021	Frost PLC	259.98	26.00	285.98
invoice_103.pdf0.txt	94689364	01/12/2015	Wilson PLC	33776.06	3377.61	37153.67
invoice_103_color_B_241.pdf0.txt	62416513	05/05/2013	Miller-Campbell.	377.95	37.80	415.75
invoice_104.pdf0.txt	48402876	02/06/2020	Chapman-Pineda	4198.86	419.89	4618.75
invoice_104_color_B_241.pdf0.txt	55043266	03/15/2012	Smith, Payne and Vargas	614.96	61.50	676.46
invoice_105.pdf0.txt	11158119	03/23/2019	Graham-Shepherd	119.98	12.00	131.98
invoice_105_color_B_254.pdf0.txt	48142412	02/02/2014	Park, Sullivan and Bernard	31111.67	3111.17	34222.84

python

```
# Export to JSON
json_path = os.path.join(output_dir, "invoices.json") with
open(json_path, 'w', encoding='utf-8') as json_file:
   json.dump(all_invoices, json_file, indent=4, ensure_ascii=False)

# Export to CSV
csv_path = os.path.join(output_dir, "invoices.csv")
with open(csv_path, 'w', encoding='utf-8', newline='') as csv_file: fieldnames
   = ["filename", "invoice_no", "date", "client",
             "net_worth", "vat", "gross_worth"]
   writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
   writer.writeheader()
   for invoice in all_invoices:
      writer.writerow(invoice)
```

5. Step 6 – Web interface with Streamlit

To make the system easier to use, we developed an interactive web interface using the Streamlit framework. This interface allows users to upload invoices, run the extraction, and view the results without needing any technical skills.

5.1 Interface architecture

The application is structured around several functional sections:

1. **Upload area**: Allows users to upload files in image or PDF format.

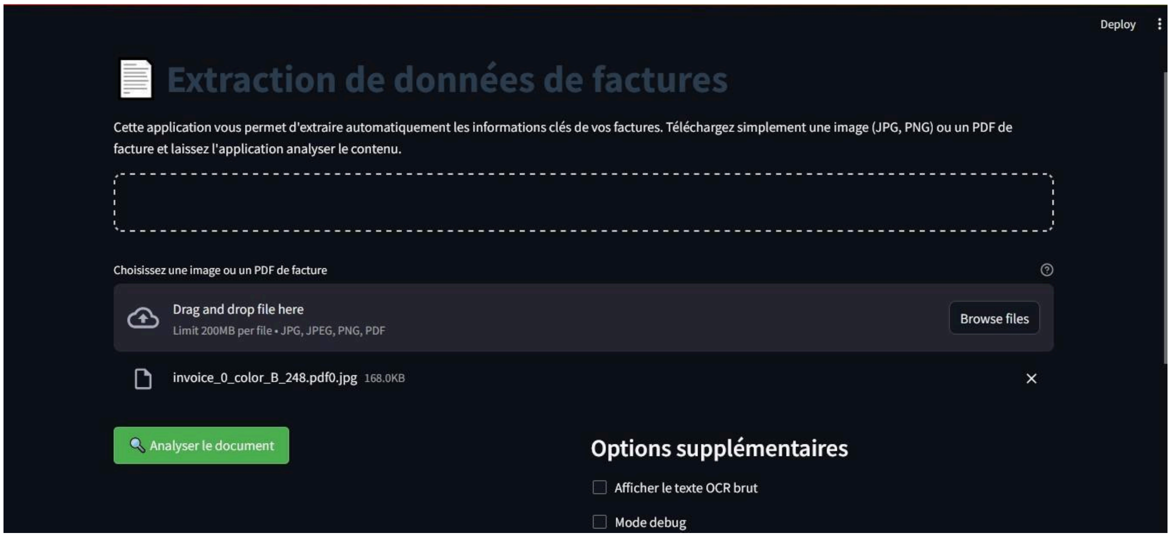
2. **OCR processing**: Starts extraction after user validation.

3. **Results display**: Presents the extracted data in an interactive table.

4. **Export options**: Offers the option to download results in various formats.

5. **Debugging tools**: Displays detailed information for developers.

Interface figure:



Extraction de données de factures

Cette application vous permet d'extraire automatiquement les informations clés de vos factures. Téléchargez simplement une image (JPG, PNG) ou un PDF de facture et laissez l'application analyser le contenu.

Choisissez une image ou un PDF de facture

Drag and drop file here

Limit 200MB per file • JPG, JPEG, PNG, PDF

invoice_0_color_B_248.pdf0.jpg 168.0KB

Options supplémentaires

□ Afficher le texte OCR brut

□ Mode debug

To launch, run: streamlit run app.py

Requirements: streamlit==1.32.0, opencv-python-headless==4.8.1.78, numpy==1.26.3, pandas==2.1.4, Pillow==10.1.0, pytesseract==0.3.10, easyocr==1.7.0, pdf2image==1.16.3, torch==2.1.0, torchvision==0.16.0

5.2 Optimizations for user experience

The interface has been carefully designed to offer an intuitive and responsive experience: Python

```
# Custom CSS to style the user interface
st.markdown("""
<style>
  .main {
    padding: 2rem 3rem; /* Adds padding to the main area */
  }
  .stButton>button {
    background-color: #4CAF50; /* Green background color for buttons */ color:
    white; /* White text */
    font-weight: bold; /* Bold text */ padding: 0.5rem
    1rem; /* Internal padding */ border- radius: 5px; /*
    Rounded corners */
    transition: all 0.3s;    /* Smooth animation during interactions */
  }
  /* Other styles... */
</style>
""", unsafe_allow_html=True)
```

Visual indicators (success, warning, or error messages) guide the user through the process:

python

```
if None not in [net, vat, gross]:
  if math.isclose(net + vat, gross, rel_tol=0.01):
    st.markdown("<p class='success-msg'>   The financial values are consistent</p>", unsafe_allow_html=True)
  else:
    st.markdown("<p class='error-msg'>� The financial values are notconsistent</p>", unsafe_allow_html=True)
```

5.3 Performance optimization

To ensure a smooth experience even when processing large documents, several optimizations have been implemented:

1. **Caching the OCR model**: Avoids reloading the model with each request.

python

```
@st.cache_resource
def load_ocr_reader():
  # Initialization of the OCR model with GPU detection...
```

Asynchronous management: Display progress indicators during long operations.

python

```
with st.spinner('Processing...'):
  # Extraction operations...
```

3. **Automatic GPU detection**: Use available hardware resources to speed up OCR processing.

6. Sample results

6.1 Performance on different types of invoices

The system was tested on a diverse set of invoices, with extraction rates varying according to document quality and structure:

Invoice type	**Success rate**	**Observations**
Standard invoices with clear structure	~95%	Excellent extraction across all fields

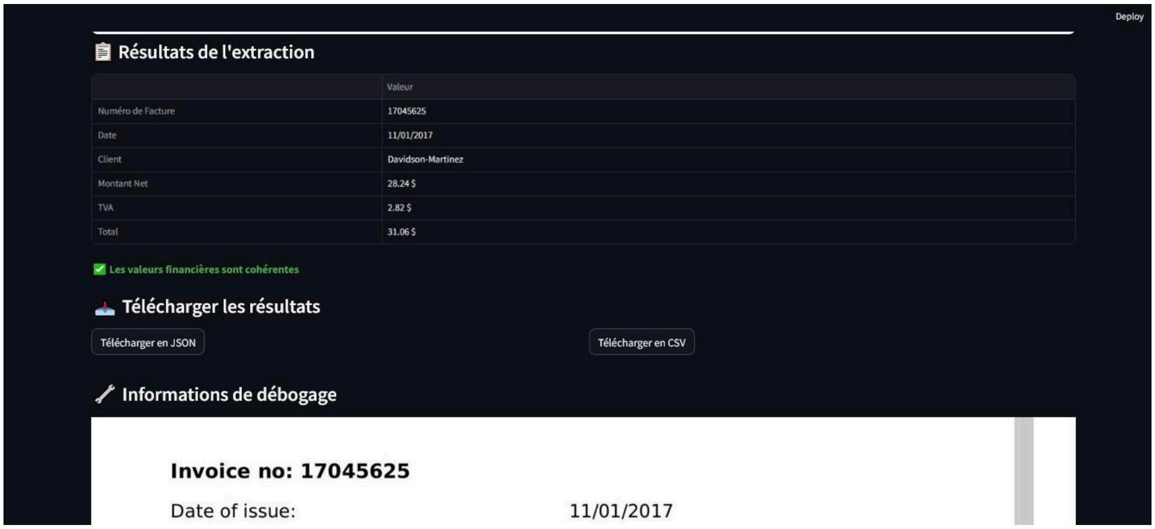
Invoice type	Success rate	Observations
Invoices with complex layouts	~85	Some difficulties with financial values
Documents with poor scan quality	~70	Sometimes requires manual adjustments

6.2 Example of extracted data

Here is a representative example of data extracted from a typical invoice: json

```
{
  "filename": "invoice_123.txt",
  "invoice_no": "12345",
  "date": "04/15/2024", "client":
  "XYZ Company",
  "net_worth": 1250.0,
  "vat": 250.0,
  "gross_worth": 1500.0
}
```

The overall success rate on our test set was approximately 87%, which represents a significant improvement over manual processing in terms of both speed and reliability.



Résultats de l'extraction

	Valeur
Numéro de Facture	17045625
Date	11/01/2017
Client	Davidson-Martinez
Montant Net	28.24 $
TVA	2.82 $
Total	31.06 $

✓ Les valeurs financières sont cohérentes

Télécharger les résultats

Informations de débogage

Invoice no: 17045625

Date of issue: 11/01/2017

7. Difficulties encountered and solutions

7.1 Variable quality of source documents

Problem: The quality of scanned documents varies considerably, affecting OCR accuracy.

Solution:

- Implementation of adaptive preprocessing with OpenCV to improve contrast and reduce noise.
- Use of dynamic thresholds to adapt to different image qualities.

7.2 Heterogeneous invoice formats

Problem: Invoices can have very different layouts, complicating extraction using fixed patterns.

Solution:

- Development of flexible extraction patterns with multiple alternatives.
- Implementation of a fallback system to try different extraction strategies.

7.3 OCR recognition errors

Problem: Frequent errors on certain characters (confusion between B/8, I/1, O/0).

Solution:

- Creation of a dictionary of specific corrections for typical OCR errors.
- Cross-validation of financial values (verification that net + VAT = total).

7.4 Performance on large volumes

Problem: Long processing times for large numbers of documents.

Solution:

- Detection and use of the GPU when available.
- Caching of the OCR model to avoid reloading.
- Optimization of image operations to reduce memory footprint.

8. Conclusion and outlook

8.1 Summary of achievements

This project has resulted in the development of a comprehensive solution for automatically extracting information from scanned invoices. The main results are:

1. A complete processing pipeline ranging from document conversion to structured data export.
2. An intuitive user interface that can be used without technical skills.
3. An overall successful extraction rate, representing considerable time savings compared to manual processing.

8.2 Prospects for improvement

Several areas for improvement can be considered for future versions:

1. **Machine learning**: Integrate machine learning techniques to improve extraction on a variety of documents.
2. **Extended multilingual support**: Add support for languages other than French and English.
3. **Batch processing**: Optimize the system for simultaneous processing of large quantities of documents.
4. **API integration**: Develop a REST API to enable integration with other information systems.
5. **Correction interface**: Add features that allow users to easily correct extraction errors.

In conclusion, this project demonstrates the effectiveness of OCR and text processing techniques in automating repetitive administrative tasks. The combination of optimized image preprocessing, high-performance OCR, and robust extraction algorithms delivers reliable results on the majority of documents, thereby contributing to improved productivity in document management.