Le Défi Initial

Le projet commence avec un objectif : numériser et extraire le contenu d’une centaine documents d’archives militaires d’Indochine, datant des années 1950. Ces documents dactylographiés présentent une structure particulière en deux colonnes : dates à gauche, événements à droite.

Le défi ? Ces documents vieillissants ont plus de 70 ans. Le papier a jauni, l’encre s’est estompée, et la qualité de la frappe varie. Impossible de simplement « scanner et extraire » – il faut un véritable pipeline de traitement.

Phase 1 : Les Premiers Pas avec Google Cloud Vision

Le choix se porte sur Google Cloud Vision API, connu pour sa précision sur les documents complexes. Le premier script `vision.py` naît avec deux optimisations cruciales :

python

```
response = client.document_text_detection(
    image=image,
    image_context={"language_hints": ["fr"]}
)
```

- `document_text_detection` : optimisé pour les documents structurés
- `language_hints: ["fr"]` : essentiel pour les accents français (é, è, ç, etc.)

Phase 2 : Le Casse-Tête du Colonage

Le premier obstacle sérieux apparaît : comment apparier intelligemment les dates et leur contenu correspondant ?

Les documents ont une structure claire visuellement, mais l’OCR retourne un flux désordonné de paragraphes. La solution ? Un algorithme d’appariement par proximité verticale :

python

```
def smart_pair_dates_content(dates, content_items):
    # Pour chaque date, trouver le contenu le plus proche en Y
    # Préserver l'ordre topologique du document
    # Fusionner les paragraphes orphelins
```

Ce système :

- Trie tous les éléments par position verticale (Y)
- Apparie chaque date avec le contenu le plus proche
- Fusionne les paragraphes sans date avec le paragraphe précédent
- Préserve l’ordre chronologique du document original

Phase 3 : La Reconnaissance des Dates

Un nouveau problème surgit : les dates prennent plusieurs formats selon les documents :

- `1° Novembre` (format original attendu)
- `14.11.1955` (format numérique)
- `Novembre 1955`
- `1955` (année seule)

La fonction `is_date()` évolue pour détecter tous ces formats via regex :

python

```
patterns = [
    r'\d{1,2}\s*°?\s*(?:Janvier|Février|...)',  # Texte
    r'\d{1,2}[\./-]\d{1,2}[\./-]\d{4}',         # 14.11.1955
    r'^\d{4}$',                                   # 1955
]
```

Phase 4 : Le Prétraitement ImageMagick

Pour les documents particulièrement dégradés, l’OCR seul ne suffit pas. Entre en jeu ImageMagick avec 6 prétraitements différents :

1. Baseline : contraste + netteté standard
2. Aggressive : contraste fort pour papier jauni
3. Denoise : réduction du bruit pour vieilles frappes
4. Enhance : netteté maximale
5. Threshold : binarisation pour documents très clairs/foncés
6. Optimal : combinaison équilibrée

Le script `batch_imagemagick.py` permet de tester visuellement quelle version

donne les meilleurs résultats OCR. On utilise majoritairement l’option *Denoise*

Phase 5 : Le Problème de l’Orientation

Surprise : certains documents ont été scannés dans le mauvais sens ! Ils ont été redressés manuellement. les métadonnées EXIF indiquent la rotation, mais Cloud Vision les ignore.

Solution : `check_orientation.py` qui :

- Lit les métadonnées EXIF
- Effectue une rotation physique de l’image
- Peut générer 4 versions (0°, 90°, 180°, 270°) pour tester

Phase 6 : L’Enfer de l’Encodage Windows

Le vrai cauchemar technique : les problèmes d’encodage. Sur Windows, les scripts Python utilisent par défaut cp1252, mais :

- Les emojis dans les messages (□ □ □) sont en UTF-8
- Les accents français nécessitent UTF-8
- Les appels `subprocess` mélangent les encodages

La bataille fait rage ligne par ligne :

- Suppression de tous les emojis : □ → `[OK]`
- Caractères spéciaux : ° → `degres`, Δ → `d`
- Force UTF-8 dans subprocess : `encoding='utf-8', errors='replace'`
- Force UTF-8 dans les scripts : `sys.stdout = io.TextIOWrapper(...)`

Phase 7 : La Grande Simplification – Adieu Node.js

Un tournant majeur : la génération des fichiers Word utilisait Node.js + docx via subprocess. Une dépendance de trop.

Refactorisation complète. 115 lignes de code JavaScript éliminées, remplacées par du Python pur :

python

```
from docx import Document
doc = Document()
table = doc.add_table(rows=len(data), cols=2)
# Supprimer les bordures, formater, sauvegarder
doc.save(output_file)
```

Phase 8 : L’Automatisation par Lot

Avec 118 documents, le traitement manuel est impensable. Naissance du pipeline batch :

Scripts développés :

1. batch_orientation.py : Corrige l’orientation de toutes les images
2. batch_imagemagick.py : Applique le prétraitement optimal à tous les documents
3. batch_ocr.bat : Lance l’OCR sur tout un dossier
4. remove_oriented.py : Nettoie les noms de fichiers

5. merge_word_simple.py : Fusionne tous les Word en un seul document

Le workflow final :

bash

```
# 1. Prétraitement (si nécessaire)
python batch_imagemagick.py Archives/ --tests 6 --output
Archives_clean/

# 2. Correction orientation
python batch_orientation.py Archives_clean/ --fix

# 3. Nettoyage noms
python remove_oriented.py Archives_clean/

# 4. OCR massif
batch_ocr.bat Archives_clean/ Resultats/

# 5. Fusion finale
python merge_word_simple.py Resultats/ Legion_Indochine_1950s.docx
```

Un seul document Word final : 118 documents d’archives structurés, searchable, exploitables.

Les Défis Techniques Surmontés

1. Détection de Structure

Transformer un flux OCR désordonné en paires date-contenu cohérentes via analyse géométrique.

2. Polyvalence des Formats

Gérer 7+ formats de dates différents dans les mêmes documents.

3. Qualité Variable

Des documents de 70 ans, jaunis, estompés → prétraitement ImageMagick adaptatif.

4. Encodage Cross-Platform

Le grand boss final : faire cohabiter UTF-8, cp1252, emojis, accents français, subprocess Python et batch Windows.

5. Simplicité d’Usage

118 documents × 5 étapes = trop complexe. Solution : scripts batch qui enchaînent tout automatiquement.

Les Leçons Apprises

Technique

1. Less is More : Éliminer Node.js a simplifié tout le projet
2. UTF-8 Everywhere : Sur Windows, forcer UTF-8 partout évite 90% des

problèmes
3. Tester d’Abord : Les 6 variantes ImageMagick permettent de choisir visuellement avant de tout traiter
4. Chemins Absolus : Dans les scripts batch, toujours utiliser `__file__` pour les chemins

Méthodologie

1. Mise en place de la team:
    - Patrick pour les spécifications et les tests
    - Claude *Anthropic* pour la réalisation
2. Itération : Le projet a évolué sur plusieurs jours, chaque problème amenant sa solution
3. Feedback Utilisateur : « Ça ne marche pas » → Debug → Correction → « Maintenant ça marche ! »
4. Pragmatisme : Abandonner les solutions complexes (NODE_PATH, npx) pour des solutions simples (python-docx direct)

Le Résultat Final

De : 118 images JPEG de documents jaunis, mal orientés, difficilement lisibles

À : Un document Word de 300+ pages, structuré, searchable, avec :

- Toutes les dates correctement identifiées
- Le contenu associé à chaque date
- Les paragraphes fusionnés intelligemment
- Un format exploitable pour la recherche historique

Technologies utilisées :

- Python 3.12
- Google Cloud Vision API
- python-docx
- ImageMagick
- PIL (Pillow)
- Regex pour la détection de dates

Lignes de code : ~2000 lignes Python + batch scripts

Temps de traitement : ~15 minutes pour 118 documents (avec prétraitement)

Pour Aller Plus Loin

Ce projet démontre que la numérisation d’archives historiques est accessible avec des outils modernes, même pour des documents complexes. Les défis techniques (encodage, structure, qualité) sont surmontables avec patience et itération.

Le code complet est modulaire et réutilisable pour d’autres projets d’archives : registres d’état civil, journaux de bord militaires, correspondances historiques, etc.

Author

- 

  Patrick

GPM Factory