Documentation Complète : Création et Utilisation d'un Modèle YOLO pour la Conversion de PDF en XML Structuré

Table des matières

Introduction

Ce document décrit en détail le processus complet pour développer et utiliser un système de conversion de documents PDF vers des fichiers XML structurés en utilisant la détection d'objets YOLO et des techniques d'OCR. Le système permet d'identifier automatiquement des éléments métiers spécifiques dans des documents et de les convertir en une structure XML conforme à un schéma prédéfini.

Cette approche hybride combine la vision par ordinateur (détection d'objets) et le traitement du langage naturel (OCR) pour extraire à la fois la structure et le contenu des documents PDF, ce qui est particulièrement utile pour l'automatisation du traitement documentaire dans des contextes professionnels.

Vue d'ensemble du processus

Le processus complet se décompose en trois grandes phases :

1. **Création du dataset** : Préparation des données d'entraînement pour le modèle YOLO
2. **Entraînement du modèle** : Configuration et exécution de l'entraînement YOLO
3. **Conversion de PDF en XML** : Utilisation du modèle entraîné pour traiter de nouveaux documents

Voici une représentation schématique du processus global :

Copier

```
Création du Dataset → Entraînement du Modèle → Conversion PDF→XML
```

Phase 1 : Création du Dataset

Préparation des images

La première étape consiste à transformer les documents PDF en images qui serviront à l'annotation et à l'entraînement du modèle YOLO.

Script de conversion PDF vers PNG

Utilisez le script `pdf-to-png-converter.py` qui exploite la bibliothèque `pdf2image` pour convertir les PDF en images PNG.

Étapes de la conversion :

1. Placez vos fichiers PDF dans un dossier `input`
2. Exécutez le script qui va :
   - Analyser tous les fichiers PDF pour déterminer le nombre maximum de pages
   - Créer une structure de dossiers organisée par numéro de page (`page_001`, `page_002`, etc.)
   - Convertir chaque page de chaque PDF en image PNG avec une résolution de 300 DPI
   - Sauvegarder les images dans les dossiers correspondants

Exemple d'utilisation :

bash Copier

```
python pdf-to-png-converter.py
```

Structure des dossiers résultante :

Copier

```
output/
  ├── page_001/
  │   ├── document1.png
  │   ├── document2.png
  │   └── ...
  ├── page_002/
  │   ├── document1.png
  │   ├── document2.png
  │   └── ...
  └── ...
```

Normalisation des labels

Avant de commencer l'annotation, établissez une liste normalisée des labels qui serviront à identifier les différents éléments métiers dans vos documents.

Exemple de liste de labels :

- titre_document
- tableau_données
- bloc_signature
- champ_montant
- date_document
- entête_société
- pied_page
- etc.

Assurez-vous que :

- Les labels sont cohérents et sans ambiguïté
- Ils correspondent aux éléments que vous souhaitez extraire dans le XML final
- Tous les annotateurs utilisent exactement les mêmes labels (si plusieurs personnes travaillent sur le projet)

Annotation des images

L'annotation consiste à identifier et délimiter manuellement les éléments d'intérêt dans chaque image.

Utilisation de labelImg :

1. Téléchargez et installez labelImg
2. Configurez l'outil pour sauvegarder au format PASCAL VOC (format XML)

3. Chargez les images d'un dossier de page spécifique
4. Pour chaque image :
    - Dessinez des rectangles autour des éléments d'intérêt
    - Attribuez le label approprié à chaque rectangle
    - Sauvegardez le fichier XML d'annotation (même nom que l'image mais avec extension .xml)

Conseils pour une annotation efficace :

- Sélectionnez un échantillon représentatif de dossiers de pages (ex: pages 1, 2, 7, 8, 9, 10, 21)
- Annotez au moins 50-100 exemples de chaque type d'élément pour un bon apprentissage
- Assurez-vous que les rectangles encadrent précisément les éléments
- Variez les exemples (taille, position, apparence) pour une meilleure généralisation

Consolidation du dataset

Une fois l'annotation terminée, consolidez les paires d'images et de fichiers XML dans un seul dossier pour l'entraînement.

Script de déplacement des paires

Utilisez le script `move_pairs.py` pour :

1. Identifier les paires PNG/XML dans les dossiers de pages sélectionnés
2. Ajouter un préfixe de numéro de page à chaque fichier
3. Déplacer les paires complètes vers un dossier dataset unique
4. Vérifier l'intégrité des paires déplacées

Exemple d'utilisation :

python Copier

```python
# Configuration dans le script move_pairs.py
dossier_sync = ['001', '002', '007', '008', '010', '009', '021']

for d in dossier_sync :
    SOURCE_DIR = f"C:/Users/H14376/Desktop/yoloDoc/output/page_{d}"
    DESTINATION_DIR = r"C:\Users\H14376\Desktop\yoloDoc\dataset"
    move_paired_files(SOURCE_DIR, DESTINATION_DIR, d)
    verify_moved_files(DESTINATION_DIR)
```

Structure du dataset final :

Copier

```
dataset/
  ├── 001_document1.png
  ├── 001_document1.xml
  ├── 001_document2.png
  ├── 001_document2.xml
  ├── 002_document1.png
  ├── 002_document1.xml
  └── ...
```

Phase 2 : Entraînement du modèle YOLO

Configuration de l'environnement

Préparez l'environnement pour l'entraînement du modèle YOLO v11 :

1. Installez les dépendances nécessaires :

bash Copier

```
pip install ultralytics
```

2. Organisez votre dataset selon la structure requise par YOLO :

Copier

```
dataset/
  ├── images/train/   # Images PNG pour l'entraînement
  ├── images/val/     # Images PNG pour la validation
  ├── labels/train/   # Fichiers de labels au format YOLO pour l'entraînement
  └── labels/val/     # Fichiers de labels au format YOLO pour la validation
```

3. Convertissez les annotations XML (format PASCAL VOC) en format YOLO :

python Copier

```python
# Exemple de script de conversion (à adapter selon vos besoins)
import xml.etree.ElementTree as ET
import os
from pathlib import Path

def convert_voc_to_yolo(xml_file, class_list):
    tree = ET.parse(xml_file)
    root = tree.getroot()

    # Dimensions de l'image
    size = root.find('size')
    width = int(size.find('width').text)
    height = int(size.find('height').text)

    yolo_lines = []

    # Pour chaque objet dans l'annotation
    for obj in root.findall('object'):
        class_name = obj.find('name').text
        class_id = class_list.index(class_name)

        # Coordonnées du rectangle
        bbox = obj.find('bndbox')
        xmin = float(bbox.find('xmin').text)
        ymin = float(bbox.find('ymin').text)
        xmax = float(bbox.find('xmax').text)
        ymax = float(bbox.find('ymax').text)

        # Conversion au format YOLO (normalisé)
        x_center = (xmin + xmax) / 2 / width
        y_center = (ymin + ymax) / 2 / height
        w = (xmax - xmin) / width
        h = (ymax - ymin) / height

        yolo_lines.append(f"{class_id} {x_center} {y_center} {w} {h}")

    return yolo_lines
```

Paramétrage de l'entraînement

Créez un fichier de configuration YAML pour votre dataset :

yaml Copier

```yaml
# dataset.yaml
path: ./dataset  # chemin vers le dossier dataset
train: images/train  # chemin relatif vers les images d'entraînement
val: images/val  # chemin relatif vers les images de validation

# Classes (à adapter selon vos labels normalisés)
names:
  0: titre_document
  1: tableau_données
  2: bloc_signature
  3: champ_montant
  4: date_document
  5: entête_société
  6: pied_page
```

Exécution de l'entraînement

Lancez l'entraînement du modèle YOLO avec les paramètres adaptés à votre cas d'usage :

bash Copier

```bash
yolo train model=yolov11.pt data=dataset.yaml epochs=100 imgsz=640 batch=8
```

Paramètres importants :

- `model` : Modèle de base à utiliser (yolov11.pt pour YOLO v11)
- `data` : Chemin vers le fichier de configuration du dataset
- `epochs` : Nombre d'itérations d'entraînement
- `imgsz` : Taille des images d'entrée
- `batch` : Nombre d'images traitées simultanément

Évaluation et ajustement du modèle

Après l'entraînement, évaluez les performances du modèle :

bash Copier

```bash
yolo val model=runs/train/exp/weights/best.pt data=dataset.yaml
```

Analysez les métriques (mAP, précision, rappel) et ajustez si nécessaire :

- Si la précision est faible : ajoutez plus d'exemples variés

- Si certaines classes sont mal détectées : augmentez le nombre d'exemples pour ces classes
- Si le modèle semble sur-apprendre : augmentez la régularisation ou réduisez la complexité

Phase 3 : Conversion de PDF en XML structuré

Prétraitement des documents PDF

À l'identique de la phase de création du dataset, commencez par convertir les PDF en images :

python Copier

```python
# Utilisation de pdf-to-png-converter.py
convert_pdf_to_png("dossier_pdf_entree", "dossier_images_sortie")
```

Détection des éléments avec YOLO

Utilisez le modèle entraîné pour détecter les éléments dans chaque page :

python Copier

```python
from ultralytics import YOLO

# Charger le modèle entraîné
model = YOLO("chemin/vers/best.pt")

# Détecter les éléments dans chaque image
def detect_elements(image_path):
    results = model(image_path)
    return results[0].boxes  # Récupère les boîtes de détection

# Traiter toutes les images
import os
from pathlib import Path

def process_all_images(images_dir):
    xml_results = {}

    for page_dir in Path(images_dir).glob("page_*"):
        page_num = page_dir.name.split("_")[1]

        for img_file in page_dir.glob("*.png"):
            # Détecter les éléments
            boxes = detect_elements(img_file)

            # Stocker les résultats dans un format intermédiaire
            doc_name = img_file.stem
            if doc_name not in xml_results:
                xml_results[doc_name] = {}

            xml_results[doc_name][page_num] = []

            for box in boxes:
                class_id = int(box.cls)
                class_name = model.names[class_id]
                coords = box.xyxy[0].tolist()  # [x1, y1, x2, y2]
                confidence = float(box.conf)

                xml_results[doc_name][page_num].append({
                    "type": class_name,
                    "coords": coords,
                    "confidence": confidence
                })

            # Sauvegarder en XML intermédiaire
```

```
            save_interim_xml(doc_name, page_num, xml_results[doc_name][page_num],
                             output_dir="interim_xml")

    return xml_results
```

Extraction et OCR du contenu

Pour chaque élément détecté, extrayez la zone correspondante et appliquez l'OCR :

python Copier

```python
import cv2
from PIL import Image
import io
import requests

# Fonction pour extraire une zone d'image
def extract_region(image_path, coords):
    img = cv2.imread(str(image_path))
    x1, y1, x2, y2 = [int(c) for c in coords]
    region = img[y1:y2, x1:x2]
    return region

# Fonction pour réaliser l'OCR avec un VLM
def ocr_with_vlm(image_region):
    # Convertir la région en format requis par le VLM
    success, encoded_image = cv2.imencode('.png', image_region)
    image_bytes = encoded_image.tobytes()

    # Appel au service VLM (exemple avec une API)
    response = requests.post(
        "URL_DU_SERVICE_VLM",
        files={"image": ("image.png", io.BytesIO(image_bytes), "image/png")},
        json={"task": "ocr"}
    )

    return response.json()["text"]

# Appliquer l'OCR à tous les éléments détectés
def process_ocr(xml_results, images_dir):
    for doc_name, pages in xml_results.items():
        for page_num, elements in pages.items():
            page_dir = Path(images_dir) / f"page_{page_num}"
            img_path = page_dir / f"{doc_name}.png"

            for i, element in enumerate(elements):
                region = extract_region(img_path, element["coords"])
                text = ocr_with_vlm(region)

                # Ajouter le texte OCR au résultat
                xml_results[doc_name][page_num][i]["content"] = text

    return xml_results
```

Génération et validation du XML final

Générez le XML structuré final à partir des informations collectées et validez-le selon le schéma XSD :

python Copier

```python
import xml.etree.ElementTree as ET
import xmlschema

# Fonction pour générer le XML structuré
def generate_structured_xml(xml_results, doc_name):
    # Créer l'élément racine
    root = ET.Element("document")
    root.set("name", doc_name)

    # Créer un élément pour chaque page
    for page_num in sorted(xml_results[doc_name].keys()):
        page_elem = ET.SubElement(root, "page")
        page_elem.set("number", page_num)

        # Ajouter chaque élément détecté
        for element in xml_results[doc_name][page_num]:
            elem_type = element["type"]
            content = element["content"]
            coords = element["coords"]

            elem = ET.SubElement(page_elem, elem_type)
            elem.text = content

            # Ajouter les coordonnées comme attributs
            elem.set("x1", str(coords[0]))
            elem.set("y1", str(coords[1]))
            elem.set("x2", str(coords[2]))
            elem.set("y2", str(coords[3]))

    # Créer l'arbre XML
    tree = ET.ElementTree(root)
    return tree

# Fonction pour valider le XML avec un schéma XSD
def validate_xml(xml_tree, xsd_path):
    schema = xmlschema.XMLSchema(xsd_path)
    is_valid = schema.is_valid(ET.tostring(xml_tree.getroot()))

    if not is_valid:
        # Tentative de correction automatique si possible
        validation_errors = schema.validate(ET.tostring(xml_tree.getroot()))
        print(f"Erreurs de validation XML : {validation_errors}")

    return is_valid
```

```python
# Traiter tous les documents
def process_all_documents(xml_results, output_dir, xsd_path):
    for doc_name in xml_results.keys():
        # Générer le XML structuré
        xml_tree = generate_structured_xml(xml_results, doc_name)

        # Valider avec le schéma XSD
        is_valid = validate_xml(xml_tree, xsd_path)

        # Sauvegarder le XML final
        output_path = Path(output_dir) / f"{doc_name}.xml"
        xml_tree.write(output_path, encoding="utf-8", xml_declaration=True)

        print(f"Document {doc_name} traité : XML {'valide' if is_valid else 'non valide'}")
```

Outils et dépendances

Logiciels requis

- Python 3.8 ou supérieur
- labelImg (pour l'annotation)
- YOLO v11 (Ultralytics)

Bibliothèques Python

- pdf2image (conversion PDF en PNG)
- ultralytics (implémentation YOLO)
- opencv-python (traitement d'image)
- requests (appels API VLM)
- xmlschema (validation XML)

Services externes

- Service VLM pour l'OCR (Vision Language Model)

Dépannage et bonnes pratiques

Problèmes courants

- **Images de mauvaise qualité** : Augmentez la résolution DPI lors de la conversion des PDF
- **Détections incorrectes** : Enrichissez le dataset avec plus d'exemples variés
- **OCR peu précis** : Améliorez la qualité des extractions ou testez un autre service VLM
- **XML non valide** : Vérifiez la conformité avec le schéma XSD et ajustez la génération

Bonnes pratiques

- Sauvegardez des points de contrôle réguliers pendant l'entraînement
- Testez le modèle sur un petit échantillon avant de traiter un lot complet
- Gardez une trace des métriques d'évaluation entre les itérations du modèle
- Documentez les classes et leurs caractéristiques visuelles
- Versionnez votre schéma XSD et assurez sa compatibilité avec les différentes versions du modèle

Annexes

Exemple de schéma XSD

xml Copier

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="document">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="page" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="titre_document" minOccurs="0" maxOccurs="1">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute name="x1" type="xs:float" use="required"/>
                      <xs:attribute name="y1" type="xs:float" use="required"/>
                      <xs:attribute name="x2" type="xs:float" use="required"/>
                      <xs:attribute name="y2" type="xs:float" use="required"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
              <!-- Définir d'autres éléments selon vos besoins -->
            </xs:sequence>
            <xs:attribute name="number" type="xs:integer" use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Exemples de commandes

bash Copier

```bash
# Conversion PDF en PNG
python pdf-to-png-converter.py

# Entraînement YOLO
yolo train model=yolov11.pt data=dataset.yaml epochs=100 imgsz=640 batch=8

# Validation du modèle
yolo val model=runs/train/exp/weights/best.pt data=dataset.yaml

# Prédiction sur une seule image
yolo predict model=runs/train/exp/weights/best.pt source=path/to/image.png
```

Ce document est conçu pour servir de guide complet au processus de création et d'utilisation du modèle. Il doit être adapté selon les besoins spécifiques du projet et mis à jour en fonction des évolutions des outils et des techniques utilisés.