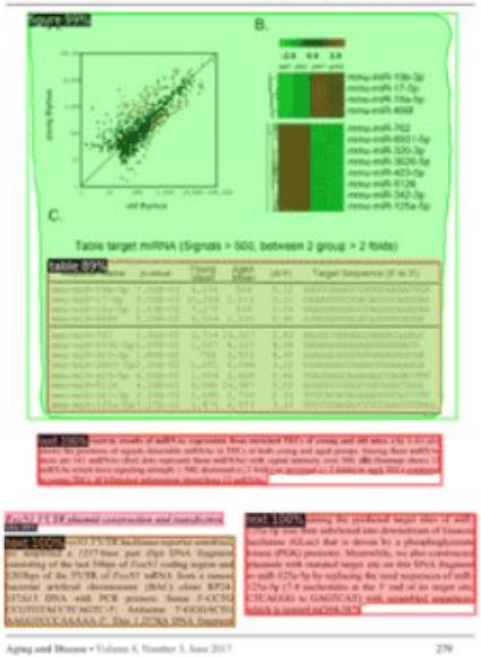
Separation and Recognition of Text and Images from Scanned PDFs Using Convolutional Neural Networks (CNNs) in Keras

The GenG Model for Documents Layout Segmentation

Presented by: Lambda Team

Date: 30.09.2024



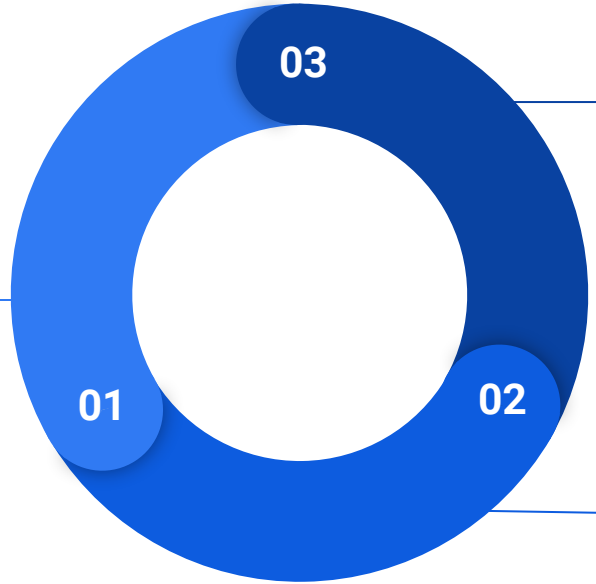
Project Objective

- **Build a deep learning model that processes scanned PDFs, separates images and text, identifies labels/captions associated with images, and stores them in a structured format (e.g., JSON or table).**
- **Develop an application (API or CLI) to automate the process, allowing users to upload PDFs and receive structured output.**

Problem Breakdown

Input

Scanned PDFs containing both text and images



Challenges

- Distinguishing between text, images, and captions in noisy scanned documents.
- Correctly associating image captions with the corresponding image.
- Processing documents of varying layouts and formats.

Output

Separated Images: Extracted from the document.
Text Extraction: Text and associated labels (headlines for images).
Structured Data: Organized into JSON format or table, indexed by document parts (text, images, captions).

Dataset Selection

Primary Dataset: PubLayNet

- Large-scale document layout dataset.
- Annotations for text blocks, figures, tables, and other document components.
- **Usage**: Train model to segment and classify regions of interest (images, text, captions).

Secondary Dataset (Optional for Fine-Tuning): PRImA

- Document layout dataset with complex layouts.
- **Usage**: Fine-tuning the model for advanced segmentation.

Proposed Solution

Model Architecture

01	**Image Segmentation**	• Model: U-Net or Mask R-CNN. • Segment the document into regions (text, image, captions).
02	**Region Classification**	• Model: Shallow CNN or fully connected layers. • Classify each segmented region (text, image, caption).
03	**Text Extraction**	• OCR Tool: Tesseract (for extracting text from segmented text regions). • Extract text content from the segmented text blocks.
03	**Post-processing**	• Output Format: Structured JSON file, containing images, captions, and remaining text.

Workflow and Sub Tasks

Data Engineering	Model Development	API/Backend Development	Data Management
Task: Data collection and preprocessing (PDF conversion, image extraction, annotation processing). **Tools**: PyMuPDF, pdf2image, OpenCV, Keras preprocessing utilities.	**Task**: Develop and train the CNN models for segmentation and classification. **Tools**: Keras (with TensorFlow backend), U-Net or Mask R-CNN architecture, Tesseract for OCR.	**Task**: Develop a Python-based API or CLI that processes uploaded PDFs and returns the extracted data in a structured format. **Tools**: Flask/FastAPI, Python, JSON handling, database storage (SQL/NoSQL).	**Task**: Design and manage the storage of extracted data (images, captions, text) in structured format (JSON/SQL). **Tools**: pandas, SQLAlchemy, JSON handling libraries.

The Model

Architecture in Keras

Part 1: Segmentation: A convolutional neural network (CNN) for document layout analysis (image segmentation).

Part 2: Classification and Text Extraction: Post-segmentation, classify regions and apply OCR on text regions.

1. Image Segmentation

We will use a **U-Net** or **Mask R-CNN** architecture for image segmentation to divide the document into regions of interest (text, images, etc.).

- **U-Net** is a popular segmentation architecture that works well for tasks where precise boundaries of regions (images, text) are needed.
- **Mask R-CNN** can be used for detecting and segmenting objects in an image and works well for more complex layouts.

Model Architecture Example (U-Net)

```python
from keras.models import Model
from keras.layers import Conv2D, MaxPooling2D, UpSampling2D, Concatenate, Input

def unet_model(input_size=(256, 256, 3)):
    inputs = Input(input_size)

    # Encoder (downsampling path)
    conv1 = Conv2D(64, 3, activation='relu', padding='same')(inputs)
    conv1 = Conv2D(64, 3, activation='relu', padding='same')(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

    # Add more encoding layers as needed
    # Decoder (upsampling path)
    up1 = UpSampling2D(size=(2, 2))(pool1)
    conv_final = Conv2D(1, 1, activation='sigmoid')(up1)

    model = Model(inputs=[inputs], outputs=[conv_final])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

2. Classification

Once the regions are segmented, we will use a small classifier (like a shallow CNN or fully connected layers) to classify the segmented regions (as image, text, or caption).

```
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D

def classifier_model(input_shape=(64, 64, 3), num_classes=3):
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))

    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

3. OCR for Text Extraction

For text regions, we will apply an OCR system like Tesseract to extract the text. This step can be done after the segmentation and classification models have identified the text regions.

```
import pytesseract
from PIL import Image

def extract_text(image_region):
    text = pytesseract.image_to_string(image_region)
    return text
```

Development Plan

Step by Step Guide

Development Plan

1. Preprocessing:

- Convert PDFs to images if needed (e.g., use `pdf2image`).
- Resize images and normalize pixel values.
- Generate segmentation masks using the PubLayNet annotations.

2. Build the Segmentation Model:

- Start with a U-Net or Mask R-CNN architecture to segment the document into regions.
- Train the segmentation model using the PubLayNet dataset, focusing on detecting text, image, and other components.

3. Build the Classification Model:

- After segmentation, use a small CNN model to classify the segmented regions (image, text, caption).
- Train this classifier using labeled data (from PubLayNet or other datasets).

4. OCR for Text Regions:

- Apply Tesseract or any other OCR library to extract text from regions classified as “text.”

5. Post-Processing:

- Combine the results into a structured format like JSON. For each PDF page, you’ll have entries for images, captions, and the remaining text, organized by their position.

6. Evaluation:

- Evaluate the segmentation and classification performance using metrics like IoU (Intersection over Union) for segmentation and accuracy for classification.
- Validate the quality of OCR-extracted text by comparing it with the ground truth.

Alternative Solution

A Combination of LayoutParser and Keras

1. **Layout Detection and Segmentation**: Use **LayoutParser** with pre-trained models (such as the PubLayNet model) for layout detection to quickly identify key document components (text blocks, images, tables, forms).

2. Custom CNN Development: Build a custom **classifier** using **Keras** to fine-tune or enhance the layout detection and perform additional tasks like:

- Classification of segmented regions.
- Post-processing, such as caption extraction, table structure recognition, or handling any document-specific customisation.

Detailed Workflow

1. Document Preprocessing:

- Convert PDFs to images using a library like `pdf2image`.
- Preprocess images (resize, normalize) and prepare them for input to both the pre-trained layout model and your custom Keras model.

2. Use LayoutParser for Layout Detection:

- **Why LayoutParser**: It comes with pre-trained models (like those trained on PubLayNet) that are already optimized for detecting key document components (text, tables, images, and forms). Using it saves time and effort in developing a new model for layout detection from scratch.

LayoutParser Code Example

```python
import layoutparser as lp
from pdf2image import convert_from_path

# Convert PDF to image
images = convert_from_path('document.pdf')

# Load the pre-trained layout model
model = lp.Detectron2LayoutModel('lp://PubLayNet/faster_rcnn_R_50_FPN_3x/config')

for img in images:
    # Detect the layout
    layout = model.detect(img)
    # Visualize the detected layout (Optional)
    lp.draw_box(img, layout, box_width=3).show()

    # Extract the layout elements (tables, text, etc.)
    text_blocks = lp.Layout([b for b in layout if b.type == 'Text'])
    table_blocks = lp.Layout([b for b in layout if b.type == 'Table'])
    image_blocks = lp.Layout([b for b in layout if b.type == 'Figure'])
```

3. Custom Keras Model for Classification and Fine-Tuning:

After using LayoutParser for initial layout detection, we can feed the detected regions into your **custom-built Keras classifier**. This custom classifier will allow us to fine-tune or improve the performance by adding another level of processing.

- **Classifier Task**: Once LayoutParser segments the document into regions (tables, text, images), our Keras model can be used to further classify or clean the segmentation. For example, it can fine-tune distinguishing between captions and text or confirm if a detected table is a form instead.
- **Table and Form Processing**: If LayoutParser detects a table, we can run it through a Keras-based classifier to check whether it's actually a table or a form.

Custom Keras Classifier Example

```python
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

def build_classifier(input_shape):
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(4, activation='softmax'))  # 4 classes: text, image, table, form

    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# Example input shape (patches of document region)
classifier = build_classifier((64, 64, 3))
```

4. Post-processing and Integration

After classification by both **LayoutParser** and our custom Keras classifier:

- **OCR**: Apply OCR to text regions to extract the actual text.
- **Table Structure Extraction**: For table regions, we can apply additional post-processing steps to recognize rows and columns, using tools like OpenCV combined with our custom logic.
- **Form Data Extraction**: Similarly, for form regions, we can detect form fields and extract key-value pairs (like a user's name and address from a form).

Combining Results into JSON

Once we have segmented and classified the document, we will compile the results into a structured format (like JSON). This JSON will contain details about the positions of text blocks, tables, images, and forms, along with any extracted content (like captions or OCR-extracted text).

```
{
    "images": [{"index": 1, "image_path": "image1.png", "caption": "Sample Caption"}],
    "tables": [{"index": 1, "table_data": [["A1", "B1"], ["A2", "B2"]]}],
    "forms": [{"index": 1, "form_fields": {"Name": "John Doe", "Age": "30"}}],
    "text": "Remaining document text here."
}
```

Why the hybrid approach is effective?

1. **Pre-trained Layout Models**: We use **LayoutParser's pre-trained models** for robust document layout detection, which saves development time and provides high accuracy.
2. **Custom Development with Keras**: We show that we are developing part of the system by creating a **Keras classifier** to handle specific document regions and add further validation or fine-tuning of the layout detection.
3. **Handling Complex Layouts**: LayoutParser efficiently handles the layout detection of multiple elements (text, tables, forms, images), and your custom Keras model can be used to further improve classification accuracy or handle custom processing for tables and forms.
4. **Balance Between Pre-trained and Custom**: This solution demonstrates both usage of cutting-edge pre-trained models (via LayoutParser) and our own implementation with Keras, making it a strong, balanced approach.

Thank You