UNIVERSIDAD CRISTÓBAL COLÓN

INTELIGENCIA ARTIFICIAL

Implementación de OCR

Equipo 4:

- Alejandro Harael Garcia Sanchez
- Ana Paula Guillen Maldonado
- Cesar Briones Martínez
- Sacnicté Cruz Arellano

Grupo A
Ingeniería en Sistemas Computacionales
Octavo Semestre

Código completo

```
from flask import Flask, request, jsonify
from werkzeug.utils import secure_filename
import os
import cv2
import easyocr
import fitz
from PIL import Image
import sys

app = Flask(_name_)

UPLOAD_FOLDER = '.'
ALLOWED_EXTENSIONS = {'pdf'}
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in
ALLOWED_EXTENSIONS

def pdf2png(file_name):
    directory_path = "./Output_files/"
    base_name = os.path.splitext(os.path.basename(file_name))[0]
    generated_files = []

    if not os.path.exists(directory_path):
        os.makedirs(directory_path)
        print(f"Directory '{directory_path}' created
successfully.")

    existing_files = [f for f in os.listdir(directory_path) if
f.startswith(base_name) and f.endswith('.png')]
    if existing_files:
        generated_files = [os.path.join(directory_path, f) for f
in existing_files]

    pdf_document = fitz.open(file_name)
    for page_number in range(pdf_document.page_count):
        page = pdf_document.load_page(page_number)
        img = page.get_pixmap(matrix=fitz.Matrix(600/300,
600/300))
        img_pillow = Image.frombytes("RGB", [img.width,
img.height], img.samples)
        new_file_name = f"{directory_path}{base_name}_{page_number
+ 1}.png"
```

```
        if not os.path.exists(new_file_name):
            img_pillow.save(new_file_name, "PNG")
            generated_files.append(new_file_name)
            print(f"File '{new_file_name}' saved successfully.")
    pdf_document.close()
    return generated_files


class Reader():
    def _init_(self):
        languages = ['en','es','fr','de','it','pt']
        self.reader = easyocr.Reader(languages,
model_storage_directory=os.path.join('models'),
download_enabled=True)

    def read_img(self, img_path):
        img = cv2.imread(img_path)
        return img

    def extract_text(self, img):
        result = self.reader.readtext(img)
        extracted_text = []
        for text in filter(lambda x: x[-1] > .45, result):
            box, acc_text, confidence = text
            img = cv2.rectangle(img, [int(i) for i in box[0]],
[int(i) for i in box[2]], (0, 255, 0), 2)
            extracted_text.append(acc_text)
        return extracted_text, img

def create_text(output_file_name, text):
    output_file_path =
f"./Output_files/{output_file_name[:-4]}.txt"
    text_joined = ','.join(text)
    with open(output_file_path, 'w') as file2write:
        file2write.write(text_joined)
    print(f"Archivo txt para {output_file_name} generado con
éxito")
    return text_joined

@app.route('/archs', methods=['POST'])
def upload_file():
    if 'file' not in request.files:
        return jsonify({'error': 'No se ha proporcionado ningún
archivo'})

    file = request.files['file']
    if file.filename == '':
```

```
        return jsonify({'error': 'No se ha seleccionado ningún
archivo'})
    if file and allowed_file(file.filename):
        filename = secure_filename(file.filename)
        filepath = os.path.join(app.config['UPLOAD_FOLDER'],
filename)
        file.save(filepath)

        generated_files = pdf2png(filepath)
        reader = Reader()  # Instancia de la clase Reader
        extracted_text_all = []
        for generated_file in generated_files:
            img = reader.read_img(generated_file)
            extracted_text, _ = reader.extract_text(img)  #
Corregido para llamar al método de la instancia
            extracted_text_all.extend(extracted_text)

        output_text = create_text(filename, extracted_text_all)
        for generated_file in generated_files:
            os.remove(generated_file)
        os.remove(filepath)

        return jsonify({'texto_extraído': output_text})
    else:
        return jsonify({'error': 'Extensión de archivo no
permitida'})

def main():
    if len(sys.argv) < 2:
        print("Uso: python archivo.py <input.pdf> [output.txt]")
        sys.exit(1)

    input_file = sys.argv[1]
    output_file = sys.argv[2] if len(sys.argv) > 2 else
input_file.replace('.pdf', '.txt')

    generated_files = pdf2png(input_file)
    reader = Reader()  # Instancia de la clase Reader
    extracted_text_all = []
    for generated_file in generated_files:
        img = reader.read_img(generated_file)
        extracted_text, _ = reader.extract_text(img)  # Corregido
para llamar al método de la instancia
        extracted_text_all.extend(extracted_text)

    output_text = create_text(output_file, extracted_text_all)
    print(output_text)
```

```
if _name_ == '_main_':
    if 'flask' in sys.argv:
        app.run(debug=True)
    else:
        main()
```

Análisis del Código

```
from flask import Flask, request, jsonify
from werkzeug.utils import secure_filename
import os
import cv2
import easyocr
import fitz
from PIL import Image
import sys
```

Importamos los módulos necesarios para acceder a la funciones que el programa requerirá.

Importamos las clases **Flask**, request y jsonify de la biblioteca flask.
Flask es un micro framework que nos servirá para crear la aplicación web en Python.

Request se utilizará para manejar las solicitudes HTTP, mientras que **jsonif**y es una función para devolver respuestas JSON.

La función **secure_filename** se usa para asegurar un nombre de archivo seguro antes de almacenarlo en el sistema de archivos.

import os: Este módulo nos proporciona funciones para interactuar con el sistema operativo, como manejo de directorios y rutas de archivos.

import cv2: Al importar esta biblioteca de open source , nos otorga funciones para el procesamiento de videos e imágenes, gracias a su enfoque de visión computacional.

import easyocr: Importamos la biblioteca de reconocimiento óptico de caracteres que nos permite extraer texto de una imagen.

import fitz: Al importar fitz, traemos la biblioteca PyMuPDF , la cual tiene funcionalidades para trabajar con archivos PDF dentro de Python.

import sys: Importamos el módulo sys al entorno de python, este módulo proporciona acceso a algunas variables y funciones que interactúan con el intérprete de Python y el entorno del sistema. Este se encuentra incluido dentro de la biblioteca estándar de python.

```
app = Flask(_name_)

UPLOAD_FOLDER = '.'
ALLOWED_EXTENSIONS = {'pdf'}
```

```
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
```

Se crea una instancia de la clase Flask , y se procede a configurar una carpeta de carga y una lista con las extensiones permitidas para los archivos a cargar.

```
def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in
ALLOWED_EXTENSIONS
```

Se define una función llamada “`allowed_file`” la cual verifica si la extensión del archivo proporcionado encaja con la extensión de archivos permitida en nuestra lista de extensiones creada.

```
def pdf2png(file_name):
    directory_path = "./Output_files/"
    base_name = os.path.splitext(os.path.basename(file_name))[0]
    generated_files = []

    if not os.path.exists(directory_path):
        os.makedirs(directory_path)
        print(f"Directory '{directory_path}' created
successfully.")

    existing_files = [f for f in os.listdir(directory_path) if
f.startswith(base_name) and f.endswith('.png')]
    if existing_files:
        generated_files = [os.path.join(directory_path, f) for f
in existing_files]
```

En este apartado se define la función para convertir un archivo pdf en varias imágenes para que puedan ser procesadas en formato de imagen. Con la variable `directory_path` creamos un directorio donde guardaremos los archivos png. De ahí se inicia una lista vacía llamada `generated_files` que se utilizará para almacenar los nombres de los archivos png generados.

Verificamos si ya existe el directorio y si no se procede a crear. Aquí se crea una lista llamada `existing_files` que contiene los nombres de todos los archivos PNG en el directorio `directory_path` que comienzan con el nombre base del archivo PDF y terminan con ".png".

```
 pdf_document = fitz.open(file_name)
    for page_number in range(pdf_document.page_count):
        page = pdf_document.load_page(page_number)
        img = page.get_pixmap(matrix=fitz.Matrix(600/300,
600/300))
```

```
        img_pillow = Image.frombytes("RGB", [img.width,
img.height], img.samples)
        new_file_name = f"{directory_path}{base_name}_{page_number
+ 1}.png"
        if not os.path.exists(new_file_name):
            img_pillow.save(new_file_name, "PNG")
            generated_files.append(new_file_name)
            print(f"File '{new_file_name}' saved successfully.")
    pdf_document.close()
    return generated_files
```

Se abre el archivo con la biblioteca PyMuPDF (fitz) , se procede a crear un bucle para iterar por cada página del documento para obtener el mapa de bits por cada página que después utilizará la biblioteca Python Imaging Library para crear la imagen.
De ahí se genera el nombre del archivo PNG para la página actual, utilizando el nombre base del archivo PDF, el número de página actual y la extensión ".png".

Si el archivo PNG no existe en el directorio de salida, se guarda utilizando `img_pillow.save()` y se agrega su nombre a la lista `generated_files`. Se imprime un mensaje indicando que el archivo se ha guardado con éxito.
Se cierra el documento PDF después de que se hayan procesado todas las páginas y la función devuelve la lista de nombres de los archivos png generados.

```
class Reader():
    def _init_(self):
        languages = ['en','es','fr','de','it','pt']
        self.reader = easyocr.Reader(languages,
model_storage_directory=os.path.join('models'),
download_enabled=True)

    def read_img(self, img_path):
        img = cv2.imread(img_path)
        return img

    def extract_text(self, img):
        result = self.reader.readtext(img)
        extracted_text = []
        for text in filter(lambda x: x[-1] > .45, result):
            box, acc_text, confidence = text
            img = cv2.rectangle(img, [int(i) for i in box[0]],
[int(i) for i in box[2]], (0, 255, 0), 2)
            extracted_text.append(acc_text)
        return extracted_text, img
```

Definimos una clase Reader con métodos para inicializar el lector de EasyOCR.
Dentro del constructor, se define una lista de lenguajes que contiene los idiomas en los que se va a realizar el reconocimiento de texto. Luego, se inicializa un objeto Reader, `read_img` toma una ruta de archivo de imagen como entrada, utilizamos opencv para leer el

archivo especificado por img_path , `extract_text` toma una imagen como entrada y ocupamos el objeto reader para extraer el texto de la imagen y lo guardamos en result .Itera sobre cada resultado de texto en result y se agrega a la lista `extracted_text`. Además, se dibuja un rectángulo alrededor del texto en la imagen original utilizando OpenCV. (cv2.rectangle).

Finalmente, devuelve la lista de texto extraído (`extracted_text`) junto con la imagen que contiene los rectángulos dibujados (img).

```
def create_text(output_file_name, text):
    output_file_path =
f"./Output_files/{output_file_name[:-4]}.txt"
    text_joined = ','.join(text)
    with open(output_file_path, 'w') as file2write:
        file2write.write(text_joined)
    print(f"Archivo txt para {output_file_name} generado con
éxito")
    return text_joined

@app.route('/archs', methods=['POST'])
```

Definimos la función create text , que toma el archivo de salida y una lista de texto, se concatena el nombre de salida con el directorio de salida , de ahí se convierte la lista de texto en una cadena única utilizando join(text).Después, escribe esta cadena en un archivo de texto en la ruta especificada (`output_file_path`).

Por último, la app route define una ruta '/archs' que escucha las solicitudes POST. Esto hace que la aplicación espere recibir archivos a través de solicitudes POST en esta ruta.

```
def upload_file():
    if 'file' not in request.files:
        return jsonify({'error': 'No se ha proporcionado ningún
archivo'})

    file = request.files['file']
    if file.filename == '':
        return jsonify({'error': 'No se ha seleccionado ningún
archivo'})
    if file and allowed_file(file.filename):
        filename = secure_filename(file.filename)
        filepath = os.path.join(app.config['UPLOAD_FOLDER'],
filename)
        file.save(filepath)

        generated_files = pdf2png(filepath)
        reader = Reader()  # Instancia de la clase Reader
        extracted_text_all = []
        for generated_file in generated_files:
            img = reader.read_img(generated_file)
            extracted_text, _ = reader.extract_text(img)  #
Corregido para llamar al método de la instancia
            extracted_text_all.extend(extracted_text)

        output_text = create_text(filename, extracted_text_all)
```

```
        for generated_file in generated_files:
            os.remove(generated_file)
        os.remove(filepath)

        return jsonify({'texto_extraído': output_text})
    else:
        return jsonify({'error': 'Extensión de archivo no
permitida'})
```

Definimos la función `upload_file` , se comprueba si hay un archivo en la solicitud. Si no hay un archivo en la solicitud, devuelve un mensaje de error JSON indicando que ningún archivo ha sido proporcionado.

Obtiene el archivo de la solicitud y comprueba si tiene un nombre de archivo. Si el nombre del archivo está vacío, devuelve un mensaje de error JSON indicando que ningún archivo ha sido seleccionado.

Comprueba si se proporciona un archivo válido (file) y si la extensión del archivo es permitida utilizando la función `allowed_file()`. Si el archivo es válido, asegura el nombre del archivo utilizando `secure_filename()`, luego crea la ruta completa del archivo utilizando el directorio de carga especificado en la configuración de la aplicación (`app.config['UPLOAD_FOLDER']`), y finalmente guarda el archivo en esa ruta.

Convierte el archivo PDF cargado en imágenes PNG utilizando la función `pdf2png()`

Llamamos a la clase Reader y utiliza sus métodos para extraer texto de las imágenes PNG generadas.Creamos un archivo de texto con el texto extraído utilizando la función `create_text()`. Se eliminan los archivos PNG generados y el archivo PDF original después de extraer el texto y por último devuelve un mensaje JSON que contiene el texto extraído.

```
def main():
    if len(sys.argv) < 2:
        print("Uso: python archivo.py <input.pdf> [output.txt]")
        sys.exit(1)

    input_file = sys.argv[1]
    output_file = sys.argv[2] if len(sys.argv) > 2 else
input_file.replace('.pdf', '.txt')

    generated_files = pdf2png(input_file)
    reader = Reader()  # Instancia de la clase Reader
    extracted_text_all = []
    for generated_file in generated_files:
        img = reader.read_img(generated_file)
        extracted_text, _ = reader.extract_text(img)  # Corregido
para llamar al método de la instancia
```

```
        extracted_text_all.extend(extracted_text)

    output_text = create_text(output_file, extracted_text_all)
    print(output_text)

if _name_ == '_main_':
    if 'flask' in sys.argv:
        app.run(debug=True)
    else:
        main()
```

Definimos la función main , verificamos si se proporcionan al menos dos argumentos en la línea de comandos. Si no,se imprime un mensaje de uso y sale del programa con un código de salida de 1. Obtiene el nombre del archivo de entrada (PDF) y el nombre del archivo de salida (texto) de los argumentos de la línea de comandos. Si no se proporcionará un nombre de archivo de salida, se genera uno automáticamente reemplazando la extensión .pdf del archivo de entrada con .txt.

Convierte el archivo PDF de entrada en imágenes PNG utilizando la función `pdf2png()`.

Llamamos a la clase Reader y utilizando sus métodos extrae el texto de las imágenes PNG generadas.Crea un archivo de texto con el texto extraído utilizando la función `create_text()` y lo imprime en la consola.

Para finalizar verifica si el script se está ejecutando directamente. De ser el caso, verifica si 'flask' está presente en los argumentos de la línea de comandos. Si 'flask' está presente, inicia la aplicación Flask en modo de depuración. De lo contrario, llama a la función `main()` para iniciar el procesamiento de los archivos PDF y extraer texto.