https://doi.org/10.22201/dgtic.ctud.2024.2.1.37

Vol. 2, Núm. 1. enero-marzo 2024, págs. 55 - 65

Comparativa de servicios OCR para documentos administrativos en PDF con Java/Tess4J y Python/EasyOCR

Información del reporte:

Para citar este reporte técnico:

Ortega Cuevas, I. (2024). Comparativa de servicios OCR para documentos administrativos en PDF con Java/Tess4J y Python/EasyOCR. *Cuadernos Técnicos Universitarios* de la DGTIC, *2* (1), páginas (55 - 65).

https://doi.org/10.22201/dgtic.ctud.2024.2.1.37

Israel Ortega Cuevas

Dirección General de Personal

Universidad Nacional Autónoma de México

iortega@dgp.unam.mx

ORCID: 0000-0001-6352-5400

Resumen

Las entidades y dependencias universitarias manejan documentos administrativos, que a menudo están almacenados en formatos digitales que consisten exclusivamente en imágenes. Esto genera demoras en el acceso al contenido, y dificulta su utilidad en búsquedas y toma de decisiones. Con el propósito de acceder al contenido de texto de estos documentos, se desarrollaron y compararon dos servicios web de procesamiento óptico de caracteres (OCR). El primer servicio está implementado en lenguaje Java, utilizando el marco de desarrollo Spring y la biblioteca Tess4J, mientras que el segundo servicio se ha desarrollado en Python, haciendo uso de la biblioteca EasyOCR. En esta comparación, se evaluaron los tiempos de respuesta al procesar 50 documentos con contenido administrativo y en formato PDF, los cuales contienen información exclusivamente en forma de imágenes. El código fuente de ambos servicios está disponible en el repositorio GitHub, lo que facilita su implementación y uso. Los resultados indican que el servicio Java presenta un tiempo de procesamiento de documentos con una ventaja de dos segundos respecto a Python; sin embargo, se destaca que los resultados pueden variar en condiciones distintas, ya sea porque los algoritmos de OCR utilizados implementan procesos diferentes, o en el caso de documentos de otro ámbito distinto al administrativo, o bien, por encontrarse en equipos con tarjeta gráfica dedicada. Es importante señalar como limitación de este estudio que no se considera la precisión del texto recuperado.

Palabras clave:

Procesamiento óptico de caracteres, benchmarking, Java, Python.

https://doi.org/10.22201/dgtic.ctud.2024.2.1.37 Vol. 2, Núm. 1. enero-marzo 2024, págs. 56 - 65

1. INTRODUCCIÓN

Los lenguajes de programación de alto nivel fueron la respuesta para que los programadores pudieran abstraer la lógica del lenguaje de máquina conformado por los dígitos '0' y '1' y, de esta manera, hacer más comprensibles los procesos que realiza una computadora para lograr una finalidad. Estos lenguajes se conforman por instrucciones que reciben entradas de un usuario, de otra computadora o incluso de sensores; las cuales son ejecutadas paso por paso en un flujo lógico, todavía determinado por un programador.

Aunque desde la década de los años cincuenta se desarrollaron los primeros lenguajes de programación modernos (destacan Fortran, LISP y Cobol) (Wexelblat, 1981), que se han encontrado en constante evolución hasta el día de hoy, dos de ellos son de los más importantes y utilizados en el desarrollo de sistemas modernos: Java y Python, por la multitud de bibliotecas que soportan y por beneficios multiplataforma que ofrecen.

Java es un lenguaje desarrollado durante la década de los años noventa por James Gosling, y publicado por la empresa Sun Microsystems en 1995 (Gosling, Joy, Steele, Bracha, y Buckley, 2015). Su diseño original fue el de ser un lenguaje informático de uso general, utilizado desde entonces en el desarrollo de aplicaciones de líneas de comandos, de uso en sistemas operativos gráficos, para servidores web, aplicaciones móviles, de videojuegos y de servicios *backend*. Tiene un uso muy extendido, contando con miles de proyectos de propósito específico y extensión de funcionalidad con marcos de desarrollo como Spring (Broadcom, 2023).

Con el propósito de que el lenguaje Java pudiera ser portable y ejecutado en ambientes multiplataforma, fue diseñado para que sus aplicativos pudieran ejecutarse bajo un programa llamado *Java Virtual Machine (JVM)*, que convierte un código binario pre-compilado e igual para todos los entornos de programación, a un código que es interpretado por una plataforma específica. Esto ha permitido que los programas de este lenguaje se ejecuten en más de 60 mil millones de JVM en el mundo (Oracle Inc., 2023).

Otro lenguaje de programación, de sentido versátil, también orientado a objetos y de alto nivel es Python, el cual cuenta con una historia que se remonta a 1991 y que fue desarrollado por Guido van Rossum (Martelli, Ravenscroft, y Holden, 2017). Además de los usos en aplicativos móviles, desarrollo web, servicios web, entre otros; su tendencia actual es su amplia utilización en ciencia de datos e inteligencia artificial, con sus abstracciones en aprendizaje automático y profundo.

El lenguaje Python también es interpretado pero, a diferencia de Java, no es convertido a un código binario intermedio, sino que su intérprete va leyendo el código fuente línea a línea. Este lenguaje también puede implementarse en multiplataforma y se destaca por una alta velocidad de respuesta.

Asimismo, se consideraron estos dos lenguajes de programación debido a su amplio uso en las aplicaciones que gestionan servicios en la Universidad Nacional Autónoma de México (UNAM); además, son accesibles desde aplicaciones desarrolladas en otros lenguajes como JavaScript, PHP, C# (C Sharp), entre otros, esto mediante servicios web. En el contexto de la creciente transformación digital actual (Llorens, 2022), varios servicios tienen una demanda significativa en la administración universitaria, entre ellos destaca el Reconocimiento Óptico de Caracteres (OCR o en inglés "*Optical Character Recognition*"), que resulta útil para la lectura de documentos escaneados en formato de imagen y su posterior transformación a caracteres que pueden ser aprovechados en otros procesos de información. Por ejemplo, en las bases de

https://doi.org/10.22201/dgtic.ctud.2024.2.1.37 Vol. 2, Núm. 1. enero-marzo 2024, págs. 57 - 65

datos institucionales de la UNAM existen documentos en formato PDF que sólo pueden ser vistos como imágenes y no se consigue acceder a su contenido en búsquedas basadas en texto.

Así, nuestro objetivo es llevar a cabo una comparación entre dos implementaciones de OCR, una en lenguaje Java y otra en Python, utilizando una arquitectura de servicios web que satisfaga las necesidades de transformación de documentos escaneados en información útil para las dependencias universitarias. Además de presentar el referente de cómo pueden realizarse estas operaciones para su replicación en otros servicios universitarios, se tendrá una comparación entre los tiempos de respuesta en estos lenguajes.

Como limitante de este estudio, sólo se comparan los resultados de procesamiento, sin considerar la precisión respecto al número de caracteres correctamente reconocidos, debido a que depende de la calidad del documento y su correcta digitalización.

2. DESARROLLO TÉCNICO

Para realizar la prueba comparativa de documentos, se desarrollan tres aplicativos, uno de lectura de imágenes guardadas con formato PDF y dos servicios para el procesamiento OCR, uno en Java y otro en Python.

- El primer programa, que invocará los servicios Java y Python, realiza la lectura de un directorio que contiene 50 documentos PDF, cada uno diferente, pero con las características de un oficio de la administración universitaria. Es relevante señalar que la medición de los tiempos de operación se lleva a cabo para cada documento, generando un resultado individual por cada operación. Después de realizar varias ejecuciones, se observó que los tiempos de lectura por documento varían mínimamente. Este comportamiento se atribuye al uso de un equipo dedicado específicamente a esta acción durante la realización de la prueba.
- El segundo y tercer aplicativo, son servicios que permiten recibir un documento PDF, aplicar algoritmos de OCR y entregar como resultado el contenido del documento en texto. La comparación se realizará entre estos dos servicios, uno desarrollado en lenguaje Java y el otro en lenguaje Python. Para evitar procesos de lectura/escritura en el equipo servidor se trabaja en lo posible con documentos temporales en memoria, aunque algunas actividades de procesamiento generan necesariamente documentos temporales.

Los dos servicios se probarán en un equipo con las siguientes características:

- Servidor RACK 2U Intel
- Dos procesadores Intel Xeon ES 2650V3 de 2 GHz
- 64 GB memoria RAM
- Disco interno mecánico 3.5 de 1TB
- Sistema operativo Debian GNU/Linux 11

https://doi.org/10.22201/dgtic.ctud.2024.2.1.37 Vol. 2, Núm. 1. enero-marzo 2024, págs. 58 - 65

El proceso de lectura de documentos e invocación de servicios se ejecuta desde un equipo diferente a donde se implementaron los servicios, con las siguientes características:

- Intel(R) Core(TM) i7-7700 CPU a 3.60GHz
- 32 GB memoria RAM
- Disco interno estado sólido 3.5 de 1TB
- Sistema operativo Windows 11 de 64 bits

Los equipos utilizados durante la ejecución de los servicios no estuvieron realizando actividades adicionales, sujetándose a realizar únicamente la ejecución requerida. Además, se realizaron corridas previas donde se garantiza que el tiempo de procesamiento tiene mínimas variaciones respecto a la que se utiliza para la validación final.

Supondremos, por lo comentado anteriormente, que las corridas que se realizaron fueron independientes, con distribución normal, por lo que estadísticamente una prueba "*t*" permitió distinguir si estos procesos se comportaron en forma similar o hubo diferencias significativas de rendimiento.

2.1 APLICATIVO DE LECTURA DE DOCUMENTOS Y TRANSFERENCIA AL SERVICIO DE OCR

El primer aplicativo fue desarrollado bajo lenguaje Java con apoyo de la librería spring-boot-starter-webflux versión 3.0.3, para el llamado de servicios web. Su labor consistió en leer un directorio con 50 documentos en formato PDF, cada uno de una página de extensión y con las características de un oficio: clave de documento, asunto, destinatario, contenido y remitente. El documento fue ingresado en formato binario a un objeto de tipo java.io.File y sólo contiene la imagen del documento sin contener datos de OCR, los cuales serán retornados por los servicios cuando se invoquen.

El programa leyó cada documento y lo envió a un servicio que regresó el contenido después de aplicar un proceso OCR. El aplicativo estableció los tiempos de respuesta en milisegundos de cada documento y, dado que se supone una distribución normal, entregó la media del procesamiento y su desviación estándar.

En la Figura 1 se muestra la sección del código que se ejecuta después de realizada la lectura de información de los archivos PDF en un objeto lista con contenido de un objeto File de Java. Se utiliza la invocación a la clase RestTemplate del marco de trabajo Spring para realizar la invocación al servicio apuntado por la variable "url" (línea 12 de la Figura 1) que contiene en la primera corrida la llamada al servicio Java y en la segunda corrida la llamada al servicio Python.

https://doi.org/10.22201/dgtic.ctud.2024.2.1.37 Vol. 2, Núm. 1. enero-marzo 2024, págs. 59 - 65

Figura 1

Sección del código en lenguaje Java que procesa 50 documentos y obtiene la media y la desviación estándar

```
1.  for (File file : files) {
2.      long tInicio = System.currentTimeMillis();
3.
4.      HttpHeaders headers = new HttpHeaders();
5.      headers.setContentType(MediaType.MULTIPART_FORM_DATA);
6.      MultiValueMap<String, Object> body = new LinkedMultiValueMap<>();
7.      body.add("file", new FileSystemResource(file));
8.      HttpEntity<MultiValueMap<String, Object>> requestEntity = new
    HttpEntity<>(body, headers);
9.
10.     RestTemplate restTemplate = new RestTemplate();
11.     ResponseEntity<String> response = restTemplate
12.             .postForEntity(url, requestEntity, String.class);
13.
14.     long tFinal = System.currentTimeMillis();
15.     long timeElapsed = tFinal - tInicio;
16.
17.     valoresEnMilisegundos.add(timeElapsed);
18. }
19.
20. System.out.println("Valores: " + valoresEnMilisegundos);
21.
22. LlamadaServicios llamadaServicios = new LlamadaServicios();
23. llamadaServicios.calculaMediaySD(valoresEnMilisegundos);
```

Nota: El código fuente completo se encuentra disponible para su consulta en la siguiente dirección: https://github.com/israelortega/LlamadaServicios

2.2 SERVICIO DE OCR EN LENGUAJE JAVA

El segundo aplicativo se elaboró en lenguaje Java y con apoyo del marco de desarrollo Spring y las bibliotecas de software net.sourceforge.tess4j –Test4J– (2022) y pdfbox.apache.org –PDFDocument– (The Apache Software Foundation, 2023). La aplicación consistió de un servicio web que es invocado a través de la entrega de un PDF y que regresa su contenido en formato de texto tras un procesamiento de OCR.

La biblioteca Tess4J es una interfaz que permite utilizar un motor de código abierto OCR llamado Tesseract, con soporte para los siguientes tipos de imágenes: TIFF, JPEG, GIF, PNG, y BMP, así como para documentos PDF.

Tesseract fue desarrollado inicialmente por HP entre 1985 y 1995; sin embargo, la compañía Google se encarga de continuar con su desarrollo (Kay, 2007). Tesseract, trabaja en cuatro etapas (Klippa, 2023):

1. Procesamiento: que realiza la conversión de la imagen a una escala de blanco y negro, con eliminación de ruido y escalado a una medida estándar.
2. Segmentación de caracteres: que aplica un proceso algorítmico que puede identificar los bordes de los caracteres.

https://doi.org/10.22201/dgtic.ctud.2024.2.1.37 Vol. 2, Núm. 1. enero-marzo 2024, págs. 60 - 65

3. Reconocimiento óptico: que utiliza una base de datos que contiene información sobre los caracteres, en este caso del idioma español.

4. Posprocesamiento: con la eliminación de caracteres no deseados, espacios en blanco y posibles errores ortográficos.

Tesseract utiliza una red neuronal LSTM (Long Short-Term Memory) para el reconocimiento de caracteres. Estas redes se caracterizan por procesar información de entrada en forma secuencial, guardando información de entradas anteriores y para su uso en la generación de las salidas esperadas; esto es logrado por el uso de estructuras de celdas de memoria que permiten que la red "recuerde" dependencias o las relaciones entre los datos de entrada y las respuestas de salida (Olah, 2015).

Otra biblioteca utilizada es PDFDocument, de características abiertas por utilizar la licencia Apache 2.0, esta permite la creación de documentos PDF, así como su manipulación y la posibilidad de extraer su contenido y metadatos (The Apache Software Foundation, 2023).

En la Figura 2 se muestra la implementación del código que procesa un objeto de tipo Archivo y que extrae su contenido después de aplicar la red neuronal LSTM.

Figura 2

Implementación del OCR en lenguaje Java

```
1.  public String getText(File file) throws Exception {
2.      String result = "";
3.      try {
4.          PDDocument document = PDDocument.load (file);
5.          PDFRenderer pdfRenderer = new PDFRenderer(document);
6.          StringBuilder out = new StringBuilder();
7.
8.          for (int page = 0; page < document.getNumberOfPages(); page++) {
9.              BufferedImage bim = pdfRenderer.renderImageWithDPI(page, 300,
    ImageType.RGB);
10.
11.             File temp = File.createTempFile("tempfile_" + page, ".png");
12.             ImageIO.write(bim, "png", temp);
13.
14.             Tesseract tesseract = new Tesseract();
15.             tesseract.setDatapath("./src/main/resources/tessdata");
16.             tesseract.setLanguage("spa");
17.             tesseract.setPageSegMode(1);
18.             tesseract.setOcrEngineMode(1);
19.             result = tesseract.doOCR(temp);
20.             temp.delete();
21.         }
22.         document.close();
23.         return result;
24.     } catch (Exception e) {
25.         throw new Exception(e.getMessage());
26.     }
27. }
```

Nota: El código completo del servicio java puede descargarse para su consulta e implementación en: https://github.com/israelortega/Ocrpdf

https://doi.org/10.22201/dgtic.ctud.2024.2.1.37 Vol. 2, Núm. 1. enero-marzo 2024, págs. 61 - 65

2.3 SERVICIO DE OCR EN LENGUAJE PYTHON

El tercer aplicativo se desarrolla en lenguaje Python con apoyo en el marco web Flask (Pallets, 2023), entorno para el desarrollo de aplicaciones web, para el OCR se utiliza la biblioteca de código abierto EasyOCR que puede reconocer textos en una diversidad de lenguajes, con compatibilidad con formatos de imagen JPG, PNG, TIFF, además de PDF.

EasyOCR utiliza algoritmos de aprendizaje profundo reentrenados y desarrollados por el área de inteligencia artificial de Google, con una secuencia de preprocesamiento de imagen, segmentación de texto y reconocimiento de caracteres. Se espera que próximamente tenga soporte para texto escrito a mano (JAIDED AI, 2023).

En el código en lenguaje Python mostrado en la Figura 3, la entrada es un archivo PDF que recupera cada página de los documentos y reconoce las imágenes contenidas por hoja.

Figura 3

Implementación del OCR en lenguaje Python

```
1.  def analiza_pdf_temp(archivo_pdf):
2.      try:
3.          reader = easyocr.Reader(['es'], gpu=False)
4.          with tempfile.NamedTemporaryFile(suffix='.pdf', delete=False) as temp_pdf:
5.              archivo_pdf.save(temp_pdf.name)
6.              doc = fitz.open(temp_pdf.name)
7.              #Ciclo para ir de página en página
8.              pagina = []
9.              for page_num in range(len(doc)):
10.                 page = doc.load_page(page_num)
11.                 image_list = page.get_images(full=True)
12.                 resultText = ""
13.                 #Ciclo para ir de imagen en imagen de cada página
14.                 for img_index, img_info in enumerate(image_list):
15.                     xref = img_info[0]
16.                     base_image = doc.extract_image(xref)
17.                     image_data = base_image["image"]
18.                     with tempfile.NamedTemporaryFile(suffix='.png', delete=False)
    as temp_image:
19.                         with open(temp_image.name, "wb") as image_file:
20.                             image_file.write(image_data)
21.                         # Utiliza EasyOCR para reconocer el texto en la imagen
22.                         result = reader.readtext(temp_image.name)
23.                     for detection in result:
24.                         text = detection[1]
25.                         resultText = resultText + " " + text
26.                 pagina.append(resultText)
27.     except Exception as e:
28.         raise Exception(str(e))
29.     return pagina
```

Nota: Código adaptado de https://www.blackbox.ai/ con el prompt: "genera el código para leer el contenido de imagen de un PDF con Python" y modificado para el tratamiento con documentos en memoria. El código puede descargarse de: https://github.com/israelortega/ServPythonOCR

https://doi.org/10.22201/dgtic.ctud.2024.2.1.37 Vol. 2, Núm. 1. enero-marzo 2024, págs. 62 - 65

3. RESULTADOS

Al ejecutar la aplicación de lectura y transferencia con la invocación al servicio Java, se obtiene un arreglo con los tiempos de procesamiento en un arreglo de 50 elementos. Los valores indican una media de 7101 milisegundos por documento y una desviación estándar de 3,027 milisegundos. Los resultados son presentados en la Figura 4.

Figura 4

Resultados obtenidos de la llamada al servicio JAVA de procesamiento OCR de un documento PDF

```
Valores Java: [5342, 17842, 17585, 4719, 4387, 6517, 4545, 5935, 6866, 4466, 5699, 4595,
6179, 12338, 10079, 10886, 5010, 12145, 9963, 11512, 7667, 6790, 3878, 11393, 4831, 6923,
7252, 11506, 5039, 5102, 7625, 5731, 5957, 5321, 6021, 5953, 6227, 6008, 6032, 5976,
6150, 5722, 6157, 5948, 5941, 5964, 5661, 5694, 5968, 5914, 5220]

Media Java: 7101.588235294118
Desviación Estándar Java:3027.1395901146043
```

Al ejecutar el servicio Python, se encontró que la media de procesar los mismos 50 documentos fue de 9,031 milisegundos y una desviación estándar de 2,942 milisegundos. Los resultados obtenidos para la invocación al servicio Python son presentados en la Figura 5.

Figura 5

Resultados obtenidos de la llamada al servicio Python de procesamiento OCR de un documento PDF

```
Valores Python: [6246, 19980, 20097, 7018, 6449, 8744, 6392, 7948, 9194, 6304, 7561,
6698, 7744, 13500, 11853, 11798, 7881, 12492, 11817, 13009, 10102, 9208, 6182, 12594,
6068, 9528, 9237, 12284, 6697, 6406, 8974, 7775, 7783, 7631, 7845, 7538, 8150, 7907,
8508, 7484, 7906, 8201, 8474, 8416, 8100, 7846, 7769, 7919, 7673, 7828, 7051]

Media Python: 9031.26
Desviación Estándar Python:2942.285560580414
```

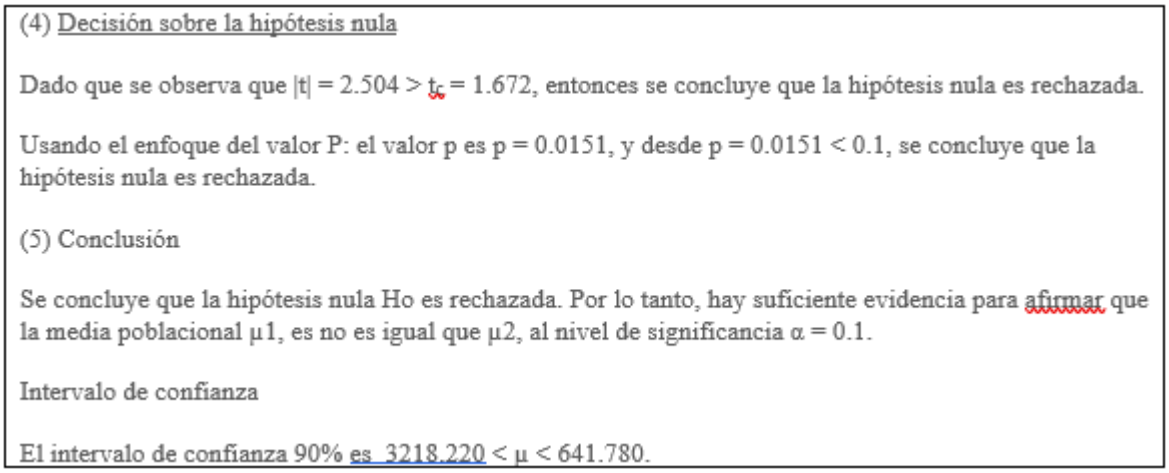
Para determinar que ambos servicios tienen diferentes tiempos de procesamiento, se utilizará una prueba estadística "t" de Student. Esta prueba estadística se puede utilizar para comparar dos secuencias de datos que validen la prueba de hipótesis nula de que las medias son iguales, y la hipótesis alternativa de que sus medias no son iguales (Lind, Marchal, y Wathen, 2019). Esto permitió determinar si el tiempo de procesamiento de cada servicio es diferente y no se debe a factores aleatorios del equipo en operación.

Para validar las hipótesis se ingresa al sitio https://mathcracker.com/es/calculadora-prueba-t-para-dos-medias#results, donde se ingresan los datos de las medias, desviaciones estándar, tamaño de muestra y nivel de significancia de 0.10. Los resultados se presentan en la Figura 6.

https://doi.org/10.22201/dgtic.ctud.2024.2.1.37 Vol. 2, Núm. 1. enero-marzo 2024, págs. 63 - 65

Figura 6

Resultados al ingresar los valores de los resultados en el sitio "Prueba t para dos medias - Desviaciones estándar de población desconocida"



(4) Decisión sobre la hipótesis nula

Dado que se observa que $|t| = 2.504 > t_c = 1.672$, entonces se concluye que la hipótesis nula es rechazada.

Usando el enfoque del valor P: el valor p es $p = 0.0151$, y desde $p = 0.0151 < 0.1$, se concluye que la hipótesis nula es rechazada.

(5) Conclusión

Se concluye que la hipótesis nula Ho es rechazada. Por lo tanto, hay suficiente evidencia para afirmar que la media poblacional $\mu 1$, es no es igual que $\mu 2$, al nivel de significancia $\alpha = 0.1$.

Intervalo de confianza

El intervalo de confianza 90% es $3218.220 < \mu < 641.780$.

Nota: Para mayor detalle consultar: https://mathcracker.com/es/calculadora-prueba-t-para-dos-medias#results

En conclusión, de esta sección, la diferencia de la duración del procesamiento entre el servicio Java y el servicio Python sí se presenta como significativa, por lo que ambos servicios realizan el procesamiento de los mismos documentos en tiempos distintos.

4. DISCUSIÓN

En cuanto a la programación de los servicios no se presentan diferencias apreciables entre los Lenguajes Java y Python; ya que el código fuente se presenta similar, cambiando únicamente las referencias a los objetos que realizan el procesamiento OCR. También hay que resaltar que los dos servicios realizan el procesamiento utilizando algoritmos distintos: Tesseract basado en redes neuronales por parte de Java, y algoritmos de aprendizaje profundo reentrenados por parte de EasyOCR en Python.

En el procesamiento con los documentos PDF, que son textos de carácter administrativo, se observa que el servicio Java tarda una media de un poco más de 7 milisegundos, mientras que un servicio Python puede tardar un poco más de 9 milisegundos con el mismo número y contenido igual de documentos. Podemos coincidir que para este ejercicio: en cuanto a procesamiento, los servicios Java y Python sí difieren en términos de velocidad, sin embargo, para un entorno donde el usuario procesa documentos individuales la diferencia de velocidad es imperceptible.

En contraste, esta prueba con documentos de diferente índole al administrativo podría resultar desigual, por lo que fuera de este ámbito la comparación podría no ser necesariamente válida. Además, no se está utilizando acceso a la potencia que puede dar una tarjeta gráfica, por lo que su ejecución también podría ser bastante desigual. En el entorno de prueba no se contó con equipos de estas características.

https://doi.org/10.22201/dgtic.ctud.2024.2.1.37 Vol. 2, Núm. 1. enero-marzo 2024, págs. 64 - 65

Respecto a la precisión en la recuperación de textos, considerando ésta como el número de caracteres correctamente reconocidos respecto al número total de caracteres, la apreciación cualitativa es que es bastante similar, lo cual puede probarse implementando minería de datos y análisis de textos con algoritmos de similitud de caracteres. Este estudio puede implementarse a futuro.

5. CONCLUSIONES

La comparación entre dos servicios web de reconocimiento óptico de caracteres (OCR) de documentos PDF implementados en los lenguajes Python y Java presentaron tiempos de procesamiento diferentes con una diferencia aproximada de 2 milisegundos por documento en el procesamiento de 50 documentos de carácter administrativo. Esta diferencia podría ser significativa en documentos de ámbito distinto o en equipos de cómputo que utilicen una tarjeta gráfica dedicada. Para el usuario final que procesa documentos individuales la diferencia podría no ser tan apreciada.

6. AGRADECIMIENTOS

Se agradece a las siguientes personas de la Dirección de Sistemas de la Dirección General de Personal de la UNAM por su apoyo técnico y revisión del presente trabajo: Oscar Álvarez Fernández, Ingrid Noemí Garrido González, Miguel Ángel Martínez Rivera, Jorge Eduardo García Santander, Margarita Velasco Perroni, Nancy Belén Ramírez Martínez y Jonathan López Suárez.

Los fragmentos de código fueron formateados en el siguiente sitio web: https://planetb.troye.io/

REFERENCIAS

Broadcom. (2023). Spring makes Java simple. Recuperado el 28 de noviembre de 2023, de https://spring.io/

Gosling, J., Joy, B., Steele, G., Bracha, G., y Buckley, A. (2015). *The Java® Language Specification Java SE 8 Edition*. Oracle America, Inc.

JAIDED AI. (2023). EasyOCR. Recuperado de https://github.com/JaidedAI/EasyOCR

Kay, A. (2007). Tesseract: an Open-Source Optical Character Recognition Engine. *Linux Journal,* (159). Recuperado de https://www.linuxjournal.com/article/9676

Klippa. (2023). Tesseract OCR: ¿Qué es y por qué lo deberías elegir en el 2023? Recuperado de https://www.klippa.com/es/blog/informativo/que-es-tesseract-ocr/

Lind, D. A., Marchal, W. G., y Wathen, S. A. (2019). *Estadística aplicada a los negocios y la economía* (19 ed.). McGraw Hill.

Llorens F. (2022). Transformación digital, ¿otro término de moda? En *Transformación digital de las universidades: hacia un futuro postpandemia / coord. por Faraón Llorens Largo, Rafael López Meseguer* (pp. 18–26).

https://doi.org/10.22201/dgtic.ctud.2024.2.1.37 Vol. 2, Núm. 1. enero-marzo 2024, págs. 65 - 65

Martelli, A., Ravenscroft, A., y Holden, S. (2017). *Python in a Nutshell, 3rd Edition*. O'Reilly Media, Inc.

Olah, C. (2015). Understanding LSTM Networks. Recuperado de colah's blog website: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Oracle Inc. (2023). Java. Recuperado el 28 de noviembre de 2023, de https://www.oracle.com/java/

Pallets. (2023). Flask. Recuperado el 30 de noviembre de 2023, de https://flask.palletsprojects.com/en/3.0.x/

Tess4J. (2022). Tess4J. Recuperado el 29 de noviembre de 2023, de https://tess4j.sourceforge.net/

The Apache Software Foundation. (2023). Apache PDFBox® - A Java PDF Library. Recuperado el 30 de noviembre de 2023, de https://pdfbox.apache.org/

Wexelblat, R. L. (1981). *History of Programming Languages*. Academic Press.