EVALUACIÓN DE RENDIMIENTO EN TÉCNICAS DE TRANSCRIPCIÓN DE TEXTO MANUSCRITO EN IMÁGENES DE DOCUMENTOS HISTÓRICOS CON CONTENIDO ECONÓMICO DEL SIGLO XVIII SIN TRATAMIENTO PREVIO

PERFORMANCE EVALUATION OF HANDWRITTEN TEXT TRANSCRIPTION TECHNIQUES ON IMAGES OF HISTORICAL DOCUMENTS WITH ECONOMIC CONTENT FROM THE 18TH CENTURY WITHOUT PRIOR TREATMENT

TRABAJO FIN DE MÁSTER
CURSO 2024-2025

AUTOR
RICARDO ENRIQUE PALOMARES MARTÍNEZ

DIRECTORES
FRANCISCO CEBREIRO ARES
ANTONIO SARASA CABEZUELO

MÁSTER EN INGENIERÍA INFORMÁTICA
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

EVALUACIÓN DE RENDIMIENTO EN TÉCNICAS DE TRANSCRIPCIÓN DE TEXTO MANUSCRITO EN IMÁGENES DE DOCUMENTOS HISTÓRICOS CON CONTENIDO ECONÓMICO DEL SIGLO XVIII SIN TRATAMIENTO PREVIO

PERFORMANCE EVALUATION OF HANDWRITTEN TEXT TRANSCRIPTION TECHNIQUES ON IMAGES OF HISTORICAL DOCUMENTS WITH ECONOMIC CONTENT FROM THE 18TH CENTURY WITHOUT PRIOR TREATMENT

TRABAJO DE FIN DE MÁSTER EN INGENIERÍA INFORMÁTICA
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

AUTOR
RICARDO ENRIQUE PALOMARES MARTÍNEZ

DIRECTORES
FRANCISCO CEBREIRO ARES
ANTONIO SARASA CABEZUELO

CONVOCATORIA: JUNIO 2025
CALIFICACIÓN: 8,5

MÁSTER EN INGENIERÍA INFORMÁTICA
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

8 DE JULIO DE 2025

DEDICATORIA

A todos quienes me han soportado durante el tiempo que he cursado este máster, desde mis parientes y compañeros de trabajo que fingían entender y estar interesados en lo que les contaba, hasta mis compañeros de clase que me trataron como un igual y no como su padre.

AGRADECIMIENTOS

Gracias a mis tutores, Antonio y Fran, por su ayuda, paciencia y comprensión todo este tiempo. Y gracias a todos los profesores que me han impartido clase en el máster, entre los que he encontrado verdaderos maestros en el arte de transmitir ideas.

RESUMEN

Evaluación de rendimiento en técnicas de transcripción de texto manuscrito en imágenes de documentos históricos con contenido económico del siglo XVIII sin tratamiento previo

La visión por computadora (CV por sus siglas en inglés) es una disciplina amplia que incluye el procesamiento de imágenes de forma que permita la extracción de sus características, de manera similar a como lo haría una persona, aunque utilizando técnicas algo diferentes. El reconocimiento de texto manuscrito (HTR por sus siglas en inglés) puede ser considerado una especialización de la CV orientada a reconocer texto escrito a mano en imágenes y convertirlo a un formato que pueda ser manipulado por las máquinas. Ambas disciplinas tienen un relativamente largo historial de investigaciones y progreso a día de hoy, remontándose incluso al siglo XIX en el caso de HTR, si bien no hay duda de que ambos se han beneficiado enormemente de las mejoras en el ámbito de la inteligencia artificial (AI, por sus siglas en inglés), especialmente el área de las redes neuronales. Este trabajo examina el escenario de HTR aplicado a un caso práctico de transcripción de documentos históricos manuscritos de finales del siglo XVIII representado en forma de imágenes en color JPEG, y compara una aproximación usando una aplicación de código abierto específicamente a historiadores con un conjunto de scripts Python adaptados a la tarea usando la biblioteca AI Tensorflow.

Palabras clave

Computer Vision, CV, Handwritten Text Recognition, HTR, CNN, RNN, CRNN, Tensorflow, Python, Spanish

ABSTRACT

Performance evaluation of handwritten text transcription techniques in images of historical documents with economic content from the 18th century without prior treatment

Computer Vision (CV) is a broad discipline that involves processing images in a way that enables the extraction of features of them, very much like a human could do, although it uses slightly different approaches. Handwritten Text Recognition (HTR) can be considered an specialization of CV aimed at recognizing text written by hand in images and converting it to a format that can be manipulated easily by machines. Both disciplines have a somehow long record of studies and progress as of today, going back even to 19th century in the case of HTR, albeit there is no doubt that both have greatly benefit of improvements in the Artificial Intelligence (AI) discipline, especially in the Neural Networks area. This work examines the current landscape of HTR applied to a practical case of transcribing of Spanish historical handwritten documents by the end of 18th century in the form of color JPEG images, and compares an approach using an open source application specifically targeted to History researchers and a tailored set ot Python scripts using Tensorflow AI library.

Keywords

Computer Vision, CV, Handwritten Text Recognition, HTR, CNN, RNN, CRNN, Tensorflow, Python, Spanish

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1 - Introducción

1.1 Motivación

El procesamiento de datos de forma digital es esencial hoy en día. Si bien los datos generados en los últimos 30-40 años normalmente se crean de forma digital nativamente, anteriormente los datos eran a menudo analógicos [1] y debían ser convertidos en un formato digital. Un ejemplo común es la conversión de texto disponible solamente en papel (ya sean periódicos, libros, informes impresos o cartas manuscritas). Surgen varios desafíos durante este proceso de transformación, desde capturar la representación visual a una imagen digital, a reconocer los símbolos contenidos en ella.

La invención de los escáneres a mediados de la década de 1950 [2] y, posteriormente, la fotografía digital marcó el primer paso, permitiendo a los ordenadores acceder a imágenes como una secuencia de bytes. Quizá más sorprendente resulta que la segunda parte, reconocer texto en papel, es una tarea que comenzó muy al principio del siglo XX [3]. Este esfuerzo inicial no era parte de ningún proceso de digitalización, sino que estaba orientado a ayudar a las personas con dificultades de visión. Los primeros avances en esta área adoptaron la forma de dispositivos mecánicos [4] y estaban enfocados sobre todo al texto impreso. Importantes mejoras tuvieron lugar posteriormente mediante el uso de técnicas de programación convencionales, desembocando en el desarrollo del reconocimiento óptico de caracteres (OCR, por sus siglas en inglés) y el reconocimiento de texto manuscrito (HTR, por sus siglas en inglés) durante las décadas de 1960 y 1970. Sin embargo, la programación tradicional basada en reglas y en heurística demostró tener grandes limitaciones.

El cambio de paradigma en las décadas de 1980 y 1990, de programación basada en reglas al aprendizaje automático, permitió mejoras significativas en HTR [5] [6]. Los avances en las redes neuronales permitió la adopción de redes neuronales convolucionales (CNN, por sus siglas en inglés) y redes neuronales recurrentes (RNN,

por sus siglas en inglés) [7] y, más recientemente, los Transformers también se han demostrado efectivos para estas tareas.

Reconocer texto presente en imágenes es útil actualmente en un amplio espectro de actividades que van más alla del ámbito de asistencia para superar dificultades visuales. Desde identificar vehículos leyendo su placa de matrícula, a traducir en tiempo real textos simplemente dirigiendo la cámara del móvil a un papel o a transcribir textos de documentos históricos a archivos electrónicos para ayudar a los investigadores en Historia a acceder a sus contenidos sin poner en riesgo la integridad de los documentos en papel con varios siglos de antigüedad.

Este trabajo de fin de máster examina alternativas para llevar a cabo HTR en un conjunto de casi 400 imágenes que contienen documentos históricos escritos en español a finales del siglo XVIII. Los documentos, que suelen estar formados por dos o más páginas (y, por tanto, imágenes), son actas notariales correspondientes a protestos de letras de cambio. El siguiente resumen condensado se ha extraído de [8].

En términos sencillos, estos documentos eran reclamaciones de promesas de pago no cumplidas entre varios actores. Si bien en su forma más básica intervenían un deudor y un acreedor, radicados en ciudades distintas, que hacían negocios entre ellos y utilizaban la letra de cambio como mecanismo para evitar el intercambio constante de dinero que debía ser transportado, cuando su utilización se generalizó y se convirtieron en un instrumento financiero considerado confiable, comenzó a ser común que aparecieran otros actores que encargaban a los actores originales hacer pagos en nombre de esos actores, o bien que aceptaban la letra como un valor equivalente al dinero, dando lugar a endosos.

Los endosos permitían al acreedor ceder su derecho de cobro a un tercero (por tanto, tratando la letra como un instrumento de pago equivalente al dinero), quien a su vez podía endosar de nuevo la letra, generando una cadena de promesas de pago y cobro. Cuando alguno de los eslabones de la cadena incumplía su promesa, se generaba un impago y, si no era posible resolverlo en un plazo determinado, se acudía al escribano de la ciudad, cabildo o plaza para proceder a su reclamación, más todos

los gastos derivados de la misma. La reclamación toma el nombre de protesto de letra de cambio.

El valor histórico y económico de los protestos de letras de cambio es que transcribían íntegramente el contenido, tanto de la letra de cambio original, como de los sucesivos endosos, incluyendo fechas, cantidades e intervinientes. También documentaban el resultado del protesto y, en su caso, las razones por las que el sujeto al que se reclamaba rechazaba atender el pago. Todo ello traducido, en este caso, al castellano, ya que las letras de cambio abarcaban ciudades en diferentes países.

Por todo ello, los protestos de letras de cambio son una fuente muy importante de información histórica y económica; información, por otra parte, que no está estructurada, ya que el acta notarial no se presenta de una forma tabulada o sintetizada, sino que, como todos los documentos notariales, es una sucesión de párrafos relatados por el escribano entre los que se distribuye la citada información.

1.2 Objetivos

Este trabajo de fin de máster trata de proporcionar soluciones a las necesidades de un proyecto de investigación sobre las crisis financieras a lo largo de la historia. La tarea concreta consiste en extraer algunos datos de letras de cambio fechadas entre 1790 y 1800, todas ellas redactadas a mano por escribanos. Para ello, en primer lugar debe transcribirse el texto completo a texto sin formato y luego enviarlo a otra herramienta que extraiga los datos relevantes para el proyecto de investigación. Este trabajo de fin de máster se centra en la primera parte, el proceso de HTR y la transcripción a texto sin formato.

1.3 Plan de trabajo

Para alcanzar el objetivo de proporcionar alternativas de transcripción de documentos manuscritos en forma de imágenes a texto, se ha propuesto el siguiente plan:

- Analizar el estado de la cuestión en HTR.

- Evaluar las herramientas existentes para la tarea anteriormente descrita y su rendimiento con el conjunto de documentos concreto que se quiere procesar.
- Desarrollar un conjunto específico de scripts Python para procesar en lote las imágenes, comprendiendo desde la normalización y saneado de las mismas hasta el entrenamiento de la red neuronal para efectuar el reconocimiento.
- Comparar el rendimiento de las alternativas mediante las métricas de CER y WER (descritas en el capítulo 2) y hacer una sugerencia sobre qué solución puede funcionar mejor para el conjunto de imágenes de documentos históricos concreto.

Capítulo 2 - Estado de la cuestión

2.1 Organización de este capítulo

En este capítulo se detalla el estado de la cuestión en lo que respecta a HTR. Se comienza con un breve resumen de los antecedentes históricos previos al uso de redes neuronales, para luego entrar en mayor detalle sobre estas y los últimos avances. Por su importancia, se dedica un apartado a presentar las técnicas de preprocesamiento de imágenes, que pueden afectar al rendimiento de HTR, cualquiera que sea la técnica elegida para ello. Seguidamente, se mencionan distintos artículos científicos que, como este trabajo, están especialmente dirigidos al proceso de HTR en documentos históricos. Los dos últimos apartados detallan, por un lado, las métricas usadas habitualmente para comparar el rendimiento de diferentes arquitecturas de redes neuronales para realizar HTR, y algunas aplicaciones de código abierto en el mercado que han sido mencionadas como sugerencias para realizar HTR.

Durante la búsqueda de referencias bibliográficas para este capítulo se ha encontrado el artículo académico *Handwritten Text Recognition: A Survey* [9]. Siendo un artículo que recoge los avances en esta materia a lo largo del tiempo, la situación actual y las tendencias en la materia, inevitablemente se observará un cierto solapamiento entre lo expuesto en este capítulo y en dicho artículo. No obstante, se tratará de sintetizar las partes más relevantes del mismo, remitiendo a dicho artículo para profundizar en el contenido original del mismo.

2.2 Antecedentes

Resulta intuitivo comprender que el reconocimiento por una computadora de texto escrito de puño y letra debe presentar un mayor desafío que el de texto impreso. En efecto, puede que existan miles de tipografías diferentes con las que imprimir y que cada una tenga unas características propias que la diferencian del resto, pero todos asumimos que, para una misma tipografía y estilo (regular, negrita, cursiva), cada vez que aparezca la letra “a” va a tener exactamente el mismo aspecto. Quizá el tamaño sea diferente, quizá una hoja fue alimentada por la impresora con una ligera

inclinación, pero una vez aplicadas las correcciones de imagen previas, el mismo carácter impreso con la misma tipografía va a ser sustancialmente igual. Llevar a cabo reconocimiento de caracteres impresos, o reconocimiento óptico de caracteres (OCR, por sus siglas en inglés) cuenta, al menos, con un cierto factor determinista e incluso puede sacar partido de la estandarización de formatos de definición de tipografías, como TrueType.

Claramente, eso no sucede con el texto manuscrito. Incluso en la misma hoja y del mismo puño y letra, el mismo carácter presentará mayores diferencias. El texto manuscrito de un mismo autor puede variar según sus emociones (algo que ha sido objeto de análisis mediante aprendizaje automático [10]). Por tanto, a los desafíos que puede afrontar un proceso OCR, que indudablemente los tiene, el HTR suma una variabilidad intrínseca en los datos recibidos con la que hay que lidiar.

Los primeros avances en HTR tuvieron lugar mediante la aplicación de reglas y aplicación de procesos estocásticos. En algunos casos se apoyaban en el uso de un léxico predefinido [11] y pronto se generalizó el uso de las cadenas ocultas de Markov (o modelos ocultos de Markov, HMM, por sus siglas en inglés) [12], [13], [14], que ya se habían usado previamente para el reconocimiento de voz.

El artículo *A tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*, de L. R. Rabiner [15] contiene una detallada introducción a las cadenas ocultas de Markov. Las cadenas de Markov permiten modelar un proceso estocástico (probabilístico) a partir de la determinación de los estados en los que puede encontrarse un modelo, las transiciones posibles entre ellos y la probabilidad de cada una de estas. Cuando todos estos parámetros son observables, el modelo es discreto.

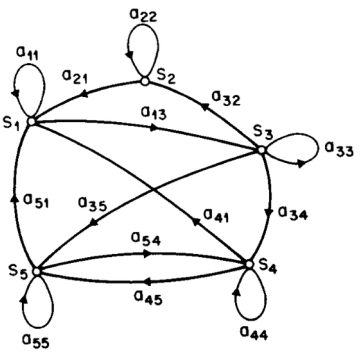

Figura 1: Representación de cadena de Markov (imagen de A tutorial on Hidden Markov Models and Selected Applications in Speech Recognition)

Sin embargo, a menudo en escenarios reales solo se dispone de una secuencia de resultados (o señales), desconociendo a priori los estados que los han originado, las posibles transiciones entre ellos y la probabilidad de las mismas, en cuyo caso se habla de modelos (o cadenas) ocultos de Markov. La secuencia de resultados o señales puede

ser un flujo de audio y mediante HMM se intenta establecer la concordancia óptima del flujo con una sucesión de estados que se identifican, a su vez, con la representación como onda de audio de fonemas, letras o palabras. Si cambiamos el flujo de audio por la matriz de bytes en la que se guardan los puntos que componen una imagen, la misma metodología puede aplicarse para reconocer texto.

Que en un HMM los estados, las transiciones y sus probabilidades no sean observables no significa que no existan. Sin embargo, puesto que no se tiene su especificación cierta, solo pueden establecerse hipótesis sobre ellas a partir de la observación de los resultados. Si se define un modelo hipotético a partir de la observación de una secuencia de resultados, luego puede aplicarse ese modelo a otra secuencia distinta para ver cómo de bien encaja en los estados inferidos. Es posible definir múltiples modelos y compararlos para ver cuál encaja mejor. Con las suficientes muestras etiquetadas (flujo de audio y su transcripción manual, imagen de un texto manuscrito y su transcripción manual) se puede llegar a generalizar un modelo basado en la transición de estados.

El problema de esta solución es que no escala bien cuando hay mucha variabilidad, como es el caso del texto manuscrito. En [12] el léxico era de solo 32 elementos, mientras que en [13] se advierte de los altos requisitos de memoria al crecer el número de caracteres diferentes y sus modelos asociados.

Esta explicación, muy superficial y sin el debido soporte matemático de HMM, pretende dar solo una idea general, dado que no se ha usado HMM para este trabajo de fin de máster. El lector interesado puede encontrar información más completa y rigurosa en el mencionado artículo de Rabiner [15].

2.3 Redes neuronales convolucionales (CNN)

El artículo *Gradient-Based Learning Applied to Document Recognition*, de Lecun, Y., Bottou, L., Bengio, Y. Y Haffner, P. [7] describe detalladamente las ventajas del uso de redes neuronales convolucionales (CNN, por sus siglas en inglés), comenzando por describir los algoritmos de aprendizaje basado en gradientes y adentrándose en la

arquitectura de las CNN. Aquí se hará un resumen de los diferentes conceptos presentados por el artículo.

El proceso de HTR se encuadra, como problema que resolver, dentro la definición más general del reconocimiento de patrones. Esta categoría de problemas consta de dos fases (o módulos, en la terminología usada por el artículo): extracción de características y clasificación entrenable.

La extracción de características consiste en identificar, en las secuencias de entrada, las propiedades relevantes y trasladarlas a vectores o cadenas más compactos que la secuencia original. Esas características pueden ser, para un flujo de audio, el tono o el ritmo, mientras que para el reconocimiento de texto pueden ser bordes, ascendentes (la línea que sube en una b, una d o una h) o descendentes (la que baja en una p o una q), etc. Este primer módulo, cuando no se usan redes neuronales, requiere una construcción manual para aportar al sistema el conocimiento que permite identificar audio o texto determinados.

Una vez codificadas en el vector las características encontradas en la secuencia de entrada, la clasificación entrenable debe ser capaz de examinar los vectores y las características que contienen y relacionarlas con los posibles resultados que se quieren encontrar en la secuencia (¿hay un ritmo que indica que el audio es una música? ¿el tono es propio de la voz de una mujer o de un hombre? ¿un ascendente seguido de un círculo puede que corresponda a una letra b?). Este segundo módulo puede ser más genérico y se le puede entrenar sin tener que rehacerlo para cada tarea.

Las redes neuronales multicapa (Multilayer Neural Networks) permiten sustituir el primer módulo de extracción de características, que se realiza a mano y para cada tarea, por un mecanismo más general apoyado en la existencia de:

- Conjuntos de datos de entrenamiento que relacionan datos de entrada con los de salida que se desea que proporcione el sistema para los primeros.
- Algoritmos de aprendizaje basados en gradientes que tratan de minimizar la diferencia entre los resultados esperados y los obtenidos, ajustando coeficientes (parámetros o pesos) que se usan junto con los datos de cada entrada tras calcular

dicha diferencia mediante una función que computa la pérdida media a través de todo el conjunto de datos de entrenamiento.

- Retropropagación (Back-propagation), un algoritmo para calcular el gradiente, que como se deduce por su nombre supone recorrer la red hacia atrás, que se reveló como simple y eficiente en un capítulo del libro *Parallel Distributed Processing* de Rumelhart et al. [16] y que se ha generalizado en la mayoría de las redes neuronales.
- Suficiente potencia de cálculo para llevar a cabo el entrenamiento, dado que se requieren varias iteraciones para ir refinando los valores de los coeficientes, cada una de ellas utilizando el conjunto completo de datos.

Las redes convolucionales son un tipo de redes neuronales multicapa que extraen patrones comunes de formas multidimensionales (siendo las más habituales de 1, 2 o 3 dimensiones) a través de patrones de conexión por cercanía (o locales) y restricciones en los pesos. Como todas las redes neuronales, utilizan aprendizaje basado en gradientes y minimización de la función de pérdida.

En el apartado II.A del artículo de Lecun, Y., Bottou, L., Bengio, Y. Y Haffner, P. [7] se identifican tres propiedades de las CNN que las hacen especialmente aptas para el propósito de reconocimiento de imágenes:

1. Campos receptivos locales.
2. Replicación de pesos, o pesos compartidos.
3. Submuestreo espacial o temporal.

Estas tres propiedades permiten que las CNN no se vean tan afectadas como otras arquitecturas de redes neuronales por cuestiones como el desplazamiento, la escala o la distorsión de la imagen de entrada de la que se quiere obtener un resultado en comparación con las imágenes utilizadas para entrenar la red.

Los campos receptivos locales consisten en analizar la entrada no por sus píxeles individuales, sino tomando un subconjunto que va recorriendo la imagen a modo de ventana de desplazamiento. En la Figura 2 se ilustra una imagen de 32x32 píxeles que compondría la entrada a la primera capa de una red convolucional con 8 mapas de características. Los recuadros rojo, azul y amarillo corresponderían al

subconjunto a medida que se desplaza por la imagen y, como puede verse, el desplazamiento permite que se solape la región observada con la iteración anterior del desplazamiento. Ese subconjunto recibe el nombre de kernel suele tener unas dimensiones de 3x3 o 5x5 píxeles.

En cada iteración se aplica, para cada capa, una operación entre la matriz recuperada por el kernel y un filtro de la misma dimensión que el kernel. Si la dimensión de la entrada es de 32x32 y el kernel es de 5x5, en cada fila habrá (32 -5 + 1= 28 iteraciones de desplazamiento del kernel, por lo que esa será la dimensión de la anchura de la capa. Lo mismo sucederá con la altura que, en este caso, también será de 28.

La replicación de pesos significa que, en cada mapa de características, el conjunto de pesos es idéntico, lo que hace más inmune la red a variaciones de la imagen; accesoriamente, también reduce el número de parámetros entrenables.

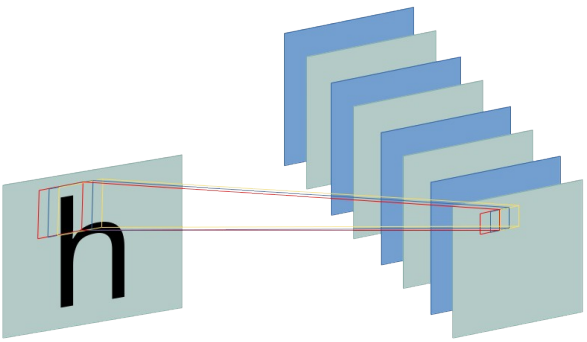

Figura 2: Campos receptivos locales en una CNN

A una capa convolucional sigue normalmente una capa de submuestreo, que toma subconjuntos de la salida de la capa convolucional (por ejemplo, de 2x2 elementos) y los *comprime* en uno mediante una operación matemática que puede ser la media de sus valores, el valor máximo, etc. En este caso no se produce solapamiento entre los subconjuntos, por lo que el tamaño de la salida de la capa de submuestreo se reduce por la dimensión del subconjunto; si este es de 2 x 2 = 4, la salida será cuatro veces más pequeña, al reducir tanto el ancho como el alto a la mitad.

La combinación de una capa convolucional y otra de submuestreo puede repetirse varias veces, dependiendo de la complejidad de la tarea de reconocimiento y las dimensiones de entrada de la imagen. Llegado un punto, en una CNN se introduce

una capa unidimensional totalmente conectada a la capa anterior y se reducen las características hasta un tamaño que se considera apropiado para calcular las probabilidades de las diferentes salidas posibles. En LeNet-5, una de las primeras soluciones HTR basada en CNN desplegada en producción [7], que identificaba dígitos escritos a mano, se optó por una dimensionalidad para la capa conectada de 84 parámetros de salida, resultado de multiplicar 7x12, que eran las dimensiones (ancho por alto) consideradas apropiadas para representar los dígitos como imágenes simples, pero reconocibles. Actualmente, como se describirá más adelante y al combinarse con otros tipos de capas, en particular CTC, es más frecuente ajustar ese número a la cantidad de símbolos distintos que se quieren identificar, incrementado en 1 (o, preferentemente, en 2). La figura 3, inspirada en la que representa LeNet-5 en [7], muestra una arquitectura simple de una CNN con dos capas convolucionales seguidas por sendas capas de submuestreo y una capa totalmente conectada que proporciona los resultados.

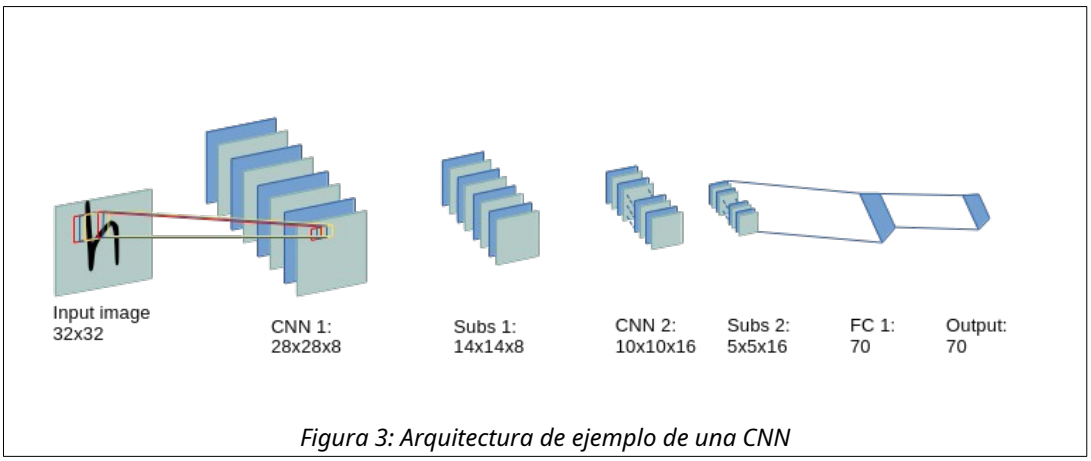

Figura 3: Arquitectura de ejemplo de una CNN

Resumiendo, las CNN supusieron un avance porque ofrecieron soluciones a varios problemas:

- Funcionan mejor que otras arquitecturas con tamaños de entrada muy grandes, como son las imágenes, tanto por la técnica de convolución como por el submuestreo.
- Las redes totalmente conectadas no incorporan invarianza para lidiar con distorsiones, traslaciones, etc., mientras que las CNN sí, gracias a la compartición de pesos en cada unidad de la capa (o mapa de características).

2.4 CTC y la arquitectura CRNN predominante actualmente

Adicionalmente a las CNN, surgieron como alternativa a los HMM las redes neuronales recurrentes, RNN. Las RNN se habían usado en arquitecturas híbridas con HMM hasta que se introdujo el uso de CTC, descrito, entre otros, por Graves, Fernández, Gómez y Schmidhuber [17]. CTC (Connectionist Temporal Classification) es descrito en ese artículo como un tipo de red, aunque actualmente está clasificada en las bibliotecas de aprendizaje automático y, en general, como una función de pérdida. CTC espera que la capa de salida devuelva el resultado de una función softmax, es decir, un vector de probabilidades de las diferentes posibles salidas, incrementado en un elemento más[1]. Si, simplificando, tenemos un alfabeto de 26 caracteres o símbolos, la salida para cada secuencia temporal de la entrada procesada por la red será un vector con la probabilidad de cada uno de los 26 símbolos más un elemento adicional en el que se indicará la probabilidad de que el símbolo encontrado no se encuentre en el vocabulario.

La expresión *secuencia temporal* se refiere a que las capas RNN van troceando y procesando la imagen en fracciones de izquierda a derecha (para un sistema de escritura que siga ese sentido). Las capas RNN no tienen, realmente, ninguna información que les permita distinguir, en una imagen que contenga la palabra “sillón”, si la fracción procesada de esa imagen contiene la letra “s” entera y aislada, solo la mitad izquierda de esa letra, o la sílaba “si” completa. Esto también da lugar a que no se pueda garantizar que el digrafo “ll” se vaya a recibir en una sola fracción o en varias. Así que, al terminar de procesar todas las secuencias temporales (o fracciones) en que se ha dividido la imagen de entrada, las capas RNN podrían identificar “s-i-l-l-ó-n”, o “ss-ii-ll-ó-ó--n”, o “s-il-ll-ó-nn”, dependiendo de cómo se distribuya la escritura manuscrita a lo largo de la imagen y de cómo esté entrenada la red. El guion representa un símbolo no encontrado.

[1] En realidad, para reconocer líneas, no solo palabras, hay que añadir dos elementos, ya que el espacio en blanco también debe ser tenido en cuenta, o bien incluirlo como posible salida.

Afortunadamente, una de las características de CTC es que es capaz de eliminar secuencias duplicadas, por lo que si recibe una secuencia como *e-xxp-e-rr-tt-o*, siendo cada guion un símbolo no encontrado (y hay que recordar en este punto que el espacio puede ser un símbolo del vocabulario si queremos reconocer frases, no palabras sueltas), es capaz de eliminar los caracteres duplicados y las salidas sin etiquetar para devolver como resultado *experto*. Con "ss-ii-ll-ó-ó--n" se eliminarían los duplicados no separados por un guion para dejar "silóón" y con "s-il-ll-ó-nn" quedaría "sillón". ¿Cómo se maneja el segundo caso, que es incorrecto? Volveremos sobre este punto algo más adelante.

La imagen de la figura 4, tomada del mencionado artículo de Graves et al., ilustra el funcionamiento con una imagen que representa la palabra *cat*. Los círculos blancos son espacios o ausencia de carácter identificado, los negros son caracteres identificados. El algoritmo analiza hacia adelante y hacia atrás, en un instante T, las probabilidades de las posibles secuencias, favoreciendo las que resultan óptimas (más probables).

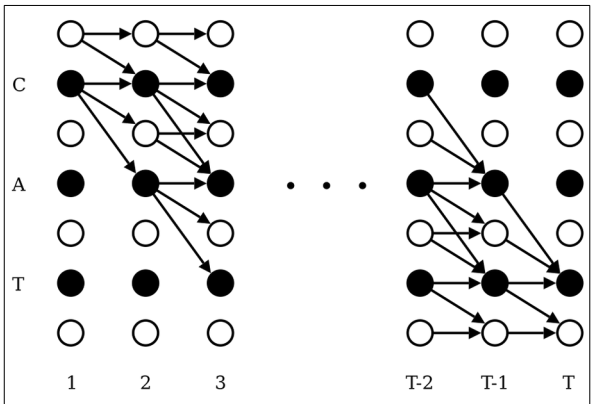

Figura 4: Funcionamiento de CTC con la palabra CAT (imagen tomada de Connectionist Temporal Classification...)

En el lado izquierdo de la imagen, para el instante 1, son secuencias posibles un espacio (que vamos a representar para este ejemplo como "_") o una letra c. En el instante 2 a _ le puede seguir otro _ o una letra c, mientras que a la letra c le puede seguir otra c, _ o a. Finalmente, podemos acabar con diversas secuencias, desde _ _ _, hasta la palabra cat sin espacios entre medias, pasando por ccc, c_a, ca_, etc. Este lado se lee siguiendo la dirección de las líneas. La repetición de símbolos o los espacios es posible por la variabilidad de la escritura manuscrita y porque, especialmente en redes RNN, no hay una segmentación explícita de los símbolos (el sistema no sabe dónde termina un carácter manuscrito y dónde comienza el siguiente, por lo que es tarea de la red aprenderlo).

En el lado derecho de la imagen, que se interpreta siguiendo las flechas en sentido contrario al que marcan, para el instante T (cuando ya se ha alcanzado el final de la imagen), son secuencias posibles _ o t. En el instante anterior T – 1, _ podría estar precedido de t o de _, mientras que t podría estar precedida de _ o de a. Como antes, al retroceder al instante T – 2 podemos obtener, con distintas probabilidades, desde la palabra cat hasta diversas combinaciones de espacios y letras. El algoritmo CTC favorecerá las secuencias que, en ambos sentidos, resulten más probables, lo que le permitirá descartar símbolos o espacios duplicados. Y las secuencias serán más o menos probables porque, durante el entrenamiento, se comparan con la etiqueta de la imagen. De esta forma, si la red devuelve “silóón”, CTC devolverá un valor de pérdida que penalizará el resultado devuelto, lo que provocará que en la siguiente época de entrenamiento la red ajuste su salida para acercarse más a “sillón”.

La arquitectura completamente RNN, no obstante sus buenos resultados para HTR, tiene el inconveniente de requerir mucho tiempo de procesamiento. Si la entrada es una imagen con dos dimensiones (ancho y alto), eran necesarias capas LSTM (Long Short Term Memory) multidimensionales que, apiladas sucesivamente, terminaban convirtiendo las salidas 2D en salidas unidimensionales (vectores) en las que poder aplicar CTC para obtener la salida comentada (un vector de probabilidades de cada posible símbolo).

En el artículo *Are Multidimensional Recurrent Layers Really Necessary for Handwritten Text Recognition?*, de Puigcerver, J. [18] se propone como alternativa utilizar capas convolucionales para extraer las características de la imagen primero, una capa que aplana las dos dimensiones de las capas convolucionales en una sola dimensión, y después capas LSTM bidireccionales (BLSTM) unidimensionales para establecer la relación entre las características resultantes y el vector de resultados. Las capas convolucionales codifican la imagen en características y las capas recurrentes las decodifican en los resultados probables. La imagen siguiente muestra la arquitectura propuesta.

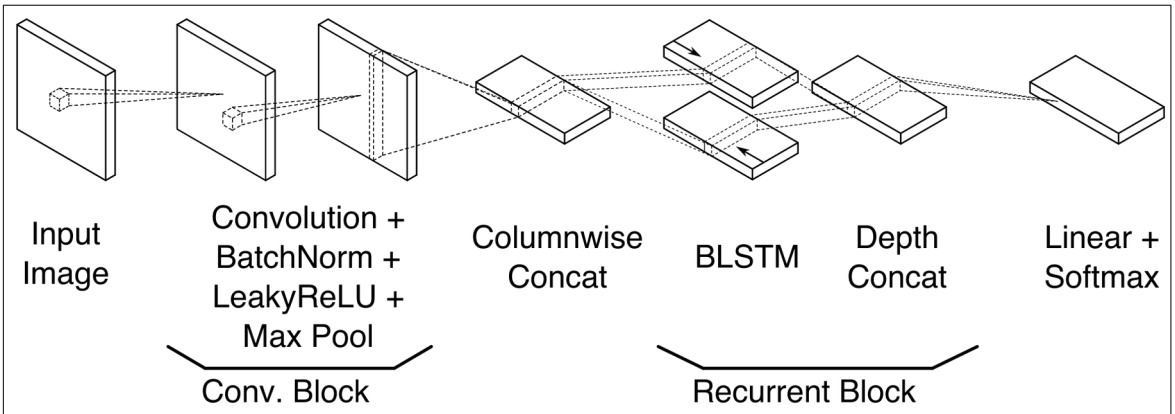

Figura 5: Arquitectura CRNN propuesta por Puigcerver (imagen de Are Multidimensional Recurrent Layers Really Necessary…

Un detalle importante para las implementaciones es que cada capa convolucional (o cada *bloque convolucional*, según la figura 5) va reduciendo las dimensiones de la imagen, pero podemos considerar que sigue siendo una imagen, en tanto que es una estructura con una altura y una anchura.

La capa que intermedia entre las CNN y las RNN (*Columnwise Concat*) transforma las columnas (la anchura) de la imagen en secuencias temporales, mientras que la altura se combina con las características que han ido descubriendo las capas CNN. Si la salida de la última capa convolucional es (ph, pw, f), donde ph es la altura condensada, pw es la anchura condensada y f es el número de características encontrado, la capa intermedia, o de transposición, debe proporcionar una estructura del tipo (pw, ph * f), donde pw se ha convertido en la lista de secuencias temporales que se quieren contemplar y ph * f la información de características que considerará la primera capa RNN para cada secuencia temporal.

Y es importante que el número de secuencias temporales pw corresponda a la longitud máxima de las posibles cadenas que se quieren identificar, ya que la salida final de la red CRNN ha de ser una matriz con ese número de secuencias temporales y las probabilidades, para cada secuencia temporal, de cada uno de los símbolos que forman el vocabulario con el que se entrena la red.

Cuando las imágenes son grandes, o contienen mucha información que se quiere extraer, puede considerarse que es necesario añadir más bloques convolucionales, pero en el artículo *Deep Residual Learning Image Recognition* [19] se

demuestra que añadir muchos bloques empeora el resultado tanto del entrenamiento como de las pruebas. En su lugar proponen *residual blocks*, que consisten en varias capas convolucionales (dos o tres, normalmente) con funciones de activación ReLU que culminan en la adición entre los datos de entrada de la primera capa y la salida de la última (para lo cual la dimensión debe mantenerse, existiendo varias técnicas para ello).

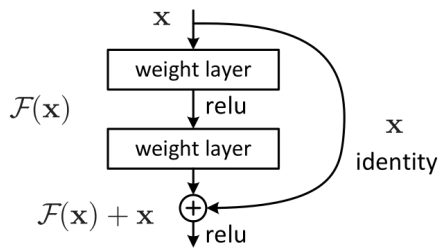

Figura 6: Residual block (imagen de Deep Residual Learning Image Recognition)

La idea es que la sucesión de capas convolucionales crean una correspondencia entre los datos de entrada que recibe la primera de ellas y las características extraídas por todas esas capas. El problema cuando hay demasiadas es que el entrenamiento da lugar a un empeoramiento de la precisión de la red neuronal y añadir más capas solo lo empeora. Como se ve en la figura 6, extraída del citado artículo, los datos de entrada originales del bloque se arrastran a la salida intactos para que la función de activación de la última capa del bloque no dependa solo del resultado de las capas, sino también de los propios datos de entrada.

Aunque el artículo utiliza datasets de clasificación de imágenes, la propuesta puede utilizarse también para otras tareas que involucran imágenes, como OCR y HTR y se usará en una de las arquitecturas probadas en este trabajo.

En general, las arquitecturas descritas hasta este punto están pensadas para el reconocimiento de texto en imágenes que contienen caracteres, palabras o líneas. Las imágenes que contienen párrafos o páginas completas deben ser segmentadas (particionadas) antes de ser usadas para entrenar el sistema.

2.5 Otros avances actuales en HTR

En los últimos tres años se han comenzado a explorar otras arquitecturas para HTR. Por un lado, se han implementado arquitecturas *Seq2Seq* que tratan de relacionar una secuencia de entrada con una secuencia de salida (lo que, durante el entrenamiento, consiste en codificar la imagen de entrada y la etiqueta que debe

obtenerse como salida) usando una arquitectura de Transformers combinada con unas capas convolucionales del tipo ResNet para la extracción de características [20].

Así mismo, se han implementado soluciones de reconocimiento de texto que tratan de operar sobre imágenes de párrafos o páginas completas. Algunos intentos han sido ejecutados usando MDLSTM (redes LSTM multidimensionales) [21], pero los mejores resultados se están consiguiendo usando también Transformers [22] [23].

No obstante, el uso de Transformers requiere importantes cantidades de datos de entrenamiento (y el correspondiente tiempo de proceso), por lo que en este trabajo, que se fijó como un requisito utilizar hardware de consumo en lugar de tiempo de computación en la nube, no se ha experimentado con ellos.

2.6 Preprocesamiento de imágenes

Un aspecto importante en el ámbito de OCR y HTR es la calidad de las imágenes, entendiendo como *calidad* que faciliten su procesamiento para la tarea pretendida. En la mayoría de los artículos científicos citados se utilizan datasets (que se explicarán en el capítulo siguiente) con imágenes especialmente preparadas.

Hay varias actuaciones sobre las imágenes en su conjunto que se han evaluado para su aplicación en OCR y HTR, orientadas a enderezar el contenido de la imagen, mejorar el contraste, el brillo y a reducir el ruido. Además de ello, cuando las imágenes representan páginas completas y el proceso de OCR o HTR trabajan a nivel de línea, es necesario a menudo segmentar la imagen, es decir, dividirla de forma que cada subdivisión contenga una sola línea del texto que se quiere reconocer.

2.6.1 Ajuste de iluminación

El ajuste de iluminación trata de evitar que la imagen tenga zonas muy oscuras o muy claras, y hacerlo sin afectar a su calidad general. Dos métodos habituales son CLAHE (Contrast Limited Adaptative Histogram Equalization), mencionado en [24], o también la normalización del array que representa la imagen. Para las imágenes utilizadas como datos de entrada, y tras combinarse con el resto de preprocesamiento, no se aprecia una gran diferencia entre uno y otro método en cuanto al resultado.

2.6.2 Conversión a escala de grises

Prácticamente todos los artículos que proponen un flujo de preprocesamiento de imagen ([24], [25], [26], [27]) incluyen como primer o segundo paso la conversión a escala de grises de la imagen. El motivo es que reduce los canales y, con ello, se facilita identificar mejor las áreas claras de las oscuras, donde probablemente hay trazos de la escritura. La conversión a escala de grises es sencilla tanto si la imagen está codificada en RGB (Red, Green, Blue) bastando sumar los tres valores de cada píxel y dividiendo el resultado entre tres, como si se utiliza codificación HSL/HSV, donde el canal L (por *lightness*, luminancia) o V (por *value*, valor) indica la luminosidad.

2.6.3 Eliminación de ruido

La eliminación de ruido se hace con técnicas como un filtro gaussiano [25], la media no local ([25], [27]) o el suavizado por mediana [26].

2.6.4 Afilado de imagen

El afilado de imagen trata de mejorar el perfilado de los trazos de la imagen, para lo cual se aplica un filtro que *resta* de la imagen original una versión borrosa de la misma. Mientras que algunos autores consideran que mejora la imagen [24], otros lo descartan [27] porque también se amplifican las marcas y deterioros en imágenes de documentos antiguos.

2.6.5 Erosión y dilatación/expansión

Estas dos técnicas, aplicadas consecutivamente, suavizan las imperfecciones de los trazos manuscritos [26], *adelgazando* primero el trazo y *engrosándolo* después.

2.6.6 Binarización

La binarización es el objetivo último ideal para las imágenes que se quieren procesar con HTR. Consiste en asegurar que todos los píxeles de la imagen son, bien completamente negros, bien completamente blancos. El artículo *OCR binarization and image pre-processing for searching historical documents* [28] describe y compara varias técnicas de binarización asumiendo que se dispone de una imagen en escala de grises

en el rango [0, 1], donde 0 es negro absoluto y 1 blanco absoluto. Desde el más simple umbral constante (se fija un umbral, que puede ser 0,5 y todos los píxeles por debajo de ese valor se cambian a 0, mientras que el resto se cambian a 1) a métodos más complejos como Otsu, que trata de encontrar el umbral que minimiza la desviación estándar de los píxeles resultantes, su variante local (el mismo algoritmo aplicado dividiendo la imagen en un mosaico), Sauvola-Niblack (similar a la variante local de Otsu pero con cálculos que consideran no solo la desviación estándar sino también la media), difusión de error marginal o modelos de Markov. La conclusión del estudio es que, siendo Otsu uno de los métodos que ofrece mejores resultados, no hay una solución única y ni siquiera la binarización es siempre aconsejable, por lo que sería necesaria una revisión manual (o alguna post-evaluación automática) para elegir cuándo binarizar y cuándo no.

Por su parte, en [27] se indica que diferentes artículos llegan a conclusiones diferentes sobre el mejor método de binarización y la causa probablemente sea los diferentes conjuntos de datos empleados para hacer la medición.

2.7 Referencias académicas sobre HTR en documentos históricos

En este apartado se citan específicamente algunos artículos centrados en HTR en relación con las Humanidades y la investigación histórica. Algunos ya han sido citados en otras partes de este trabajo, pero se relacionan aquí para condensar toda la información relativa a este ámbito que, al fin y al cabo, es el origen de este trabajo. La lista está ordenada cronológicamente.

- *OCR binarization and image pre-processing for searching historical documents*, de Gupta et al [28]. Técnicas de preprocesamiento de imágenes para documentos históricos.
- *Handwritten Text Recognition (HTR) of Historical Documents as a Shared Task for Archivists, Computer Scientists and Humanities Scholars. The Model of a Transcription & Recognition Platform (TRP)*, de Mühlberger et al [29]. Presentación de una plataforma para HTR para aglutinar los proyectos en esa materia de todas las bibliotecas y archivos. El proyecto estaba conectado con tranScriptorium (no confundir con

eScriptorium), coordinado por la Universidad Politécnica de Valencia, que actualmente es una plataforma comercial renombrada a tranSkriptorium[2].

- *Handwritten Text Recognition in Historical Documents*, de Scheidl, H [30]. Trabajo de fin de grado en ingeniería en el que se implementa un modelo CRNN con MDLSTM.
- *Transformer-based HTR for Historical Documents*, de Ströbel et al [31]. Aplicación de TrOCR, una red de tipo Transformer para OCR y HTR a una colección de 4.000 imágenes de un libro del siglo XVI junto con otras 16.500 líneas en latín.
- *Sharing Data for Handwritten Text Recognition (HTR)*, de Stokes y Kiessling [32]. Discusión sobre estándares e iniciativas para la efectiva compartición de modelos entrenados para HTR, especialmente en idiomas menos populares, los criterios que deberían regular las transcripciones para hacerlas lo más interoperables posible y la apropiación que algunas plataformas HTR y las grandes compañías hacen de los datos que se proporcionan cuando se publican o usan sus servicios para estas tareas.
- *Historical Documents and Automatic Text Recognition: Introduction*, de Pinche y Stokes [33]. Recopilación de avances (hasta marzo de 2024) sobre HTR, centrándose en eScriptorium y Kraken como aplicaciones, y HTR-United como repositorio de modelos OCR y HTR.
- *An End-to-End Approach for Handwriting Recognition: From Handwritten Text Lines to Complete Pages*, de Castro et al [34]. Presentación de una arquitectura basada en Transformers entrenada y evaluada con textos históricos en alemán entre los siglos XV y XVIII (dataset READ 2016).
- *Enhanced HTR Accuracy for Tibetan Historical Texts - Optimising Image Pre-processing for Improved Transcription Quality*, de Sabbagh [27]. Técnicas de preprocesamiento de imágenes aplicadas sobre colecciones de documentos impresos (periódicos del siglo XX) en tibetano.

[2] https://www.transkriptorium.com/

2.8 CER y WER

Para evaluar la tasa de aprendizaje de cada época de entrenamiento, así como el rendimiento de las distintas implementaciones y arquitecturas en el ámbito de OCR y HTR se utilizan dos métricas.

2.8.1 CER (Character Error Rate)

También conocida como distancia de edición, o distancia Levenshtein, esta métrica compara la cadena que predice el sistema para una imagen con la etiqueta con el texto correcto. La fórmula es la siguiente:

$$CER = \frac{S + D + I}{N} \times 100$$

Donde:

- S es el número de sustituciones. Una sustitución es un carácter incorrecto en la predicción donde la etiqueta con el texto correcto tiene otro diferente.
- D es el número de eliminaciones. Una eliminación es un carácter eliminado en la predicción respecto a la etiqueta con el texto correcto.
- I es el número de inserciones. Una inserción es un carácter añadido de más por la predicción respecto a la etiqueta con el texto correcto.
- N es el número total de caracteres de la etiqueta del texto correcto.

La multiplicación por 100 puede hacerse si se quiere expresar como porcentaje. En caso contrario, un CER de 1 significa que la tasa de errores es igual que el número de caracteres de la etiqueta con el texto correcto (lo que es mucho, obviamente).

Una peculiaridad del CER es que puede tomar valores mayores que 1 (o 100), si la predicción es más larga que la etiqueta y contiene muchos errores, de tal forma que la suma de todos ellos es mayor que N.

2.8.2 WER (Word Error Rate)

La tasa de error de palabras se calcula igual que la de caracteres, pero contando palabras en lugar de caracteres. Si tenemos una etiqueta de texto con 35 caracteres y siete palabras, y la predicción yerra en tres caracteres, cada uno de ellos en una

palabra, tendremos un CER de 3/35 = 0,0857 (o un 8,6 %), mientras que el WER será de 3/7 = 0,4286 (o un 42,86 %). Por tanto, los valores de WER suelen ser siempre mucho mayores que los de CER y algo menos representativos.

2.9 Aplicaciones en el mercado

Hasta ahora se ha descrito el estado de la cuestión en cuanto a las técnicas e investigaciones realizadas. Aunque los artículos científicos en las que se describen ocasionalmente mencionan que fueron directamente aplicadas en uso industrial, en la mayoría de los casos la implementación es usada solo para realizar comparativas de rendimiento con otras alternativas y como pruebas de concepto.

En este apartado se mencionan, sin embargo, aplicaciones completas orientadas a realizar HTR, que normalmente implementan los avances más recientes en cuanto a las técnicas descritas anteriormente. Entre las aplicaciones disponibles, se han revisado las siguientes:

- Tesseract OCR [35]: Tesseract OCR nació en el seno de Hewlett-Packard en 1985 como un motor OCR. En 2005 fue publicado como código abierto y su desarrollo fue retomado por Google hasta 2018. Actualmente es un proyecto de código abierto disponible en GitHub.com.
- OCR4All [36]: OCR4All es una aplicación de código abierto nacida en la Universidad de Würzburg que puede obtenerse y usarse mediante un contenedor Docker para una experiencia similar a la de una aplicación comercial.
- eScriptorium [37]: eScriptorium es una aplicación de código abierto financiada con fondos europeos y cuyo desarrollo se dirige desde la École Pratique des Hautes Études – Université Paris Sciences et Lettres, en AOROC en París. La aplicación puede obtenerse como una colección de contenedores Docker y su repositorio se aloja en Gitlab.com.

Merece la pena mencionar también Transkribus. Transkribus es una aplicación de pago ofrecida como PaaS (Platform as a Service), que no ha sido evaluada aquí por su carácter propietario.

Capítulo 3 - Los datos

En este capítulo se describen los datos de entrada, que como se explicó al principio son imágenes de documentos históricos, sus características, los desafíos que suponen para el HTR y las estrategias seguidas para mejorar su calidad.

Además, se hablará de los datasets encontrados para HTR, sus características y las diferencias con las imágenes de los datos de entrada.

3.1 Los datos de entrada

Dentro de un ambicioso proyecto de investigación histórica sobre las crisis económicas a lo largo de la historia, una de las fuentes de información que deben analizarse está compuesta por documentos con contenido económico. En concreto, fueron facilitadas 399 imágenes en formato JPEG.

Como se puede ver en el ejemplo de la derecha, las imágenes representan actas que forman parte de protocolos notariales del año 1793. En ellas se recogen protestos de letras de cambio. El objetivo del proyecto, en última instancia, es extraer ciertos datos relevantes de cada protesto para realizar un análisis sistematizado de la información que contiene.

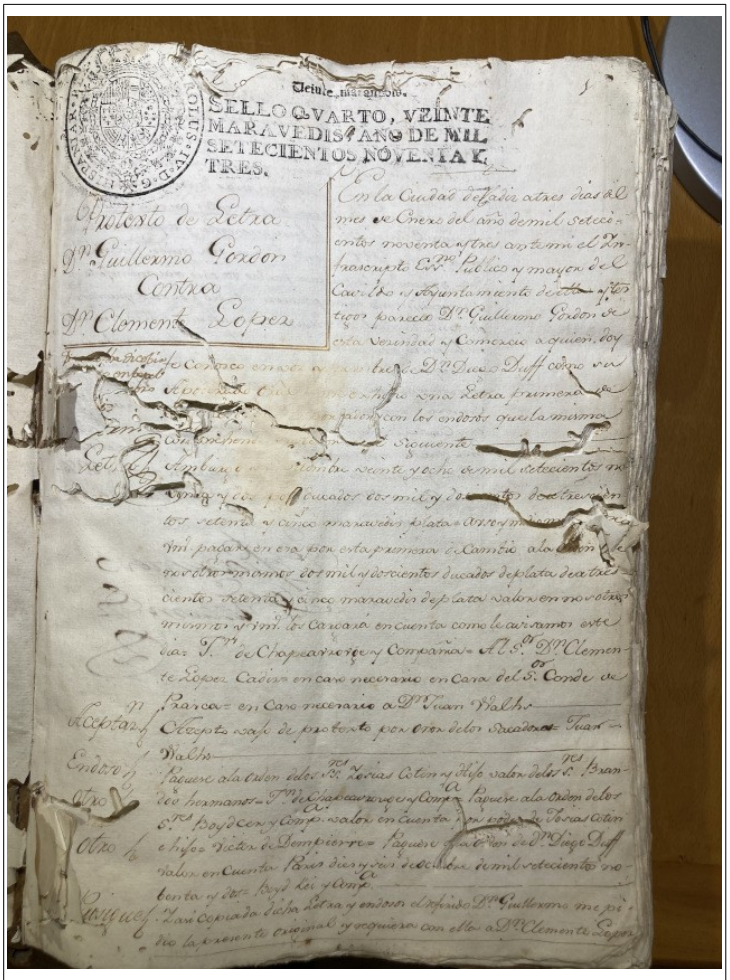
SELLO QVARTO, VEINTE MARAVEDIS AÑO DE MIL SETECIENTOS NOVENTA Y TRES.

Figura 7: Ejemplo de imagen de los datos de entrada

3.1.1 Características técnicas de las imágenes

Las características técnicas de las imágenes se resumen en la tabla siguiente:

Parámetro	Valor
Número de imágenes proporcionadas	399
Dimensiones (ancho por alto)	3024 x 4032 píxeles
Resolución	72 x 72 ppp
Tamaño medio	3,46 Mbytes
Canales	Color verdadero (16,7 millones)
Dispositivo de captura	Apple iPhone SE (segunda generación)

Tabla 1: Características técnicas de las imágenes de entrada

El conjunto completo ocupa aproximadamente 1,4 Gbytes en disco.

3.1.2 Características visuales

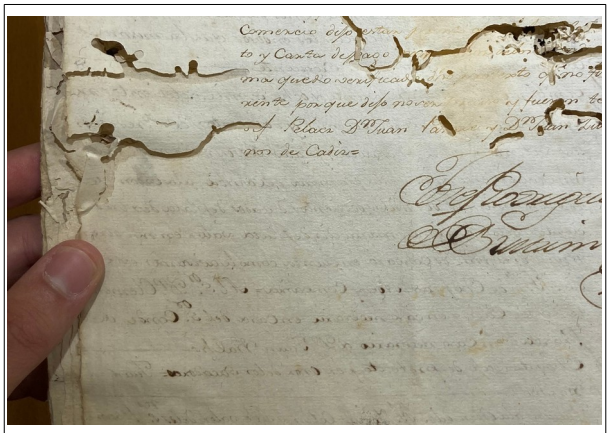

Figura 8: Características no relevantes para HTR

Como ya se ha dicho, las imágenes representan páginas completas tomadas directamente de un libro o cuaderno notarial, por lo que no es totalmente infrecuente que para conseguir que la página estuviera aproximadamente plana fuese necesario sujetarla, de modo que a veces se capta en la imagen el dedo con el que se mantuvo la página estable. También es casi constante que la imagen incluya un margen exterior a la página. En ningún momento estas circunstancias supusieron que se ocultara el texto de la página ni dieron lugar a la inclusión de texto ajeno a la misma.

En cambio, se considera que las siguientes características visuales de las imágenes sí tienen importancia para su procesamiento en el ámbito de HTR:

- Imagen en color: aunque tiene una solución sencilla, como se comentará en el capítulo 5, el hecho de que las imágenes originales estén en color se incluye en esta

lista porque afectó críticamente a la segmentación de líneas de texto en una de las aplicaciones evaluadas, eScriptorium[3].

- Márgenes laterales de contrapágina con texto: como las imágenes no están ajustadas para contener estrictamente la hoja que se quería fotografiar y esta es parte de un libro, es frecuente que contengan el margen de la contrapágina del libro y, en ocasiones, se aprecia texto manuscrito que será detectado, confundiendo el proceso.

Figura 9: Detalle de imagen con texto en contrapágina

- Contraste variable de una página a otra: inevitablemente, en una colección de casi 400 imágenes tomadas manualmente, no todas las páginas tienen el mismo nivel de brillo y contraste, por lo que los procesos automáticos de corrección no pueden configurarse con valores fijos.
- Contraste y brillo variable en el eje vertical de la página: no solo el contraste y brillo varían de una página a otra. En la imagen de una página, a menudo el nivel de contraste e iluminación es diferente en la parte superior de la imagen respecto a la inferior. Aunque el ojo humano puede separar fácilmente la escritura del fondo cuando se observa la imagen en color, para el procesamiento automático esta variabilidad supone cierto desafío.
- Marcas de texto escrito en el reverso: en muchas páginas, la imagen deja ver abundantes trazas del texto escrito en el reverso de la hoja fotografiada, lo que contribuye a confundir al proceso de HTR.
- Manchas de tinta y suciedad por el paso del tiempo: dependiendo del autor, algunas

Figura 10: Ejemplo de imagen con marcas de texto del reverso y manchas

[3] Se hizo un primer procesamiento con imágenes en escala de grises que dio buenos resultados, tras lo cual se repitió con un conjunto mayor, pero por error se usaron las imágenes en color. El resultado fue que ¡la aplicación identificó regiones de líneas de texto verticales!

páginas están escritas con sobrecarga de tinta, lo que produce goteos y manchas en las páginas. Así mismo, debido al paso del tiempo, hay manchas de suciedad, texto difuminado, etc.

- Deterioro del soporte papel por el paso del tiempo: en la figura 10 puede verse también que faltan trozos del papel por acción de las bacterias a lo largo del tiempo. El efecto en la imagen es el mismo que si hubiera manchas de gran tamaño.
- Disposición variable del contenido en la página: en la Figura 7, al principio de la sección *Los datos de entrada*, se ve la disposición típica de la primera página que compone un protesto. Lo más frecuente es que cada protesto esté compuesto por dos o tres páginas, llevando la firma del escribano en la última página. En la figura 11 se han marcado las secciones presentes en la primera página. Con una T figura el timbre (que indicaba una tasa por la oficialización del documento), con una S el sello, con una D el título del documento e intervinientes, y con una A una de las anotaciones al margen. Todos estos elementos afectan al flujo de lectura del documento (identificado como RO en los artículos académicos sobre HTR y OCR).

Figura 11: Secciones típicas en primera página

- Diferentes autores del texto manuscrito: todos los documentos que componen los datos de entrada facilitados pertenecen a una misma escribanía, pero están redactados por diferentes autores, por lo que hay diferentes caligrafías.
- Inclinación de las líneas escritas dentro de la misma página: algunos renglones de texto están redactados con inclinación respecto al margen horizontal de la página, lo que puede dificultar el proceso de segmentación de líneas.

3.1.3 Características lingüísticas

Los documentos que componen los datos de entrada corresponden al año 1793. Aunque el castellano es relativamente moderno y resulta sencillo entender su

significado, pueden apreciarse bastantes detalles que los distinguen de un texto moderno:

- Ausencia mayoritaria de tilde en palabras que, de acuerdo a las reglas actuales de ortografía, deben llevarla (por ejemplo, *segun* en lugar de *según*).

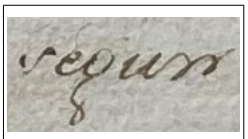

Figura 12: Tilde no presente

- Representación gráfica de fonemas diferente a la actual, como por ejemplo *cavildo* en lugar de *cabildo*, o *vezino* en lugar de *vecino*.
- Unión de lemas (palabras) en la redacción, de modo que es frecuente encontrar *dea* en lugar de *de a*, o *delos* en lugar de *de los*.

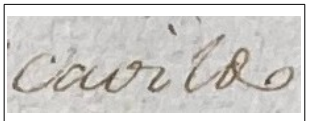

Figura 13: Ortografía diferente para un mismo fonema

Estas cuestiones no afectan a la extracción de características en sí, pero deben ser tenidas en cuenta si la red neuronal se apoya en un vocabulario o bien se ejecuta un posprocesamiento similar al de un corrector ortográfico.

3.2 Desafíos para HTR

El tipo de imagen ideal para un proceso de HTR es una imagen que no tendrá manchas, los renglones estarán escritos perfectamente horizontales con una disposición estable en columnas (ya sea una de lado a lado, o varias, siempre que se mantengan constantes en toda la página) y en la que cada píxel de la misma es, bien totalmente blanco, bien totalmente negro. Cuando el procesamiento se hace a nivel de línea, la imagen representará un único renglón, sin que se muestren los ascendentes y descendentes de las letras de los renglones inferior superior, respectivamente.

Por supuesto, no es así como son las imágenes, al menos inicialmente. Por ello, es necesario realizar un tratamiento de las imágenes antes de ponerlas a disposición del proceso de HTR. A este tratamiento se le suele llamar preprocesamiento de la imagen y puede realizarse individualmente, fichero por fichero, utilizando editores gráficos. Obviamente, esta tarea consume mucho tiempo, por lo que debe intentar automatizarse para que cubra el mayor número de casos.

Algunas de las aplicaciones evaluadas tienen incorporado ese preprocesamiento en su flujo de trabajo, mientras que otras pueden funcionar sin él, aunque se benefician de un preprocesamiento previo de las imágenes. Existen diversas bibliotecas de funciones para tratamiento gráfico, entre las que se ha utilizado en los scripts de proceso en lote una muy popular llamada OpenCV. Ambas cuestiones se tratarán más en detalle en los capítulos siguientes.

3.3 Datasets existentes

Para poder entrenar una red neuronal es preciso disponer de datos de entrenamiento, que en el caso de HTR que nos ocupa deben consistir en:

- Varios ficheros de imágenes que representen texto manuscrito.
- Uno o varios ficheros que asocian las imágenes al texto que representan, o su *etiqueta*. Generalmente ese otro fichero es de texto sencillo, en el que cada línea contiene el nombre o ruta de un fichero que representa una imagen, junto con el texto que contiene. A veces el fichero incorpora también otros metadatos, como la longitud del texto, si la transcripción es correcta, etc.

Al conjunto de imágenes y ficheros de etiquetado se les conoce como datasets o databases. Para HTR hay varios datasets públicos populares, que se describen aquí y que se han utilizado para probar los scripts Python que implementan la red CRNN. Al final de este apartado se resumen las características de todos ellos en la Tabla 2.

3.3.1 IAM

El dataset IAM [38] puede descargarse, previo registro, desde su página[4] en la Universidad de Berna, en Suiza. Contiene más de 13.000 líneas de texto escritas en inglés, procedentes de más de 600 autores, que transcribieron el Corpus Lancaster-Oslo/Bergen de inglés británico.

3.3.2 RIMES

El dataset RIMES [39] contiene 12.723 páginas, que corresponden a 5.605 correos (postales) de dos a tres páginas cada uno, escritos en francés moderno por

[4] https://fki.tic.heia-fr.ch/databases/iam-handwriting-database

más de 1.300 autores. El dataset principal contiene una imagen por cada página, con su correspondiente fichero XML con la estructura del documento y la transcripción. Hay datasets auxiliares con las imágenes de los párrafos (sin encabezados ni logotipos), líneas y palabras, pero no tienen ficheros con las transcripciones; en su lugar, a partir de los nombres de los ficheros de imágenes deben extraerse las transcripciones analizando los XML del dataset principal.

3.3.3 READ 2016

Este dataset [40] está compuesto por un subconjunto de documentos de la colección Ratsprotokolle, formada por actas de las sesiones del consejo mantenidas desde 1470 hasta 1805. Aunque la colección contiene cerca de 30.000 páginas, el dataset está formado por un subconjunto de unas 400 (350 para entrenamiento y 50 para test), escrito en un *temprano alemán moderno*. Las imágenes están en formato JPEG, en color, con su correspondiente fichero XML en estructura PAGE con la segmentación de líneas y la transcripción de cada una de estas. Hay aproximadamente 20 líneas por página, de modo que el total de líneas ronda las 8.000.

3.3.4 Rodrigo

El dataset Rodrigo [41] corresponde a la digitalización del libro *Historia de España del arçobispo Don Rodrigo*, de 1545, escrito en español antiguo. Consta de 15.010 líneas manuscritas de un mismo autor, con las imágenes saneadas.

3.3.5 GERMANA

Lamentablemente, el dataset GERMANA [42] no parece estar disponible (la URL que se menciona en el artículo ya no está operativa), pero se menciona porque sería óptimo para los datos de entrada con los que se cuenta en este TFM. El dataset era el resultado de digitalizar y anotar el manuscrito *Noticias y documentos relativos a Doña Germana de Foix, última Reina de Aragón*, escrito en 1891 por Vicent Salvador, marqués de Cruïlles. Constaba de unas 21.000 líneas de texto, mayoritariamente en español, con apariciones de catalán, francés, latín y, ocasionalmente, alemán e italiano.

Dataset	Nº líneas	Autores	Idioma	Formato etiquetas

IAM	13353	657	Inglés	TXT
RIMES	12723	1300	Francés	XML
READ 2016	8000 (aprox.)	(desconocido)	Alemán	XML
Rodrigo	15010	1	Español antiguo	TXT
Germana	20529	1	Español XVIII	N/D

Tabla 2: Datasets para HTR y sus características

3.4 Diferencia entre los datasets y los datos de entrada

Todos los datasets, salvo READ 2016, tienen en común entre ellos una serie de diferencias con los datos de entrada:

- Están segmentados en líneas (aunque RIMES precisa un tratamiento previo para obtener el fichero de etiquetas), mientras que los datos de entrada son de imágenes completas.
- Cada imagen contiene exclusivamente el texto etiquetado, mientras que en las imágenes de entrada a veces hay texto de la página contigua a la que se fotografió.
- En las imágenes de los datasets se ha llevado a cabo un preproceso para eliminar, en general, ruidos, pendientes del renglón de escritura, ascendentes y descendentes de las líneas precedente y siguiente y otras alteraciones.

En el caso de READ 2016, aunque las imágenes son de cada página completa, la información proporcionada por los ficheros XML permite dividir aquellas en cada una de las líneas de texto. Comparten ciertas dificultades, como la tinta que se transluce desde el reverso de cada página, con las imágenes de los datos de entrada, pero se encuentran en mejor estado general.

Todas estas diferencias tendrán un gran impacto en los resultados, como se explicará en los siguientes capítulos.

Capítulo 4 - Aplicaciones para HTR evaluadas

En este capítulo se detallan algunas aplicaciones de código abierto que han sido evaluadas. La lista de aplicaciones se obtuvo de diferentes fuentes: recomendaciones del director de este trabajo, respuestas a consultas a IA generativa y menciones en foros técnicos en consultas sobre OCR/HTR. Si bien algunas de las fuentes pueden no parecer muy académicas, dos de ellas (y alguna más que ya no está disponible) fueron presentadas a través de artículos científicos o tienen su desarrollo impulsado y dirigido desde instituciones académicas de primer nivel en ámbito histórico y lingüístico.

Para cada aplicación se hará un pequeño resumen de la misma, se indicará el procedimiento de instalación y puesta en marcha, se explicará el flujo de trabajo y se hará un análisis de sus resultados y facilidad de uso para el procesamiento de los 399 documentos.

4.1 Tesseract OCR

Tesseract OCR está disponible como proyecto en GitHub [35], donde podemos encontrar información sobre el proyecto. Nació en 1985 en los laboratorios de Hewlett-Packard de Bristol y se desarrolló en la empresa hasta 1996. En 2005 se publicó su código fuente y pasó a ser un proyecto de código abierto. Hasta la versión 3 funcionaba mediante reconocimiento de patrones, y a partir de la versión 4 implementa en su lugar una red neuronal LSTM. La última versión principal publicada es la 5, en 2021, aunque la última versión de correcciones a fecha 8 de junio de 2025 es la 5.5.1.

4.1.1 Instalación

Tesseract OCR está disponible para diversas plataformas, incluyendo varias distribuciones de Linux y Mac OS. También hay instaladores para Windows. La información completa está en la página de documentación de Tesseract[5].

[5] https://tesseract-ocr.github.io/tessdoc/Installation.html

En el caso de Linux, se puede realizar la instalación mediante el gestor de paquetes de la propia distribución mediante estas órdenes:

```
sudo apt install tesseract-ocr
sudo apt install libtesseract-dev
```

Tesseract no incluye, por sí mismo, una interfaz gráfica. Si se desea utilizar una interfaz gráfica, puede instalarse gImageReader. Como referencia, la tabla de la derecha lista los paquetes que se instalaron en el sistema de desarrollo para usar Tesseract y gImageReader. Muchos son dependencias y se instalarán automáticamente sin necesidad de especificarlos en la línea de órdenes; otros son paquetes de idiomas que se consideró oportuno incluir.

liblept5	gimagereader
libtesseract5	gimagereader-common
tesseract-ocr	libgtksourceviewmm-3.0-0v5
tesseract-ocr-cat	libgtkspell3-3-0
tesseract-ocr-eng	libgtkspellmm-3.0-0v5
tesseract-ocr-glg	libpodofo0.9.8t64
tesseract-ocr-osd	libxml++2.6-2v5
tesseract-ocr-por	
tesseract-ocr-spa	

Tabla 3: Lista de paquetes para Tesseract OCR y gImageReader

4.1.2 Notas sobre operativa

Se comenta aquí la operativa usando gImageReader. Aunque es posible utilizar Tesseract desde la línea de órdenes, como se estaba evaluando su capacidad para reconocer texto manuscrito se usó la interfaz gráfica (y las pruebas demostraron que no era la solución adecuada). La figura 14 muestra la interfaz descrita en la página siguiente.

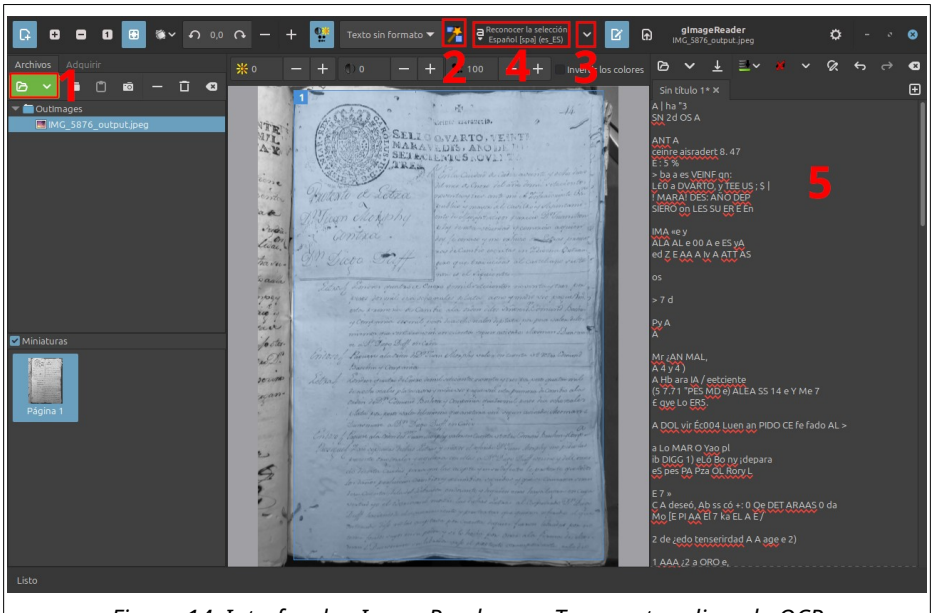

Figura 14: Interfaz de gImageReader con Tesseract realizando OCR

En la imagen se han marcado con números en rojo las interacciones habituales:

1. Permite elegir un fichero del sistema. Se puede crear una lista de ficheros para procesarlos todos en una sola sesión.
2. Hace que Tesseract detecte la disposición y región de la imagen que contiene texto. También es posible, como se hizo en la imagen de ejemplo, arrastrar para seleccionar la región manualmente.
3. En el desplegable marcado es posible indicar el idioma en el que está escrito el texto para facilitar a Tesseract el proceso.
4. Lleva a cabo el proceso en sí de reconocimiento.
5. En esa sección aparece el resultado del reconocimiento de texto, donde puede ser modificado si se quiere corregir algo. El texto puede seleccionarse y copiarse, o bien guardarse en un fichero de texto.

4.1.3 Pruebas

Se hicieron pruebas con imágenes que contenían texto impreso, obteniendo resultados bastante cercanos (en ocasiones del 100 %) al texto real. Sin embargo, en todas las pruebas con texto manuscrito el resultado fue, como el mostrado en la figura 14, totalmente inservible a simple vista, por lo que no se calculó el CER.

Buscando información sobre el motivo por el que Tesseract OCR ofrece tan malos resultados con texto manuscrito se encontró una de las preguntas frecuentes en la que se indicaba que Tesseract está diseñado para texto impreso y que deben buscarse proyectos centrados en HTR.

4.2 OCR4All

OCR4All se presentó en el mundo científico en 2019 como un software de codigo abierto para OCR sobre documentos históricos impresos. En la página *Acerca de* del proyecto[6], sin embargo, se dice que *OCR4all combina varias soluciones de código abierto para proporcionar un flujo de trabajo totalmente automatizado para el reconocimiento de texto automático de material histórico impreso (OCR) y manuscrito (HTR),* extendiendo por tanto su aplicabilidad al proceso HTR.

[6] https://www.ocr4all.org/about/ocr4all

Las soluciones de código abierto a las que se refiere el párrafo anterior son Larex y Calamari, además de otras soluciones para el preprocesamiento de imágenes.

4.2.1 Instalación

OCR4All se instala como un contenedor Docker que ofrece una interfaz web accesible desde el navegador del equipo local. Previamente a la instalación hay que crear una estructura de directorios local (ver figura 15, tomada de la web de OCR4All) donde se depositarán los ficheros de entrada y se recogerán los de salida, que será accesible al contenedor Docker.

```
...
├── ocr4all
|   ├── data
|   |   ├── [Your book]
|   |   |   ├── input
|   |   |   |   ├── 0001.png
|   |   |   |   ├── 0002.png
|   |   |   |   ├── ...
|   |   ├── [Another book]
|   |   |   ├── input
|   |   |   |   ...
|   ├── models
...
```

Figura 15: Estructura de directorios para OCR4All

La orden para recuperar la imagen del contenedor Docker es la siguiente:

```
docker pull uniwuezpd/ocr4all
```

Una vez instalado, se puede iniciar el contenedor con la siguiente orden:

```
sudo docker run -p 1476:8080 \
    -u `id -u root`:`id -g $USER` \
    --name ocr4all \
    -v $PWD/data:/var/ocr4all/data \
    -v $PWD/models:/var/ocr4all/models/custom \
    -it uniwuezpd/ocr4all
```

Y, una vez totalmente inicializado el contenedor Docker, se puede acceder a través de un navegador en la siguiente URL:

```
http://localhost:1476/ocr4all/
```

El aspecto de la aplicación es el siguiente:

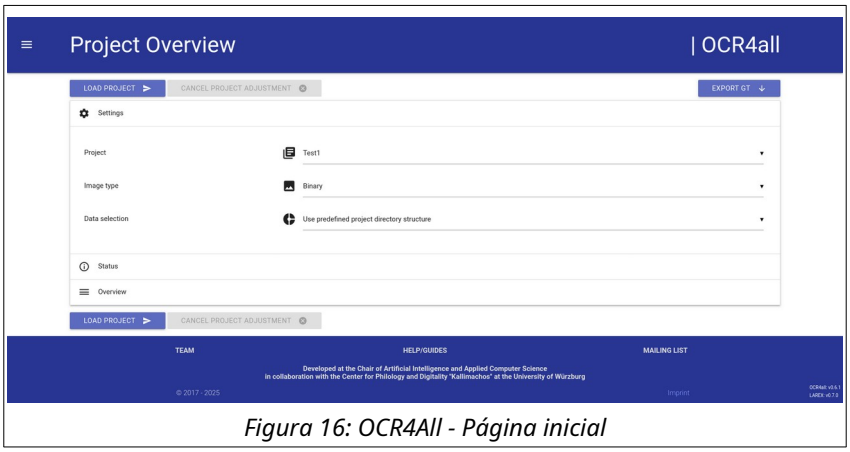

Figura 16: OCR4All - Página inicial

4.2.2 Creación de proyecto

OCR4All trabaja con proyectos, los cuales no se crean en la aplicación. En su lugar, se debe crear en la carpeta `data` de la estructura de directorios una subcarpeta que representa el proyecto, con el nombre que queremos que tenga en la aplicación. Dentro de la subcarpeta del proyecto se crea una subcarpeta llamada `input`, en la cual se depositarán los ficheros de imágenes que componen nuestro proyecto.

Una vez realizado el paso anterior, el proyecto aparecerá en el desplegable *Project* de la página inicial de OCR4All.

4.2.3 Flujo de trabajo

La página de introducción de la guía de usuario de OCR4All contiene la imagen de la figura 17 con el flujo de trabajo de la aplicación.

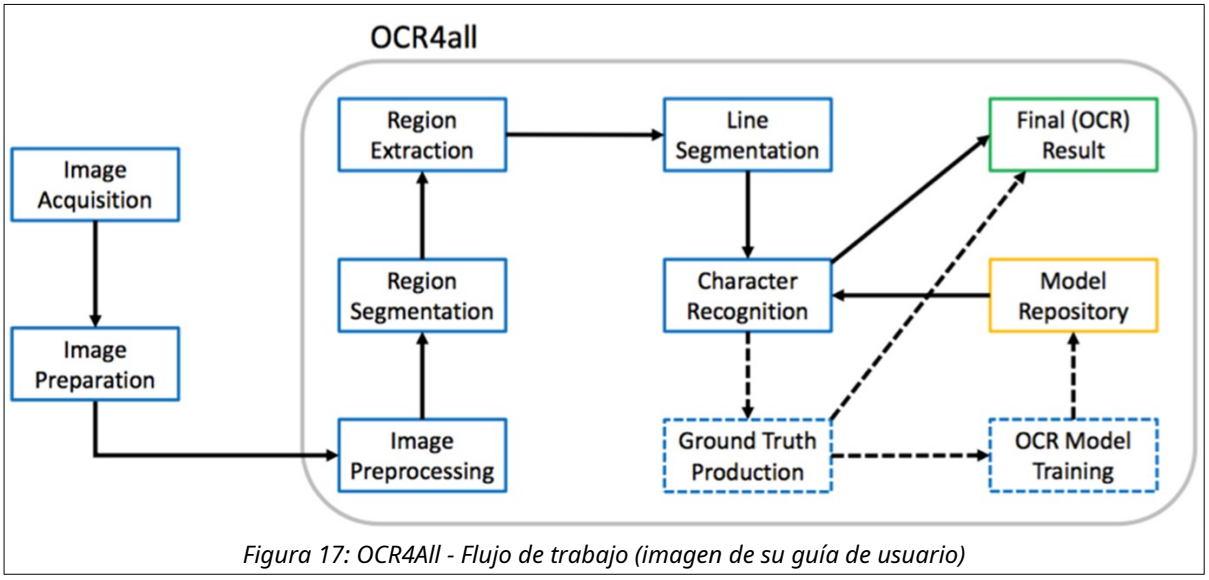

Figura 17: OCR4All - Flujo de trabajo (imagen de su guía de usuario)

Al cargar el proyecto, si las imágenes no están en formato PNG y con nombre secuencial 0001.png, 0002.png, etc. OCR4All advertirá de ello y ofrecerá realizar los cambios necesarios directamente. También se realiza cierto preprocesamiento de las imágenes, binarizándolas, convirtiéndolas a escala de grises y eliminando ruido. Cada una de estas conversiones se realiza de manera independiente (es decir, no se aplican todas sucesivamente para obtener un resultado final).

A partir de ahí, se van siguiendo las opciones del menú que aparece al hacer clic en las tres líneas horizontales de la parte superior izquierda de la página. El menú completo es el que se muestra en la figura 18, a la derecha en esta página.

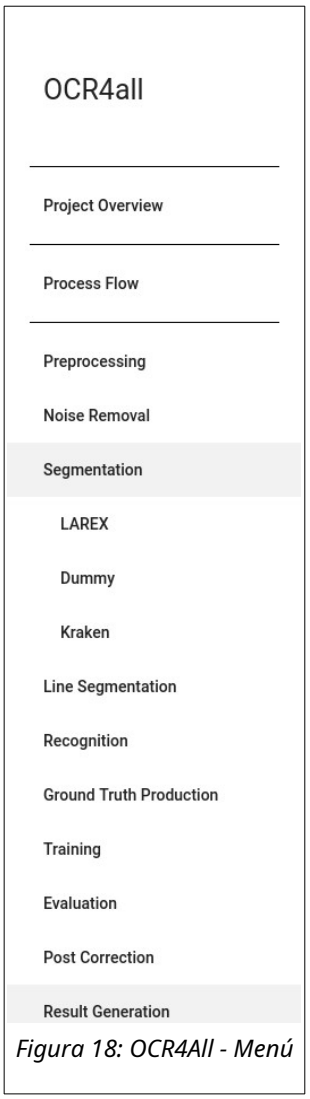

Figura 18: OCR4All - Menú

Para hacer la segmentación se abre una aplicación web separada desde el menú. Puede elegirse LAREX (la opción documentada), Dummy (que significa hacer la segmentación manualmente) o Kraken (usado también por eScriptorium). LAREX y Kraken hacen un trabajo similar, con la diferencia de LAREX proporciona una interfaz gráfica (como una aplicación web que se abre en una pestaña separada) para permitir al usuario corregir manualmente la segmentación, mientras que el proceso de Kraken es totalmente desatendido.

La segmentación de líneas se realiza de forma automática a partir de la información recopilada en el paso anterior. Seguidamente se ejecuta el proceso de reconocimiento (que usa la herramienta Calamari), en el cual podemos suministrar hasta 5 modelos previamente existentes que podrán combinarse para realizar la predicción del texto.

En este punto, se debe pasar a producir la información *Ground Truth*, que no es más que la corrección manual por el usuario del reconocimiento estimado en el paso anterior, para permitir generar un fichero de etiquetado con el que hacer un entrenamiento. La corrección se hace siempre con LAREX, cuya interfaz puede verse en la figura 19.

En la parte superior de la interfaz puede verse que hay tres vistas o pestañas:

- Segment, que muestra, para cada página, la zona que se ha identificado que contiene regiones con imágenes, encabezados, párrafos o texto que no se puede catalogar.
- Lines, que muestra cada subconjunto de la imagen que se considera que contiene una línea de texto.

- Text, que muestra la imagen de cada línea seguida de una caja con el texto predicho.

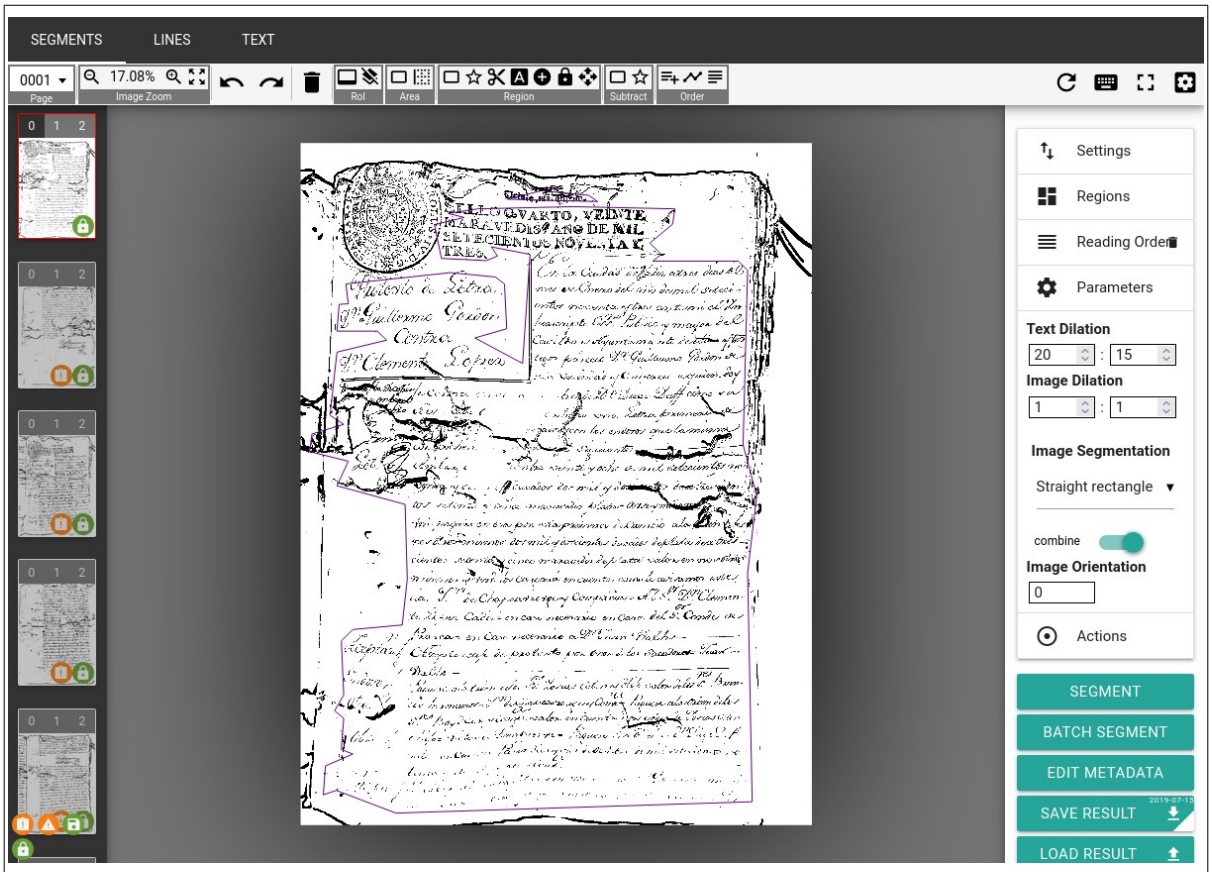

Figura 19: OCR4All - Revisión de la predicción con LAREX

Las imágenes del proyecto, mostradas en un panel vertical a la izquierda, tienen sobre ellas tres números que, al pulsarlos, permiten ver la versión binarizada, la que tiene eliminado el ruido y la original. La barra de herramientas superior permite editar la segmentación y las líneas detectadas, mientras que el panel de la derecha permite establecer el orden de las líneas detectadas y guardar cualquiera de los cambios realizados.

En la vista de segmentos se pueden modificar los vértices que componen cada uno, eliminar los que se hayan identificado incorrectamente o añadir nuevos. También se

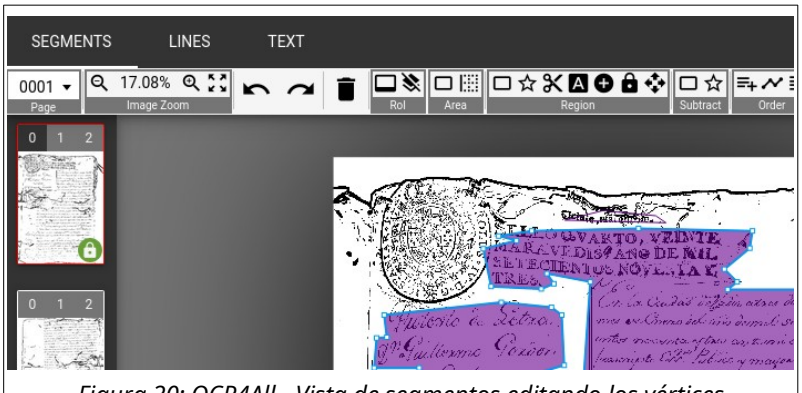

Figura 20: OCR4All - Vista de segmentos editando los vértices

puede variar el tipo de segmento (imagen, encabezado, párrafo, texto), para aportar información estructural sobre la disposición de la página.

Lo mismo puede hacerse en la vista de líneas, con las líneas detectadas. Editar los bloques que identifican las líneas, tanto en su forma como en su categorización, puede servir, con el número suficiente de documentos, para entrenar el sistema.

Figura 21: OCR4All - Vista de líneas

Por último, la vista de texto muestra cada línea identificada con el texto predicho debajo, donde puede ser modificado.

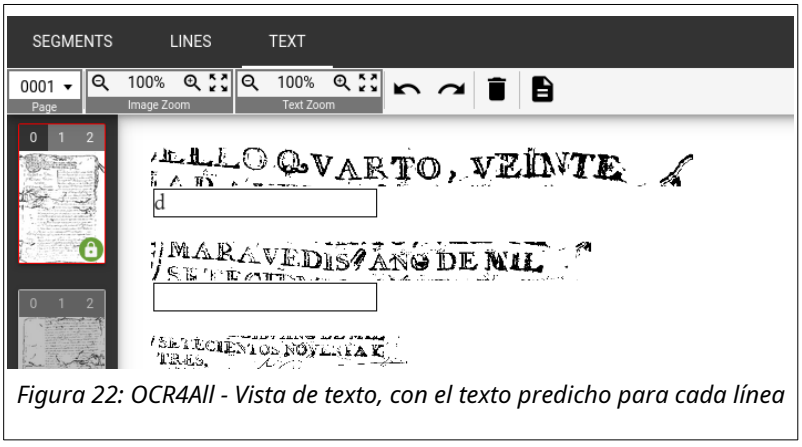

Figura 22: OCR4All - Vista de texto, con el texto predicho para cada línea

La información generada o corregida a lo largo de los tres paneles debe ser guardada para cada página, lo que permitirá construir el etiquetado, incluyendo tanto la segmentación (identificación de las líneas en la imagen de la página completa) como el texto asociado. El proceso se repite con cada página del proyecto.

Tras la revisión de la predicción se pasa al entrenamiento. Con las imágenes del proyecto que hayamos corregido se podrá generar así un modelo que luego podrá ser utilizado en posteriores reconocimientos (una forma de *fine-tuning*).

Concluido el entrenamiento, el apartado de evaluación pemite obtener una media de errores por cada carácter (una estadística diferente de las métricas CER o WER). El apartado de post-corrección nos lleva a LAREX una vez más, para llevar a cabo la misma operativa tras el entrenamiento. Para concluir, el apartado de generación de resultados permite generar un fichero de etiquetas en formato texto, o bien en formato PAGE XML, que contiene también la información de la segmentación.

4.2.4 Resultados

Los resultados con OCR4All, como se puede intuir por las figuras mostradas anteriormente, tampoco fue útil. Concretamente, en la figura 22 de la página anterior puede verse que el resultado de la predicción es inservible (incluso a pesar de que las primeras líneas se pueden considerar texto impreso, no manuscrito). Se ha atribuido el mal resultado en el reconocimiento a varios factores:

- Los modelos incluidos no contemplan HTR, solo OCR con tipografías antiguas, y no hay un repositorio de modelos adicional. Este es probablemente el motivo más importante de los malos resultados en la predicción.
- El proceso de segmentación de líneas es bastante mejorable y el esfuerzo en corregir las regiones y líneas detectadas por el sistema no rinde resultados debido a que no se conserva entre cada interacción con LAREX. Posiblemente el preprocesamiento de imágenes, especialmente la binarización, perjudica en ocasiones al proceso de segmentación, ya que cuando el contraste entre el fondo y el texto manuscrito es pequeño (lo que sucede con frecuencia si el papel está muy amarillento y la tinta ha perdido presencia) se acaban borrando trazos del texto.
- La combinación de modelos previamente entrenados no tiene efectos prácticos porque el número de páginas necesario para que el modelo generado cause algún impacto es mucho mayor del que puede conseguirse en un tiempo razonable, sobre todo con tantas interacciones con LAREX (y su baja tasa de acierto en la segmentación).

4.3 eScriptorium y Kraken

eScriptorium y Kraken [33] son dos herramientas de código abierto que trabajan en conjunto para proporcionar una solución de reconocimiento automático de texto tanto impreso como manuscrito. Kraken es un motor OCR, en desarrollo desde aproximadamente 2015, que combina análisis de la disposición de la página, segmentación de líneas, reconocimiento OCR/HTR y serialización (exportación de resultados). Por su parte, eScriptorium nació en 2019 y su propósito es proporcionar una interfaz gráfica amigable para Kraken.

4.3.1 Instalación

eScriptorium no tiene una página web donde se presente el producto como tal. En su lugar, toda la información se concentra en su repositorio en GitLab[7] donde se coordina el desarrollo. De la misma forma, tras visitar el repositorio se puede concluir que no existe una instalación descargable como tal. En el README del repositorio se ofrecen dos opciones: instalación Docker o instalación completa, la cual está desaconsejada si no se desea contribuir activamente al proyecto.

Para la instalación Docker, de acuerdo con su página[8], hay que tener instalado y configurado Docker (en Linux, Mac OS o Windows con WSL), descargar el repositorio con Git, situarse en el directorio raíz del repositorio y crear un fichero de configuración (`variables.env`). Tras ello, se obtienen las imágenes y se puede iniciar el sistema. La secuencia completa de instrucciones sería la siguiente:

```
$ git clone https://gitlab.com/scripta/escriptorium.git
$ cd escriptorium && cp variables.env_example variables.env
$ docker compose pull
$ docker compose up -d
```

Lamentablemente, si se siguen las instrucciones tal cual, los (numerosos) contenedores que componen la aplicación se iniciarán, pero una vez completado el proceso, si se intenta acceder a la URL de entrada a través de un navegador:

```
http://localhost:8080/
```

se obtiene un mensaje de error porque la aplicación no es accesible.

4.3.1.1 Bug

Revisando los *issues* existentes en el repositorio de eScriptorium se encontraron varios que podían estar relacionados con el problema, pero no se llegó a ninguna solución, por lo que se creó uno nuevo[9]. Las indicaciones proporcionadas allí no solucionaron directamente el problema, pero sí las de otro issue (el 1139). Finalmente, parece que en el estado de desarrollo a finales de enero de 2025 de eScriptorium, las

[7] https://gitlab.com/scripta/escriptorium

[8] https://gitlab.com/scripta/escriptorium/-/wikis/docker-install

[9] https://gitlab.com/scripta/escriptorium/-/issues/1132

imágenes Docker que se descargan siguiendo las instrucciones de instalación del wiki de eScriptorium no son totalmente compatibles con las versiones actuales de Docker. La solución es aplicar estos cambios:

- Modificar el archivo `ngnix/Dockerfile`, añadiendo al final la línea `COPY nginx.conf /etc/nginx/conf.d/nginx.conf`.
- Modificar el archivo `docker-composer.yml` del directorio raíz del repositorio, comentando las líneas `image` de x-app, ngnix and mail, así como una copia del archivo nginx.conf que queda reemplazada por el cambio en el fichero ngnix/Dockerfile.
- Ejecutar desde el directorio raíz del repositorio la orden `docker compose build`. Esto regenera las imágenes de los contenedores de manera local a partir del estado del repositorio, en lugar de descargarlas de DockerHub. Es un proceso relativamente largo que puede llevar unos 4~5 minutos.
- Ejecutar `docker compose up`.

Los ficheros mencionados quedan como sigue (marcado en negrita los cambios que se aplican).

Fichero ngnix/Dockerfile

```
FROM nginx:1.26.1-alpine

RUN rm /etc/nginx/conf.d/default.conf
COPY nginx.conf /etc/nginx/conf.d/nginx.conf
```

Fichero docker-compose.yml

```
version: "3.9"

x-app:
  &app
  # image: registry.gitlab.com/scripta/escriptorium:latest
  build:
    context: .
  env_file: variables.env
  volumes:
    # - ./app/:/usr/src/app/
    - static:/usr/src/app/static
    - media:/usr/src/app/media
  command: /bin/true
```

```
services:

  web:
    <<: *app
    command: uwsgi --ini /usr/src/app/uwsgi.ini
    expose:
      - 8000

  channelserver:
    <<: *app
    command: daphne --bind 0.0.0.0 --port 5000 -v 1
escriptorium.asgi:application
    expose:
      - 5000

  db:
    image: docker.io/library/postgres:15
    volumes:
      - postgres:/var/lib/postgresql/data/
    env_file: variables.env

  redis:
    image: "docker.io/redis:alpine"

  nginx:
    # image: registry.gitlab.com/scripta/escriptorium/nginx
    build: ./nginx
    environment:
      - SERVERNAME=${DOMAIN:-localhost}
    volumes:
#      - type: bind
#        source: $PWD/nginx/nginx.conf
#        target: /etc/nginx/conf.d/nginx.conf
      - static:/usr/src/app/static
      - media:/usr/src/app/media
    ports:
      - 8080:80

  celery-main:
    <<: *app
    environment:
      - OMP_NUM_THREADS=1
    command: "celery -A escriptorium worker -l INFO -E -Ofair --
prefetch-multiplier 1 -Q default -c ${CELERY_MAIN_CONC:-10} --max-
tasks-per-child=10"

  celery-live:
    <<: *app
```

```
    command: "celery -A escriptorium worker -l INFO -E -Ofair --
prefetch-multiplier 1 -Q live -c ${CELERY_LIVE_CONC:-10} --max-tasks-
per-child=10"

  celery-low-priority:
    <<: *app
    command: "celery -A escriptorium worker -l INFO -E -Ofair --
prefetch-multiplier 1 -Q low-priority -c ${CELERY_LOW_CONC:-10} --max-
tasks-per-child=10"

  celery-gpu: &celery-gpu
    <<: *app
    environment:
      - KRAKEN_TRAINING_DEVICE=cpu
    command: "celery -A escriptorium worker -l INFO -E -Ofair --
prefetch-multiplier 1 -Q gpu -c 1 --max-tasks-per-child=1"
    shm_size: '3gb'

  flower:
    image: docker.io/mher/flower
    command: ["celery", "--broker=redis://redis:6379/0", "flower", "--
port=5555"]
    ports:
      - 5555:5555

  # No need while we don't have regular tasks
  # celerybeat:
  #   build:
  #     context: ./app
  #   env_file: variables.env
  #   volumes:
  #     - ./static/:/usr/src/app/static
  #     - ./media/:/usr/src/app/media
  #     - static:./static/
  #     - media:./media/
  #   command: celery -E -A escriptorium beat -l INFO

  mail:
    build: ./exim
    # image: registry.gitlab.com/scripta/escriptorium/mail
    expose:
      - 25
    environment:
      - PRIMARY_HOST=${DOMAIN:-localhost}
      - ALLOWED_HOSTS=web ; celery-main ; celery-low-priority; docker0

volumes:
   static:
   media:
```

```
  postgres:
  esdata:
```

Una vez aplicados los cambios, al ejecutar `docker compose up` se debería observar algo similar a la figura 23. En la figura se ha añadido el parámetro -d para que no se vuelquen los mensajes de registro de todos los contenedores, pero en las primeras ejecuciones puede ser conveniente visualizarlos para diagnosticar cualquier problema. Por ejemplo, podría suceder que el contenedor ngnix-1 se inicie excesivamente pronto y acabe cerrándose al no encontrar operativo el contenedor web-1. Si esto sucede, los mensajes de registro lo indicarán. Una solución sencilla, una vez diagnosticado, es arrancar la aplicación con el parámetro -d y, si no es posible acceder por el navegador, simplemente ejecutar de nuevo la misma orden. Los contenedores en ejecución seguirán operativos y ngnix-1 se iniciará sin problema.

```
rpalomares@Grandfather ~/Repos/escriptorium $ sudo docker compose up -d
WARN[0000] /home/rpalomares/Repos/escriptorium/docker-compose.yml: the attribute
 `version` is obsolete, it will be ignored, please remove it to avoid potential
confusion
[+] Running 12/12
 ✔ Network escriptorium_default                 Created                  0.2s
 ✔ Container escriptorium-channelserver-1       Started                 34.1s
 ✔ Container escriptorium-celery-low-priority-1 Started                 11.3s
 ✔ Container escriptorium-flower-1              Started                 29.5s
 ✔ Container escriptorium-db-1                  Started                 23.0s
 ✔ Container escriptorium-mail-1                Started                 32.1s
 ✔ Container escriptorium-redis-1               Started                 18.8s
 ✔ Container escriptorium-celery-gpu-1          Started                 22.7s
 ✔ Container escriptorium-nginx-1               Started                 10.7s
 ✔ Container escriptorium-celery-main-1         Started                 12.0s
 ✔ Container escriptorium-web-1                 Started                 32.3s
 ✔ Container escriptorium-celery-live-1         Started                 17.1s
rpalomares@Grandfather ~/Repos/escriptorium $
```

Figura 23: eScriptorium - Arranque con Docker Compose

Una vez iniciada la aplicación, al visitar la URL indicada encontraremos una página de bienvenida como la de la derecha. Como puede verse, hay que iniciar sesión con un usuario. Por defecto se crea un usuario administrador con el nombre `admin` y la misma contraseña.

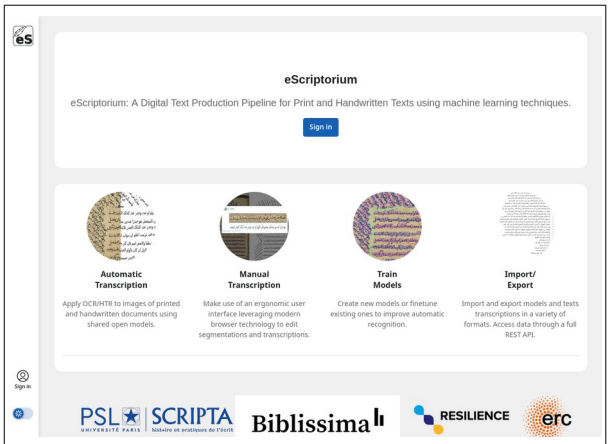

Figura 24: eScriptorium - Página de inicio

4.3.2 Flujo de trabajo

En primer lugar hay que indicar que hay varias fuentes de documentación sobre eScriptorium. El wiki del proyecto[10] contiene cierta documentación de carácter más bien técnico, pero en la página *Getting started with the interface* remite como documentación principal a readthedocs.io[11], donde hay una documentación muy

[10] https://gitlab.com/scripta/escriptorium/-/wikis/home

[11] https://escriptorium.readthedocs.io/en/latest/quick-start/

completa. No se va a repetir aquí el excelente contenido allí disponible, por lo que se hará un resumen muy breve para, en la siguiente subsección, explicar cómo hacer un *fine-tuning*, que es algo que no se puede realizar desde la interfaz de eScriptorium al estar usando la rama de desarrollo para generar y usar los contenedores.

Una vez iniciada la sesión en eScriptorium se accede a la lista de proyectos, donde es posible también crear uno nuevo. En cada proyecto podemos crear uno o varios *documentos*. Los documentos, en terminología de eScriptorium, se pueden asimilar a colecciones de páginas de libros, revistas o cuadernos. En cada documento podemos importar, a su vez, una colección de imágenes.

Los proyectos no tienen una configuración especial, más allá del nombre y una URL opcional para documentar guías de estilo de transcripción. Sin embargo, una vez se añaden documentos e imágenes y se realizan transcripciones, ofrece un panel de informes sobre el contenido del mismo.

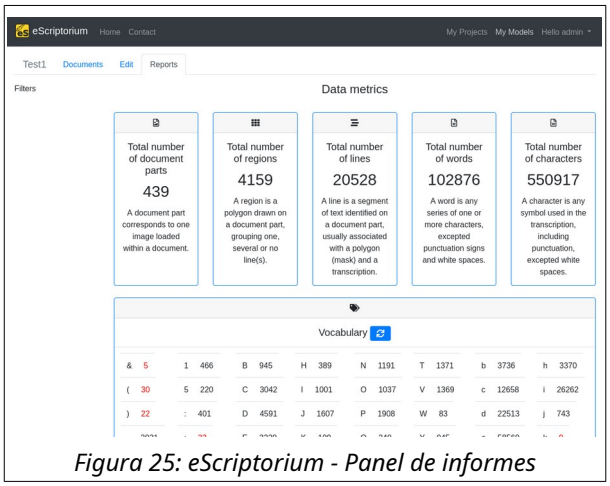

Figura 25: eScriptorium - Panel de informes

Los documentos sí contienen más configuración, como el tipo de alfabeto (latín, árabe, hebreo, etc.), la dirección de escritura, etc. También tienen asociada una ontología (clasificación de las regiones, líneas e imágenes...), informes similares a los del proyecto y, muy importante, modelos de entrenamiento, donde podremos importar modelos ajustados mediante *fine-tuning*. También tienen un panel de imágenes (ver figura 26), donde podemos importar nuestros ficheros y donde veremos su estado, tanto de segmentación como de transcripción. Con los botones a la derecha sobre la colección de imágenes podremos lanzar el proceso de segmentación y el de transcripción.

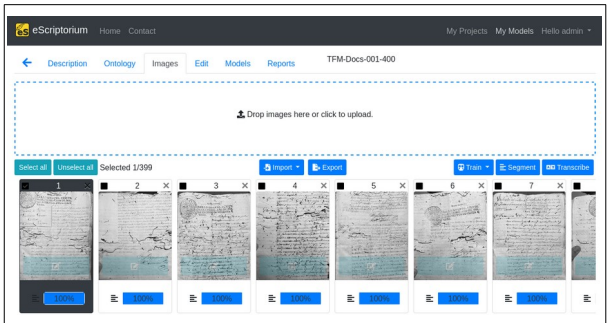

Figura 26: eScriptorium- Panel de documento con sus imágenes

Descrita brevemente la interfaz y la configuración inicial de los proyectos, el flujo de trabajo consiste en realizar primero la segmentación automática, revisarla manualmente, realizar la transcripción automática y revisarla manualmente.

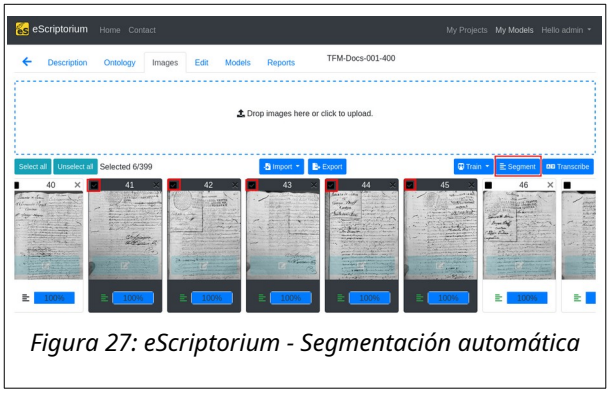

Figura 27: eScriptorium - Segmentación automática

Para realizar la segmentación automática hay que marcar las imágenes sobre las que se desea realizar (pueden ser todas o unas pocas, si queremos trabajar con paquetes de un tamaño controlado, dado que después conviene hacer una revisión manual). La segmentación la realiza el motor Kraken mediante un modelo de red neuronal, por lo que se pueden entrenar modelos de segmentación propios si se desea. Por defecto se incluye el modelo *blla.mlmodel*, que ofrece buenos resultados. La segmentación lleva unos segundos por página, siendo un proceso que se ejecuta en fondo. Hay que tener en cuenta que segmentar páginas ya transcritas eliminará la transcripción.

Figura 28: eScriptorium - Revisión de segmentación

Tras concluir la segmentación hay que revisar manualmente las imágenes. Dependiendo de sus características, la segmentación puede requerir ajustes (como es el caso en las imágenes que componen los datos de entrada). Pulsando sobre el botón de edición sobre la imagen deseada se accede al panel de edición de la misma. En la figura 28 se observa la segmentación. Las líneas azules representan la línea base del texto, mientras que las regiones coloreadas componen el segmento completo de la imagen. Como puede verse, hay diversos errores de segmentación, aunque para ser justos, el ejemplo contiene una tasa de errores inusualmente alta debido a la dificultad

de la imagen, con texto del reverso transparentado, texto de la página contigua y pendientes irregulares en las líneas superiores. Entrando en detalles sobre dichos errores, se aprecian los siguientes:

- Un segmento en el sello que no contiene ningún texto y se puede eliminar.
- Los tres segmentos en la parte superior derecha corresponden en realidad a una mancha y a texto transparentado desde el reverso de la hoja, pudiendo eliminarse también.
- Algunos segmentos sobre el texto del sello superior central no detectan correctamente todo el texto de la línea, creándose varios segmentos para diferentes partes de la misma. Es preciso eliminar algunos (o unirlos) y ajustar un único segmento para la línea completa.
- En el extremo derecho, junto al párrafo principal, hay dos segmentos que corresponden realmente a texto de la página que hay a la derecha de la fotografiada. Por tanto, también deben eliminarse.
- Por debajo de la firma el escribano hay varios segmentos de líneas que también corresponden en realidad a texto del reverso de la página y deben eliminarse.

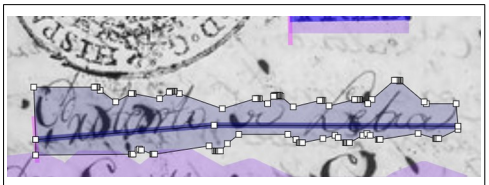

Figura 29: eScriptorium - Edición de segmento de línea

La edición con mayor o menor cuidado por el detalle depende de si tenemos pensado utilizar la segmentación y posterior transcripción como base para generar un modelo con *fine-tuning*. En ese caso conviene ser cuidadoso con la definición de los segmentos y líneas, porque con ello definiremos regiones identificado las líneas manuscritas de cada imagen, lo que a su vez permitirá generar imágenes individuales de cada renglón manuscrito.

Para conocer las herramientas de edición de la segmentación y los atajos de teclado se remite a la documentación citada. El proceso resulta, inicialmente, muy incómodo, manual y propenso a errores, aunque con la práctica se va ganando tiempo. En cualquier caso, es importante remarcar que, con todo este proceso de revisión, se está creando un dataset de entrenamiento para poder hacer *fine-tuning*.

Una vez revisada la segmentación, se puede proceder a la transcripción automática. Como con la segmentación, podemos marcar todas las imágenes o solo algunas, y pulsar el botón Transcribe.

Figura 30: eScriptorium - Transcripción automática

Se abrirá entonces la ventana de diálogo de transcripción, que nos recuerda la importancia de haber revisado previamente la segmentación y nos permite elegir el modelo y la transcripción.

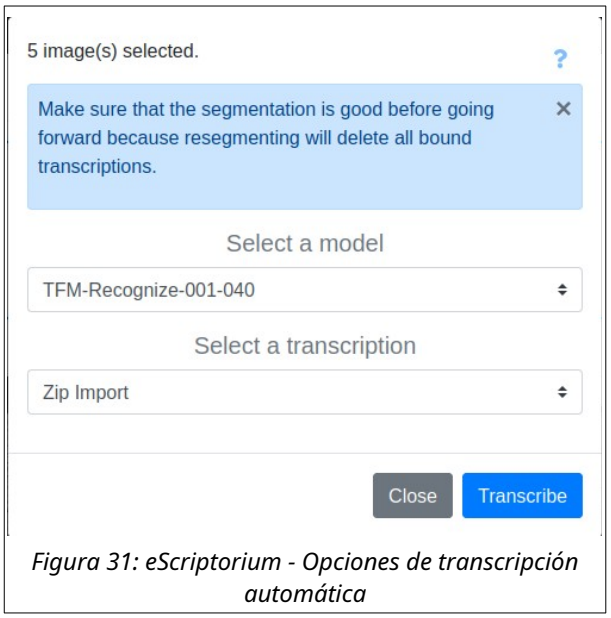

Figura 31: eScriptorium - Opciones de transcripción automática

El modelo nos permite elegir el entrenamiento más adecuado al tipo de documentos que queremos transcribir. Para texto manuscrito en idiomas basados en el latín, especialmente francés, el modelo recomendado es McCATMuS_nfd_nofix. En la imagen se ha elegido un modelo personalizado creado mediante fine-tuning a partir de McCATMuS.

Por su parte, la transcripción es el espacio del documento en el que se va a depositar el resultado. En una primera transcripción elegiremos New o Manual. En la imagen se utiliza Zip Import, que es una transcripción importada de otro documento del proyecto que contenía las primeras 40 páginas de los datos de entrada. De esta forma, se reaprovecha completamente la revisión de la segmentación y de la transcripción previamente hechas.

Una vez concluida la transcripción automática debe revisarse. Al pulsar el botón Edit de la imagen que queramos revisar se accede al mismo panel de revisión que ya se vio para la segmentación. En la imagen siguiente se muestra con los tres paneles disponibles (la imagen original, la versión esquemática y la transcripción de texto), que pueden activarse o desactivarse usando los botones de la esquina superior derecha marcados en la imagen con un recuadro rojo.

El panel del texto permite revisar otra de las características inferidas por la transcripción automática: el orden de lectura. Activando la reordenación, con los botones remarcados en azul, es posible arrastrar y soltar el texto. En la imagen se puede observar que la quinta y sexta líneas tienen su orden intercambiado.

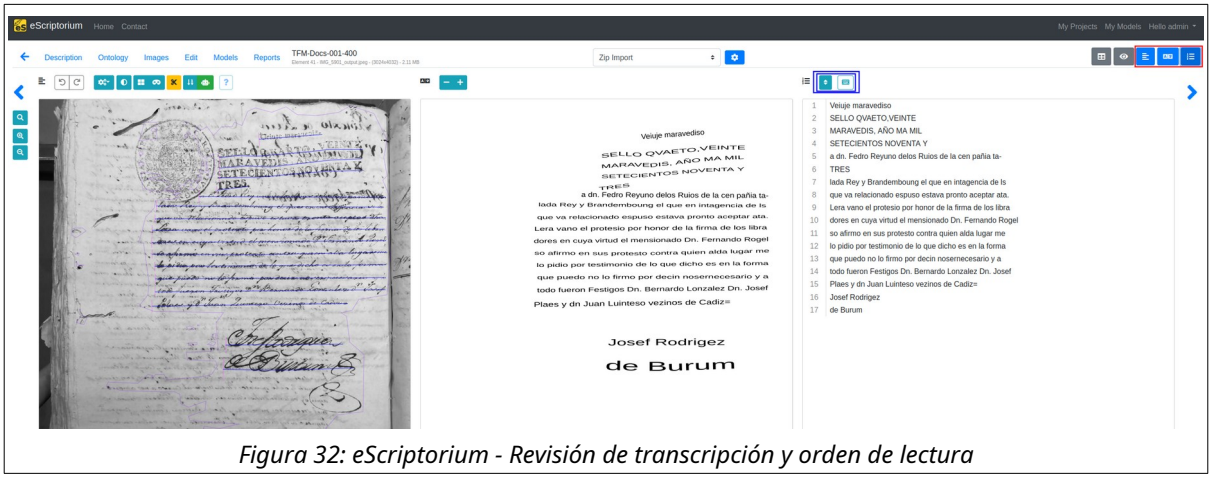

Figura 32: eScriptorium - Revisión de transcripción y orden de lectura

Para concluir con este subapartado, se deben comentar las opciones de entrenamiento y fine-tuning, así como de exportación. En el estado actual de eScriptorium, el entrenamiento no puede realizarse desde la aplicación (ese es el motivo por el que no han publicado una nueva versión de las imágenes Docker que sean compatibles con las versiones recientes del gestor de contenedores).

Por este motivo, es necesario exportar nuestro trabajo en el documento para poder hacer el entrenamiento o *fine-tuning* fuera de eScriptorium. Para ello, accediendo al documento y pasando a la lista de imágenes hay que marcar todas aquellas que se deseen exportar y pulsar el botón Export. En la figura 33 pueden verse las opciones disponibles, que son las siguientes:

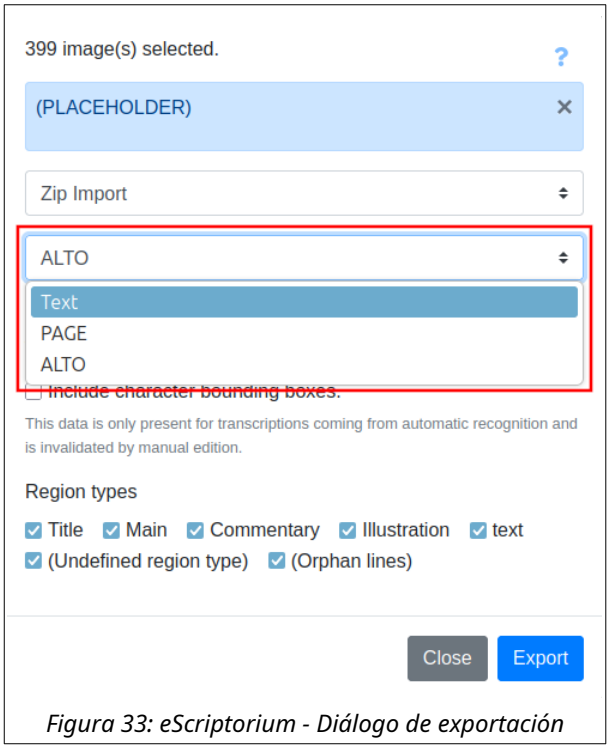

Figura 33: eScriptorium - Diálogo de exportación

- Transcripción que se desea exportar: el espacio en el que estamos guardando

la transcripción y su revisión. Como se comentó anteriormente, en la imagen aparece Zip Import porque se importó la exportación de las primeras 40 páginas.

- Formato de exportación: podemos obtener la transcripción en texto sencillo, con una línea a modo de encabezado que identifica el comienzo de la transcripción de cada imagen incluida. Si queremos usar los datos de la exportación para un entrenamiento o fine-tuning, debemos elegir entonces uno de los formatos XML (ALTO o PAGE), dependiendo de cuál se ajuste mejor a nuestra cadena de herramientas de entrenamiento. Se generará un XML por cada imagen, que contiene no solo el texto, sino la definición exacta de cada región que delimita una línea de texto en la imagen.
- Incluir imágenes: oculto por el desplegable en la figura 33, podemos acompañar la exportación con las imágenes que se han seleccionado, lo que puede resultar útil para compartir la exportación como dataset.

Tras confirmar la exportación, esta se lleva a cabo en segundo plano. Al finalizar el trabajo se obtendrá un mensaje con un enlace para su descarga.

4.3.3 Fine-tuning con Kraken

Como se ha indicado, actualmente no es posible realizar entrenamientos ni *fine-tuning* dentro de la propia aplicación eScriptorium, por lo que es necesario exportar la transcripción en formatos ALTO o PAGE y utilizar directamente Kraken[12], el motor de reconocimiento al que eScriptorium proporciona la interfaz gráfica. Para utilizar Kraken hay que instalarlo como paquete de Python (idealmente creando un entorno virtual de Python primero y ejecutando `pip install kraken`). Las instrucciones detalladas pueden encontrarse en la página principal de Kraken.

Kraken permite entrenar desde cero o hacer fine-tuning de un modelo existente, y lo permite para tres tipos de modelos:

- Modelo de segmentación: segmentación de líneas de texto en imágenes.
- Modelo de reconocimiento: reconocimiento de texto.

[12] https://kraken.re/main/index.html

- Modelo de orden de lectura: orden de lectura de las líneas de texto. Va necesariamente vinculado al entrenamiento de un modelo de segmentación.

La cantidad de opciones disponibles es muy extensa, por lo que aquí simplemente se comentan los entrenamientos realizados, que fueron de tipo *fine-tuning*.

Lo primero para hacer *fine-tuning* es disponer del modelo original. Los modelos se pueden descargar usando el propio ejecutable de Kraken; nuevamente, las instrucciones detalladas para este proceso están en la página inicial de la web de Kraken, por lo que se reproducen aquí las órdenes con las que se descargaron los modelos:

```
$ # List models and retrieve McCATMuS
$ kraken list
Retrieving model list ━━━━━━━━━━━━━━━━━━ ━━━━━━━━━━━━  60% 28/47 0:00:05 0:00:08
10.5281/zenodo.13942714 (pytorch) – Kuzushiji
10.5281/zenodo.13814200 (pytorch) – Segmentation model for historical Samaritan Manuscripts for one column pages, model trained on 13 pentateuchal Samaritan manuscripts
10.5281/zenodo.13788177 (pytorch) – McCATMuS – Transcription model for handwritten, printed and typewritten documents from the 16th century to the 21st century
10.5281/zenodo.13741957 (pytorch) – Model trained on 11th century manuscripts to produce graphematic transcription (Latin).
10.5281/zenodo.13736584 (pytorch) – Model trained on 11th century manuscripts to produce expanded transcription (Latin).
10.5281/zenodo.11113737 (pytorch) – Latin Incunabula and Early Prints
10.5281/zenodo.10886224 (pytorch) – Model train on openly licensed data from HTR-United from the 17th century to the 21st were used.
(…)
$ kraken get 10.5281/zenodo.13788177
Processing ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 16.2/16.2 MB 0:00:00 0:00:01
Model name: McCATMuS_nfd_nofix_V1.mlmodel
$ # Fine-tune recognition model joining McCATMuS and 40 first pages
$ # of Protestos
$ find path/to/xml/directory/ -type f -name “*.xml” >xml_files.txt
$ ketos compile -F xml_files.txt -f alto
$ ketos train -f binary -d cuda:0 -B 4 -r 0.0002 --workers 4 --threads 16 --resize both -i /path/to/McCATMuS_nfd_nofix_V1.mlmodel -o train_model/ dataset.arrow
Trainable params: 4.0 M
Non-trainable params: 0
Total params: 4.0 M
Total estimated model params size (MB): 16
Modules in train mode: 40
```

```
Modules in eval mode: 0
stage 0/∞ ———————————— 282/282 0:00:13 •    20.37it/s val_accurac… early_stoppi…
                                0:00:00                0.851        0/10 0.85101
                                                       val_word_ac…
                                                       0.520
(…)
stage 26/∞ ———————————— 282/282 0:00:14 •    19.81it/s val_accura… early_stoppi…
                                 0:00:00                0.913       10/10 0.91882
                                                        val_word_a…
                                                        0.691
Moving best model train_model/_16.mlmodel (0.918818473815918) to
train_model/_best.mlmodel
$ # Fine-tune segmentation model joining blla.model and 40 first pages
$ # of Protestos
$ ketos segtrain -d cuda:0 -f alto -t output-01-40.txt --resize both -
i blla.mlmodel -o seg_model/
Training line types:
  default   2      1253
Training region types:
  text      3      40
(…)
Trainable params: 1.3 M
Non-trainable params: 0
Total params: 1.3 M
Total estimated model params size (MB): 5
Modules in train mode: 39
Modules in eval mode: 0
stage 0/50 ———————————————— 36/36 0:00:23 • 0:00:00 1.58it/s val_accuracy:
                                                          0.984
                                                          val_mean_acc:
                                                          0.984 val_mean_iu:
                                                          0.503 val_freq_iu:
                                                          0.928
(…)
stage 49/50 ———————————————— 36/36 0:00:23 • 0:00:00 1.57it/s val_accuracy:
                                                           0.981
                                                           val_mean_acc:
                                                           0.981
                                                           val_mean_iu:
                                                           0.552
                                                           val_freq_iu:
                                                           0.916
```

Una vez generados los modelos con *fine-tuning*, pueden importarse desde la interfaz de eScriptorium accediendo a *My Models*, en la esquina superior derecha. Ahí se pueden encontrar los modelos cargados, ya sean del repositorio proporcionado por Kraken o generados mediante *fine-tuning*, y ya sean de reconocimiento o de segmentación.

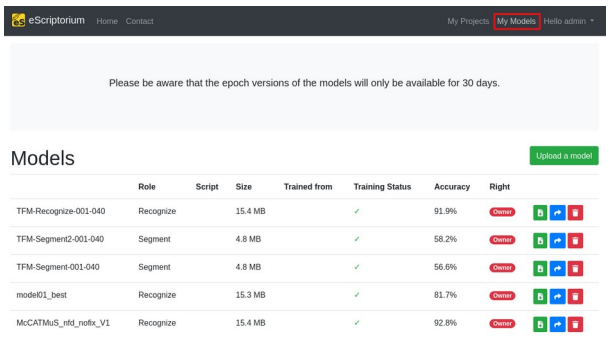

Figura 34: eScriptorium - Gestión de modelos

En las pruebas realizadas con fine-tuning de segmentación los resultados no fueron buenos, como puede verse en la figura 35. Otras áreas de la página exhibieron un comportamiento similar, siendo peor la segmentación con *fine-tuning* que la estándar. Por este motivo, no se ha explorado más el entrenamiento de la segmentación y, como el del orden de lectura exige ajustar el modelo de segmentación, tampoco se ha intentado ajustarlo.

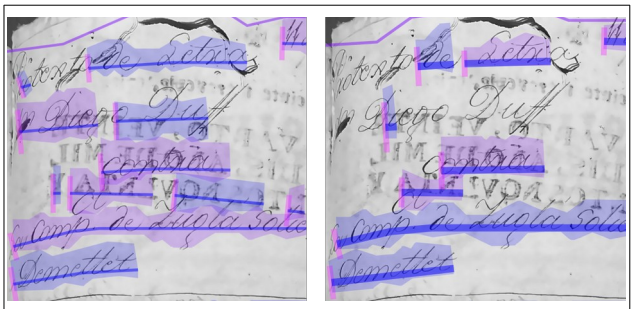

Figura 35: eScriptorium - segmentación estándar (izda.) vs. fine-tuned (dcha.)

Respecto al reconocimiento, se ha transcrito automáticamente una página utilizando el modelo McCATMuS sin ajustar, luego se ha transcrito con el modelo ajustado con la transcripción revisada de las 40 primeras páginas, y tras revisar la transcripción se han comparado las dos transcripciones automáticas con la revisada utilizando una calculadora online de distancia Levenshtein. La tabla 4 resume la comparación entre ambos modelos (en ella, *t.c.* significa *texto corregido*).

	McCATMuS	McCATMuS ajustado
Muestra de transcripción	Cnl Cicidad de Cadir à oein. tey fair diar defmer de Mar r0 del ans de mibsereuentes noventa y trie Ante mi et e Bar xipts Ce xivanr Publi ce 4 mayon del Cavitde Aguntamier 10 de crta Plara, y taitigos vare " cie S^nr Siego Dust de ou Sécin da ner n comercis aguien doy le conoiir, y me exciois iente decra primesa de l'am? bio con iu dequnda de equat tenor Everitai en Bdionen Enangero que traduridar al Cartellans con los Endosos que la pimera contiene du con terto'er et que vique	En lo ciudad de Cadiz a vein. Cestesto de Lotra te y Seis dias del mes de Mar zo del año de ml setecientos Dn. Diego Dieff. noventa, y tres Ante mi el Y- Comtra hascripto Escrivano Publico y mayor del Cavildo Ayuntamien- Dn. Felipe Sabar y to de esta Plaza, y tatigos pare Compañia cio Dn. Diego Duff de su Vecinda rio y comercio a quien doy le conosio, y me excivio una Letra primera de Cam- bio con su Segunda de egual tenor Eseritas en Ydioma Extrangero que traducidas al castellano con los endosos que la primera contiene su con testo es el que vigue-
Caracteres (t.c.: 1199)	1191	1190

Dist. Levenshtein con t. c.	340	188
Similitud	72,49 %	84,79 %
CER (dist. Lev. / long. t.c.)	0,28	0,16

Tabla 4: Efecto de fine-tuning en eScriptorium/Kraken

El siguiente apartado ofrece más datos sobre el CER estimado cuando se aprovechan las capacidades de fine-tuning en eScriptorium / Kraken.

4.3.4 Resultados (CER)

eScriptorium no ofrece un valor de CER para sus transcripciones. En su lugar ofrece una métrica llamada *Average transcription confidence* con valor entre 0 y 1, siendo los valores más altos mejores. Tras importar las 40 primeras páginas, que se configuraron en un documento de eScriptorium aparte y se habían segmentado, transcrito y revisado previamente en el documento, el valor era 1, ya que la aplicación considera que la importación es la transcripción fiel y, por tanto, no contiene error alguno.

Tras transcribir las 5 páginas seleccionadas en la figura 30 y corregirlas manualmente, el valor de *Average transcription confidence* bajó al 0,98. Usando la misma calculadora online de distancia Levenshtein del apartado anterior, que ofrece también la similitud entre la cadena original y la revisada, el valor medio fue de 95 %. Este valor puede considerarse el complemento de CER, lo que sugiere un CER del 5 %, que es un cifra competitiva con las obtenidas por las arquitecturas CRNN sobre datasets como IAM, RIMES o READ 2016. No obstante, no hay que olvidar que la muestra sobre la que se ha hecho el cálculo es muy pequeña y que las imágenes de esa muestra, al pertenecer a la misma escribanía, probablemente han sido escritas por personas que también escribieron algunas de las 40 páginas previamente transcritas y revisadas.

Hay que indicar que estas cifras se han obtenido con la versión de las imágenes tratada con uno de los scripts que se documentan en el próximo capítulo. Las imágenes originales en color ofrecieron mucho peor resultado, comenzando por una segmentación muy deficiente. Las imágenes tratadas también se han usado en las evaluaciones de Tesseract y OCR4All.

Capítulo 5 - Desarrollos en Python para HTR

En el capítulo anterior se examinaron algunas aplicaciones de código abierto para OCR/HTR y se concluyó una candidata que ofrece buenos resultados, aunque para ello es preciso realizar cierto preprocesamiento de las imágenes que componen los datos de entrada facilitados para este TFM. Además de eso, implica dos circunstancias:

- La revisión manual, especialmente de la segmentación, requiere mucho tiempo del usuario.
- Por la situación actual del proyecto, la instalación y puesta en marcha no constituye una experiencia sencilla, sino que son precisos ciertos conocimientos técnicos.

Por otro lado, la interactividad permite corregir fácilmente cualquier deficiencia en la que incurran los procesos automáticos. Cabe preguntarse qué resultados se pueden obtener con un flujo totalmente automatizado y si la previsible menor calidad derivada de la ausencia de una revisión individualizada compensa el ahorro de tiempo obtenido.

En este capítulo nos centraremos en dos aspectos:

- Qué transformaciones son necesarias en las imágenes y cuáles de ellas pueden hacerse de manera automatizada con un resultado adecuado.
- Si una red neuronal orientada a la transcripción automática, especialmente entrenada para el conjunto de datos de entrada, puede obtener resultados similares o mejores que eScriptorium/Kraken.

5.1 Preprocesamiento de imágenes

La figura 36 muestra un mosaico con tres imágenes del conjunto de entrada en su formato original (redimensionadas y con dos líneas rojas para ayudar a distinguir los límites entre ellas). En el capítulo 3 ya se detallaron las características de las imágenes y el impacto que esas características, o deficiencias, podían tener. Ahora se van a ir enumerando muchas de esas deficiencias, junto con las técnicas que se han probado para reducir la afectación que suponen al proceso de segmentación y reconocimiento de texto. Se indicará si las técnicas han funcionado o no, siempre

teniendo en cuenta que las técnicas se aplicarían al conjunto, no individualmente, por lo que debe ser el propio programa el que pueda tomar la decisión de los parámetros que se tienen que aplicar para cada técnica e imagen. Para el tratamiento de las imágenes se ha usado la biblioteca OpenCV, junto con NumPy y Pandas.

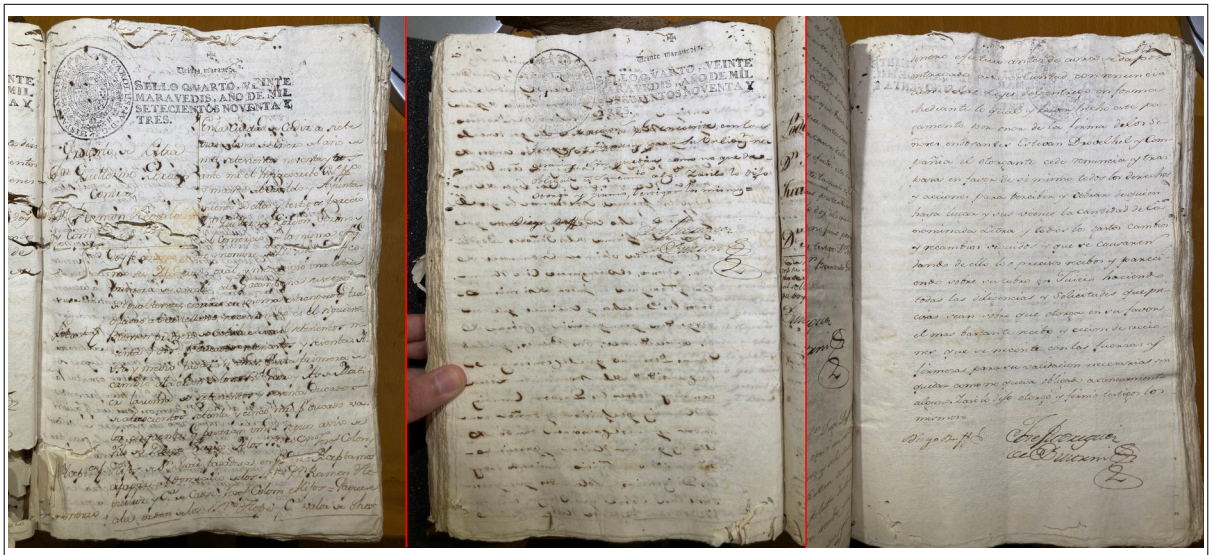

Figura 36: Tres ejemplos de imágenes del conjunto de entrada

5.1.1 Márgenes exteriores y encuadre de página

Si se observan de nuevo las imágenes de la figura 36, puede verse que las tres imágenes no contienen exactamente la página que se pretende fotografiar, sino que en torno a la página siempre hay un contenido adicional. Ese contenido puede ser parte del escritorio en el que se apoya el libro con los documentos fotografiados, una porción de la página contigua a la que se quiere fotografiar o incluso los medios utilizados para mantener estable el libro durante la fotografía.

Si las imágenes se pudieran recortar automáticamente a los límites exactos de la página concreta fotografiada, se conseguiría como beneficio eliminar los fragmentos de la página contigua que se incluyen a menudo, y por tanto el texto que contienen.

Si consideramos que las imágenes son, para una computadora, una matriz de valores, y que en la escala de grises o en el espacio de color de los colores primarios (rojo, azul, verde) los valores más altos corresponden a puntos más brillantes, podemos pensar que, seguramente, todos los puntos que componen la recta que delimita un borde de la página tendrán valores parecidos y muy diferentes a los de los puntos que les rodean.

En una imagen real que no esté perfectamente encuadrada, la línea no será recta y, además, habrá sombras junto a los bordes, pero el cambio más o menos abrupto de valores en la matriz existirá, sin duda La figura 37 muestra una porción de una matriz donde hay una esquina de una página con los valores numéricos y el tono (en escala de grises, por simplicidad) que ofrecería a la vista. Con técnicas matemáticas (gradientes) se puede binarizar la imagen y conseguir detectar dónde están los bordes.

255	255	255	255	255
254	200	190	201	205
252	180	160	120	80
253	100	40	32	22
254	80	20	60	45
250	64	22	68	120
255	64	25	50	160
254	64	23	48	161
253	64	21	52	158

Figura 37: Representación de la esquina de una página en la matriz de la imagen

Una de esas técnicas es el algoritmo de detección de bordes Canny [43], que está implementado en OpenCV. La idea consiste en obtener la versión de la imagen binarizada invertida, con los bordes detectados en blanco sobre fondo negro. A continuación se aplica la función findContours de OpenCV, que devuelve una colección de contornos y se obtiene la lista de los 10 que contienen una mayor área. De ellos, el primer contorno con cuatro esquinas debería corresponder al de la página. La figura 38 muestra el resultado aplicado a la página del centro del mosaico de la figura 36. El punteado en rojo debería identificar el mayor contorno de cuatro esquinas encontrado. Pero ninguno de los contornos detectados está cerrado, ni mucho menos tiene cuatro puntos.

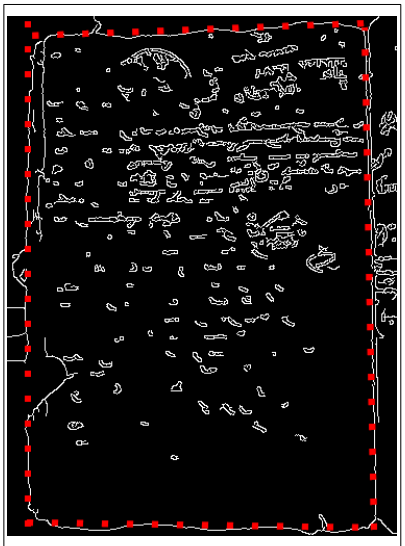

Figura 38: Resultado de algoritmo Canny

Sería necesario refinar el algoritmo genérico para extrapolar los segmentos más largos encontrados, obtener mediante regresión la recta aproximada y extrapolada y conseguir así encontrar el rectángulo. Por limitaciones de tiempo no ha sido posible explorar esta posibilidad y, por tanto, no se pudo aplicar esta técnica y se descartó el encuadre de página.

5.1.2 Imagen en color

Convertir una imagen en color a escala de grises no solo es sencillo desde el punto de vista matemático (en el espacio de colores primario basta con sumar los valores de los tres canales y dividir el resultado entre tres), sino que OpenCV permite hacerlo al abrir el fichero indicando un parámetro:

```
image = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
```

El resultado es apropiado, por lo que esta operación se hace por el script durante el preprocesamiento de las imágenes.

5.1.3 Contraste y brillo variables en la colección y en cada página

No todas las imágenes de la colección están tomadas con exactamente el mismo ajuste de contraste y brillo. Algunas de ellas no requieren ningún ajuste, pero otras tienen un contraste excesivo y otras son demasiado tenues. Además, para una imagen puede existir una diferencia de iluminación entre los extremos de la misma (más frecuentemente en el eje vertical) derivados del diferente ángulo de incidencia de la luz al tomar la foto.

El objetivo de mejorar el contraste y el brillo es reducir ambas deficiencias, inevitables por otra parte si no se dispone de equipo especializado para la captura. Para ello, se han probado tres técnicas: la ecualización por histograma, el algoritmo CLAHE y la normalización de valores en rango. Los tres están implementados en OpenCV como funciones que operan, de nuevo, matemáticamente sobre los valores numéricos que componen la imagen.

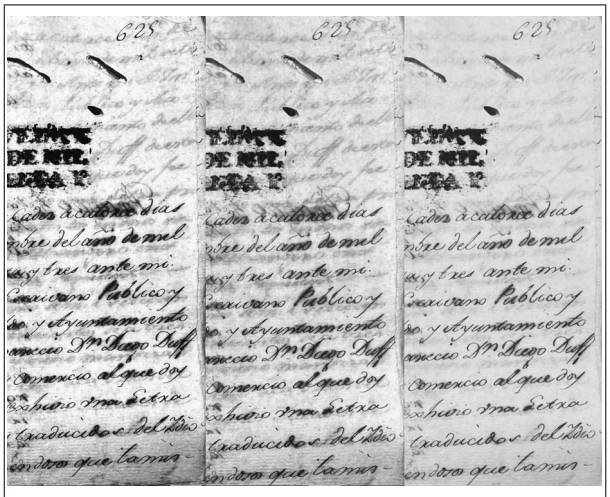

Figura 39: Mejora de contraste y brillo (ecualización por histograma, CLAHE y normalización)

La figura 39 muestra el resultado de cada uno de ellos. Como se ve, la ecualización por histograma no elimina, sino que más bien amplifica la diferencia de billo a lo alto de la imagen por la exposición durante la captura, mientras que la

normalización obtiene un tono más homogéneo (con el beneficio añadido de eliminar algo de ruido). Se ha optado por la normalización, aunque el script permite elegir usar CLAHE si se desea, que ofrece resultados intermedios.

5.1.4 Ruido en la imagen

El ruido, entendido como manchas de suciedad, deterioro del soporte papel o vislumbramiento del texto escrito en el reverso de la página fotografiada, también puede ser interpretado matemáticamente. En la matriz de valores numéricos que compone la imagen el ruido se distingue porque es una concentración de valores muy bajos (al ser manchas oscuras) en un área pequeña[13] y que no sigue un trazo continuo. La eliminación de ruido se hace con el algoritmo de eliminación de ruido por media no local, utilizados en [25] y [27]. Aunque la ganancia es marginal, la operación es rápida y nunca empeoró la calidad.

5.1.5 Perfilado de los trazos

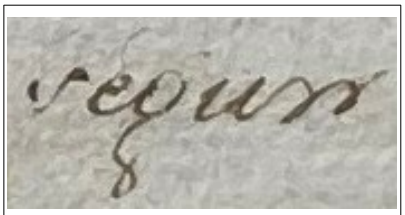

Figura 40: Trazos ausente y ligero en s y g, respectivamente

Al escribir, especialmente con pluma, como en las imágenes del conjunto de entrada, a veces se hace más presión sobre el papel y a veces menos. También puede haber más o menos tinta en la pluma; todo ello da lugar a que el trazo ocasionalmente sea muy ligero y casi inapreciable, como puede verse en la figura 40 (repetición de la figura 12). El ser humano ha aprendido a inferir esos trazos ausentes, siempre que no sean muy notables, de manera inconsciente. No es el caso de las computadoras.

La combinación de los algoritmos de erosión y dilatación permite mejorar los trazos de la escritura manuscrita. La erosión reduce ligeramente el grosor de los trazos, eliminando las imperfecciones, mientras que la dilatación restaura el grosor previo, sin volver a reproducir las imperfecciones. Para ello se recorre la matriz examinando a la vez un pequeño subconjunto de puntos contiguos, aplicando una operación matemática sobre todos ellos. Al subconjunto se le conoce como *kernel*.

[13] Las marcas de deterioro y los textos del reverso a menudo no forman *áreas pequeñas* y, por ello, no se pueden eliminar correctamente.

Se probó con diferentes valores de *kernel*, pero se comprobó que el resultado no era óptimo. En la figura 41 se ve que el trazo ausente de la letra b en el lado izquierdo no mejora; en cambio, las oquedades de las letras d, e y l se rellenan, lo que se consideró perjudicial para el procesamiento posterior.

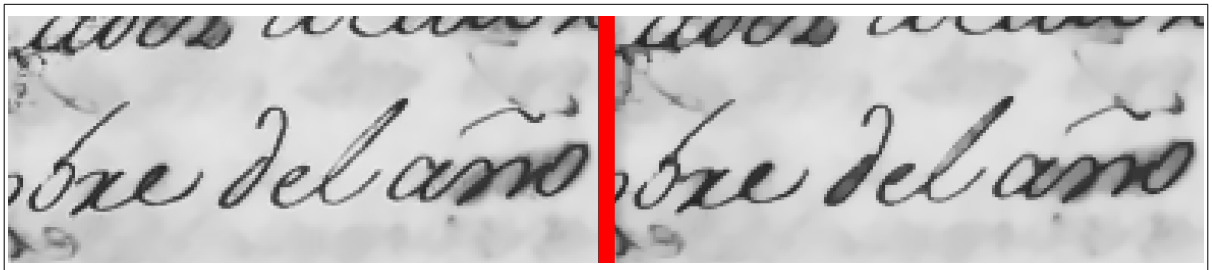

Figura 41: Erosión y dilatación (a la izquierda, antes de aplicarlas; a la derecha, tras hacerlo con un kernel 3)

No obstante, el script incluye una opción para aplicarlo con un *kernel* de 3x3.

5.1.6 Información no relevante y supresión mediante binarización

En una página manuscrita como las del conjunto de entrada el ser humano solo necesita prestar atención al texto escrito. Una vez enfocada la imagen para centrarse en los límites de la página, el cerebro puede descartar el tono más o menos amarillento del papel y las manchas. Solo necesita centrarse en identificar dónde hay tinta y dónde hay papel en blanco para construir los trazos de la escritura y, con ellos, las letras, las palabras y las frases. Esa simplificación conceptual de la información de la página se hace en la computadora mediante la binarización, que es la conversión de la escala de grises a únicamente dos valores, de forma que todos los píxeles sean o completamente blancos o completamente negros.

Para realizar la binarización se han probado dos opciones: umbral adaptativo gaussiano, y binarización mediante el algoritmo de Otsu. Los resultados son los de la figura 42. Como puede verse, ninguno de los dos mejora la imagen; el umbral adaptativo gaussiano consigue unos trazos muy claros, pero también amplifica las manchas y el ruido de la imagen, mientras que la binarización de Otsu elimina muchos trazos, por lo que no se ejecuta por defecto en el script, aunque se incluye como opción.

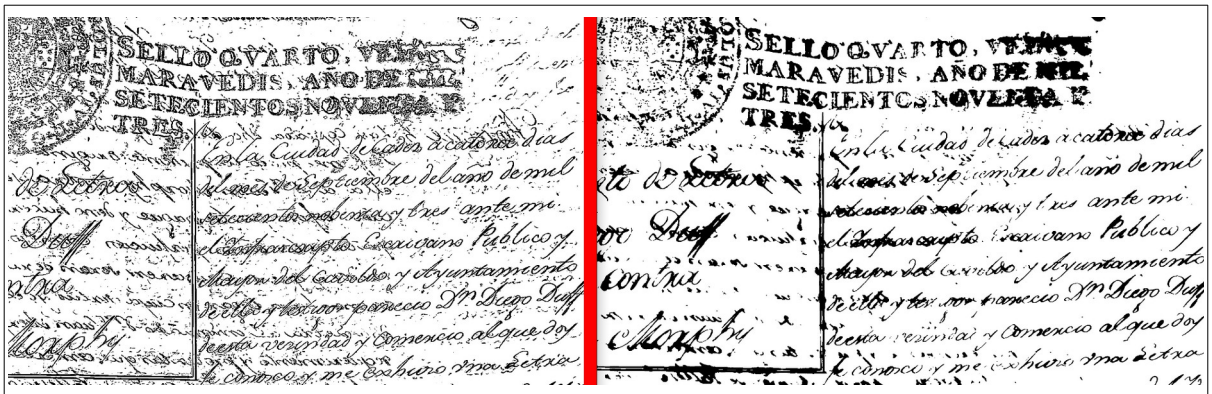

Figura 42: Binarización gaussiana (izquierda) y Otsu (derecha)

El script, llamado `image_preprocessing.py`, que agrupa todas las técnicas de tratamiento descritas hasta aquí que se han considerado potencialmente útiles, permite procesar una sola imagen o un directorio completo. Las opciones disponibles en el script son estas:

```
./image_preprocessing.py -h
usage: image_preprocessing.py [-h] [-i INPUT] [-I INPUT_DIR] [-o OUTPUT]
                              [-O OUTPUT_DIR] [-c] [-b] [-e]

options:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        path to single input image
  -I INPUT_DIR, --input_dir INPUT_DIR
                        path to folder containing input images
  -o OUTPUT, --output OUTPUT
                        path to single output image
  -O OUTPUT_DIR, --output_dir OUTPUT_DIR
                        path to folder containing output images
  -c, --clahe           use CLAHE to enhance contrast and brightness (slower
                        and better)
  -b, --binarize        binarize image
  -e, --erode_dilate    apply erosion and dilation using a 3x3 kernel size
```

5.2 Segmentación

Para la segmentación de las imágenes de entrada en las líneas de texto que contiene se intentó usar la funcionalidad proporcionada por OpenCV (findContours, boundingRect), pero el resultado era muy confuso, como se puede ver en la imagen de la figura 43, por lo que la vía algorítmica quedó descartada. Puesto que no se dispone de páginas suficientes en los datos de entrada que estén previamente segmentadas, no se ha explorado la posibilidad de crear una red

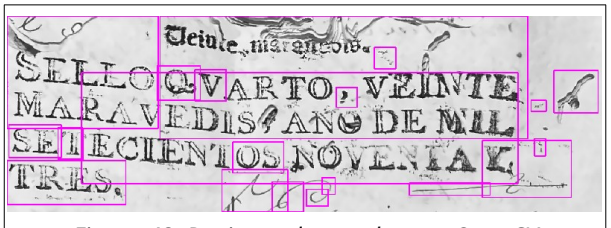

Figura 43: Regiones detectadas por OpenCV

neuronal que aprenda a segmentar correctamente. Como se comentó en el capítulo anterior, incluso realizar *fine-tuning* sobre el modelo de segmentación previamente entrenado de Kraken no condujo a obtener mejores resultados que con el modelo estándar ya incluido. Por todo lo anterior, se considera que la mejor opción actualmente es utilizar Kraken, bien a través de eScriptorium, bien directamente invocando a Kraken por la línea de órdenes, para realizar la segmentación y obtener los ficheros en formato ALTO o PAGE.

Una vez en ese formato, se ha preparado un script Python que lee los archivos XML y las imágenes asociadas de páginas completas y extrae cada línea a partir de la información de segmentación de cada archivo XML, depositándolas en un directorio junto con un archivo TXT que enlaza cada imagen de línea de texto con la transcripción correcta, lo que facilita la prepración de los datasets.

5.3 Entrenamiento con CRNN – Biblioteca MLTU

Durante la búsqueda de información sobre la construcción de redes neuronales con Python, se encontraron diversos tutoriales. Entre ellos se descubrió uno [44] basado en Tensorflow y Keras, especialmente completo porque encapsula en una biblioteca diversas utilidades como el aumentado de datos, el ajuste de la tasa de aprendizaje, el guardado automático y otras posibilidades. Para la explicación del tutorial se utiliza el mencionado dataset IAM, obteniendo unos valores de CER y WER de 0,0438 y 0,1451, respectivamente.

La arquitectura que usa para ello es bastante compleja, con hasta 9 bloques de tipo *residual block* seguidos de dos bloques BLSTM y una capa de salida. Cada *residual block* está formado por dos capas convolucionales, seguida cada una de una capa de normalización. Como función de activación se usa LeakyReLU. En paralelo, otra capa convolucional recibe la entrada del bloque y su salida se suma a la de la segunda capa convolucional del bloque justo antes de la segunda función de activación. La figura 44 muestra la estructura, con las capas que forman el *residual block* rodeadas por la línea punteada.

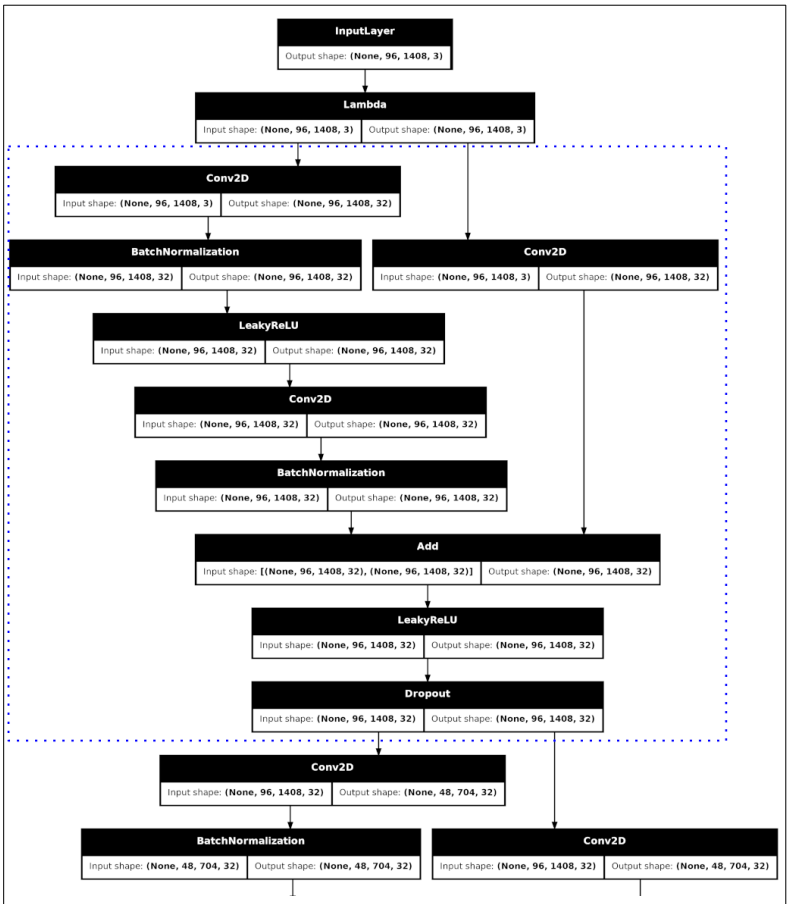

Figura 44: Residual block implementado en biblioteca MLTU

Se decidió comenzar el proceso entrenando directamente la arquitectura con la transcripción de las primeras 40 páginas de los datos de entrada, segmentados por líneas, lo que ofrecía un total de 1253 elementos.

Las modificaciones necesarias para procesar el dataset respecto al de IAM fueron bastante pequeñas, ya que en ambos casos la correspondencia entre imágenes y transcripciones se guardaron en un fichero de texto. No obstante, una vez concluido el entrenamiento surgieron varias dificultades, debido en parte a que el tutorial tenía unos dos años de antigüedad y usaba versiones de Tensorflow y Keras diferentes a las actuales, que son las que se decidió utilizar para este trabajo. En concreto:

- El modelo se guardaba en formato H5, que en las versiones actuales de Keras solo conserva los pesos, no la arquitectura.

- Las clases de la biblioteca MLTU que implementa la función de pérdida CTC y las de métricas CER y WER no estaban convenientemente anotadas para ser persistidas al grabar modelo en formato Keras.
- El modelo final se exportaba a formato Onnx, pero actualmente ese formato no es capaz de recuperar capas con la operación CudnnRNNV3.

Los dos primeros puntos se han corregido localmente y se va a preparar un Pull Request para su consideración por el autor de la biblioteca. El tercero no se ha podido solucionar por el momento, ya que todas las soluciones propuestas que se han encontrado no han funcionado (el modelo se guarda en formato ONNX, pero luego no se puede recuperar). En su lugar, se hace la inferencia cargando el modelo en formato Keras guardado al final del entrenamiento.

Todo lo anterior impidió durante más de un mes poder probar la validación e inferencia del modelo entrenado. Cuando por fin se hizo, quedó claro que el dataset de 1253 elementos es excesivamente diminuto para poder hacer un entrenamiento desde cero, dada la complejidad de la tarea[14].

Se ha entrenado la red desde cero también con el dataset Rodrigo y los resultados de CER y WER son considerablemente más altos que con IAM, resultando 0,1644 para CER y 0,5094 para WER. Hay que indicar, no obstante, que el entrenamiento con IAM alcanzó 1.000 épocas, mientras que con Rodrigo, por cuestión de tiempo, se fijó un máximo de 500 épocas. En ambos casos, la tasa de aprendizaje se fijó en 0,0005, con una reducción de 0,9 tras 20 épocas sin mejora y un tamaño de lote de 32.

También se ha intentado entrenar la red con el dataset READ 2016, pero durante el progreso de la primera época, el valor de CER, que comienza descendiendo, pasa a ser inf(inito) y el entrenamiento ya no progresa. En el Apéndice B -

[14] Honestamente, se pensó que el CER de 2,73 era bastante bueno (los artículos académicos mencionan cifras del 5 % como aceptables), hasta que al poder hacer la validación y ver las predicciones se comprendió que el 2,73 no era un porcentaje y que, de haberlo sido, la cifra habría sido 273 %.

Entrenamiento con READ 2016 se muestra el resultado y la explicación encontrada al respecto.

Sobre los modelos entrenados con IAM y Rodrigo se hizo una validación de inferencia con 126 imágenes del conjunto de datos de entrada, con resultados que imposibilitan utilizar los modelos tal cual están con ellos. Todos los datos mencionados se resumen en la tabla 5.

	IAM	Rodrigo
Número de épocas	1000	500
Batch size	32	32
Tasa de aprendizaje inicial	0,0005	0,0005
Reducción tras 20 épocas sin mejora	0,9	0,9
Uso de RAM GPU	88 %	51 %
CER	0,04	0,15
WER	0,16	0,51
CER con datos de entrada	0,68	0,86
WER con datos de entrada	1,02	1,01

Tabla 5: Datos de entrenamiento e inferencia de datasets populares

5.4 Entrenamiento con CRNN – Keras y Tensorflow

Durante las dificultades para solventar los problemas descritos en el apartado anterior con el modelo que usa la biblioteca MLTU, se preparó un script alternativo que permite probar varias arquitecturas diferentes para comparar resultados. Esto se hizo antes de confirmar que el problema residía en la escasez de los datos disponibles para realizar un entrenamiento desde cero. No obstante, por considerarse que puede resultar de utilidad, se incluye en el repositorio asociado a este trabajo.

Se han implementado tres arquitecturas:

- La propuesta por el tutorial de Keras Handwriting recognition. Se trata de una arquitectura muy sencilla con dos bloques convolucionales y dos capas recurrentes bidireccionales LSTM.

- La propuesta en la tesis de Harald Scheidl [30]. Es sustancialmente igual a la anterior, pero con cinco capas convolucionales.
- La propuesta en el artículo Best Practices for a Handwritten Text Recognition System, de Retsinas et al. [45] . Es una arquitectura más compleja, con varios bloques *residual block*, y tres capas recurrentes bidireccionales LSTM.

5.5 Ubicación del repositorio asociado a este trabajo

El repositorio con los scripts mencionados en este capítulo se encuentra en GitHub, en la siguiente dirección:

```
https://github.com/RickieES/HTR-HNPD
```

El contenido que debe considerarse alineado con esta memoria está marcado con la etiqueta v0.1.

Capítulo 6 - Conclusiones y trabajo futuro

6.1 Conclusiones

Uno de los apartados del plan de trabajo consistía en evaluar el rendimiento de algunas aplicaciones populares de código abierto en el ámbito de OCR y HTR. En este sentido se han confirmado las limitaciones de algunas de ellas para tratar con textos manuscritos antiguos, y se ha determinado que eScriptorium es una de las que mejores resultados consigue, incluso a pesar de las actuales dificultades que supone instalarla y hacerla funcionar si no se tienen conocimientos avanzados en gestión de contenedores.

La evaluación de los datos de entrada ha permitido identificar las dificultades que plantean las imágenes adquiridas para su tratamiento, desde los elementos en la imagen completa que sobrepasan el contenido concreto de la página que se quiere recoger en ella (márgenes, contrapáginas), hasta las manchas y partes de la página deterioradas, pasando por la especial disposición del texto, que obliga prácticamente siempre a reordenar el texto transcrito automáticamente. También se han presentado diversos mecanismos con los que intentar correcciones masivas y desatendidas de las imágenes, explicando cuáles de ellos proporcionan buen resultado y cuáles no, implementando un script en Python que puede aceptar un directorio de entrada con imágenes y procesarlas de manera automatizada.

Se ha concluido que otro de los factores importantes para llevar a cabo HTR en imágenes que representan páginas completas es la segmentación de las mismas en imágenes más pequeñas que contengan, cada una, una sola línea manuscrita. Se ha confirmado que las soluciones algorítmicas no resuelven esta tarea adecuadamente y que los mejores resultados se obtienen con arquitecturas de redes neuronales entrenadas para la tarea, entre las que los modelos proporcionados con Kraken proporcionan resultados relativamente eficientes. No puede dejar de apuntarse aquí, sin embargo, que las arquitecturas Transformer hacen innecesaria la segmentación. Desafortunadamente, no parece que existan datasets etiquetados suficientes que permitan entrenar ese tipo de redes para obtener buenos resultados por el momento.

Diversas pruebas recogidas en la tabla 5 muestran que los modelos entrenados en datasets populares utilizados no han resultado útiles para obtener transcripciones automáticas de los datos de entrada tal cual. Aunque uno de los datasets utilizados está en idioma español, al corresponder a castellano antiguo y tener un tipo de letra muy diferente del utilizado en los protestos de letras de cambio, no pudo aprovecharse para obtener transcripciones automáticas mínimamente válidas de los datos de entrada. Sin embargo, la aplicación eScriptorium y el modelo McCATMuS, tras recibir un ajuste fino con el primer subconjunto de 40 páginas transcritas y revisadas, consiguió proporcionar un CER estimado del 5 %, lo que sugiere que la vía de mejora pasa por realizar *fine-tuning* de modelos lo más extensos posibles previamente entrenados y, probablemente, aprovechar los avances en la revisión de la transcripción para seguir mejorando el conocimiento del modelo paulatinamente.

Con todo, quizá la principal conclusión obtenida hasta el momento es que los datasets utilizados para evaluar los avances en las arquitecturas de redes neuronales para HTR pueden considerarse *buenos samaritanos* a la hora de someterse a las pruebas, al menos si se comparan con los datos de entrada reales con los que se ha trabajado. Permiten obtener unas métricas de eficiencia razonables, incluso sorprendentes para alguien que observa por primera vez cómo un sistema es capaz de leer texto escrito a mano. Pero cuando se intenta extrapolar la inferencia a datos totalmente ajenos al dataset con el que se ha entrenado el modelo, los valores obtenidos son radicalmente más altos. Es cierto que las imágenes que componen los datos de entrada tienen muchas características (en diversidad y cantidad) cuando se comparan con las de los datasets, pero no son debidas en general a deficiencias en la captura de la imagen en sí, sino a la propia naturaleza de los documentos históricos. Cabe recordar que estamos tratando con documentos que aún no han cumplido 250 años, y que seguramente no serán los que en peor estado se vayan a encontrar en el ámbito de la investigación histórica. Si bien las notas de actas de sesiones del consejo que componen el dataset READ 2016 tienen algunos aspectos en común con los documentos utilizados en este trabajo, aún se encuentran claramente en mejor estado de conservación y presentan una configuración del texto en la página bastante homogénea.

Por ello, sería conveniente componer y publicar más datasets con contenido histórico, debidamente etiquetados, para permitir estudiar cuáles son las expectativas reales con documentos con cierto nivel de deterioro y falta de claridad.

6.2 Trabajo futuro

Hay varias vías por explorar para tratar de mejorar los resultados documentados en este trabajo:

- Aprovechando que se conoce la disposición general de las imágenes, se puede tratar de implementar un algoritmo de detección de bordes específico que consiga realizar el encuadre automático de las páginas.
- Se pueden realizar más entrenamientos con las arquitecturas modelo ajustando hiperparámetros.
- Se deben implementar mecanismos de *fine-tuning* una vez se consigan buenos resultados con los datasets populares. Aunque sea necesario realizar la transcripción semiautomática con eScriptorium de un porcentaje importante de las imágenes facilitadas, se obtendrá a cambio un buen dataset apto para *fine-tuning*.
- No se ha facilitado un script que haga un proceso en lote de inferencia sobre un modelo que se haya verificado que ofrezca buenos resultados, pero una vez se consiga tener un modelo así, es un paso obvio para acelerar la transcripción.
- Cabe la posibilidad de implementar un postproceso del resultado de la transcripción que implemente corrección ortográfica utilizando diccionarios ortográficos, lo que permitiría corregir muchos de los errores que no consiga resolver la red neuronal. Esto es una posible estrategia de decodificación CTC para la inferencia, pero lo que se propone es realizarla fuera del entrenamiento. Plantea como dificultad, no obstante, que los diccionarios suelen ser de español actual, mientras que algunas palabras que forman parte de la transcripción de los datos de entrada son arcaísmos que quizá no se incluyan en los diccionarios ortográficos.

Chapter - Introduction

Motivation

Data processing in a digital way is essential nowadays. While data generated in the last 30-40 years is usually natively digital, older data was often originally analog [1] and must be converted into digital form. A common example is converting text available only on paper—whether newspapers, books, printed reports, or handwritten letters. Several challenges arise in this transformation process, from capturing the visual representation as a digital image to recognizing the symbols within it.

The invention of scanners in the mid-1950s [2] and, later, digital photography marked the first step, enabling computers to access images as a stream of bytes. Somewhat surprisingly, the second part, recognizing text on paper, is a task that began in the early 20th century [3]. This initial effort was not part of digitalization processes but rather aimed to assist visually impaired individuals. Early advances in this area took the form of mechanical devices [4] and were primarily focused on printed text. Significant improvements occurred with the use of conventional programming techniques, leading to the development of OCR (Optical Character Recognition) and HTR (Handwritten Text Recognition) in the 1960s and 1970s. However, traditional rule-based and heuristic programming proved to have significant limitations.

The shift in the 1980s and 1990s from rule-based programming to machine learning enabled significant progress in handwritten text recognition [5] [6]. Advances in neural networks led to the adoption of CNNs (Convolutional Neural Networks) and RNNs (Recurrent Neural Networks) for both OCR and HTR [7], and more recently, Transformers have also proven effective for this task .

Recognizing text present in images is currently useful in a very broad range of activities that go beyond the scope of helping to overcome visual disabilities. From identifying vehicles by reading their license plates, to translating in real time texts just by pointing to a paper with the phone camera, or to transcribing texts in historical

documents to electronic files to help History researches to access their contents without risking the integrity of paper documents several centuries old.

This Master thesis examines alternatives to perform Handwritten Text Recognition on a set of almost 400 images that contains historical documents written in Spanish by the end of 18th century. The documents, usually comprised of two or more pages (thus, images) are notary documents corresponding to claims of bills of exchange. The next summary has been extracted from [8].

In simple terms, these documents were claims regarding unmet payment promises involving multiple parties. At their most basic, they involved a debtor and a creditor —each located in a different city— who conducted business with one another and used bills of exchange to avoid the constant transportation of money. However, as their use became widespread and they evolved into a trusted financial instrument, it became common for additional parties to emerge. These new actors either commissioned the original parties to make payments on their behalf or accepted bills of exchange as a monetary equivalent, leading to endorsements.

Endorsements allowed the creditor to transfer their right of collection to a third party (thus reinforcing the bill of exchange's role as a currency equivalent title). This third party could, in turn, endorse the bill again, creating a chain of payment and collection promises. If any link in the chain failed to honor its obligation, a default occurred. If the default remained unresolved after a certain period, the creditor could fill a formal claim before the city notary, town council, or marketplace authorities, including any associated costs. The Spanish term for this claim is *Protesto de letra de cambio*.

The historical and economic value of claims of bills of exchange lies in that those recorded the complete content —not only of original bill itself, but also of all subsequent endorsements— including dates, amounts and parties involved. They also documented the outcome of the protest and, where applicable, the reasons for which the debtor refused payment. All of this was transcribed to Spanish (in this case), since bills of exchange circulated across cities in different countries.

For all these reasons, protested bills of exchange are a highly significant source of historical and economic information. The information, however, is unstructured, as the notarial deed is not presented in a tabulated or summarized format. Rather, like all notary documents, it consists of a series of paragraphs recorded by the scribe, with the relevant information dispersed throughout.

Goals

This Master thesis aims to provide solutions to the needs of a History Research Effort on Financial Crisis through history. The specific task consists of extracting some data from bills of exchanges dated between 1790 and 1800, all of them written by hand by scribes. To accomplish this, the full text should first be transcripted to plain text and then passed to another tool that extract the relevant data to the Research Effort. This Master thesis covers the first part, the handwritten text recognition and transcription to plain text.

Planning

To achieve the goal of providing alternatives for transcribing handwritten documents in the form of images to text, the following plan has been proposed:

- Analyze State of the Art in Handwritten Text Recognition (HTR).
- Evaluate existing tools for the above task and how well they perform with the specific document set to process.
- Develop a specific set of Python scripts to batch processing the images, comprising from normalizing and sanitizing images to the actual training of a neural network to recognize handwritten text.
- Compare the performance of the alternatives through the CER and WER metrics (as definedin Chapter 2) and make a suggestion about which solution might perform better for the specific historical document image set.

Chapter - Conclusions and future work

Conclusions

A section of the work plan involved evaluating the performance of some popular open-source applications in the field of OCR and HTR. In this regard, the limitations of some of these tools for processing ancient handwritten texts have been confirmed, and it has been determined that eScriptorium is one of the most effective, despite the current difficulties in installing and running it without advanced knowledge of container management.

The evaluation of input data has helped identify the challenges posed by the acquired images for processing—from elements in the full image that extend beyond the specific content of the page to be captured (margins, opposite pages) to stains and deteriorated sections of the page, as well as the unique text layout, which almost always requires reordering the automatically transcribed text. Various mechanisms for attempting bulk and unattended image corrections have also been presented, explaining which ones yield good results and which do not. A Python script was implemented to accept an input directory of images and process them automatically.

It has been concluded that another important factor for performing HTR on images representing full pages is segmenting them into smaller images, each containing a single handwritten line. Algorithmic solutions have been confirmed to be inadequate for this task, with the best results coming from neural network architectures trained specifically for it—among which the models provided by Kraken deliver relatively efficient results. However, it should be noted that Transformer architectures eliminate the need for segmentation. Unfortunately, there do not yet seem to be sufficient labeled datasets available to train such networks effectively.

Various tests summarized in Table 5 show that models trained on popular existing datasets were not useful for obtaining automatic transcriptions of the input data as-is. Although one of the datasets used is in Spanish, its archaic Castilian language and vastly different handwriting style from that found in bills of exchange

made it unsuitable for generating even minimally valid automatic transcriptions. However, the eScriptorium application and the McCATMuS model, after fine-tuning with the first subset of 40 transcribed and reviewed pages, achieved an estimated CER of 5%. This suggests that improvement lies in fine-tuning the largest possible pre-trained models and likely leveraging advances in transcription review to gradually enhance the model's knowledge.

That said, perhaps the key conclusion so far is that the datasets used to evaluate advances in neural network architectures for HTR can be considered "good Samaritans" when subjected to testing—at least compared to the real-world input data used in this work. They allow for reasonable, even surprisingly good efficiency metrics, especially for someone seeing a system read handwritten text for the first time. But when attempting to extrapolate inference to data completely unrelated to the training dataset, the resulting error rates are drastically higher. While it is true that the input images have many more features (in both diversity and quantity) compared to those in the datasets, these are generally not due to deficiencies in image capture but rather the inherent nature of historical documents. It is worth remembering that we are dealing with documents that are not yet 250 years old—and they are likely not the worst-preserved ones to be encountered in historical research. Although the meeting minutes from the READ 2016 dataset share some similarities with the documents used here, they are still in much better condition and feature a far more homogeneous page layout.

For this reason, it would be advisable to compile and publish more properly labeled historical datasets to allow for realistic expectations when working with documents that exhibit deterioration and lack of clarity.

Future work

There is room for improvement to enhance the results documented in this Ms Thesis:

- Automated page framing: Since the general layout of the images is known, a specialized edge-detection algorithm could be implemented to automatically crop the pages.
- Hyperparameter tuning: Additional training runs could be conducted with model architectures while adjusting hyperparameters.
- Fine-tuning mechanisms: Once good results are achieved with popular datasets, fine-tuning methods should be implemented. Even if semi-automatic transcription in eScriptorium is required for a significant portion of the provided images, the result will be a high-quality dataset suitable for fine-tuning.
- Batch inference script: While no batch-processing script for inference on a verified high-performance model has been provided yet, this would be an obvious next step to speed up transcription once such a performing model is obtained.
- Post-processing with spell-checking: Implementing a post-processing step for transcription results using orthographic dictionaries could correct many errors the neural network fails to resolve. This resembles a CTC decoding strategy but is proposed as an external step rather than part of training. However, a challenge arises because most dictionaries are based on modern Spanish, while some words in the transcriptions are archaic and may not be included.

BIBLIOGRAFÍA

[1] S. Johnson, *How We Got to Now: Six Innovations That Made the Modern World*. Penguin Random House, 2018.

[2] P. A. Loubere, *A History of Communication Technology*. Routledge, Taylor & Francis Group, 2021.

[3] E. E. Fournier d'Albe, «On a type-reading optophone», *Proc. R. Soc. Lond.*, vol. 90, n.º 619, pp. 373-375, jul. 1914.

[4] H. F. Schantz, *History of OCR: optical character recognition*. Recognition Technologies Users Association, 1982.

[5] C. Y. Suen, C. Nadal, R. Legault, T. A. Mai, y L. Lam, «Computer recognition of unconstrained handwritten numerals», *Proc. IEEE*, vol. 80, pp. 1162-1180, jul. 1992.

[6] Y. Lecun *et al.*, «Handwritten digit recognition: Applications of neural net chips and automatic learning», *IEEE Trans. Commun.*, vol. 37, n.º 41-46, p. November 1989.

[7] Y. Lecun, L. Bottou, Y. Bengio, y P. Haffner, «Gradient-based learning applied to document recognition», *Proc. IEEE*, vol. 86, n.º 11, pp. 2278-2324, 1998, doi: 10.1109/5.726791.

[8] F. Cebreiro Ares, E. Jorge-Sotelo, L. Caruana De Las Cagigas, y C. Larrinaga-Rodríguez, *De comerciantes, pagos y crisis: elementos para una unidad didáctica en torno a la letra de cambio*, vol. Superando el Covid-19 en las aulas de historia económica. Universidad de Grandada, 2022.

[9] C. Garrido-Munoz, A. Rios-Vila, y J. Calvo-Zaragoza, «Handwritten Text Recognition: A Survey», 12 de febrero de 2025, *arXiv*: arXiv:2502.08417. doi: 10.48550/arXiv.2502.08417.

[10] A. CHUDÁREK, «Analyzing a person's handwriting for recognizing his/her emotional state», BRNO UNIVERSITY OF TECHNOLOGY, BRNO, 2023.

[11] G. Kim y V. Govindaraju, «A lexicon driven approach to handwritten word recognition for real-time applications», *IEEE Trans. Patterns Anal. Mach. Intell.*, vol. 19, n.º 4, pp. 366-379, abr. 1997, doi: 10.1109/34.588017.

[12] J. Hu, M. K. Brown, y W. Turin, «HMM based online handwriting recognition», *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 18, n.º 10, pp. 1039-1045, oct. 1996, doi: 10.1109/34.541414.

[13] H. Yasuda, K. Takahashi, y T. Matsumoto, «A discrete hmm for online handwriting recognition», *Int. J. Pattern Recognit. Artif. Intell.*, vol. 14, n.º 05, pp. 675-688, ago. 2000, doi: 10.1142/S021800140000043X.

[14] U.-V. Marti y H. Bunke, «Using a statistical language model to improve the performance of an hmm-based cursive handwriting recognition system», *Int. J. Pattern Recognit. Artif. Intell.*, vol. 15, n.º 01, pp. 65-90, feb. 2001, doi: 10.1142/S0218001401000848.

[15] L. R. Rabiner, «A tutorial on hidden Markov models and selected applications in speech recognition», *Proc. IEEE*, vol. 77, n.º 2, pp. 257-286, feb. 1989, doi: 10.1109/5.18626.

[16] D. E. Rumelhart, Ed., «Parallel distributed processing. 1: Foundations / David E. Rumelhart», 12. print., vol. I, Cambridge, Mass: MIT Pr, 1999.

[17] A. Graves, S. Fernandez, F. Gomez, y J. Schmidhuber, «Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks».

[18] J. Puigcerver, «Are Multidimensional Recurrent Layers Really Necessary for Handwritten Text Recognition?», en *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, Kyoto: IEEE, nov. 2017, pp. 67-72. doi: 10.1109/ICDAR.2017.20.

[19] K. He, X. Zhang, S. Ren, y J. Sun, «Deep Residual Learning for Image Recognition», 10 de diciembre de 2015, *arXiv*: arXiv:1512.03385. doi: 10.48550/arXiv.1512.03385.

[20] L. Kang, P. Riba, M. Rusiñol, A. Fornés, y M. Villegas, «Pay Attention to What You Read: Non-recurrent Handwritten Text-Line Recognition», 26 de mayo de 2020, *arXiv*: arXiv:2005.13044. doi: 10.48550/arXiv.2005.13044.

[21] T. Bluche, J. Louradour, y R. Messina, «Scan, Attend and Read: End-to-End Handwritten Paragraph Recognition with MDLSTM Attention», 23 de agosto de 2016, *arXiv*: arXiv:1604.03286. doi: 10.48550/arXiv.1604.03286.

[22] D. Coquenet, C. Chatelain, y T. Paquet, «End-to-end Handwritten Paragraph Text Recognition Using a Vertical Attention Network», *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, n.º 1, pp. 508-524, ene. 2023, doi: 10.1109/TPAMI.2022.3144899.

[23] D. Coquenet, «Meta-DAN: towards an efficient prediction strategy for page-level handwritten text recognition», 4 de abril de 2025, *arXiv*: arXiv:2504.03349. doi: 10.48550/arXiv.2504.03349.

[24] A. E. Harraj y N. Raissouni, «OCR accuracy improvement on document images through a novel pre-processing approach», *Signal Image Process. Int. J.*, vol. 6, n.º 4, pp. 01-18, ago. 2015, doi: 10.5121/sipij.2015.6401.

[25] Q. A. Bui, D. Mollard, y S. Tabbone, «Selecting Automatically Pre-Processing Methods to Improve OCR Performances», en *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, Kyoto: IEEE, nov. 2017, pp. 169-174. doi: 10.1109/ICDAR.2017.36.

[26] L. R. Mursari y A. Wibowo, «The Effectiveness of Image Preprocessing on Digital Handwritten Scripts Recognition with The Implementation of OCR Tesseract», *Comput. Eng. Appl. J.*, vol. 10, n.º 3, pp. 177-186, oct. 2021, doi: 10.18495/comengapp.v10i3.386.

[27] C. Sabbagh, «Enhanced HTR Accuracy for Tibetan Historical Texts - Optimising Image Pre-processing for Improved Transcription Quality», *Rev. D'Etudes Tibétaines*, n.º 74, pp. 81-127, feb. 2025.

[28] M. R. Gupta, N. P. Jacobson, y E. K. Garcia, «OCR binarization and image pre-processing for searching historical documents», *Pattern Recognit.*, vol. 40, n.º 2, pp. 389-397, feb. 2007, doi: 10.1016/j.patcog.2006.04.043.

[29] G. Mühlberger, S. Colutto, y P. Kahle, «Handwritten Text Recognition (HTR) of Historical Documents as a Shared Task for Archivists, Computer Scientists and Humanities Scholars. The Model of a Transcription & Recognition Platform (TRP)», oct. 2014.

[30] H. Scheidl, «Handwritten text recognition in historical documents», Thesis, Technische Universität Wien, 2018. doi: 10.34726/hss.2018.43931.

[31] M. Li *et al.*, «TrOCR: Transformer-based Optical Character Recognition with Pre-trained Models», arXiv.org. Accedido: 7 de junio de 2025. [En línea]. Disponible en: https://arxiv.org/abs/2109.10282v5

[32] P. Stokes y B. Kiessling, «Sharing Data for Handwritten Text Recognition (HTR)», *Digit. Humanit. Pract.*, n.º hal-04444641, feb. 2024, [En línea]. Disponible en: https://hal.science/hal-04444641v1

[33] A. Pinche y P. Stokes, «Historical Documents and Automatic Text Recognition: Introduction», *J. Data Min. Digit. Humanit.*, vol. Historical Documents and..., p. 13247, mar. 2024, doi: 10.46298/jdmdh.13247.

[34] D. Castro, B. L. D. Bezerra, y C. Zanchettin, «An End-to-End Approach for Handwriting Recognition: From Handwritten Text Lines to Complete Pages», en *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Seattle, WA, USA: IEEE, jun. 2024, pp. 264-273. doi: 10.1109/cvprw63382.2024.00031.

[35] *tesseract-ocr/tesseract*. (2 de junio de 2025). C++. tesseract-ocr. Accedido: 3 de junio de 2025. [En línea]. Disponible en: https://github.com/tesseract-ocr/tesseract

[36] C. Reul *et al.*, «OCR4all -- An Open-Source Tool Providing a (Semi-)Automatic OCR Workflow for Historical Printings», *Appl. Sci.*, vol. 9, n.º 22, p. 4853, nov. 2019, doi: 10.3390/app9224853.

[37] «Scripta / escriptorium · GitLab», GitLab. Accedido: 3 de junio de 2025. [En línea]. Disponible en: https://gitlab.com/scripta/escriptorium

[38] U.-V. Marti y H. Bunke, «The IAM-database: an English sentence database for offline handwriting recognition», *Int. J. Doc. Anal. Recognit.*, vol. 5, n.º 1, pp. 39-46, nov. 2002, doi: 10.1007/s100320200071.

[39] E. Grosicki, M. Carré, E. Geoffrois, E. Augustin, F. Preteux, y R. Messina, «RIMES, complete». Mitek Systems, Inc, 13 de marzo de 2024. Accedido: 7 de junio de 2025. [En línea]. Disponible en: https://zenodo.org/records/10812725

[40] J. A. Sánchez, V. Romero, A. H. Toselli, y E. Vidal, «READ dataset Bozen». Zenodo, 22 de diciembre de 2016. Accedido: 10 de junio de 2025. [En línea]. Disponible en: https://zenodo.org/records/218236

[41] E. Granell y C.-D. Martínez-Hinarejos, «The Rodrigo corpus». Zenodo, 16 de noviembre de 2018. Accedido: 7 de junio de 2025. [En línea]. Disponible en: https://zenodo.org/records/1490009

[42] D. Pérez, L. Tarazón, N. Serrano, F. Castro, O. R. Terrades, y A. Juan, «The GERMANA Database», en *2009 10th International Conference on Document Analysis and Recognition*, Barcelona, Spain: IEEE, 2009, pp. 301-305. doi: 10.1109/ICDAR.2009.10.

[43] J. Canny, «A Computational Approach to Edge Detection», *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-8, n.° 6, pp. 679-698, nov. 1986, doi: 10.1109/TPAMI.1986.4767851.

[44] R. Liuberskis, «Handwriting words recognition with TensorFlow», Python Lessons. Accedido: 13 de junio de 2025. [En línea]. Disponible en: https://pylessons.com/handwriting-recognition

[45] G. Retsinas, G. Sfikas, B. Gatos, y C. Nikou, «Best Practices for a Handwritten Text Recognition System», 17 de abril de 2024, *arXiv*: arXiv:2404.11339. doi: 10.48550/arXiv.2404.11339.

APÉNDICES

Apéndice A - Especificaciones de sistema

Para la realización de este trabajo se ha utilizado un equipo local, con la intención de tener un presupuesto cerrado del coste operativo. La computación en la nube puede ser más efectiva, especialmente en tiempo, cuando hay una buena planificación del tiempo de computación necesario, o bien si los recursos necesarios para llevar a cabo el entrenamiento en instalaciones locales son prohibitivos. En este trabajo, que evalúa y compara opciones, resultaba más flexible utilizar un hardware local.

La configuración hardware ha sido la siguiente:

- CPU: AMD Ryzen 7900X.
- RAM: 64 GB DDR5 4800 Mhz.
- Almacenamiento: HDD Seagate 2 TB + SSD Kingston 1 TB.
- GPU: nVidia RTX 4060 Ti 16 GB.
- Fuente de alimentación: Corsair 750 W.

La configuración software ha sido la siguiente:

- Sistema operativo: Linux Mint 22.0.
- Gestor de contenedores: Docker 28.2.2.
- Python 3.12.3.
- Tensorflow 2.19.0.
- Keras 3.9.2.

Otras bibliotecas Python utilizadas están recogidas en los ficheros requirements.txt en el repositorio.

El consumo eléctrico del sistema utilizando la CPU con esa configuración ronda los 100 W. Durante el entrenamiento de las redes neuronales, la GPU reportó consumos en torno a los 100 W que han de sumarse al del resto del sistema.

Apéndice B - Entrenamiento con READ 2016

El entrenamiento de la arquitectura que utiliza la biblioteca MLTU con el dataset READ 2016 dio este resultado:

```
 1/30 ━━━━━━━━━━━━━━━━━━━━ 12:36 26s/step - CER: 21.8879 - WER: 1.1380 - loss: 2
 2/30 ━━━━━━━━━━━━━━━━━━━━ 16s 602ms/step - CER: 16.7480 - WER: 1.1035 - loss: 1
 3/30 ━━━━━━━━━━━━━━━━━━━━ 16s 601ms/step - CER: 13.8518 - WER: 1.0843 - loss: 1
 4/30 ━━━━━━━━━━━━━━━━━━━━ 15s 602ms/step - CER: 11.9582 - WER: 1.0719 - loss: 1
 5/30 ━━━━━━━━━━━━━━━━━━━━ 15s 601ms/step - CER: 10.7105 - WER: 1.0630 - loss: 1
 6/30 ━━━━━━━━━━━━━━━━━━━━ 14s 601ms/step - CER: 10.0726 - WER: 1.0564 - loss: 1
 7/30 ━━━━━━━━━━━━━━━━━━━━ 13s 601ms/step - CER: 9.7355 - WER: 1.0511 - loss: 16
 8/30 ━━━━━━━━━━━━━━━━━━━━ 13s 601ms/step - CER: 9.4303 - WER: 1.0469 - loss: 16
 9/30 ━━━━━━━━━━━━━━━━━━━━ 12s 601ms/step - CER: 9.1161 - WER: 1.0434 - loss: 16
10/30 ━━━━━━━━━━━━━━━━━━━━ 12s 601ms/step - CER: 8.8082 - WER: 1.0404 - loss: 15
11/30 ━━━━━━━━━━━━━━━━━━━━ 11s 601ms/step - CER: 8.5144 - WER: 1.0379 - loss: 15
12/30 ━━━━━━━━━━━━━━━━━━━━ 10s 601ms/step - CER: 8.2376 - WER: 1.0357 - loss: 15
13/30 ━━━━━━━━━━━━━━━━━━━━ 10s 601ms/step - CER: inf - WER: 1.0338 - loss: 1485.
14/30 ━━━━━━━━━━━━━━━━━━━━ 9s 601ms/step - CER: inf - WER: 1.0321 - loss: 1461.6
15/30 ━━━━━━━━━━━━━━━━━━━━ 9s 601ms/step - CER: inf - WER: 1.0305 - loss: 1438.6
16/30 ━━━━━━━━━━━━━━━━━━━━ 8s 601ms/step - CER: inf - WER: 1.0292 - loss: 1416.6
17/30 ━━━━━━━━━━━━━━━━━━━━ 7s 601ms/step - CER: inf - WER: 1.0279 - loss: 1395.4
18/30 ━━━━━━━━━━━━━━━━━━━━ 7s 601ms/step - CER: inf - WER: 1.0268 - loss: 1375.1
19/30 ━━━━━━━━━━━━━━━━━━━━ 6s 601ms/step - CER: inf - WER: 1.0258 - loss: 1355.5
20/30 ━━━━━━━━━━━━━━━━━━━━ 6s 601ms/step - CER: inf - WER: 1.0248 - loss: 1336.5
21/30 ━━━━━━━━━━━━━━━━━━━━ 5s 601ms/step - CER: inf - WER: 1.0240 - loss: 1318.5
22/30 ━━━━━━━━━━━━━━━━━━━━ 4s 601ms/step - CER: inf - WER: 1.0232 - loss: 1301.0
23/30 ━━━━━━━━━━━━━━━━━━━━ 4s 601ms/step - CER: inf - WER: 1.0224 - loss: 1284.2
24/30 ━━━━━━━━━━━━━━━━━━━━ 4s 832ms/step - CER: inf - WER: 1.0217 - loss: 1268.6
25/30 ━━━━━━━━━━━━━━━━━━━━ 4s 823ms/step - CER: inf - WER: 1.0211 - loss: 1253.5
26/30 ━━━━━━━━━━━━━━━━━━━━ 3s 814ms/step - CER: inf - WER: 1.0205 - loss: 1238.9
27/30 ━━━━━━━━━━━━━━━━━━━━ 2s 806ms/step - CER: inf - WER: 1.0199 - loss: 1224.8
28/30 ━━━━━━━━━━━━━━━━━━━━ 1s 798ms/step - CER: inf - WER: 1.0194 - loss: 1211.3
29/30 ━━━━━━━━━━━━━━━━━━━━ 0s 791ms/step - CER: inf - WER: 1.0189 - loss: 1198.1
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 784ms/step - CER: inf - WER: 1.0184 - loss: 1185.4
```

Se ha encontrado, ya a punto de entregar la memoria, un *issue*[15] que documenta un problema similar y al que se le ofrece como consejo asegurarse de que:

1. Todas las etiquetas tienen contenido (no están vacías).
2. La longitud de la secuencia predicha es mayor que 2 * longitud(etiqueta) + 1.

El último punto no se ha visto documentado en los artículos científicos ni en los tutoriales encontrados sobre CRNN.

[15] https://github.com/meijieru/crnn.pytorch/issues/235

Apéndice C - Métricas de algunos modelos entrenados

NOTA: ESTE APÉNDICE SE HA AÑADIDO TRAS LA EVALUACIÓN POR EL TRIBUNAL DE ESTE TRABAJO, SIGUIENDO SUS RECOMENDACIONES.

A continuación se reproducen algunas gráficas con las métricas de CER y WER de diferentes datasets usando el script CRNN descrito en el apartado 5.3 que utiliza la biblioteca MLTU de PyLessons.

Subconjunto imágenes 01-40 Protestos de letras de cambio (1)

Estos resultados corresponden a un entrenamiento realizado justo antes de realizar la entrega de este trabajo para su evaluación. Puede observarse que el CER está en torno al 151,3 % y el WER en torno al 100,6 %. Aunque notablemente más bajas que las de los primeros intentos (un 273 %), las cifras siguen indicando que el entrenamiento no progresa correctamente. Cabe recordar en este punto que un CER superior al 10 % puede considerarse una tasa de fallos bastante alta.

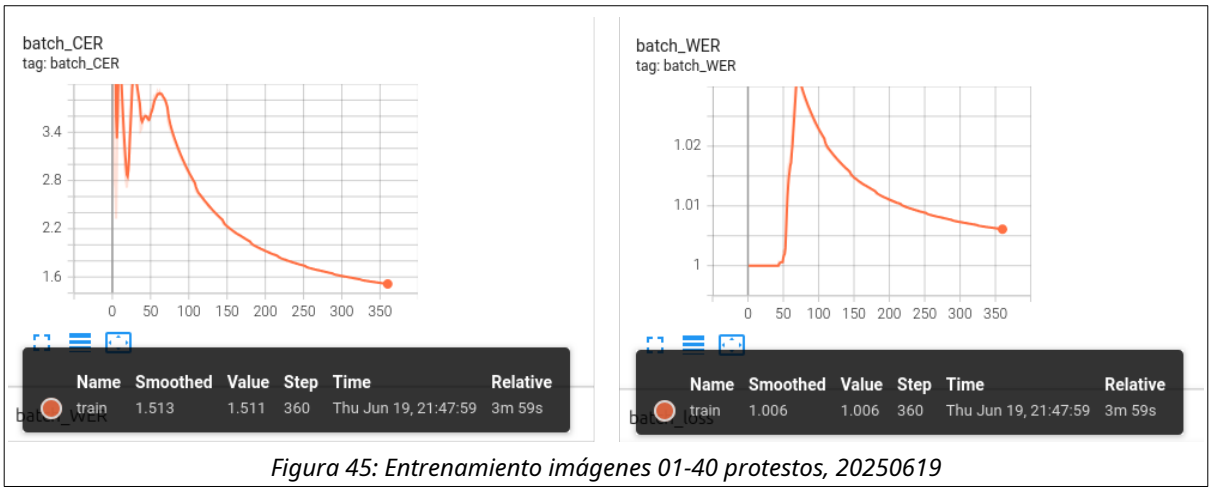

Figura 45: Entrenamiento imágenes 01-40 protestos, 20250619

Subconjunto imágenes 01-40 Protestos de letras de cambio (2)

Estos resultados corresponden a un entrenamiento realizado con posterioridad a la entrega de este trabajo y las conclusiones reflejadas en él. Puede observarse que el CER está en torno al 13,8 % mientras que el WER se fija en el 28,1 %, unas cifras peores que las obtenidas en numerosos artículos científicos, pero que mejoran en más de un orden de magnitud las obtenidas inicialmente. No se hicieron cambios en la arquitectura de la red neuronal utilizada entre los distintos entrenamientos, por lo que la causa de la mejora está por determinar. Una hipótesis consistiría en que, por ser un dataset de un tamaño excesivamente pequeño, la mezcla que se realiza en cada intento para aelatorizar el proceso puede tener cierto impacto en el resultado.

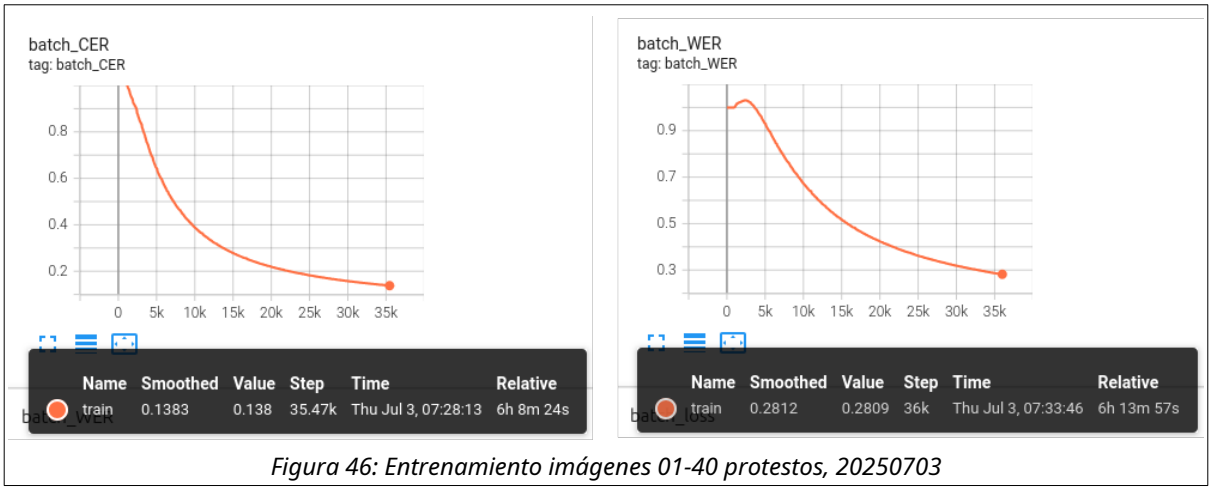

Figura 46: Entrenamiento imágenes 01-40 protestos, 20250703

Dataset READ 2016

Estos resultados son los descritos en el apartado 5.3 de este trabajo. El estilo de las gráficas cambia ligeramente para poder mostrar que el valor de CER es infinito. El valor de WER supera el 100 %, por lo que tampoco resulta útil.

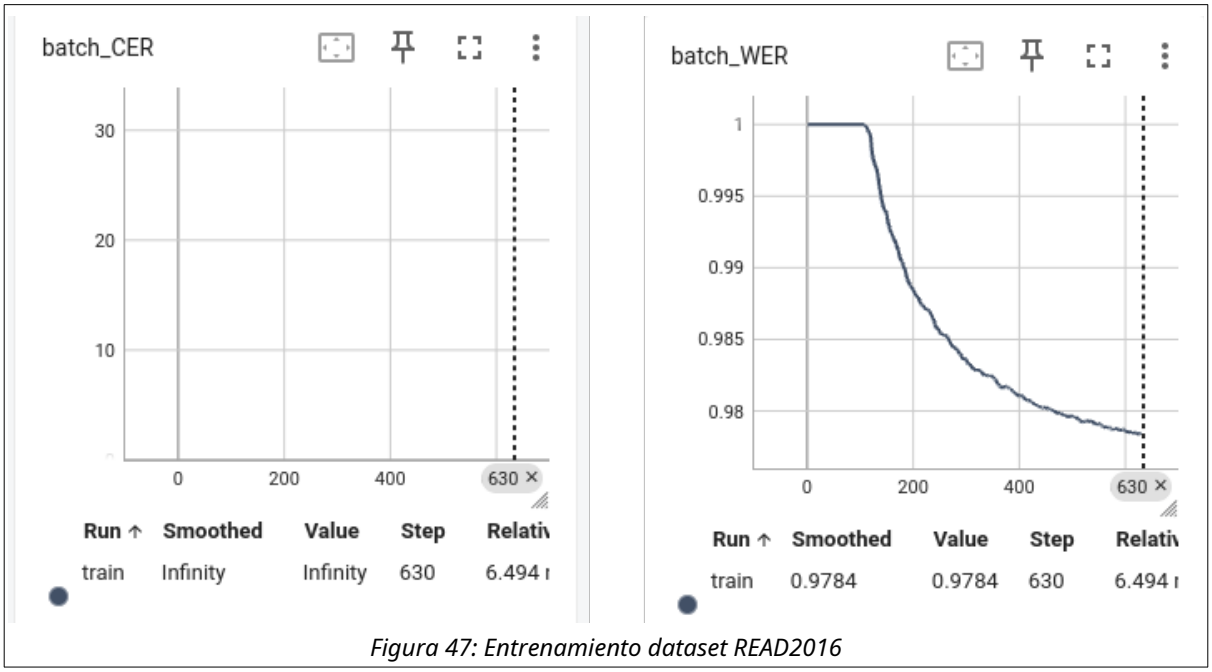

Figura 47: Entrenamiento dataset READ2016

Dataset Rodrigo

Estos son los resultados del entrenamiento con el dataset Rodrigo. Las cifras de CER y WER (35,3 % y 74,9 %, respectivamente) son más altas que las descritas en el apartado 5.3 porque las de estas gráficas corresponden al entrenamiento, en lugar de al test de inferencia.

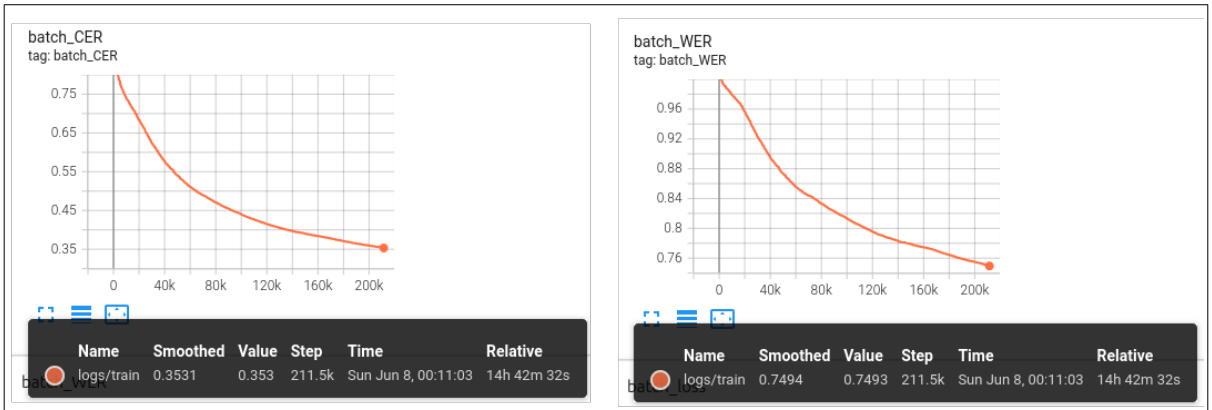

Figura 48: Entrenamiento dataset Rodrigo

Dataset IAM

Por último, estos son los resultados con el dataset IAM, el utilizado por el autor de la biblioteca MLTU para ilustrar el funcionamiento de su arquitectura CRNN. Como puede verse, las cifras son muy competitivas respecto de las mostradas en la mayoría de los artículos científicos citados.

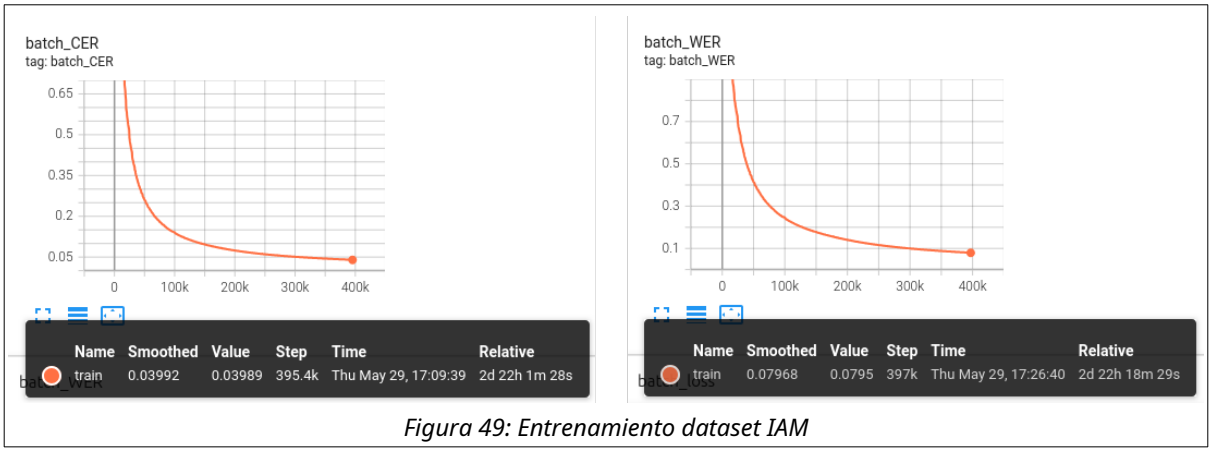

Figura 49: Entrenamiento dataset IAM

Apéndice D - Glosario de siglas

Nota: este apéndice se ha añadido tras la evaluación por el tribunal de este trabajo, siguiendo sus recomendaciones.

- AI: Artificial Intelligence.
- BLSTM: Bidirectional Long Short Term Memory.
- CER: Character Error Rate.
- CLAHE: Contrast Limited Adaptative Histogram Equalization.
- CNN: Convolutional Neural Network.
- CRNN: Convolutional Recurrent Neural Network.
- CTC: Connectionist Temporal Classifcation.
- CV: Computer Vision.
- HMM: Hidden Markov Model.
- HSL: Hue, Saturation, Luminance.
- HSV/B: Hue, Saturation, Value/Brightness.
- HTR: Handwritten Text Recognition.
- LSTM: Long Short Tem Memory.
- MDLSTM: Multidimensional Long Short Term Memory.
- OCR: Optical Character Recognition.
- RGB: Red, Green, Blue.
- RNN: Recurrent Neural Network.
- RO: Reading Order.
- WER: Word Error Character.