Instituto de Computación, Facultad de Ingeniería,
Universidad de la República

Proyecto de Grado

Modelos *Seq2Seq* para la transcripción de documentos del Archivo Berrutti

Autor
Felipe Chavat Pérez
felipe.chavat@fing.edu.uy

Tutores
Diego Garat
Guillermo Moncecchi
{dgarat, gmonce}@fing.edu.uy

Uruguay
2022

Resumen

Archivo Berrutti es el nombre que recibe un conjunto de documentos generados por las Fuerzas Armadas entre los años 1968 y 1985, años marcados por sucesos de terrorismo de Estado y la reciente dictadura Uruguaya (1973-1985). El Proyecto *CRUZAR* busca la sistematización de la versión digital del archivo, de forma de facilitar el estudio de la estructura y forma de accionar de los organismos represivos. Es en este marco que el proyecto *LUISA* (*Leyendo Unidos para Interpretar loS Archivos*) desarrolla herramientas para semi automatizar esta tarea, como la transcripción automática de las imágenes digitales a un formato procesable por sistemas de extracción de información.

El presente trabajo enfoca su estudio en la evaluación de métodos de aprendizaje automático profundo orientado a la transcripción automática de imágenes del *Archivo Berrutti*, a partir de las transcripciones manuales obtenidas a través de la plataforma *LUISA*. En particular, se considera la arquitectura *Seq2Seq*, la cual fue desarrollada en principio para la traducción automática, pero que luego se aplicó a otros problemas, como el *OCR*, mostrando resultados que compiten con el estado del arte en el tema.

El modelo implementado alcanza una tasa de error a nivel de caracteres —*CER*— del 28.10% frente al 23.74% obtenido por la última herramienta *OCR* usada por el equipo *LUISA* y el 52% obtenido por una herramienta anterior. Los resultados son promisorios y alientan a seguir avanzando en la implementación de nuevas características que mejoren el modelo.

Palabras clave: aprendizaje profundo, arquitectura Seq2Seq, Archivo Berrutti, LUISA, OCR

Agradecimientos

Tuve la suerte de tener de tutores a quienes dictaron el curso que marcó mi interés por el Aprendizaje Automático. Gracias Diego y Guillermo, no he dejado de aprender de ustedes.

Gracias al equipo de LUISA: por la disposición que tuvieron en todo momento y por llevar adelante lo que creen justo.

Mi madre, mi padre, mis hermanos y mis amigos: pilares fundamentales, sostén y medio para la tranquilidad. Gracias. A ellas/os les dedico el resultado del tiempo destinado.

Índice general

1 Introducción **6**

1.1 Cronograma 8

1.2 Estructura del Informe 9

2 Marco Teórico **10**

2.1 Reconocimiento Óptico de Caracteres 10

2.2 Modelado de Secuencias 11

2.3 Redes Neuronales *feedforward* 13

2.4 Redes Neuronales Recurrentes 14

2.5 Redes Neuronales Convolucionales 15

2.6 Modelado de secuencias en Aprendizaje Automático 19

2.6.1 Distintos enfoques anteriores y sus limitantes 19

2.6.2 Enfoques basados en redes neuronales recurrentes 21

2.7 Trabajo relacionado 24

3 Datos de entrenamiento **28**

3.1 Construcción de conjunto de bloques 28

3.1.1 Conjunto de bloques con transcripción seleccionada 32

3.1.2 Conjunto de bloques con todas sus transcripciones válidas 34

3.2 Construcción de conjunto de líneas 35

3.2.1 Composición de la línea 36

3.2.2 Filtrado de bloques con transcripción no deseada 38

3.2.3 Bloques que contienen sub-palabras 39

3.2.4 Ensamblado final y generación del conjunto de datos de entrenamiento 43

3.2.5 Descomposición de líneas a bloques originales . . . 44
3.3 Conjunto de líneas curado . . . 44

4 Seq2Seq para la transcripción de documentos del Archivo Berrutti 48
4.1 Extracción de características de la imagen de entrada . . . 48
4.2 Codificador . . . 50
4.3 Decodificador . . . 50
4.4 Mecanismo de atención . . . 52

5 Análisis experimental 57
5.1 Métricas . . . 57
5.2 Función de costo . . . 58
5.3 Especificaciones del entrenamiento . . . 59
5.4 Escenarios de prueba . . . 62
5.4.1 Escenario 1: entrenamiento sobre conjunto de bloques . . . 62
5.4.2 Escenario 2: uso de las múltiples transcripciones por bloque . . . 63
5.4.3 Escenario 3: entrenamiento con conjunto de líneas construido a partir del ensamblado de bloques de *LUISA* . . . 64
5.4.4 Escenario 4: entrenamiento con conjunto de líneas provisto por el equipo *LUISA* y comparación con Calamari-OCR . . . 65
5.5 Implementación y ejecución . . . 66
5.6 Resultados . . . 67
5.6.1 A nivel de bloques . . . 67
5.6.2 A nivel de líneas . . . 68

6 Conclusiones 71
6.1 Trabajo a futuro . . . 72

Referencias 74

7 Anexo 79
7.1 Evaluación de instancias de entrenamiento . . . 79
7.1.1 Entrenamiento sobre bloques . . . 79
7.1.2 Uso de múltiples instancias por bloque . . . 81
7.1.3 Entrenamiento sobre conjunto de líneas ensambladas . . . 82

Capítulo 1

Introducción

La Facultad de Ingeniería y la Facultad de Información y Comunicación tienen a su disposición un conjunto de imágenes correspondientes a documentos de inteligencia de las Fuerzas Armadas durante la última dictadura Uruguaya (1968-1985), período en el que se sufrió de graves sucesos de Terrorismo de Estado y violaciones de los Derechos Humanos. Este conjunto de documentos es llamado *Archivo Berrutti*, y corresponde a un archivo encontrado en el Ministerio de Defensa en el año 2006-2007. El archivo contiene una cantidad significativamente grande de páginas escaneadas, superior a las dos millones, y podría arrojar luz sobre hechos que hasta el día de hoy no han logrado ser clarificados debido a la falta de información sobre lo sucedido. Lo que está disponible del *Archivo Berrutti* es un conjunto de imágenes digitales en formato *TIFF* (*Tagged Image File Format*), formato para almacenamiento de imágenes en forma de mapa de bits. Este formato, al ser de imagen, no contiene el texto correspondiente y por lo tanto no permite la extracción de información. Además, las imágenes se encuentran *binarizadas*, es decir que cada píxel de la imagen es de color negro o blanco[1].

Es en este marco que el proyecto *CRUZAR*[2] tiene por objetivo el desarrollo de software que *«permita el cruzamiento de la información contenida en esos archivos. El cruzamiento facilita la investigación y el análisis de los temas referidos al terrorismo de Estado, como un aporte más en la búsqueda de la verdad»*. Transcribir manualmente cada imagen es posible, pero el tiempo necesario para transcribir millones de páginas es alto. Es a raíz de esto que se crea un subproyecto llamado *LUISA*[3]: una plataforma colaborativa que tiene por objetivo generar un conjunto de datos que contenga transcripciones asociadas a segmentos de las imágenes, obtenidas a través de la colaboración de voluntarios y voluntarias. La segmentación se realiza en unidades llamadas *bloques* (Figura 1.1) que, en general, incluyen una sola palabra de la hoja. Estos bloques son presentados

[1] A estas imágenes puede llamárseles también del tipo 1bpp (*one bit per pixel*)
[2] "CRUZAR - Archivos del pasado reciente", https://cruzar.edu.uy
[3] "LUISA - Leyendo Unidos para Interpretar loS Archivos", https://mh.udelar.edu.uy/luisa/

en la plataforma junto con una imagen de contexto, que corresponde a un segmento más grande que incluye a los bloques mostrados (Figura 1.2). El voluntario o la voluntaria introduce para cada bloque el texto que considera correcto según lo que observa y luego se almacena esta información en una base de datos. Previo a la segmentación se ejecuta sobre las imágenes de páginas un algoritmo de alineación —proceso llamado *deskewing*— que se encarga de enderezar las imágenes de forma tal que las líneas de texto queden completamente horizontales. Se aplica también un algoritmo de eliminación de ruido —*denoising*— que tiene por objetivo mejorar la calidad visual de las imágenes y, con ello, la calidad de transcripción.

Gracias a la participación de personas voluntarias en *LUISA* se ha podido obtener un conjunto de transcripciones lo suficientemente grande, mayor a dos millones y medio y aún en aumento, como para entrenar modelos de transcripción automática (modelos de *Optical Character Recognition*). A partir de aquí resulta de interés estudiar distintas alternativas para la construcción de modelos que resuelvan la tarea.

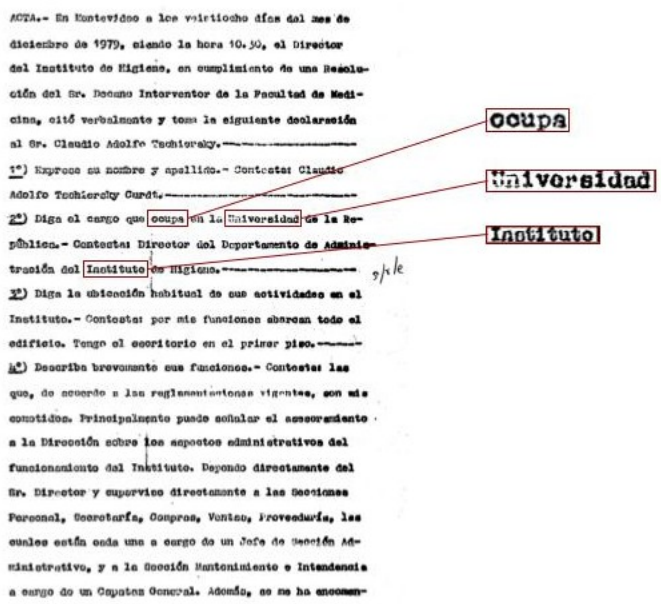


Figura 1.1: Del lado izquierdo la imagen resultante del escaneo de una página. A la derecha, tres ejemplos de bloques válidos que son mostrados en la plataforma *LUISA*.

Debido al avance del Aprendizaje Automático Profundo en los últimos años se han presentado arquitecturas de modelos de automatización de tareas en múltiples áreas que mejoran el estado del arte. El área de *OCR* ha sido una de ellas. En particular, existe un tipo de arquitectura denominada *Seq2Seq*, que ha sido usada para varios tipos de problemas de secuencias tales como la traducción automática, la transcripción automática de texto contenido en imágenes, la identificación del texto

Figura 1.2: Ejemplo de caso de uso de la plataforma *LUISA* donde al usuario se le presentan un conjunto de bloques que necesitan ser transcritos junto con un recuadro de contexto correspondiente a un área más grande que contiene a los bloques. Mediante la observación de los bloques y ayudándose de la imagen de contexto, el usuario o la usuaria rellena los *inputs* de texto con el texto que considera el adecuado.

en discursos de audio, entre otros.

Es entonces de motivación para este proyecto el estudio de la arquitectura *Seq2Seq* y la realización de pruebas con distintos modelos entrenados sobre los datos obtenidos en la plataforma *LUISA*, con la intención de mejorar la calidad de la información extraída por el sistema *CRUZAR*.

Más en específico, se construyen múltiples conjuntos de datos a partir de los datos generados en la plataforma *LUISA*, y luego se entrenan distintos modelos de Aprendizaje Automático Profundo que hacen uso de la arquitectura *Seq2Seq* para la transcripción automática de texto.

Como resultado del proyecto, no solo se obtienen múltiples modelos entrenados con la arquitectura *Seq2Seq*, sino también un grupo de conjuntos de datos construidos a partir de las transcripciones realizadas por voluntarias y voluntarios, donde se aplican algunas heurísticas analizadas con la intención de mejorar la calidad de los datos utilizados para el entrenamiento.

1.1 Cronograma

En la Tabla 1.1 se muestran las distintas etapas que se recorrieron en el proyecto junto con el tiempo estimado y el que llevó realizarlas, y una descripción de cada una. El Proyecto logró recorrer cada una de las etapas que se plantearon en principio, con una diferencia en el tiempo real en algunas ellas. En el caso de la etapa de implementación debido a la generación de nuevas pruebas en función de las realizadas al principio, y en el informe final debido a tiempos que fueron necesarios para generar un buen documento.

Fecha estimada	Fecha real	Descripción de etapa
Septiembre 2020 Octubre 2020	Septiembre 2020 Octubre 2020	Aproximación al problema
Octubre 2020 Diciembre 2020	Octubre 2020 Marzo 2020	Estado del arte
Diciembre 2020 Enero 2021	Diciembre 2020 Enero 2021	Diseño y puesta en marcha del entorno de trabajo
Enero 2021 Marzo 2021	Febrero 2021 Mayo 2021	Diseño de la solución de aprendizaje
Marzo 2021 Mayo 2021	Marzo 2021 Octubre 2021	Implementación: incluyó construcción de los conjuntos de datos, implementación de modelo, entrenamiento de distintos modelos
Mayo 2021 Julio 2021	Octubre 2021 Abril 2022	Informe final

Cuadro 1.1: Cronograma del Proyecto de Grado. Las distintas etapas previstas junto con el tiempo estimado, el tiempo real de realización y la descripción de cada una de ellas.

1.2 Estructura del Informe

El informe está estructurado de forma tal que el Capítulo 2 se centra en el Marco Teórico, haciendo un pasaje por la síntesis de la literatura vinculada a los métodos de aprendizaje automático de secuencias. En particular, aquellos aplicados al *OCR*.

El Capítulo 3 hace un recorrido por el proceso mediante el cual se obtuvieron los distintos conjuntos de datos utilizados en la experimentación.

En el Capítulo 4 se especifica la arquitectura del modelo propuesto.

El Capítulo 5 está relacionado con el Análisis Experimental, conteniendo la especificación, las métricas utilizadas para evaluar los modelos, los escenarios de prueba ejecutados, otras especificaciones de la etapa de experimentación y los resultados obtenidos.

Finalmente, el Capítulo 6 está destinado a las conclusiones finales del Proyecto junto con líneas de posibles trabajos a futuro.

Capítulo 2

Marco Teórico

Este capítulo aborda los diferentes conceptos teóricos utilizados a lo largo del proyecto. Primero se presentan los problemas cuya naturaleza involucra secuencias de elementos. Luego se hace un recorrido sobre los modelos conocidos para la resolución de problemas de modelado de secuencias dentro del Aprendizaje Automático, pasando por los conceptos de redes neuronales estándar, recurrentes y convolucionales. Por último se estudia la arquitectura *Seq2Seq*, observando los distintos usos, finalizando en el análisis de esta arquitectura sobre el problema específico de Reconocimiento Óptico de Caracteres (*Optical Character Recognition, OCR*).

2.1 Reconocimiento Óptico de Caracteres

El Reconocimiento Óptico de Caracteres engloba el conjunto de técnicas que tienen por objetivo reconocer y transcribir texto que se encuentra contenido en formato digital (imagen), es decir, la transformación de imágenes que contienen texto no manipulable por una computadora a un formato que sí lo sea (Islam et al., 2017).

Las técnicas de OCR se han vuelto herramientas muy útiles en la vida moderna, tanto en procesos de digitalización de documentos físicos existentes (Philips et al., 2020), como en tareas de control, entre las que se encuentra, por ejemplo, el reconocimiento de matrículas de vehículos (Qin et al., 2021). Para la transcripción de los documentos es necesario primero obtener la imagen digital por medio de su escaneo. Luego es posible aplicar las técnicas *OCR* para obtener el texto a partir de ellas. También es posible que se tengan disponibles solo las imágenes digitales de los documentos si en un período previo se decidió hacer un respaldo virtual y luego se perdieron los originales. En este tipo de casos existen complejidades relacionadas con la calidad de las digitalizaciones de los documentos y la imposibilidad de lograr nuevas versiones digitales.

2.2 Modelado de Secuencias

Existen problemas que requieren el procesamiento o generación de datos cuya estructura está dada en forma de secuencias. Por ejemplo: si se quisiera traducir un conjunto de oraciones de un lenguaje a otro, sería necesario analizar, en cada una, no solo la correspondencia de cada palabra al lenguaje destino, sino también el orden de cada una de las palabras y cómo se relacionan. Por lo tanto, para modelar este tipo de problemas, es necesario modelar también la naturaleza secuencial de los datos. A este tipo se les llama problemas de modelado de secuencias, y analizándolos se puede observar que existen distintos tipos, que describimos a continuación.

El primer caso es aquel donde, en función de un único elemento de entrada, se genera una secuencia de elementos de salida asociada (Figura 2.1). Un ejemplo de este tipo de problema es el de generar la descripción de una imagen (*Image Captioning*, Bernardi et al., 2016).

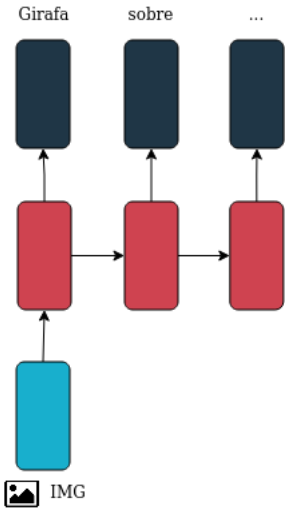


Figura 2.1: Problema de modelado de secuencias *one to many*. La entrada está formada por un solo elemento, la salida es una secuencia asociada al elemento de entrada. A modo de ejemplo puede considerarse un modelo que recibe como entrada una imagen y da como salida una secuencia de palabras que la describe.

Otro tipo de problema de modelado de secuencias es aquel donde se desea clasificar una secuencia de elementos (que puede ser de largo variable). Es decir, se quiere asignar una clase a una secuencia de elementos de entrada, como se observa en la Figura 2.2. Un ejemplo es aquel donde se quiere conocer el sentimiento asociado a un comentario compuesto por una secuencia de palabras (*Sentiment analysis*, B. Liu, 2020), o el de determinar si una secuencia de palabras corresponde a un chiste (*Humor analysis*, Yang et al., 2015).

Tercero, aquellos problemas de modelado de secuencias donde, en función de una secuencia de

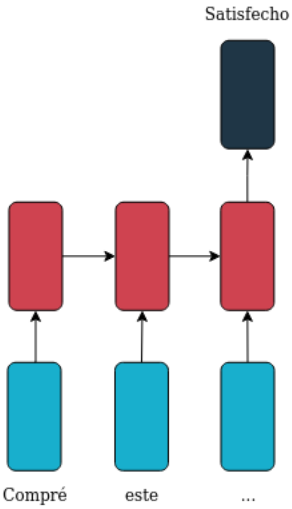


Figura 2.2: Modelado de secuencias *many to one*. Es el caso en el que se clasifica una secuencia de entrada. La entrada está conformada por una secuencia de elementos y la salida un único elemento que le asigna una clase. Un ejemplo es la clasificación de comentarios, donde dada una secuencia de palabras que compone un comentario se busca saber si la persona que lo escribió expresa satisfacción o no.

elementos, se genera otra secuencia que tenga correspondencia con la primera (según cuál sea el objetivo del problema), como se observa en la Figura 2.3. Algunos ejemplos: generar la secuencia de palabras correspondiente a un discurso hablado (*Speech Recognition*, Jelinek, 1998) o la traducción automática, donde dada una secuencia de palabras en un idioma se genera otra secuencia de palabras en el lenguaje objetivo que tenga correspondencia en su significado (*Machine Translation*, Guerra et al., 2000), la generación del texto correspondiente al texto contenido en una secuencia de imágenes o la secuencia de columnas de una sola imagen de entrada, como será en este caso (*Optical Character Recognition*, Rice et al., 2012). Estos problemas son conocidos como problemas de secuencia a secuencia (*Sequence to Sequence*).

Por último, a veces se requiere predecir, en función de una secuencia de entrada, cuál es el siguiente elemento más probable (Figura 2.4). Por ejemplo, obtener, a partir de una secuencia de palabras, cuál es la siguiente más probable (*Language Models*, Koul et al., 2019). Otro ejemplo de este tipo de problema es aquel donde, en función de una secuencia de valores obtenidos en el tiempo y con una misma distancia temporal se busca predecir cuál será el siguiente valor (*Time-series Forecasting*, Brockwell et al., 2006). Este tipo de problemas es otra variación de problema *many-to-many*. El proyecto pondrá foco en el tercer tipo comentado, los problemas *sequence to sequence*.

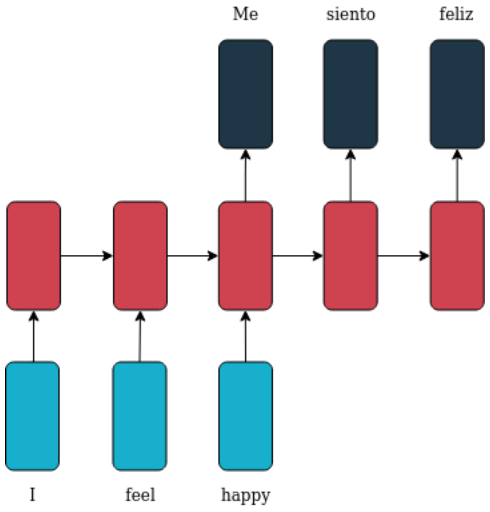


Figura 2.3: Modelado de secuencias *many to many*. Problema de secuencias donde dada una secuencia de entrada se busca generar una secuencia de salida asociada a la secuencia de entrada. La traducción automática es el ejemplo en este diagrama.

2.3 Redes Neuronales *feedforward*

Según LeCun et al. (2015), las redes neuronales *feedforward* (*Feedforward Neural Networks*) son aquellas que aprenden a mapear una entrada de largo fijo a una salida de largo fijo, procesando la entrada a través de las capas que conforman la red.

Cada una de las capas está compuesta por unidades (*neural units*). Cada una de estas unidades recibe un conjunto de valores de la capa anterior (salvo por la capa de entrada, que recibe los valores de los datos de entrada), lo procesa haciendo la suma ponderada de los valores junto con los pesos y evaluando este resultado con una función no lineal, llamada función de activación.

Las redes neuronales *feedforward* están compuestas por varias capas, en donde cada una es un conjunto de unidades que tienen por objetivo procesar la entrada recibida. En principio se clasifican las capas en tres tipos: la capa de entrada, la o las capas ocultas y la capa de salida. La capa de entrada es la que recibe los valores correspondientes a la entrada de la red neuronal. Las capas ocultas son las encargadas de ir transformando la información que será recibida por la capa final, llamada capa de salida, que aplicará la transformación necesaria en función del problema que se busca solucionar. Por ejemplo, en el caso de una clasificación binaria como puede ser reconocer si una imagen contiene un dígito en particular, podría usarse una capa de salida con un solo nodo que aplica la función de activación de forma tal que si el valor resultante está por debajo de un umbral se entiende que se clasificó como negativo para ese dígito, en caso contrario se clasifica positivo.

El algoritmo más conocido para aprender los pesos de estas redes es *backpropagation* (retropro-

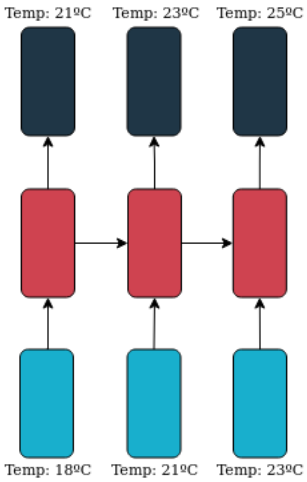


Figura 2.4: Variación de modelado de secuencias *many to many*. Tipo de problema de secuencias donde se recibe una secuencia de entrada y se busca predecir, en cada salto de tiempo, el siguiente elemento. A modo de ejemplo puede considerarse un modelo que busca predecir en la línea de tiempo la evolución de la temperatura de un proceso. Entonces, dada una temperatura de entrada se predice cuál va a ser la temperatura en el siguiente intervalo de tiempo. Esta salida será utilizada como entrada en el siguiente salto de tiempo para modelar la evolución.

pagación), que consiste en calcular el gradiente del error cometido por la red y propagarlo desde la última capa hacia el comienzo, actualizando en cada paso los pesos de las unidades de cada capa. Es necesario para esto una función que retorne el error cometido en la clasificación de la red. A esta se le llama *loss function* o función de pérdida.

2.4 Redes Neuronales Recurrentes

Las Redes Neuronales Recurrentes (*Recurrent Neural Networks, RNN*) son modelos neuronales, al igual que las redes neuronales *feedforward*. Sin embargo, como puede verse en la Figura 2.5, tienen la capacidad de retener información contextual de etapas anteriores a medida que procesan cada elemento de una secuencia, dado que las unidades que la conforman poseen conexiones especiales para retroalimentarse. Esto les da la posibilidad de procesar secuencias de elementos y además de aprender las relaciones temporales existentes, haciéndolas idóneas para el modelado de problemas *sequence to sequence*.

Las primeras redes recurrentes fueron propuestas por Jordan (1997) y Elman (1990). Estas redes, además de tener las ya conocidas unidades neuronales, poseen otras unidades especiales encargadas de recibir información procesada anteriormente para volver a pasársela a las unidades

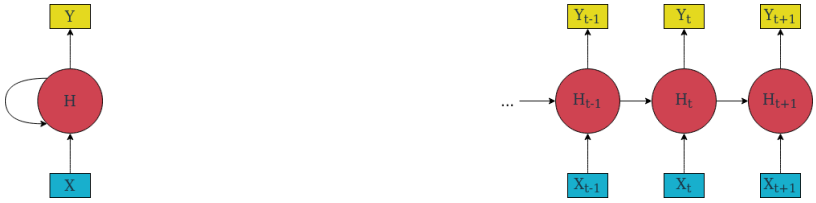


Figura 2.5: Unidad estándar de Red Neuronal Recurrente. En este caso, X corresponde a la entrada, H a la unidad de procesamiento e Y a la salida. Puede verse que existe una conexión consigo misma, que le permitirá incluir información contextual para el siguiente salto de tiempo. En la subfigura b, la unidad desplegada a través del tiempo. Aquí queda más claro como la conexión que existe consigo misma «transporta» información de contexto hacia el siguiente salto de tiempo.

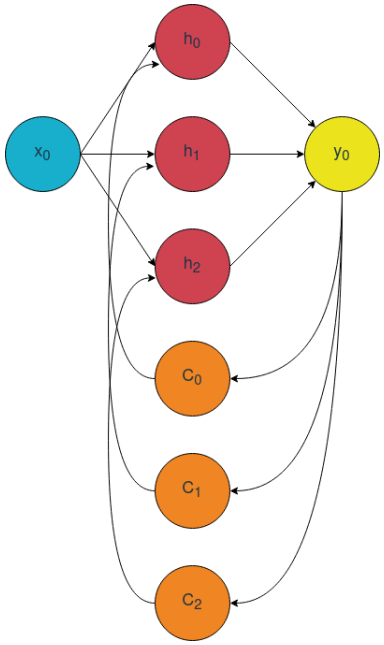
neuronales (Figura 2.6). De esta forma se mantiene contexto de saltos de tiempo anteriores.

Las *RNN* sufren de lo que se llama problema de desvanecimiento de gradiente y problema de gradiente explosivo (*vanishing gradient problem* y *exploding gradient problem* respectivamente, Bengio et al., 1994; Hochreiter et al., 2001). El problema de desvanecimiento de gradiente se da cuando un mismo error se propaga hacia atrás durante demasiados saltos de tiempo, generando así que el gradiente tienda a cero, estancando el entrenamiento de la red. O en caso contrario (si el error es suficientemente grande), que el gradiente «explote» aumentando en demasía y eliminando también la posibilidad de que el modelo aprenda.

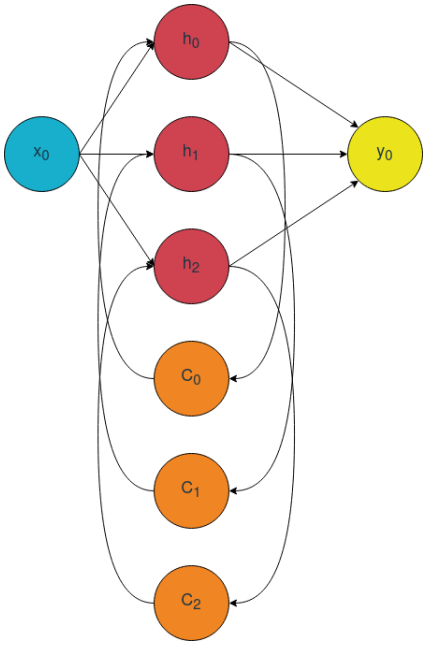
Hochreiter et al. (1997) propone lo que hoy es una de las unidades recurrentes más usadas, que soluciona estos problemas. La unidad, llamada célula de memoria, reemplaza las unidades estándar usadas hasta el momento en estas redes y tiene la función de no solo retener información mediante una célula de retroalimentación interna, sino también de controlar el flujo de información que entra y sale mediante tres compuertas distintas, evitando de esta forma los problemas de gradiente y permitiendo a la red aprender dependencias temporales más distanciadas. Esta arquitectura recibe el nombre de *Long Short-Term Memory (LSTM)* en inglés y es ampliamente usada para modelado de problemas secuenciales en la actualidad. Algunos ejemplos son Sutskever et al. (2014) y Wu et al. (2016) para la traducción automática, Graves, Fernández, Liwicki et al. (2007) en reconocimiento de texto escrito a mano.

2.5 Redes Neuronales Convolucionales

Las redes neuronales convolucionales (*CNN*) han demostrado ser buenas herramientas para problemas de clasificación de imágenes, dado que son capaces de aprender a identificar características relevantes para lograr generar la clasificación correcta. Uno de los primeros trabajos de clasificación



(a) Red Neuronal Recurrente descrita por Jordan



(b) Red Neuronal Recurrente descrita por Elman

Figura 2.6: Primeras redes neuronales recurrentes presentadas

con redes *CNN* es el de Bottou et al. (2005), que las usa para la clasificación de dígitos y caracteres escritos a mano.

En una *CNN* existen dos tipos de capas. Las capas convolucionales son las encargadas de extraer características aplicando una serie de transformaciones al elemento de entrada a través de lo que se llaman *kernels* (núcleos) y obteniendo como salida lo que recibe el nombre de *feature map* (mapa de características). Como puede observarse en la Figura 2.7, el *kernel* es aplicado sucesivamente sobre la imagen (de izquierda a derecha, desde el extremo superior hacia el inferior), moviéndose cada vez una cantidad de elementos equivalente a lo que se llama *stride* (paso). En este ejemplo se asume que la matriz de valores correspondiente a la imagen de entrada tiene un solo canal, lo que no es siempre así, y podría existir por ejemplo una imagen codificada según los valores de color rojo, verde y azul (*RGB*), por lo que existiría una matriz correspondiente a los valores de rojo en cada pixel, otra para el color verde y otra para el color azul, en total 3 canales. Las capas convolucionales pueden también aumentar la cantidad de canales resultantes al aumentar la cantidad de *kernels* usados, donde cada uno generará una transformación distinta sobre la imagen de entrada y dará por resultado un *feature map* correspondiente. Por ejemplo, de usar 5 *kernels* sobre una imagen de un solo canal, se obtendrán 5 *feature maps* en total, que se lee como 5 canales resultantes. La Figura 2.8 intenta expresar este ejemplo. Dado que un *kernel* funciona como un identificador de características de los elementos que se procesan (Y. Liu, 2018), se deduce que usar varios puede mejorar en la capacidad de extracción de características de la red.

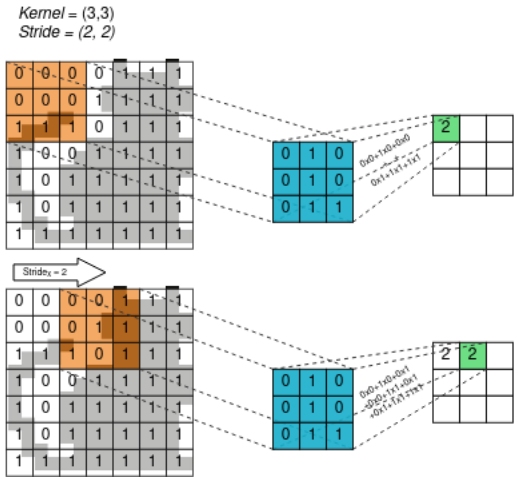


Figura 2.7: Aplicación de un *kernel* en una red convolucional a modo de ejemplo. En esta, el *kernel* de dimensión 3x3 es aplicado sucesivamente sobre la imagen de entrada y dando por resultado una matriz, o canal correspondiente al *kernel*, de dimensiones menores. En este ejemplo, el *stride* es de 2x2, que significa que el *kernel* se moverá dos elementos hacia la derecha antes de volver a calcular una nueva transformación y análogamente dos elementos hacia abajo cuando llegue al final de la fila y retorne a la primera columna.

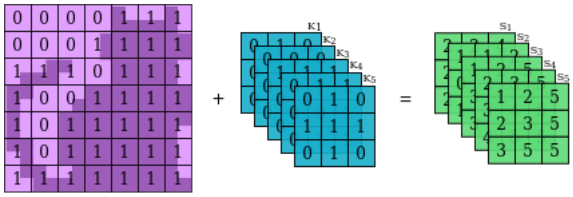


Figura 2.8: Aplicación de varios *kernels* sobre un elemento de entrada para la obtención de múltiples canales de salida. Cada canal aportará información relevante sobre las características de la imagen de entrada.

Las capas de agrupación (*pooling*) son las que tienen el objetivo de reducir la dimensión de la entrada y funcionan de forma similar a las convolucionales, aunque no aplican la suma del producto de los valores de un *kernel*, sino que tienen el objetivo de agrupar de alguna forma los valores del área a la que aplican la transformación. Los tipos de capas de *pooling* más usadas son las que promedian los valores del área afectada (*average pooling*) o las que toman el valor máximo del área afectada (*max pooling*). La Figura 2.9 refleja la transformación generada por una capa de *max-pooling*. Estas capas son generalmente usadas entre conjuntos de capas convolucionales.

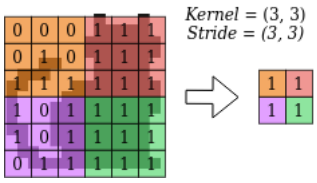


Figura 2.9: Aplicación de capa de *max pooling* sobre un elemento en una red convolucional. En este caso la transformación se hace usando un *kernel* de dimensión 3x3 y un *stride* de 3x3 también, que genera que cada elemento de salida no comparta información de origen con ninguno de los demás elementos.

Se le llama *receptive field* a aquella área a través de la que se llegó a uno de los valores resultantes mediante las transformaciones realizadas. En la Figura 2.10 podemos ver enmarcado con línea punteada el *receptive field* del primer elemento de salida de aplicar un *kernel* sobre una imagen. En la Figura 2.11 puede observarse el *receptive field* en una *CNN* con varias capas convolucionales. El concepto de *receptive field* no describe un conjunto de valores que será luego utilizado, sino que sirve para entender a qué área de la imagen de entrada corresponde cada uno de los valores obtenidos al final de la red. Dicho de otra manera, cada uno de los valores de salida contiene información de características de la imagen de un área específica (su *receptive field*).

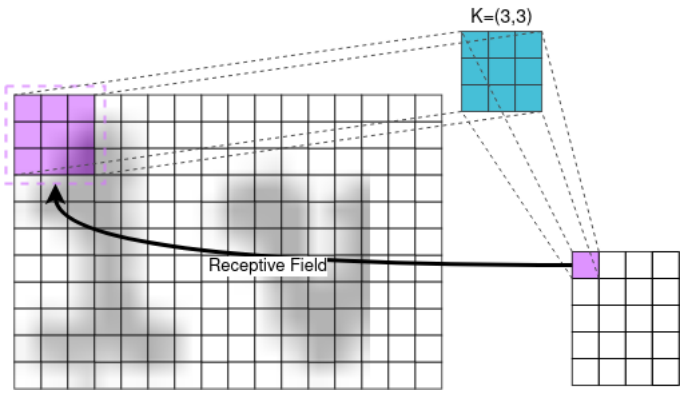


Figura 2.10: *Receptive field* del elemento marcado. La línea punteada corresponde al *receptive field* del elemento final (de color violeta).

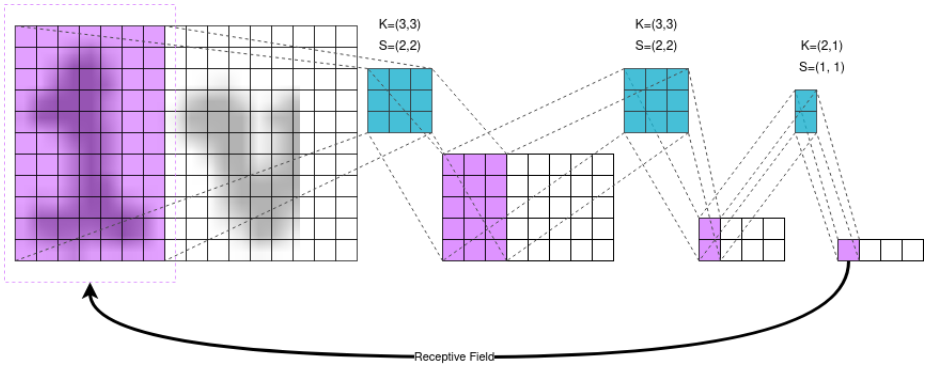


Figura 2.11: *Receptive field* (marcado con líneapunteada) de un elemento luego de haberse aplicado múltiples transformaciones.

2.6 Modelado de secuencias en Aprendizaje Automático

Distintos métodos han sido presentados a lo largo de la extensa línea de soluciones de aprendizaje automático para construir modelos que resuelvan problemas de secuencia a secuencia. En esta sección se hace un pasaje somero sobre distintos enfoques.

2.6.1 Distintos enfoques anteriores y sus limitantes

Distintos enfoques, previos al estudiado, han sido usados para tareas de OCR. Entre ellos se encuentran los modelos *Markovianos*, en especial los modelos ocultos de *Markov* descritos en 1950 y estudiados desde 1960 (Stratonovich, 1960), modelos que generan distribuciones probabilísticas sobre la ocurrencia conjunta de la secuencia de entrada y la secuencia de salida. Más en específico,

definen dos distribuciones probabilísticas: aquella que describe cómo se relacionan los elementos adyacentes de la secuencia de salida, llamada "distribución de transición" dado que las posibles clases que la componen se modelan como nodos y la posibilidad de adyacencia en la secuencia objetivo como una transición cuya probabilidad es la descrita por la distribución. La segunda es llamada distribución de observación y es la que describe la probabilidad de observar la secuencia de entrada condicionada a las posibles secuencias de salida. Mediante estas dos distribuciones es posible predecir la secuencia objetivo.

Si bien los modelos son capaces de representar las dependencias temporales o de orden inherentes en las secuencias de datos, esta capacidad es realmente limitada, teniendo que aumentar exponencialmente el espacio de nodos del modelo si es que se desea aumentar la potencia de representar relaciones de mayor distancia, convirtiéndose en una restricción. Otros métodos similares usados han sido *Conditional Random Fields* (Lafferty et al., 2001) y *Graph Transformer Networks* (Bottou et al., 2005).

A medida que emergen conjuntos de datos que ponen a disposición grandes cantidades de ejemplos etiquetados junto con el avance en los recursos de hardware necesarios para generar modelos que los procesen en un tiempo razonable, los métodos de aprendizaje profundo, sobre todo las redes neuronales profundas (*Deep Neural Networks*) y las redes neuronales convolucionales (*Convolutive Neural Networks*) comienzan a encontrar su lugar dentro de los métodos que lideran los puntajes de rendimiento en distintas áreas como el procesamiento de imágenes, procesamiento de lenguaje natural, procesamiento de audio, entre otras. Ejemplo de esto son Krizhevsky et al. (2012) y Schmidhuber et al. (2012) en el área de clasificación de imágenes. No solo logran los modelos alcanzar los puntajes del estado del arte, sino que logran puntajes que compiten o superan el rendimiento humano.

Aún así, las redes neuronales estándar tienen limitaciones. La primera es que se basan en la suposición de independencia entre los elementos de entrada. Es decir, luego de que un elemento de entrada es procesado a través de cada una de las capas y se obtiene un elemento de salida (el resultado de procesarlo), toda la información inherente al estado de la red generado durante el procesamiento se pierde y no vuelve a ser usada en los futuros elementos procesados, perdiendo información que podría usarse como contexto. Esto no presenta limitantes si los ejemplos que componen el conjunto de datos son independientes. Sin embargo, en aquellos problemas donde existe correlación temporal o de orden entre los elementos de la secuencia a procesar, como lo pueden ser imágenes de un video o palabras en una oración, esta suposición falla.

La segunda limitante es que en la mayoría de los casos pierden la capacidad de generalizar si la entrada es de largo variable, debido a la estructura de la red. Es por esto que las redes neuronales estándar, por sí solas, se ven restringidas para resolver problemas de modelado de secuencias.

2.6.2 Enfoques basados en redes neuronales recurrentes

Las redes neuronales recurrentes presentan gran capacidad para identificar dependencias temporales, hecho que las convierte en una herramienta útil para atacar problemas de modelado de secuencias. Sin embargo, por sí solas requieren para el entrenamiento que la entrada sea una secuencia de elementos alineada con la secuencia de salida esperada. Es decir, sería necesario tener previamente segmentada la entrada con su respectiva secuencia *ground truth* también segmentada y alineada. Un ejemplo de esto puede ser un audio para el que se quiere reconocer el texto correspondiente: sería necesario tener dicho audio segmentado fonema a fonema con su texto correspondiente, para luego entrenar una red neuronal recurrente que reciba el audio como una secuencia de fonemas y prediga, para cada uno, el texto que le corresponde. Es poco común conseguir conjuntos de datos cuyos elementos se encuentren segmentados previamente y alineados.

Graves et al. (2006) abordan este problema introduciendo en 2006 el método *Connectionist Temporal Classification*, que consiste en interpretar la salida de la red neuronal como una distribución de probabilidad sobre todas las secuencias de salida posibles, condicionadas a la secuencia de entrada. A partir de esto se derivará una función objetivo para maximizar la probabilidad de la secuencia correcta de entre todas las secuencias posibles. Se muestra luego en este mismo artículo que un modelo de ejemplo que hace uso de este método, una red neuronal recurrente con función objetivo *CTC*, supera al estado del arte en la tarea de reconocimiento de fonemas en audios. El proceso de *CTC* sería entonces: procesar la secuencia de entrada, que no está segmentada, post procesar la salida para generar el conjunto de posibles secuencias resultantes, y luego calcular aquella secuencia que maximiza la probabilidad de ser la correcta.

Existen limitantes en el método *CTC*: el largo de la secuencia de salida queda limitada por el largo de la secuencia de entrada, ya que este método asume que el alineamiento entre ellas es del tipo muchos-a-uno (*many-to-one*), por lo que a cada elemento de entrada le corresponderá a lo sumo un elemento de salida. Además, el método asume también que existe alineamiento monótono entre las dos secuencias, existiendo el mismo orden entre los elementos de una y otra, convirtiéndolo en un método no viable para problemas como el de traducción automática, donde el orden de las palabras correspondientes a la traducción puede cambiar en función de la gramática de los idiomas (Sutskever et al., 2014).

Si bien este método fue ampliamente usado sobre todo para problemas de reconocimiento óptico de caracteres (OCR), razón por la que se le da mayor énfasis en este capítulo, este tipo de modelos no entraría dentro del conjunto a los que la literatura hace referencia como arquitectura *Seq2Seq*. Como puede verse en Sueiras et al. (2018), las arquitecturas *Seq2Seq* están diseñadas para resolver problemas que necesitan transformar la secuencia de entrada a otra secuencia de largo posiblemente distinto, una limitante del método *CTC*. Sin embargo, este método es relativamente actual, sobre todo dentro de los problemas de *OCR*.

Los primeros modelos *Seq2Seq* fueron introducidos por Cho et al. (2014) y Sutskever et al.

(2014) para problemas de traducción automática. Ambos formularon una arquitectura similar que consiste en una red neuronal recurrente que mapea o codifica la entrada de largo variable a un vector de valores de largo fijo, llamado vector de características. Luego, otra red neuronal recurrente oficia de decodificador, mapeando este vector de largo fijo a una secuencia de símbolos de salida y largo variable e independiente del largo de entrada. La Figura 2.12 muestra la arquitectura del primer modelo nombrado.

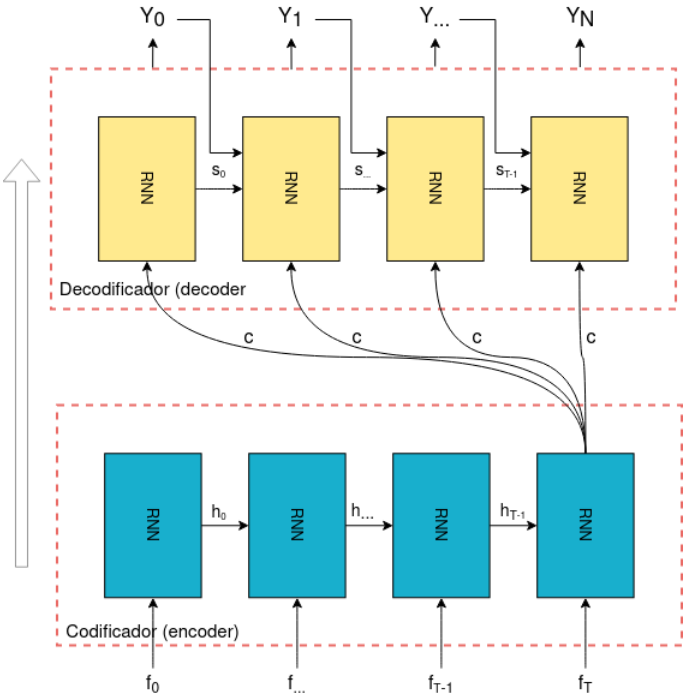


Figura 2.12: Arquitectura de modelo *Seq2Seq* planteada por (Cho et al., 2014). Un módulo llamado codificador procesa la secuencia de palabras, codificadas en *word embeddings* $f_1, ..., f_T$, y genera un vector de largo fijo c que será procesado por el módulo decodificador dando por resultado una secuencia de palabras de largo variable N, que es la traducción correspondiente.

En el caso de Sutskever et al. (2014) usan unidades tipo *Long Short Term Memory (LSTM)* para construir las redes neuronales recurrentes, y así beneficiarse de las cualidades antes nombradas. En el caso de Cho et al. (2014) se usa una unidad recurrente presentada por ellos, llamada *Gated Recurrent Unit* (unidad recurrente con compuertas), basada en la unidad *LSTM* pero simplificada. Ambas arquitecturas tienen la ventaja de poder ser entrenadas de comienzo a fin sin la necesidad de hacerlo componente a componente por separado, característica identificada como *end-to-end*[1]. De esta forma, a partir del costo asociado a la predicción hecha por el modelo es posible actualizar los

[1]Podría traducirse como entrenamiento punta a punta

pesos tanto de las redes encargadas de extraer las características de la imagen como del codificador y el decodificador. El entrenamiento *end-to-end* es característico dentro de los modelos de aprendizaje profundo, como los *Seq2Seq*.

Este tipo de arquitectura *Seq2Seq* es llamada *encoder-decoder* (codificador-decodificador) y consiste en un codificador que se encargará de leer la secuencia de entrada, transformándola en un vector de representación de largo fijo y un decodificador que dado un vector de representación de largo fijo devolverá la secuencia de símbolos de largo variable que maximiza la probabilidad condicionada a la entrada. El decodificador puede verse como un modelo de lenguaje basado en redes neuronales recurrentes (Mikolov et al., 2010), condicionado además a la entrada de largo fijo, es decir, al vector de representación de la entrada, ya que no solo predice el siguiente símbolo de la salida en función del símbolo anterior predicho, sino que también tiene en cuenta el vector de representación que consume de la salida del codificador.

Algunos enfoques sobre esta arquitectura, como Shi et al. (2017) y Bahdanau et al. (2015), modifican el codificador usando redes recurrentes bidireccionales, dándole al modelo la posibilidad de incluir contexto en las dos direcciones de la secuencia de elementos de entrada, mejorando el rendimiento en problemas donde naturalmente existe ventaja al tener en cuenta tanto las relaciones con los elementos anteriores como los posteriores.

Una de las limitantes de las arquitecturas *encoder-decoder* presentadas hasta aquí es que el vector de características de largo fijo obtenido de procesar la entrada a través del codificador debe sintetizar de alguna forma la información contenida en la secuencia, que puede ser de largo variable, lo que genera que por ejemplo, en casos donde la secuencia de entrada es más larga que el largo promedio en el conjunto de datos de entrada, se pierda información relevante para la decodificación (Cho et al., 2014b). Es decir, el hecho de que se construya un vector de largo fijo para contener las características de una secuencia de largo variable impone una limitante en la cantidad de información que puede contener ese vector.

Bahdanau et al. (2015) introduce una modificación al modelo en la que en vez de convertir la secuencia de entrada en un vector de representación de largo fijo, la transforma en un conjunto de vectores de representación, y usa para decodificar, en cada salto de tiempo (*timestep*), un vector resultante de sumar de forma ponderada este conjunto de vectores, de forma tal que tengan más peso aquellos elementos que aportan información de valor en la predicción actual. Dicho de otra manera, la entrada del decodificador ahora es un vector resultante de la suma ponderada de los elementos de salida del codificador, y pasa a tener el nombre de vector de contexto. Las ponderaciones utilizadas para generar el vector de contexto son llamados puntajes de atención y son computados a partir del estado oculto (*hidden state*) del codificador, el estado oculto actual del decodificador (antes de dar por salida el nuevo elemento) y la predicción anterior realizada por este. La Figura 2.13 intenta esquematizar el mecanismo de atención nombrado, llamado *content-based attention mechanism*, o mecanismo de atención basado en contenido.

Desde el momento en el que se introduce el mecanismo de atención a la arquitectura *Seq2Seq*

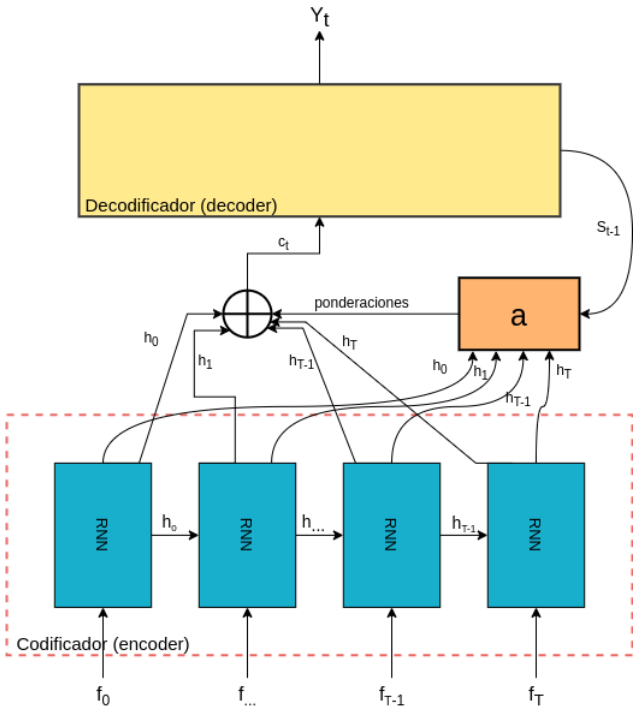


Figura 2.13: Mecanismo de atención basado en contenido. A partir del vector de estado correspondiente al salto de tiempo actual en el decodificador s_{t-1} junto con el estado de la red que forma el codificador $h_0, ..., h_T$ se obtiene un vector de ponderaciones que será utilizado para formar el vector de contexto c_t, que es la suma ponderada (con los pesos obtenidos) de los vectores de estado del codificador.

ha sido usado debido a la mejora significativa sobre el modelo estándar. Algunos trabajos que hacen uso de los mecanismos son Luong et al. (2015) y Wu et al. (2016) para la traducción automática de texto, Poulos et al. (2021), Kang et al. (2022) y Chowdhury et al. (2018) para el reconocimiento de texto escrito a mano, entre otros.

Existen distintos mecanismos de atención. Puede ser de interés la lectura del análisis hecho por Michael et al. (2019) sobre distintos mecanismos de atención sobre un modelo *encoder-decoder* para la tarea de reconocimiento de texto escrito a mano.

2.7 Trabajo relacionado

Distintas técnicas de aprendizaje automático para problemas de secuencias fueron observadas en la sección 2.6. En esta sección se hace énfasis en la arquitectura *Seq2Seq* destinada a OCR.

Para este problema la salida es una secuencia, ya que se busca la secuencia de caracteres correspondiente al texto contenido en la imagen. Sin embargo, puede resultar no tan directo si la entrada corresponde a una secuencia de elementos. Los métodos de OCR con la arquitectura experimentada en este proyecto descomponen la imagen en una secuencia de características. Según Sueiras et al. (2018) se utilizan comúnmente dos estrategias para la extracción de las características. La primera corresponde a la aplicación de técnicas de *Computer Vision* como pueden ser *Principal Component Analysis* sobre recuadros de la imagen de entrada, o como se observó antes en el estado del arte, hacer uso de redes convolucionales para la extracción de características. La segunda estrategia consta de convertir la entrada a una secuencia de elementos correspondientes a características identificadas. Puede entenderse el paso de extracción de características de la imagen como una etapa de codificación inicial, ya que se transforma la entrada, que es un archivo imagen, en una secuencia de elementos manipulables por las redes neuronales.

Chowdhury et al. (2018) proponen un modelo para la transcripción de líneas escritas a mano cuya arquitectura la describen como formada por dos módulos. El primero, llamado módulo de extracción de características, genera una secuencia de vectores de características a partir de la imagen de entrada mediante el uso de una Red Neuronal Convolucional (*CNN*). Luego, el segundo módulo, llamado módulo de aprendizaje de secuencias, es una componente que sigue la arquitectura *encoder-decoder* de los modelos *Seq2Seq*: el *encoder* está formado por una red *LSTM* bidireccional que se encarga de procesar la secuencia de vectores de características, un mecanismo de atención que procesa la salida del *encoder* para generar, en cada salto de tiempo, un vector de contexto con la información que usará el decodificador, que es una RNN, para predecir el siguiente símbolo de la secuencia objetivo. Supera en su momento al estado del arte (Puigcerver, 2017) que presentaba un modelo compuesto por una red convolucional para la extracción de características de la imagen y una *RNN* aplicando el método *Connectionist Temporal Classification* (*CTC*).

En el caso de Michael et al. (2019), también para reconocimiento de líneas de texto escrito a mano y actual estado del arte, se hace uso de una arquitectura *encoder-decoder* junto con un mecanismo de atención. El *encoder* consta de una *CNN* profunda para extraer las características de la imagen. Luego una red neuronal recurrente, en este caso una red *LSTM* bidireccional (*BLSTM*) para extraer dependencias temporales, y una red convolucional de dimensión 1 que tiene por objetivo generar la secuencia de vectores de estado final. Esta secuencia pasará a través del componente de atención, que la transformará, en función del mecanismo de atención utilizado y en cada salto de tiempo, en el vector de contexto que llega al componente de decodificación, compuesto por una red *LSTM* unidireccional. Este enfoque está basado en aquel llamado atención monótona (Raffel et al., 2017) extendido de forma tal que este reciba los pesos usados en el proceso de atención del salto anterior para ponderar el vector de estados al generar el vector de contexto. Esto es llamado atención híbrida (Chorowski et al., 2015).

Enfoques previos a los nombrados hacen uso redes *LSTM* multidimensionales (*MDLSTM*) (Graves, Fernández & Schmidhuber, 2007) dado que estas explotarían la naturaleza en dos dimen-

siones de las imágenes de texto. Bluche (2016) presenta un modelo *encoder-decoder* para transcribir texto escrito a mano a nivel de párrafos en el que el módulo de codificación está conformado por una pila de capas *LSTM* multidimensionales que se encargaran de transformar la imagen de entrada en un vector de características. Sin embargo, usar redes multidimensionales tiene un costo de entrenamiento mucho mayor al de usar redes convolucionales, y una reducida mejora de rendimiento, según argumenta Puigcerver (2017).

Para el caso de la transcripción a partir de imágenes de texto impreso, puede verse que no hay una diferencia significativa en los modelos *Seq2Seq* presentados. Sahu et al. (2015) presentan un modelo *Seq2Seq* básico, sin mecanismo de atención. Este modelo supera distintos *frameworks* de *OCR* y también compite con el modelo estado del arte en su momento (Graves et al., 2006). Conjeturan que extender el modelo haciendo uso de algún mecanismo de atención puede mejorar el rendimiento, lo cual parecería ser cierto observando las referencias anteriores.

Puede observarse en la Figura 2.14, como intento de descripción general de los modelos presentados, que los modelos *Seq2Seq* para tareas de OCR que se encuentran en el estado del arte son aquellos que siguen la arquitectura *encoder-decoder* y hacen uso de mecanismos de atención. Primero un módulo codificador conformado por una red para extraer las características de la imagen de entrada (generalmente una red convolucional) y una red recurrente que transforma la secuencia de vectores de características obtenida a una secuencia de vectores de estado. Luego, el decodificador predice en cada salto de tiempo un nuevo símbolo de la secuencia de salida, tomando como insumo la predicción del salto de tiempo anterior, el estado oculto del salto anterior y la salida del módulo de atención correspondiente al salto de tiempo actual. El módulo de atención toma por entrada, en cada salto de tiempo, la secuencia de vectores de estado obtenido por el codificador junto con el estado oculto actual del decodificador (por ende el correspondiente a la predicción anterior) para dar por resultado un vector llamado vector de contexto, que está conformado por la suma ponderada de los vectores del codificador.

Cabe destacar que si bien los modelos en los que se está haciendo énfasis son los *Seq2Seq encoder-decoder*, no son los únicos que presentan un rendimiento competitivo. Desde que Graves et al. (2006) presentó el método *CTC*, múltiples enfoques han ido por el camino de una arquitectura que combina una red neuronal recurrente con el método *CTC*. Kang et al. (2022) presentaron recientemente una nueva arquitectura basada en *transformers* (Vaswani et al., 2017) que prescinde del uso de redes recurrentes, haciendo uso solo de mecanismos de atención, logrando competir con modelos *Seq2Seq encoder-decoder* y *CTC*, igualando o superando su rendimiento. Es interesante la sección de comparación de modelos del estado del arte de Kang et al. (2022) para poder observar otros modelos no *Seq2Seq* que se encuentran compitiendo dentro del área de reconocimiento de texto escrito a mano.

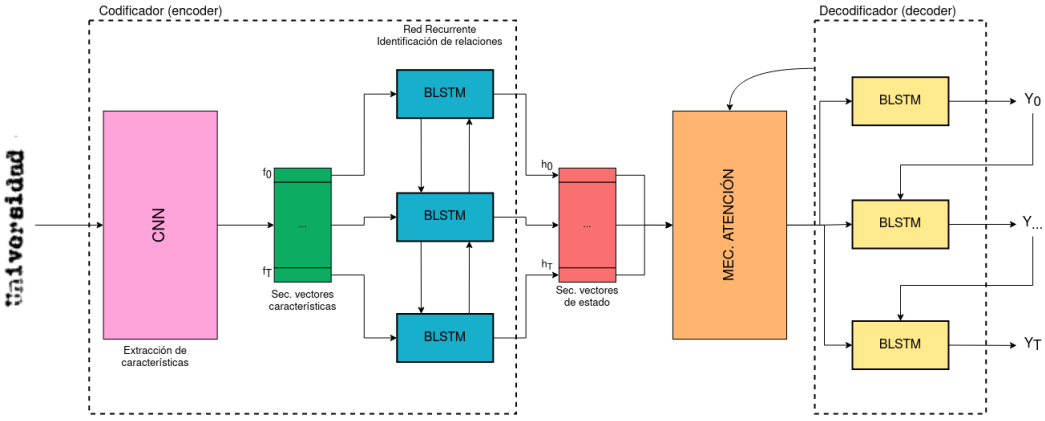


Figura 2.14: Esquema que describe de forma general los modelos *Seq2Seq* observados en el estado del arte para reconocimiento óptico de caracteres. Primero la imagen de entrada pasa por el módulo codificador que involucra una red convolucional para la extracción de características (en algunos artículos puede verse a esta red fuera del decodificador, pero no es una diferencia operativa), la secuencia de salida se usa como entrada en una red recurrente (*LSTM*, *LSTM* bidireccional, *Gated Recurrent Unit*). Luego un mecanismo de atención hace uso de la salida del codificador junto con el estado actual del decodificador para formar el vector de contexto. Finalmente el decodificador predice el siguiente símbolo en función de este vector de contexto y la predicción anterior. Los mecanismos de atención usados varían, así como el tipo de unidad recurrente y otros detalles como dimensiones de los vectores y cantidad de capas usadas en las redes del codificador.

Resumen

El reconocimiento óptico de caracteres, abreviado *OCR*, es el conjunto de métodos que tiene por objetivo la transcripción automática de texto contenido en imágenes. Existen múltiples arquitecturas destinadas a OCR, hoy principalmente basadas en aprendizaje profundo, mediante el uso de redes estándar, convolucionales (*CNN*) y recurrentes, particularmente *LSTM*. La arquitectura estudiada, *Seq2Seq encoder-decoder*, está compuesta por una red convolucional que extrae características de la imagen de entrada, una red recurrente que identifica relaciones de dependencia temporal entre estas características y un decodificador que predice la secuencia de símbolos de texto que reconoce en la imagen, apoyándose para cada predicción en un mecanismo de atención que sirve para tomar de la salida del codificador la información que más valor aporte para la predicción.

En el siguiente capítulo se hace un pasaje por el proceso de obtención de cada uno de los conjuntos de datos utilizados para entrenar el modelo con la arquitectura estudiada, explicando, en cada caso, las complejidades encontradas y de qué forma fueron resueltas. Estos conjuntos fueron construidos a partir de la base de datos de transcripciones manuales hechas por personas voluntarias en la plataforma colaborativa *LUISA*.

Capítulo 3

Datos de entrenamiento

Como ya fue mencionado anteriormente, el proceso mediante el cuál se generan transcripciones manuales de las páginas disponibles es segmentándolas en porciones llamadas *bloques*. Estos bloques tienen como objetivo cubrir una palabra, lo que se cumple en la mayoría de los casos, ya que algunas veces, por confusión del algoritmo de segmentación, un bloque puede abarcar menos de una palabra o varias palabras juntas. Luego, un conjunto de bloques adyacentes son presentados al voluntario o la voluntaria que ingresa a la plataforma de *LUISA* y esta rellena con texto aquello que logra leer en ellos. Como resultado, *LUISA* posee una base de datos cargada con transcripciones manuales de bloques de páginas. De hecho, debido a que cada bloque es presentado más de una vez en la plataforma, se tiene para muchos bloques más de una transcripción asociada. Es a partir de esta base de datos que se construyen los conjuntos utilizados para entrenar los modelos.

El primer conjunto construido es un conjunto de bloques con una transcripción seleccionada para cada uno de entre las múltiples disponibles. Luego, se construye un conjunto de bloques con todas sus transcripciones disponibles. Finalmente, se construye un conjunto de elementos cuya secuencia de texto objetivo corresponde a una línea de texto más que al de una palabra. La generación de las líneas se hace a partir de la conjunción de bloques adyacentes, generando el texto correspondiente a partir de la unión de las transcripciones de cada bloque usado. En el proceso de generación de líneas existen algunas complejidades identificadas y atendidas. También se cuenta con un conjunto de líneas curadas por el equipo técnico de *LUISA*. En el presente capítulo se describen los distintos conjuntos y la forma en la que se los obtuvo.

3.1 Construcción de conjunto de bloques

Al momento de la obtención del *dump* de la base de datos de *LUISA*, la misma está conformada por 644245 bloques. Sin embargo, no todos estos bloques se consideran aptos, ya que muchos de

ellos no cumplen con el criterio de “estar completo”.

Este criterio ha sido modificado en varias ocasiones en el avance del proyecto *LUISA*. Algunos de los criterios han sido:

- que el bloque tenga al menos 3 transcripciones iguales o 5 transcripciones diferentes;
- 2 transcripciones iguales o 3 diferentes;
- en un principio no había criterio definido para bloque completo, por lo que existen bloques con más transcripciones.

En la Figura 3.1 puede observarse la cantidad de bloques completos que existen según la cantidad de apariciones que tuvieron en la plataforma *LUISA*.

Hay que tener en cuenta que existen características de los bloques completos que no ayudan al correcto entrenamiento del modelo.

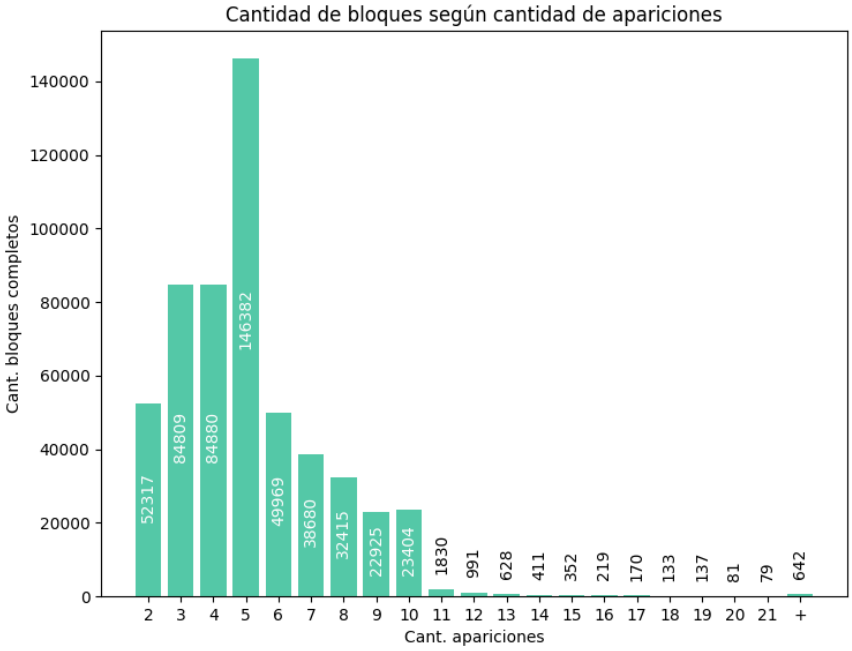


Figura 3.1: Histograma representativo de la cantidad de bloques que se consideran completos (no volverán a aparecer en la plataforma para ser transcritos) según la cantidad de apariciones que tuvieron. Es oportuno observar que en un principio no existía criterio de completo, por lo que hay bloques que aparecieron más de 5 veces.

La primera de ellas está relacionada con las reglas asociadas a la transcripción de los bloques en la plataforma.

A la hora de transcribir un bloque en *LUISA*, existe un conjunto reducido de reglas que es necesario seguir y están asociadas con aquello que es posible interpretar en la imagen y aquello que no. Las reglas que interesan para entender la naturaleza de las transcripciones disponibles son:

- Escribir todo lo que se lee: incluyendo acentos, comas y demás símbolos legibles en la transcripción.
- Si la imagen muestra texto que no es legible, corresponde escribir un símbolo arroba (@).
- Si alguna de las palabras o números de la imagen no son legibles, pero otras sí, escriba lo que se pueda leer y utilice un símbolo arroba (@) en la posición de cada palabra ilegible.
- Si la imagen muestra algo que no es texto (manchas, firmas, o puntos de un formulario), el cuadro de texto debe quedar vacío (transcripción vacía).

Por consecuencia, existen en la base de datos de *LUISA*: transcripciones que son vacías, que están compuestas solo por el símbolo arroba (@), o transcripciones donde alguna de las palabras fue transcrita con el símbolo arroba por no ser legible para el usuario o la usuaria. Además, dado que no está especificado qué hacer si es un solo carácter el que no es legible, se encuentran transcripciones que contienen arrobas en lugar de caracteres no legibles.

Otra característica importante es que hay muchos bloques que engloban un área de la página escaneada que no contiene texto. Se muestran algunos casos de ejemplo de estos bloques en la Figura 3.2. Estos son fácilmente reconocibles desde las transcripciones, ya que suelen ser vacías o arrobas.

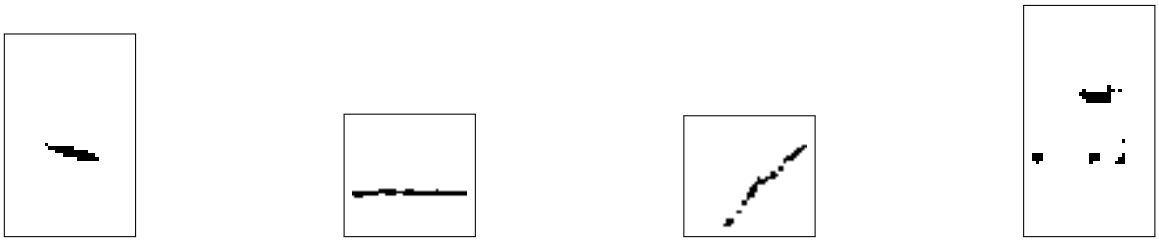


Figura 3.2: Casos de bloques que no cubren ninguna palabra de la imagen de la página que segmentan. Comúnmente tienen en la base de datos un conjunto de transcripciones asociadas que son vacías o contienen el símbolo arroba '@', representando que no hay nada o que no es legible. Para los cuatro casos mostrados aquí, sus transcripciones son vacías.

Por estas razones se decide filtrar todas las transcripciones que contengan al menos un arroba o sean vacías. De esta forma queda restringido el dominio del problema a bloques que tengan transcripciones sin símbolos no reconocidos, o mejor dicho, sin símbolos que representan texto no legible para el usuario o la usuaria que generó la transcripción.

Luego de aplicado el filtro sobre el conjunto inicial, se obtienen en total 359814 bloques con al menos una transcripción vinculada. Aproximadamente 66% del conjunto de bloques inicial (Figura 3.3), distribuidos según cantidad de transcripciones, como se muestra en la Figura 3.4.

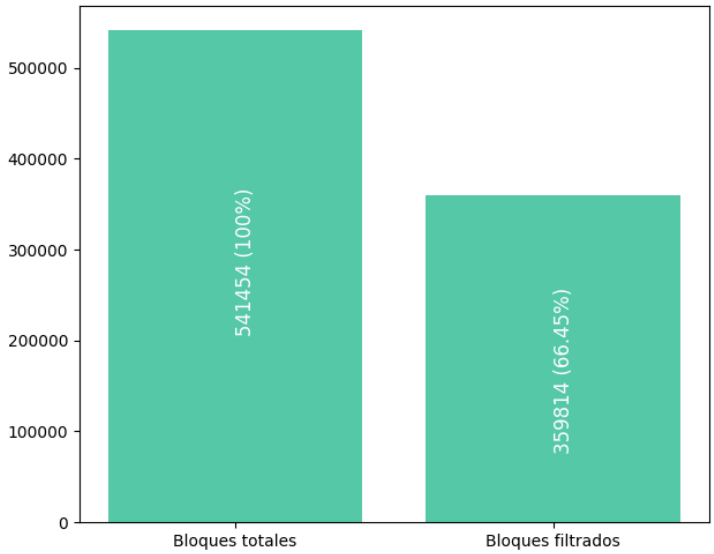


Figura 3.3: Proporción de bloques totales y bloques válidos luego de aplicado el filtro

Para la obtención de la imagen de cada bloque se utiliza la imagen de la página correspondiente junto con información de la ubicación. Tanto la información del bloque como la de su página se encuentra en la base de datos de *LUISA*, teniendo para cada uno de ellos el nombre de la página y dos coordenadas que especifican la posición del vértice superior izquierdo y el inferior derecho, como se muestra en la Figura 3.5, permitiendo así recortar de la imagen de la página el recuadro que el bloque abarca.

Esto fue posible dado que el equipo de *LUISA* dio acceso no solo a la base de datos de los bloques con su información, sino también al conjunto de imágenes de las páginas del *Archivo Berrutti* que se procesaron en la plataforma.

Dado que para cada bloque existe más de una transcripción posible, se toma la decisión de experimentar con dos conjuntos de bloques distintos. El primero es aquel donde para cada bloque se selecciona la transcripción que se considera más fiel al texto contenido en la imagen. Para esto es necesario generar una heurística de selección de lo que se considera la mejor transcripción. La segunda opción es la de tomar, para cada bloque, todas las transcripciones existentes y generar un conjunto que se puede considerar *aumentado* en relación con el primero.

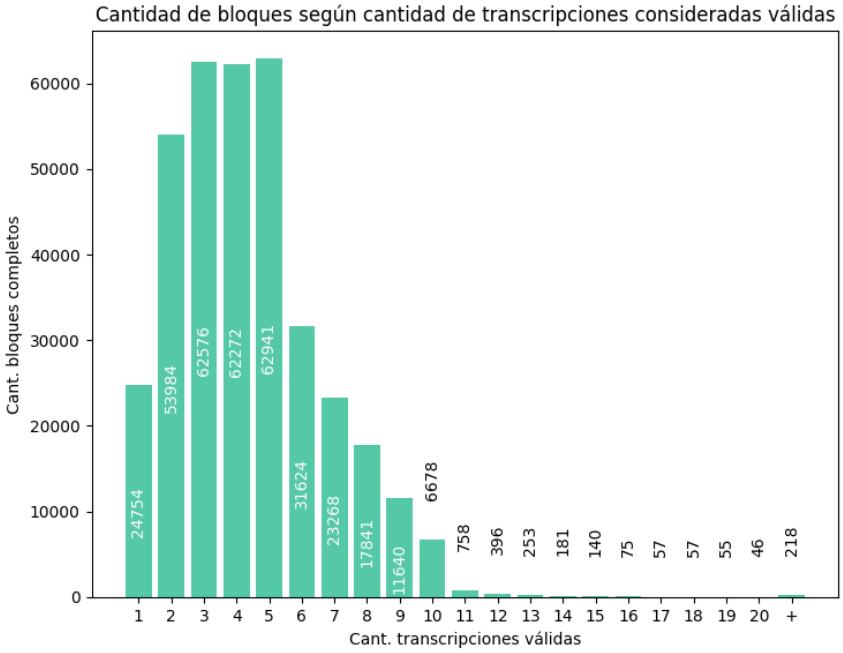


Figura 3.4: Histograma de cantidad de bloques según cantidad de transcripciones disponibles, luego de filtrarlas

3.1.1 Conjunto de bloques con transcripción seleccionada

Para construir el primer conjunto de bloques se genera una heurística de selección de transcripción, basándose en la ya implementada por el equipo técnico de *LUISA*. Queda definida en función de las siguientes reglas:

- Elegir aquella transcripción que tiene el mayor número de ocurrencias entre todas las existentes para ese bloque.
- Si existe un empate, dado por más de una transcripción posible con igual cantidad de apariciones, elegir aquella que cumpla tener la menor distancia con todas las demás, dado un criterio de distancia.

De esta forma queda definida la heurística mediante la que se elige la transcripción correspondiente a cada bloque:

- Para cada transcripción del bloque, se calcula la distancia *Levenshtein* con cada una de las

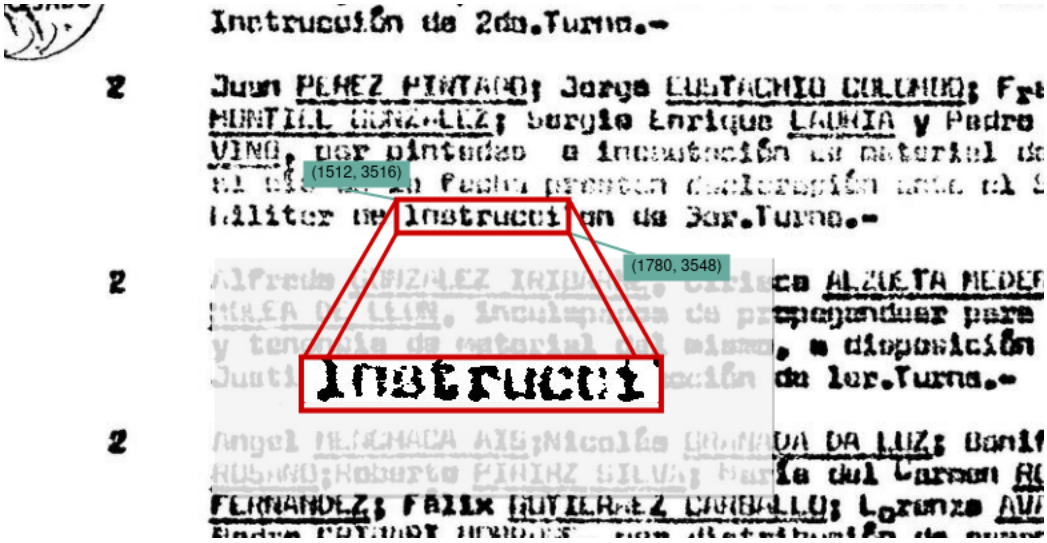


Figura 3.5: En la base de datos obtenida de *LUISA*, cada bloque tiene asociado dos coordenadas correspondientes a la esquina superior izquierda y la esquina inferior derecha que forman el rectángulo asociado al área del bloque en esa página.

transcripciones existentes. Esto genera una matriz de distancias de tamaño $n \times n$, siendo n la cantidad de transcripciones disponibles para el bloque.

- Se elige aquella que tenga la menor distancia promedio con todas las demás (Figura 3.6). Esta regla implica que si existe alguna transcripción repetida obtenga una distancia promedio menor, abarcando así el caso donde existe una transcripción que debe ganar por mayoría de apariciones, y también el caso donde existe empate en cantidades, pero hay una de ellas que tiene mayor similitud en distancia *Levenshtein* con todas las demás.

Las transcripciones disponibles para cada bloque no son modificadas de ninguna forma. Es decir, en el proceso de selección de la transcripción más adecuada no se les elimina ningún símbolo y se respeta la distinción entre mayúsculas y minúsculas a la hora de calcular las distancias.

Como resultado, se ha obtenido un conjunto de bloques con una transcripción asociada. Finalmente se procede a dividir el conjunto total de datos en tres subconjuntos de proporción 70%, 10%, 20% (Figura 3.7), llamados conjunto de *entrenamiento*, *validación* y *test* respectivamente. El conjunto de entrenamiento es el usado para entrenar los modelos, junto con el conjunto de validación que es usado para analizar la mejora en el entrenamiento de los modelos. El conjunto de *test* es utilizado para realizar una evaluación final a los modelos seleccionados y hacer comparaciones que permitan concluir sobre el rendimiento.

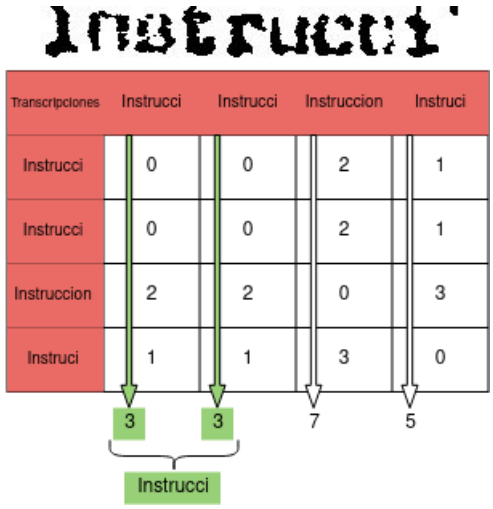


Figura 3.6: Proceso en el que se selecciona la mejor transcripción para el bloque. Primero se calcula la distancia *Levenshtein* de cada transcripción con todas las demás existentes. Luego, se elige aquella cuya suma de distancias sea la menor.

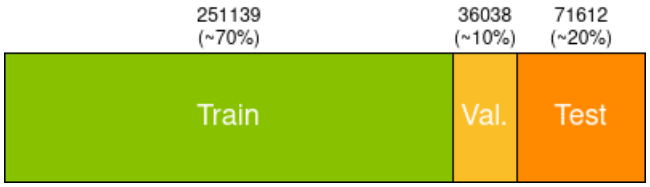
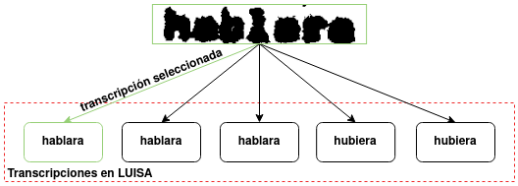


Figura 3.7: Proporción de los distintos subconjuntos usados

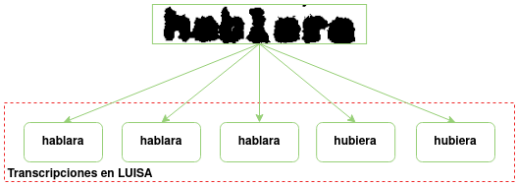
3.1.2 Conjunto de bloques con todas sus transcripciones válidas

Este conjunto de bloques se construye eligiendo para cada uno la totalidad de las transcripciones disponibles que no sean vacías ni contengan arrobas. La Figura 3.8 refleja la forma en la que se construye el conjunto de bloques con transcripción seleccionada y cómo se construye este conjunto, que será llamado *conjunto aumentado*, haciendo referencia a que aumenta con nuevos elementos a cada bloque del primer conjunto.

Con el objetivo de mantener la misma distribución de elementos con respecto al primer conjunto construido, se genera el conjunto aumentado agregando para cada elemento del primero todas las transcripciones que quedaron fuera de la selección, y se crean nuevos elementos bloque-transcripción en el conjunto aumentado, dando por resultado, para cada bloque, tantos elementos como transcripciones tenga. Los nuevos elementos son añadidos de manera que todas las transcripciones para un bloque queden juntas. No se realiza un reordenamiento aleatorio de los elementos (no se hace *shuffling*), ya que en pruebas preliminares, al entrenar modelos usando el conjunto aumentado



(a) Para el primer conjunto construido se elige una sola transcripción para el bloque en función de una heurística definida.

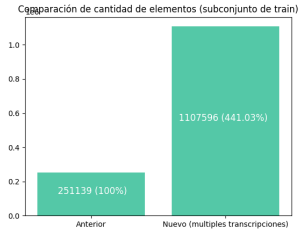


(b) Para esta prueba se genera un conjunto aumentado donde cada bloque tiene una entrada por cada transcripción no vacía y sin arrobas que tenga.

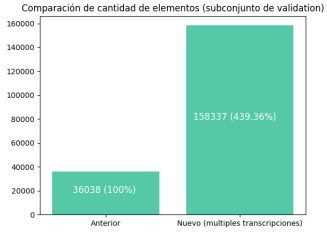
Figura 3.8: Comparación entre la manera de seleccionar transcripciones de los bloques antes y ahora.

previamente reordenado aleatoriamente, el modelo no aprendió.

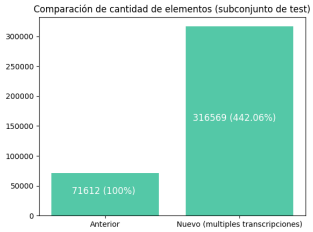
Los nuevos subconjuntos generados tienen muchos más elementos en comparación a los primeros. La Figura 3.9 refleja la comparación entre los anteriores y los nuevos obtenidos. Cada uno de los subconjuntos, al ser aumentados, crecieron con una relación cercana al 440% (Figura 3.10), por lo que se mantiene casi con exactitud la relación de proporción entre ellos: 70% entrenamiento, 10% validación, 20% *test*.



(a) Comparación subconjunto entrenamiento



(b) Comparación subconjunto validación



(c) Comparación subconjunto *test*

Figura 3.9: Comparación de las proporciones entre los subconjuntos anteriores y los obtenidos en este procedimiento.

3.2 Construcción de conjunto de líneas

Se construye un conjunto de entrenamiento cuyas imágenes contengan más texto que el que contiene un solo bloque. Un bloque tiene por objetivo contener una palabra. Sin embargo, es posible a partir del ensamblado de los bloques generar imágenes que contengan una línea de texto, obtenida a partir de la conjunción de las transcripciones correspondientes a cada bloque. Realizar pruebas de entrenamiento con líneas permite evaluar el potencial que tiene el modelo de beneficiarse de que

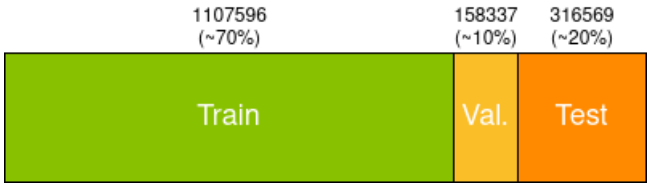


Figura 3.10: Comparación de las proporciones entre los subconjuntos anteriores y los obtenidos en este procedimiento.

existan múltiples palabras en la imagen. Una mayor cantidad de caracteres de texto corresponde a una secuencia de elementos de mayor tamaño, que podría beneficiar al modelo, si este tuviese la capacidad de aprender relaciones de mayor distancia.

3.2.1 Composición de la línea

Dada la forma en que los bloques fueron generados a partir de una página, es posible obtener todos los bloques pertenecientes a una misma altura en función de su coordenada en el eje vertical del vértice superior izquierdo, disponible en la estructura de información de los bloques. Los bloques pertenecientes a la misma línea de texto comparten la misma coordenada vertical.
De esta manera es posible ensamblar las líneas tomando todos los bloques para un mismo valor de Y en una misma página, luego ordenarlos según la coordenada en la que comienzan (eje horizontal del vértice superior izquierdo) y unir el texto correspondiente a la transcripción seleccionada para cada uno de ellos. La Figura 3.11 expresa visualmente el primer enfoque posible, y el más sencillo de todos.

Sin embargo, dentro del conjunto de páginas pueden observarse distintas formas en la que el texto está organizado. La forma canónica para este enfoque sería aquella donde la página contiene solo texto, ocupando todo el espacio y en una sola columna. Sin embargo, existen páginas que contienen tablas, o más de una columna de texto, u otros elementos visuales como pueden ser sellos y recuadros. La Figura 3.12 muestra el caso de una página donde existen dos columnas de texto. Cada una de ellas contiene oraciones no relacionadas. Ensamblar una línea de texto concatenando la transcripción seleccionada de cada bloque nos daría como resultado una secuencia de palabras que de leerse no tendría sentido, y esto es algo que no se desea, ya que se busca que el modelo *Seq2Seq* aprenda relaciones existentes entre los elementos de la secuencia.

A partir del análisis de este tipo de problemas se decide ensamblar bloques vecinos que respeten un criterio de distancia para aumentar la probabilidad de que el texto correspondiente a la transcripción tenga sentido.

Se define que dos bloques son *adyacentes* si pertenecen a la misma línea, suceden consecutivamente, no hay ningún bloque entre medio, y la distancia horizontal entre el vértice inferior derecho del primer bloque y el vértice superior izquierdo del siguiente está a menos de un umbral

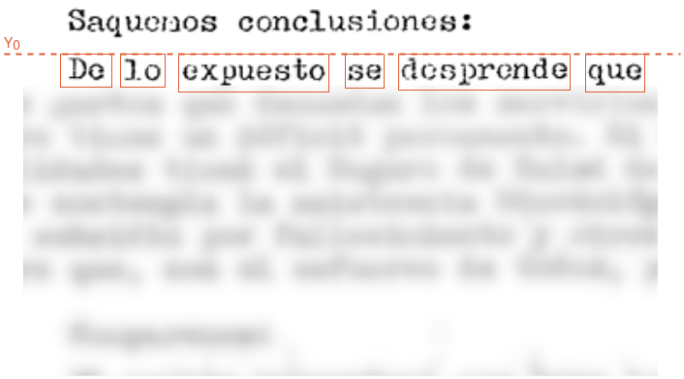


Figura 3.11: Primera alternativa para ensamblado de líneas. Se toman todos los bloques con la misma coordenada del eje vertical en su vértice superior izquierdo. El texto de la línea se deduce como la concatenación de las transcripciones de los bloques ordenados según su posición horizontal.

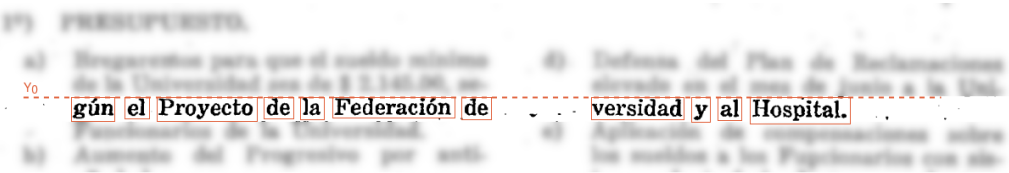


Figura 3.12: Problema que se presenta en una página con dos columnas de texto si se desea ensamblar líneas abarcando todos los bloques comenzando en la misma coordenada vertical.

u definido (Figura 3.13). La imagen 3.14a muestra las cantidades acumuladas según el umbral. La imagen 3.14b muestra cómo se distribuyen los bloques vecinos según el umbral de distancia. Se puede observar que para un umbral de 30 ya son el 85% de los bloques vecinos los que pueden ser considerados adyacentes. Además, de la distribución de bloques vecinos se puede observar que para umbrales mayores a 30 se podría estar cayendo en casos correspondientes a una minoría alejada de la norma, donde ya es muy probable que se incluyan casos donde en realidad los bloques vecinos corresponden a líneas distintas, conjuntos de bloques cuyo textos pertenecen a diferentes oraciones. Se considera entonces que 30 píxeles de distancia es un umbral adecuado para tomar como criterio de adyacencia.

Como ejemplo, volviendo al caso de una página con más de una columna de texto —que puede verse en la Figura 3.15— para la franja de bloques del ejemplo se tendrán dos posibles líneas, ya que hay dos conjuntos de bloques adyacentes que se ven separados en un momento por una distancia mayor al umbral, generando la finalización de una línea y el comienzo de otra.

A la hora de generar el texto correspondiente a una línea, el modo más simple es el de unir la

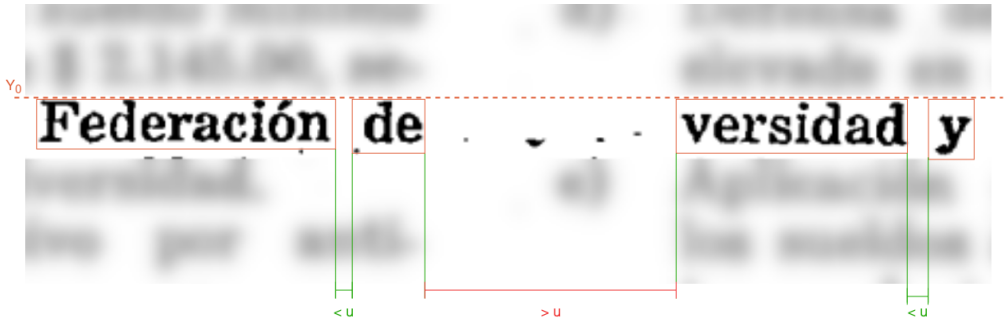


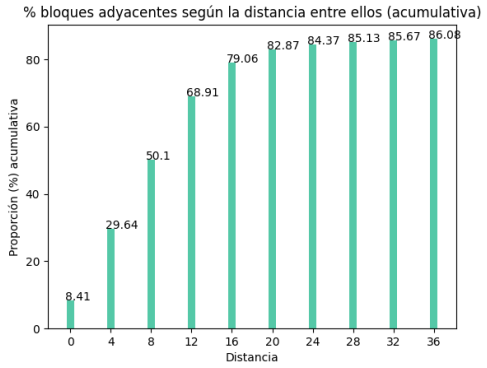
Figura 3.13: Definición de bloques adyacentes por umbral. Se desea definir un umbral u utilizado para definir como bloques adyacentes todo par de bloques vecinos cuya distancia horizontal sea menor al umbral definido. Los bloques vecinos considerados adyacentes son usados para ensamblar una misma línea, no así los considerados no adyacentes.

transcripción de cada bloque que la conforma separado por un espacio. Sin embargo, existen dos complejidades identificadas que pueden afectar la calidad de los datos generados: las transcripciones de bloques no deseadas, donde la transcripción que realizó el usuario o la usuaria no sigue el objetivo del problema, afectando el texto que corresponde a la línea, y casos donde existen palabras que fueron subsegmentadas en varios bloques. En las secciones siguientes se analizan estos problemas.

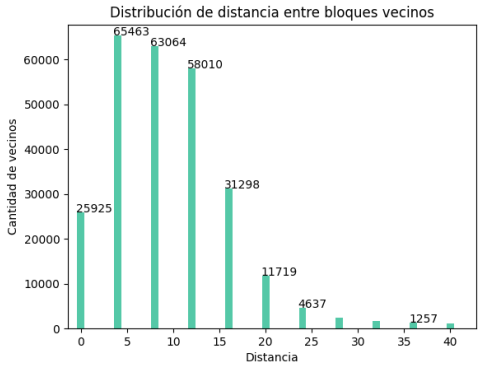
3.2.2 Filtrado de bloques con transcripción no deseada

La primera complejidad está relacionada con transcripciones no deseadas. Existen bloques cuya transcripción vinculada no es el texto que se encuentra en la imagen, sino una descripción de lo que se ve en ellas. Por ejemplo, se pueden ver bloques cuya transcripción es «mancha», «puntos de tinta», «raya del lado derecho», «firma». Sin embargo, el bloque no contiene ese texto, sino una mancha, puntos de tinta, etc. Este tipo de transcripciones no son deseadas a la hora de ensamblar líneas, ya que el texto correspondiente a la línea quedaría con palabras que no representan el texto. Una característica común en muchos de estos casos es que el bloque corresponde a simplemente una mancha, o puntos, o líneas, por lo que la longitud del bloque es más bien pequeña.

Como un intento posible para atacar este problema se propone un enfoque estadístico, que permita tomar un criterio de exclusión del conjunto de la transcripción que no pertenezca a la norma. Se genera una distribución de longitud de secuencia de texto (de la transcripción del bloque) por cada ancho de bloque observado. Es decir, para cada ancho, se calcula la media y la desviación estándar de longitudes de las transcripciones de todos los bloques que tienen ese ancho. De esa forma se puede tener una idea de qué longitud de texto es esperada para un bloque con un ancho dado, y luego es posible filtrar transcripciones que son muy sospechosas por tener una longitud mayor a la esperada.



(a) Cantidad acumulada de bloques vecinos con una distancia menor al umbral de distancia



(b) Distribución de bloques vecinos con distancia menor a umbral de distancia

Figura 3.14: Análisis de distribución de bloques según distancia de adyacencia

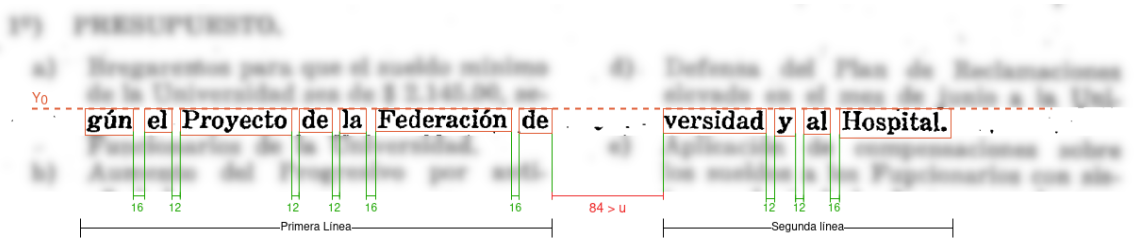


Figura 3.15: Ejemplo de página con dos columnas de texto y como funciona el ensamblado de líneas de bloques adyacentes.

Se define un umbral que queda definido como $\mu + 3\sigma$, siendo μ la media de la distribución correspondiente y σ la desviación estándar. Los elementos cuya longitud de transcripción están por sobre este umbral corresponden a menos del 0.3% de los casos observados. Podría entenderse a este conjunto de transcripciones como el 0.3% más alejado de la masa de observaciones. La Figura 3.16 muestra algunas de las distribuciones halladas. La Figura 3.17 muestra algunos ejemplos de bloques que fueron filtrados dado que la transcripción vinculada era muy irregular.

3.2.3 Bloques que contienen sub-palabras

La segunda complejidad está relacionada con el texto que abarca cada bloque. Existen muchos casos donde el bloque generado no abarca la palabra entera, sino que el algoritmo que los genera reconoció la separación entre dos caracteres de la palabra como un espacio, generando un bloque para un subconjunto de caracteres de la palabra y no para la palabra entera. Si al generar el texto correspondiente a una línea ensamblada se tienen bloques de este tipo, se obtendría por resultado

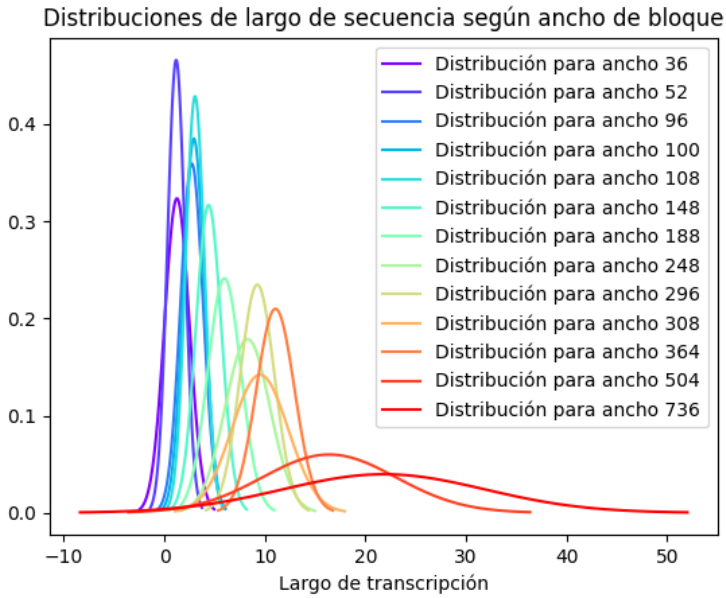


Figura 3.16: Distribuciones de largo de transcripciones para cada ancho de bloque posible. Se observa como a medida que aumenta el ancho es esperable que el largo de la transcripción tenga una mayor variación de largos posibles.

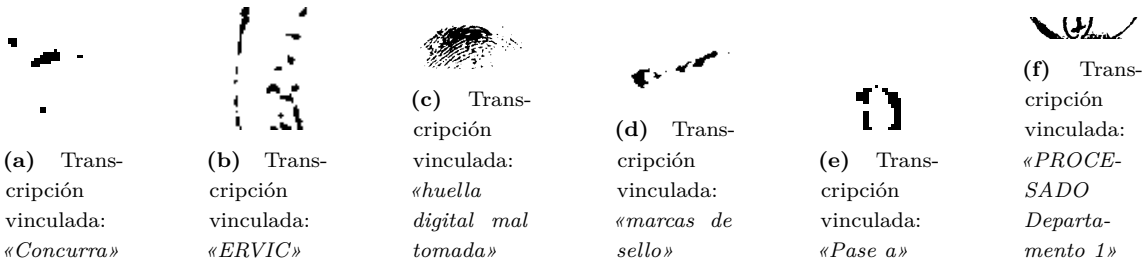
un texto que tiene espacios entre tokens que son segmentos de palabras y deberían estar juntos, formando una palabra entera.
La Figura 3.18 muestra un ejemplo donde sucede el problema mencionado. La línea está conformada por las palabras «Saquemos» y «conclusiones». Sin embargo, el área donde se encuentra la palabra «conclusiones» está segmentada en 3 bloques distintos, uno con el texto «concl» otro con el texto «us» y otro con el texto «iones». Todos los bloques satisfacen el criterio de adyacencia entre ellos, por lo tanto el texto final de la línea ensamblada, en caso de ser la unión de las transcripciones de los bloques separada por un espacio quedaría «Saquemos concl us iones:», que es erróneo.

En el Proyecto de Grado sobre post-procesamiento de *OCR* para detección y corrección de errores (Stabile et al., 2020) enfrentan el mismo problema con una heurística propuesta, que hace uso de un léxico para reconocer posibles palabras que han sido separadas en más de un bloque.

Esta heurística comienza definiendo un conjunto de umbrales de distancia. Los umbrales se utilizan para generar grupos de bloques vecinos y luego se clasifican los grupos según la aparición de los tokens separados y los tokens unidos sin espacios dentro de un diccionario de palabras.

Las categorías posibles definidas para los grupos son las siguientes:

- **Caso 1**: grupos de bloques donde las transcripciones por separado no pertenecen al diccionario,



(a) Transcripción vinculada: *«Concurra»*

(b) Transcripción vinculada: *«ERVIC»*

(c) Transcripción vinculada: *«huella digital mal tomada»*

(d) Transcripción vinculada: *«marcas de sello»*

(e) Transcripción vinculada: *«Pase a»*

(f) Transcripción vinculada: *«PROCESADO Departamento 1»*

Figura 3.17: Ejemplos de bloques que fueron dejados afuera a la hora de generar líneas ensambladas por tener una relación entre ancho y largo de transcripción demasiado irregular.

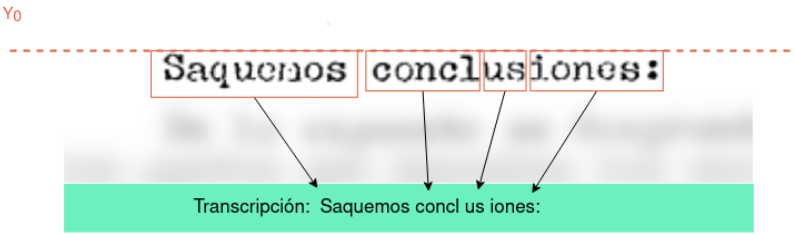


Figura 3.18: Ejemplo de ensamblado de línea con más de un bloque para una palabra.

pero la unión de ellas sí lo hacen. Este es el caso más obvio de unir.

- **Caso 2**: es un conjunto formado según el umbral donde todos los tokens por separado pertenecen al diccionario de palabras, pero la palabra formada de unirlos *no* pertenece. No nos interesan estos casos. De hecho, de unirlos se estarían perdiendo palabras originalmente válidas y generando otras no válidas.
- **Caso 3**: los tokens del grupo por separado no pertenecen al diccionario y tampoco lo hace la palabra generada al unirlos. No se unen estos casos. Puede pasar mucho en nombres propios que fueron cortados, pero no habría forma a priori de demostrar que unir todos los casos es una buena decisión.
- **Caso 4**: algunos tokens por separado pertenecen al diccionario, otros no, pero la unión de todos *sí* pertenece. En este caso es discutible si se quiere unir al final las transcripciones en una sola palabra, dado que podríamos estar cayendo en casos donde se unen palabras válidas en otra que es válida también («a» y «la» →«ala»).

Los grupos de tokens que son clasificados dentro del caso 1 se unen siempre en una palabra nueva porque es el caso ideal para hacerlo, ya que separados no son palabras reconocibles, pero el resultado de unirlos sí lo es. Los grupos que entran dentro de los casos 2 y 3, donde la unión de

ellos es incorrecta, no tiene sentido que sean unidos en una palabra sola, dado que esta no es una palabra reconocible, al menos según el diccionario de palabras usado.

Luego está el caso 4, que requiere un poco más de discusión. En este grupo podría existir una subcategorización que sería: primero, algunos de los tokens por separado pertenecen al diccionario y otros no, pero la unión de ellos si lo hace. En este caso se decide unirlos. La otra posibilidad es aquella donde tanto los tokens por separado como la unión de ellos pertenece al diccionario.

Este último caso es una línea gris, ya que pueden existir escenarios donde no se deberían unir las partes por separado, como lo es el caso de «a» y «la» en «ala», pero también existen otros donde sí valdría la pena unirlos, como «coman» «dante» en «comandante». Haciendo un análisis sobre esta complejidad se logran dos conclusiones importantes:

- La primera de ellas es que para los grupos que caen en este caso, las instancias donde realmente no se quiere unir las palabras están vinculadas con combinaciones de tokens usados con frecuencia, como pueden ser «a» y «la», «es» y «o», «ser» y «es», «tal» y «la», etc. Por lo que podría hacerse un análisis sobre una cantidad grande de texto en búsqueda de las palabras más frecuentes para eliminarlas del diccionario utilizado para esta etapa.
- Se observó que en la mayoría de los casos vistos de forma manual, los bloques que deberían estar unidos se encuentran a una distancia de 0 o -4 píxeles, siendo el -4 una superposición de 4 píxeles.

De este modo, sería posible reducir la franja gris permitiendo unir bloques de esta categoría cuando la distancia es menor o igual a un umbral de distancia 0 y la palabra resultante de unirlos pertenece a un diccionario donde se filtraron aquellas palabras que muy comúnmente pueden ser formadas por unir tokens válidos.

Para generar este conjunto de palabras que se quieren filtrar del diccionario se procesa un conjunto formado por más de 6 millones de oraciones extraídas de Wikipedia (Cadavid, 2009). Se toman los conjuntos de 2-gramas para cada oración, se forman con ellos pseudopalabras (eliminando el espacio entre ellos) y se lleva un conteo de la frecuencia de ocurrencia de cada una. Como ejemplo, dada la oración del conjunto «Es por eso que aún hoy se busca la verdad» se tienen los 2-gramas «Es por», «por eso», «eso que», «aún hoy», etc. Luego se forman las pseudopalabras uniendo los tokens de los 2-gramas y se obtiene «Espor», «poreso», «esoque», «aúnhoy», etc.
Luego se toma el conjunto de pseudopalabras que acumula el 50% de las apariciones, que en el caso de este experimento es equivalente al 1% del total de pseudopalabras generada. Finalmente se elimina este conjunto de palabras del diccionario de palabras que se va a usar y cambia la heurística de forma tal que se permita unir bloques cuando se encuentran a una distancia igual o por debajo del umbral de distancia 0.
Algunas de las palabras que fueron filtradas del diccionario original luego de este proceso: *«verla», «verlos», «lastres», «launa», «lay», «yala», «aboca», «acampo», «acierta», «aciertas», «aciertos», «acondiciones»*.

3.2.4 Ensamblado final y generación del conjunto de datos de entrenamiento

Finalmente, luego de filtrados los bloques con transcripción no deseada y de implementada la heurística usada para unir transcripciones de bloques que cortaron palabras, se procesa el conjunto de bloques para ensamblar líneas y generar el conjunto a nivel de líneas.
El conjunto resultante tiene la siguiente información para cada línea:

- ***Posición horizontal de comienzo de línea***: posición horizontal del comienzo de la línea en la imagen de la página a la que pertenece. Corresponde a la posición inicial del primer bloque que compone la línea.
- ***Posición vertical de comienzo de línea***: posición vertical del comienzo de la línea. Corresponde a la coordenada vertical que se usó para obtener el conjunto de bloques que empezaba a esta altura.
- ***Posición horizontal de fin de línea***: posición horizontal del final de la línea. Corresponde a la posición horizontal final del último bloque que compone la línea.
- ***Posición vertical de fin de línea***: posición vertical donde termina la línea. En este caso no es tan directo, ya que los distintos bloques pueden tener valores distintos, teniendo por consiguiente valores de *y1* distintos. Se toma el máximo y1 del conjunto de bloques. De esta forma se evita cortar texto válido.
- ***Identificador de hoja***: *hash* de la hoja cuya línea pertenece
- ***Nombre de la hoja***: nombre de la hoja a la que pertenece la línea
- ***Nombres del archivo de la hoja***: nombre del archivo formato *TIFF* del recorte de la línea
- ***Texto***: texto final que se construyó a partir de las transcripciones de los bloques que componen a la línea y que corresponde al texto de esta.

A la hora de procesar el conjunto de bloques para ensamblar líneas, de 358789 bloques procesados se filtraron 3632 bloques por contener una transcripción demasiado irregular con respecto a las distribuciones de relación entre ancho de bloque y largo de secuencia, y se formaron 7224 palabras nuevas resultado de unir bloques que posiblemente abarcaban menos de una palabra.

El conjunto de líneas final está compuesto por 95615 líneas. La media del largo de las transcripciones objetivo es de 23 caracteres y la desviación estándar es de 24.19 caracteres. Esto es casi unas 4 veces mayor en ambas medidas con respecto al conjunto a nivel de bloques, que tiene una media de 6.36 y una desviación estándar de 6.01.

3.2.5 Descomposición de líneas a bloques originales

Uno de los objetivos principales de la construcción de este conjunto es el de permitir volver a descomponer las líneas a los bloques que fueron usados para ensamblarlas, y de esta forma hacer posible distintas pruebas.

Dado que se guardó la información tanto de la página de la línea como la posición que tiene la línea en la página original, es posible, haciendo uso de la posición de los bloques, información que está disponible en la base de datos de *LUISA* y en el conjunto a nivel de bloques, volver a obtener los bloques que conforman la línea.

El algoritmo para volver a descomponer una línea es el siguiente:

- Para una línea L que se quiere descomponer, se obtiene la coordenada $y_{L,0}$ correspondiente a la posición vertical donde inicia la línea en la hoja y también el *hash* h de la hoja.
- Se obtiene del conjunto total de bloques (b_i) todos aquellos que estén en la hoja h y que comiencen a esa altura vertical (aquellos cuya coordenada $y_{b_i,0}$ es igual a $y_{L,0}$).
- Se ordena este conjunto de bloques según su posición horizontal inicial.
- Se toma el subconjunto que cumpla que el primer bloque tiene coordenada horizontal inicial igual a la coordenada horizontal inicial de la línea y el último bloque tiene coordenada horizontal final igual a la coordenada horizontal final de la línea.

De esta forma queda construido un conjunto de imágenes correspondientes a líneas de texto y que se pueden volver a descomponer en los bloques que las conforman, permitiendo realizar distintas pruebas que combinen estas características. Como ejemplo del tipo de dato resultante puede verse en la Figura 3.19 una línea obtenida de ensamblar bloques. Sabiendo que esta línea comienza verticalmente en el píxel 640, y que el ancho va desde el píxel 420 hasta 1292, se toman todos los bloques que cumplan encontrarse dentro de estas coordenadas, dando por resultado los bloques por separado que se muestran en la Tabla 3.1.

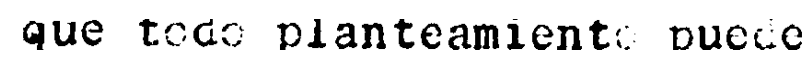
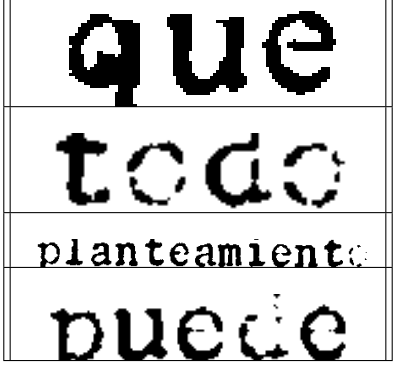


Figura 3.19: Ejemplo de imagen con línea de texto, resultado de ensamblar bloques adyacentes. Las coordenadas de esta línea en su hoja correspondiente es $y = 640, x_0 = 420, x_1 = 1292$, donde y hace referencia a la coordenada vertical donde comienza la línea, y x_0 y x_1 a las coordenadas de comienzo y fin horizontal de la línea.

3.3 Conjunto de líneas curado

Con el objetivo de obtener un conjunto de datos *ground truth* a nivel de líneas que sea de la mejor calidad posible, el equipo *LUISA* implementó una instancia de curado de líneas. Para

Bloques


Cuadro 3.1: Tabla de bloques resultantes de descomponer una línea del conjunto de líneas ensambladas. Es posible mediante la información de una línea y la información existente para los bloques obtener aquellos que conforman la línea.

esto, primero se generan las imágenes de líneas a partir de bloques a la misma altura en la hoja y se genera el texto *ground truth* como la concatenación de las transcripciones de cada bloque separadas por un espacio. Luego, estas líneas quedan a la espera de ser curadas. En este proceso de curación se busca eliminar texto indeseado, corregir errores de transcripción, errores generados al concatenar transcripciones, como pueden ser espacios entre bloques que contenían menos de una palabra, eliminación de sellos a los costados de las imágenes, entre otras características.

Dado que es un proceso de curado manual y con menos participantes que *LUISA*, al momento de obtenido el conjunto, este no contiene una gran cantidad de líneas en comparación a las totales que existen. El subconjunto de entrenamiento contiene 8982 elementos y el de *test* 1283 elementos.

El conjunto está conformado por archivos de extensión *TIFF* (*Tagged Image File Format*) al igual que las imágenes de bloques, que contiene el segmento de página correspondiente a la línea. También está disponible para cada imagen el texto de la transcripción (*ground truth*) obtenido desde *LUISA* y posteriormente curado manualmente.

Además, las imágenes de las líneas del conjunto han pasado por una etapa de procesamiento, que incluye la reducción de la dimensión a la mitad y transformación a escala de grises de 8 bits. Es decir, cada píxel de la imagen resultante tiene un color gris dentro de una gama de 256 grises en total, que es la cantidad de valores representables con 8 bits, por eso a esta configuración se le llama *8 bits per pixel (8bpp)*. La Figura 3.20 muestra un ejemplo de esto. Si bien originalmente las imágenes correspondientes a las páginas escaneadas son binarias (1 bit por píxel, negro o blanco), pruebas preliminares de entrenamiento realizadas por el equipo *LUISA* dieron mejores resultados con las imágenes procesadas de esta forma. Es por eso que se utiliza este conjunto.

Se procede a segmentar el conjunto de entrenamiento obtenido en subconjuntos de entrena-

miento y validación de tal forma que las proporciones queden de 70% para entrenamiento, 20% para validación y 10% *test*.

En este documento, el conjunto queda nombrado como *Calamari-Half8*. *Calamari* en relación a la herramienta usada por *LUISA* para entrenar el modelo actualmente usado, *Calamari-OCR* Wick et al. (2020), y *half8* por la reducción de dimensión y la transformación a escala de grises que han recibido las imágenes.

Este conjunto, al haber sido utilizado para entrenar el modelo de OCR que se usa actualmente en *LUISA*, sirve como medio de comparación con el modelo propuesto en el proyecto.

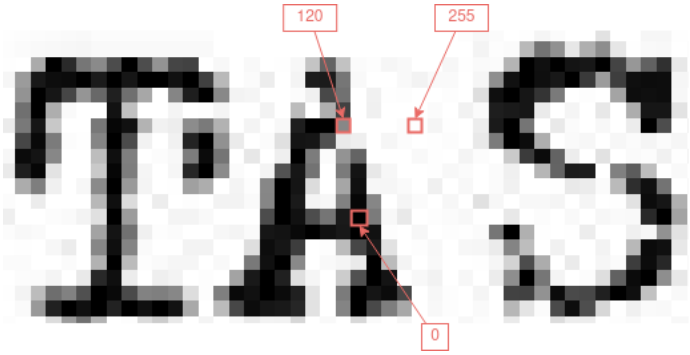


Figura 3.20: Ejemplo de imagen del conjunto de líneas *Calamari-Half8*, donde el valor de cada píxel oscila entre 0 y 255 para determinar 256 posibles valores de gris.

Resumen

A partir de la base de datos de transcripciones manuales realizadas en la plataforma colaborativa *LUISA* se construyen múltiples conjuntos de datos, usados para entrenar instancias del modelo propuesto.

A grandes rasgos, se obtienen conjuntos en dos tipos de granularidad distinto: a nivel de bloques y a nivel de líneas. Finalmente, se obtiene además un conjunto de líneas curadas por el equipo *LUISA*, con el que se ha entrenado la actual herramienta de *OCR* utilizada. Este conjunto permite hacer una comparación del rendimiento entre el modelo propuesto y la herramienta. La Tabla 3.2 resume los conjuntos nombrados.

En el siguiente capítulo se recorren los distintos componentes que forman la arquitectura *Seq2Seq* presentada y se especifican los detalles del modelo que se implementa para la experimentación.

Nombre del conjunto	Tamaño en imágenes	Leve descripción
Conjunto de bloques con transcripción seleccionada	358789 imágenes	Conjunto conformado por imágenes de bloques junto con una sola transcripción asociada, elegida a partir de una heurística de selección
Conjunto de bloques aumentado	1582502 imágenes	Conjunto conformado por imágenes de bloques junto con cada una de las transcripciones asociadas
Conjunto de líneas	95615 imágenes	Conjunto de imágenes generadas a partir de la conjunción de imágenes de bloques adyacentes, generando imágenes de líneas, y un texto asociado correspondiente a la conjunción de las transcripciones seleccionadas de cada uno de los bloques usados
Conjunto de líneas curadas	10265 imágenes	Conjunto de imágenes de líneas, igual al anterior, pero que ha pasado por un proceso de curado adicional, mejorando la calidad de la imagen y de la transcripción asociada. Es un conjunto mucho más reducido que el anterior, dado que no se ha completado el proceso de curado

Cuadro 3.2: Tabla de los distintos conjuntos de datos utilizados. Tener en cuenta que el conjunto de líneas construido es utilizado de diversas formas en los múltiples escenarios de prueba que hacen uso de este: solo líneas, líneas y sus bloques, bloques y luego líneas, entre otros.

Capítulo 4

Seq2Seq para la transcripción de documentos del Archivo Berrutti

Los modelos *Seq2Seq encoder-decoder* (codificador-decodificador) observados en el estado del arte para tareas de *OCR*, consisten en un módulo de extracción de características de la imagen de entrada, mediante el uso de una red convolucional profunda, junto con una red recurrente que transforma la secuencia de características a una secuencia de vectores de estado que busca identificar relaciones. Luego, introducido en arquitecturas más actuales, un mecanismo de atención, que transforma los estados de salida del codificador, en cada salto de tiempo, a un vector de contexto, hallado mediante el cálculo de la suma ponderada y según información pasada de ambos módulos. El decodificador, a partir de este vector de contexto y mediante el uso de una red recurrente, predice en cada salto de tiempo el siguiente símbolo de la secuencia objetivo. La Figura 2.14 expresa de forma general la arquitectura descrita.
En este capítulo se describen los distintos componentes y se presenta la especificación del modelo *Seq2Seq encoder-decoder* propuesto para implementar el modelo que será evaluado con los conjuntos de datos de entrenamiento construidos.

4.1 Extracción de características de la imagen de entrada

La intención, siguiendo la misma idea de los modelos *Seq2Seq* para *OCR* vistos, es codificar la imagen de entrada a una secuencia de vectores de características. Cada vector de características se construye a partir de las múltiples salidas de la red *CNN* usada de tal forma que todos los elementos del vector tienen el mismo *receptive field*. De este modo se obtiene en cada vector de la secuencia un conjunto de características de la misma área de la imagen. Además, se busca que cada *receptive field* sea un segmento vertical, como puede verse en la Figura 2.11, de la imagen de entrada, para

así generar una analogía entre el orden de los elementos de la secuencia y la forma en la que se lee naturalmente. El hecho de que a cada elemento de cada canal de salida le corresponda un *receptive field* de alto igual al de la imagen de entrada hace que cada columna de cada canal de salida de la *CNN* esté conformada por solo un elemento.

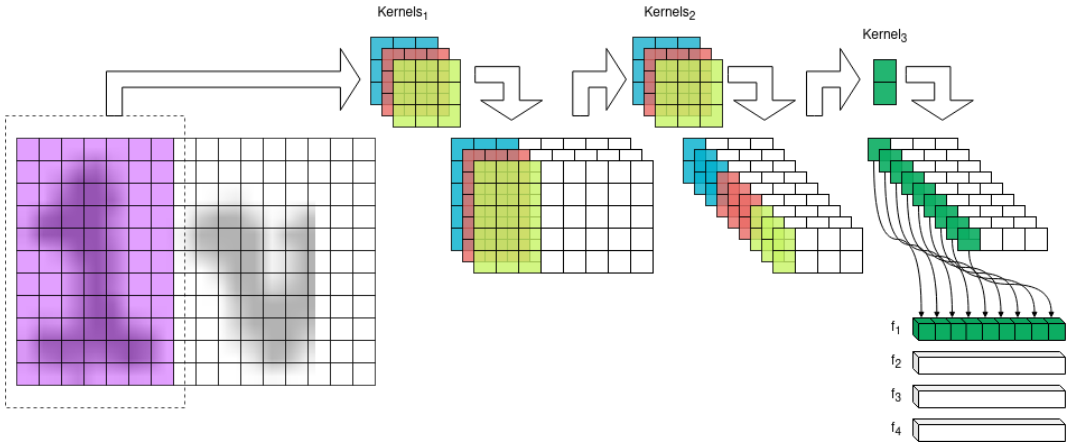


Figura 4.1: Cada imagen es procesada por las capas de la red *CNN*. Como resultado se obtiene un conjunto de canales de salida, dado que se procesa con más de un *kernel* en cada etapa, generando múltiples canales. Uniendo las columnas con el mismo índice de cada uno de los canales de salida, supongamos el índice 1, generamos el vector de características correspondiente, vector de características 1. Si además tenemos que los canales de salida finales solo tienen una fila, siendo *receptive field* de cada elemento un área que abarca todo el alto de la imagen de entrada y un fragmento de su ancho, entonces tenemos que el vector de características de salidas tiene también una sola fila y sus elementos corresponden a las características extraídas de ese espacio de la imagen.

Dado que cada capa de la *CNN* genera una transformación sobre el elemento de entrada que modifica su dimensión, y para lograr elementos finales cuyo *receptive field* ocupe todo el alto de la imagen de entrada, se fija el alto de la imagen en 64 píxeles, modificando el largo en caso de ser necesario para mantener el *aspect ratio* en función de las capas usadas y la configuración de los parámetros del *kernel* y el *stride* de la red convolucional. Las capas que conforman la red quedan especificadas como se muestra en la Expresión 4.1, donde C y P representan una capa convolucional y de *pooling* respectivamente, y $C/P^{S_y x S_x}_{K_y x K_x}$ representan las dimensiones del *kernel* K y el *stride* S en los ejes x e y.

$$C^{4x2}_{6x4}[8] -> C^{1x1}_{6x4}[32] -> P^{2x2}_{2x2} -> C^{1x1}_{4x3}[64] -> P^{2x2}_{2x2} \tag{4.1}$$

Esta configuración de red *CNN* utilizada para la extracción de características está basada en los artículos leídos sobre estos modelos (Chowdhury et al., 2018; Michael et al., 2019). Poca teoría

se encontró relacionada con qué tipo de configuración usar vinculada a *OCR*, y en los artículos estudiados no se prueban distintas configuraciones, sino que se utiliza una lo suficientemente grande como para que la red aprende a extraer características y un conjunto de capas que permita fijar las dimensiones de la imagen de entrada sin tener problemas.

4.2 Codificador

La secuencia de vectores de características obtenida a partir de procesar la imagen con la *CNN* será la entrada de la red recurrente que conforma el codificador. Esta se encargará de identificar relaciones existentes entre los elementos de la secuencia, que servirán luego para generar el vector de contexto usado por el decodificador para predecir la secuencia de caracteres correspondiente al texto contenido en la imagen.
El codificador del modelo a implementar que se propone estará formado por una red recurrente bidireccional de tres capas, usando unidades *LSTM* (una red *BLSTM*). Como puede verse en la Figura 4.2, la secuencia de salida de la *CNN*, secuencia de vectores de características, es usada como insumo de entrada para el codificador, obteniendo como resultado de este procesamiento un conjunto de vectores de estado. Dado que se usa una red *LSTM* bidireccional, la información de entrada es procesada en ambas direcciones, desde el comienzo hacia el final y desde el final hacia el comienzo, generando como resultado dos secuencias de vectores de estado interno de la red (uno por cada dirección). De este par de secuencias de estado se genera una sola, siento cada vector de esta el resultado de concatenar los vectores de estado del procesamiento del mismo elemento para ambas direcciones.

4.3 Decodificador

El decodificador es el módulo que está encargado de generar la secuencia objetivo de símbolos. En cada salto de tiempo se busca predecir el símbolo con mayor probabilidad condicionado a los símbolos predichos anteriormente y la información que se obtiene del codificador, información sobre las relaciones temporales de las características de la imagen de entrada.

Formalmente, el decodificador aprende una distribución probabilística sobre el conjunto de símbolos que conforman el diccionario utilizado. En cada salto de tiempo se da por salida un vector de probabilidades con un valor de probabilidad asignado a cada uno de los símbolos. Esto se modela haciendo uso de una capa con función de activación *softmax*. La distribución se aprende de forma tal que el siguiente símbolo elegido será aquel que maximice la probabilidad condicionada a los símbolos predichos anteriormente y la información de contexto, como puede verse en la Ecuación 4.2, donde V representa el conjunto de símbolos del diccionario usado, y_i la serie de símbolos predichos anteriormente y c_t la información de contexto en el salto de tiempo i.

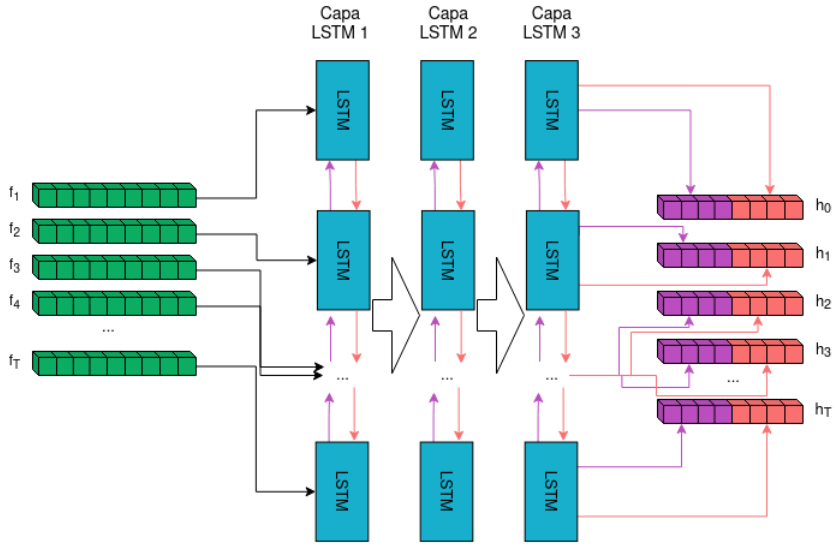


Figura 4.2: Arquitectura del codificador, formado por una red neuronal recurrente que usa celdas *LSTM* y compuesta por tres capas bidireccionales. Este recibe la secuencia de características obtenida por la *CNN* y se encarga de procesarla para identificar relaciones existentes. El resultado es una secuencia de estados que se usarán para generar el vector de contexto, insumo para el decodificador.

$$y_t = \arg\max_{y \in V} P(y|y_1, ..., y_{t-1}, c_t) \tag{4.2}$$

El decodificador se modela con una red recurrente *LSTM* unidireccional de una capa (Figura 4.3), ya que como se observó en modelos presentados en el estado del arte, hacer uso de redes recurrentes bidireccionales no aumenta el rendimiento, junto con una capa de red *fully connected* con función de activación *softmax* para generar la distribución probabilística sobre los símbolos existentes.

El uso del mismo vector de contexto para la predicción del elemento de salida en todos los saltos de tiempo puede provocar que información de valor para cada uno de los saltos de tiempo en particular quede perdida, ya que no queda contenida en este vector. Es por esto que empiezan a usarse mecanismos de atención que, en función de los vectores de estado del codificador, y teniendo en cuenta otros valores temporales como el estado anterior del decodificador, genera para cada salto de tiempo t una serie de pesos que se usan para obtener un vector de contexto c_t como la suma ponderada de los vectores de estados del codificador. De esta forma se le permite al modelo, en cada salto de tiempo, hacer uso de una zona parcial de la entrada que es de valor para la predicción del nuevo elemento, evitando la necesidad de comprimir toda la información del codificador en un vector de largo fijo.

Se modela la probabilidad condicionada como puede verse en la Ecuación 4.3, donde f hace referencia a la red recurrente usada, s_{t-1} al vector de estado de esta en el salto de tiempo anterior y $softmax$ a la función de activación que se aplica sobre el resultado para transformar la salida en valores probabilísticos.

$$P(y_t|y_1, ..., y_{t-1}, c_t) = softmax(f(y_{t-1}, s_{t-1}, c_{t-1}), c_t) \tag{4.3}$$

Los símbolos del diccionario procesados por el modelo deben ser codificados para poder manipularlos computacionalmente. Para esto se hace uso de una capa de *embeddings* a nivel de símbolos (*character embedding*), que transforma cada símbolo de texto a un vector de valores reales de dimensión específica que representa a ese símbolo en el espacio de símbolos usados. La dimensión definida es de 64, por lo que el largo del vector del símbolo codificado será ese largo.

En los distintos artículos estudiados pudo verse que existen pequeñas diferencias en las arquitecturas planteadas, relacionadas con cuál es la red que recibe como entrada el vector de contexto, o si se utiliza no solo el vector actual, sino también el anterior como entrada a alguna red del decodificador, método llamado *input feeding* (Luong et al., 2015) y que beneficia al modelo, al ayudarle a generar una noción del contexto local. Por ejemplo, en el caso de Michael et al. (2019), la red *fully connected* del decodificador, usada para predecir la probabilidad sobre los posibles símbolos de salida, recibe como entrada tanto el estado de la red recurrente como el vector de contexto actual, y la red recurrente recibe tanto el *embedding* del símbolo anterior predicho como el vector de contexto anterior. En el caso de Chowdhury et al. (2018) también se emplean ambos vectores de contexto, sin embargo, el vector actual es usado como entrada en la red recurrente. Otros artículos usan solo el vector de contexto actual, no usan *input feeding*. En el modelo propuesto se usa *input feeding*, igual a la Figura 4.3, la red *LSTM* del decodificador recibe como entrada no solo el *embedding* del símbolo anterior predicho, sino también el vector de contexto anterior.

4.4 Mecanismo de atención

El mecanismo de atención usado es uno de los primeros presentados, llamado mecanismo de atención basado en el contenido (*content-based attention*, Bahdanau et al., 2015).

Este consiste en calcular el vector de contexto c_t como se expresa en la Ecuación 4.4, donde α_{tj} representa la ponderación correspondiente al vector de estado h_j del codificador en el salto de tiempo t, hallado como se expresa en la Ecuación 4.5.

$$c_t = \Sigma_{j=1}^{T_c} \alpha_{tj} h_j \tag{4.4}$$

$$\alpha_{tj} = \frac{exp(e_{tj})}{\Sigma_{k=1}^{T_c} exp(e_{tk})} \tag{4.5}$$

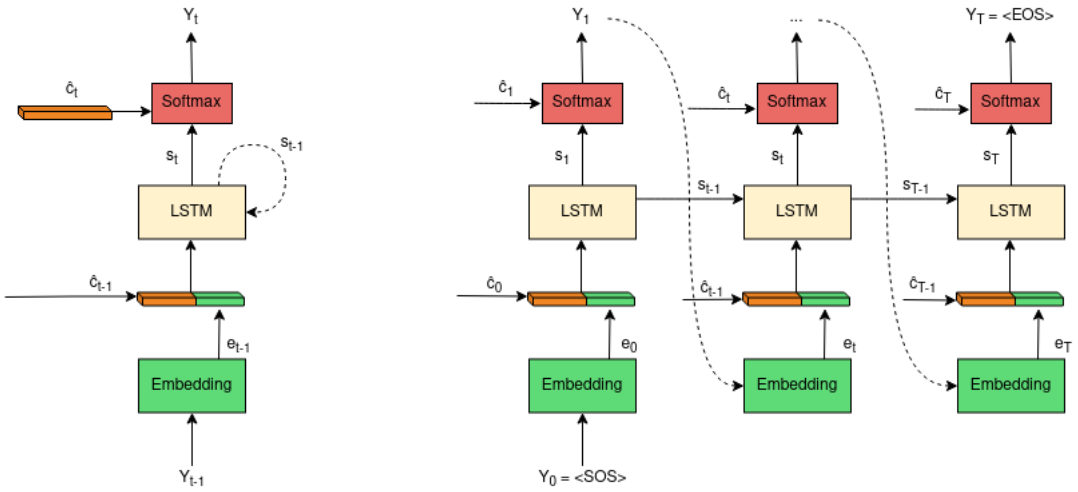


Figura 4.3: (a) Arquitectura del decodificador. En cada salto de tiempo, una red recurrente *LSTM* de una sola capa recibe el vector de contexto anterior c_{t-1} —método llamado *input feeding*—, el vector de estado actual s_{t-1} y el vector *embedding* del símbolo predicho en el salto anterior e_{t-1}. El vector de estado resultante de procesar esta información s_t es procesada por una capa *softmax*, junto con el vector de contexto actual $\hat{c}_t$, para dar por resultado un vector de probabilidades sobre el conjunto de símbolos posibles para elegir el símbolo y_t que maximiza la probabilidad. (b) Arquitectura del decodificador extendido en el tiempo. Si es el primer salto de tiempo recibe el *embedding* del símbolo adicional *<SOS>*, que hace referencia al comienzo de la secuencia. El decodificador repite el ciclo de predicción de símbolos hasta que se predice el símbolo *<EOS>*, que marca el final de la secuencia.

$$e_{tj} = a(s_{t-1}, h_j) \tag{4.6}$$

Siendo lo expresado en la Ecuación 4.6 el llamado vector de energías, obtenido a partir de una función a que puntúa qué tan similares son los vectores de estado alrededor de la posición j con la salida en la posición t. Por lo tanto, relaciona el posible símbolo de salida con la zona de vectores de estado que le corresponde. Se obtiene, en cada *timestep*, un valor de energía para cada vector de estado del codificador, usando siempre el vector de estado del decodificador resultante de predecir el símbolo anterior. En la Figura 4.4 se muestra la arquitectura del mecanismo de atención usado en la arquitectura propuesta.

La función de las ponderaciones y las energías quedan modeladas como se muestra en las Ecuaciones 4.7 y 4.8, donde v, W_s, W_h y b son parámetros entrenables:

$$\alpha_{tj} = softmax(e_{tj}) \tag{4.7}$$

$$e_{tj} = v^T tanh(W_s s_{t-1} + W_h h_j + b) \tag{4.8}$$

Basado en el artículo de Michael et al. (2019), se usa una red neuronal *fully connected* para combinar al vector de contexto generado junto con el vector de estado del decodificador, y así obtener un nuevo vector de contexto, que es el utilizado finalmente como entrada al decodificador.

La Figura 4.5 tiene la intención de reflejar visualmente cómo queda compuesta la arquitectura *Seq2Seq encoder-decoder* propuesta.

Resumen

De los múltiples enfoques existentes que hacen uso de la arquitectura *Seq2Seq encoder-decoder* para *OCR* se construye la arquitectura propuesta en este proyecto: un codificador formado por una *CNN* y una red *BLSTM* para extraer primero las características de la imagen de entrada y luego identificar relaciones de orden entre estas. Luego, un mecanismo de atención basado en contenido que busca identificar, en cada salto de tiempo, el grupo de vectores de estado del codificador que más valor suma a la predicción del siguiente símbolo. Finalmente, el decodificador, que hace uso del vector generado, junto con información extra del salto de tiempo anterior, para predecir la probabilidad sobre cada uno de los posibles símbolos de ser el elegido como el siguiente en la secuencia objetivo.

El siguiente capítulo está destinado a la especificación de la experimentación realizada. Se definen detalles de entrenamiento y evaluación, junto con los escenarios de prueba que serán ejecutados para medir el potencial del modelo propuesto.

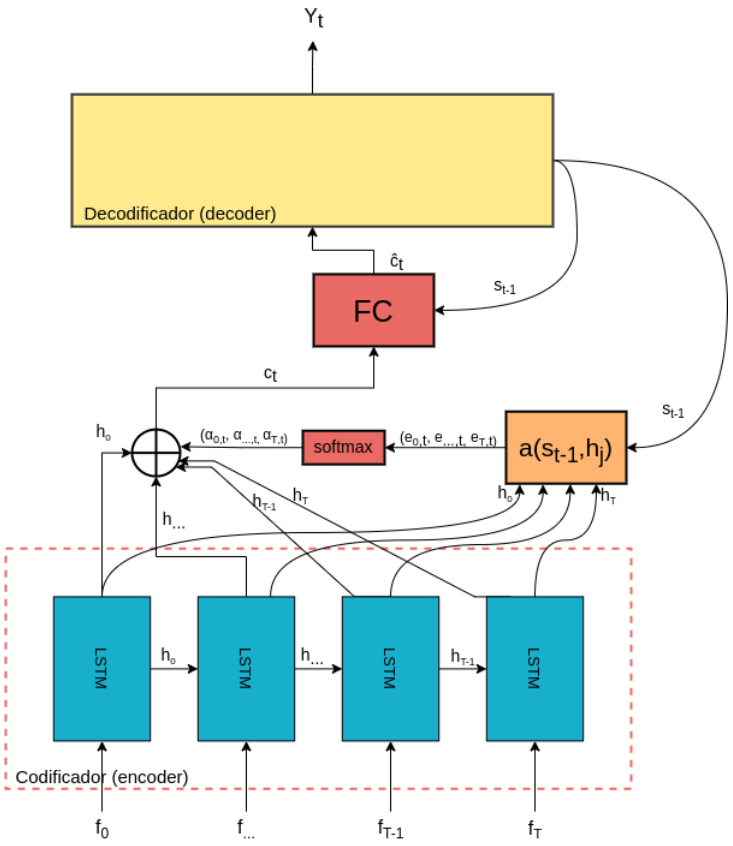


Figura 4.4: Mecanismo de atención basado en contenido, usado en la arquitectura propuesta. El vector de contexto se obtiene como una suma ponderada de los vectores de estado del codificador. El conjunto de ponderaciones se genera a partir de los vectores de estado del codificador y el vector de estado del decodificador a la hora de predecir el símbolo anterior. El vector de contexto hallado es luego procesdao por una red *fully-connected* junto con el vector de estado del decodificador en el salto de tiempo anterior. Este vector resultante es el usado finalmente para predecir el siguiente símbolo.

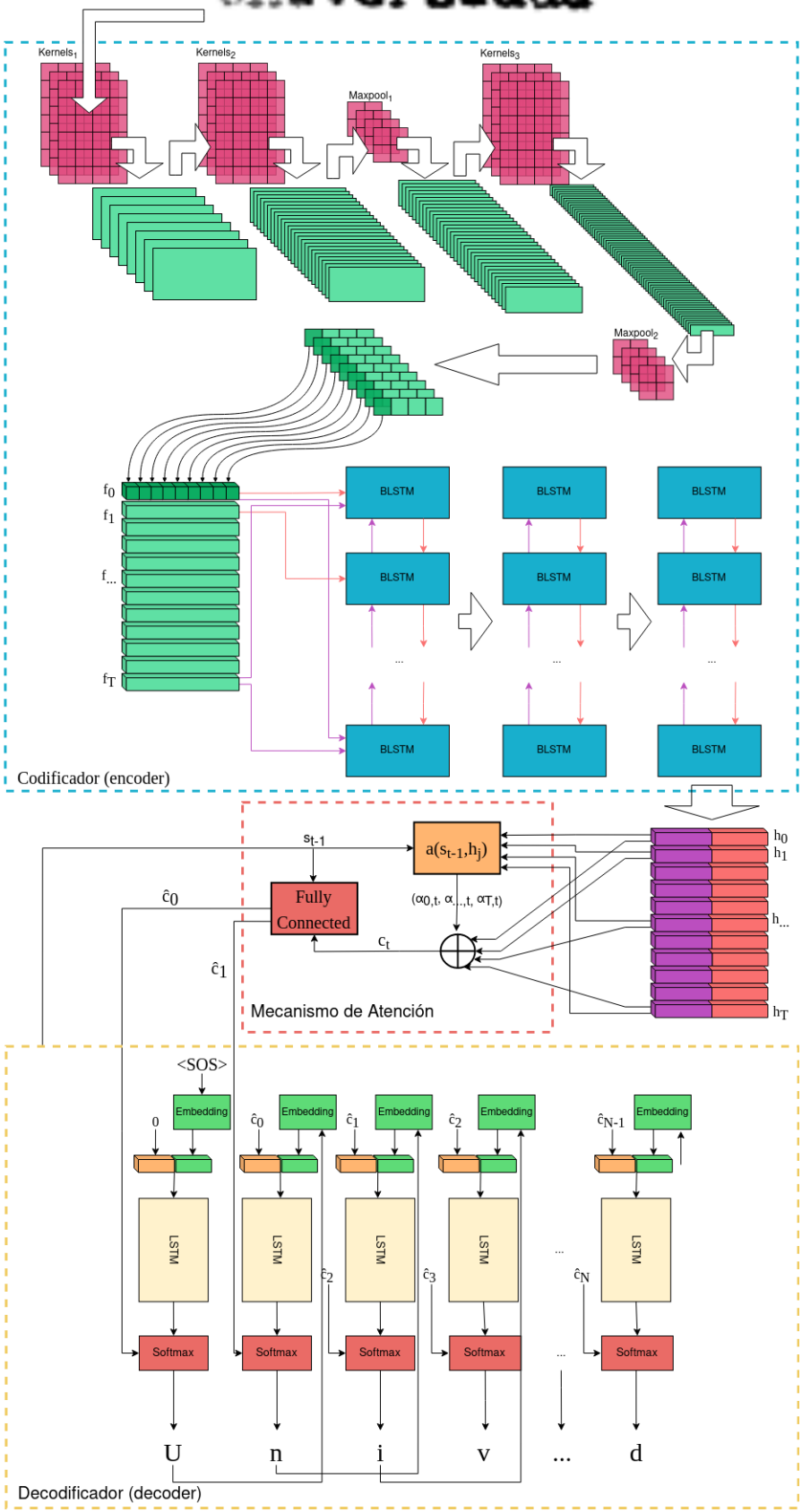


Figura 4.5: Arquitectura del modelo propuesto. La imagen que se va a transcribir es primero procesada por la red convolucional, compuesta por tres capas convolucionales y dos capas de *pooling*. Los vectores $f_0, ..., f_T$ obtenidos, llamados vectores de características, son luego procesados por la red recurrente bidireccional con unidades *LSTM*. Se guarda el estado de la red recurrente al procesar cada uno de los vectores de características y se obtienen los vectores de estado $h_0, ..., h_T$. Estos vectores de estado son los que van a ser procesados por el mecanismo de atención, que genera el vector de contexto c_t en cada salto de tiempo t usado para que el decodificador prediga el siguiente símbolo de la secuencia final.

Capítulo 5

Análisis experimental

A partir del modelo definido en el capítulo anterior, se propone una serie de escenarios de prueba centrados principalmente en los conjuntos de datos construidos, en dos niveles de transcripción: a nivel de bloques y de líneas. El objetivo de experimentar con la arquitectura elegida es validar su funcionamiento y comparar los resultados con los modelos de *OCR* ya usados por el equipo de *LUISA*, uno entrenado con la herramienta *Tesseract* (Kay, 2007), y otro entrenado con la herramienta *Calamari-OCR* (Wick et al., 2020).

En primera instancia se describen las métricas utilizadas para realizar las evaluaciones. Luego se describe la función de costo empleada y la especificación de la etapa de entrenamiento. Finalmente se presentan los distintos escenarios de prueba, y una muestra de los resultados obtenidos junto con algunas conclusiones.

5.1 Métricas

Las dos métricas principales usadas en el proyecto son *Character Error Rate*, que en español significa tasa de error a nivel de caracteres; y *Longest Common Subsequence Ratio*, que significa tasa de subsecuencia común más larga.

La primera, abreviada CER, tiene por objetivo reflejar el porcentaje de caracteres predichos de forma incorrecta. La medida es calculada como se muestra en la Fórmula 5.1, donde I, S y E hacen referencia a Inserciones, Sustituciones y Eliminaciones, significando la cantidad de operaciones que en conjunto hay que realizar para que dos secuencias de texto sean iguales, y N a la cantidad de caracteres. Para obtener $I + S + E$ se utiliza la distancia *Levenshtein* entre la secuencia de símbolos predicha y la secuencia objetivo. Luego se normaliza la distancia *Levenshtein* para obtener el *CER*.

$$CER = \frac{(I + S + E)}{N} * 100 \tag{5.1}$$

La medida *LCS Ratio* es usada por el equipo de *LUISA* Etcheverry et al. (2021). La métrica mide la similitud entre dos secuencias de texto con un valor comprendido entre 0 y 1: 0 si los textos son completamente disimilares y 1 si son idénticos. Para calcular este cociente, dadas dos secuencias de texto, como ejemplo $R =$ «la casa de su madre» y $T =$ «la caso ce eu nadre», primero se halla recursivamente la subsecuencia idéntica de mayor longitud entre ambos textos, y luego entre las dos mitades resultantes de extraer de cada uno esta subsecuencia idéntica. En el ejemplo, el primer *LCS* entre R y T es $C_1 =$ «la cas». Luego, de extraer esta secuencia de R se obtiene por izquierda el texto vacío y por derecha el texto «a de su madre». Para T, similarmente, se obtiene el texto vacío y «o ce eu nadre». Repitiendo este paso se obtiene $C_2 =$ «*adre*» y dos nuevos textos para cada secuencia, el texto de la izquierda y derecha de la secuencia idéntica. Este ciclo se repite hasta que no es posible obtener otra *LCS*. Finalmente, el *LCS Ratio* queda determinado como se muestra en la Fórmula 5.2, donde $|C_i|$ es el largo de la i-ésima subsecuencia común más larga, y $|R|$ $|T|$ el largo de las secuencias iniciales R y T.

$$LCSRatio = \frac{2 * \sum_i |C_i|}{|R| + |T|} \tag{5.2}$$

Para el ejemplo propuesto, y como puede visualizarse en la Figura 5.1, el *LCS Ratio* es $2 * (6 + 4 + 2 + 1 + 2)/19 = 15/19 = 0.789$. En esta medida se usa el *LCS* de elementos contiguos, las secuencias idénticas que se buscan son de caracteres que suceden en orden y de forma contigua.

A la hora de evaluar sobre el conjunto de validación, también se usa lo que queda nombrado como *Soft CER*, que es un *CER case-insensitive*, ya que muchas transcripciones realizadas en *LUISA* no respetan las mayúsculas y las minúsculas que se encuentran en el texto de las imágenes. Esta medida no es usada para tomar alguna decisión, pero sí sirve para reflejar la diferencia existente por no tener en diferencias entre mayúsculas y minúsculas.

Se incluye también como medida de las experimentaciones lo que se nombró como *Block Error Rate (BER)*, la proporción de bloques donde se cometió al menos un error. De esta forma, refleja una noción de precisión sobre la totalidad del texto del bloque.

5.2 Función de costo

La función de costo o función de *loss* (pérdida) es usada para calcular el valor de costo de la decisión tomada en cada predicción de símbolo objetivo. Esta función se utiliza para ajustar los pesos de las redes que conforman el modelo. La función elegida, y empleada por la mayoría de los artículos estudiados, es la función de *Negative Log Likelihood*. Esta recibe la probabilidad que el modelo asignó al carácter que debería de haber sido el elegido y devuelve el valor negativo del logaritmo de ese valor. Como se puede observar en la Figura 5.2, el valor de la función $-log$ tiende a 0 a medida que el valor del argumento tiende a 1 y tiende a infinito a medida que el

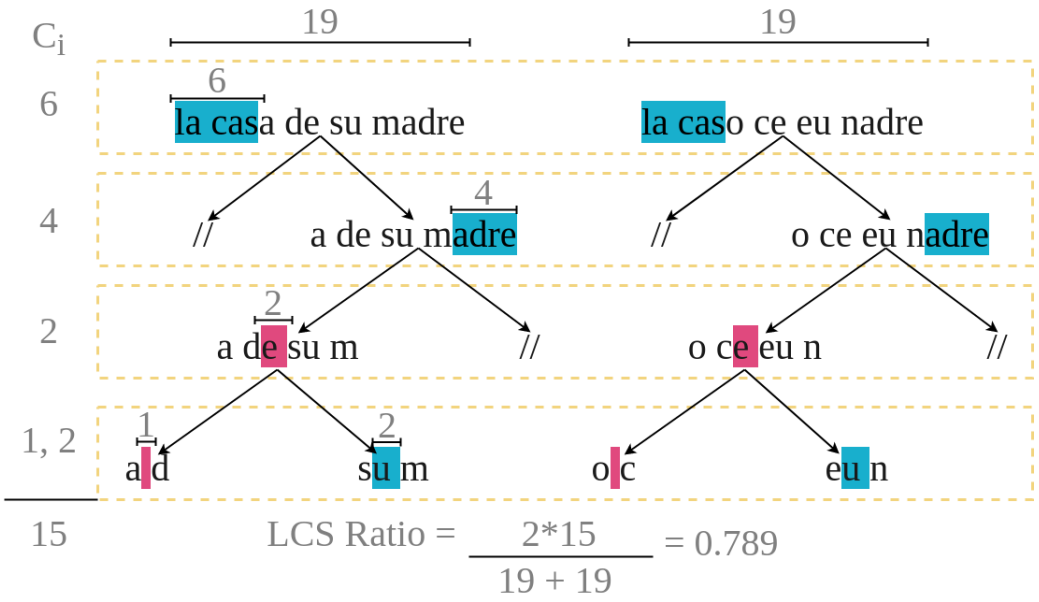


Figura 5.1: Ejemplo del cálculo de la métrica *LCS Ratio* sobre las secuencias de texto R = «la casa de su madre» y T = «la caso ce eu nadre». En una primera iteración se haya la subsecuencia común más larga C_1 = «la cas» de largo 6 y se generan las subsecuencias por el lado derecho «a de su madre» y «o ce eu nadre». Se itera nuevamente sobre estos subconjuntos y se halla la subsecuencia común más larga C_2 = «adre», lo cual deja por izquierda las subsecuencias «a de su m» y «o ce eu n». Repitiendo el proceso se hallan las subsecuencias comúnes C_3 = «e », C_4 = « » y $_5$ = «C2». Luego, no es posible encontrar nuevas subsecuencias comúnes, por lo que se calcula la fórmula y se obtiene un $LCSRatio = 0.789$.

valor del argumento tiende a 0. Por lo tanto, esta función de costo penalizará al modelo de forma inversamente proporcional a la probabilidad asignada a la clase correcta.

5.3 Especificaciones del entrenamiento

El método usado para actualizar los pesos correspondientes a las redes que conforman al modelo es el de *Stochastic Gradient Descent* (*SGD*) o Descenso de Gradiente Estocástico en español, extendido con el algoritmo de optimización *Adam* (Kingma et al., 2015). Este algoritmo de optimización es usado ampliamente en la literatura estudiada para *OCR* y permite al modelo obtener resultados considerados cercanos al óptimo en menor tiempo en comparación con el algoritmo clásico de *SGD*. El *learning rate* (*LR*) o *tasa de aprendizaje* es un parámetro que permite definir la magnitud con la que los valores son actualizados. Podría entenderse como la magnitud del «salto» que se quiere realizar en cada instancia de actualización de los pesos de la red.

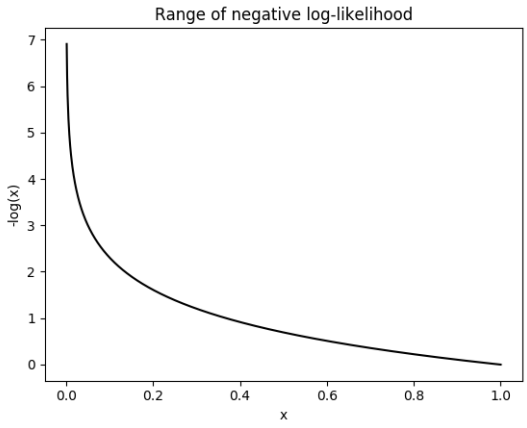


Figura 5.2: Gráfica correspondiente a la función $-log(x)$

Es necesario definir un valor de tasa de aprendizaje tal que el modelo aprenda a una velocidad razonable. Si es demasiado pequeño no avanzará tan rápido como sería posible, y que llegue a un valor considerado óptimo, si es muy alto el modelo tiende a actualizar los pesos demasiado rápido y perder mínimos.

Se hace uso también del método llamado *teacher forcing* (*TF*, Williams et al. (1989)), que consiste en sustituir el símbolo predicho en el salto de tiempo anterior por el símbolo que le corresponde en la secuencia objetivo, para que el decodificador, en el salto de tiempo actual, haga uso del símbolo correcto al predecir el siguiente, basándose en información correcta. Este mecanismo se activa bajo cierta probabilidad definida.

Debido al uso de este método, que depende de una probabilidad para ser usado en cada salto de tiempo, y por otros detalles que tienen que ver con las bibliotecas usadas en la implementación y la configuración de *hardware* al entrenar, el algoritmo de entrenamiento no es determinista, implicando así que dos instancias bajo la misma configuración de parámetros den por resultado dos modelos que obtienen distinto rendimiento. Es por esto que se entrenan tres instancias de modelos en cada experimento, con el objetivo de hacer una comparación más robusta. Estos son luego evaluados sobre el conjunto de validación, y el mejor es el que finalmente se evalúa sobre el conjunto de *test*, para ser comparado con los de otros escenarios de prueba.

El modelo es entrenando haciendo uso de *minibatches*, que consiste en entrenar con varios elementos a la vez, realizando el proceso de predicción para cada uno de ellos antes de actualizar los pesos de las redes, haciendo uso de todos los resultados de ese *minibatch*. En este caso, se define el tamaño del *minibatch* en 8. Por lo que se realizará la transcripción de a 8 imágenes antes de actualizar los pesos.

También se utiliza el concepto de *epoch* en la etapa de entrenamiento. Una *epoch* significa

entrenar el modelo sobre todo el conjunto de entrenamiento. Una nueva *epoch*, una nueva iteración sobre todo el conjunto. Es importante tener cuidado con la cantidad de *epochs* de entrenamiento seleccionada. Una cantidad exagerada de pasadas por todos los elementos hará que el modelo actualice sus pesos de forma que tendrá buen rendimiento con estos, pero no será capaz de generalizar, obteniendo mal rendimiento con elementos antes no vistos, fenómeno llamado *sobreajuste*.

Es posible reconocer sobreajuste si se compara, al final de cada epoch, el rendimiento sobre el conjunto de entrenamiento y el rendimiento sobre el conjunto de validación, compuesto de elementos no vistos a la hora de entrenar. Se identifica que el modelo está sobreajustando cuando los valores de la función de pérdida en ambos conjuntos empiezan a alejarse, como se muestra en la Figura 5.3. De esta forma sería posible detener el entrenamiento cuando se reconoce que empieza a sobreajustar.

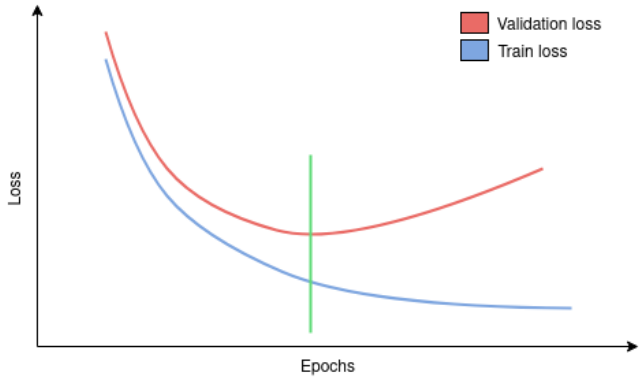
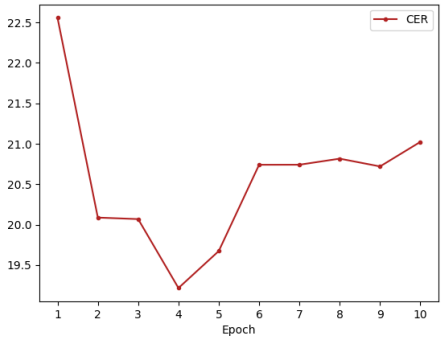


Figura 5.3: Ejemplo de gráfica de función de pérdida sobre los conjuntos de entrenamiento y validación a medida que se entrena el modelo por varias *epochs*. Puede observarse que el modelo empieza a sobreajustar cuando las líneas empiezan a alejarse. Esto significa que el modelo está aprendiendo a obtener buen rendimiento con el conjunto de entrenamiento pero que pierde capacidad de rendir bien sobre elementos antes no vistos, que son los que componen el conjunto de validación.

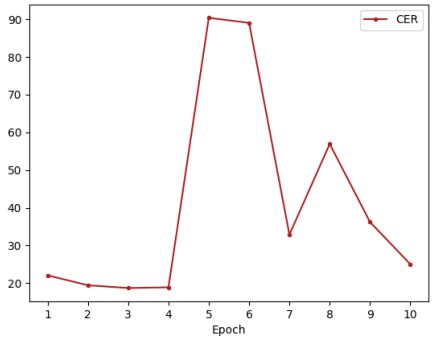
En pruebas preliminares pudo verse que en el caso de entrenamiento sobre bloques, 10 *epochs* eran suficientes para reconocer que el modelo empezaba a sobreajustar, y las métricas sobre el conjunto de validación a estancarse. En el caso de los escenarios sobre líneas propias, se pudo ver que con 20 *epochs* sucedía lo mismo; y en el caso de líneas de *Calamari-Half8*, 100 epochs. La diferencia de magnitud en cantidad de *epochs* se debe a que los conjuntos de bloques son mucho más grandes que los conjuntos de líneas, evidentemente porque las líneas están conformadas por múltiples bloques.

Se observa también que el mejor modelo no necesariamente se obtiene al finalizar el entrenamiento, en la última *epoch*, como se muestra en la Figura 5.4, por lo que no se asume que este sea

el mejor. Además, es posible que el modelo logre el mejor resultado luego de haber tenido varias *epochs* consecutivas de bajo rendimiento, por lo que se decide no implementar una heurística de *early stopping*, que consiste en detener el entrenamiento antes de finalizar, al observar una reducción del rendimiento. En cambio, se guarda la instancia del modelo, llamada *checkpoint*, al finalizar cada una de las *epochs*, para finalmente seleccionar aquel donde se obtuvo el mejor valor de *CER* sobre validación. Un *checkpoint* incluye los pesos de todas las redes que conforman el modelo, la configuración y los valores correspondientes al entrenamiento en ese instante, permitiendo en un futuro deseado volver a cargar la instancia del modelo y continuar, o utilizar esta instancia para procesar datos nuevos.



(a) *Teach Forcing = 0.5, Tasa de aprendizaje = 0.0005*



(b) *Teach Forcing = 0.8, Tasa de aprendizaje = 0.0005*

Figura 5.4: Evolución de *CER* sobre el subconjunto de validación a través de las *epochs* durante la etapa de entrenamiento. Corresponden a una de las instancias de entrenamiento con parámetro de *Teacher forcing* = 0.5, 0.8 y tasa de aprendizaje = 0.0005. Puede observarse que el mejor valor de *CER* no se alcanza en la última *epoch*, sino antes.

5.4 Escenarios de prueba

Los entrenamientos se realizan sobre los conjuntos de datos obtenidos, que fueron mencionados en el Capítulo 3. Las experimentaciones se enmarcan en escenarios de prueba, en función del conjunto utilizado. A continuación se presentan los distintos escenarios de prueba.

5.4.1 Escenario 1: entrenamiento sobre conjunto de bloques

Cada bloque que pasa por la plataforma de *LUISA* tiene más de una transcripción asociada. En muchos casos las transcripciones realizadas para un mismo bloque son distintas, y dado que

evidentemente el texto del bloque es único, alguna de esas transcripciones será la más fiel. En este sentido parece razonable querer seleccionar para cada bloque esa transcripción, como se hizo al construir el primer conjunto de bloques, buscando reducir el ruido introducido por transcripciones que no son fieles. El primer escenario de pruebas consta de entrenar los modelos sobre el conjunto de bloques con transcripción seleccionada.

En este primer escenario se realiza *hiperparametrización* sobre los parámetros *learning search* y *teacher forcing* antes nombrados, que consta de ir variando los valores para luego hacer uso en futuros entrenamientos de aquellos con los que se obtuvo mejores resultados. Para cada una de las configuraciones (variaciones) posibles se realizan tres instancias de entrenamiento. El mejor de los tres es el usado a la hora de hacer la comparación.
Los valores experimentados son:

- Tasa de aprendizaje: 0.001, 0.0005, 0.0001, 0.00005
- Probabilidad de *teacher forcing*: 0.5, 0.8, 1

Aplicando estas configuraciones combinadas, se obtienen 4x3 = 12 configuraciones posibles, por lo que se ejecutan 36 instancias de entrenamiento en total.

Luego de obtenidos los resultados para el escenario de prueba se encuentra que los mejores valores para LR y TF son 0.0005 y 0.8 respectivamente. Estos valores son utilizados en los siguientes escenarios de prueba.

5.4.2 Escenario 2: uso de las múltiples transcripciones por bloque

En el escenario anterior se selecciona una transcripción por cada bloque, considerando que esta seleccionada es la más fiel al texto contenido en la imagen, con el objetivo de reducir la cantidad de ruido en el conjunto usado para entrenar el modelo. Sin embargo, ¿es esto lo que garantiza un mejor aprendizaje del modelo? O más aún, ¿se garantiza realmente que las transcripciones elegidas son las más fieles? Otro enfoque es el de generar una entrada al conjunto por cada combinación *bloque-transcripción* válida posible: aquella que no es vacía ni contiene arrobas (*'@'*).
Es interesante notar que de esta forma quedaría delegado al modelo neuronal la tarea de lidiar con los efectos del ruido existente en las transcripciones realizadas. Entiéndase por ruido a las transcripciones diferentes al texto real contenido en el bloque.

Este escenario de prueba hace uso del segundo conjunto de bloques construido, y de los parámetros de entrenamiento que mejor resultado obtuvieron en el primer escenario de prueba.

5.4.3 Escenario 3: entrenamiento con conjunto de líneas construido a partir del ensamblado de bloques de *LUISA*

Existe a la hora de realizar las pruebas un conjunto de líneas utilizado por el equipo *LUISA* para entrenar el modelo *OCR* de *Calamari-OCR*. A este conjunto se le ha nombrado en este proyecto como *Calamari-Half8* dado que las imágenes fueron reducidas a la mitad y traducidas a escala de grises (*8 bits per pixel*). Si bien la disponibilidad de este conjunto de líneas curadas manualmente representa un *ground truth* con un nivel de fidelidad quizás inmejorable, no fue posible realizar un rastreo por línea de cuáles son los bloques que las conforman. Además, este conjunto de líneas disponible es un subconjunto del total de líneas disponibles en la plataforma *LUISA*, ya que pasan por un proceso de curado adicional, y al momento de obtenerlo se habían procesado menos de la mitad.

Interesa poder hacer uso de todos los datos que se encuentran disponibles en *LUISA*, además de tener la posibilidad de obtener los bloques que conforman cada línea (capacidad de descomponer las líneas en los bloques usados para generarlas). Es por esto que se construyó el conjunto de líneas propio a partir del ensamblado de bloques, permitiendo realizar distintas pruebas que se mencionan a continuación:

- **Modelo base líneas**: entrenamiento del modelo a nivel de líneas con el conjunto de líneas propias
- **Modelo combinado**: entrenar el modelo tanto con líneas como con los bloques que componen esas líneas conformando un mismo conjunto, al que llamaremos conjunto combinado. La intención de esta prueba es la de comprobar si el modelo se ve beneficiado del uso de elementos con texto de menor largo en el entrenamiento a nivel de líneas.
- **Modelo en dos etapas**: entrenar primero a nivel de bloques y luego a nivel de líneas. Entrenando primero con bloques se está limitando el problema a aprender las características visuales de la imagen de entrada y las posibles relaciones entre los caracteres de la secuencia de texto del bloque, que es en promedio una palabra, pero no se aprenderían relaciones entre palabras. Luego de entrenado con bloques, se entrena a nivel de líneas, intentando que aprenda las relaciones entre palabras.
- **Modelo de descomposición**: entrenar el modelo solo a nivel de bloques, haciendo uso de los bloques resultantes de descomponer el conjunto de líneas. Luego, evaluar el modelo sobre el conjunto de líneas mediante el siguiente proceso: descomponer las líneas de evaluación en bloques, transcribir esos bloques. Finalmente, unir las transcripciones de los bloques para componer el texto de la línea de evaluación y evaluar usando el texto esperado para la línea. Esta prueba permite, mediante la comparación del rendimiento con otros modelos entrenados sobre líneas, generar una noción del potencial del modelo para aprender relaciones entre

secuencias más largas. Dicho de otra forma, ¿mejora el rendimiento del modelo si es entrenado a nivel de líneas, o puede un modelo entrenado a nivel de bloques tener el mismo rendimiento al transcribir líneas enteras?

- **Modelo de descomposición aumentado**: entrenamiento análogo al caso anterior, pero generando a partir del conjunto de bloques, un conjunto aumentado análogo al usado en el escenario 5.4.2, y entrenando con este.

5.4.4 Escenario 4: entrenamiento con conjunto de líneas provisto por el equipo *LUISA* y comparación con Calamari-OCR

Inicialmente el equipo de *LUISA* hizo uso del OCR *Tesseract* (Kay, 2007) para transcribir algunos documentos automáticamente sin la necesidad de que pasaran por la plataforma de *LUISA*, y también para generar una primera instancia de transcripción de todos los documentos disponibles. La calidad de las transcripciones realizadas por este sistema no fueron buenas en general.

Tesseract es un sistema de *OCR open-source* que comenzó su desarrollo en 1995. El sistema incluye herramientas de pre procesamiento de la imagen a transcribir junto con un módulo de transcripción automática. La versión original del módulo OCR de Tesseract incluye heurísticas para la detección de patrones de caracteres con el objetivo de reconocer cuales son los caracteres de la imagen. Versiones más actuales implementan un módulo de OCR basado en redes *LSTM*[1].

Casi en paralelo con el Proyecto de Grado actual, el equipo comenzó a probar una nueva herramienta de OCR llamada *Calamari-OCR*, con resultados más prometedores.

La herramienta está construida haciendo uso de redes *CNN* y *LSTM*, la cual se entrena haciendo uso de la función de costo *CTC*, vista en el análisis del estado del arte como uno de los métodos con mayor rendimiento alcanzado en la tarea de *OCR*. Calamari-OCR tiene la ventaja de haber sido utilizada sobre la biblioteca de *Python TensorFlow* (Martín Abadi et al., 2015), de aprendizaje automático, optimizada para reducir al mínimo el tiempo necesario para entrenar los modelos neuronales.

Interesa evaluar qué tan bien rinde el modelo propuesto en comparación con este. Para esto se entrena el modelo haciendo uso del conjunto de datos *Calamari-Half8*, y luego se compara la evaluación con los resultados ya obtenidos por la herramienta.

La configuración de parámetros usada es aquella que logró los mejores resultados a nivel de bloques. Se entrena por 100 *epochs*, un orden de magnitud mayor al entrenamiento con bloques, dado que la cantidad de elementos disponibles es mucho menor. El *batch size* se mantiene igual, de tamaño 8 elementos. Se realizan 3 instancias de entrenamiento y se mantiene el modelo para la *epoch* que obtiene la mejor medida *CER* sobre el conjunto de validación.

[1] Más información puede encontrarse en las *release notes* de Tesseract. https://tesseract-ocr.github.io/tessdoc/tesseract-with-lstm

En la Figura 5.5 puede observarse la evolución de la medida *CER* sobre el conjunto de validación según la *epoch*. Se observa, como se esperaba, que le lleva más *epochs* estabilizarse, (cerca de 50 *epochs* para comenzar a estabilizarse).

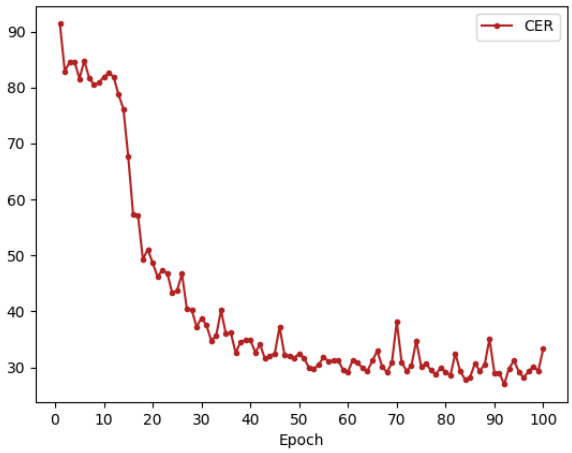


Figura 5.5: Evolución del *CER* sobre validación en función del *epoch* en etapa de entrenamiento.

5.5 Implementación y ejecución

Los modelos entrenados, junto con todos los *scripts* necesarios para la realización de los experimentos, fueron implementados haciendo uso del lenguaje *Python*. Los modelos, junto con los procedimientos de entrenamiento y predicción se implementaron haciendo uso de la librería *PyTorch* (Paszke et al., 2019), un *framework open source* en el lenguaje *Python* para la realización de modelos de aprendizaje automático. Esta biblioteca trae ya implementadas las redes neuronales más usadas, distintas funciones de activación, funciones de costo, optimizadores, entre otras utilidades que permiten generar no solo modelos con distintas arquitecturas que combinen distintas redes, sino también procedimientos de entrenamiento y evaluación.

No fue necesario implementar un modelo distinto para el entrenamiento a distintos niveles, ya que la entrada siempre es una sola imagen, variando entonces, entre bloques y líneas, solo la cantidad máxima de símbolos que se espera a la hora de predecir el texto de una imagen. Para el caso de los bloques se tomó como largo máximo de transcripción 20 caracteres, ya que este es el valor que abarca estadísticamente el 97.6% de los largos observados en el conjunto de entrenamiento a nivel de bloques. En el caso de las líneas ensambladas este valor es 72 y en el caso de las líneas *Calamari-Half8* este valor es 105.

En cuanto a la ejecución del entrenamiento, gracias a la disponibilidad de *ClusterUY* (Nesmachnow et al., 2019) y la posibilidad de usarlo que se le da a estudiantes, fue posible entrenar los modelos haciendo uso de los siguientes recursos:

- 32 GB de memoria RAM
- 10 GB de memoria SSD
- 1x12 GB de memoria gráfica (NVIDIA Tesla P100)

ClusterUY habilita un usuario para acceder, mediante el servicio *ssh*, a los nodos que componen el clúster. En este es posible subir el código *Python* implementado y luego hacer uso de alguno de los nodos para ejecutar el código.

5.6 Resultados

Esta sección está formada por dos subsecciones de resultados, relacionada la primera con los obtenidos a nivel de bloques y la segunda a nivel de línea. Dado que los largos de las secuencias promedio son distintas a nivel de bloque y a nivel de línea, las métricas usadas, que están vinculadas al largo, son distintas, por lo que hacer una comparación directa no tiene sentido. Sin embargo, es posible mediante los experimentos de transcripción de líneas con modelos entrenados con imágenes de bloques comparar el potencial de los modelos de las dos etapas para una misma tarea.

Los resultados observados en las tablas de este capítulo corresponden al mejor modelo obtenido de las tres instancias de entrenamiento realizadas por cada prueba. Se comparan los modelos entrenados de cada prueba sobre el conjunto de validación y se toma aquel que obtenga mejor valor de *CER*. Luego cada modelo elegido es evaluado sobre el conjunto de *test*, calculándose las métricas *CER*, *Soft CER* y *LCS*.
Las tablas de evaluación sobre el conjunto de *validación* de la mejor *epoch* para cada modelo entrenado se agregan al anexo.

5.6.1 A nivel de bloques

En las Tablas 5.1 y 5.2 se muestran los resultados obtenidos en la evaluación sobre el conjunto de *test*. El llamado «Modelo base» es el primer modelo entrenado, haciendo uso del primer conjunto de entrenamiento construido: bloques con transcripción seleccionada. El modelo «Modelo aumentado» es el modelo obtenido a partir de entrenar haciendo uso del conjunto de bloques aumentado.
En la sección del Anexo 7.1 y 7.2 se pueden observar los mejores resultados obtenidos sobre el conjunto de validación en las distintas instancias de entrenamiento realizadas.

A nivel de bloques, el modelo obtuvo mejores resultados al ser entrenado sobre el conjunto de bloques aumentado, lo que indica una mejora en rendimiento cuando se hace uso de todas las

Modelo	CER	Soft CER	BER	LCS
Modelo *base*	18.65	18.14	34.90	00.83
Modelo *aumentado*	**17.57**	**17.00**	**31.14**	**00.84**

Cuadro 5.1: Resultados de la evaluación de los modelos seleccionados (el mejor entrenado sobre el conjunto base y el mejor entrenado sobre el conjunto aumentado) sobre el conjunto de *test* no aumentado.

Entrenado sobre	CER	Soft CER	BER	LCS
Modelo *base*	17.82	16.64	38.58	00.84
Modelo *aumentado*	**16.63**	**15.38**	**34.96**	**00.85**

Cuadro 5.2: Resultados de la evaluación de los modelos seleccionados sobre el conjunto de *test* aumentado.

transcripciones disponibles sin necesidad de hacer un intento por seleccionar la mejor transcripción posible.

En términos de tiempo, como es de esperarse, dada la diferencia en el tamaño de los conjuntos, entrenar el modelo 10 *epochs* sobre el primer conjunto de bloques, con transcripción seleccionada, lleva en promedio entre 20 y 24 horas, la mitad del tiempo que lleva entrenarlo sobre el conjunto aumentado, entre 44 y 48 horas.

5.6.2 A nivel de líneas

Los resultados de los modelos obtenidos en los distintos escenarios de prueba, evaluados sobre el conjunto de *test*, se pueden ver en la Tabla 5.3.

En la sección del Anexo 7.3 se observan los mejores resultados obtenidos en las distintas instancias de entrenamiento sobre el conjunto de validación.

En el caso de las pruebas realizadas sobre el conjunto de líneas construido en el Proyecto, se observa que el modelo que obtiene el mejor valor de *Character Error Rate* es aquel entrenado haciendo uso tanto de las líneas ensambladas como de los bloques que las conforman. Sin embargo, la diferencia es casi despreciable en comparación con el modelo que se entrena en base a bloques con todas sus transcripciones y luego transcribe líneas descomponiéndolas (*modelo de desc. + aumentado*). De hecho, este último supera a los demás en la métrica *Soft CER* e iguala en *LCS*. La nula diferencia de rendimiento sugiere que el modelo no se beneficia de una entrada más larga.

Prueba	CER	Soft CER	LCS
Modelo base líneas	28.22	27.74	00.74
Modelo combinado	**26.96**	26.47	**00.75**
Modelo de dos etapas	27.59	27.20	00.74
Modelo de descomposición	27.32	26.83	**00.75**
Modelo de desc. + aumentado	27.02	**26.30**	**00.75**

Cuadro 5.3: Tabla de resultados sobre el conjunto de *test* agregando los resultados de la prueba de descomposición de líneas.

Sobre el escenario de prueba que hace uso del conjunto de líneas curadas, llamado *Calamari-Half8*), se muestran los resultados obtenidos en la Tabla 5.4.

Modelo	CER	Soft CER	LCS
Seq2Seq	28.10%	27.50%	00.77
Calamari-OCR	**23.74%**	**23.23%**	**00.81**

Cuadro 5.4: Resultados de rendimiento de los modelos *Seq2Seq* y *Calamari-OCR* sobre conjunto de *test*.

En cuanto a este experimento, se observa que el potencial de aprendizaje de *Calamari-OCR* ha sido mejor que el del modelo implementado en el proyecto. Sin embargo, la diferencia no es tan grande si tenemos en cuenta que el *OCR* antes utilizado, *Tesseract*, tenía un *CER* en el orden del 52%. En ese sentido, el resultado alienta a seguir avanzando en la prueba de distintas características que puedan mejorar el rendimiento.

Resumen

Se definieron los detalles relacionados con la experimentación del modelo propuesto, los escenarios de prueba ejecutados, y los resultados obtenidos en cada uno.

El modelo alcanza un mejor rendimiento cuando se incluyen todas las transcripciones por bloque. Además, los resultados muestran que el modelo no se está beneficiando de recibir por entrada una secuencia de largo mayor, dada la nula diferencia de rendimiento entre el modelo entrenado a partir de líneas y aquel entrenado solo con bloques y luego usado para transcribir líneas.

En el siguiente capítulo se expresan las distintas conclusiones que se obtuvieron, junto con las posibles líneas de trabajo a futuro que se consideran de valor.

Capítulo 6

Conclusiones

El objetivo principal del presente proyecto fue el de estudiar y evaluar la arquitectura *Seq2Seq* para la transcripción de documentos históricos pertenecientes al *Archivo Berrutti*.

Se construyeron dos conjuntos a nivel de bloques: uno en el que se selecciona para cada bloque la transcripción considerada más fiel al texto contenido en la imagen, y otro donde para cada bloque se selecciona toda transcripción válida disponible. Se construye otro conjunto a nivel de líneas, mediante el ensamblado de los bloques disponibles. Estas líneas pueden descomponerse nuevamente a los bloques que las conforman para permitir distintos escenarios de prueba y no únicamente el entrenamiento solo con líneas. Los distintos conjuntos fueron construidos mediante la utilización de las transcripciones de muchos voluntarios y voluntarias en la plataforma colaborativa *LUISA*, enfrentando en el proceso diversas complejidades técnicas. Además, se obtuvo un conjunto de líneas previamente curado, que es usado para comparar la arquitectura propuesta con la herramienta de *OCR* utilizada actualmente en *LUISA*. Los escenarios de prueba involucraron el entrenamiento del modelo propuesto haciendo uso de estos conjuntos.

Del entrenamiento a nivel de bloques, el modelo entrenado sobre el conjunto aumentado logró un mejor rendimiento con ambos conjuntos de *test* de bloques, superando el *CER* en 1.08% en la evaluación sobre *test* no aumentado y 1.19% sobre *test* aumentado. Llama la atención también la mejora en la tasa de bloques con errores (*BER*) en el modelo aumentado, con una diferencia cercana al 4% en ambos conjuntos. Esto deja en evidencia que el modelo neuronal no solo es capaz de absorber el ruido generado por transcripciones diferentes al texto de la imagen, introducido al elegir todas las transcripciones disponibles, sino que es capaz de aprender.

A nivel de líneas, el modelo que mejor valor de *CER* obtiene es aquel entrenado sobre el conjunto que combina líneas y los bloques que las conforman. Sin embargo, el modelo entrenado solo sobre el conjunto aumentado de los bloques obtiene un valor de *CER* muy cercano, con una diferencia de 0.06%. De hecho, en la métrica *Soft CER* este último logra el mejor valor. Ambos modelos han

sido entrenados sobre conjuntos aumentados de alguna manera: el primero son líneas y bloques y el segundo bloques con todas sus transcripciones. Entonces, la arquitectura parece verse beneficiada de los conjuntos de datos más grandes. Más investigación en este sentido podría ser útil para lograr generar conjuntos extendidos a partir de la información que ya está disponible, sin necesidad de generar más datos.

Además, la casi nula diferencia de rendimiento entre los modelos entrenados para transcribir imágenes a nivel de línea y los modelos entrenados para transcribir los bloques que componen la línea por separado (*Descomposición* y *Descomposición + aumentado*) da a entender que la arquitectura *Seq2Seq encoder-decoder* implementada podría no estar generando las transcripciones teniendo en cuenta las relaciones entre palabras, o más específicamente, no estaría encontrando beneficio en entradas cuya secuencia de caracteres es más larga. Dado que los *receptive fields* correspondientes a los vectores de característica son columnas de la imagen de entrada cuyo ancho está más cercano a ser del de un carácter más que el de una palabra, se podría conjeturar que la arquitectura esté aprendiendo a identificar relaciones entre elementos con distancia tal que pueda reconocer relaciones entre caracteres cercanos, pero se vea limitada para aprender relaciones entre conjuntos de elementos tan distantes como para generar predicciones en función de palabras anteriores, hipótesis respaldada por el hecho de que existan enfoques que fusionan la arquitectura con un modelo de lenguaje (Sriram et al., 2018) o que intentan corregir la salida post-procesándola (Kang et al., 2022).

En el caso de la comparación con *Calamari-OCR*, dado que el conjunto de entrenamiento usado para entrenar ambos modelos es el mismo, la diferencia de rendimiento está directamente relacionada a la diferencia en el potencial de los modelos para aprender a transcribir las líneas. Como se observa en la tabla de resultados, *Calamari-OCR* supera al modelo *Seq2Seq encoder-decoder* implementado, con un puntaje de *CER* de 28.10% y 23.74% respectivamente.

6.1 Trabajo a futuro

La arquitectura usada por *Calamari-OCR*, compuesta por una red *CNN*, una *LSTM* y la función de costo *CTC*, sigue siendo actual y compite con otras como la implementada en este proyecto, o la reciente arquitectura de *Transformers*, por lo que esta cercanía de resultados no es del todo inesperada. Sin embargo, el hecho de que *Calamari-OCR* sea una herramienta mucho más elaborada que el modelo propuesto alienta a destinar tiempo en refinarlo, buscando alcanzar o superar los resultados obtenidos.

Si bien parece evidente que la diferencia en rendimiento es la suficiente como elegir *Calamari-OCR* por sobre el modelo *Seq2Seq*, esta diferencia es lo suficientemente pequeña como para que el modelo *Seq2Seq* abra las puertas a posibilidades de sistemas que combinan múltiples módulos *OCR* para generar transcripciones más robustas. Por ejemplo, se podría hacer uso del porcentaje de confianza de la predicción del decodificador para generar un mecanismo de votación con otros modelos *OCR*. También es posible usar algoritmos de alineación de transcripciones de distintos

OCR, para luego utilizar alguna heurística como votación y/o basadas en léxicos para formar la secuencia nueva con los tokens considerados más aptos.

Resulta de mucho interés continuar la personalización del modelo implementado con el objetivo de abarcar lo mejor posible el dominio del problema planteado. Fue intención al comienzo del proyecto realizar pruebas con distintos mecanismos de atención para realizar la comparación de cuál funciona mejor en este caso.

También quedó abierta la pregunta de cuál será la mejora de fusionar el modelo implementado con un modelo de lenguaje que asista a la decisión de los siguientes elementos objetivos. De esta forma podría entrenarse un modelo de lenguaje sobre un conjunto grande y sumar el beneficio que tiene modelar el lenguaje sobre el modelo de transcripción.

Referencias

Bahdanau, D., Cho, K. & Bengio, Y. (2015). Neural Machine Translation by Jointly Learning to Align and Translate. In Y. Bengio & Y. LeCun (Eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA*.

Bengio, Y., Simard, P. & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157-166.

Bernardi, R., Cakici, R., Elliott, D., Erdem, A., Erdem, E., Ikizler-Cinbis, N., Keller, F., Muscat, A. & Plank, B. (2016). Automatic description generation from images: A survey of models, datasets, and evaluation measures. *Journal of Artificial Intelligence Research*, *55*, 409-442.

Bluche, T. (2016). Joint Line Segmentation and Transcription for End-to-End Handwritten Paragraph Recognition. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon & R. Garnett (Eds.), *Proceedings of the 30th International Conference on Neural Information Processing Systems, Barcelona, Spain* (pp. 838-846). Curran Associates Inc.

Bottou, L. & LeCun, Y. (2005). Graph Transformer Networks for Image Recognition [55th Session]. *Bulletin of the International Statistical Institute (ISI)*.

Brockwell, P. & Davis, R. (2006). *Introduction to Time Series and Forecasting*. Springer, New York.

Cadavid, H. (2009). *Spanish text corpus for NLP/linguistics research* (zenodo.4319956; Version 1) [Data set]. zenodo.4319956; Version 1. Zenodo.

Cho, K., van Merriënboer, B., Bahdanau, D. & Bengio, Y. (2014b). On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. In D. Wu, M. Carpuat, X. Carreras & E. M. Vecchi (Eds.), *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Doha, Qatar* (pp. 103-111). Association for Computational Linguistics.

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In A. Moschitti, B. Pang & W. Daelemans (Eds.), *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, Doha, Qatar* (pp. 1724-1734). Association for Computational Linguistics.

Chorowski, J., Bahdanau, D., Serdyuk, D., Cho, K. & Bengio, Y. (2015). Attention-Based Models for Speech Recognition. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama & R. Garnett (Eds.), *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1, Montreal, Canada* (pp. 577-585). MIT Press.

Chowdhury, A. & Vig, L. (2018). An Efficient End-to-End Neural Model for Handwritten Text Recognition. *British Machine Vision Conference 2018, BMVC 2018, Northumbria University, Newcastle, UK*, 202.

Elman, J. L. (1990). Finding Structure in Time. *Cognitive Science*, *14*(2), 179-211.

Etcheverry, L., Agorio, L., Bacigalupe, V., Barreiro, S., Bing, E., Blixen, S., Calegari, D., Cardozo, L., Carpani, F., Chavat, F., Garat, D., Gómez, A., Hernández, F., Marabotto, V., Moncecchi, G., Ramírez Paulino, I., Rosa, A., Tiscornia, J., Wonsever, D., … Laguna, R. (2021). A computational framework for the analysis of the Uruguayan dictatorship archives. In A. Paschke, G. Rehm, J. A. Qundus, C. Neudecker & L. Pintscher (Eds.), *Qurator 2021 - Conference on Digital Curation Technologies, Berlin, Germany*.

Graves, A., Fernández, S., Gomez, F. & Schmidhuber, J. (2006). Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In A. M. William Cohen (Ed.), *Proceedings of the 23rd international conference on Machine learning - ICML'06, Pittsburgh, Pennsylvania* (pp. 369-376). ACM Press.

Graves, A., Fernández, S., Liwicki, M., Bunke, H. & Schmidhuber, J. (2007). Unconstrained Online Handwriting Recognition with Recurrent Neural Networks. In J. Platt, D. Koller, Y. Singer & S. Roweis (Eds.), *Proceedings of the 20th International Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada* (pp. 577-584). Curran Associates Inc.

Graves, A., Fernández, S. & Schmidhuber, J. (2007). Multi-dimensional Recurrent Neural Networks. In J. M. de Sá, L. A. Alexandre, W. Duch & D. Mandic (Eds.), *Artificial Neural Networks – ICANN 2007, Berlin, Heidelberg* (pp. 549-558). Springer.

Guerra, A. & Fernández, F. (2000). *Machine Translation. Capabilities and limitations*. Publicacions de la Universitat de València, España.

Hochreiter, S., Bengio, Y., Frasconi, P. & Schmidhuber, J. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In S. C. Kremer & J. F. Kolen (Eds.), *A Field Guide to Dynamical Recurrent Neural Networks* (pp. 237-243). IEEE Press.

Hochreiter, S. & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735-1780.

Islam, N., Islam, Z. & Noor, N. (2017). A Survey on Optical Character Recognition System. *arXiv e-prints*.

Jelinek, F. (1998). *Statistical methods for speech recognition*. MIT Press, Cambridge.

Jordan, M. I. (1997). Serial Order: A Parallel Distributed Processing Approach. In J. W. Donahoe & V. Packard Dorsel (Eds.), *Neural-Network Models of Cognition* (pp. 471-495). North-Holland, USA.

Kang, L., Riba, P., Rusiñol, M., Fornés, A. & Villegas, M. (2022). Pay attention to what you read: Non-recurrent handwritten text-Line recognition. *Pattern Recognition*, *129*, 108766.

Kay, A. (2007). Tesseract: An Open-Source Optical Character Recognition Engine. *Linux J.*, *2007*(159), 2.

Kingma, D. P. & Ba, J. (2015). Adam: A Method for Stochastic Optimization. In Y. Bengio & Y. LeCun (Eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA*.

Koul, A., Ganju, S. & Kasam, M. (2019). *Practical Deep Learning for Cloud, Mobile, and Edge: Real-World AI & Computer-Vision Projects Using Python, Keras & TensorFlow*. O'Reilly Media.

Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84-90.

Lafferty, J. D., McCallum, A. & Pereira, F. C. N. (2001). Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In C. E. Brodley & A. P. Danyluk (Eds.), *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001), Williams College, Williamstown, MA, USA* (pp. 282-289). Morgan Kaufmann.

LeCun, Y., Bengio, Y. & Hinton, G. (2015). Deep Learning. *Nature*, *521*, 436-44.

Liu, B. (2020). *Sentiment Analysis: Mining Opinions, Sentiments, and Emotions*. Cambridge University Press.

Liu, Y. (2018). Feature Extraction and Image Recognition with Convolutional Neural Networks. *Journal of Physics: Conference Series*, *1087*, 062032.

Luong, T., Pham, H. & Manning, C. D. (2015). Effective Approaches to Attention-based Neural Machine Translation. In L. Màrquez, C. Callison-Burch & J. Su (Eds.), *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, Lisbon, Portugal* (pp. 1412-1421). Association for Computational Linguistics.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Jia, Y., Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, … Xiaoqiang Zheng. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems [Software available from tensorflow.org].

Michael, J., Labahn, R., Grüning, T. & Zöllner, J. (2019). Evaluating Sequence-to-Sequence Models for Handwritten Text Recognition [arXiv: 1903.07377]. *arXiv:1903.07377 [cs]*.

Mikolov, T., Karafiát, M. & Burget, L. (2010). Recurrent neural network based.

Nesmachnow, S. & Iturriaga, S. (2019). Cluster-UY: Collaborative Scientific High Performance Computing in Uruguay. In M. Torres & J. Klapp (Eds.), *Supercomputing* (pp. 188-202). Springer International Publishing.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 32* (pp. 8024-8035). Curran Associates, Inc.

Philips, J. P. & Tabrizi, N. (2020). *Historical Document Processing: A Survey of Techniques, Tools, and Trends*. arXiv.

Poulos, J. & Valle, R. (2021). Character-Based Handwritten Text Transcription with Attention Networks. *Neural Comput. Appl.*, *33*(16), 10563-10573.

Puigcerver, J. (2017). Are Multidimensional Recurrent Layers Really Necessary for Handwritten Text Recognition? *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR), Kyoto, Japan*, 67-72.

Qin, S. & Liu, S. (2021). Towards end-to-end car license plate location and recognition in unconstrained scenarios. *Neural Computing and Applications*.

Raffel, C., Luong, M.-T., Liu, P. J., Weiss, R. J. & Eck, D. (2017). Online and Linear-Time Attention by Enforcing Monotonic Alignments. In D. Precup & Y. W. Teh (Eds.), *Proceedings of the 34th International Conference on Machine Learning, Sydney, NSW, Australia* (pp. 2837-2846). PMLR.

Rice, S., Nagy, G. & Nartker, T. (2012). *Optical Character Recognition: An Illustrated Guide to the Frontier*. Springer US.

Sahu, D. K. & Sukhwani, M. (2015). Sequence to Sequence Learning for Optical Character Recognition. *CoRR*, *abs/1511.04176*.

Schmidhuber, J., Meier, U. & Ciresan, D. (2012). Multi-column deep neural networks for image classification. *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Los Alamitos, CA, USA*, 3642-3649.

Shi, B., Bai, X. & Yao, C. (2017). An End-to-End Trainable Neural Network for Image-Based Sequence Recognition and Its Application to Scene Text Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*, 2298-2304.

Sriram, A., Jun, H., Satheesh, S. & Coates, A. (2018). Cold fusion: Training Seq2seq Models Together with Language Models. *6th International Conference on Learning Representations, ICLR 2018, Workshop Track Proceedings, Vancouver, BC, Canada*.

Stabile, J., Fernandez, E. & Fioritto, F. (2020). Procesamiento de Lenguaje Natural (PLN) para la reconstrucción de textos a partir de imágenes correspondientes a archivos históricos de

la década del 70 [Tesis de grado]. *Universidad de la República (Uruguay). Facultad de Ingeniería*.

Stratonovich, R. L. (1960). Conditional Markov Processes. *Theory of Probability & Its Applications*, *5*(2), 156-178.

Sueiras, J., Ruiz, V., Sanchez, A. & Velez, J. F. (2018). Offline continuous handwriting recognition using sequence to sequence neural networks. *Neurocomputing*, *289*, 119-128.

Sutskever, I., Vinyals, O. & Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence & K. Weinberger (Eds.), *Advances in Neural Information Processing Systems*. Curran Associates, Inc.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. & Polosukhin, I. (2017). Attention is All you Need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan & R. Garnett (Eds.), *Advances in Neural Information Processing Systems*. Curran Associates, Inc.

Wick, C., Reul, C. & Puppe, F. (2020). Calamari - A High-Performance Tensorflow-based Deep Learning Package for Optical Character Recognition. *Digital Humanities Quarterly*, *14*(1).

Williams, R. J. & Zipser, D. (1989). A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural Computation*, *1*(2), 270-280.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., … Dean, J. (2016). Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *CoRR*, *abs/1609.08144*.

Yang, D., Lavie, A., Dyer, C. & Hovy, E. (2015). Humor Recognition and Humor Anchor Extraction. In L. Màrquez, C. Callison-Burch & J. Su (Eds.), *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, Lisbon, Portugal* (pp. 2367-2376). Association for Computational Linguistics.

Capítulo 7

Anexo

7.1 Evaluación de instancias de entrenamiento

7.1.1 Entrenamiento sobre bloques

TF	LR	#i	CER	Soft CER	BER	LCS	Val. Loss	#Mejor Epoch
0.5	5e-05	1	19.79	19.09	35.29	00.81	00.76	10
0.5	5e-05	2	20.77	20.19	36.66	00.80	00.82	9
0.5	5e-05	3	19.62	19.05	35.63	00.81	00.75	10
0.5	0.0001	1	19.31	18.81	34.78	**00.82**	00.78	10
0.5	0.0001	2	18.77	**18.23**	34.28	**00.82**	00.75	8
0.5	0.0001	3	19.24	18.72	35.02	**00.82**	**00.72**	6
0.5	0.0005	1	19.56	19.05	36.53	**00.82**	00.75	4
0.5	0.0005	2	19.22	18.72	35.69	**00.82**	00.76	4
0.5	0.0005	3	19.33	18.85	35.40	**00.82**	00.73	5
0.5	0.0010	1	24.49	23.53	45.65	00.77	00.91	1
0.5	0.0010	2	22.70	22.05	42.66	00.79	00.83	1
0.5	0.0010	3	21.27	20.72	38.61	00.80	00.90	4
0.8	5e-05	1	19.50	18.92	34.39	**00.82**	00.86	9
0.8	5e-05	2	20.42	19.77	35.63	00.81	00.88	9
0.8	5e-05	3	19.88	19.29	35.07	00.81	00.85	10
0.8	0.0001	1	19.32	18.80	34.47	**00.82**	00.94	10
0.8	0.0001	2	19.63	19.03	34.84	**00.82**	00.89	8
0.8	0.0001	3	19.54	18.99	34.12	**00.82**	00.85	6
0.8	0.0005	1	19.32	18.84	34.51	**00.82**	00.83	6
0.8	0.0005	2	18.96	18.43	**33.97**	**00.82**	00.86	7
0.8	0.0005	3	**18.70**	**18.23**	35.09	**00.82**	00.76	3
0.8	0.0010	1	21.99	21.30	40.82	00.79	00.99	8
0.8	0.0010	2	37.23	36.37	62.81	00.64	01.63	2
0.8	0.0010	3	19.91	19.35	36.70	00.81	00.88	7
1.0	5e-05	1	20.53	19.93	35.54	00.81	01.06	9
1.0	5e-05	2	21.20	20.68	36.54	00.80	01.08	10

1.0	5e-05	3	20.49	19.99	35.68	00.81	01.10	10
1.0	0.0001	1	20.20	19.65	35.14	00.81	01.04	7
1.0	0.0001	2	20.09	19.59	35.31	00.81	01.01	6
1.0	0.0001	3	19.91	19.43	34.83	00.81	01.03	8
1.0	0.0005	1	20.69	20.04	37.63	00.81	00.91	2
1.0	0.0005	2	21.17	20.63	38.12	00.80	00.95	2
1.0	0.0005	3	18.99	18.41	34.64	**00.82**	00.89	3
1.0	0.0010	1	23.95	23.26	44.00	00.77	01.10	1
1.0	0.0010	2	27.56	26.73	49.03	00.74	01.31	1
1.0	0.0010	3	65.24	64.83	81.26	00.37	03.00	2

Cuadro 7.1: En este cuadro pueden observarse cada una de las instancias de entrenamiento para cada configuración con la que se experimentó. Para cada instancia se obtiene la *epoch* para la que se obtuvo el mejor resultado de *CER* junto con los demás valores de esa *epoch*. En negrita se marcan los mejores valores de la tabla.

7.1.2 Uso de múltiples instancias por bloque

#i	CER	LCS	Loss	#Mejor Epoch
1	17.74	0.84	00.91	6
2	18.28	00.83	**00.84**	1
3	**16.77**	**00.85**	00.86	5

Cuadro 7.2: Tabla de resultados para las instancias de entrenamiento de modelo sobre conjunto aumentado.

7.1.3 Entrenamiento sobre conjunto de líneas ensambladas

Prueba	#i	CER	LCS	Loss	#Mejor Epoch
Solo líneas	1	28.24	00.74	02.05	11
Solo líneas	2	29.30	00.73	02.00	8
Solo líneas	3	47.66	00.56	02.85	9
Combinado	1	26.93	**00.75**	02.04	6
Combinado	2	**26.77**	**00.75**	02.00	6
Combinado	3	26.85	**00.75**	02.08	8
Dos etapas	1	27.33	**00.75**	01.93	2
Dos etapas	2	27.94	00.74	01.96	2
Dos etapas	3	27.35	**00.75**	01.95	1
Dos etapas, bajo LR	1	27.20	**00.75**	02.06	5
Dos etapas, bajo LR	2	27.95	00.74	01.99	3
Dos etapas, bajo LR	3	27.25	**00.75**	01.99	1

Cuadro 7.3: Tabla de resultados sobre el conjunto de validación para la mejor *epoch*. El caso de «Dos etapas, bajo LR» es el entrenamiento del escenario «Dos etapas» con una tasa de aprendizaje menor.