Universidad de Valladolid
Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Mención en Computación

DocScan: Deep Learning para la segmentación de documentos escaneados

---

Alumno: Jorge San José Lorza

Tutor: Teodoro Calonge Cano

Índice general

Lista de figuras III

Lista de tablas V

Resumen IX

Abstract XI

1. Introducción 1
- 1.1. Motivación . . . 2
- 1.2. Estructura . . . 3

2. Marco teórico 5
- 2.1. Redes Neuronales Artificiales . . . 5
- 2.2. Redes Neuronales Convolucionales . . . 7
  - 2.2.1. Operación Convolución . . . 7
  - 2.2.2. Pooling . . . 11
  - 2.2.3. Estructura de una CNN para tratamiento de imágenes . . . 13
- 2.3. Arquitectura UNET . . . 14
  - 2.3.1. Propuesta original . . . 15
  - 2.3.2. Upsampling . . . 17
  - 2.3.3. Técnica del residuo . . . 21
- 2.4. Otros aspectos relevantes . . . 23
  - 2.4.1. Normalización . . . 23
  - 2.4.2. Funciones de activación . . . 24
  - 2.4.3. Funciones de pérdida . . . 27

3. Gestión del proyecto 29
- 3.1. Metodología de trabajo . . . 29
- 3.2. Épicas del proyecto . . . 30
- 3.3. Planificación . . . 30
  - 3.3.1. Planificación inicial . . . 31
  - 3.3.2. Variaciones respecto a la planificación inicial . . . 32
- 3.4. Gestión de la configuración . . . 32

4. Plataforma de trabajo e implementación **35**
4.1. Conjunto de datos 35
4.1.1. Descripción 36
4.1.2. Objetivos 38
4.2. Tratamiento de los datos 41
4.2.1. Generación del Ground Truth 41
4.2.2. Algoritmos de Image Chopping 44
4.3. Implementación 51
4.3.1. Pytorch vs Keras 51
4.3.2. Descripción de la aplicación desarollada 52
4.3.3. Configuración de la red neuronal 56
4.3.4. Parámetros relativos al entrenamiento 61
4.3.5. Creación de un modelo en PyTorch 63

5. Resultados y API **69**
5.1. Resultados 69
5.1.1. Evaluación caso continuo 72
5.1.2. Evaluación del caso binario 73
5.2. API 82
5.2.1. Composición fichero HTML 82
5.2.2. Implementación del servidor 85

6. Conclusiones **89**

A. Manual de instalación **93**

B. Manual de usuario **95**

Bibliografía **99**

Índice de figuras

2.1. Partes de una neurona biológica. 5
2.2. Similitud entre una neurona biológica y una artificial. 6
2.3. Ejemplo de una convolución 2D. 9
2.4. Comparación conectividad densa y conectividad dispersa. 11
2.5. Ejemplo de average pooling con mosaico 2x2. 12
2.6. Ejemplo de max pooling con mosaico 2x2. 12
2.7. Ejemplo de convolución en una imagen con tres canales de información. . . 13
2.8. Arquitectura UNET propuesta por sus autores. 16
2.9. Entrada y núcleo para el ejemplo de convolución transpuesta [10]. 18
2.10. Ejemplo de convolución transpuesta [10]. 18
2.11. Ejemplo de bilinear upsampling [11]. 19
2.12. Ejemplo de pixel shuffle con factor de escalado igual a 2. 21
2.13. Ejemplo explicativo técnica del residuo [13]. 22
2.14. Gráfica de la función de activación *ReLU*. 26
2.15. Gráfica de la función de activación *Leaky ReLU*. 27

3.1. Planificación inicial del proyecto. 31
3.2. Organización del repositorio. 33

4.1. Mosaico de cuatro documentos pertenecientes al conjunto de datos. 36
4.2. Ejemplo de una imagen del dataset junto a su groundtruth. 37
4.3. Ejemplo de elementos pertenecientes a la categoría separadores. 40
4.4. Ejemplo de imágenes de Ground Truth generadas. 42
4.5. Generación de sub-cuadrantes en recursive chopping. 46
4.6. Imagen del dataset sobre la que se mostrará el proceso de recursive chopping. 47
4.7. Cuatro primeros chunks obtenidos para la imagen de la Figura 4.6. 48
4.8. Generación de chunks en iterative chopping. 50
4.9. Relación de los elementos del esqueleto software. 52
4.10. Arquitectura de la red UNET implementada. 57
4.11. Arquitectura red neuronal I. 58
4.12. Arquitectura red neuronal II. 59
4.13. Arquitectura red neuronal III. 59
4.14. Arquitectura red neuronal IV. 60

5.1. Evolución de las tasas de error a lo largo del entrenamiento. 70
5.2. Salida obtenida por la red para una imagen dada. 71
5.3. Binarización, con un threshold de 0.5, de la salida de la red mostrada en la Figura 5.2. 71
5.4. Curva ROC de la clase *pie de foto*. 75
5.5. Curva ROC de la clase *texto flotante*. 76
5.6. Curva ROC de la clase *pie de página*. 77
5.7. Curva ROC de la clase *títulos*. 78
5.8. Curva ROC de la clase *imágenes*. 79
5.9. Curva ROC de la clase *número de página*. 80
5.10. Curva ROC de la clase *párrafos*. 81
5.11. Comparación de un documento escaneado con su recreación en formato HTML. 84
5.12. Pantalla principal del servidor. 86
5.13. Pantalla de espera del servidor. 86

Índice de cuadros

4.1. Número de apariciones de cada categoría en el dataset. 38
4.2. Número de apariciones de cada sub-categoría del tipo Texto. 39
4.3. Resumen de las categorías de interés. 40
4.4. Sufijos de las categorías de interés. 43
4.5. Imágenes de Ground Truth generadas por categoría. 44

5.1. Evaluación continua por categoría. 72
5.2. Métricas de evaluación binaria para la categoría *pie de foto*. 74
5.3. Métricas de evaluación binaria para la categoría *texto flotante*. 75
5.4. Métricas de evaluación binaria para la categoría *pie de página*. 76
5.5. Métricas de evaluación binaria para la categoría *títulos*. 77
5.6. Métricas de evaluación binaria para la categoría *imágenes*. 78
5.7. Métricas de evaluación binaria para la categoría *número de página*. 79
5.8. Métricas de evaluación binaria para la categoría *párrafos*. 80
5.9. Valores de threshold recomendados para cada clase. 81
5.10. Área mínima para considerar un elemento. 84

Agradecimientos

En primer lugar, a los tutores de la empresa HP: Alejandro Viloria Lanero, Héctor Chamorro Álvarez y Guillermo Ménguez Álvarez, por su involucración en este proyecto, en el que han puesto todo su conocimiento a mi disposición.

Gracias a todos los profesores que han participado en mi formación académica, en especial a Teodoro, por guiarme en el transcurso de este proyecto, siendo una parte activa del grupo de trabajo establecido con HP.

A mi ex-compañero y amigo Guillermo, por prestar su ordenador personal para el entrenamiento de la red y por su preocupación por la evolución del proyecto durante estos meses.

Gracias a mi pareja, Marta, y mis amigos, por confiar en mí y animarme en los momentos más difíciles, sin ellos este trabajo no hubiese salido adelante.

Y sobre todo, gracias a mi familia, por su esfuerzo durante estos años en los que siempre me han brindado todas las facilidades posibles para que mi única preocupación fuese completar mi formación académica.

Resumen

Desde que existe Internet, los grandes archivos de documentos se han convertido en uno de los sitios más interesantes para acceder. Sus consultas exigen cada día una recuperación de información más sofisticada. Estas técnicas de búsqueda se basan en varios paradigmas diferentes. En particular, la Inteligencia Artificial se está aplicando hoy en día en el procesamiento emergente de la gestión de documentos, por ejemplo, identificación de áreas de papel como fotos, gráficos, encabezados, resumenes, pies de página, etc. Una posterior ejecución de OCR nos traerá el texto al interior de cada una de las zonas de papel mencionadas. Esta información se almacena en el campo correspondiente en el Sistema de Base de Datos disponible a nivel mundial, ya que estos campos serán el objetivo de la búsqueda de los usuarios.

Además, el reconocimiento de áreas mencionadas en un determinado papel también se puede utilizar para convertir el documento original escaneado a otro formato más adecuado como XML. En el presente trabajo, esta tarea se presenta bajo un problema de Visión Artificial, en particular, como una segmentación de imágenes que reconocerá cada una de estas áreas del documento.

Concretamente, nuestro enfoque se basa en Redes Neuronales Artificiales Convolucionales bajo una arquitectura UNET. Originalmente se propuso en 2015 para ser utilizado en un reconocimiento de imágenes biomédicas. La implementación se ha realizado utilizando PyTorch con una precisión satisfactoria.

Finalmente, con el fin de mejorar la accesibilidad de nuestro sistema, se creó un servidor html. El escaneo de documentos se introducirá en esta aplicación y devolverá su verdad básica.

Palabras claves: Aprendizaje Profundo, segmentación, documentos, UNET, HTML, OCR.

Abstract

Since internet exits, the large document archives became one of the most interesting sites to be accessed. Theirs consults are demanding more sophisticated information retrieving everyday. These searching techniques are based on several different paradigms. In particular, the Artificial Intelligence is nowadays applying in the emergent document managing processing, for instance, paper areas identification like photo, graphics, header, abstract, footer, etc. It is really the first step to lead to a useful document searching, because a posterior OCR execution will bring us the text inside in each of the mentioned paper zones. This information is stored in the corresponding field in the Database System available worldwide, since these fields will be the target of the users searching.

In addition, the mentioned areas recognition in a given paper can be also used to convert the original scanning document into another more suitable format like XML. In the present work, this task is presented under a Vision Artificial problem, in particular, as image segmentation that will recognize each one of these document areas.

More precisely, our approach is based on Convolutional Artificial Neuronal Networks under a UNET architecture. It was originally proposed en 2015 to be used in a biomedical image recognition. The implementation has been made using PyTorch with a satisfactory accuracy.

Finally, in order to improve the accessibility of our system, a html server was created. Scanning documents will be introduce to this application and it will return their ground-truth.

Key words: Deep Learning, segmentation, documents, UNET, HTML, OCR.

Capítulo 1

Introducción

Para medir los distintos periodos históricos que se han sucedido a lo largo de la *Historia de la Humanidad* podemos utilizar las *eras* o las más comúnmente usadas *edades*. Las *eras* se basan en la relevancia e impacto en la sociedad de un cierto personaje, proceso o hecho para determinar un periodo histórico. En la actualidad nos encontramos inmersos en la era de la *digitaliación*.

La digitalización es el proceso de transformación de procesos analógicos y objetos físicos a un formato digital. Durante los últimos años este término ha estado en boca de todos debido a los procesos de *transformación digital* llevados a cabo por empresas e instituciones, tanto públicas como privadas.

La digitalización y la transformación digital han traído consigo una gran cantidad de cambios en los hábitos de la sociedad. Cada vez más se opta por un consumo digital por encima de lo analógico o tradicional. Un claro ejemplo podemos encontrarlo en la disminución año a año de la venta de ejemplares impresos de periódicos y/o revistas, mientras sus portales web van ganando en visitas e importancia. De la misma forma, el número de consultas a bibliotecas digitales ha sufrido un crecimiento exponencial, por lo que muchas de ellas han optado por ir incorporando a sus catálogos una amplia variedad de libros y documentos, para que sus usuarios puedan disfrutar de ellos.

En estas bibliotecas, es frecuente encontrarse con documentos cuyo proceso de digitalización ha consistido en someterlos a un escaneo. Esta técnica es bastante usada, ya que es rápida, barata y ofrece un documento final con un alto nivel de resolución. A pesar de ello, no suele ser del gusto del usuario pues al tratarse de una imagen no puede realizarse una búsqueda de información por palabras y/o frases, lo que dificulta la labor de investigación que se esté realizando.

En el presente *Trabajo Final de Grado* (en adelante *TFG*) se presenta una posible solución para este problema. Está basada en la identificación de las diferentes zonas de un documento (texto, fotos, gráficos, encabezados, etc.) para su posterior recreación en

formato *HTML* de forma que el texto, una vez procesado por un OCR, pueda ser consultado dentro de su contexto.

Para el reconocimiento de las distintas zonas de un documento se ha implementado una red neuronal convolucional, en particular, bajo una arquitectura *UNET*. Esta se caracteriza por asignar a cada píxel de la imagen de entrada una categoría o probabilidad de pertenencia a la misma, de tal manera que, para cada zona del documento, se genera una nueva imagen del mismo tamaño que la original, en la que se encuentran aislados los elementos que pertenecen a dicha categoría. En nuestro caso, reconoceremos hasta siete categorías distintas, por lo que la red implementada devolverá siete imágenes de salida por cada imagen original cargada.

Tras identificar las diversas zonas del documento, estamos en disposición de recrearlo en un nuevo formato pero, antes, algunas de las zonas han de ser sometidas a un procesamiento previo. Este es el caso de las categorías de texto, como pueden ser los encabezados, los párrafos o los pies de foto. Mediante el *reconocimiento óptico de caracteres* (*OCR*) es factible identificar en una imagen los caracteres o símbolos pertenecientes a un determinado alfabeto, lo que posibilita su almacenamiento en formato de datos editable. Es decir, a través del OCR, podemos extraer en texto plano las palabras y frases que aparecen en una imagen. Esta tecnología es fundamental en nuestra solución, ya que es la que permite realizar la búsqueda de una palabra o expresión en el documento, algo que no se podía hacer sobre el archivo original, porque se mostraba el texto como parte de una imagen.

Tras reconocer las diversas zonas del documento en cuestión y haber realizado convenientemente los procesos de OCR necesarios, se genera el documento en el nuevo formato escogido. En este TFG, se ha optado por una recreación en formato *HTML* pero, tanto la red neuronal, como los procesos de OCR, proporcionan todo lo necesario para reproducir el documento en cualquier otro formato.

El conjunto de datos con el que se ha trabajado se denomina *Layout Analysis Dataset* [1]. Fue recabado y proporcionado por la *Universidad de Salford*. Está conformado por casi 500 imágenes en alta resolución, que se corresponden con páginas escaneadas de documentos de diversos tipos. Para cada una de estas imágenes, se cuenta con un archivo en formato *XML*, en el que se contiene la información necesaria para identificar la categoría presentada en cada zona del documento.

1.1. Motivación

Las redes neuronales tienen su origen en el año 1943, con la publicación por parte de *McChulloch* y *Pitts* [2] del primer modelo de neurona artificial. Desde entonces, se han popularizado hasta convertirse, a día de hoy, en una de los modelos computacionales

más utilizados. Cuando en dichos modelos existen una gran cantidad de capas de neuronas, que permiten un mayor nivel de extracción de características y, por tanto, un mayor aprendizaje, hablamos de *Aprendizaje Profundo* (en adelante AP).

Dos aspectos fueron clave en la popularización del AP en las últimas décadas. Por un lado, la *ingeniería de características* encontró en el AP un método eficaz y objetivo de filtrado de atributos. Hasta entonces, esta tarea estaba supeditada al conocimiento del ingeniero, lo que la hacía una etapa crucial y delicada, ya que una incorrecta interpretación de las variables seleccionadas daba lugar a malos resultados. Con la inclusión de un mayor número de capas, esta tarea de selección de variables queda a cargo de las capas de más bajo nivel, mientras que las capas más cercanas a la salida se encargan de realizar la clasificación basándose en las características que han sobrevivido al filtrado. Por otro lado, el auge de la *computación heterogénea*, promovida por el abaratamiento del hardware, ha permitido, y permite, contar con grandes unidades de procesamiento a bajo coste. Esto es fundamental, ya que el AP precisa de unas prestaciones elevadas, por lo que la existencia de estas potentes unidades de cómputo a un precio bajo ha hecho que muchas personas hayan podido formar parte de la comunidad del AP que, en buena medida, se sustenta en las aportaciones de los usuarios finales, como parte de un desarrollo colaborativo a gran escala.

1.2. Estructura

Para una correcta exposición, el TFG se ha dividido en:

- **Capítulo 2. Marco teórico.** Se explican los conceptos teóricos sobre los que se sustenta el trabajo. En primera instancia, se realiza una breve exposición sobre las redes neuronales en general, para introducir una exposición más detalladas de las redes convolucionales, que han sido el núcleo de este trabajo. En segunda instancia, se trata con un alto nivel de detalle el modelo de red *UNET*, arquitectura que combina de una determinada manera diversas capas de redes convolucionales.

- **Capítulo 3. Gestión del proyecto.** Apartado en el que se desgranan las decisiones tomadas sobre la organización temporal de las tareas a realizar.

- **Capítulo 4. Plataforma de trabajo e implementación**. La primera parte de este capítulo se centra en la descripción del conjunto de datos utilizado y los tratamientos a los que han sido sometidos previamente a su entrada al sistema. En la segunda parte, se explica el esqueleto software de la solución y se detallan las características de la red UNET implementada.

- **Capítulo 5. Resultados y API.** Se evalúa la red implementada y se explica el proceso de recreación del documento en formato HTML. Asimismo, se detallan los pasos para el lanzamiento de un servidor, que permite la interacción con la

red neuronal, para que cualquier usuario pueda enviar su documento escaneado y obtener el fichero HTML correspondiente.

- **Capítulo 6. Conclusiones.** Se realiza una reflexión sobre el trabajo realizado y el aprendizaje que ha supuesto, así como las líneas de trabajo futuras.

Capítulo 2

Marco teórico

En este capítulo, se expone el concepto de red neuronal artificial y los fundamentos teóricos sobre los que se sustentan las redes convolucionales. A partir de ellas, se construirá una red UNET, de la que se hará un estudio más detallado, debido a que no se ha visto en la carrera y, por lo tanto, ha sido uno de los retos destacados de este TFG.

2.1. Redes Neuronales Artificiales

Una red neuronal artificial es un modelo computacional conformado por un conjunto de *neuronas artificiales* conectadas entre sí para transmitirse señales, con el objetivo de producir un valor de salida, que dé solución a un problema concreto.

El concepto de neurona artificial fue introducido por primera vez por *Warren McCulloch* y *Walter Pitts* en su trabajo [2] del año 1943. En él, propusieron el primer modelo neuronal moderno, definiendo a la neurona artificial como una unidad de cálculo que intenta imitar el comportamiento de una neurona natural del cerebro humano. Para comprender su funcionamiento, vamos a fijarnos antes en cómo funciona, a pequeña escala, una neurona natural o biológica.

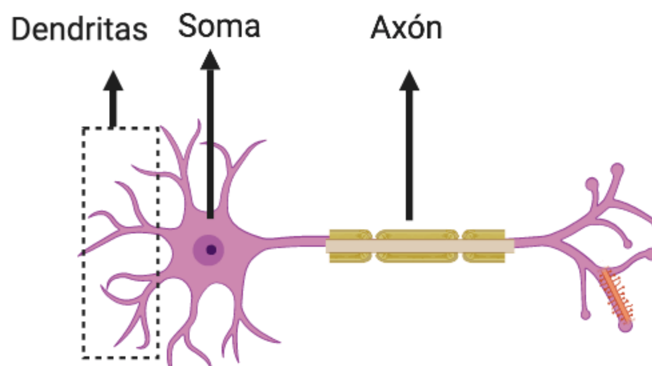


Figura 2.1: Partes de una neurona biológica.

Las principales partes de una neurona biológica son las dendritas, el soma y el axón. A través de las dendritas, las neuronas reciben información externa que procesan en el núcleo o soma, la cual es enviada a otras neuronas a través del axón.

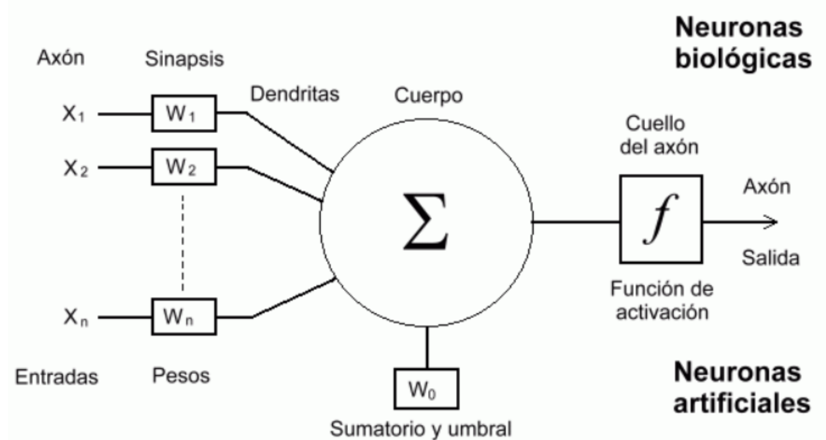


Figura 2.2: Similitud entre una neurona biológica y una artificial.

Como se puede ver en la Figura 2.2, en el modelo matemático planteado por McCulloch y Pitts [2], las entradas $x_1, x_2, x_3, ..., x_n$ simulan la utilidad de las dendritas en la neurona biológica. Cada una de estas entradas tiene asociado un peso, $w_1, w_2, w_3, ..., w_n$, que se emplea para replicar el proceso de la sinapsis en la neurona. La representación del núcleo viene dada por la suma de los procesos de sinapsis y se denomina *salida analógica (u)*. Por último, para obtener la *salida de interés (y)*, se somete a u a una transformación por medio de la *función de activación (F(u))*, que emula el cuello del axón. La formulación del modelo es la siguiente:

$$u = w_0 + \sum_{j=1}^{n} w_j \cdot x_j \tag{2.1}$$

$$y = F(u) \tag{2.2}$$

El objetivo es ajustar el valor de los pesos $w_0, ..., w_n$ de tal manera, que se obtenga una salida de la red lo más parecida posible a la deseada. En el modelo propuesto por McCulloch y Pitts, la información se propaga únicamente hacia delante, por lo que los pesos de una neurona están determinados exclusivamente por el resultado generado mediante las neuronas previas. En el año 1974, *Werbos* [3], motivado por la búsqueda de una mejora

en el rendimiento de las redes, dio forma a la idea de la *propagación hacia atrás*. Mediante este mecanismo, las neuronas se sirven de la salida de los nodos posteriores para reajustar sus pesos e ir obteniendo una solución más próxima a la real. A este fin, la función de activación contribuye con un papel determinante en el rendimiento de la red, ya que se encarga de modelar la salida definitiva de cada una de las neuronas, por lo que una mala elección puede lastrar el modelo.

Existen multitud de topologías de redes neuronales distintas. Algunas de ellas son: Perceptrón Simple, Perceptrón Multicapa, Redes Neuronales Recurrentes, Redes Neuronales Autoorganizadas... Para el desarrollo de este trabajo estamos interesados en las *Redes Neuronales Convolucionales*.

2.2. Redes Neuronales Convolucionales

Los inicios de las Redes Neuronales Convolucionales (CNN) se remontan al año 1979 con la publicación por parte de *Kunihiko Fukushima* de la red neuronal denominada *Neocognitron* [4], que sentó las bases de las CNN. El Neocognitron es una red jerárquica y multicapa, que se usó principalmente para el reconocimiento de caracteres manuscritos en japonés.

Para el desarrollo del Neocognitron, Kunihiko Fukushima se basó en el hallazgo por parte de *Hubel* y *Wiesel* [5], en el año 1959, de dos tipos de células distintas en el cortex primario. La combinación de dichas células, en un modelo en cascada, permitía reconocer patrones visuales complejos mediante una correcta unión de componentes de menor nivel, que habían sido previamente reconocidos por el modelo.

En el año 1998, *Yann LeCun* [6] mejoró el rendimiento del Neocognitron introduciendo la propagación hacia atrás en el modelo. Desde entonces, las redes neuronales convolucionales han ido ganando en importancia, hasta convertirse en una de las mejores alternativas para afrontar el procesamiento de imágenes.

2.2.1. Operación Convolución

La *convolución* es un tipo especializado de operación lineal que da nombre a este tipo de redes. Varios autores han descrito a las redes convolucionales como *"simples redes neuronales que emplean la operación convolución en lugar de la multiplicación general de matrices en, al menos, una de sus capa"* [7].

La convolución permite expresar, en términos de una función, la relación existente entre dos funciones de entrada. Matemáticamente la expresión para esta operación es:

$$H(x) = \int_{-\infty}^{\infty} I(z)K(x - z)\,dz \tag{2.3}$$

A partir de la expresión en forma continua, se llega a la versión discretizada de la misma, que es la que se emplea en este trabajo:

$$H(x) = \sum_{m=-\infty}^{\infty} I(m)K(x - m) \tag{2.4}$$

siendo:

- I los datos de entrada, normalmente un array multidimensional.
- K el kernel, que es un array multidimensional de los parámetros de los que se desean aprender los valores. Suele ser de menor tamaño que I.
- H la salida de la operación convolución, denominada en el argot de las CNN como el *mapa de características*.

Tal y como se puede observar en (2.4) la suma se realiza sobre todo el espacio. Para poder realizar la operación, se determina que I y K sean dos tensores, que valen 0 en todas sus posiciones, a excepción de aquellas que se corresponden con el dominio de interés en los que toma los valores correspondientes. De esta forma, la suma se reduce al conjunto finito de puntos deseado.

En el caso que nos atañe, los datos de entrada son imágenes. Para poder emplear la operación convolución, es necesario interpretar cada una de ellas como un array bidimensional. Igualmente, el kernel también puede ser considerado en dos dimensiones, de tal forma que la formulación discreta queda de la siguiente forma:

$$H(i, j) = \sum_{m} \sum_{n} I(m, n)K(i - m, j - n) \tag{2.5}$$

Existen otras formulaciones de (2.5) que son más óptimas para la resolución de problemas de Aprendizaje Profundo. Las dos más habituales son:

$$H(i, j) = \sum_{m} \sum_{n} I(i - m, j - n)K(m, n) \tag{2.6}$$

$$H(i, j) = \sum_{m} \sum_{n} I(i + m, j + n)K(m, n) \tag{2.7}$$

La formulación propuesta en (2.6) se obtiene al aplicar la propiedad conmutativa en (2.5), mientras que (2.7), denominada *correlación cruzada*, no representa estrictamente la operación convolución, ya que no emplea el kernel volteado para hacer los cálculos. Aun así, en muchas bibliotecas de Aprendizaje Profundo, se utiliza bajo el nombre de convolución.

Como se ha especificado con anterioridad, lo más habitual es que el Kernel sea de menor tamaño que los datos de entrada por lo que, para realizar la convolución, es necesario interpretar el Kernel como una ventana móvil que se va desplazando a lo largo de los datos de entrada para ir realizando la operación. Esta idea puede verse claramente en la Figura 2.3, en la que se toma como datos de entrada un array bi-dimensional, como puede serlo una imagen.

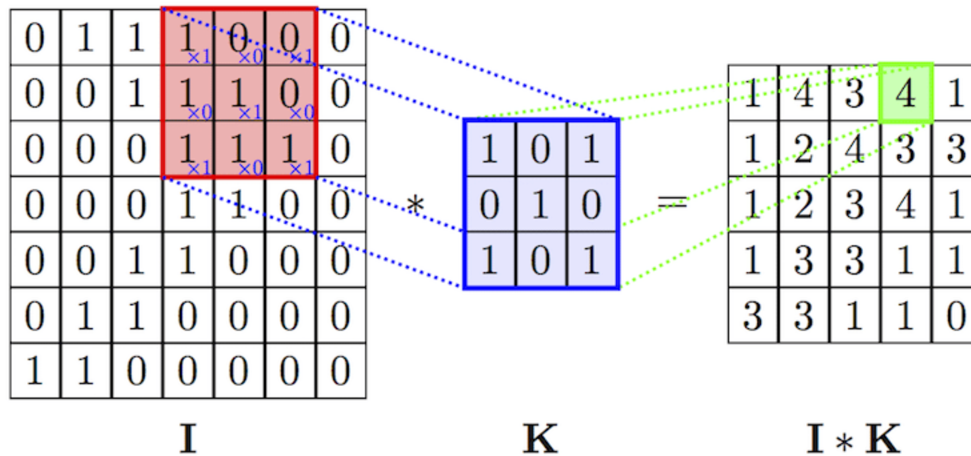


Figura 2.3: Ejemplo de una convolución 2D.

El kernel se desplaza a lo largo de la imagen siguiendo un movimiento de izquierda a derecha y de arriba a abajo, al igual que nuestra forma de lectura. Para cuantificar el desplazamiento entre una aplicación del kernel y la siguiente tenemos el concepto de *strides*. Mediante él, podemos modelizar el número de píxeles que debe moverse el kernel en cada desplazamiento que realice, tanto en la dirección horizontal como en la vertical. Si no se indica lo contrario, el kernel empleado es el (1,1) lo que significa que en cada desplazamiento ya sea horizontal o vertical, el núcleo avanzará un pixel. En el caso de que el *stride* sea (2,1), el núcleo debe desplazarse dos píxeles en cada movimiento horizontal y uno por cada desplazamiento vertical.

Tal y como se observa en la Figura 2.3, el mapa de características resultante es de menor tamaño que los datos de entrada, fenómeno que se denomina *efecto de bordes*. Aunque en un principio no pueda parecer problemático, realmente sí lo es, ya que si la red neuronal cuenta con múltiples capas, es posible que en algún punto de la misma se deje de contar

con datos de entrada al haber sido reducida su dimensionalidad, incluso, hasta llegar a cero. Para poner remedio a este fenómeno, se introduce el *padding*, que consiste en añadir el número de filas y columnas necesarias a cada lado del mapa de características tomado como entrada por una convolución. De esta forma, la salida tiene el mismo tamaño que la entrada. Lo más habitual es que las filas y columnas añadidas sean nulas, es decir, estén compuestas por 0's.

En función de la decisión tomada sobre el padding, se pueden identificar tres posibles casos [8]:

- *valid convolution*: no se realiza padding, lo que tiene como consecuencia la disminución del mapa de características después de cada convolución.

- *same convolution*: se añaden filas y columnas hasta que el tamaño del mapa de características de salida sea igual al tamaño de la entrada. Tiene como consecuencia una disminución de la influencia de los pixeles cercanos al borde.

- *full convolution*: se añaden tantas filas y columnas como sea necesario para que todos los pixeles sean "visitados"por el kernel el mismo número de veces. De esta manera se solventa el problema de una influencia mayor de unos pixeles respecto a otros.

Entre las ventajas que ofrece la operación convolución frente al multiplicado tradicional de matrices en las redes neuronales, se encuentran las siguientes:

- *Conectividad dispersa*: A diferencia de lo que ocurre con las redes tradicionales, en las convolucionales, no es necesario que las neuronas de una capa tengan como entrada las salidas de todos los nodos de la capa previa, la conocida como conectividad densa. Esto se debe al empleo de un kernel de menor tamaño que la entrada. Teniendo en cuenta que, cada unión entre dos neuronas, se considera como un parámetro en la red, esto supone una reducción de los requisitos de memoria y procesamiento del modelo, así como una mejora en su eficiencia. Igualmente, permite detectar características simples, como líneas, cuya composición puede dar lugar a la detección de características más complejas.

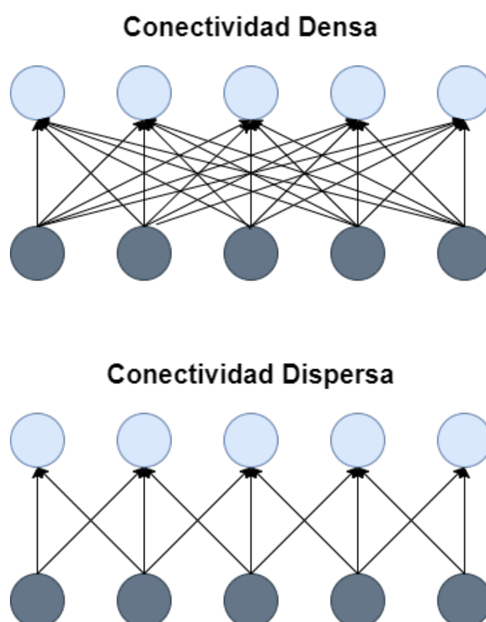


Figura 2.4: Comparación conectividad densa y conectividad dispersa.

- *Compartición de parámetros*: mientras que en las redes neuronales, cada elemento de la matriz de pesos se usa una vez, en las redes convolucionales cada peso del kernel se emplea en cada operación de convolución. Esto tiene como consecuencia un aprendizaje basado en un único conjunto de datos.

- *Representaciones equivalentes*: el impacto de una traslación en el conjunto de entrada no afecta a la salida, ya que la convolución verifica la equivarianza entre ambas. Esto no es trasladable a la rotación o el escalado.

La convolución es la operación fundamental de las CNN, pero no es la única función diferencial de este tipo de redes, también es de vital importancia el submuestreo de características o *pooling*.

2.2.2. Pooling

El submuestreo de características es fundamental para el correcto funcionamiento de las CNN, ya que permite dotar a la red neuronal de la capacidad suficiente como para clasificar imágenes correctamente, sin que importe que existan variaciones en las posiciones de los objetos en la imagen. Esto quiere decir que la clasificación se centrará en detectar la presencia o ausencia de unas ciertas características, sin importar su localización.

Asimismo, la aplicación del pooling aporta a las CNN una capacidad de invarianza frente a ciertas distorsiones del conjunto de entrada, como las rotaciones o los escalados, algo que no se conseguía a través de la convolución.

La reducción de características se lleva acabo combinando las salidas de los grupos de neuronas de una capa en una sola. Se suelen utilizar grupos pequeños en mosaicos, como por ejemplo, 2x2. Los dos tipos de pooling más populares son:

- *Average pooling*: sustituye el mosaico por la media estadística de sus celdas

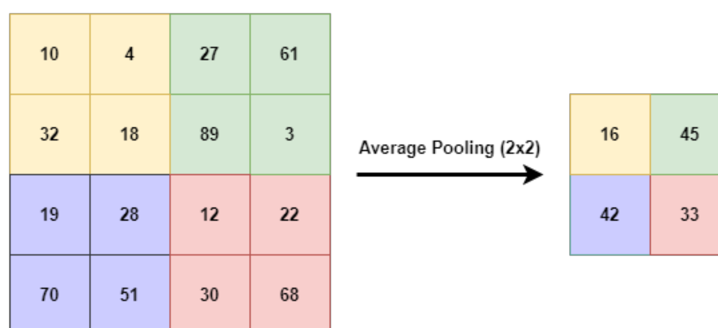


Figura 2.5: Ejemplo de average pooling con mosaico 2x2.

- *Max pooling*: igual que el anterior, pero la operación empleada es el cálculo del máximo, en vez de la media.

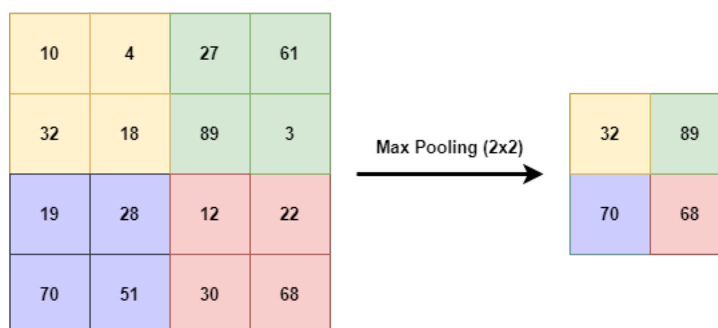


Figura 2.6: Ejemplo de max pooling con mosaico 2x2.

La elección de uno u otro tipo depende del objetivo que se persiga y del conjunto de datos con el que se va a trabajar. Ambos proporcionan buenos resultados en situaciones genéricas pero, en función del problema, será más apropiado el uso de un tipo frente al otro.

2.2.3. Estructura de una CNN para tratamiento de imágenes

A lo largo de las secciones previas, se ha introducido la idea del tratamiento de una imagen como una matriz de pixeles para realizar la operación convolución. En el planteamiento inicial, se considera que la imagen está en escala de grises, es decir, tiene sólo un canal de información. En la realidad, lo más habitual es tratar con imágenes con tres canales de información *(R,G,B)*, a las que comúnmente denominamos "en color".

Para poder tratar correctamente estas imágenes, se realiza una operación convolución por cada canal de entrada y, a continuación, se suman las salidas obtenidas.

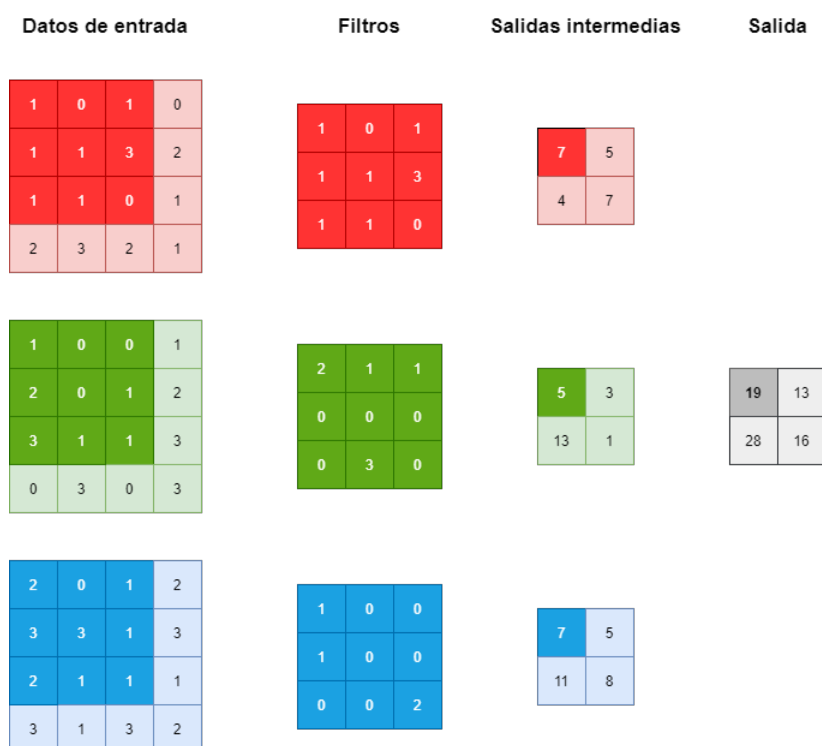


Figura 2.7: Ejemplo de convolución en una imagen con tres canales de información.

Como se puede ver en la Figura 2.7, el kernel no tiene por qué ser igual en todos los canales, sino que pueden almacenar sus propios pesos. Además, en la tarea de clasificación de imágenes es necesaria la aplicación de varios kernel distintos en el mismo canal para poder extraer un mayor número de caraterísticas. Para ello, se aplican, de forma paralela e independiente, varios núcleos distintos a cada capa de entrada, de tal manera, que se generan varios mapas de características por capa y operación convolución. La formulación de esta operación queda definida de la siguiente forma:

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m-1,k+n-1} K_{i,l,m,n} \tag{2.8}$$

siendo:

- V los datos de entrada, con $V_{i,j,k}$ el elemento *(j,k)* del canal i de la imagen
- K el kernel. Ahora es un tensor en 4D, siendo el elemento $K_{i,j,k,l}$ el elemento relativo a *(k,l)* del canal j de entrada y del canal l de salida.

La realización de una operación convolución suele ir asociada a un proceso de *normalización*, que permite redefinir todos los valores de la entrada en un mismo rango. Tras realizar ambos procesos, la salida obtenida se traslada a una *función de activación* que proporciona el resultado final. Estos tres procesos pueden repetirse un número indefinido de veces, dando lugar a una capa convolucional. Una CNN está formada por varias capas convolucionales que toman como entrada la salida de la capa anterior, tras ser sometida a un submuestreo o pooling.

Aunque no se haya mencionado hasta ahora, estas redes también realizan una propagación hacia atrás, como se explicó en la sección 2.1, para ajustar los pesos y mejorar así su eficiencia.

Existen multitud de arquitecturas que se pueden implementar para el tratamiento de imágenes, que tienen como base los conceptos que se han explicado a lo largo de la Sección 2.2, como pueden ser *ResNet*, *CapsNet* o *GoogleNet*. Para el desarrollo de este TFG, se ha optado por el uso de la arquitectura *UNET*.

2.3. Arquitectura UNET

El modelo de red UNET fue desarrollado por *Olaf Ronneberger*, *Philipp Fischer* y *Thomas Brox* en el año 2015, como mejora a las soluciones existentes en los problemas de segmentación de imágenes en el ámbito de la biomedicina [9].

Esta arquitectura está basada en las *Fully Convolutional Networks (FCN)*, o redes completamente convolucionales. El uso típico de las FCN es clasificar una imagen en una determinada categoría, lo cual es insuficiente en muchas tareas de clasificación, como pueden ser el procesamiento de imágenes médicas. En este caso, no se desea una valoración global de la imagen, sino que el objetivo es reconocer la clase a la que corresponde cada pixel de la imagen y así, por ejemplo, poder detectar tumores u otros problemas de salud. Con este objetivo, *Cieran* y el resto de su equipo entrenaron una red neuronal que se ayudaba de una ventana móvil para ir recorriendo las distintas zonas de la imagen, e ir prediciendo la clase de cada pixel en función del resto a su alrededor. Estos determinan el *contexto* del pixel a clasificar y se corresponden con todos aquellos contenidos en la

ventana de observación. Gracias a este enfoque, es posible localizar la ubicación en la imagen de las diversas entidades a reconocer.

Para los autores de la arquitectura UNET, la solución propuesta por *Cieran* tenía dos claros inconvenientes:

- El procesado por parte de la red era demasiado lento, ya que tenía que trabajar por separado con cada región de la imagen. Además, el grado alto de solapamiento entre las ventanas hacía que existiese mucha redundancia.
- No se prestaba especial atención a la relación entre el uso del contexto y la precisión de la localización. Las ventanas de mayor tamaño requerían de un mayor número de capas de submuestreo, lo que reduce la precisión de la localización, mientras que las ventanas de menor tamaño reducen considerablemente el contexto que observa la red.

La arquitectura UNET surge para mitigar estos dos incovenientes.

2.3.1. Propuesta original

En esta sección, se va a explicar la estructura general de las redes UNET utilizando, para ello, la implementación realizada por sus autores en [9].

La arquitectura (ver Figura 2.8) tiene clara forma de *U*, de ahí el nombre de *U-Net*. En ella, se pueden diferenciar tres partes:

- **Ruta de contracción**: es la primera parte de la arquitectura y se trata de una FCN corriente. Se corresponde con la parte izquierda de la Figura 2.8.
- **Cuello de botella**: hace de enlace entre la zona izquierda de la red y la derecha. Se localiza en la parte inferior de la Figura 2.8.
- **Ruta de expansión**: es simétrica a la ruta de contracción y se trata de una FCN en la que se ha sustituido el pooling por una operación de *upsampling*, que permite ir aumentando el número de canales de un nivel a otro. Abarca toda la parte derecha de la Figura 2.8.

Algunos autores encuadran el cuello de botella como parte de la ruta de contracción. Realmente se trata de una parte diferenciada de la arquitectura, ya que es la encargada de unir las dos partes principales de la misma, por lo que no debe considerarse como parte

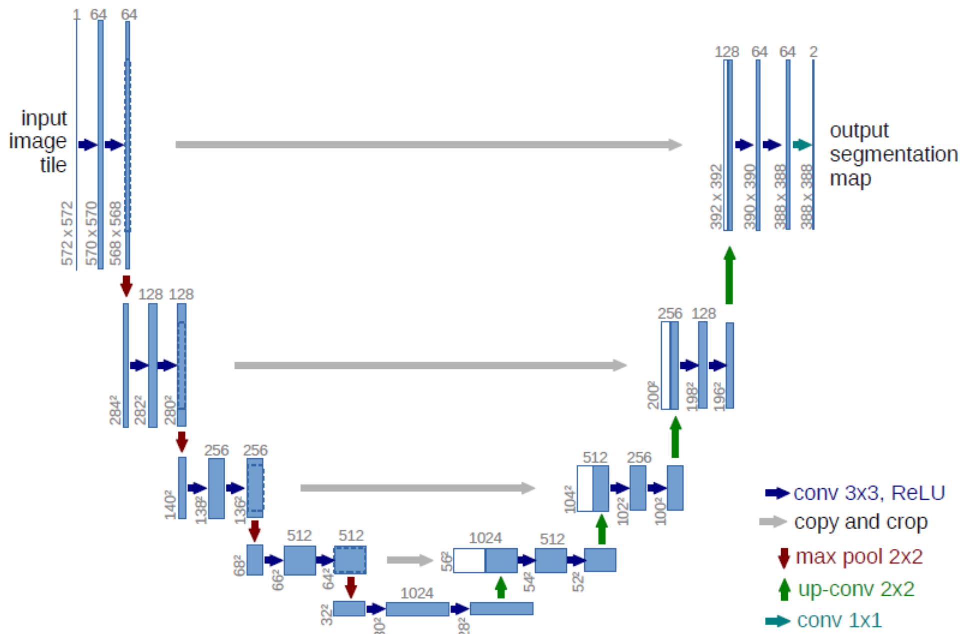


Figura 2.8: Arquitectura UNET propuesta por sus autores.

de ninguna de ellas.

El comportamiento de la red en la ruta de contracción no es novedoso. Cada capa en esta parte de la red está conformada de dos operaciones convolucionales, cada una de ellas seguidas del uso de la función de activación ReLu. Para pasar de una capa a la siguiente, se realiza una operación de pooling. Tal y como está conformada la red, en este caso particular, se duplica el número de canales y se divide a la mitad las dimensiones de entrada, ya que se emplea un mosaico 2x2 con un stride de 2.

En el cuello de botella, se realizan de igual manera las convoluciones, pero la salida obtenida no es sometida a un proceso de pooling. En su lugar, se sustituye este proceso por uno de upsampling, que permite contar con un mayor número de canales para la siguiente capa. En el escenario planteado, se utiliza una convolución transpuesta 2x2 para este proceso, por lo que el número de canales se ve reducido a la mitad para el siguiente nivel, mientras que las dimensiones duplican su valor.

La ruta de expansión toma como entrada la salida del cuello de botella después del upsampling y, en cada nivel, se realizan las mismas operaciones que se dan en los niveles respectivos de la ruta de contracción. La única diferencia es el uso del upsampling, en lugar del pooling, lo que provoca que los datos de entrada dupliquen sus dimensiones, a la vez que ve reducido a la mitad el número de sus canales.

De esta forma, dada una imagen de entrada esta se ve sometida a un proceso de extracción de características, en la que cada vez, se utiliza un contexto de menor tamaño. Finalizada esta extracción de características, bien porque se consideran suficientes, bien porque el contexto es demasiado pequeño, se lleva a cabo un proceso de reconstrucción de la imagen original por medio de las características extraídas. Mediante el upsampling es posible reorganizar la información recopilada para ir generando la imagen final. Esto permite un aprovechamiento máximo del contexto por parte de las capas de más alta resolución, lo que mejora la eficiencia de la red respecto a las FCN. Igualmente, como puede verse en la Figura 2.8, se utiliza la información obtenida en los diversos niveles de la ruta de contracción en las entradas correspondientes a la ruta de expansión, lo que favorece que la red mejore su precisión. A esta práctica se le denomina *técnica del residuo*.

2.3.2. Upsampling

La operación de *upsampling* es la principal novedad de la arquitectura UNET respecto a las CNN tradicionales. Se realiza con el objetivo de reconstruir la imagen de entrada a partir de las características detectadas a lo largo de la ruta de contracción, lo que permite obtener una segmentación en imágenes de cada una de las categorías a reconocer.

El proceso de upsampling propuesto en [9] consiste en una convolución transpuesta 2x2, pero no es la única forma de realizarlo. A continuación, se explicarán algunos de los procedimientos más habituales.

- **Convolución transpuesta**:
  Al igual que ocurre con la operación convolución, la transpuesta [10] está definida por los valores de padding y stride. Lo más habitual es utilizar los mismos valores para estos dos parámetros, que los empleados en la operación convolución. De esta forma, la salida obtenida será del mismo tamaño que la entrada original.

  Sea la entrada de tamaño **(i,i)**, el kernel de tamaño **(k,k)**, el valor del stride denotado como **s** y el del padding como **p**. Los pasos para realizar esta operación son los siguientes:

  1. Calcular los nuevos parámetros z (Ec. 2.9) y p' (Ec. 2.10).

     $$z = s - 1 \tag{2.9}$$

     $$p' = k - p - 1 \tag{2.10}$$

  2. Insertar entre cada columna y fila un número de ceros igual a z. Esto provoca que las dimensiones de la entrada pasen de ser (i,i) a ser $(2 * i - 1, 2 * i - 1)$.

3. Añadir alrededor de la imagen un número de ceros igual a p'.
4. Aplicar la operación convolucional con un stride, **s'** igual a 1, a la imagen de entrada, tras haber sido modificada siguiendo los pasos anteriores.

A modo de ejemplo, sean una entrada y un kernel de dimensiones (4,4) y (3,3), respectivamente, como los mostrados en la Figura 2.9.

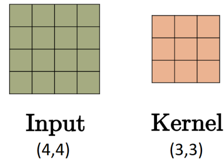


Figura 2.9: Entrada y núcleo para el ejemplo de convolución transpuesta [10].

Considerando un stride y un padding igual a 1 (p = s = 1), los resultados de cada uno de los pasos descritos son los que se observan en la Figura 2.10.

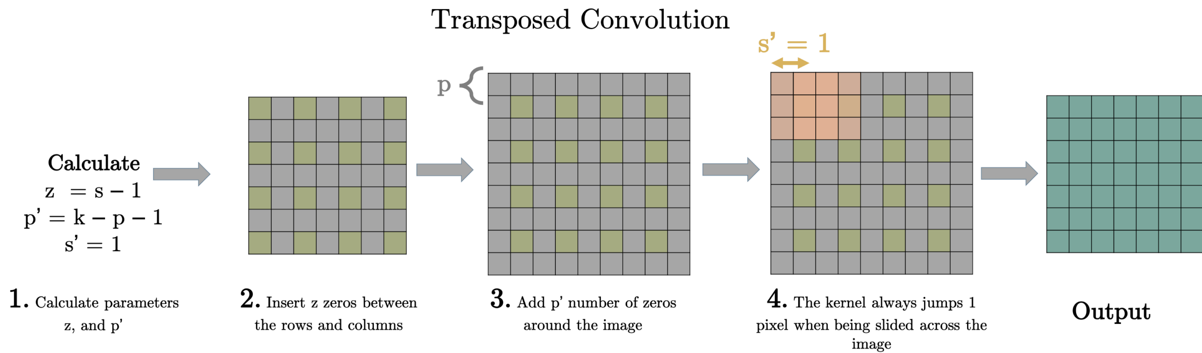


Figura 2.10: Ejemplo de convolución transpuesta [10].

Como podemos ver en la Figura 2.10, la salida obtenida es de tamaño (7,7). Para cualquier conjunto de parámetros que definan la operación convolución transpuesta, el tamaño de salida será **(o,o)**, pudiéndose calcular o como en la Ecuación 2.11.

$$o = (i - 1)xs + k - 2p \tag{2.11}$$

- **Bilinear upsampling**:
  Técnica basada en la interpolación bilineal [11]. Actualmente es una de las más empleadas en el procesamiento de imágenes.

Mediante la interpolación bilineal aplicada a cada pixel de la salida es posible realizar la operación de upscampling. Para cada uno de los nuevos pixels de la salida, se consideran los cuatro pixels conocidos más cercanos para realizar una media ponderada de los mismos e interpolar dicho resultado. Este valor es el que se asigna al pixel desconocido.

Los cuatro pixels conocidos conforman un rectángulo que, en su interior, alberga los nuevos pixels, cuyos valores han de ser calculados. Los pasos para calcular el valor de un pixel nuevo son:

1. Hallar la media ponderada de los valores conocidos situados en las esquinas superiores del rectángulo, teniendo en cuenta la posición relativa del pixel que queremos interpolar respecto a los conocidos.
2. De igual manera, calcular la media ponderada de los valores situados en las esquinas inferiores.
3. Calcular la media ponderada de los dos resultados previos teniendo en cuenta la posición del pixel de referencia.

Para ilustrar los pasos propuestos contamos con la Figura 2.11.

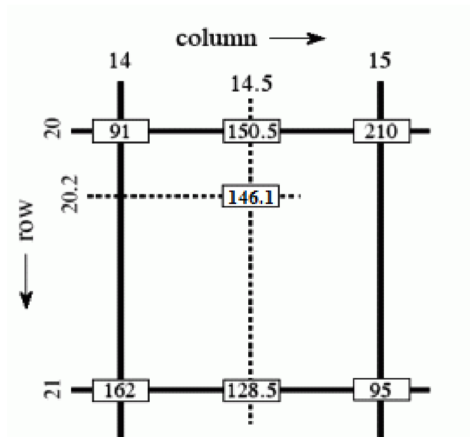


Figura 2.11: Ejemplo de bilinear upsampling [11].

Sean los pixeles conocidos, los correspondientes a las posiciones (20,14), (20,15), (21,14) y (21,15), siendo el primer valor la referencia a la fila y el segundo a la columna. Si el pixel a interpolar es el situado en (20.2,14.5), entonces:

1. Interpolamos para calcular (20,14.5):

$$I_A = \frac{15 - 14{,}5}{15 - 14} \cdot 91 + \frac{14{,}5 - 14}{15 - 14} \cdot 210 = 150{,}5$$

2. Idem con los pixels inferiores, hallando (21,14.5):

$$I_B = \frac{15 - 14{,}5}{15 - 14} \cdot 162 + \frac{14{,}5 - 14}{15 - 14} \cdot 95 = 128{,}5$$

3. Haciendo uso de I_A y I_B calculamos (20.2,14.5):

$$I_{pixel} = \frac{21 - 20{,}2}{21 - 20} \cdot I_A + \frac{20{,}2 - 20}{21 - 20} \cdot I_B = 146{,}1$$

Al igual que la interpolación bilineal, también se utilizan con frecuencia la *nearest-neighbor interpolation* y la *bicubic interpolation*, entre muchas otras.

- **Pixel Shuffle**:
Tiene su origen en el año 2016, cuando se propuso como parte de una red neuronal para el tratamiento en tiempo real de imágenes y vídeos en súper-resolución [12].

Sus autores se basaron en la idea de los sub-pixels de una imagen para crear esta técnica. Un sub-pixel es la unidad mínima que compone un pixel. Por ejemplo, en imágenes en formato *(R,G,B)*, cada pixel está conformado por tres sub-pixels (uno para la componente R, otro para G y un último para B). Trasladando esta idea de sub-pixels al problema del upsampling, cada pixel de la salida obtenida está formado por varios pixels de la entrada, que actúan como si fuesen sus sub-pixels. La relación entre las dimensiones de la entrada y la salida de esta operación es la siguiente:

$$(entrada) \quad (r^2 \cdot C)\ x\ H\ x\ W -> \ C\ x\ (H \cdot r)\ x\ (W \cdot r) \quad (salida)$$

donde:

- r: es el factor de escalado.
- C: indica el número de canales que se generarán en la salida.
- H: número de pixels en la componente “y” de la entrada.
- W: número de pixels en la componente “x” de la entrada.

El número de pixels en la entrada es el mismo que el obtenido en la salida, ya que únicamente se produce una reorganización de los pixels en un menor número de canales, tal y como puede verse en la Figura 2.12.

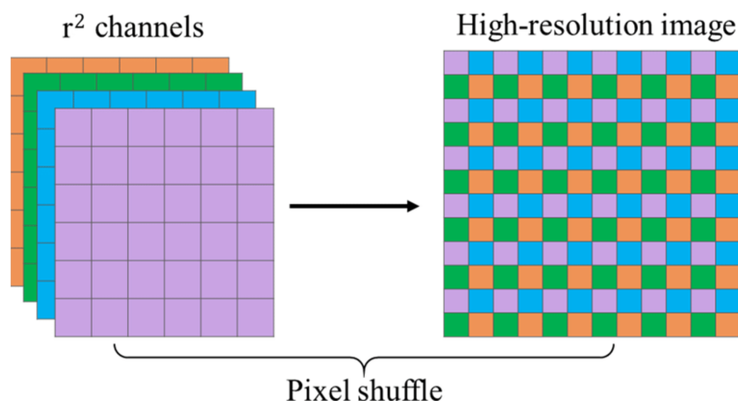


Figura 2.12: Ejemplo de pixel shuffle con factor de escalado igual a 2.

En el caso mostrado en la Figura 2.12, se obtiene un único canal de salida, ya que se considera un factor de escalado igual a la raíz del número de canales. En caso de haber contado con 8 canales de entrada y haber mantenido $r = 2$, se habrían obtenido dos canales de salida generados mediante esta técnica.

Para realizar la reorganización de los pixels, se realiza una convolución sobre la entrada con un stride igual a $\frac{1}{r}$. Si esta operación se realiza con un filtro W_s y un espaciado entre pesos de $\frac{1}{r}$, entonces está demostrado que sólo ciertas partes de W_s estarán activas en un mismo instante de la convolución. En total, existen hasta r^2 patrones de activación diferentes, que se van sucediendo a lo largo de la convolución. Los autores del Pixel Shuffle realizaron una reformulación de la operación convolución, con $stride = \frac{1}{r}$, para aprovechar estos patrones y mejorar así su rapidez. Según los resultados presentados en [12], con esta nueva formulación, la ejecución del upsampling fue $log_2\ r^2$ veces más rápida que con la convolución transpuesta.

2.3.3. Técnica del residuo

Como se mencionó en la Sección 2.1 las redes neuronales utilizan la retropropagación del error para corregir su comportamiento y mejorar su precisión. Para ello, los distintos parámetros de la red ven modificados su valor en función de la magnitud del error cometido en la salida. En las redes que cuentan con múltiples capas, las denominadas redes profundas, es posible que esta retropropagación no alcance a las primeras capas, si las últimas "absorben" todo el error. Esto provoca que las capas iniciales no reajusten sus parámetros y, en consecuencia, no mejoren sus resultados. A este fenómeno se le denomina

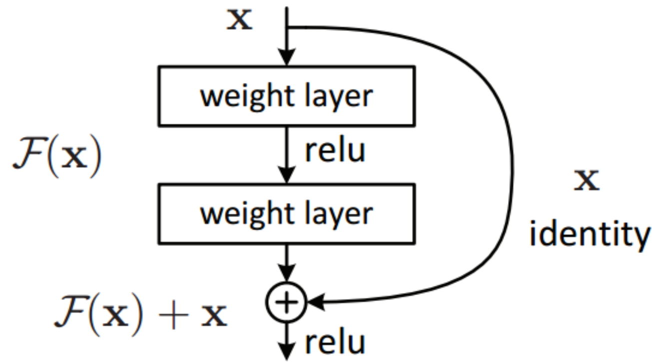


Figura 2.13: Ejemplo explicativo técnica del residuo [13].

desvanecimiento del gradiente.

Para evitar este fenómeno, los autores de la red UNET proponen trasladar las distintas salidas de las capas de la ruta de contracción, a las entradas de las respectivas capas de la ruta de expansión (ver Figura 2.8). En su propuesta, estas salidas son concatenadas con las que deberían ser las entradas de cada una de las capas. De esta forma, la red toma como entrada el doble de canales de los esperados, correspondiendo la mitad a la salida obtenida hasta ese momento y la otra mitad a la salida obtenida en varias capas anteriores. Gracias a ello, la red percibe los errores que se producen en las primeras capas y evita que se produzca el mencionado desvanecimiento del gradiente.

Otra forma de hacer frente a este problema es la denominada *técnica del residuo* [13]. Para explicar el funcionamiento de la misma, se plantea la situación de la Figura 2.13.

Como puede observarse en la Figura 2.13 la técnica del residuo propone trasladar el valor obtenido en cierto punto de la red a otro punto más lejano de la misma, creando lo que se denomina *lazo de residuo*. En el punto final del lazo de residuo se suma, al valor obtenido hasta dicho punto, el obtenido por la red en el punto inicial del lazo, x. La suma de ambos da como resultado un nuevo valor denominado Y, que se puede descomponer como $Y = x + F(x)$. Teniendo en cuenta que x es el valor al inicio del lazo e Y del final del mismo, entonces $F(x)$ es la variación sufrida por x a lo largo de las capas contenidas en el lazo. Por ello, a esto se le denomina *residuo* y, gracias a ellos, la red es capaz de paliar el problema del desvanecimiento del gradiente.

En el caso de la arquitectura UNET, lo más habitual es establecer un lazo de residuo por cada nivel de profundidad de la red. Así, para el caso de la arquitectura original (ver Figura 2.8) se pueden establecer hasta cuatro lazos que lógicamente, irán desde cada una de las capas de la ruta de contracción, a las respectivas en la ruta de expansión.

2.4. Otros aspectos relevantes

A la hora de modelizar una red neuronal existen ciertos elementos, que son imprescindibles, y que no se han descrito en las secciones previas. Es el caso de la *normalización*, las *funciones de activación* y las *funciones de pérdida*.

2.4.1. Normalización

La normalización es un proceso que se lleva a cabo sobre los datos que van a servir de entrada de una red neuronal. Consiste en ajustar los valores de dichos datos a un mismo rango, para que la red pueda interpretarlos correctamente.

En las redes convolucionales, también es habitual realizar este proceso cada vez que se realiza una convolución, ya sea antes, o después, de la misma, como vimos en la Sección 2.2.3. Algunas de las ventajas asociadas al uso de la normalización en las diferentes capas de la red son las siguientes [14]:

- Evita que las características que cuentan con valores más altos, tomen un mayor protagonismo sobre otras con valores más bajos, pero que pueden ser igual o más determinantes.
- Reduce el *cambio de covarianza interna*. Este concepto hace referencia a los cambios que se producen en la distribución de las activaciones de la red, como consecuencia de la modificación de los parámetros de la misma a lo largo del entrenamiento.
- Provoca que la optimización de la red se dé más rápido, ya que los valores de los pesos se encuentran restringidos a un determinado rango.
- Ayuda a la red en la regularización, aunque no de forma significativa.

Matemáticamente, la normalización en las capas de una red neuronal se puede expresar como:

$$y = \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}} * \gamma + \beta \qquad (2.12)$$

donde:

- *x*, representa los valores a normalizar. En el caso del procesamiento de imágenes se corresponde con un tensor de 4 dimensiones *(M,C,H,W)*, que son: M el tamaño del lote, C el número de canales, y H, W las dos dimensiones de las imágenes.
- *E(x)*, la esperanza de los datos.

- *Var(x)*, la varianza de los datos. Se calcula mediante es estimador del *bias*.
- ϵ, valor que aporta estabilidad numérica.
- γ y β, parámetros de aprendizaje de la red. Son vectores de tamaño C.

En las redes convolucionales es habitual que se procesen varias imágenes a la vez, formando un **batch** o lote. Retomando la nomenclatura anterior, el batch se corresponde con el tensor de 4 dimensiones al que denominamos x y cuya primera dimensión, M, sería igual al tamaño del lote, es decir, al número de imágenes que está procesando la red en una iteración.

La formulación de la normalización (ver Ecuacion 2.12, permite varias interpretaciones en función de cómo se quiera emplear el lote.

Batch Normalization [15] es el nombre que recibe la idea de realizar la normalización considerando todo el lote como una única unidad. De esta forma, se emplean los datos de todos los canales, de todos los componentes del lote, para calcular los parámetros necesarios y aplicar la misma normalización a todos ellos. En términos matemáticos, esto se traduce al cálculo de una única esperanza y varianza para llevar a cabo la normalización.

Por el contrario, **Instance Normalization** [16] realiza una normalización diferente para cada imagen del lote. Para cada una de ellas, se calcula una esperanza y varianza utilizando únicamente los canales de dicha imagen, lo que hace que el proceso de normalización sea distinto para cada una de ellas. En total, si el lote cuenta con T imágenes, se calculan T esperanzas y T varianzas, que sirven para llevar a cabo T normalizaciones.

Existen otros tipos de normalización [14], como **Group Normalization**, que se ejecutan sobre distintos grupos de canales, o **Layer Normalization**. Así pues, realiza una normalización de la entrada a través de sus características. Incluso, se puede optar por normalizar los pesos de la red, en vez de la entrada como propone la **Weight Normalization**.

2.4.2. Funciones de activación

Son una parte fundamental de las redes neuronales, ya que se encargan de generar la salida de una neurona en función de la entrada recibida. Se pueden diferenciar en dos grandes grupos: *funciones de activación lineales* y *funciones de activación no lineales*.

Las lineales presentan dos problemas principales:

1. No permiten la retropropagación, ya que su derivada es una constante y por tanto no guarda ningún tipo de relación con la entrada.

2. Provoca un comportamiento lineal de la red. Por ejemplo, si se emplea la función identidad, la red se comportará como una regresión simple.

Para solventar estos problemas, se utilizan las funciones no lineales. Para las tareas de multi-clasificación con modelos de regresión logística, se suele emplear la función *softmax*. En el caso de clasificaciones binarias, es habitual el uso de la función *sigmoide* o la *tangente hiperbólica*.

La función *softmax* [17] devuelve, para una entrada, su probabilidad de pertenencia a cada grupo existente. Por tanto, la suma de todos los valores devueltos por esta función suman 1. Sea la entrada z, compuesta por i elementos y K, el número de posibles clases. Su formulación matemática es la mostrada en la Ecuación 2.13.

$$softmax(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2.13}$$

Por su parte, la función **sigmoide** [17] (ver Ecuacion 2.14) devuelve valores normalizados entre cero y uno, pero que no representan probabilidades de pertenencia.

$$y = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1} \tag{2.14}$$

Entre los inconvenientes del uso de esta función destacan los siguientes:

- Coste computacional elevado.
- La salida no está centrada en el cero, lo que provoca que las actualizaciones de los gradientes vayan demasiado lejos en diferentes direcciones. Esto lleva a una optimización más costosa.
- El aprendizaje de la red es muy lento.

La función de la **tangente hiperbólica** [17] (ver Ecuación 2.15) presenta su salida acotada en el intervalo $[-1, 1]$. Es considerada como un escalamiento de la función sigmoide por lo que, a pesar de tener una salida centrada en el 0, sigue siendo problemática en cuanto al gradiente.

$$y = \frac{2}{1+e^{-2x}} - 1 \tag{2.15}$$

En la actualidad existe otra función de activación que es usada con más frecuencia que estas dos. Esta función es la **ReLU**, *Rectificador Lineal Unitario*, y es la que se empleó en la propuesta de la arquitectura UNET [9]. Su formulación matemática es la siguiente.

$$f(x) \;=\; max(0, x) \tag{2.16}$$

La función $ReLU$ [17] imita el comportamiento de la función identidad para valores de entrada positivos, y, para los valores negativos, devuelve el valor cero. Gráficamente podemos ver su comportamiento en la Figura 2.14.

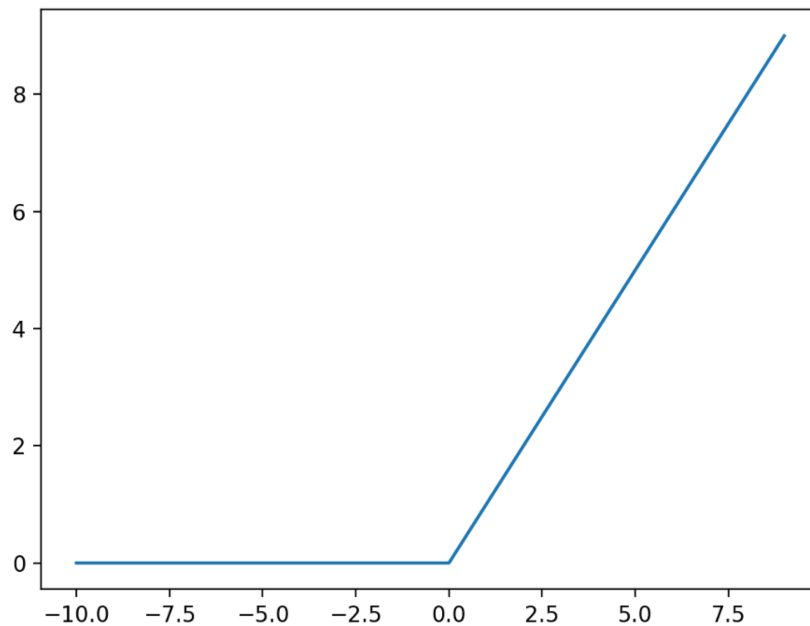


Figura 2.14: Gráfica de la función de activación $ReLU$.

Las principal ventaja de esta función es su bajo coste computacional, lo que permite que la red aprenda y converja rápidamente. Como inconveniente, provoca la aparición de neuronas *zombies*, que son aquellas que no pueden llevar a cabo el aprendizaje. Este problema surge al considerar una salida igual a cero para los valores negativos de entrada, porque provoca que el gradiente correspondiente a las respectivas neuronas también sea cero, imposibilitando cualquier atisbo de aprendizaje en las mismas.

La función **Leaky ReLU** ([17]), ver Ecuación 2.17, propone una modificación de la

función ReLU para evitar la aparición de neuronas zombies.

$$f(x) = \begin{cases} x \quad si\ x > 0 \\ \\ x \cdot \alpha \quad si\ x \leq 0 \end{cases} \tag{2.17}$$

Como puede verse en la Figura 2.15, los valores negativos presentan una pequeña pendiente que permite llevar a cabo la retropropagación del error a las neuronas correspondientes. Sin embargo, las predicciones realizadas sobre estos valores no son consistentes.

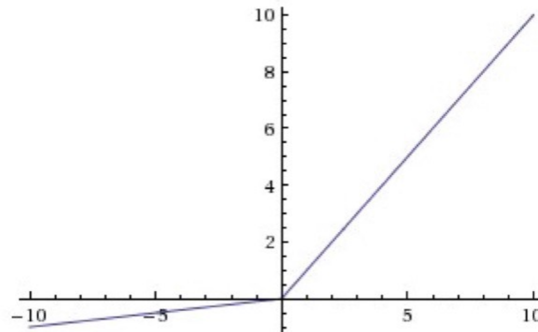


Figura 2.15: Gráfica de la función de activación *Leaky ReLU.*

Podemos encontrar muchas otras funciones basadas en modificaciones sobre la función ReLU, como pueden ser **PReLU**, **ELU** o **Threshold ReLU Function**, pero no van a ser consideradas en este trabajo [17].

2.4.3. Funciones de pérdida

En las redes neuronales, se utiliza la función de pérdida para evaluar la calidad de la salida obtenida, por medio de una comparación respecto a la salida deseada. El grado de similitud entre la salida obtenida y la real determina el proceso de aprendizaje de la red, por lo que la elección de la función de pérdida es un paso crítico.

Para establecer comparaciones entre imágenes, existen varias funciones que pueden ser utilizadas. A continuación se proponen tres de ellas.

- **MSE**:
  El MSE, o *error cuadrático medio*, es una de las funciones de pérdidas más utilizadas. Sea x la salida obtenida por la red e y la salida deseada, ambos tensores de un tamaño n, entonces:

$$MSE(x, y) = \frac{1}{n} \sum_{n} (x_n - y_n)^2 \tag{2.18}$$

- **MAE**:
  También conocida como *error absoluto medio* establece una comparación basada en la diferencia absoluta entre la salida obtenida y la deseada. Matemáticamente, la formulación es la siguiente:

$$MAE(x, y) = \frac{1}{n} \sum_{n} |x_n - y_n| \tag{2.19}$$

  siendo, al igual que antes, x la salida obtenida e y la deseada, ambos tensores de un mismo tamaño n.

- **SmoothL1**:
  Está inspirada en la *función de pérdida de Huber*, que emplea una aproximación a la métrica MSE o MAE, en función del elemento que se esté comparando [18]. Sea x la salida obtenida por la red e y la salida deseada, ambos tensores de tamaño n. La comparación entre x_j e y_j viene determinada por:

$$l_j = \begin{cases} 0{,}5 \cdot (x_n - y_n)^2, & si\ |x_n - y_n| < \beta \\ \\ |x_n - y_n| - 0{,}5 \cdot \beta, & en\ caso\ contrario \end{cases} \tag{2.20}$$

  Realizando la comparación de cada par de elementos, como en la Ecuación 2.20, se puede definir la diferencia entre x e y como:

$$SmoothL1 = \frac{1}{n} \sum_{n} l_n \tag{2.21}$$

  Esta métrica es menos sensible a los outliers que MSE, gracias a que sólo se utiliza el error cuadrático medio, cuando el MAE correspondiente es menor que un cierto umbral, denominado β. Esto evita que se eleven al cuadrado términos que de por sí son demasiado grandes reduciendo, de esta forma, el valor total de la función $SmoothL1$. Asimismo, al evitar la aparición de términos demasiado elevados, en ocasiones, también se consigue evitar el problema de la *explosión del gradiente.*

Capítulo 3

Gestión del proyecto

La gestión de un proyecto software no presenta grandes diferencias respecto a la de cualquier otro proyecto de Ingeniería, pero las características intrínsecas al software como la invisibilidad, la complejidad y la flexibilidad aportan un mayor grado de dificultad en la gestión de los mismos. Por ello, realizar una correcta planificación del proyecto es fundamental para alcanzar con éxito las metas fijadas.

3.1. Metodología de trabajo

Consiste en un conjunto de técnicas y de organización, que se aplican con el objetivo de que un equipo de trabajo desarrolle el mejor producto posible. En la actualidad, dentro del desarrollo software, existen dos tipos de metodologías bien diferenciadas:

- Metodologías tradicionales: el desarrollo del producto final se realiza a través de un proceso secuencial en una única dirección, haciendo del mismo un proceso predecible y, en consecuencia, eficiente. La elicitación de requisitos es la fase más importante del proceso en estas metodologías ya que, una vez fijados, no pueden ser modificados, por ello suele destinarse una gran cantidad de recursos a su identificación.

- Metodologías ágiles: definen un proceso de desarrollo flexible, capaz de adaptarse a las distintas necesidades que puedan ir surgiendo. Se cuentan con plazos de entregas reducidos para evitar la dispersión y centrar toda la atención en una tarea.

Desde la publicación en el año 2001 del *Manifesto Agile* [19] las metodologías ágiles han ido ganando en popularidad e importancia. En la actualidad, su uso es mucho más frecuente que el de las metodologías tradicionales.

Para la realización de este TFG, no se ha seguido de forma estricta ninguna de estos dos tipos de metodologías. Dado que se realiza en colaboración con la empresa *HP*, se ha trabajado en base a *sprints* típicos de la metodología ágil *scrum*, pero sin un número de

reuniones entre los miembros tan elevadas. En nuestro caso, se realizaban reuniones cada dos semanas para llevar a cabo un correcto seguimiento de la evolución del proyecto y fijar nuevos objetivos en caso de que fuese necesario. Desde el comienzo del proyecto, se fijaron una serie de épicas en unos ciertos intervalos de tiempo, lo que ha ayudado llevar una mejor organización de las tareas realizadas y del tiempo disponible para ello.

3.2. Épicas del proyecto

Una épica, en metodología ágil, se define como *úna historia de usuario que no puede ser entregrada tal y como se ha definido dentro de una sola iteración, o que es suficientemente grande como para ser partida en historias de usuario más pequeñas'*. Para el desarrollo de este trabajo se definieron las siguientes:

- Acondicionamiento de los datos y adaptación del esqueleto software: en este hito se recogen las tareas de encontrar un conjunto de datos para el desarrollo del trabajo, su acondicionamiento a las características del proyecto, así como la adaptación y modificación del esqueleto software proporcionado por HP para conseguir los objetivos marcados.

- Evaluación de diferentes modelos de redes neuronales: se centra en el estudio de diferentes estructuras de redes neuronales e implementaciones, valorando los resultados obtenidos con ellas.

- Composición de la salida: elección del formato de salida deseado y creación de las herramientas necesarias para mostrar la salida de la red bajo el formato elegido.

- Traslado del software a un docker.

3.3. Planificación

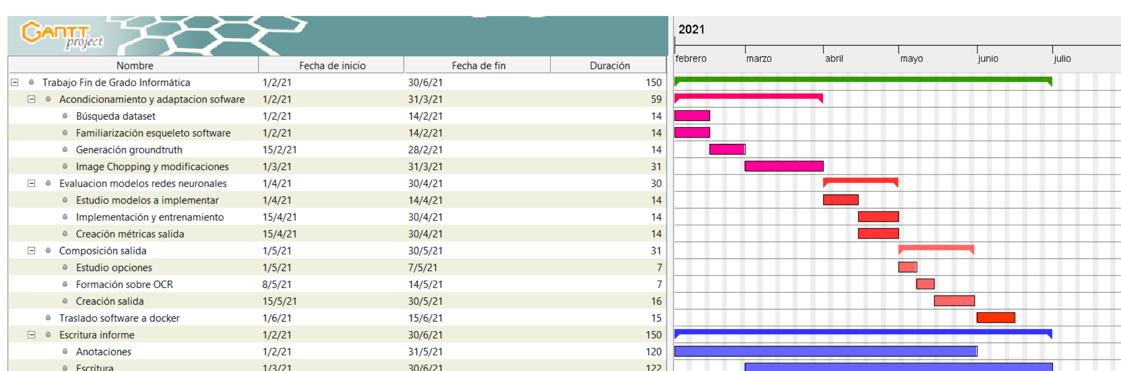
El presente trabajo representa mi *Trabajo Final de Grado de Ingeniería Informática, mención en Computación* y supone una carga de 12 ECTS en dicha formación. Teniendo en cuenta que 1 ECTS supone la inversión de 25 horas de trabajo, en total este TFG está pensado para ser realizado en 300 horas.

A pesar de que los contactos con *HP* comenzaron en Octubre de 2020, el proyecto comenzó el día 1 de Febrero de 2021 y su finalización estuvo prevista para el 30 de junio del mismo año.

En las siguientes líneas, se presenta la planificación inicial realizada, así como las variaciones que esta sufrió.

3.3.1. Planificación inicial

La primera tarea en el desarrollo de este trabajo fue realizar la planificación del proyecto. Para ello, se desglosaron, en diferentes tareas, las distintas épicas mencionadas con anterioridad, asignando un tiempo estimado de duración a cada una de ellas. La planificación inicial realizada fue la siguiente:



Nombre	Fecha de inicio	Fecha de fin	Duración
Trabajo Fin de Grado Informática	1/2/21	30/6/21	150
Acondicionamiento y adaptacion sofware	1/2/21	31/3/21	59
Búsqueda dataset	1/2/21	14/2/21	14
Familiarización esqueleto software	1/2/21	14/2/21	14
Generación groundtruth	15/2/21	28/2/21	14
Image Chopping y modificaciones	1/3/21	31/3/21	31
Evaluacion modelos redes neuronales	1/4/21	30/4/21	30
Estudio modelos a implementar	1/4/21	14/4/21	14
Implementación y entrenamiento	15/4/21	30/4/21	14
Creación métricas salida	15/4/21	30/4/21	14
Composición salida	1/5/21	30/5/21	31
Estudio opciones	1/5/21	7/5/21	7
Formación sobre OCR	8/5/21	14/5/21	7
Creación salida	15/5/21	30/5/21	16
Traslado software a docker	1/6/21	15/6/21	15
Escritura informe	1/2/21	30/6/21	150
Anotaciones	1/2/21	31/5/21	120
Escritura	1/3/21	30/6/21	122

Figura 3.1: Planificación inicial del proyecto.

Tal y como podemos observar en el diagrama de Gantt de la Figura 3.1, la épica de mayor duración es la escritura del informe, ya que se planteó su realización de manera continuada, según se iba avanzando en el transcurso del proyecto. Sin considerar esa épica, el acondicionamiento de los datos y la adaptación del esqueleto software es el hito de mayor duración extendiéndose durante dos meses. Se trata de la parte más delicada del proyecto, por lo que era importante disponer del tiempo necesario para su correcta realización. El resto de épicas tienen una duración aproximada de dos a cuatro semanas.

Dentro de esta planificación no se contemplan las reuniones mantenidas con los tutores de *HP*, que tuvieron lugar de manera periódica cada catorce días. Estas reuniones no figuran en el diagrama de la Figura 3.1, pero sí que fueron consideradas para la organización temporal de las tareas y por ello, sus plazos de realización son múltiplos de siete o catorce días.

3.3.2. Variaciones respecto a la planificación inicial

A pesar de haber realizado una planificación inicial lo más realista posible, existieron una serie de condicionantes durante el transcurso del proyecto que hicieron variar los plazos iniciales.

Más allá de pequeñas desviaciones respecto de las fechas planificadas de ciertas tareas, y que fueron compensadas con otras realizadas en un menor tiempo del estimado, dos son los principales problemas acontecidos durante el desarrollo del proyecto:

- La implementación del algoritmo de *image chopping*, así como las distintas modificaciones que hubo que realizar en la aplicación para el correcto funcionamiento de la misma. Esto hizo que la primera épica se extendiese 10 días más de lo esperado.
- El entrenamiento de la red neuronal fue más complicado de lo esperado. Por una parte, se tuvo que buscar un entorno adecuado donde realizar dicho entrenamiento, ya que ni mi máquina personal, ni la máquina virtual prestada por el Departamento de Informática, disponían de las prestaciones suficientes para ello. Tras probar los servicios de *Google Collab* y *Kaggle*, con la consiguiente adaptación la aplicación a cada una de ellas, se terminó tomando la decisión de realizar el entrenamiento en la máquina prestada por un particular, dotada de más prestaciones de computo que las referidas anteriormente. Por otra parte, el proceso de entrenamiento llevó mucho más tiempo del que inicialmente se creía, por lo que finalizó a mediados-finales del mes de mayo, en vez de a finales de abril.

Aunque la fase de entrenamiento finalizó unas tres semanas más tarde de lo previsto no tuvo un gran impacto en la planificación del resto de las tareas, ya que se trabajó en la siguiente épica de manera paralela al entrenamiento y se consiguió retomar el ritmo de trabajo pensado inicialmente. Sí tuvo repercusión en el contenido del trabajo, teniendo que renunciar a la posibilidad de entrenar diferentes variaciones del modelo de red escogido y comparar sus comportamientos, así como descartar la idea inicial de trasladar el software a un docker y, en su lugar, crear un servidor.

Por último, cabe mencionar que la redacción de este documento se realizó el último mes de trabajo y no durante los tres meses previos, tal y como se había planificado.

3.4. Gestión de la configuración

Para llevar a cabo una evolución controlada del proyecto y poder mantener una retroalimentación adecuada, entre los diferentes actores involucrados en este trabajo, sobre los avances realizados, se ha empleado la herramienta *GitLab* [20].

El servicio web GitLab, basado en Git, ha permitido llevar un control automático de versiones, con las ventajas que conlleva, así como hacer un seguimiento más certero de las distintas tareas realizadas y en curso, mediante su gestor de hitos y tareas. A continuación, se muestra la organización del repositorio de GitLab:

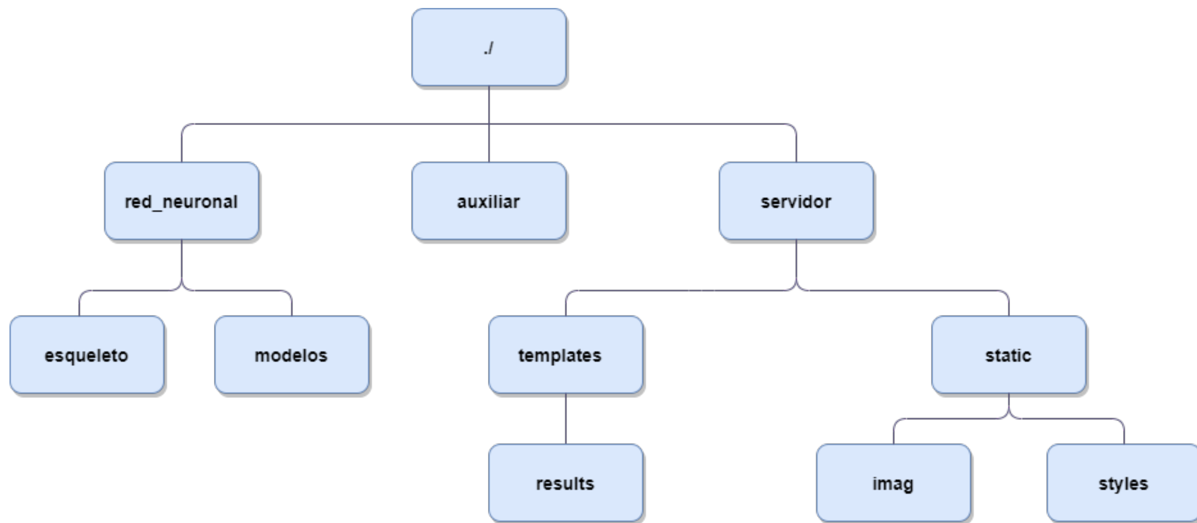


Figura 3.2: Organización del repositorio.

El contenido de cada una de las ramas mostradas en la Figura 3.2 es:

- *auxiliar*: contiene los ficheros *.py* utilizados para explorar la información de las imágenes utilizadas, así como para generar el Ground Truth de cada una de ellas.
- *red_neuronal*: en la carpeta *esqueleto*, se contienen los ficheros *.py* que conforman la aplicación de la red neuronal. En la carpeta *modelos*, se albergan los ficheros *.pt* que representan las configuraciones de la red con mejores resultados en entrenamiento y validación.
- *servidor*: almacena los ficheros necesarios para construir el servidor implementado para interaccionar con la red y generar la salida deseada. En su carpeta *templates*, se tienen los archivos *HTML* necesarios para la configuración del servidor y los resultados obtenidos del procesamiento de una imagen *(en results)*. En *static*, se almacenan los ficheros de estilo de los HTML generados (carpeta *styles*) y las imágenes a mostrar en cada documento (carpeta *imag*).

No se contempla, en la estructura mostrada en la Figura 3.2, la presencia del conjunto de datos utilizado. Esto se debe a que no ha sido subido a GitLab, ya que su peso es

superior a 11GB. La organización de los directorios en la máquina utilizada es la misma que la mostrada, a excepción de una nueva rama que contiene los datos.

Finalmente, para la documentación del proyecto se ha utilizado el editor de texto *TexStudio* que permite la generación de documentos bajo el sistema de composición de textos LaTeX.

Capítulo 4

Plataforma de trabajo e implementación

En primer lugar, se realiza una exposición del conjunto de datos, sobre el que se ha trabajado, definiendo tanto las características del mismo, como los objetivos fijados. En segundo lugar, se explican los procesos de tratamiento a los que han sido sometidos los datos, para adecuarlos a la realidad del problema. En tercer, y último lugar, se justifica el uso de *pytorch* frente a *keras* para alcanzar los objetivos. Se detalla la configuración de la aplicación desarrollada, así como las características de la red implementada.

4.1. Conjunto de datos

El conjunto de datos empleado en este trabajo fue elaborado por la *Universidad de Salford* y recibe el nombre de *Layout Analysis Dataset* [1] (en adelante LAD). Sus autores lo definen como un *"conjunto realista de documentos contemporáneos"* y está disponible para su uso bajo petición a la Universidad. Cuenta con una página web [21], en la que es posible explorar su contenido.

Fue creado para su utilización en reconocimiento de *layouts* de documentos. Dicha tarea consiste en reconocer y categorizar las distintas zonas de un documento, en función del contenido de las mismas. Aunque ya existían múltiples conjuntos de datos elaborados para estos fines, los autores de LAD consideraban que no eran lo suficientemente óptimos, ya que ninguno cumplía con los tres requisitos que, a su juicio, deberían reunir:

- *Realismo*: ser representativo de los documentos reales que probablemente se escaneen en situaciones cotidianas.

- *Completitud*: contener información detallada que permita una evaluación en profundidad.

- *Estructuración flexible*: facilidad para buscar y seleccionar subconjuntos que cumplan ciertas condiciones específicas.

4.1.1. Descripción

Para generar un conjunto de datos que cumpliese con el primero de los requisitos descritos, contaron con un grupo de expertos, que los ayudó a realizar una selección lo más próxima a la realidad posible. La mayor parte de los documentos, que conforman el conjunto, se pueden dividir en dos temáticas: revistas o periódicos, y, artículos científicos o técnicos. Los expertos estimaron la frecuencia de cada uno de estos grupos en la vida cotidiana y propusieron una proporción 7 *a* 1 a favor de las revistas y periódicos. En menor medida, existen otro tipo de documentos, que también son incluidos en el dataset, como pueden ser *formularios*, *extractos bancarios* y *anuncios*.

En total, el número de instancias contenidas en el dataset proporcionado es de 478, siendo, cada una de ellas, una página escaneada de un documento que se presenta bajo el formato de imagen *.tif*. Las imágenes están en alta resolución y tienen un gran tamaño, siendo las dimensiones medias de 3179 pixels de alto y 2377 de largo.

In the Arena
Joe Klein
When Bad Missions Happen to Good Generals

Four Tech Waves To Watch
UTILITY COMPUTING
JUST TURN ON THE DATA
The idea is to make computing power into another pay-as-you-go service—like water or electricity. But beware of the hype

Strange customs
Exclusively written for Holland Herald by John S. Burnett
The old man in the boat slammed back the cocking lever on his assault rifle and pointed it first at me and then at my girlfriend. I figured I had better do what he said. He barked orders and the four others in the small boat below raised their guns. "You come! Come now!" he said in broken English.

Figura 4.1: Mosaico de cuatro documentos pertenecientes al conjunto de datos.

La estructura, o layout, de cada uno de estos documentos se presenta en un archivo *XML* bajo el mismo nombre que la imagen correspondiente. Cada una de las zonas del documento tiene asignado un tipo, que indica su naturaleza. Algunas de estas categorías son: *texto*, *línea*, *gráficos*, *tablas*, *fórmulas matemáticas*... También tiene asignadas una serie de coordenadas que permiten cercar el área correspondiente, utilizando polígonos irregulares, que aportan un mayor grado de precisión que formas más simples, como pueden ser los rectángulos empleados en otros datasets. Adicionalmente, en función del tipo de zona de la que se trate, se cuenta con información extra que, en muchos casos, puede ser de utilidad. Por ejemplo, si la zona se corresponde con un texto, se tiene información sobre el lenguaje en el que está escrito, el tamaño y color de la fuente, la dirección de lectura, etcétera.

JOY RIDE

BY SUE ZESIGER CALLAWAY

BMW's Cheap Thrill

With the 1 Series, the automaker achieves outsized performance from the lowest-priced BMW on the market.

I HAVE COMMITTED a Sue sin the size of Texas: I have knowingly requested, driven, and (wo)manhandled the least expensive, least horsepowered BMW in the U.S. today, the 128i. And I loved every minute of it.

The 1 Series, as it's called, is a decades-overdue successor to the beloved 2002 model, which arrived in the U.S. in 1968. Pricewise, it slots neatly between BMW's Mini lineup and the 3 Series. The production versions, the 128i and the 135i, which come in coupe and convertible iterations, just arrived in U.S. dealerships.

Before I drove the 128i, I couldn't be sure from the outside whether BMW had simply succeeded in making a solid entry-level car or a serious if diminutive instrument of performance. Other BMWs have a menacing face—the front fender lines swoop down into the hood, creating an angrily furrowed brow over the headlights. Chris Bangle, the car's ultimate Design Daddy, likens the 1 Series' face to that of an ingénue, not unlike the "car-icatures" in Pixar's *Cars*. I'd agree to an extent: My convertible's face certainly looked wide-eyed, but I'd call it more of an unblinking stare that signaled, I hoped, less Bambi and more hungry wolf.

I tested my 128i soft-top in one of America's toughest road-quality war zones: Manhattan, with its potholes, crabby cabbies, and impertinent pedestrians. Dropping into the pilot's seat, I was immediately pleased to see a handsome dash, nice wood trim, a precision shifter, and all the familiar BMW high-tech black switches and gauges. Even the faux-leather seats felt pleasant to the touch and well bolstered (real leather is also available).

I set off after dark, the cool LED rear lights glowing like afterburners. A few laps around Central Park (a perennially favorite track of mine) was enough to get it: The 128i is 100% BMW, which means nimble, playful, and competent. Even with a languorous zero to 60 of 6.4 seconds in my non-turbocharged version, the car gave me good road feel, sporty yet comfortable suspension, very active steering (even without the optional Active Steering), smooth acceleration, great brakes. I headed down Broadway and up Madison, hunting for open stretches. I grinned as I popped off the line at every light. The next day I awoke early and immediately went out again for another spin with the top dropped. Daylight was no less fun: I felt like a teenager, power-sliding around corners, awakening the traction control, blaring the iPod, and grinning some more—sure signs of my gut-level automotive approval. Granted, it's no full-grown wolf like the M3, but it's one heck of a wolf pup.

Later I regained possession of my adult self and tucked my two young children into the back seats. They had room to spare, but I suspect full-sized passengers might not be as content on a long trip.

BMW has done something smart: Instead of shrinking and "decontenting" a 3 Series to create the 1, the company has loaded its new littlest one with a full spectrum of Bavarian delights, from the multifunction leather sport steering wheel and extensive aluminum componentry to enough performance-enhancing computer systems to run a small corporation. It may sound obvious to retain a bevy of features as you move down the product range instead of stripping them away, but that's a strategy too few car manufacturers employ today.

BMW 128i CONVERTIBLE
PRICE
From $31.100
ENGINE
3-liter in-line 6-cylinder
PERFORMANCE
230 hp, 28 mpg/hwy
SPEED
0 to 60 in 6.4 seconds

FEEDBACK *fortunejoyride@hotmail.com*

26 FORTUNE *April 14, 2008*

Figura 4.2: Ejemplo de una imagen del dataset junto a su groundtruth.

En la Figura 4.1.1, se muestra un ejemplo de los distintos campos que se pueden reconocer dentro de una imagen, gracias a su correspondiente fichero XML. Como se puede observar, no existe ni un solo campo que no esté reconocido y delimitado, ya que hasta las líneas están identificadas. Las zonas están perfectamente ajustadas a las dimensiones del contenido, incluso en el caso de las figuras más complejas como el automóvil.

4.1.2. Objetivos

Como se mencionó en el Capítulo 1, el objetivo de este TFG era desarrollar un modelo de segmentación de imágenes, que permitiese reconocer las diversas zonas de un documento, para posteriormente poder recomponerlo en un formato distinto.

La variedad de información, que puede encontrarse en documentos de diferentes ámbitos, es muy amplia, por lo que existen múltiples categorías que pueden ser reconocidas en un dataset, como con el que vamos a trabajar. Por ello, nuestro modelo de segmentación va a restringirse a un pequeño grupo de estas categorías, ya que intentar abarcarlas todas implicaría un esfuerzo computacional demasiado elevado.

Se ha decidido escoger aquellas con mayor presencia en el dataset, para así obtener los mejores resultados posibles. Para ello, se realizó un recuento de las distintas regiones que se contemplan en el conjunto de datos. Los resultados obtenidos se muestran en el Cuadro 4.1.

Categoría	**Nº de apariciones**
Texto	8293
Separadores	937
Imágenes	583
Gráficos	374
Ruido	93
Tablas	48
Cuadro	42
Form. Matemática	42
Marcos	15
Líneas	14
Total	10441

Cuadro 4.1: Número de apariciones de cada categoría en el dataset.

Como era de esperar, prácticamente todas las regiones detectadas se corresponden con texto, siendo el porcentaje sobre el total del 79.45 %. Dentro de la categoría texto podemos encontrar múltiples sub-categorías, por lo que no van a considerarse todas ya que, como vemos en el Cuadro 4.2, muchas de ellas apenas tienen presencia.

Sub-categoría	**Nº de apariciones**
Párrafos	5188
Títulos	945
Pies de página	467
Nº de página	418
Pie de foto	390
Créditos	280
Texto flotante	276
Letra mayúscula	176
Encabezado de página	153
Total	8293

Cuadro 4.2: Número de apariciones de cada sub-categoría del tipo Texto.

Tras hablar con los tutores de *HP*, se acordó elegir entre cinco y ocho categorías distintas. En vista a los resultados mostrados en los Cuadros 4.1 y 4.2, finalmente se optó por un total de siete, que son: *párrafos*, *títulos*, *imágenes*, *pies de página*, *nº de página*, *pie de foto* y *texto flotante*.

La categoría de texto flotante es, de las escogidas, la que tiene un nombre menos descriptivo. Hace referencia a textos que no terminan de encajarse a la definición de párrafo pero podrían serlo perfectamente. De hecho, en ocasiones, estos textos son parte de un párrafo que comienza en una columna y acaba en otra distinta, siendo el texto de la nueva columna la que se considera como texto flotante. Otras veces, hace referencia a textos entre párrafos que se salen del formato, para dar una mayor notoriedad a la idea de la que se está hablando. En definitiva, se considera como texto flotante, a todo aquel que no termina de encajar en la categoría de párrafo, pero si se tuviese que encasillar en alguna categoría, sería en ella.

Existen dos clases que, a pesar de estar entre las siete con mayor frecuencia, no han sido escogidas. Para la introducción de la categoría texto flotante, se ha tenido que prescindir de la categoría *créditos*. El texto etiquetado como "créditos" es todo aquel que aporta información, al lector, sobre la autoría del propio documento o de alguno de los contenidos del mismo. La relevancia, que puede llegar a tener el texto etiquetado como flotante es mucho mayor para el lector, que el que puede tener los créditos por lo que, a pesar de tener una menor frecuencia, se ha elegido la categoría flotante en detrimento de los créditos.

La clase *separadores*, aunque es la segunda con mayor número de apariciones, no ha sido considerada, ya que carece de interés práctico. Se tratan de elementos gráficos empleados para remarcar una separación entre distintas zonas (ver Figura 4.1.2). Para el lector del documento, estos elementos no aportan información alguna, por lo que se ha optado por no considerarlos, en favor de otros grupos más interesantes.

WOMEN IN LUXURY

Aisles of Style

She put her indelible stamp on the bridal-dress industry, then the red carpet. Now Vera Wang is set to revolutionize mass-market ready-to-wear with her new line for Kohl's

By Kristina Zimbalist

VITAL STATISTICS:

NAME VERA WANG

CURRENT JOB CHAIRMAN AND CEO, VERA WANG

FIRST JOB YSL BOUTIQUE SALES ASSOCIATE

INSIDE TRACK SPENT 16 YEARS AT *VOGUE* BEFORE WORKING FOR RALPH LAUREN

CLAIM TO FAME ONCE TRAINED TO BE AN OLYMPIC FIGURE SKATER

VERA WANG HAS HAD A STORYBOOK CAREER, building a thriving bridal, fashion and housewares empire, but to hear her tell it, she would just as soon have eloped. "I had always wanted to be a designer, and my father said, 'How do you know you have what it takes to be in fashion?' And, boy, was I determined. And I showed him. And I've lived to regret it," she says, and then tops the statement off with a gleeful laugh. Wang is kidding, of course. Her sharp, native–New Yorker wit and sarcasm don't stay hidden under the hem for long. "I say things like this, and people take it and say, 'Vera hates fashion.' And I'm trying to be funny, because if you don't laugh, you're going to keel over."

Wang is hyperintelligent, hilariously funny, sensitive—and self-deprecating. Having started out as a fashion editor at *Vogue* among some pretty big personalities, she has every reason to have adopted some strain of diva behavior. "I am not a diva," she says. And she's not. Born to traditional-minded Asian parents and raised with what she jokingly calls "good Midwestern values," Wang says, she's "a worker." Her work ethic—few could argue—has paid off.

What began nearly two decades ago as a tiny shop bearing her name on Madison Avenue has become Vera Wang the corporation. Her name is synonymous with fashionable brides' and bridesmaids' dresses. Her evening gowns are red-carpet staples. After an early venture into ready-to-wear proved a critical and commercial disappointment in 2001, she made a risky second dive in with great success—she was named Womenswear Designer of the Year by the CFDA in 2005. She has since launched a younger, secondary line, Lavender, and this fall teams up with Kohl's to launch a mass-market minicollection, Simply Vera. She has two fragrances and a line of diamonds and sells housewares, linens and china under her moniker. This list goes on and on. "It has been a labor of love," Wang says of her creations and career, if not always a skip down the aisle.

"A lot of my life has been predicated on not getting what I hoped I would get," Wang confesses. She spent the first third of her life training to be an Olympic figure skater. Every waking hour was devoted to the sport; she studied classical ballet at George Balanchine's School of American Ballet. When she didn't make the Olympic team, "I had to learn a big life lesson," she says, "which is that when you are so obsessed by something and you can no longer do it, you dust yourself off and keep going."

Her next passion was fashion, and it led her from summer jobs at the Yves Saint Laurent boutique in New York City to 16 years as a fashion editor at *Vogue*, where she did covers and influential shoots with photographers like Richard Avedon and Deborah Turbeville. There too she eyed the top prize. "I don't think you stay for 16 years and not want the ultimate say," Wang explains. When she realized she wasn't going to get it, that she would never be editor in chief, "I had to try to recover," she says.

She hopped over to Ralph Lauren as design director of women's accessories. There, "much as I loved sittings," Wang says, "creating made me feel so good. And it was just genius fun."

ERIKA MCCONNELL

54 TIME STYLE & DESIGN FALL 2007

Figura 4.3: Ejemplo de elementos pertenecientes a la categoría separadores.

Entre todas las categorías escogidas, se contemplan el 79.18 % de las distintas zonas de los documentos, correspondiéndose el 93 % diferentes sub-categorías del tipo texto y, tan sólo el 7 % restante, a imágenes. A modo de resumen, en el Cuadro 4.3, se muestran las clases escogidas y su relevancia dentro del conjunto de datos.

Categoría	**Total**	**% del considerado**	**% del total**
Párrafos	5188	62.76	49.69
Títulos	945	11.43	9.05
Imágenes	583	7.05	5.59
Pies de página	467	5.65	4.47
Nº de página	418	5.06	4.00
Pie de foto	390	4.72	3.74
Texto flotante	276	3.33	2.64
Total	8267	100	79.18

Cuadro 4.3: Resumen de las categorías de interés.

4.2. Tratamiento de los datos

Con el objetivo de poder utilizar el conjunto de datos en el entrenamiento de la red neuronal, es necesario realizar sobre él dos tratamientos diferentes. El primero de ellos es la generación del Ground Truth, que se realiza en frío, es decir, antes de comenzar el entrenamiento; el segundo de ellos, es someter las imágenes a un proceso de chopping que, se realiza en caliente, es decir, durante la ejecución.

4.2.1. Generación del Ground Truth

El Ground Truth es un término que hace referencia a la variable respuesta de un conjunto de datos. Su mayor pecularidad es que no tiene por qué ser exacta, sino que en la mayoría de las ocasiones está sujeta a la interpretación de los desarrolladores del conjunto de datos. Por ejemplo, en un dataset de imágenes de perros el Ground Truth contendrá las etiquetas de “peligroso” o “no peligroso”, en función de lo que determinen las personas responsables.

En nuestro problema, el Ground Truth no se restringe a catalogar una imagen como perteneciente a un grupo o a otro, sino que va un paso más allá: se realiza una clasificación pixel a pixel. Para cada categoría, de la que se tiene constancia, se puede definir su propio Ground Truth como una clasificación de las zonas del documento, en función de su pertenencia a dicha categoría. De esta forma, el Ground Truth para una determinada clase, es una clasificación binaria de sus pixels, tomando el valor 1 aquellos que se corresponden con ella y, 0 en caso contrario.

Como dijimos en la Sección 4.1.1, queremos ser capaces de reconocer hasta siete categorías distintas. Esto quiere decir que, para cada muestra del conjunto de datos, se van a generar siete imágenes diferentes de idénticas dimensiones a la original, en las que las zonas correspondientes a cada clase se marcan con un color blanco y, el resto, en color negro. De esta forma, si se superponen las imágenes generadas, se obtiene una única, en la que sólo las zonas de las categorías de interés están remarcadas.

Gracias a que para cada documento tenemos un fichero *XML*, en el que se indican el tipo de todas sus zonas y las coordenadas que abarcan, es posible llevar a cabo este proceso de segmentación del Ground Truth por categorías. La secuenciación de pasos seguida para generar el Ground Truth es:

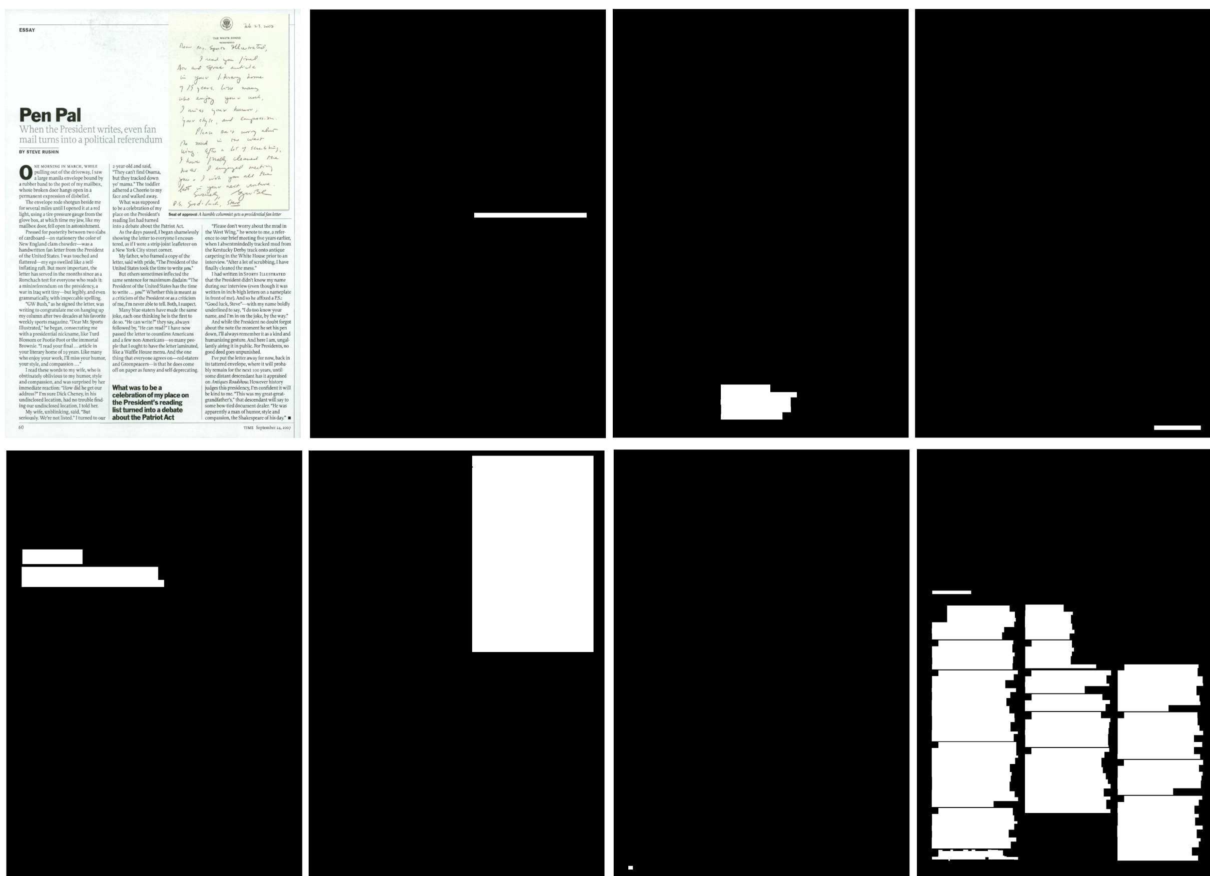
1. Elegir una imagen.
2. Para cada categoría de interés presente en dicha imagen:
   - 2.1. Obtener una imagen en negro de las mismas dimensiones que la original.
   - 2.2. Para cada zona detectada:

2.2.1. Generar la polilínea correspondiente a sus coordenadas.

2.2.2. Trazar, sobre la imagen en negro, un polígono relleno de color blanco, usando la polilínea del paso anterior.

2.3. Guardar la imagen bajo el mismo nombre que la original y un sufijo indicativo de la categoría a la que se corresponde.

A modo de ejemplo se muestran los resultados obtenidos en este proceso para una de las imágenes del dataset.



ESSAY

Pen Pal

When the President writes, even fan mail turns into a political referendum

BY STEVE RUSHIN

ONE MORNING IN MARCH, WHILE pulling out of the driveway, I saw a large manila envelope bound by a rubber band to the post of my mailbox, whose broken door hangs open in a permanent expression of disbelief.

The envelope rode shotgun beside me for several miles until I opened it at a red light, using a tire-pressure gauge from the glove box, at which time my jaw, like my mailbox door, fell open in astonishment.

Pressed for posterity between two slabs of cardboard—on stationery the color of New England clam chowder—was a handwritten fan letter from the President of the United States. I was touched and flattered—my ego swelled like a self-inflating raft. But more important, the letter has served in the months since as a Rorschach test for everyone who reads it: a minireferendum on the presidency, a war in Iraq writ tiny—but legibly, and even grammatically, with impeccable spelling.

"GW Bush," as he signed the letter, was writing to congratulate me on hanging up my column after two decades at his favorite weekly sports magazine. "Dear Mr. Sports Illustrated," he began, consecrating me with a presidential nickname, like Turd Blossom or Pootie-Poot or the immortal Brownie. "I read your final ... article in your literary home of 19 years. Like many who enjoy your work, I'll miss your humor, your style, and compassion ..."

I read these words to my wife, who is obstinately oblivious to my humor, style and compassion, and was surprised by her immediate reaction: "How did he get our address?" I'm sure Dick Cheney, in his undisclosed location, had no trouble finding our undisclosed location, I told her.

My wife, unblinking, said, "But seriously. We're not listed." I turned to our 2-year-old and said, "They can't find Osama, but they tracked down yo' mama." The toddler adhered a Cheerio to my face and walked away.

What was supposed to be a celebration of my place on the President's reading list had turned into a debate about the Patriot Act.

As the days passed, I began shamelessly showing the letter to everyone I encountered, as if I were a strip-joint leafleteer on a New York City street corner.

My father, who framed a copy of the letter, said with pride, "The President of the United States took the time to write *you*."

But others sometimes inflected the same sentence for maximum disdain: "The President of the United States has the time to write ... *you*?" Whether this is meant as a criticism of the President or as a criticism of me, I'm never able to tell. Both, I suspect.

Many blue-staters have made the same joke, each one thinking he is the first to do so. "He can write?" they say, always followed by, "He can read?" I have now passed the letter to countless Americans and a few non-Americans—so many people that I ought to have the letter laminated, like a Waffle House menu. And the one thing that everyone agrees on—red-staters and Greenpeacers—is that he does come off on paper as funny and self-deprecating.

What was to be a celebration of my place on the President's reading list turned into a debate about the Patriot Act

Seal of approval *A humble columnist gets a presidential fan letter*

"Please don't worry about the mud in the West Wing," he wrote to me, a reference to our brief meeting five years earlier, when I absentmindedly tracked mud from the Kentucky Derby track onto antique carpeting in the White House prior to an interview. "After a lot of scrubbing, I have finally cleaned the mess."

I had written in SPORTS ILLUSTRATED that the President didn't know my name during our interview (even though it was written in inch-high letters on a nameplate in front of me). And so he affixed a P.S.: "Good luck, Steve"—with my name boldly underlined to say, "I do too know your name, and I'm in on the joke, by the way."

And while the President no doubt forgot about the note the moment he set his pen down, I'll always remember it as a kind and humanizing gesture. And here I am, ungallantly airing it in public. For Presidents, no good deed goes unpunished.

I've put the letter away for now, back in its tattered envelope, where it will probably remain for the next 100 years, until some distant descendant has it appraised on *Antiques Roadshow*. However history judges this presidency, I'm confident it will be kind to me. "This was my great-great-grandfather's," that descendant will say to some bow-tied document dealer. "He was apparently a man of humor, style and compassion, the Shakespeare of his day." ■

60 TIME September 24, 2007

Figura 4.4: Ejemplo de imágenes de Ground Truth generadas.

De esta forma, si la imagen original es denominada *001.tif*, las del Ground Truth generadas se denominan *001_categoria.tif*. Los posibles sufijos para indicar la categoría se muestran en el Cuadro 4.4.

Categoría	Sufijo
Párrafos	para
Títulos	head
Imágenes	imag
Pies de página	foot
Nº de página	npag
Pie de foto	capt
Texto flotante	float

Cuadro 4.4: Sufijos de las categorías de interés.

Los nombres de las imágenes de la Figura 4.2.1 de izquierda a derecha y de arriba a abajo, son: *00001174.tif*, *00001174_capt.tif*, *00001174_float.tif*, *00001174_foot.tif*, *00001174_head.tif*, *00001174_imag.tif*, *00001174_npag.tif* y *00001174_para.tif*. El conocimiento de estos sufijos es de gran importancia, ya que son los que permiten, tanto generar adecuadamente los datos de entrada a la red neuronal, como poder realizar una correcta evaluación de los resultados obtenidos tras su entrenamiento.

Se ha decidido realizar este proceso "en frío", lo que significa que las imágenes son generadas antes de comenzar el entrenamiento de la red. De esta manera, se evita tener que repetir el proceso de creación cada vez que se reanude el entrenamiento pero, a cambio, es necesario realizar las pertinentes operaciones de lectura. En total, se deberían generar 3346 imágenes, al haber 478 instancias y siete categorías pero, en realidad, se han obtenido 2281. Esto se debe a que no todas las instancias presentan zonas de todas las categorías por lo que, para esos casos, se ha descartado la creación de la imagen correspondiente. La decisión fue tomada para evitar el mayor número de operaciones de lectura posible; en esos casos, se genera la imagen "en caliente", cuando es requerida. Al tratarse de una imagen totalmente en negro, todos sus pixels valen 0, por lo que su creación en tiempo de ejecución es muy rápida. En el Cuadro 4.5, se muestra la distribución por categoría de las 2281 imágenes creadas.

Categoría	Nº imágenes
Párrafos	471
Títulos	355
Imágenes	254
Pies de página	411
Nº de página	411
Pie de foto	210
Texto flotante	169
Total	2281

Cuadro 4.5: Imágenes de Ground Truth generadas por categoría.

Cabe destacar que ninguna de las categorías está presente en todos las imágenes, ni siquiera la de los párrafos, que no figura en siete documentos. A excepción del pie de foto y del texto flotante, el resto de categorías está presente en, al menos, la mitad de los documentos.

4.2.2. Algoritmos de Image Chopping

Image Chopping es una técnica de descomposición de imágenes consistente en dividirla en múltiples sub-imágenes de idénticas dimensiones (a partir de ahora *chunks*), de manera ordenada, de tal forma que se pueda recomponer fácilmente la imagen original.

En este trabajo se ha utilizado esta técnica para poder utilizar *gpu's* en los procesamientos de la red neuronal. Al ser las imágenes del conjunto de datos de un gran tamaño, éstas no podían ser cargadas en dichas unidades de procesamiento, por lo que ha sido necesario trocearlas (chunks).

Cada uno de estas porciones tiene asociado unos pixels extras, a los que comúnmente se les denomina margen. Esto se debe a que, sin ellos, los cálculos de la red neuronal realizados sobre los pixels de los bordes serían de menor precisión, ya que no tendrían un contexto adecuado. Al incluir estos márgenes, la red cuenta con unos pixels extras que permiten realizar unas estimaciones más correctas. Es importante destacar que, a la hora de recomponer la imagen, estos márgenes se solapan con los chunks adyacentes, por lo que los cálculos realizados sobre sus pixels no son tenidos en cuenta.

Por tanto, en un chunk existen dos zonas diferenciadas:

- *zona útil*: aquellos pixels que se emplean en la recomposición de la imagen.
- *márgenes*: pixels situados en los bordes y que únicamente se emplean para realizar con mayor precisión los cálculos de la zona útil.

Con el objetivo de poder recomponer la imagen original a partir de los chunks, se debe conocer para cada uno de ellos:

- Su localización en la imagen original.
- La ubicación de la zona útil dentro del chunk.
- La posición en la imagen original, de la zona útil.

Para este cálculo, las dos opciones más eficientes son:

- Utilizar las coordenadas de la esquina superior derecha y de la esquina inferior izquierda de la zona de interés.
- Considerar las coordenadas de la esquina superior derecha, junto al ancho y largo de la zona de interés.

Independientemente de la opción escogida para determinar las ubicaciones, el proceso de Image Chopping puede realizarse bajo dos planteamientos distintos: el enfoque recursivo o el iterativo.

- **Recursive Chopping**:
  Consiste en generar cuatro subcuadrantes de idénticas dimensiones los cuales, a su vez, se dividen en otros cuatro y estos, en otros cuatro, etcétera. Así, hasta que cada subcuadrante, o chunk, tenga unas dimensiones inferiores a las establecidas.

  En función de la ubicación del cuadrante, los márgenes se establecen en unos lados u otros. Por ejemplo, si el chunk representa la esquina superior izquierda de la imagen, los márgenes se establecen en su zona inferior y en el lateral derecho. Si por el contrario, se corresponde con la esquina inferior izquierda, los márgenes se sitúan en su zona superior y su lateral izquierdo (ver Figura 4.5).

  El valor del margen no tiene por qué ser el mismo en cada uno de los chunks obtenidos, aunque sí que está garantizado un número mínimo de pixels. Como los chunks deben ser de las mismas dimensiones, y sus zonas útiles no lo son, se aprovechan los márgenes para incluir tantos pixels, como se precisen, para alcanzar el tamaño deseado del chunk.

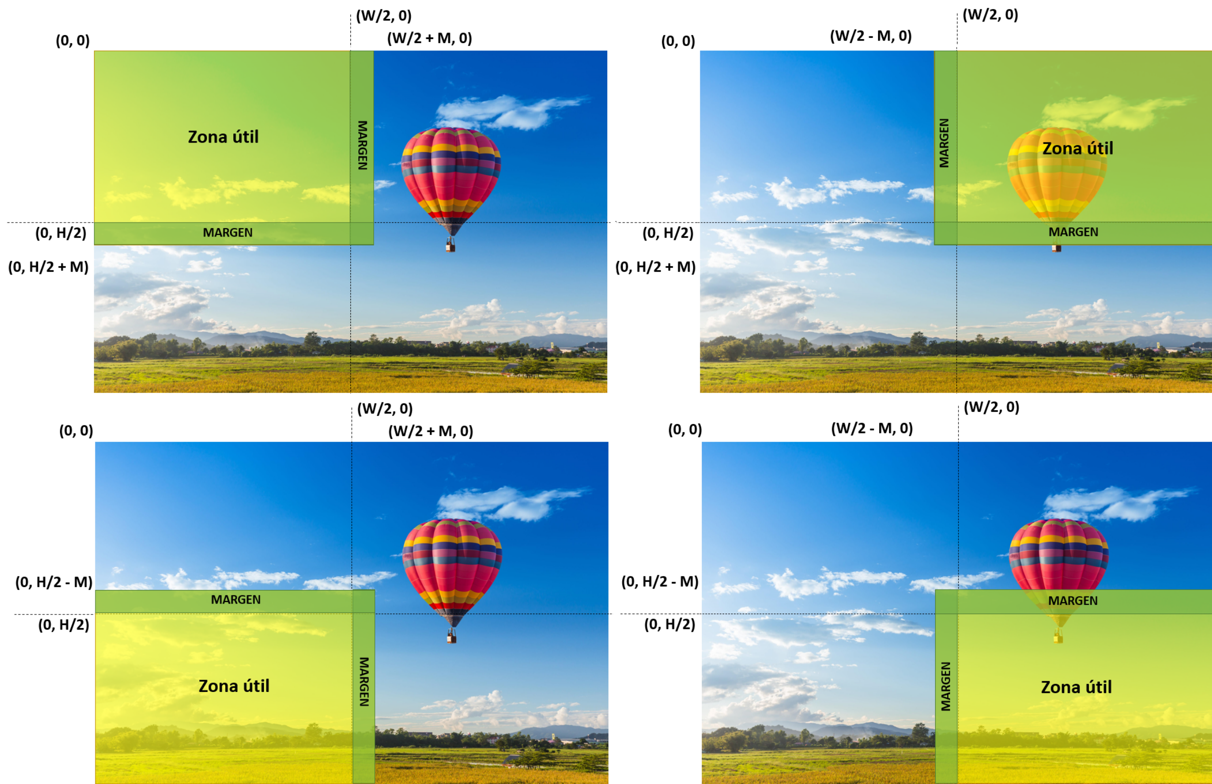


Figura 4.5: Generación de sub-cuadrantes en recursive chopping.

Los pasos en este algoritmo son:

1. Fijar la dimensión deseada para los chunks.
2. Determinar el número mínimo de pixels para los márgenes.
3. Verificar que la imagen, o chunk, de entrada sea de menor tamaño que el deseado. De ser así, realizar el caso base; si no, seguir con la secuencia alternativa.
   a) *Caso base*:
      a.1) Desplazar las coordenadas del chunk tantos pixels como sean necesarios para lograr que sus dimensiones sean las del Paso 1.
      a.2) Utilizar las coordenadas del chunk y de la zona útil en la imagen original para calcular las coordenadas relativas de la zona útil dentro del chunk.
      a.3) Almacenar, en una lista, los tres pares de coordenadas.
   b) *Secuencia alternativa*:
      b.1) Generar cuatro nuevos sub-cuadrantes o chunks, que están definidos por sus coordenadas respecto a la imagen original.

b.2) Calcular las coordenadas de la zona útil en la imagen original de cada nuevo chunk.

c.3) Para cada chunk se realiza el Paso 3, utilizando las coordenadas obtenidas en los dos pasos anteriores.

Tras finalizar este proceso, tendremos una lista con los pares de coordenadas que definen la ubicación de cada chunk en la imagen original por lo que, para generarlos, basta con ir recorriendo la lista y extrayendo los pixels correspondientes. En esta lista también tenemos las coordenadas de las zonas útiles dentro de cada chunk y su correspondencia con la imagen original. Estos dos pares de coordenadas serán de utilidad para recomponer la imagen original, tras su procesamiento en la red neuronal.

A modo de ejemplo, considérese la imagen de la Figura 4.6, de dimensiones (3034, 2217).

BOOKS DANIEL OKRENT

SUMMER LOVIN'? Edwina and Nehru

Rule Breakers

Sex, politics, and the dawn of **Indian independence.**

ABOUT TWO-THIRDS of the way through Alex von Tunzelmann's stirring book on how India won its independence, the author describes the climactic political moment, at midnight in New Delhi, Aug. 14, 1947.

"As the chimes sounded and the unexpected blast from a conch shell startled the delegates in the chamber of the Constituent Assembly," she writes, "a nation that had struggled for so many years, and sacrificed so much, was freed at last from the shackles of empire."

"Yes," von Tunzelmann concludes, "Britain was finally free." It's a wisecrack, but more wise than crack: Centuries of entanglement had by then become a disastrous knot disabling the colonizer as well as the colonized. The thrilling and horrific (and occasionally comic) events so artfully unspooled in *Indian Summer* will not convince you that Thomas Babington Macaulay was correct when he wrote in 1835 that Indian independence would result in "the proudest day in British history." But they will absolutely persuade you that it was one of the most momentous.

How could it not be with a cast of characters like this one? Von Tunzelmann's brisk narrative is propelled forward by the personalities of five memorable individuals who all wanted and worked for independence. One, of course, is Mohandas Gandhi, whose moral integrity smothers political expediency and efficiency (even if his admiration for Hitler's organizational abilities is a little, um, unsettling). Another is the cigar-smoking, foppish Mohammad Ali Jinnah, a nonobservant Muslim who almost single-handedly creates the Islamic state of Pakistan. But even they are upstaged by the trio of Louis Mountbatten, the last British viceroy, who presided over independence and the partition of India and Pakistan; his wife, Edwina; and Jawaharlal Nehru, the father of modern India.

For "trio," read "triangle." If Lady Mountbatten, as elegant and charismatic as she was rich (and she was one of the richest women in England), didn't have a physical relationship with Nehru (von Tunzelmann clearly thinks she did), it wasn't for lack of opportunity. Her husband, known to friends and family as Dickie (and to disdainful military colleagues as the "Master of Disaster"), had long tolerated and even encouraged her affairs. And Nehru—wealthy himself, educated at Harrow and Cambridge, the illustrious son of a family so thoroughly Westernized that his sisters were called Betty and Nan—was merely one of the most compelling men of the century.

The author is a matriculant in the school that believes the personal drives the political. Nehru's stubborn personal attachment to Kashmir—he called it a "mild intoxication," but he was in fact blotto on the place—exacerbated the monstrous Muslim-Hindu violence that immediately followed independence. And in Edwina's distaste for the feckless Dickie's pretensions resided the roots of her deeply democratic attachment to the Indian people.

Needless to say, the role of a Western power that heedlessly imposes itself on an Asian culture has contemporary resonance. So does the sectarian slaughter that accompanied the partition of India and Pakistan. Such echoes make this fine book relevant. But as these are the last days of summer, I'll take absorbingly readable instead—and *Indian Summer* is that too.

INDIAN SUMMER
BY ALEX VON TUNZELMANN
Henry Holt, $25.95

MY MUST-READS
BY ANTHONY BOURDAIN
The chef, TV personality, and author (The Nasty Bits, Kitchen Confidential) *dishes on his favorite food books.*

THE SOUL OF A CHEF
BY MICHAEL RUHLMAN
Superb investigation of great chefs and what drives them toward excellence.

THE APPRENTICE
BY JACQUES PEPIN
A classic account of coming up through the ranks the old-school French way—by the grandfather of French cooking in America.

HEAT
BY BILL BUFORD
Total immersion—by an outsider—into the real world of big-city cooking. It's a fascinating portrait of Mario Batali, Marco Pierre White, and Dario Cecchini, three major figures, each as different as three people could be—yet eerily similar in their passions, fears, and ambitions.

RESEARCH ASSOCIATE *Christopher Tkaczyk*

NEHRU AND MOUNTBATTEN: KEYSTONE—HULTON/GETTY; BOURDAIN: TIM PELLING

22 • FORTUNE September 3, 2007

Figura 4.6: Imagen del dataset sobre la que se mostrará el proceso de recursive chopping.

Fijando unas dimensiones para los chunks de (512, 512) y estableciendo un margen

mínimo de 15 pixels, los cuatro primeros chunks obtenidos mediante recursive chopping, para la imagen de la Figura 4.6, son los que se muestran en la Figura 4.7. En color verde claro, se presenta la zona útil del chunk y en verde oscuro, sus márgenes.

Figura 4.7: Cuatro primeros chunks obtenidos para la imagen de la Figura 4.6.

- **Iterative Chopping**:
  Consiste en utilizar un marco flotante, de dimensiones (B_W, B_H), que se va desplazando a lo largo de la imagen para generar los diferentes chunks (ver Figura 4.8). Los márgenes de cada de uno de ellos se fijan a un tamaño M que, a diferencia del recursive chopping, es el número exacto de pixels que los compondrán.

  Al igual que ocurría con el algoritmo anterior, los márgenes se establecen en torno al chunk, en función de su ubicación dentro de la imagen original. Por ejemplo, si un margen pertenece a la zona central, entonces presentará margenes en sus cuatro lados. Si por el contrario, el chunk contiene los pixels frontera de alguno de los laterales de la imagen, sólo tendrá tres márgenes. Al ser, desde un inicio, todos los chunks del mismo tamaño, las dimensiones de la zona útil vienen determinadas por el número de márgenes que presente.

  Una vez fijado la dimensión de la ventana, (B_W, B_H), y el tamaño para los márgenes, M, el algoritmo de iterative chopping, aplicado sobre una imagen de anchura W y altura H, se reduce a:

  1. Sea $b_H = B_H - M$.
  2. Mientras $b_H < H$, hacer:
     - 2.1. Sea $b_W = B_W - M$
     - 2.2. Mientras $b_W < W$, hacer:
       - 2.2.1. Cálculo de las coordenadas (ver Figura 4.8).
       - 2.2.2. Guardar coordenadas en lista.
       - 2.2.3. Actualizar b_W como: $b_W = b_W + B_W - M$.
     - 2.3. Calcular coordenadas en el eje X para el chunk correspondiente a b_W y b_H.
     - 2.4. Guardar las coordenadas en lista.
     - 2.5. Actualizar b_H como: $b_H = b_H + B_H - M$.
  3. Calcular coordenadas en el eje Y para el chunk correspondiente a b_H y b_W.
  4. Guardar las coordenadas en lista.

  Los valores b_W y b_H determinan la coordenada de la esquina inferior izquierda de cada uno de los chunks. Por ello, cuando se alcanza el margen derecho de la imagen, o el inferior, el valor de b_W, o de b_H, excede las dimensiones de la imagen, siendo necesario calcular las coordenadas del último chunk fuera de los bucles (2.2.) y (2.), respectivamente.

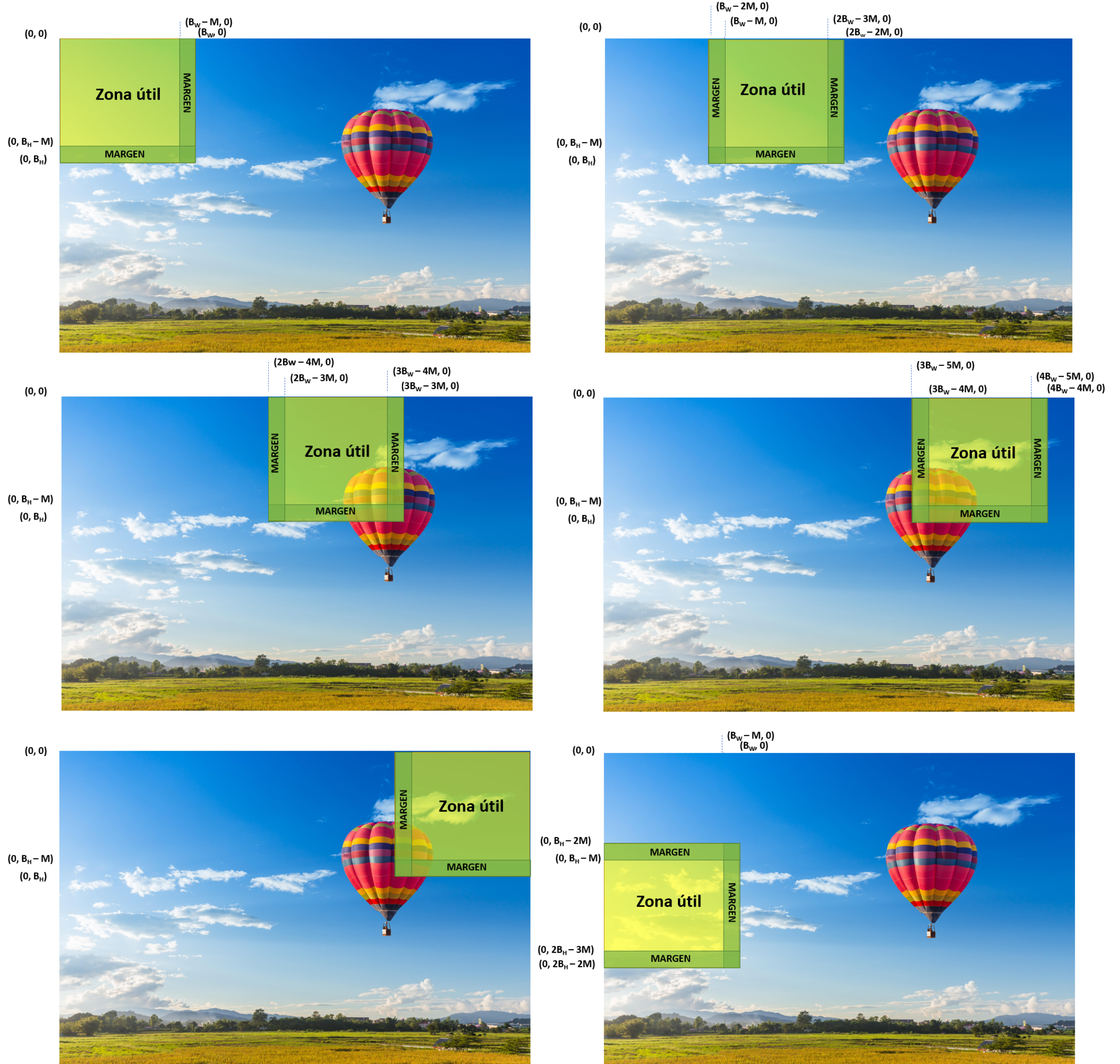


Figura 4.8: Generación de chunks en iterative chopping.

Tras este proceso, obtenemos un listado con todas las coordenadas necesarias para determinar cada uno de los chunks de la imagen en cuestión, así como todas sus partes (zona útil y márgenes). Es importante tener en cuenta, que los chunks obtenidos no serán los mismos que con el algoritmo recursivo, aunque ambos cumplen con el cometido pretendido.

Independientemente del algoritmo elegido, el proceso de chopping es recomendable realizarlo en caliente, logrando así, una mayor flexibilidad a la hora de incluir, o excluir, imágenes del conjunto a procesar por la red. Si se realizase en frío, sería necesario recalcular el listado de coordenadas cada vez que se realizase una modificación en dicho conjunto para, posteriormente, pasárselo a la aplicación como parámetro. Al realizarlo en caliente, la aplicación calcula el listado automáticamente, evitando así tener que hacer el proceso de forma manual para cada modificación realizada. Al ser un cómputo ligero, su realización en caliente no supone un inconveniente, ya que la porción de tiempo destinada a este proceso, en comparación a los cálculos realizados en la red, es insignificante.

4.3. Implementación

En esta Sección se va a justificar la decisión de la utilización de la biblioteca *PyTorch*, frente a *Keras* estudiada durante la formación académica. A continuación, se describirán los detalles de la aplicación desarrollada. Por último, se detallarán las decisiones tomadas sobre la arquitectura de red UNET implementada, así como su creación haciendo uso de la librería PyTorch.

4.3.1. Pytorch vs Keras

Son muy populares para la implementación de Aprendizaje Profundo haciendo uso del lenguaje de programación *Python*, que es el empleado en este trabajo.

A continuación, se van a exponer una serie de puntos comparativos entre ambas opciones:

- *Facilidad de uso y flexibilidad*:
  Keras es un marco de trabajo de alto nivel, que encapsula las operaciones y las capas de Aprendizaje Profundo en bloques de procesamiento, haciendo el código más legible y conciso. Como contrapunto, supone una limitación a la hora de jugar con los hiperparámetros de la red. Por su parte, PyTorch proporciona un entorno de nivel relativamente bajo, que permite una mayor libertad en la implementación de la red neuronal. Sin embargo, es menos legible.

- *Depuración e instrospección*:
  PyTorch proporciona una experiencia de depuración más sencilla y completa que la proporcionada por Keras, siendo posible ejecutar el script línea a línea, así como acceder de manera sencilla a cada uno de los objetos declarados en el código.

- *Rendimiento*:
  La velocidad de cómputo es un factor clave en el desarrollo del presente trabajo. En este aspecto, PyTorch supera con claridad a Keras [22].

Teniendo en cuenta las consideraciones realizadas, parece claro que Keras constituye una opción más simple que PyTorch, a costa de un menor nivel de personalización y profundización en los conceptos teóricos subyacentes. Sin embargo, en este trabajo se persigue ahondar en los conocimientos de las redes convolucionales y, en particular, de la arquitectura UNET, por lo que la flexibilidad de PyTorch, así como su mayor velocidad, la convierten en la opción más conveniente.

Pero la elección no sólo está sustentada en los motivos anteriormente descritos, sino que los tutores de HP me expresaron su preferencia por el desarrollo en esta plataforma, debido a su alto grado de conocimiento en la misma. Para facilitar mi familiarización con este framework, me proporcionaron el esqueleto software de una aplicación para una red neuronal haciendo uso de PyTorch. Este esqueleto ha sido la base para el desarrollo de este trabajo, y su estructura es la que se presenta a continuación.

4.3.2. Descripción de la aplicación desarollada

La aplicación proporcionada por HP, y que ha sido utilizada como base para el desarrollo de este trabajo, presenta cuatro partes diferenciadas, que son: *Main*, *NetManager*, *NetArchitecture* y *NetDataSet*.

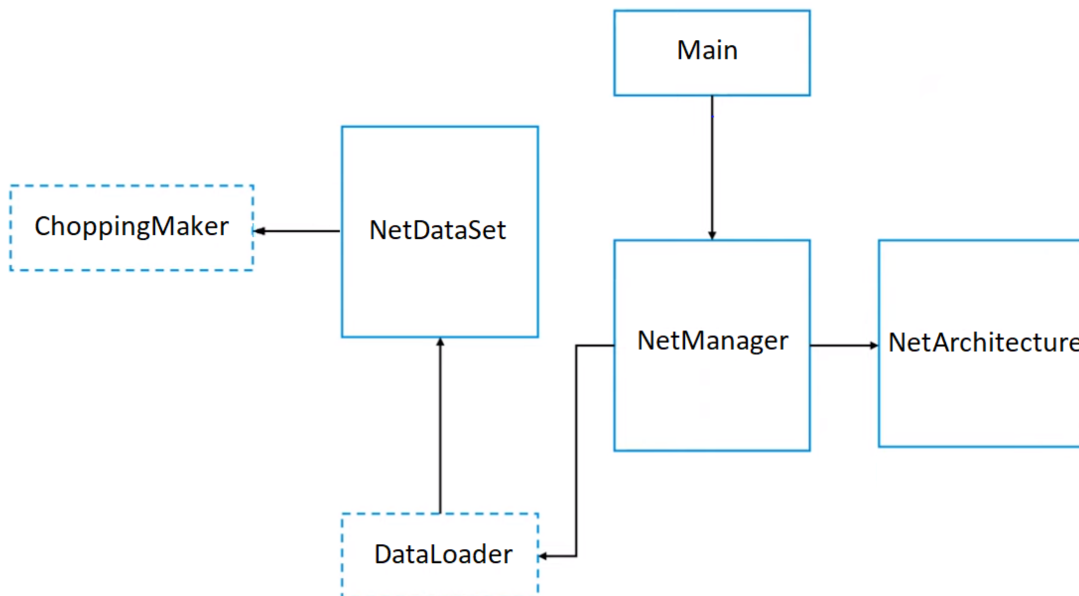


Figura 4.9: Relación de los elementos del esqueleto software.

En la Figura 4.9, se muestra la configuración final de la aplicación. Se cuenta con las cuatro partes mecionadas, que se reparten la lógica de la aplicación, más dos clases auxiliares que contribuyen al buen funcionamiento de la misma. La clase *DataLoader*, perteneciente a PyTorch, es utilizada por las instancias de NetManager para un correcto uso de las instancias de la clase DataSet. Por su parte, la clase *ChoppingMaker* ha sido específicamente creada para el presente trabajo: se encarga de generar los chunks del conjunto de entrada a la red.

A continuación, se explican las utilidades contenidas en cada una de estas clases:

- **Main**:
  Es el programa inicial, se encarga de recoger la línea de comandos escrita por el usuario para inicializar la aplicación y, en función de los parámetros leídos, establecer las características del gestor de la red. Este proceso se realiza haciendo uso de la librería *docopt*.

  El listado de parámetros de entrada que acepta esta clase, es el siguiente:

  - *--train*: indica que se quiere ejecutar la red neuronal en modo entrenamiento, incluye validación.
  - *--validate*: para correr la red neuronal sólo en modo de validación.
  - *--test*: si se quiere usar la red neuronal en modo test.
  - *--ld = <file>*: proporciona a la aplicación la ruta para localizar un modelo de red ya entrenado, a partir del cual, se desea realizar entrenamiento/validación/test.
  - *--in = <input_source>*: ruta a la carpeta de las imágenes que la red tomará como datos de entrada para el entrenamiento, o test.
  - *--gt = <gt_imag>*: ruta a la carpeta que almacena las imágenes de Ground Truth de las imágenes tomadas como entrada por la red para el entrenamiento.
  - *--inv = <valid_input_source>*: ruta a la carpeta que contiene las imágenes que la red tomará como datos de entrada para la validación.
  - *--gtv = <valid_gt_image>*: ruta a la carpeta que alberga las imágenes de Ground Truth de las imágenes tomadas como entrada por la red para la validación.
  - *--e = <max_epoch>*: Número máximo de épocas para el entrenamiento.
  - *--cpu*: fuerza el uso de CPU para los cálculos. Si no se indica, se realizarán en GPU.
  - *--out = <test_output>*: ruta de destino para las imágenes obtenidas tras usar la red neuronal en modo test.

En función del modo en el que se desee utilizar la red neuronal, se utilizarán unos parámetros u otros. Para más información se remite al lector al Apéndice B, correspondiente al *Manual del Usuario*.

- **NetManager**:
  Esta clase se encarga de gestionar todo lo relacionado con la red neuronal y su uso.

  Entre sus atributos, se encuentran todos aquellos parámetros que determinan el comportamiento de la red, como son el optimizador y la función de pérdida. Asimismo, se especifica el propio modelo de red, mediante una instancia de la clase NetArchitecture.

  Aquí se definen los métodos para cargar la información de un modelo ya entrenado, pasado como parámetro a la aplicación, y para guardar el modelo actual.

  Es el responsable de gestionar el entrenamiento, la validación y el test sobre la red neuronal contando, para ello, con un método diferente para cada modo. En cada uno de estos métodos, se crea una instancia de la clase *NetDataset*, que devuelve las imágenes de entrada, ya listas para su uso en la red neuronal. Mediante la clase *DataLoader*, se transfieren los lotes de imágenes a la red.

  En los modos de entrenamiento y validación, el sistema lleva un recuento del mejor modelo conseguido hasta ese momento, por lo que en caso de que se obtenga una mejora, se guarda la configuración correspondiente y se actualiza la información pertinente. Después de cada época de entrenamiento, la aplicación guarda los modelos obtenidos tras el último entrenamiento y la última validación, con el fin de poder retomar el entrenamiento en otro momento, si así se desea.

- **NetDataSet**:
  Consiste en una redefinición de la clase *Dataset* de PyTorch. Es la clase responsable de la gestión y adecuación de las imágenes para poder ser utilizadas como entrada en la red neuronal. De igual manera, se encarga de recomponer la salida obtenida por la red en el modo test.

  Cuando se inicializa una instancia de esta clase, se hace uso de la clase *ChoppingMaker*, para generar un listado con todos los chunks de las imágenes, que se desean usar como entrada en la red. Con el objetivo de que la red reciba cada uno de estos chunks como entrada, y no la imagen correspondiente, se han tenido que redefinir los métodos: *__len__()* y *__getitem__(index)*.

El método *__len__()* devuelve el número de instancias en el conjunto de entrada. En nuestro caso, no se trata del número de imágenes, sino de la cantidad total de chunks entre todas las imágenes de entrada.

Por su parte, el método *__getitem__(index)* es el encargado de preparar la instancia (chunk) número "index" para ser utilizada por la red. Es necesario generar dos tensores: el de entrada a la red y el que contiene la información de la salida deseada. A continuación, se explica cómo se genera cada uno de ellos:

- tensor de entrada, conformado por los canales (R,G,B), correspondientes a la zona de la imagen original, que abarca el chunk. Asimismo, cuenta con dos canales extra, que aportan información sobre la ubicación de cada uno de los pixels del chunk. A modo de ejemplo, dado un chunk que tiene su esquina superior izquierda en los pixels (0,255) y su inferior derecha en (256,511), los dos canales extras serán: $C_x = [0, 1, 2, ..., 255]$ y $C_y = [255, 256, 257, ..., 511]$. Por tanto, las dimensiones de este tensor son: [5, $width_{chunk}$, $height_{chunk}$].
- tensor para la salida, alberga la información correspondiente al Ground Truth del chunk. Para cada categoría, se genera un tensor, de dimensiones [1, $width_{chunk}$, $height_{chunk}$], tras leer la imagen de su Ground Truth en escala de grises (por eso sólo presenta un canal). En caso de que no exista la imagen de su Ground Truth, el contenido del tensor será 0, ya que esto significa que la imagen no presenta dicha categoría en ninguna de sus zonas. Tras obtener los siete tensores, uno por cada categoría, se concatenan en un único tensor estático, de dimensiones [7, $width_{chunk}$, $height_{chunk}$], que es el utilizado para evaluar la salida de la red.

Esta clase también se encarga de recomponer, en las pertinentes imágenes, los chunks de salida obtenidos por la red, cuando se usa en modo test. Para ello, utiliza las coordenadas del listado obtenido mediante la clase *ChoppingMaker*, para extraer la zona útil de cada chunk e ir recomponiendo las imágenes de las distintas categorías.

- **ChoppingMaker**:
  Esta clase se encarga de realizar el proceso de *Image Chopping* sobre el conjunto de imágenes existentes en una determinada ruta, pasada como parámetro. Como resultado, se obtiene un listado con la siguiente información:
  - *nombre de la imagen original*, para poder identificar unos chunks de otros, tanto en la fase de preparación para la entrada a la red, como en la de recomposición de la salida.
  - *coordenadas del chunk*, en la imagen original.
  - *coordenadas de la zona útil*, en la imagen original.

◦ *coordenadas relativas de la zona útil*, respecto al chunk correspondiente.

Como se mencionó en la Sección 4.2.1, el algoritmo elegido para el desarrollo de este trabajo es el *recursive chopping*, por lo que es la versión implementada en esta clase.

- **NetArchitecture**:
  En esta clase únicamente se definen las características de la red neuronal implementada. Es una subclase de la clase *Module* de PyTorch.

  Las instancias de esta clase son generadas por instancias de la clase NetManager, que son las encargadas de realizar el entrenamiento, o gestionar el proceso de validación o test, de la red neuronal.

4.3.3. Configuración de la red neuronal

En este capítulo, se presentan las características de la red neuronal elegida. Cabe recordar, que la arquitectura escogida ha sido la UNET, que fue propuesta por primera vez en el año 2015 para su uso en problemas de segmentación de imágenes biomédicas [9].

Los detalles más teóricos sobre esta arquitectura fueron presentados en el Capítulo 2.3, por lo que ahora nos centraremos en los detalles propios de su implementación.

La red neuronal utilizada en este trabajo, y que se muestra en la Figura 4.10, difiere en muchos aspectos de la red UNET original de la Figura 2.8. Las principales diferencias se encuentran en:

- *Número de canales*: en la arquitectura original, por cada nivel de profundidad, el número de canales se veía duplicado, en la ruta de contracción, y dividido a la mitad, en la de expansión. En esta implementación, para cada nivel, se mantiene una misma secuencia del número de canales en cada capa convolucional.

- *Dimensión de los mapas de características*: el número de pixels por dimensión se mantiene constante a lo largo de una misma capa convolucional. En la red original, por cada convolución, se perdían dos pixels en cada dimensión.

- *Técnica del residuo*: en la arquitectura original, la salida de cada nivel de la ruta de contracción se concatenaba con la entrada correspondiente a la ruta de expansión. En este trabajo, se ha optado por emplear la técnica del residuo, que como vimos en el Capítulo 2.3.3, consiste en sumar ambos tensores.

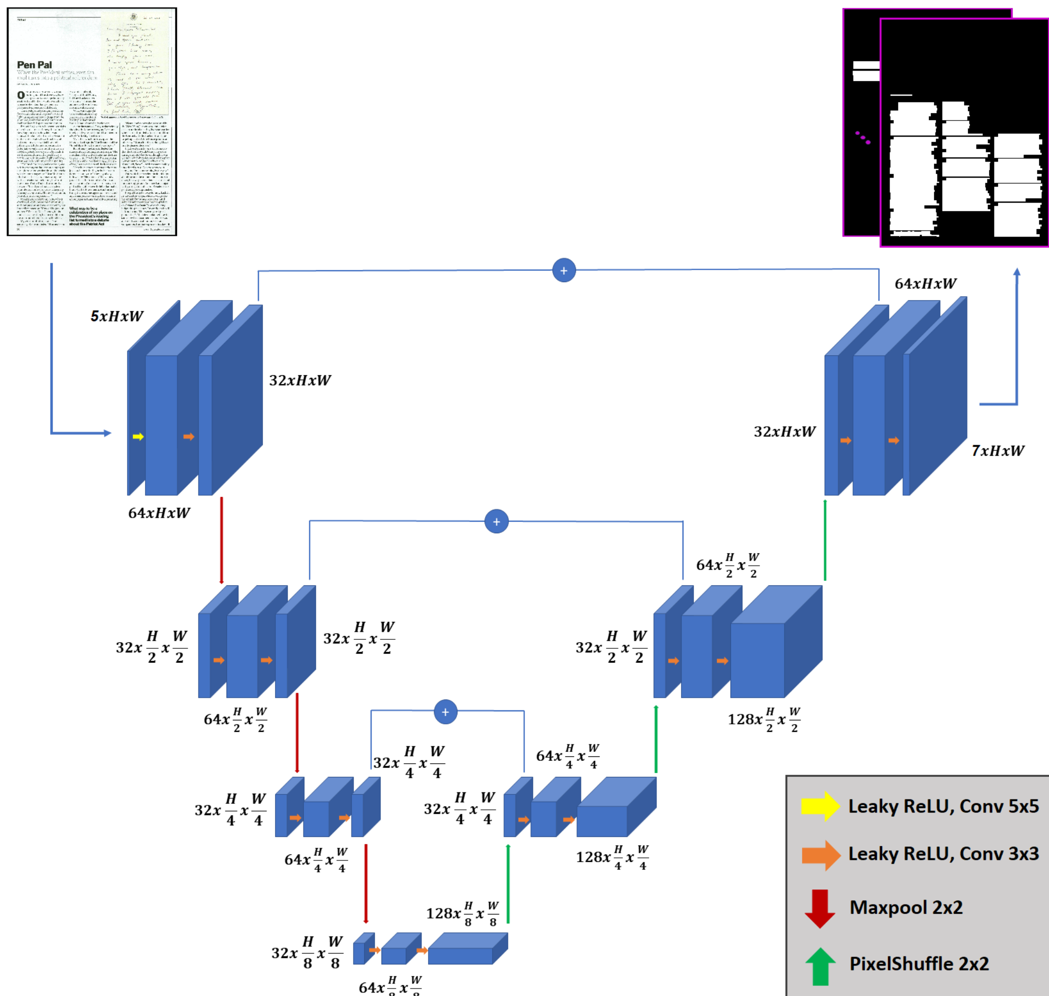


Figura 4.10: Arquitectura de la red UNET implementada.

La red toma como entrada cada uno de los chunks que hemos generado mediante el algoritmo de recursive chopping y, por esta razón, el tensor de entrada presenta cinco canales de tamaño (H, W). Dos de ellos, como se vió en la sección anterior, se corresponden con la información de la ubicación espacial de los pixels del chunk, mientras que los otros tres es el contenido de cada pixel en la imagen original en formato *(R,G,B)*.

Con el objetivo de utilizar al máximo la información del chunk de entrada, en la primera convolución que realiza la red, se utiliza un núcleo 5x5, que permite crear filtros más complejos. Para evitar la pérdida de pixels, como consecuencia de esta operación, se utiliza un *padding* igual a 2, estableciéndose 64 canales de salida. A continuación, se realiza una nueva convolución, ahora con un núcleo 3x3 y un *padding* igual a 1, dando lugar a 32 canales. Estas dos operaciones conforman la primera capa de la red, que puede verse en mayor detalle en la Figura 4.11.

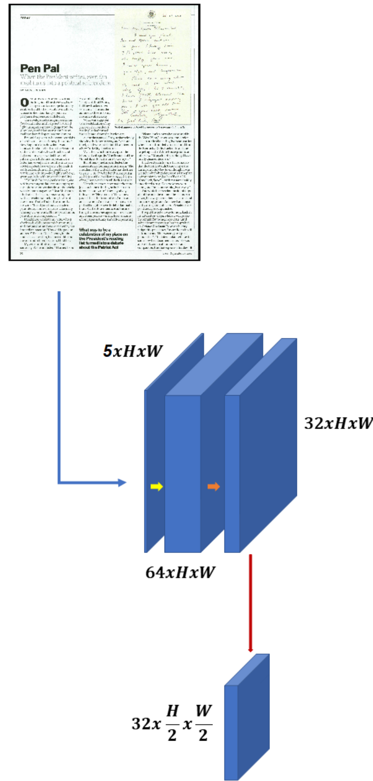


Figura 4.11: Arquitectura red neuronal I.

Tras el procesamiento de la primera capa, tiene lugar la operación de submuestreo de características (ver Figura 4.11). Cómo se vió en el Capítulo 2.2.2, existen dos opciones principales para llevarlo acabo: *average pooling* y *max pooling*. En esta implementación, se ha decidido hacer uso del max pooling, con un mosaico 2x2, el cual permite reducir las dimensiones H y W a la mitad.

El segundo nivel de profundidad, recibe como entrada el tensor de 32 canales y dimensiones $(\frac{H}{2}, \frac{W}{2})$. Mediante una primera convolución con núcleo 3x3 y padding igual a 1, se generan 64 canales, que vuelven a ser 32 tras una nueva convolución de idénticas características. Tras estas dos operaciones, se produce de nuevo el max pooling con mosaico 2x2, que genera un tensor de 32 canales y dimensiones $(\frac{H}{4}, \frac{W}{4})$. Esta salida es procesada en la siguiente capa, siguiendo el mismo procedimiento que en el nivel anterior, como puede verse en la Figura 4.12.

De esta forma, al cuarto y último nivel de profundidad, el denominado cuello de botella, llega un tensor con 32 canales de dimensiones $(\frac{H}{8}, \frac{W}{8})$. De nuevo, se realizan las dos operaciones convolucionales anteriores, pasando de 32 a 64 canales en la primera y, de 64 a 128, en la segunda.

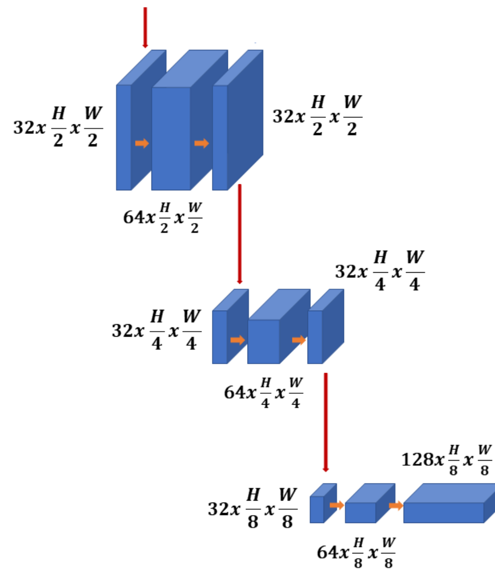


Figura 4.12: Arquitectura red neuronal II.

Para realizar la operación de upsampling, entre todas las posibilidades vistas en el capítulo 2.3.2, se ha optado por la utilización del *pixel shuffle*. Mediante el uso de esta técnica es posible pasar de un tensor de dimensiones $(r^2 \cdot C, H, W)$ a otro con dimensiones $(C, H \cdot r, W \cdot r)$. En nuestro caso, como queremos ir duplicando las dimensiones según vamos subiendo por la ruta de expansión, el valor de r elegido es 2. Por ello, en la última convolución se ha pasado de 64 canales a 128 (ver Figura 4.12) para que, tras realizar el pixel shuffle, se obtengan 32 canales.

Después de esto, el tensor obtenido es de dimensiones $(32, \frac{H}{4}, \frac{W}{4})$. Antes de ser utilizado por la siguiente capa de la red, se aplica, sobre él, la técnica del residuo, sumándole la salida que se había obtenido en el tercer nivel de la ruta de contracción (ver Figura 4.13). Una vez sumados ambos tensores, el resultado es sometido a dos convoluciones de idénticas características que las realizadas en el cuello de botella.

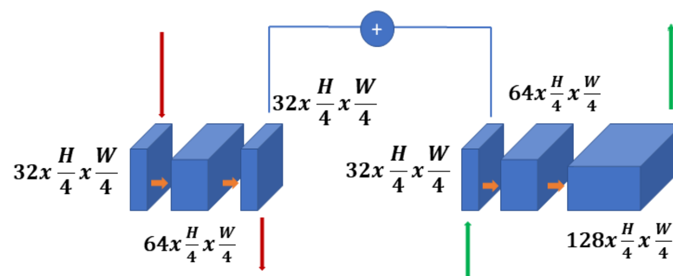


Figura 4.13: Arquitectura red neuronal III.

De nuevo, se realiza el pixel shuffle sobre la salida de la última convolución, para posteriormente aplicar la técnica del residuo y obtener, así, la entrada a la penúltima capa de la red. En este nivel, se repiten tanto las convoluciones, como el upsampling, obteniendo un tensor de salida de dimensiones $(32, H, W)$. Esto sirve de entrada, tras aplicar el residuo, a la última capa. En ella, la primera convolución es igual a las que se vienen realizando, mientras que la segunda difiere. En particular, al ser la salida definitiva de la red, se ha de adaptar el tensor obtenido a las condiciones del problema, por ello el número de canales de salida pasa a ser 7, uno por cada categoría, como se muestra en la Figura 4.14).

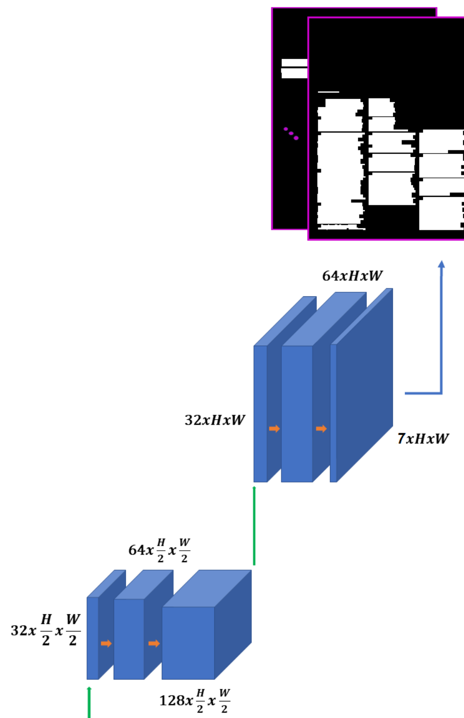


Figura 4.14: Arquitectura red neuronal IV.

El tensor de salida obtenido, por tanto, es de dimensiones $(7, H, W)$, representando cada canal una de las distintas categorías del problema. Esto es posible gracias a que este tensor de salida es comparado, usando la función de pérdida, con el de la salida deseada. Como se vió en la sección anterior, dicho tensor está conformado por el Ground Truth de las siete categorías del chunk, por lo que, de esta forma, se consigue que cada canal se especialice en una de las clases.

Para que este proceso dé buenos resultados, el tensor de salida deseada debe ser generado siempre de igual manera, es decir, haciendo corresponder cada categoría al mismo canal durante todo el entrenamiento. En nuestro caso, los canales, del 1 al 7, se corresponden con: “pie de foto”, “texto flotante”, “pie de página”, “títulos”, “imágenes”, “número de página” y “párrafos”, respectivamente.

Cómo hemos visto, en el procesamiento de la red, se utilizan dos configuraciones distintas para las convoluciones:

- *Convolución con kernel 5x5*, y padding igual a 2, para extraer la máxima información disponible del chunk de entrada. Se realiza únicamente en el momento de entrada de datos a la red.
- *Convolución con kernel 3x3*, y padding igual a 1, en el resto de capas de la red.

Pero las convoluciones no se realizan solas, sino que van acompañadas de una normalización y una función de activación previas. A este esquema se le conoce como *NAC (Normalization, Activation & Convolution)*, y según me explicaron los tutores de HP, está dando mejores resultados, que el esquema habitual *CNA (Convolution, Normalization & Activation)*, por lo que ha sido la opción escogida.

Como método de normalización, se ha optado por *Instance Normalization* (ver Capítulo 2.4.1), mientras que la función de activación escogida ha sido la *Leaky ReLU* (ver Capítulo 2.4.2). La elección de ambas ha sido motivada, en gran medida, por las recomendaciones realizadas por los tutores.

4.3.4. Parámetros relativos al entrenamiento

El éxito de un modelo no depende exclusivamente de la arquitectura de la red escogida, sino que los parámetros de entrenamiento juegan un papel muy importante en la eficiencia y precisión del clasificador. A continuación, se explicarán los parámetros considerados, así como los valores que se les ha otorgado:

- *Número de épocas*: a diferencia de otros proyectos, en los que se establece un número de épocas de antemano, en este trabajo, este valor ha estado sujeto a la disponibilidad de tiempo. En total, se han realizado 52 épocas.
- *Learning rate*: valor escalar usado para determinar la magnitud del siguiente punto en los algoritmos de descenso del gradiente. El rango de valores recomendado es [0.001, 0.0001]. En este trabajo, siguiendo las recomendaciones de los tutores, se ha optado por un valor de 0.0001.
- *Optimizador*: es el algoritmo encargado de actualizar los parámetros de peso de la red con el objetivo de minimizar la función de pérdida. En este trabajo, se ha optado por el optimizador *Adam*, debido a que combina varias de las mejores propiedades de otros optimizadores [23].
    - *betas*: son la tasas de decaimiento iniciales utilizadas para estimar el primer y segundo momento del gradiente. Estas tasas, determinan la velocidad de aprendizaje según el algoritmo se vaya acercando a un mínimo local logrando,

de esta manera, que no se den saltos demasiado grandes, obteniéndose una mayor precisión. Los valores escogidos han sido $\beta_1 = 0{,}9$ y $\beta_2 = 0{,}999$, siguiendo las recomendaciones de los tutores.

- *Criterion*: o función de pérdida, es la encargada de comparar la salida de la red con la salida deseada. Se ha optado por el error cuadrático medio *(MSE)*, que fue explicado en el capítulo 2.4.3.

Para realizar el chopping de las imágenes, se ha establecido un margen mínimo de 10 pixels y unas dimensiones de chunks de $(256, 256)$ que serán, por tanto, el tamaño de los canales de entrada a la red.

Otro aspecto relevante, aunque no es un parámetro como tal, es la inicialización de los pesos de la red. Este proceso es de vital importancia para conseguir que la red sea capaz de aprender, ya que si no se inicializan correctamente, se corre el riesgo de que los valores de los gradientes sean infinitesimales. Para solventar este problema, siempre que el entrenamiento se realice desde cero, y no partiendo de un modelo pre-entreenado, se utiliza la *Inicialización Normalizada* [24].

Esta operación es conocida como *Inicialización de Xavier*, debido al nombre de uno de sus autores. Propone una inicialización de pesos a partir de una distribución gaussiana. De esta forma, se persigue el objetivo de mantener la varianza de las activaciones y los gradientes repropagados a lo largo de las capas de la red consiguiendo, así, que sea capaz de aprender. Los valores de la distribución se mantienen en el intervalo:

$$\pm\, gain \cdot \sqrt{\frac{6}{n_i + n_{i+1}}} \tag{4.1}$$

siendo:

- n_i, el número de conexiones neuronales en la entrada de la capa i (fan-in).
- n_{i+1}, el número de conexiones neuronales en la salida de la capa i (fan-out).
- $gain$, un factor de escalado.

4.3.5. Creación de un modelo en PyTorch

Una vez explicada, tanto la configuración de la red, como los parámetros y aspectos relevantes en su entrenamiento, es el momento de conocer cómo se ha desarrollado el modelo mediante PyTorch.

Dentro de todas las opciones que ofrece PyTorch, nos centraremos en su paquete **torch.nn**. En él, contamos con multitud de clases que nos permiten crear un modelo de manera intuitiva, sin perder flexibilidad y control sobre sus componentes.

Comenzaremos la explicación centrándonos en su clase **Module**. Se trata de una clase abstracta en la que es posible contener otras instancias de Module, con sus propias configuraciones, formando de esta forma una estructura recursiva de árbol. Como se establece en su documentación [25], cualquier **modelo** de red neuronal que deseemos llevar a cabo, debe ser una **sub-clase** de ésta. Gracias a esto, nuestro modelo podrá aprovechar los métodos de la clase Module, entre los que destacan, para nuestro propósito, los siguientes:

- *train(mode=True)*: establece el modo entrenamiento para el modelo.
- *test()*: cambia a modo de prueba para el modelo.
- *apply(fn)*: lleva a cabo la función *fn* sobre todos sus sub-módulos.
- *parameters(recurse=True)*: devuelve un iterador sobre todos sus parámetros. Si recurse es True, entonces devuelve los de todos sus módulos y sub-módulos; en caso contrario, sólo los directos del módulo.
- *cpu()*: mueve los parámetros del modelo y sus buffers a la CPU.
- *cuda(device=None)*: traslada los parámetros del modelo y sus buffers a GPU.
- *xpu(device=None)*: cambia los parámetros del modelo y sus buffers a la XPU.
- *load_state_dict(state_dict,strict=True)*: copia los parámetros y buffers de *state_dict* en sus módulos y descendientes. Si el segundo parámetro, *strict*, es True, los nombres de los campos de state_dict deben coincidir con las claves devueltas por la función *state_dict()*.
- *to(*args,**kwargs)*: permite mover y/o hacer un cast de los parámetros y buffers. Como parámetro de entrada existen las siguientes posibilidades:
  - *device(torch.device)*: el dispositivo al que se desea enviar los parámetros y buffers.
  - *dtype(torch.dtyoe)*: el tipo de punto flotante o dtype complejo al que se desea realizar el cast.

- *tensor(torch.Tensor)*: Tensor cuyo dtype y dispositivo asociado son los deseados para todos los parámetros y buffers del módulo.
- *memory_format(torch.memory_format)*: el formato de memoria deseado para los parámetros y buffers 4D.

Para crear nuestro modelo como una sub-clase de Module, es preciso llevar a cabo dos tareas. La primera de ellas, consiste en definir, en el método *init()*, el constructor del modelo, inicializando la clase Module mediante *super.init()* e instanciando los componentes de la red. La segunda de ellas consiste en definir las operaciones de forward y la salida de la red, haciendo uso de la función *forward()*.

Con el objetivo de conseguir una mayor comodidad y legibilidad del código, se ha utilizado la clase **Sequential** para instanciar los componentes por niveles de profundidad. Esta clase es un “contenedor”, en el que los objetos de la clase Module son añadidos en el mismo orden, en el que se desea que sean pasados al constructor. Haciendo uso de esta clase, podemos encapsular todos los elementos de un nivel de la red bajo el mismo nombre, lo que permite un código más claro.

Los componentes de una capa, como vimos en la sección anterior, son dos operaciones convolución, precedidas cada una de ellas de una normalización y una función de activación. Por tanto, nuestros bloques *Sequential* estarán compuestos por seis componentes, que son pasados como parámetros de entrada siguiendo el orden: $normalizacion_1$, $funcion_activacion_1$, $convolucion_1$, $normalizacion_2$, $funcion_activacion_2$, $convolucion_2$.

La operación **convolución** se puede realizar a través de la clase **Conv2D**, en la que se pueden establecer los siguientes atributos:

- *in_channels*: número de canales de entrada.
- *out_channels*: cantidad de canales de salida.
- *kernel_size*: dimensiones del kernel o núcleo. Puede ser un int, y el núcleo será de dimensiones AxA, o una tupla para kernels no cuadrados, AxB.
- *stride=1*: paso entre aplicaciones consecutivas del kernel. Al igual que kernel_size, puede ser un int o una tupla.
- *padding=0*: número de filas y columnas a insertar. Al igual que kernel_size y stide, puede ser un int o una tupla.
- *padding_mode='zeros'*: tipo de padding a realizar. Otras opciones son: *'reflect'*, *'replicate'* y *'circular'* [26].
- *dilation=1*: controla el espaciado entre los puntos del kernel. Es un concepto no desarrollado en este trabajo pero que puede ser consultado en [27].

- *groups=1*: controla la relación entre las entradas y las salidas. Es necesario que, tanto in_channels, como out_channels sean divisibles, entre el groups. Su utilidad se explica en detalle en [26].
- *bias=True*: si es True, añade el bias de aprendizaje a la salida.

Aunque en este trabajo se ha utilizado la convolución 2D, también se puede llevar a cabo en una o tres dimensiones, con las clases **Con1D** y **Conv3D**, respectivamente, que tienen los mismos atributos que la versión en dos dimensiones.

Para realizar la **normalización**, se ha hecho uso de la clase **InstanceNorm2D**, que cuenta con los siguientes parámetros:

- *num_features*: número de canales de la entrada.
- *eps*: se corresponde con el valor ϵ visto en la Ecuación 2.12. Añade estabilidad numérica a los cálculos. Por defecto su valor es: 1e-5.
- *momentum*: valor utilizado para el cálculo de running_mean y running_var [28].
- *affine*: valor booleano que determina la existencia o no de parámetros de "afinación" [28]. Por defecto, su valor es False.
- *track_running_stats*: variable booleana que cuando se establece a True, fuerza el cálculo de la media y la varianza en tiempo de ejecución. En caso contrario, se utiliza las estadísticas por lotes en los modos de entrenamiento y evaluación [28]. Por defecto, su valor es False.

En caso de haber optado por el uso de Batch Normalization en nuestra red, en vez de Instance, tendríamos que haber hecho uso de la clase **BatchNorm2D**, que cuenta con los mismos atributos que los descritos para InstanceNorm2D. Ambas opciones cuentan, a su vez, con versiones para datos de entrada en una y tres dimensiones.

Por su parte, para la **función de activación** se ha utilizado la clase **LeakyReLU**, que cuenta con dos atributos:

- *negative_slope*: controla el ángulo de la pendiente de la recta conformada por los valores negativos en el eje de las X. Por defecto, su valor es 0.01.
- *inplace*: determina si los cálculos realizados pueden sobreescribir a los valores previos. Por defecto, se establece en False.

De igual manera, se cuenta con suficientes clases como para poder utilizar el resto de funciones de activación vistas en el capítulo 2.4.2.

Una vez explicado cómo podemos crear los distintos niveles de profundidad, es el momento de hablar sobre las operaciones de subsampling y upsampling.

La opción escogida para realizar el **subsampling**, o muestreo de características, fue el max pool. Mediante la clase **MaxPool2D** de PyTorch podemos llevarla a cabo. Los atributos de esta clase, son:

- *kernel_size*: establece las dimensiones del mosaico. Puede ser un int, mosaico cuadrado, o una tupla.
- *stride=None*: el stride de la ventana. Por defecto, se corresponde con kernel_size.
- *padding=0*: número de filas y columnas, de ceros, a añadir tras el cálculo.
- *dilation=1*: controla el stride de los elementos de la ventana.
- *return_indices=False*: permite devolver, junto a la salida, los índices de los valores máximos.
- *ceil_mode=False*: realiza un redondeo al alza si es True, y a la baja si es False.

Por su parte, para implementar el pixel shuffle, opción escogida para el **upsampling**, se hace uso de la clase **PixelShuffle**, cuyo único atributo es *upscale_factor*, que determina el valor por el que la salida verá aumentada sus dimensiones H y W respecto a la entrada, cómo se explicó en el capítulo 2.3.2.

Tras tener definidos, e instanciados en el método *__init__()*, todos los componentes de la red, es el momento de definir como se relacionan entre ellos. En particular, debemos aclarar como fluye la información entre las instancias Sequential, ya que el orden de procesamiento interno en cada una de ellas está determinado por el orden de declaración de sus componentes.

Con este fin, se utiliza la función *forward()*, anteriormente mencionada. En ella, se define el orden en el que los datos de entrada a la red deben atravesar las distintas instancias Sequential generadas. Se debe tener en cuenta que, la salida de una de estas instancias, debe ser sometida al correspondiente proceso de subsampling/upsampling, antes de servir como entrada de la siguiente. De igual manera, en esta función también se lleva a cabo la técnica del residuo explicada en el capítulo 2.3.3.

Teniendo en cuenta todos estos aspectos, podemos definir la arquitectura de red escogida, haciendo uso exclusivamente del paquete *torch.nn*. Pero, para determinar el comportamiento de la red, debemos considerar, al menos, dos aspectos más: la función de pérdida y el optimizador.

Las **funciones de pérdida** se incluyen, también, en el paquete torch.nn. En este trabajo se ha empleado el error cuadrático medio, MSE, que en PyTorch se encuentra bajo el nombre **MSELoss**. Otras posibilidades, como las mencionadas en el capítulo 2.4.3, son **L1Loss** (MAE) y **SmoothL1** (Hubber).

En cuanto a los **optimizadores**, cuentan con su propio paquete denominado **torch.optim**. Como se mencionó en la sección anterior, en la implementación desarrollada, se ha optado por el optimizador **Adam**, que se puede encontrar con idéntico nombre en el paquete. Sus parámetros son:

- *params*: iterable de los parámetros de la red a optimizar.
- *lf*: tasa de aprendizaje (learning rate). Por defecto, su valor es 1e-3.
- *betas(Tuple[float,float])*: coeficientes utilizados para el cálculo de gradiente. Sus valores, por defecto, son $(0{,}9, 0{,}999)$.
- *eps*: valor que busca introducir estabilidad en el cálculo del gradiente. Por defecto, su valor es 1e-8.
- *weight_decay*: técnica de regularización L2. Por defecto, 0.
- *amsgrad*: si es cierto, se utiliza la variante AMSGrad [29] como optimizador.

Otras implementaciones populares que pueden encontrarse en PyTorch son: **SGD** (Stochastic Gradient Descent), **ASGD** (Average Stochastig Gradient Descent), **Adagrad**, **Rprop**, **RMSprop** [30]... Cada uno de ellos cuenta con sus propios parámetros, aunque muchos son compartidos.

En el caso de que se desee llevar a cabo una **inicialización de los pesos**, como es el nuestro, se cuenta con varias posibilidades. En este trabajo se ha hecho uso de la función **xavier_uniform_** (Inicialización de Xavier), que cuenta con dos parámetros: *tensor* que es el tensor donde se almacenará el resultado y *gain=1.0*, que se trata del factor multiplicativo visto en la Ecuación 4.1. Para llevar a cabo la incialización, se utiliza la función *apply(fn)* sobre todas las capas, donde *fn* es la función de inicialización. Otras opciones son: **xavier_normal**, **kaiming_normal**, **dirac_**, **sparse_**... Todas ellas pertenecen al paquete **torch.nn.init**.

Por último, para **gestionar** los datos de **entrada a la red**, se ha utilizado la clase **DataLoader** del paquete **torch.utils.data** [31], como se mencionó en el capítulo 4.3.2.

Capítulo 5

Resultados y API

En este capítulo, en primer lugar, se realiza una evaluación del rendimiento del modelo desarrollado, haciendo uso de varias métricas de distinta naturaleza. En segundo lugar, se detallan los pasos seguidos para el desarrollo de un servidor local, con el que poder interactuar con la red, pasándole una imagen de entrada y obteniendo el fichero reconstruido en un archivo formato HTML.

5.1. Resultados

El proceso de entrenamiento de la red fue bastante costoso, extendiéndose durante 34 días y un total de 275 horas de cómputo, a razón de 5h y 15 minutos, aproximadamente, por época. En total, en el conjunto de datos estaba integrado por 478 documentos (ver capítulo 4.1), que se repartieron en los siguientes porcentajes:

- **Entrenamiento**, 50 %. Un total de 239 imágenes, representadas en 65792 chunks.
- **Validación**, 20 %. Hasta 93 imágenes, dando lugar a 26112 chunks.
- **Test**, 30 %. Asciende a 146 imágenes, divididas en 39680 chunks.

En un primer momento, el entrenamiento se realizó en los servidores de *Kaggle*, por lo que los porcentajes de entrenamiento y validación fueron establecidos tratando de optimizar el tiempo máximo de ejecución continuada permitido: 9 horas. Tras un par de épocas, el entrenamiento se pasó a realizar en la máquina personal prestada por un conocido, en el que los tiempos de cómputo eran más rápidos, pero al partir del modelo ya entrenado en Kaggle, se mantuvieron los conjuntos de entrenamiento, validación y test ya utilizados.

En total, se realizaron 52 épocas de entrenamiento, obteniéndose los errores, como el error cuadrático medio, que se muestran en la Figura 5.1.

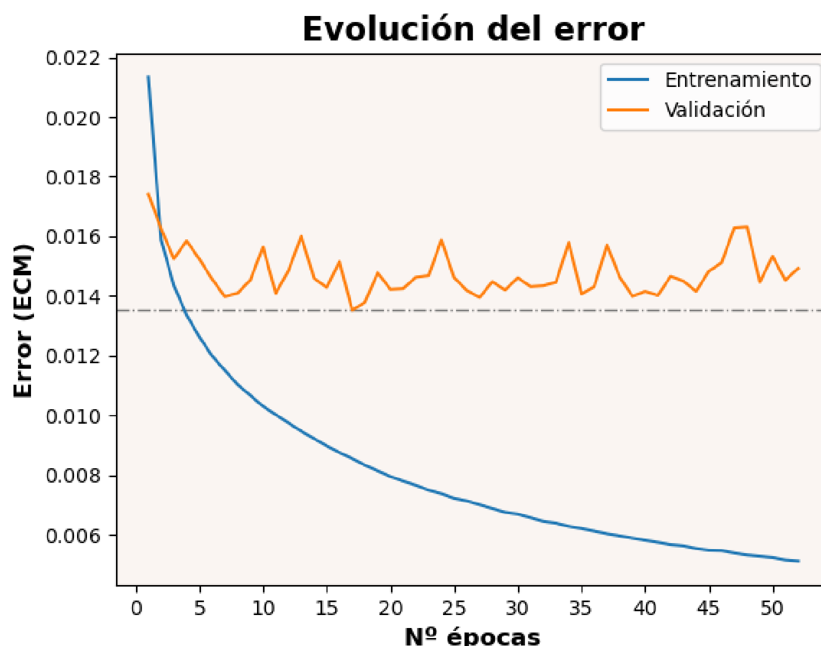


Figura 5.1: Evolución de las tasas de error a lo largo del entrenamiento.

Mientras que el error de entrenamiento disminuye de forma exponencial, el error en validación oscila en el intervalo [0.01352, 0.0174], alcanzándose este valor mínimo en la época número 16, como puede observarse en la Figura 5.1. Por ello, para los resultados que se muestran a continuación, se ha utilizado la configuración de la red obtenida en esta época, ya que se considera que es la óptima.

Como se mencionó en el capítulo 4.3.2, la salida de la red está compuesta por siete imágenes de idénticas dimensiones, que la tomada como entrada, siendo cada una ellas una clasificación de los pixels para cada categoría. De esta forma, los valores de los pixels representan su probabilidad de pertenencia a la categoría en cuestión. Para realizar una evaluación del rendimiento de la red, tenemos dos enfoques principales:

- **Evaluación continua**: compara la salida obtenida de la red, con la salida deseada. Se utilizan las probabilidades de los pixels, para valorar la similitud de la imagen resultado, con la imagen deseada.
- **Evaluación binaria**: Los pixels con un valor superior a un cierto umbral, o *threshold*, se clasifican como clase positiva, y los demás como negativa. A este proceso se le denomina binarización. Tras él, se calcula el porcentaje de pixels clasificados correcta e incorrectamente.

En la Figura 5.2, se muestra un ejemplo de salida de la red, y, en la Figura 5.3, su binarización con un threshold de 0.5. El orden de las imágenes, de izquierda a derecha y de arriba a abajo, es: original, *pie de foto*, *texto flotante*, *pie de página*, *títulos*, *imágenes*, *número de página* y *párrafos*.

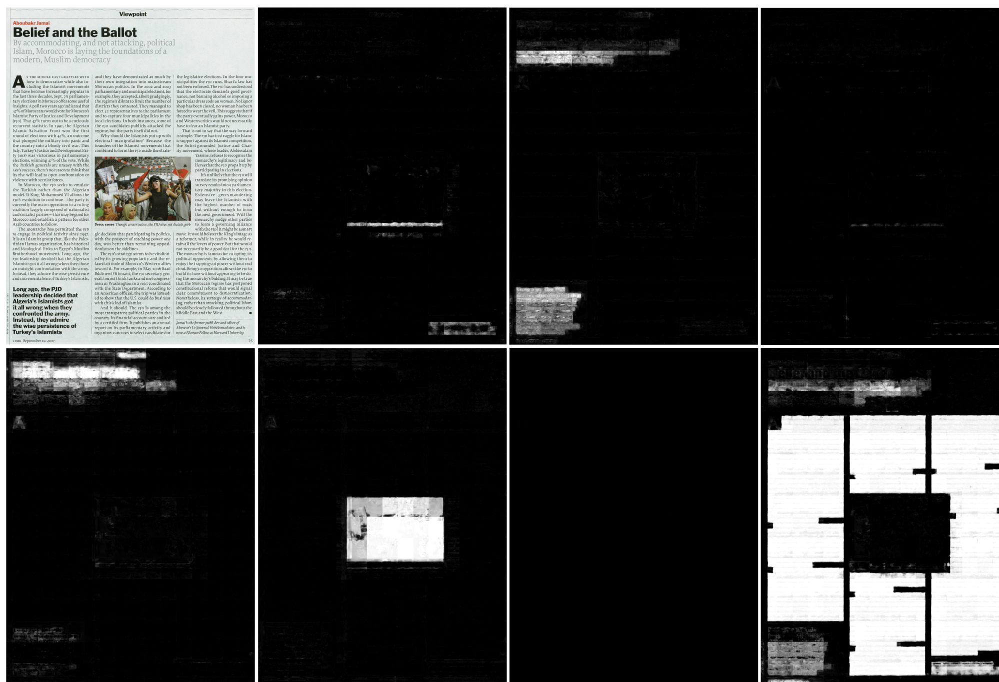


Figura 5.2: Salida obtenida por la red para una imagen dada.

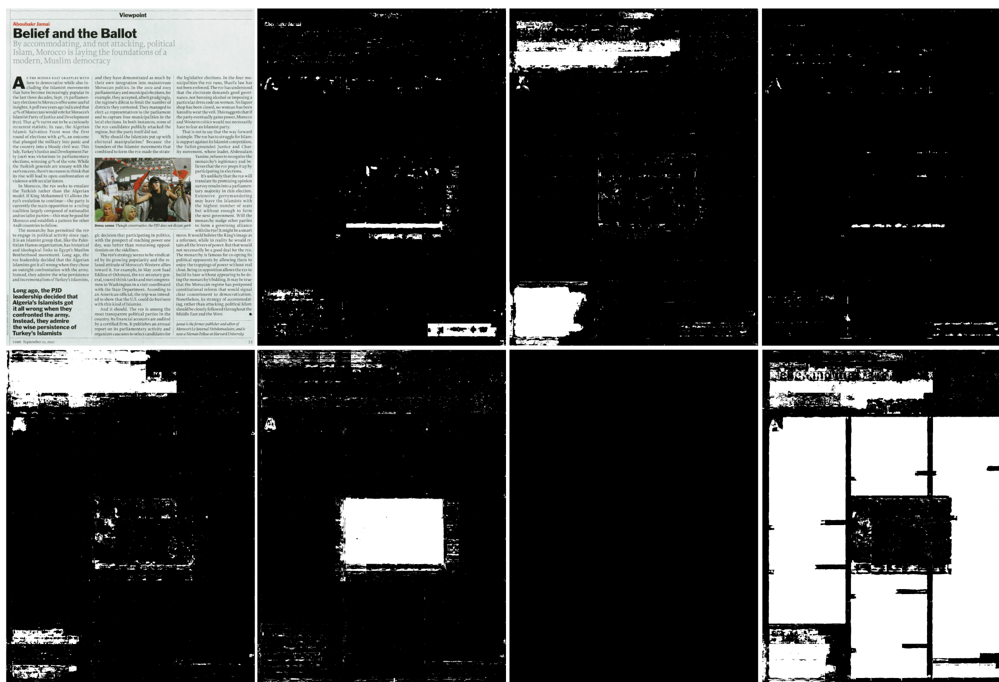


Figura 5.3: Binarización, con un threshold de 0.5, de la salida de la red mostrada en la Figura 5.2.

5.1.1. Evaluación caso continuo

Para este tipo de evaluación, también conocida como *fuzzy*, se han utilizado las métricas *MAE* y *RMSE*. La primera de ellas, fue vista en el capítulo 2.4.3. La segunda, es la raíz cuadrada de la métrica *MSE*, vista en ese mismo capítulo.

A continuación, en el Cuadro 5.1, se muestran los resultados obtenidos, desglosados por categorías, para dichas métricas.

Categoría	**MAE**	**RMSE**
Pie de foto	2.877099	4.715101
Texto flotante	3.512344	6.359271
Pie de página	0.849714	1.788237
Títulos	8.740358	7.411484
Imágenes	20.525129	7.757297
Nº página	0.013197	0.113618
Párrafos	65.246538	6.121016
Media	14.53777	4.895024

Cuadro 5.1: Evaluación continua por categoría.

La métrica MAE, al ser una puntuación lineal, pondera todas las diferencias individuales por igual. Como se puede observar en el Cuadro 5.1, la categoría con mayor valor para esta medida es *párrafos* y, la de menor, *número de página*. La diferencia entre ambos valores es muy elevado, y tiene su explicación: las zonas que se corresponden con párrafos, son mucho más amplias que las de número de página, por lo que la red no es tan precisa a la hora de delimitar la posición exacta. Por ello, los pixels colindantes a un párrafo, la red los detecta como posibles a considerar. Al ser zonas amplias y con alta frecuencia, termina desembocando en una métrica alta. Si nos fijamos en las categorías *imágenes* y *título*, podemos observar que ocurre lo mismo.

Por su parte, la métrica RMSE asigna un mayor peso a los errores más grandes. Como podemos ver en el Cuadro 5.1, la categoría *párrafos*, que era la que tenía, con diferencia, un mayor MAE, no es la que cuenta con un RMSE más elevado. Esto significa que los pixels colindantes a los que hacíamos mención, no son considerados con una alta probabilidad como párrafo, sino que sus probabilidades son relativamente bajas pero, al ser muchos, se obtiene una MAE elevada. Por el contrario, podemos destacar los valores de la categoría *texto flotante*, que tiene un RMSE superior al de párrafos, siendo su MAE casi 22 veces menor. La interpretación de este hecho, es que la estimación de esta categoría no es del todo precisa, ya que si lo fuese, los pixels colindates a dichas zonas tendrían valores menos elevados, dando lugar a una métrica más baja.

5.1.2. Evaluación del caso binario

Los valores asociados a cada pixel de las imágenes devueltas por la red pueden ser interpretados como probabilidades de pertenencia a la clase en cuestión. De esta forma, se puede considerar que todos los pixels, con un valor superior a un umbral, o threshold, son clasificados como instancias positivas y, el resto, como negativas. Así, es posible conocer el número de predicciones correctas e incorrectas hechas por la red.

Las métricas básicas para realizar este tipo de evaluación son:

- **True Positive (TP)**: ratio de instancias positivas, en nuestro caso pixels correspondientes a la categoría en cuestión, bien clasificadas por el modelo.
- **True Negative (TN)**: porcentaje de instancias negativas, en este caso pixels que no pertenecen a la categoría, correctamente clasificadas por la red.
- **False Positive (FP)**: cociente de instancias clasificadas como positivas cuando realmente son negativas. En este trabajo, es el porcentaje de los pixels clasificados incorrectamente dentro de una categoría.
- **False Negative (FN)**: tasa de instancias clasificadas incorrectamente como negativas. En nuestro caso, es la cuota de pixels pertenecientes a una categoría, pero que no fueron considerados como tal.

A partir de éstas, se pueden calcular otras métricas más avanzadas que permiten hacer una valoración más completa del rendimiento de la red:

- **Sensibilidad**: probabilidad de clasificar correctamente una instancia positiva. Se calcula como:

$$Sensibilidad = \frac{TP}{TP + FN}$$

- **Especificidad**: probabilidad de clasificar correctamente una instancia negativa. Su formulación es:

$$Especificidad = \frac{TN}{TN + FP}$$

- **Precisión**: porcentaje de instancias clasificadas correctamente.

$$Precision = TP + TN$$

Mediante los valores de la sensibilidad y la especificidad es posible construir las denominadas *curvas ROC*. Estas gráficas sirven para conocer el rendimiento global del clasificador, así como elegir el threshold adecuado. Se representa el valor de la sensibilidad, también denominado como *True Positive Rate (TPR)*, frente al *False Positive Rate(FPR)*, que es el complementario de la especificidad.

A continuación, para cada categoría, se muestran los resultados obtenidos para estas métricas, así como su curva ROC.

- **Pie de foto**:
  Como se puede observar en el Cuadro 5.2, existe un incremento muy elevado en la precisión a partir del umbral 0.1, alcanzándose en este punto el 98.53 % de instancias bien clasificadas. Si nos fijamos en la especificidad, entendemos que el aumento de precisión está provocado por una mejoría en la clasificación de las instancias negativas a partir de ese umbral.

Threshold	**Sensibilidad**	**Especificidad**	**Precisión**
0.0	0.95806	0.545648	0.547664
0.1	0.780848	0.984508	0.983513
0.2	0.702856	0.991156	0.989747
0.3	0.628333	0.994269	0.992481
0.4	0.535731	0.996448	0.994196
0.5	0.430591	0.997966	0.995193
0.6	0.312392	0.998988	0.995633
0.7	0.198046	0.999548	0.995631
0.8	0.095459	0.99984	0.995421
0.9	0.0343	0.999959	0.99524
1.0	0	1	0.995113

Cuadro 5.2: Métricas de evaluación binaria para la categoría *pie de foto*.

Es en el umbral 0.1, donde el modelo presenta una relación más balanceada entre su sensibilidad y especificidad, por lo que éste parece ser el *threshold* más adecuado. En este caso, la precisión es tan elevada, debido a la gran presencia de instancias negativas, lo que permite que el clasificador detecte sin problema la mayor parte de ellas, aunque le cueste detectar las positivas, la precisión apenas se ve afectada. Este hecho también se manifiesta en su curva ROC (ver Figura 5.4).

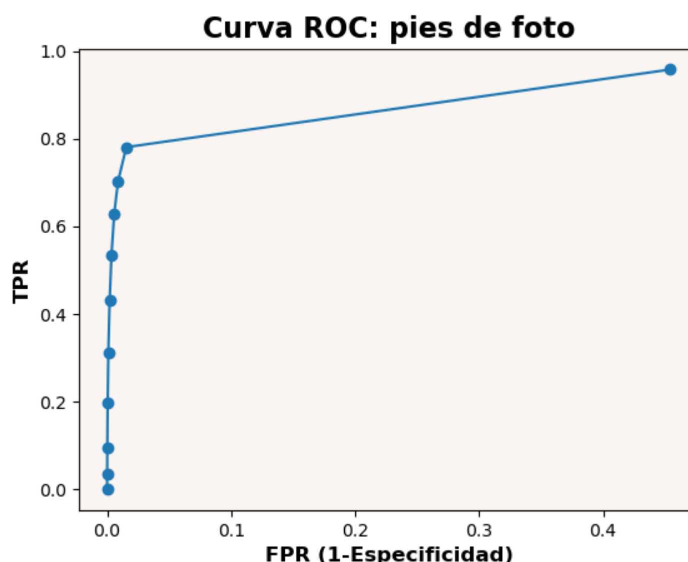


Figura 5.4: Curva ROC de la clase *pie de foto*.

- **Texto flotante**:
  Al igual que ocurría con categoría “pie de foto”, a partir del umbral 0.1 (ver Cuadro 5.3) existe una mejora significativa de la especificidad, con el correspondiente descenso en sensibilidad. A partir de un threshold de 0.3, el modelo clasifica las instancias positivas correctamente con una probabilidad menor a 0.5

Threshold	Sensibilidad	Especificidad	Precisión
0.0	0.905246	0.69813	0.700492
0.1	0.707701	0.98062	0.977507
0.2	0.60114	0.989978	0.985544
0.3	0.512286	0.993567	0.988078
0.4	0.427862	0.995722	0.989246
0.5	0.352378	0.997202	0.989848
0.6	0.272843	0.998327	0.990053
0.7	0.196101	0.999171	0.990013
0.8	0.121313	0.999702	0.989684
0.9	0.057749	0.999933	0.989187
1.0	0	1	0.988595

Cuadro 5.3: Métricas de evaluación binaria para la categoría *texto flotante*.

Su curva ROC, tiene una forma muy parecida a la categoría “pie de foto”, ya que tiene un comportamiento muy similar (ver Figura 5.3).

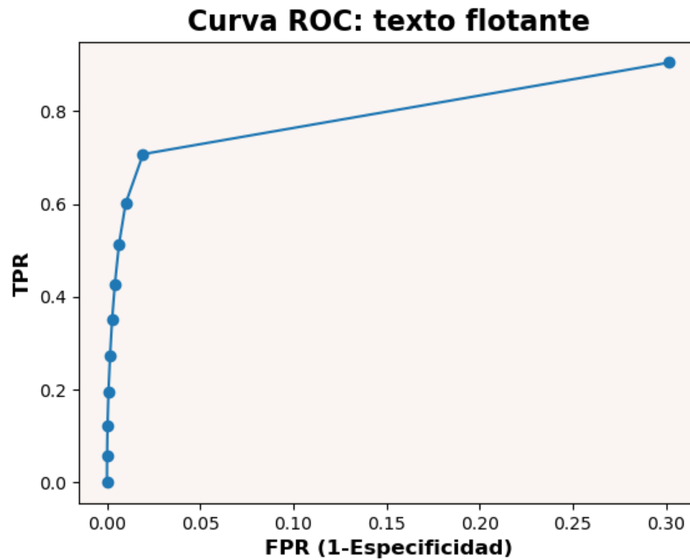


Figura 5.5: Curva ROC de la clase *texto flotante*.

- **Pie de página**:
  Como ocurría con las dos categorías previas, el paso del umbral 0.0 al 0.1 supone un gran incremento en la precisión del modelo. En este caso, si se consideran todas las instancias como positivas, la precisión es del 75 %. Sin embargo, existe un descenso muy significativo en la sensibilidad a partir del umbral 0.1, lo que indica que el modelo no es preciso en la identificación de esta clase. Los distintos valores de las métricas se muestran en el Cuadro 5.4 y su curva ROC se representa en la Figura 5.6.

Threshold	Sensibilidad	Especificidad	Precisión
0.0	0.920916	0.754601	0.754903
0.1	0.552079	0.997386	0.996579
0.2	0.412258	0.999239	0.998175
0.3	0.327896	0.9997	0.998482
0.4	0.256729	0.999885	0.998538
0.5	0.195128	0.999954	0.998496
0.6	0.136219	0.999984	0.998419
0.7	0.079607	0.999995	0.998327
0.8	0.027005	0.999999	0.998235
0.9	0.004413	1	0.998195
1.0	0	1	0.998187

Cuadro 5.4: Métricas de evaluación binaria para la categoría *pie de página*.

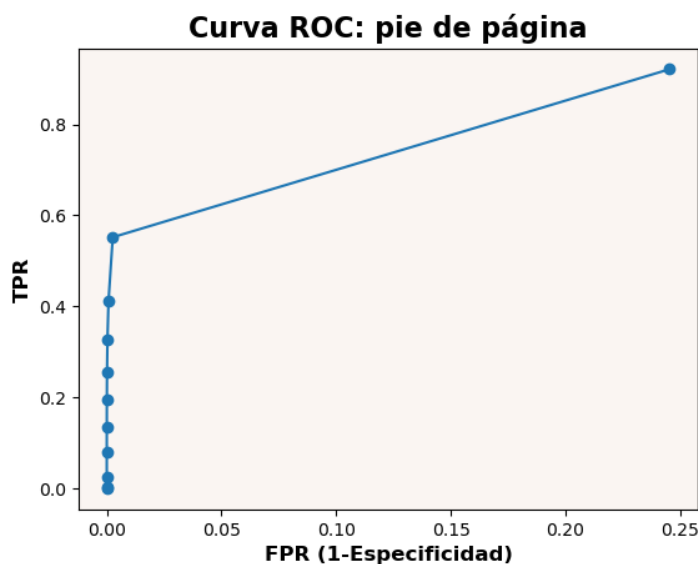


Figura 5.6: Curva ROC de la clase *pie de página*.

- **Títulos**:
  Existe un aumento muy significativo de la especificidad en el paso del threshold 0.0 al 0.1, que no afecta a la sensibilidad, manteniéndose por encima del 80 % hasta el umbral 0.4, como se puede ver en el Cuadro 5.5.

Threshold	Sensibilidad	Especificidad	Precisión
0.0	0.983235	0.505985	0.517457
0.1	0.910778	0.955027	0.953963
0.2	0.861782	0.970918	0.968295
0.3	0.81553	0.978301	0.974388
0.4	0.762581	0.983424	0.978116
0.5	0.707267	0.987151	0.980424
0.6	0.644166	0.990196	0.981879
0.7	0.575316	0.992619	0.982589
0.8	0.490743	0.994798	0.982682
0.9	0.376002	0.996679	0.98176
1.0	0	1	0.975963

Cuadro 5.5: Métricas de evaluación binaria para la categoría *títulos*.

En general, existe un mayor nivel de equilibrio entre la sensibilidad y la especificidad, en todos los umbrales, que en las categorías previas. Este hecho se refleja en la curva ROC (ver Figura 5.7), siendo todos sus puntos mucho más cercanos a la esquina superior izquierda, que indica un ajuste perfecto, que en el resto de categorías vistas.

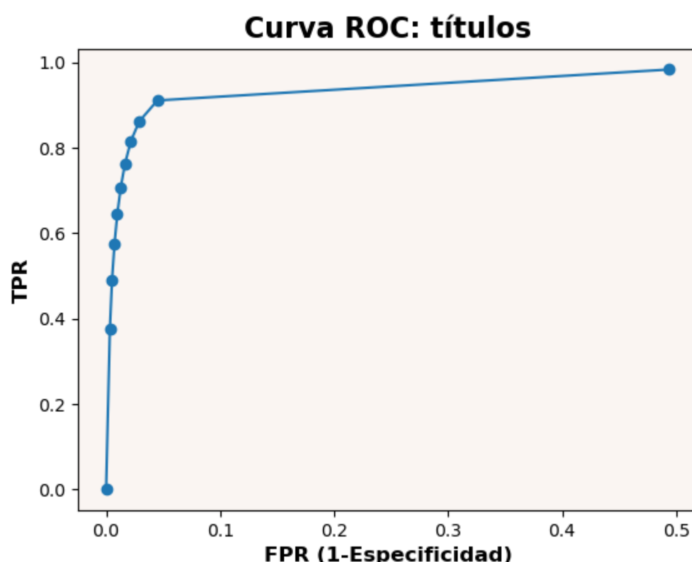


Figura 5.7: Curva ROC de la clase *títulos*.

- **Imágenes**:
  Como se puede observar en el Cuadro 5.6, el modelo presenta valores muy altos de especificidad en todos los umbrales manteniendo, a su vez, una alta sensibilidad. Se trata, de las categorías vistas hasta ahora, la que mejor es clasificada por la red neuronal.

Threshold	Sensibilidad	Especificidad	Precisión
0.0	0.983833	0.506459	0.562659
0.1	0.941818	0.958459	0.9565
0.2	0.919796	0.972939	0.966683
0.3	0.899201	0.97892	0.969535
0.4	0.877131	0.983346	0.970842
0.5	0.852684	0.986851	0.971056
0.6	0.821641	0.989575	0.969804
0.7	0.782405	0.991776	0.967127
0.8	0.723458	0.993947	0.962103
0.9	0.607242	0.995962	0.950199
1.0	0	1	0.882273

Cuadro 5.6: Métricas de evaluación binaria para la categoría *imágenes*.

Su curva ROC (ver Figura 5.8) difiere en forma de las anteriores, alojando bajo ella un área considerablemente mayor, dado que se trata de una categoría sobre la que la red es capaz de realizar una mejor clasificación.

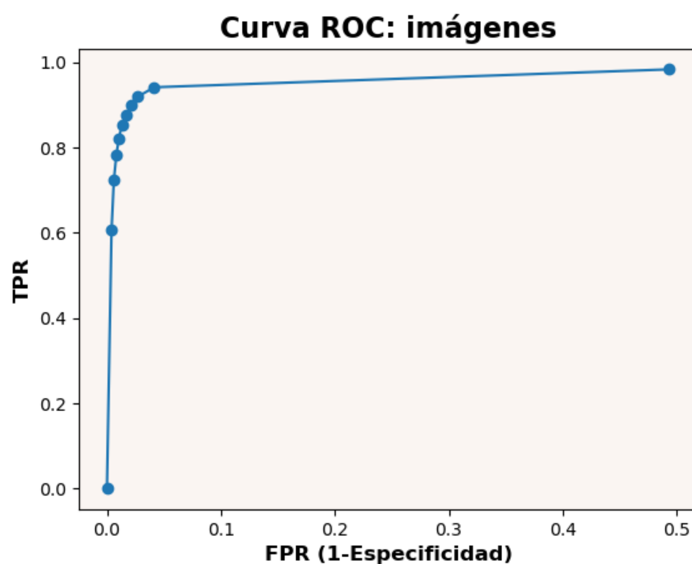


Figura 5.8: Curva ROC de la clase *imágenes*.

- **Número de página**:
  Presenta una precisión casi perfecta en todos los umbrales (ver Cuadro 5.7), pero su sensibilidad es nula en todos ellos menos en el 0.0. Esto significa que la red no ha aprendido a reconocer esta categoría. Dado que el número de pixels relacionados con esta clase es muy bajo, la precisión es muy elevada, ya que clasifica todos como instancias negativas.

Threshold	Sensibilidad	Especificidad	Precisión
0.0	0.605752	0.989563	0.989493
0.1	0	1	0.999816
0.2	0	1	0.999816
0.3	0	1	0.999816
0.4	0	1	0.999816
0.5	0	1	0.999816
0.6	0	1	0.999816
0.7	0	1	0.999816
0.8	0	1	0.999816
0.9	0	1	0.999816
1.0	0	1	0.999816

Cuadro 5.7: Métricas de evaluación binaria para la categoría *número de página*.

Su curva ROC realmente se trata de una recta, debido a su bajo rendimiento, tal y como puede verse en la Figura 5.9.

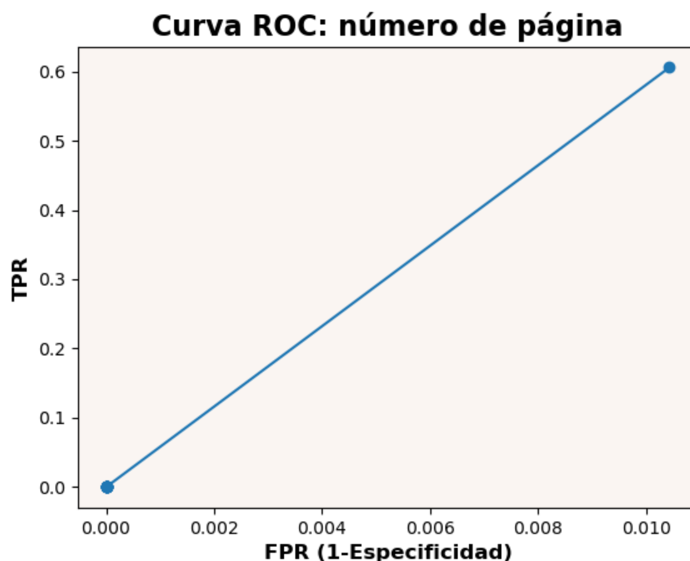


Figura 5.9: Curva ROC de la clase *número de página*.

- **Párrafos**:
  Se trata, con diferencia, de la categoría que mejor reconoce la red. Su sensibilidad (ver Cuadro 5.8), en todos los umbrales considerados, es muy elevada, siendo superior al 90 % en todos ellos menos en el 0.9 y 1.0. La especificidad también se encuentra por encima del 90 % en todos los valores del threshold, a excepción del 0.0.

Threshold	Sensibilidad	Especificidad	Precisión
0.0	0.995185	0.52455	0.701555
0.1	0.983787	0.920412	0.944247
0.2	0.977476	0.938538	0.953183
0.3	0.971788	0.947203	0.956449
0.4	0.965483	0.953583	0.958059
0.5	0.958942	0.958711	0.958798
0.6	0.951066	0.963467	0.958803
0.7	0.940847	0.967933	0.957746
0.8	0.923403	0.973278	0.95452
0.9	0.875079	0.981068	0.941205
1.0	0	1	0.6239

Cuadro 5.8: Métricas de evaluación binaria para la categoría *párrafos*.

Como se puede observar mediante su curva ROC (ver Figura 5.10), en la mayoría de los umbrales se tiene un rendimiento óptimo y muy similar, lo que provoca que el área bajo su curva sea mucho más grande que en el resto de categorías.

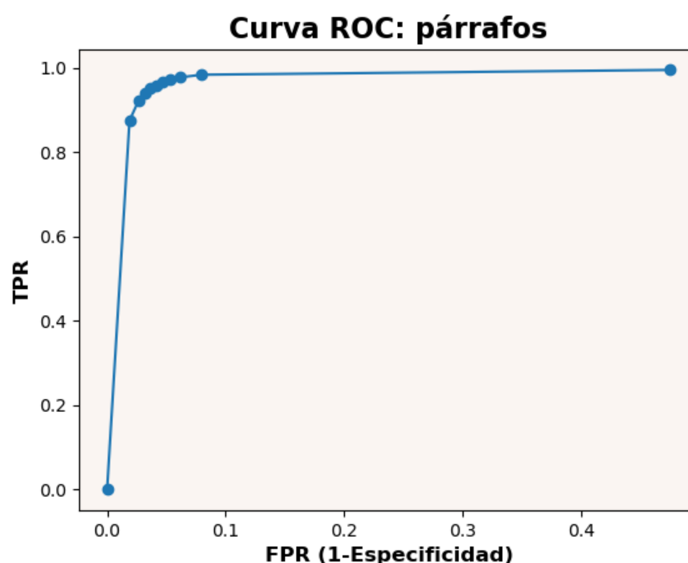


Figura 5.10: Curva ROC de la clase *párrafos*.

A través de estas métricas se ha podido observar la nula capacidad de reconocimiento de la red para la categoría *número de página*. así como el bajo rendimiento para las clases *pies de foto*, *pies de página* y *texto flotante*. Estas categorías se corresponden con las que menor presencia suelen tener en los documentos, por lo que, en parte, es esperable su peor rendimiento en ellas. Por el contrario, las categorías con más presencia son las mejor clasificadas siendo, en orden descendente, *párrafos*, *imágenes* y *títulos*. En el Cuadro 5.9 se muestra el threshold recomendado, para cada una de ellas.

Clase	**Threshold**
Pie de foto	0.1
Texto flotante	0.1
Pie de página	0.0
Títulos	0.1
Imágenes	0.1
Número de página	0.0
Párrafos	0.4

Cuadro 5.9: Valores de threshold recomendados para cada clase.

5.2. API

En esta sección, se expone el procedimiento de recomposición de las imágenes de salida de la red en un nuevo documento en formato HTML, así como la implementación de un servidor local que permite enviar a la red un documento escaneado y obtener el fichero HTML correspondiente.

5.2.1. Composición fichero HTML

Dado que la red es capaz de distinguir las categorías a las que pertenece cada zona de un documento, es posible utilizar esta información para recomponerlo en un formato distinto. En este trabajo se ha optado por *HTML*, aunque se valoraron otras alternativas como *PDF* o LaTeX.

Con el objetivo de encapsular todo el código necesario para integrar esta funcionalidad, se ha definido una clase denominada *htmlCreator()* que cuenta con un único método público que se encarga de generar el documento final. Esta clase cuenta con los siguientes atributos:

- *in*: ruta de la carpeta en la que se encuentra la imagen original.
- *gt*: ubicación donde están almacenadas las salidas obtenidas por la red para el documento original.
- *out*: lugar dónde se desea salvar los ficheros HTML y CSS, así como el resto de información necesaria.
- *max*: valor máximo de la dimensión mayor en el fichero generado. La segunda dimensión se establece de forma proporcional.

En dicha clase, se ha recurrido al uso de diversas bibliotecas para afrontar con garantías el proceso, entre las que destacan:

- **OpenCV**, permite llevar a cabo la binarización de las imágenes, así como el reconocimiento de los distintos clusters de pixels presentes en cada una de ellas.
- **Yattag**, facilita la creación de ficheros HTML y CSS gracias a su intuitiva gestión de etiquetas.
- **Pytesseract**, es un wrapper para Python del OCR Tesseract desarrollado por Google. En este trabajo, se utilizará para convertir a texto plano el contenido de las respectivas zonas del documento.

El primer paso en el proceso de recomposición es determinar las coordenadas de cada zona reconocida por la red. Con este fin, cada imagen obtenida es binarizada, utilizando su respectivo threshold óptimo (ver Cuadro 5.9), y pasada como argumento a la función *connectedComponentsWithStats* de la biblioteca OpenCV. Esta función, devuelve un listado con información sobre la ubicación en la imagen de cada uno de los clusters de pixels (nuestras zonas de interés) existentes. De esta forma, se obtiene un listado con las coordenadas de cada una de las zonas de interés junto a la categoría a la que corresponde.

Una vez generado el listado, se llama a una función interna encargada de escribir la base de los ficheros HTML y CSS correspondientes, para más tarde ir añadiendo sobre ellos la información correspondiente a cada zona reconocida.

Respecto al fichero HTML, se establece el título, haciéndolo coincidir con el nombre del documento procesado, y el encabezado, en el que se determina la ruta al fichero CSS que contiene su estilo. Para el archivo CSS, se establecen las dimensiones del cuerpo del HTML, y las características de cada una de las etiquetas que se van a utilizar: *p1* (párrafos, texto flotante y número de página), *p2* (pie de foto), *p3* (pie de página) y *h1* (títulos).

Tras ello, se recorren los distintos elementos de la lista previamente generada, y se va escribiendo en paralelo, en los fichero CSS y HTML, la información correspondiente. Para cada uno de los elementos, se establece su contenido en un *div* distinto, lo que facilita la composición final del documento ya que es más sencillo delimitar cada una de las zonas por medio de las características de cada componente *div*.

Se realiza una distinción entre las zonas que se corresponden con texto y las que lo hacen con imágenes. Para las primeras, se envía al OCR, proporcionado por pytesseract, el correspondiente recuadro de la imagen original, obteniéndose el texto en plano que figura en dicha zona, y escribiéndose en el fichero HTML bajo la etiqueta pertinente.

En el caso de las imágenes, en lugar de ser enviadas al OCR, se guarda en una carpeta el respectivo recorte y se escribe en el HTML la ruta correspondiente. Asimismo, se establecen los porcentajes de ancho y largo en los que se debe mostrar para conservar las proporciones con el resto de elementos.

En la Figura 5.11, se muestra una de las imágenes del dataset junto a su recreación en formato HTML, haciendo uso de su Ground Truth original.

LIVING IN THE PAST

"The vast majority of Europeans, be the

genome is disturbingly reminiscent of that of the fruit fly, and 99% of it all is shared with chimpanzees.

Somewhere amid what it says about our ancient past and possible future, our DNA conveys information about our more recent past—and perhaps the means to extend history into the realm of prehistory. And so it is that in the last decade, and at an increasing pace over the past few years, genetics has rejuvenated and somewhat confused the far older and dustier field of prehistoric archaeology. What had been a largely humanistic quest for first principles has been bolstered by complex statistical analyses of genetic evidence, allowing new voices to emerge from the long silence that makes up almost all of the history of *Homo sapiens sapiens*. In Europe alone, which manifests the least human genetic variation of any continent, genetic research is changing long-held notions of ethnic identity and origin. Among the most startling findings: some 80% of the gene pool of modern Europeans stems from ancestors who came to the Continent more than 11,000 years ago. The vast majority of Europeans, be they Italians or Swedes, whether they pride themselves on their aristocratic or their peasant origins, can trace their ancestry to just seven female lineages and as few as 10 male ones. Most of them came to Europe as Paleolithic hunter-gatherers, and far from being wiped out by the superior technologies later brought from the Middle East by Neolithic farmers, they might well have lived side by side with the newcomers for millennia.

Alongside the classical archaeologists unburying, dating and correlating physical artifacts, a growing number of molecular anthropologists—or archaeogeneticists—are working to reconstruct the genetic heritage of modern Europeans. The data from the digs and the labs don't always sit easily with one another, and when increasingly bold

SEVEN DAUGHTERS OF EVE

All About My Mother

Plowing through church records and birth rolls will only get you so far in reconstructing your family tree. For the really deep stuff, you must look to your genes, and Bryan Sykes, professor of human genetics at the University of Oxford, is there to help you. His company, Oxford Ancestors (motto: "We put the Genes in Genealogy"), can identify portions of your DNA that chronicle an unbroken chain of descent back to the Stone Age. All it takes is a swab from the inside of your cheek and $180.

The DNA in the mitochondria—cell organelles whose genetic makeup is determined by the mother alone—of all native Europeans except the Sami people of Northern Scandinavia reveal descent from one of what Sykes grandly calls "The Seven Daughters of Eve" (the title, incidentally, of his upcoming book to be published in June). He contends that these ancient matriarchs migrated to Europe from Africa via the Middle East or Asia as long as 50,000 years ago, and he has even assigned evocatively mythical names to each one: Ursula, Helena, Velda, Tara, Katrine, Xenia and Jasmine. He has also delineated rough population histories for the particular mutation patterns that define and date each of those ancestral groups.

Some of Ursula's people, for instance, were among the first modern humans to arrive in Europe, while others of her group came from the Middle East much later. The clan of Helena, the most common matrilineal ancestor of modern Europeans, is thought to have emerged from Ice Age refuges around the Pyrenees and migrated northward as the ice sheets began receding around 13,000 years ago. Velda's people similarly dispersed northward from the Iberian Peninsula, while Tara and Katrine's clans spread northward from present-day Italy. The descendants of Xenia are thought to have spread from an origin in the Caucasus not only westward into Europe, but also eastward and into the Americas. Beginning some 10,000 years ago, Jasmine's people brought agriculture with them from Syria.

Descendants of these seven lineages show up in all European ethnic groups, so correlating them to our modern notions of ethnicity is pointless. There is no evidence that differences in mitochondrial DNA track with any other inherited traits. Besides, there is the small matter of men, whose genetic contributions don't show up in mitochondrial DNA. In the coming months, Sykes expects to start offering codification of the prehistoric ancestral information imbedded in the Y chromosome. Other recent research suggests that there are 10 patrilineal lineages in Europe; Sykes says there are more, and he is likely to create names and histories for them as well.

As databases improve, Oxford Ancestors will be able to fill in the somewhat sketchier information for non-Europeans, for whom at least 30 further mitochondrial clans have been identified. In Africa, where human genetic variation is the highest, there are thought to be 14 of them. The noncombining elements of our genetic makeup strongly point to common ancestors there for all humans. Genealogy can't go any deeper than that. —J.G.

EVE BY LUCIEN LEVY-DHURMER; ARCHIVO ICONOGRAFICO, S.A./CORBIS

Prehistoric Eves: There may have been seven European ones

62 TIME, APRIL 30, 2001

Figura 5.11: Comparación de un documento escaneado con su recreación en formato HTML.

Tras comprobar el funcionamiento de la clase sobre las imágenes obtenidas por medio de la red, se observa que existen múltiples zonas que son reconocidas pero que realmente son carentes de valor. Esto se debe a que, como se vió en la Figura 5.3, existen muchos pixels aislados que son asignados a una categoría y por tanto considerados en el proceso de recomposición. Para solventar este problema, se establecen un área mínima, en función de la categoría, que un elemento debe superar para ser considerado en el proceso de recomposición.

Clase	Área mínima ($pixels^2$)
Pie de foto	20000
Texto flotante	10000
Pie de página	10000
Títulos	80000
Imágenes	300000
Número de página	750
Párrafos	100000

Cuadro 5.10: Área mínima para considerar un elemento.

Las áreas mínimas más grandes se corresponden con la categorías que se presuponen que sus elementos son más amplios, y son *párrafos*, *imágenes* y *títulos*. Por el contrario, para *número de página* tenemos con diferencia el valor más bajo, sin llegar a superar los 1000 $pixels^2$. Para el resto de categorías se establece un umbral similar. Estos valores han sido escogidos para optimizar el rendimiento sobre el conjunto de test, por lo que es posible que para otros conjuntos sea preciso una revisión.

5.2.2. Implementación del servidor

Como último punto del proyecto, se desarrolla un pequeño servidor en el que es posible procesar un documento escaneado, obteniéndose su recreación en formato HTML, mediante el proceso descrito en la sección anterior. Debido a la restricción temporal a la que está sujeta la realización de este TFG, se ha optado por un desarrollo web lo más sencillo posible, ya que su único fin es su utilización en la defensa del proyecto.

Siguiendo las recomendaciones de los tutores de *HP* se ha utilizado el framework *Flask*, de Python, para integrar la funcionalidad del servidor. Este marco destaca por presentar una curva de aprendizaje poco elevada, debido a que su estilo es muy similar al de Python. Asimismo, al hacer uso del motor *jinja2*, proporciona una forma cómoda y sencilla para el paso de variables entre el script en Python y los distintos ficheros HTML.

En el Capítulo 3.4, concretamente en la Figura 3.2, se mostró la estructura de directorios del proyecto, siendo de interés en este caso la rama *servidor*. Dentro de este directorio, se encuentran todos los ficheros necesarios para el funcionamiento del servidor.

Por un lado, se tiene el script de Python que alberga el código de las funcionalidades del servidor, haciendo uso del framework Flask previamente comentado. Por otro lado, se encuentran las carpetas *templates* y *static*:

- *templates*:
  Contiene el fichero HTML de la vista principal del servidor. Asimismo, en una subcarpeta denominada *results* se almacenan todos los ficheros HTML generados en las peticiones al servidor.

- *static*:
  Su contenido se divide en dos sub-directorios: *imag* y *styles*. En el primero de ellos, se guardan los recursos gráficos correspondientes a los distintos documentos procesados en el servidor. En el segundo, se albergan los ficheros de estilo de cada uno de los archivos HTML generados en el servidor.

La interfaz gráfica del servidor se ha hecho lo más minimalista posible, para que toda la atención se centre en el fin último para el que se ha desarrollado: el procesamiento

de documentos escaneados. De esta forma, la pantalla inicial se reduce a la composición mostrada en la Figura 5.12.

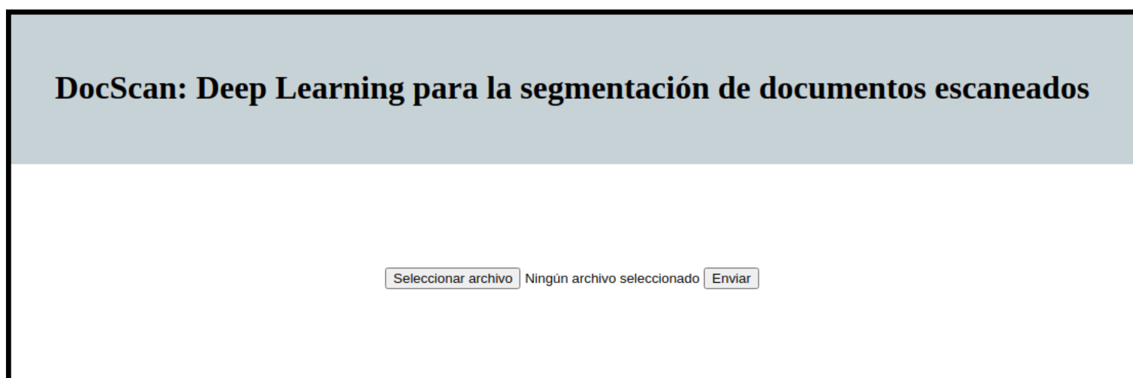


Figura 5.12: Pantalla principal del servidor.

Tras pulsar en el botón "Seleccionar archivo" se abre un explorador de archivos para elegir el documento que se quiere procesar. En este punto, es importante mencionar que únicamente se aceptan ficheros de cuatro formatos: **tif**, **npg**, **jpg** y **jpeg**.

Una vez elegido el archivo que se desea procesar, tras pulsar el botón "Enviar" pueden darse dos situaciones: si el fichero elegido de un formato válido, el servidor muestra un mensaje de información al usuario, como el que se muestra en la Figura 5.13; en caso contrario, la elección del usuario queda invalidada.

Figura 5.13: Pantalla de espera del servidor.

Si el fichero ha sido aceptado, se envía a la red neuronal entrenada, obteniéndose de esta manera la clasificación de las distintas zonas del documento. Tras ello, las siete imágenes generadas son tratadas por un objeto de la clase descrita en la sección anterior, que se encarga de recomponer el fichero en formato HTML, adecuándose a la estructura de contenidos del servidor. Una vez terminada la espera, el servidor carga en pantalla el fichero obtenido.

Asimismo, es posible ver los resultados obtenidos para ficheros previamente procesados, indicando la ruta pertinente en el servidor. Los detalles sobre cómo conocer la ruta a ficheros ya procesados se detallan en el *Manual del Usuario* (ver Apéndice B).

Capítulo 6

Conclusiones

Durante mi formación universitaria siempre he sentido un especial interés por el Aprendizaje Profundo, incrementándose aún más tras cursar las asignaturas de *Técnicas de Aprendizaje Automático* y *Minería de Datos*. Debido a las restricciones de contenido a las que están sujetas las asignaturas, no se profundizó el conocimiento sobre este área tanto como me hubiese gustado. Por ello, y teniendo en cuenta la importancia de este campo en la actualidad, he tenido en este TFG la oportunidad perfecta para adquirir un mayor grado de conocimiento sobre el Aprendizaje Profundo.

Asimismo, el problema propuesto también ha supuesto un reto en lo personal ya que como estudiante me he encontrado en múltiples ocasiones con documentos escaneados que me han obligado a hacer una lectura más concreta de lo deseado para localizar la información buscada. A pesar de que no se perseguía generar una herramienta con un alto nivel de precisión, afrontar esta problemática me ha permitido adquirir conocimientos útiles y extrapolables a otros desafíos que me pueda encontrar en mi vida profesional.

De igual manera, la metodología empleada ha supuesto para mí un gran aprendizaje ya que durante mi formación siempre había utilizado enfoques mucho más cercanos a las metodologías tradicionales. A pesar de que no se ha trabajado con una metodología estrictamente ágil, sí que me ha obligado a salir de mi zona de confort y tener que hacer frente a un método de trabajo más similar a los empleados fuera del ámbito académico.

El proyecto contaba con tres grandes objetivos fijados, que se han alcanzado con éxito. El primero de ellos, era la creación o adecuación de un conjunto de datos para el problema planteado. Concretamente, se ha tenido que llevar a cabo un proceso de generación del Ground Truth, algo novedoso para mí, ya que siempre había trabajado con datos ya preparados. Para cumplir con este objetivo, he tenido que trabajar con estructuras de datos en formato *XML*, vista en la asignatura *Sistemas Avanzados de Integración de la Información*, sobre las que he tenido que profundizar en conocimientos.

El segundo objetivo, era realizar un correcto pre-procesamiento de las imágenes para

poder ser utilizadas por la red. Aquí destacan los algoritmos de chopping de imágenes, de los cuáles no tenía constancia hasta el momento de realización de este trabajo. Se optó por el uso del algoritmo recursivo, y a pesar de contar con la ayuda de los tutores, fue un proceso algo complicado ya que a penas existe literatura al respecto. De esta forma, el plazo de finalización de esta parte del proyecto se alargó más de lo deseado, pero finalmente se alcanzó con éxito el objetivo.

Tras completar los dos primeros objetivos, se hizo frente al objetivo principal del proyecto: el desarrollo de una red neuronal para segmentar documentos escaneados. La arquitectura de red utilizada, UNET, me ha permitido poner en práctica los conocimientos aprendidos en las asignaturas de *Aprendizaje Automático* y *Minería de Datos*, así como conocer de primera mano la operación de upsampling y las posibles formas de llevarla a cabo, optándose por la técnica de pixel shuffle. De igual manera, su realización haciendo uso de PyTorch, ha sido muy fructífera desde el punto de vista personal, ya que era un framework con el que no había trabajado y que amplía mis conocimientos en este área.

Se propuso la identificación de hasta siete clases en los documentos, obteniéndose resultados diversos para cada una de ellas. Las categorías con zonas más pequeñas y menos frecuentes, son las que peor rendimiento presentan. En algunos casos, como el de la clase "número de página", la red reconoce todas las zonas de los documentos como no pertenecientes a dicha clase, por lo que no se alcanza el fin perseguido. Por el contrario, existe un rendimiento elevado para las categorías "párrafos" e "imágenes", presentando un alto nivel de sensibilidad en la mayoría de los thresholds estudiados.

Para el resto de categorías, los resultados son bastante similares entre sí, presentando una alta sensibilidad para el umbral 0.0, junto a una especificidad en el intervalo [0.50, 0.70]. Para valores del threshold superiores, la sensibilidad se ve claramente afectada, lo que indica que la red no es capaz de llevar a cabo una buena clasificación.

Por tanto, de las siete clases propuestas, se ha conseguido un nivel de precisión muy elevado para "párrafos", "imágenes" y, en menor medida, "títulos", mientras que los resultados para "'pie de página", "pie de foto", "texto flotante" y "número de página" no son óptimos.

Tras obtener los resultados de la red, se propone la reconstrucción del documento en un nuevo formato, en este caso HTML. Durante el desarrollo de este proceso, he podido profundizar en el uso de la biblioteca OpenCV, una de las más usadas para el tratamiento de imágenes, y he tenido un primer contacto con los OCR, concretamente la versión Tesseract de Google. Asimismo, para facilitar al usuario la obtención del documento reconstruido, se ha desarrollado un pequeño servidor local haciendo uso de *Flask*.

Evidentemente, tras este proyecto quedan abiertas muchas líneas de trabajo, las cuales no han podido abordarse debido a la restricción temporal. Algunas de ellas son:

- Mejorar el algoritmo recursivo de chopping para generar chunks con zonas útiles más amplias, a costa de reducir el excedente de los márgenes.
- Considerar otras configuraciones de la red UNET, en busca de la mejora de los resultados obtenidos para las categorías con zonas de menor tamaño y presencia, así como explorar otras opciones para las operaciones de downsampling y upsampling.
- Mejorar la precisión en el proceso de reconstrucción de los documentos en formato HMTL.
- Implementación de un Docker en el que integrar tanto la funcionalidad de la red como la del servidor.

Apéndice A

Manual de instalación

En este anexo se presentan los pasos a seguir para llevar a cabo la instalación y configuración de la aplicación de la red neuronal y el servidor web.

Los requisitos básicos a satisfacer para la instalación son:

- Contar con un entorno de trabajo en Python.
- Respetar la organización de directorios mostrada en la Figura 3.2.
- Instalar PyTorch, OpenCV y Flask, así como sus dependencias.

Para el entorno Python, se recomienda el uso de distribuciones como Anaconda o Miniconda, siendo la primera la utilizada en el desarrollo del trabajo. Se puede instalar desde:

https://www.anaconda.com/products/individual

La versión de Python empleada es la 3.8.5, por lo que se recomienda su utilización para evitar posibles problemas de incompatibilidades.

Respecto a la organización de los directorios, la carpeta comprimida adjunta a este informe contiene la estructura requerida para el correcto funcionamiento, por lo que únicamente requiere de su descompresión.

Para la instalación de las bibliotecas y sus dependencias, haciendo uso de la distribución Anaconda, los pasos son:

- PyTorch (versión 1.7.1):
```
conda install pytorch torchvision torchaudio cudatoolkit=11.0
-c pytorch
```
- OpenCV (versión 4.5.1):
```
conda install -c conda-forge opencv
```
- Flask (versión 1.1.2):
```
conda install -c anaconda flask
```
- tqdm (versión 4.56.0):
```
conda install -c conda-forge tqdm
```
- docopt (versión 0.6.2):
```
conda install -c anaconda docopt
```
- pytesseract (versión 0.3.7):
```
conda install -c conda-forge pytesseract
```
- ninja (versión 1.10.2):
```
conda install -c conda-forge ninja
```

Apéndice B

Manual de usuario

En este anexo se pretende proporcionar a los usuarios el conocimiento necesario para la utilización de la aplicación de la red neuronal y el servidor web

Preparación de los datos

El conjunto de datos utilizado no puede ser redistribuido, por lo que para su utilización es necesario solicitar su descarga a PRImA [1].

Para generar el Ground Truth, como se explicó en el Capítulo 4.2.1, se proporciona el script "generateGroundTruth.py". Una vez descargado y descomprimido el dataset, basta con ejecutar dicho script para generar el Ground Truth de los distintos documentos.

Para obtener los conjuntos de entrenamiento, validación y test, se proporciona el script "splitImages.py". La configuración es: 50 % para entrenamiento, 20 % para validación y el 30 % para el test. Estos porcentajes pueden ser modificados en el script.

Por último, mencionar que ambos scripts están preparados para ser ejecutados desde la ruta de la carpeta "procesamiento", teniendo los datos en el directorio raíz (ver Figura 3.2). Si se desean ejecutar con otra disposición, es necesario modificar las rutas a los directorios utilizadas en los scripts.

Red Neuronal

Como se detalló en el Capítulo 4.3.2, la red neuronal cuenta con tres modos de utilización diferentes: entrenamiento, validación y test.

Respetando la configuración de directorios mostrada en la Figura 3.2, los comandos para cada uno de estos modos, son:

- **Entrenamiento**:

```
python Main.py --train --in=<input_source>
--gt=<groundtruth_source> --inv=<valid_input_source>
--gtv=<valid_groundtruth_source> --e=<max_epoch>
```

  Como parámetro opcional se cuenta con --ld=<pretrained_model> en el caso de que se desee reanudar el entrenamiento desde un punto anterior.

- **Validación**:

```
python Main.py --validate --ld=<pretrained_model>
--in=<input_source> --gt=<groundtruth_source>
```

- **Test**:

```
python Main.py --test --ld=<pretrained_model>
--in=<test_source>
```

  Si se desea especificar una ruta concreta para guardar el resultado obtenido por la red, se puede utilizar el parámetro --out=<output_source>.

Asimismo, los tres métodos cuentan con la opción de ser ejecutados bajo la CPU si así se especifica mediante el uso del parámetro --cpu.

Servidor web

El primer paso es lanzar el servidor. Para ello, estando situado en el directorio raíz de la Figura 3.2, se deben ejecutar los siguientes comandos en una terminal Anaconda:

1. export FLASK_APP='./servidor/app'.
2. export FLASK_ENV=development.
3. flask run.

Una vez el servidor está en ejecución, se puede acceder a él, desde la propia máquina, dirigiéndose a la dirección web:

http://127.0.0.1:5000/

En dicha dirección, se cargará la interfaz en el servidor, que fue mostrada en la Figura 5.12, dónde podremos subir la imagen del documento que se desee procesar, siempre teniendo en cuenta que sólo acepta los formatos: *tif*, *npg*, *jpg* y *jpeg*.

Tras elegir una imagen de un formato adecuado y haber pulsado el botón de "Enviar", comienza el procesamiento de la misma y el servidor muestra la pantalla de espera de la Figura 5.13. Una vez finalizado el proceso, automáticamente se carga su recomposición en formato HTML.

En la dirección `http://127.0.0.1:5000/servidor/<image_name>.html`, se muestra el resultado generado, siendo `<image_name>` el nombre de la imagen enviada. Haciendo uso de esta ruta es posible visualizar resultados obtenidos previamente con el servidor, sin tener que volver a realizar el procesamiento.

Bibliografía

[1] A. Antonacopoulos, D. Bridson, C. Papadopoulos y S. Pletschacher, "A Realistic Dataset for Performance Evaluation of Document Layout Analysis", en *2009 10th International Conference on Document Analysis and Recognition*, 2009, págs. 296-300. DOI: `10.1109/ICDAR.2009.271`.

[2] W. Mcculloch y W. Pitts, "A Logical Calculus of Ideas Immanent in Nervous Activity", *Bulletin of Mathematical Biophysics*, vol. 5, págs. 127-147, 1943.

[3] P. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Harvard University, 1975. dirección: `https://books.google.es/books?id=z81XmgEACAAJ`.

[4] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position", *Biological Cybernetics*, vol. 36, n.º 4, págs. 193-202, abr. de 1980. DOI: `10.1007/bf00344251`. dirección: `https://doi.org/10.1007/bf00344251`.

[5] D. H. Hubel y T. N. Wiesel, "Receptive fields of single neurones in the cat's striate cortex", *The Journal of Physiology*, vol. 148, n.º 3, págs. 574-591, oct. de 1959. DOI: `10.1113/jphysiol.1959.sp006308`. dirección: `https://doi.org/10.1113/jphysiol.1959.sp006308`.

[6] Y. LeCun, P. Haffner, L. Bottou e Y. Bengio, "Object Recognition with Gradient-Based Learning", en *Shape, Contour and Grouping in Computer Vision*, Springer Berlin Heidelberg, 1999, págs. 319-345. DOI: `10.1007/3-540-46805-6_19`.

[7] I. Goodfellow, Y. Bengio y A. Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[8] J. Kumar, *Types of padding in convolutional layers*, `https://www.geeksforgeeks.org/types-of-padding-in-convolution-layer/`, 2019.

[9] O. Ronneberger, P. Fischer y T. Brox, *U-Net: Convolutional Networks for Biomedical Image Segmentation*, 2015. arXiv: `1505.04597 [cs.CV]`.

[10] D. Mishra, *Transposed Convolution*, `https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba`, 2020.

[11] Wikipedia, *Bilinear Interpolation*, `https://en.wikipedia.org/wiki/Bilinear_interpolation`.

[12] W. Shi, J. Caballero, F. Huszár y col., *Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network*, 2016. arXiv: 1609.05158 [cs.CV].

[13] S. Sahoo, *Residual blocks - Building blocks of ResNet*, https://towardsdatascience.com/residual-blocks-building-blocks-of-resnet-fd90ca15d6ec, 2018.

[14] A. Bindal, *Normalization Techniques in Deep Neural Networks*, https://medium.com/techspace-usict/normalization-techniques-in-deep-neural-networks-9121bf100d8, 2019.

[15] S. Ioffe y C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", *CoRR*, vol. abs/1502.03167, 2015. arXiv: 1502.03167. dirección: http://arxiv.org/abs/1502.03167.

[16] D. Ulyanov, A. Vedaldi y V. S. Lempitsky, "Instance Normalization: The Missing Ingredient for Fast Stylization", *CoRR*, vol. abs/1607.08022, 2016. arXiv: 1607.08022. dirección: http://arxiv.org/abs/1607.08022.

[17] H. S, *Activation Functions: Sigmoid, tanh, ReLU, Leaky ReLU, PReLU, ELU, Threshold ReLU and Softmax basics for Neural Networks and Deep Learning*, https://himanshuxd.medium.com/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c70eed91e, 2019.

[18] PyTorch, *SmoothL1Loss*, https://pytorch.org/docs/stable/generated/torch.nn.SmoothL1Loss.html.

[19] K. Beck, M. Beedle, A. van Bennekum y col., *Manifesto for Agile Software Development*, 2001. dirección: http://www.agilemanifesto.org/.

[20] GitLab, https://about.gitlab.com/.

[21] P. R. I. A. Research, *Layout Analysis Dataset*, https://www.primaresearch.org/dataset/index.php.

[22] P. Migdal y R. Jakubanis, *Keras or PyTorch as your first depp learning framework*, https://deepsense.ai/keras-or-pytorch/.

[23] J. Brownlee, *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning*, https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

[24] X. Glorot e Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks", en *AISTATS*, 2010.

[25] *Pytorch Documentation: Module*, https://pytorch.org/docs/stable/generated/torch.nn.Module.html.

[26] *Pytorch Documentation: CONV2D*, https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html.

[27] @fvisin, *Convolution arithmetic*, https://github.com/vdumoulin/conv_arithmetic/blob/master/README.md.

[28] *Pytorch Documentation: INSTANCENORM2D*, https://pytorch.org/docs/stable/generated/torch.nn.InstanceNorm2d.html.

[29] S. J. Reddi, S. Kale y S. Kumar, "On the Convergence of Adam and Beyond", en *International Conference on Learning Representations*, 2018. dirección: https://openreview.net/forum?id=ryQu7f-RZ.

[30] *Pytorch Documentation: TORCH.OPTIM*, https://pytorch.org/docs/stable/optim.html.

[31] *Pytorch Documentation: TORCH.UTILS.DATA*, https://pytorch.org/docs/stable/data.html.