Escuela
Politécnica
Superior

Recuperación de Grafos para Optical Music Recognition

Máster Universitario en Ciencia de Datos

Trabajo Fin de Máster

Autor:
Carlos Garrido Muñoz
Tutor/es:
Jorge Calvo Zaragoza
Antonio Ríos Vila

Junio 2022

Universitat d'Alacant
Universidad de Alicante

Recuperación de Grafos para Optical Music Recognition

Recuperación de Grafos para Optical Music Recognition

Autor
Carlos Garrido Muñoz

Tutor/es
Jorge Calvo Zaragoza
Departamento de Lenguajes y Sistemas Informáticos
Antonio Ríos Vila
Departamento de Lenguajes y Sistemas Informáticos

Máster Universitario en Ciencia de Datos

Escuela
Politécnica
Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, Junio 2022

Agradecimientos

Son muchas las páginas que tendría que escribir para expresar a toda la gente que quiero lo agradecido que estoy. La gente que me rodea sabe que ha sido un año complicado. A pesar de ello, siento que he crecido exponencialmente con este año en todos los aspectos de mi vida. Soy un verdadero afortunado por todas las oportunidades que tengo. Me gustaría dedicar este trabajo a esas personas que han estado apoyándome en cada momento.

A mi familia, por su apoyo incondicional, ya que sin ellos nada de esto sería posible.

Para mis amistades, que siempre han estado ahí sabiéndome sacar de las largas semanas de trabajo, escuchando mis inquietudes y compartiendo momentos inolvidables.

Al grupo del "Lab", que hacéis muchísimos más amenos y complacientes los largos días de trabajo. Espero que podamos compartir muchos más momentos juntos (sobre todo fuera del laboratorio).

A Antonio, que a pesar de sus largas y dolorosas correcciones, estoy tremendamente agradecido por todo lo que estoy aprendiendo.

A Jorge, tutor académico y mentor, gracias por introducirme en este mundo hace apenas dos años. Espero, como mínimo, poder devolver la confianza depositada en mí en forma de muchos éxitos juntos.

Resumen

En un mundo en el que la información nos rodea, tratarla de manera adecuada cobra vital importancia. Con el auge de las técnicas de *Machine Learning*, extraer valor de este preciado recurso supone un avance en muchos ámbitos. Uno de ellos que puede beneficiarse gracias a estas técnicas es el cultural, en concreto, el musical.

Este patrimonio se ha transmitido siempre en forma de partituras, describiéndose por medio de la notación musical. Gracias a los avances tecnológicos, existen sistemas que permiten digitalizar estas partituras para interpretar; exportar a distintos formatos; editar, o simplemente preservar esta información. En concreto, el área de investigación que se encarga de leer e interpretar la notación musical es el del *Optical Music Recognition* (OMR).

A pesar de los grandes avances en este campo, la investigación actual se centra en la transcripción secuencial de partituras, olvidando la importancia de generar formatos que permitan a un ordenador interpretar la notación musical de manera adecuada. En busca de mejorar esta interpretabilidad y, dada la naturaleza relacional de la música, algunos autores proponen que esta notación puede ser representada por un grafo. En dicha estructura, los símbolos musicales componen los nodos y las relaciones entre ellos, las aristas. Desafortunadamente, no existe casi investigación al respecto sobre cómo extraer grafos que describan esta notación.

En este trabajo de fin de máster, se aborda este hueco en la literatura y se estudia cómo extraer grafos a partir de partituras musicales. Para ello, se plantea el problema desde tres perspectivas distintas. En la primera, se investiga la posibilidad de extraer únicamente las relaciones entre símbolos; en la segunda, se propone detectar los símbolos musicales (nodos) en extractos de partituras con sistemas que no utilicen detección de objetos; finalmente, se propone un método que permite sacar grafos de manera holística a partir de extractos de partituras. Los resultados obtenidos en estos trabajos sugieren que es posible recuperar relaciones y grafos completos capaces de representar la notación musical con precisión.

Daría todo lo que sé por la mitad de lo que ignoro.

René Descartes.

Contents

1 Introducción **1**
1.1 Objetivos . . . 2

2 Estado del arte **5**
2.1 Optical Music Recognition . . . 5
2.2 Predicción de grafos . . . 6

3 Marco teórico **7**
3.1 Introducción . . . 7
3.2 Inteligencia Artificial. Definición y concepto . . . 7
3.3 El proceso de aprendizaje . . . 8
3.4 *Machine Learning*: extrayendo patrones . . . 9
3.4.1 Tipos de aprendizaje . . . 9
3.5 *Deep Learning* . . . 10
3.5.1 Cibernética: El perceptrón simple y la Neurona Artificial . . . 11
3.5.2 Conexionismo: Redes Neuronales y *Backpropagation* . . . 14
3.5.3 *Deep Learning* moderno. Aspectos específicos . . . 20

4 Materiales y herramientas **31**
4.1 Datos utilizados . . . 31
4.2 Herramientas . . . 32
4.2.1 Desarrollo del software . . . 32
4.2.2 Hardware utilizado . . . 34

5 Metodología y experimentación **35**
5.1 Recuperación de relaciones . . . 35
5.1.1 Metodología . . . 35
5.1.2 Experimentos . . . 37
5.1.3 Resultados . . . 39
5.1.4 Conclusión . . . 43
5.2 Predicción de nodos/símbolos . . . 45
5.2.1 Metodología . . . 45
5.2.2 Experimentos . . . 47
5.2.3 Resultados . . . 50
5.2.4 Conclusión . . . 52
5.3 Recuperación completa de grafos . . . 54
5.3.1 Metodología . . . 54
5.3.2 Experimentos . . . 57
5.3.3 Resultados . . . 61

5.3.4 Conclusión 63

6 Conclusiones **67**
6.1 Trabajos futuros 67
6.1.1 Recuperación de relaciones 68
6.1.2 Predicción de nodos/símbolos 68
6.1.3 Recuperación completa de grafos 68

Bibliography **71**

Acrónimos y lista de abreviaciones **75**

List of Figures

3.1 Diagrama de Venn de los campos y subcampos de la IA (GoodFellow et al., 2015). 7
3.2 Esquema de dos neuronas biológicas. Fuente: psicoactiva.com 11
3.3 Figura del perceptrón simple. Fuente: wikipedia.org 12
3.4 Figura del perceptrón simple con el sesgo añadido como peso w_0. Fuente: creación propia con el paquete TikZ 12
3.5 Problema de las puertas lógicas. Fuente: quora.com 14
3.6 *Multilayer Perceptron* con dos neuronas por capa. Fuente: creación propia con TikZ 15
3.7 Neurona artificial con función de activación genérica. Fuente: datacamp.com 16
3.8 Funciones de activación comunes. Fuente: kaggle.com 17
3.9 Esquema general de una red neuronal. Fuente: cs.swarthmore.edu 18
3.10 Ejemplo de operación de convolución. Fuente: researchgate.net 22
3.11 Ejemplo de operaciones de convolución con varios *kernels*. Fuente: towards-datascience.com 22
3.12 Ejemplo de extracción de características con filtro de Sobel sobre una imagen de Albert Einstein. Fuente: cs.toronto.edu 23
3.13 Ejemplo de *Max pooling* y *Average Pooling*. Fuente: towardsdatascience.com . 23
3.14 Arquitectura LeNet-5 por Yann LeCun, 1989. Fuente: researchgate.net 24
3.15 Ejemplo de dígitos manuscritos de la base de datos del *MNIST*. 24
3.16 Grafo computacional completo de una unidad en una LSTM. Fuente: Jurafsky & Martin (2009) 28
3.17 Representación en forma de *embedding* en un eje de coordenadas de dos dimensiones. Eje X: Público al que va dirigida la película. Eje Y: Representación de la película en función de lo taquillera o lo artístico de ella. Fuente: developers.google.com/machine-learning/ 30

4.1 Ejemplos de distintas partituras musicales extraídas de la MUSCIMA++. Las figuras están redimensionadas para ocupar anchuras y alturas similares. Sin embargo, cada partitura tiene un tamaño distinto del resto. 32

5.1 Esquema general de la metodología para recuperar las aristas del grafo de notación musical. 36
5.2 Distribución del número de nodos y de aristas positivas en el dataset MUSCIMA++. 38
5.3 Media de los resultados (5-CV) en términos de F_1 con respecto al modelo y las características consideradas. La línea punteada representa el resultado de Pacha et al. Pacha et al. (2019) (*upper bound*). 41
5.4 Recuperación de aristas por el método del *MLP*. 42

5.5 Recuperación de aristas por el método *Asymmetric Kernels*. 42

5.6 Visualización del rendimiento de los enfoques. Las líneas verdes representan TP, mientras que las líneas azules y rojas representan FP y FN, respectivamente. Las cajas turquesas representan los nodos. 42

5.7 Esquema general de la metodología. El proceso de extracción de anotaciones se representa en la figura. Cada anotación a_i está compuesta por las características del píxel i a lo largo de todos los N canales extraídos del mapa de características de la CNN. 46

5.8 Ejemplos de diferentes estructuras musicales seleccionadas para nuestros experimentos. 48

5.9 Distribución del tamaño de los multisets tras la selección de datos. 49

5.10 Resultados de cada modelo medidos en términos de la métrica *Intersection-Over-Union* (IoU). La gráfica muestra los resultados medios obtenidos en un 5-CV. Cada barra representa el rendimiento según el porcentaje de datos utilizados para el entrenamiento, mientras que el eje x indica la configuración específica. 51

5.11 Ejemplos de diferentes predicciones obtenidas con el mejor modelo. (a) Este ejemplo corresponde a una imagen con 23 símbolos: 10 cabezas de nota completas, 10 plicas y 3 haces. La ácrata obtenida es 100. (b) Este ejemplo contiene 16 símbolos: 6 cabezas de nota completas, 6 plicas y 4 *beams*. Al modelo le falta un *beam*. La puntuación obtenida es de 94%. 53

5.12 Esquema general de la metodología. Una CNN extrae la representación de la imagen, que es la entrada al *decoder* RNN. Esta RNN se entrena para producir cada símbolo en cada *timestep* t junto con la representación de los nodos. Finalmente, predecimos las aristas utilizando estas representaciones de nodos por pares. 55

5.13 Estructuras musicales extraídas de la MUSCIMA++. 59

5.14 Distribución del tamaño de los grafos en el *dataset* considerado. 60

5.15 Visualización de los resultados presentados en la tabla 5.7. Se muestra la media $\pm$ std. por cada resultado de 5-CV. 62

5.16 Diagrama de puntos que correlaciona los resultados en el conjunto de test obtenidos en cuanto al *Symbol Accuracy* y F_1 en cada *fold* del 5-CV para todos los modelos. 64

5.17 Ejemplo de una predicción correcta en la que los nodos y aristas se recuperan con éxito. 65

5.18 Ejemplo de predicción con errores. El recuadro rojo representa que el nodo ha sido etiquetado erróneamente, mientras que la línea roja discontinua indica que la arista entre los nodos correspondientes no ha sido predicha. 65

5.19 Ejemplos de predicciones del conjunto de test. Las imágenes de la izquierda representan la imagen de entrada dada al modelo, mientras que las de la derecha representan los grafos recuperados por nuestro mejor modelo. Obsérvese que los cuadros delimitadores (*bounding boxes*) representados son simplemente una ayuda visual, ya que nuestro enfoque no los recupera. El verde indica las predicciones correctas y el rojo los errores, que se describen en cada ejemplo. 65

List of Tables

3.1 Ejemplo de una representación de tres clases en formato *one hot*. 29

5.1 Media de los resultados (5-CV) en términos de F_1 con respecto al modelo y las características consideradas. 40
5.2 Rendimiento temporal de cada método propuesto según el número de nodos en una partitura. Los tiempos mostrados son el resultado de una media tras 10 ejecuciones. 43
5.3 Estadísticas según el número de primitivas en las estructuras musicales seleccionadas del *dataset* MUSCIMA++. Nótese que únicamente consideramos para este problema los símbolos que aparecen en la tabla. 48
5.4 Resultados promedios de IoU en 5-CV con respecto al enfoque (el uso o no de mecanismos de atención, aumentado de datos, o el tipo de ordenación de la secuencia) y el porcentaje de los datos de entrenamiento utilizados. Los resultados en negrita indican el mejor IoU por % de los datos de entrenamiento. 51
5.5 Estadísticas acerca de las primitivas de la notación musical (Σ) en las estructuras seleccionadas del *dataset* MUSCIMA++. 58
5.6 Estadísticas de los datos obtenidos una vez filtrada la MUSCIMA++. Avg. indica la media $\pm$ desviación típica. 58
5.7 Resultados medios en el conjunto de test para los diferentes escenarios de evaluación en un 5-CV. Se destacan los mejores resultados para cada porcentaje según la disponibilidad de datos. 62

1 Introducción

La cantidad de información que se genera hoy en día es de magnitudes inimaginables. Gracias al avance científico y tecnológico exponencial de las últimas décadas, tenemos la posibilidad de procesar, almacenar y consultar toda esta información generada y obtener valor a partir de ella. Como consecuencia de estos avances, somos capaces de aprovechar conocimiento previo y utilizarlo para mejorar diversos ámbitos sociales, como son la salud, la educación, la economía, la tecnología, el medioambiente y la cultura, entre muchos otros. La mejora global de estos ámbitos supone un aumento de la calidad de vida y el bienestar de las personas. El tratamiento adecuado de cualquier tipo de información en forma de datos es de vital importancia para extraer valor. La capacidad de contrastar información de diversas fuentes y extraer conocimiento es lo que nos proporciona esa posibilidad de entender el mundo que nos rodea y poder mejorarlo.

En esta extracción de conocimiento, juega un papel fundamental la capacidad de detectar relaciones y anomalías en los datos, así como identificar propiedades relevantes en forma de patrones. Debido a la magnitud en la que estos vienen, es inviable—en cuanto a recursos económicos y temporales atañe—que su procesado se realice de forma manual. El Aprendizaje Automático—o *Machine Learning*—es el conjunto de técnicas y algoritmos que permite a los ordenadores extraer automáticamente conocimiento estructurado a partir de datos (Good-Fellow et al., 2015). De esta forma, es posible extraer valor de los datos de manera eficiente y permite ahorrar una enorme cantidad de recursos. Uno de los ámbitos que se puede beneficiar gracias a estos sistemas es el cultural. En este trabajo, nos centraremos en el ámbito musical.

La música representa un bien cultural de especial interés, ya que simboliza un patrimonio histórico-artístico de incalculable valor para la sociedad. Crear sistemas que permitan preservar y explotar dicho patrimonio es de gran importancia. Este patrimonio se ha transmitido históricamente en forma de partituras, tratando de capturar de manera precisa la música a expresar mediante la notación musical. El campo que se encarga de la lectura e interpretación de la música desde un punto de vista computacional es el de *Optical Music Recognition* (OMR). En concreto, esta área de investigación se encarga de estudiar e interpretar la notación musical a partir de partituras musicales (Calvo-Zaragoza et al., 2020).

Esta área se ha centrado tradicionalmente en obtener la notación musical a partir de partituras. La literatura actual aplica diversas técnicas para obtener la notación de manera secuencial, donde se recuperan símbolos independientes sin relación entre sí. Sin embargo, no hay mucha investigación al respecto sobre cómo obtener las relaciones entre dichas primitivas musicales (símbolos), que es lo que permite interpretar esta notación de manera precisa. Algunos autores sugieren que, dada la naturaleza relacional de la música, una posible solución es representar dicha notación musical como un grafo, en el que los símbolos musicales corresponden a los nodos y las relaciones entre estos a las aristas. Actualmente, no se ha tratado cómo obtener estos grafos.

En este trabajo de fin de máster, abordamos esta brecha en la literatura, estudiando desde

diversas perspectivas cómo obtener un grafo a partir de partituras musicales. En el primer enfoque, se aborda la cuestión de extraer directamente las relaciones entre símbolos, partiendo de una detección previa de los símbolos, ámbito ampliamente estudiado en la literatura de OMR. En este trabajo se proponen dos métodos que permiten realizar dicha extracción de relaciones, uno de que trabaja de manera más eficaz y otro de manera más eficiente. En el segundo enfoque, se estudia la viabilidad de extraer los símbolos (nodos en el grafo) a partir de extracciones de partituras en forma de imágenes con sistemas que no utilicen detección de objetos, ya que el coste de anotar *datasets* de música es altamente costoso. En este trabajo se aborda esta tarea utilizando sistemas de *image captioning* como en Vinyals et al. (2015), en la que a un sistema recibe una imagen como entrada y este describe la imagen por medio del lenguaje natural. Por último, en el tercer enfoque, se aborda la tarea de extraer grafos de manera holística a partir de extractos de partituras musicales. En esta última propuesta, se propone una arquitectura que permite sacar tanto los símbolos como sus relaciones de manera completa, obteniendo grafos como salida.

Los resultados en los distintos escenarios experimentales realizados utilizando el conjunto de datos MUSCIMA++ Jan Hajič & Pecina (2017) sobre notación musical manuscrita, muestran que es posible extraer tanto las relaciones como los grafos de manera holística con alta precisión. Los resultados empíricos demuestran que existen dos cuestiones vitales para obtener un buen rendimiento en estas tareas: (1) la información que se tiene acerca de los símbolos cobra especial relevancia en el despeño de los modelos y (2) la representación concreta sobre los grafos repercute de manera directa en la precisión con la que se obtienen estos.

1.1 Objetivos

En esta sección se describen los principales objetivos de este trabajo de fin de máster.

Objetivos académicos/investigación

Por una parte, el objetivo principal es tratar de abordar este hueco en la literatura de OMR respectivo a predecir grafos que representen la notación musical. Para ello, se aborda este tema desde 3 perspectivas distintas. Los objetivos son los siguientes:

I Analizar la precisión con la que se pueden obtener las relaciones entre símbolos musicales, partiendo de la localización e identificación de estos. Proponer métodos para la extracción de estas relaciones.

II Estudiar la viabilidad de obtener los símbolos a partir de extractos de partituras con enfoques que no utilicen técnicas de detección de objetos.

III Proponer métodos que recuperen de manera holística grafos que representen la notación musical: tanto símbolos musicales (nodos) como las relaciones entre ellos (aristas) a partir de extractos de partituras musicales.

Para el primer objetivo, se tratará de estudiar la literatura actual específica sobre recuperación relaciones entre símbolos para OMR. Posteriormente, se propondrán métodos para recuperar dichas relaciones y se compararán con el estado del arte actual desde el punto de vista de la eficacia y la eficiencia.

En lo que respecta al segundo objetivo y, dado que supone la primera aproximación para recuperar una parte del grafo, (en este caso solo los nodos/símbolos), se propondrán métodos para recuperar estos símbolos a partir de extractos de partituras, estudiando la viabilidad de esta tarea con enfoques que no usen detección de símbolos. La elección estos métodos se deben a la complejidad de etiquetado de los *datasets* de música y la cantidad de recursos humanos y temporales a invertir en ello, por lo que no es deseable que los métodos dependan de conjuntos de datos que tengan estas etiquetas espaciales.

Para el último objetivo, dado que no existe literatura al respecto, se tratará de establecer un *baseline* en lo que supone la recuperación holística de grafos para OMR a partir de extractos de partituras. Se estudiará la literatura actual para predicción de grafos (en otros ámbitos fuera de OMR) y se propondrán soluciones aplicables para la tarea en cuestión. Además, se estudiarán las distintas formas de obtener representaciones canónicas y que sean viables para la predicción de grafos condicionada a una imagen.

2 Estado del arte

Este capítulo muestra los antecedentes fundamentales de los dos campos de investigación que convergen en este trabajo: OMR y la recuperación de grafos. Con el fin de valorar las aportaciones de este trabajo para cada campo, se presentan ambos antecedentes de forma individual.

2.1 Optical Music Recognition

El OMR se ha dividido tradicionalmente en varias etapas abordadas de forma independiente Rebelo et al. (2012). Fundamentalmente, hay un primer paso en el que se detectan las primitivas musicales, como las cabezas de nota, los barrados o los *flags*. Esto implica el procesamiento de la imagen de entrada con el fin de aislar y categorizar estos componentes, lo cual no es sencillo debido a la presencia de elementos como líneas de pentagrama y símbolos compuestos Dalitz et al. (2008). En la segunda etapa, se infieren las relaciones sintácticas entre las primitivas obtenidas para recuperar la estructura de la partitura. Estas etapas se resuelven combinando técnicas de procesamiento de imágenes con estrategias heurísticas basadas en reglas elaboradas manualmente Rossant & Bloch (2007). Desgraciadamente, estas soluciones han demostrado ser muy insuficientes Byrd & Simonsen (2015).

Algunos autores han propuesto soluciones para OMR siguiendo este enfoque holístico del estado del arte antes mencionado, utilizando una versión serializada (secuencial) de la notación musical Baró et al. (2019); Torras et al. (2021), de forma muy similar a como se ha intentado en otros dominios visuales Zhang et al. (2017). En el caso de las partituras, esta serialización representa una gran simplificación que sigue desperdiciando la mayor parte de la información relacional. En los casos más sencillos, como la música monofónica o ciertos sistemas notacionales precedentes -como la notación mensural o neumática- los métodos holísticos secuenciales han mostrado resultados prometedores Calvo-Zaragoza et al. (2019). Este enfoque, sin embargo, nunca se ha extendido para tratar cualquier tipo de partitura debido a las limitaciones actuales relacionadas con la expresividad de la estructura de salida Calvo-Zaragoza et al. (2020).

Una prometedora formulación holística para OMR requiere que el dominio de salida sea más expresivo que una secuencia, ya que de lo contrario podría desperdiciarse información relevante. Este problema no es específico de OMR, ya que este mismo reto aparece en otros dominios gráficos, como el reconocimiento de expresiones matemáticas Chan & Yeung (2000) o la generación de trazados de carreteras Chu et al. (2019), por citar algunos. En el caso de la notación musical, un grafo sí representa una estructura que se ajusta de forma natural al tipo de expresividad que se quiere plasmar Jan Hajič & Pecina (2017). Sin embargo, a día de hoy no se ha intentado recuperar holísticamente un grafo a partir de una imagen o, más concretamente, de una partitura musical.

2.2 Predicción de grafos

Como se ha dicho anteriormente, existe gran variedad investigación centrado en el procesamiento de grafos de entrada utilizando GNNs. Sin embargo, el objetivo aquí no es procesar un grafo, sino recuperar estas estructuras a partir de una imagen de entrada.

El trabajo más cercano a nuestro objetivo es el conocido como *Deep Graph Generation*. Este campo de investigación se centra en los modelos neuronales generativos para grafos. Actualmente, existen dos enfoques principales para la generación de grafos: (i) los métodos de generación de un solo paso Simonovsky & Komodakis (n.d.); Cao & Kipf (n.d.) y (ii) los de generación secuencial You et al. (n.d.); Li et al. (n.d.). El primero trata de generar grafos en un solo paso, mientras que el segunda utiliza redes neuronales secuenciales, como las que emplean unidades recurrentes, que generan iterativamente la secuencia de nodos y aristas con las que se construye el grafo. También se han realizado trabajos recientes sobre esta formulación específica, incluyendo una propuesta de marco general Yang et al. (2019) y algunas aproximaciones prácticas Jonas (2019); Li et al. (2018). En todos los casos, sin embargo, el problema se aborda aprendiendo un modelo generativo a partir de un conjunto de datos de grafos dado. Sin embargo, en este trabajo se pretende aprender un modelo discriminativo condicionado a una entrada de imágenes.

Otra área de investigación que merece la pena destacar es la de *Scene Graphs*. Este campo se centra en la extracción y la construcción de grafos que modelan las relaciones semánticas entre las entidades detectadas en las imágenes Chang et al. (n.d.). Sin embargo, todo el proceso de construcción se basa en la premisa de disponer de información espacial sobre los objetos para su detección. Esto supone que, para establecer dichas relaciones semánticas, se dispone de información precisa sobre la posición de los objetos en la escena.

Hasta donde sabemos, solo existe un trabajo que aborda la recuperación de grafos discriminativos condicionados por una imagen de entrada Belli & Kipf (n.d.). Sin embargo, este trabajo se centra únicamente en la estructura del grafo, ya que el método predice solo las aristas entre los nodos no etiquetados. Desgraciadamente, esto no es aplicable a OMR, para el que los nodos deben tener una categoría que represente una primitiva de anotación musical.

3 Marco teórico

3.1 Introducción

Antes de explicar las bases teóricas exclusivas de este trabajo, es importante conocer los conceptos teóricos en los que se fundamentan las técnicas de *Deep Learning* utilizadas. Para ello, en esta sección describimos los distintos campos en un enfoque *top-down*, profundizando en cada uno de ellos de una forma similar a la figura 3.1. En ella, se muestra la jerarquía de las disciplinas mencionadas con un ejemplo representativo en cada una de ellas. Por tanto, en esta sección se tratan los campos de Inteligencia Artificial (IA), *Machine Learning* (ML) y, posteriormente, un estudio en profundidad de las bases del DL junto con los conceptos específicos relativos a este trabajo.

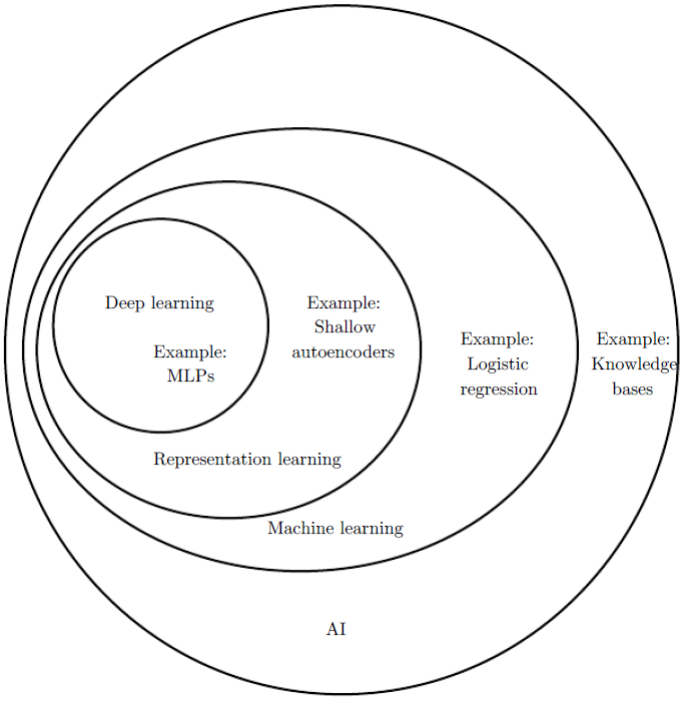


Figure 3.1: Diagrama de Venn de los campos y subcampos de la IA (GoodFellow et al., 2015).

3.2 Inteligencia Artificial. Definición y concepto

Centrándonos en el campo principal (la Inteligencia Artificial), no hay un consenso bien establecido acerca de este término o sobre qué se considera comportamiento inteligente o no,

a pesar del creciente uso del término en los últimos años. ¿Qué es entonces la Inteligencia Artificial?

Podríamos definir la Inteligencia Artificial (IA) como el campo de la investigación que trata de atribuir a las máquinas o sistemas la capacidad de comportarse de manera inteligente. Entendemos por comportamientos inteligentes: capacidad de razonar; aprender; memorizar; realizar inferencias; usar la información para realizar razonamientos lógicos y/o tomar decisiones sobre una tarea o cálculo que se tiene como objetivo. Aquí se engloba el conjunto de técnicas y mecanismos que permiten a una máquina imitar el proceso de aprendizaje.

En los primeros años de la IA, este campo estaba enfocado en resolver problemas que son intelectualmente complejos para el ser humano, pero sencillos para una máquina: cálculos matemáticos; problemas de formalización matemática; problemas que estén formados por el seguimiento de ciertas reglas (juegos como el ajedrez, las damas, etc.), que puedan estar compuestas por una serie de reglas o secuencias de tareas), etc. Una máquina puede resolver estos problemas fácilmente, ya que todo aquello que pueda formalizarse de manera matemática o por una serie de reglas también puede ser interpretado como un algoritmo[1]. Sin embargo, el principal problema al que se enfrenta este campo en auge, es el de resolver problemas que sean justo lo contrario: intelectualmente sencillos para al ser humano, y que estos se basen, *grosso modo*, en tareas "intuitivas", pero complicadas de formalizar de manera matemática o mediante una serie de reglas (p. ej., saber diferenciar si una imagen es un coche o un avión). Al no tener una manera unívoca de formalizar este tipo de problemas a los que llamamos "intuitivos", nuestro sistema inteligente ha de tener la capacidad de adquirir conocimiento, extraer información o patrones[2] de los datos que recibe, tal y como los humanos recibimos la información, ya sea por medio de una imagen, un texto, un sonido, etc. En definitiva, el problema al que nos enfrentamos es que nuestro sistema sea capaz de "aprender".

3.3 El proceso de aprendizaje

En el aprendizaje intervienen una gran cantidad de sucesos o procesos que aún son desconocidos debido al desconocimiento del cerebro humano. El proceso neurológico del aprendizaje no es objeto de este trabajo, aunque sí nos sirve para introducir este proceso desde el punto de vista computacional. Una analogía muy utilizada en el campo de la IA y del *Machine Learning* es el proceso de aprendizaje de un niño. Por ejemplo, si un niño trata de aprender a identificar un objeto como un coche, la intuición nos dice que este primero debería tratar de aprender a visualizar objetos individuales o conjuntos de objetos en forma de patrones que compartan unas características y que son comunes a todos los coches (ruedas, ventanas, etc.). De esta forma, la identificación de estos objetos que se repiten (patrones) es asociada con el objeto global que se compone de todos los objetos (coche), facilitando la tarea y asociando así los elementos individuales con el global. Para los humanos esto puede parecernos una tarea trivial propia de un infante y obviamos la complejidad de esta tarea. Sin embargo, el mismo proceso en una máquina no es una tarea trivial. Extraer patrones de los datos que observamos es algo que aprendemos de manera intuitiva, y que no sabemos exactamente (al menos hasta

[1] Entendemos como algoritmo el conjunto de operaciones finitas que se realiza de forma ordenada con el objetivo de resolver un problema o tarea dados.

[2] Un patrón es la aparición de un cierto tipo de suceso o elemento de manera recurrente, de tal forma que este se repite con cierta frecuencia.

la fecha) cómo es este proceso, por lo que formalizarlo resulta realmente complejo. Es en esto en lo que se centra el *Machine Learning*.

3.4 *Machine Learning*: extrayendo patrones

El proceso de aprendizaje nos adentra en la definición de *Machine Learning* (ML): capacidad que tiene una máquina para adquirir conocimiento a partir de la extracción de patrones en los datos (GoodFellow et al., 2015). El *Machine Learning* o aprendizaje automático, permitió a los ordenadores afrontar aquellos problemas de naturaleza más intuitiva o difícil de explicar, añadiendo conocimiento real, directamente de los datos que se analizaban.

Este enfoque (*Machine Learning*), nos permite extraer patrones de los datos, pero sólo para un conjunto limitado de problemas en los que el nivel de abstracción de las características que se requiere extraer de los datos no es muy elevado. Como vemos, a medida aumentamos el nivel de abstracción y de sofisticación en la tarea de extraer patrones sobre datos, nos adentramos en los subcampos de la IA. Podemos concluir, por tanto, que a medida que profundizamos en el diagrama de la figura 3.1, el nivel de abstracción y calidad en la extracción de patrones aumenta. Cada subcampo de la Inteligencia Artificial concreta más acerca de la tarea que se desempeña, aumentando el grado de complejidad o refinamiento en la tarea llevada a cabo. En el caso que nos atañe, la tarea es la extracción de patrones sobre datos.

3.4.1 Tipos de aprendizaje

Según la naturaleza del problema y a la disponibilidad de datos etiquetados[3], podemos clasificar el aprendizaje en diferentes categorías: aprendizaje supervisado, no supervisado o semi-supervisado y aprendizaje por refuerzo. En esta sección tratamos cada tipo de aprendizaje por separado.

3.4.1.1 Aprendizaje supervisado

En este tipo de aprendizaje, se utilizan algoritmos que procesan datos cuya etiqueta es conocida. De esta forma, se trata de que el algoritmo aprenda a asociar unos ejemplos de entrada con unas etiquetas concretas. El término ”supervisado” proviene desde el punto de vista que el objetivo es fijo y proporcionado (se sabe su salida real con la etiqueta) como si existiese un profesor o instructor que le dice al sistema qué tiene que hacer. Este trabajo se centra en este tipo de aprendizaje. Podemos agrupar los problemas de aprendizaje supervisado en 2 tipos:

- Problemas de clasificación. En este tipo de problemas, se predice una etiqueta discreta que hace referencia a un grupo concreto que comparten una serie de características. Ejemplo: predecir si una imagen es un coche, un barco o un avión es un problema de clasificación.

- Problemas de regresión. En problemas de regresión, se predice un valor o etiqueta continua. Ejemplo: predicción del precio de viviendas en una zona concreta de España.

[3]Etiquetado hace referencia a tener un dato o conjunto de datos cuyo grupo, valor o categoría a la que pertenecen es conocida de antemano.

3.4.1.2 Aprendizaje no supervisado

En el aprendizaje no supervisado, los algoritmos reciben datos sin etiquetar, sin saber la respuesta "correcta" a una entrada. Un caso típico es el de Adaptación al dominio o *Domain Adaptation* (DA), que pretende aprovechar lo aprendido a partir de un conjunto de datos o dominios etiquetado para procesar datos del dominio no etiquetado.

Hay muchos tipos de aprendizaje no supervisado, pero podemos agruparlos en:

- *Clustering*. Se trata de encontrar grupos o asociaciones entre los datos, realizando particiones entre conjuntos de datos que compartan una serie de características similares, etc. Ejemplo: algoritmos de k-means. En la práctica, agrupar usuarios según sus gustos musicales.
- Asociación. Se trata de averiguar reglas de asociación que describan justificaciones o relaciones entre datos o sucesos: Ejemplo: asociar personas que compran X producto, tienden a comprar Y.

3.4.1.3 Aprendizaje semi-supervisado

El aprendizaje semi-supervisado, por su parte, es un tipo de aprendizaje supervisado en el que los datos contienen pocos ejemplos etiquetados y una gran mayoría de ejemplos sin etiquetar (sin saber el resultado "correcto"). Este tipo de problemas es muy común dado que el coste computacional del etiquetado masivo de datos suele ser elevado. En este tipo de aprendizaje, queremos que nuestro algoritmo aprenda de los datos etiquetados para poder usarlos posteriormente en aquellos en los que no existe etiqueta.

Ejemplo: procesamiento del lenguaje natural (NLP) o clasificación de grandes *corpora* de textos.

3.4.1.4 Aprendizaje por refuerzo

En el aprendizaje por refuerzo, un agente[4] aprende a mejorar su conducta en un entorno o ambiente mediante la atribución de recompensas (positivas o negativas). En este caso, no hay un *dataset* fijo, sino más bien una meta o conjunto de ellas que se pretenden alcanzar. Al agente se le proporciona *feedback* por cada acción en el entorno en forma de recompensa acerca del rendimiento según que esté desempeñando en realizar las tareas objetivas.

Ejemplo: aprender a jugar a ciertos tipos de juegos tratando de alcanzar la máxima puntuación posible mediante acciones propias del juego.

3.5 *Deep Learning*

En este apartado profundizamos en el campo del *Deep Learning*. Para ello, dividiremos las 3 épocas principales en las que este se desarrolla: Cibernética, Conexionismo y la época del *Deep Learning* moderno hasta la fecha actual, tal y como lo conocemos hoy en día. En el último apartado, nos adentraremos en los aspectos teóricos específicos fundamentales para el desarrollo de este trabajo: las redes neuronales (a nivel general), las redes neuronales convolucionales y las redes neuronales recurrentes.

[4]Un agente es el programa que entrena con el objetivo de realizar una o varias tareas

3.5.1 Cibernética: El perceptrón simple y la Neurona Artificial

Esta primera etapa del *Deep Learning* abarca el período de los años 1940 y 1960, y dio paso a la aparición de los primeros modelos[5] como el perceptrón y lo que hoy en día conocemos como la Neurona Artificial. Es importante recalcar que no es hasta 1943 (inicio de la Cibernética) donde Warren McCulloch y Walter Pitts presentan en el ensayo *"A Logical Calculus of the Ideas Immanent in Nervous Activity"* el modelo matemático de una neurona biológica, emulando la activación de los impulsos eléctricos generados por la sinapsis entre neuronas. El trabajo de McCullouch y Pitts tratan de emular el esquema de la figura 3.2, donde se muestra un esquema básico de la conexión de dos neuronas biológicas mediante el mecanismo de sinapsis. Este mecanismo biológico que transmite los impulsos nerviosos entre dos neuronas o neurona-célula es el que se intentó simular en el ensayo.

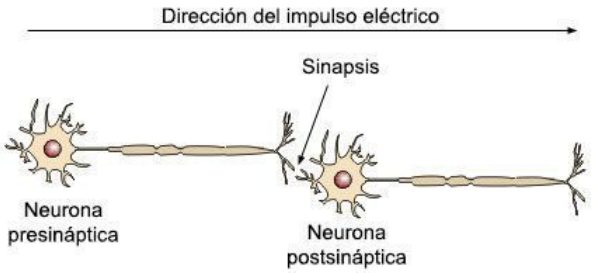


Figure 3.2: Esquema de dos neuronas biológicas. Fuente: psicoactiva.com

Sin embargo, este modelo era muy limitado y no tenía ningún mecanismo de aprendizaje o de retención de información, pero supuso la fundación de lo que hoy en día denominamos Redes Neuronales y *Deep Learning*.

En 1957, Frank Rosenbaltt, basándose en el trabajo previo de McCulloch y Pitts, publica *"The Perceptron: A Perceiving and Recognizing Automaton"* Rosenblatt (1957), donde el autor describía un nuevo concepto: el perceptrón. Este mejoró el propuesto anteriormente e introdujo algunos términos como los "pesos", que serán de vital relevancia para el *Deep Learning* actual. Aquí se establecieron las bases de la relación entre el modelo biológico y el matemático que se usará como base de otros modelos más complejos. Este ensayo dio lugar a lo que hoy en día conocemos como Neurona Artificial (NA) o o *Artificial Neuron* (AN).

Este modelo descrito por Rosenbaltt trata de simular el funcionamiento cerebral de una única neurona con muchas conexiones.

Como observamos en la figura 3.3, la neurona recibe una serie de señales[6] de entrada x_i, y a cada una de las conexiones se le asigna un peso w_i. Por tanto, cada conexión que le llega a la neurona está compuesta por una señal de entrada y un peso asignado a dicha señal. Cada conexión aporta un valor concreto, que viene dado por el valor de la señal de entrada multiplicado por el peso asignado, de la forma $x_i w_i$. Este peso asignado nos sirve para asignar importancia a una señal concreta. La señal total de la neurona es la suma de todas las señales

[5] Aunque en la literatura los términos algoritmo y modelo se suele usar indistintamente, cabe clarificar que la palabra modelo hace referencia a la función que trata de ajustarse a unos datos en concreto.

[6] Utilizaremos indistintamente la palabra señal o valor para referirnos a una información concreta, medida cuantitativamente, que se proporciona como entrada.

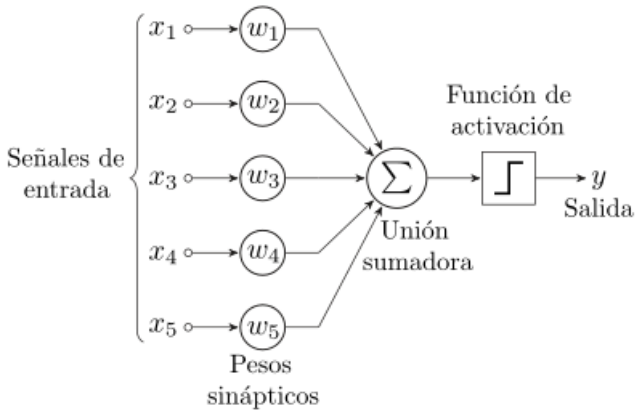


Figure 3.3: Figura del perceptrón simple. Fuente: wikipedia.org

de entrada multiplicadas sus pesos asignados, en la forma $\sum_{i=1}^{n} x_i * w_i$, siendo n el número de conexiones totales. Al igual que una neurona biológica, para poder transmitir la señal de una neurona a otra mediante un impulso sináptico, esta señal total también debe superar un umbral. Este umbral también es conocido como *bias*, *threshold* o sesgo y viene dado por la letra b, aunque también puede representarte con la letra T.

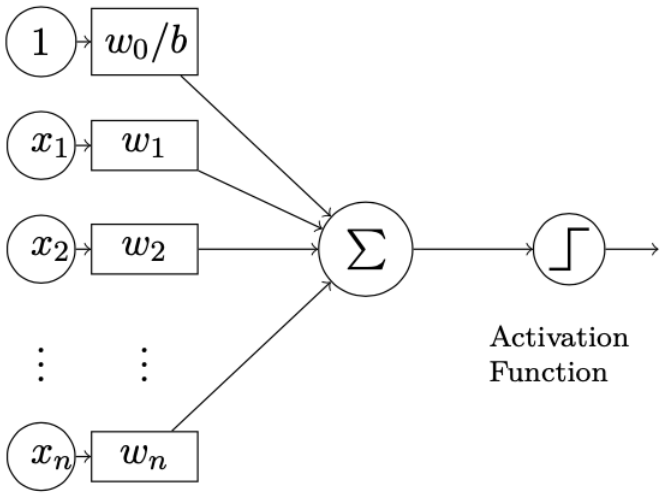


Figure 3.4: Figura del perceptrón simple con el sesgo añadido como peso w_0. Fuente: creación propia con el paquete TikZ

La figura 3.4 muestra la representación habitual del perceptrón simple con el *bias* incluido. Como vemos, se suele añadir un peso w_0 que representa la conexión del *bias*, y cuya señal de entrada siempre vale 1. Al valer siempre 1, este peso actúa como término independiente y permite desplazar recta, separando mejor los datos. Esto puede verse como la función lineal $y = mx + b$, donde la b representa el término independiente. De esta forma, se permite que la recta no tenga por qué pasar por el origen de coordenadas (0,0).

En el caso del perceptrón, al ser un modelo que tiene dos posibles salidas (0 o 1) la función que describe que la señal total supere el umbral es la siguiente:

$$f(x) = \begin{cases} 1 & \sum_{i=1}^{n} x_i w_i + b > T \\ 0 & otherwise \end{cases} \tag{3.1}$$

Esta función lineal proporcionará como salida (y en la figura 3.3) un 1 si la señal de la función de activación supera el umbral o *threshold* T dado y 0 en el caso de que no lo supere. La función que describe la salida de una neurona se conoce como función de activación. Como veremos más adelante, las funciones de activación son de vital importancia en cualquier modelo, ya que nos permite definir el comportamiento de una neurona dada una señal o valor de entrada.

Una notación muy extendida para denominar los pesos del perceptrón y los datos de entrada es forma matricial o vectorial

$$W = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ ... \\ w_n \end{bmatrix} \; X = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ ... \\ x_n \end{bmatrix} \tag{3.2}$$

Teniendo esta notación en cuenta, el sumatorio se reformula como $\mathcal{W}^T \cdot X$. Tal y como describíamos anteriormente, el sesgo o *bias* se asocia con el primer peso del perceptrón w_0 y con un valor adicional en los datos de entrada ubicado en la primera posición x_0 con valor constante de 1.

3.5.1.1 Limitaciones del perceptrón.

El modelo del perceptrón es un modelo muy simple. Además, el hecho tener el polinomio del sumatorio de grado 1, nos proporciona una función lineal. Este es uno de los problemas de este modelo, su poca capacidad[7]. Esto se debe a que únicamente se ajusta a un conjunto de funciones: las lineales. Tras haber hecho un análisis de este, sabemos que predice una salida binaria, que es interpretado como pertenencia o no a una clase, lo que limita bastante las posibilidades de uso.

El pseudocódigo 1 muestra el algoritmo del perceptrón simple. Primero, necesitamos el conjunto datos de entrada (X y una inicialización de los pesos con valores aleatorios cercanos a 0[8]. Como vemos, el perceptrón simple actualiza los pesos w y el *bias* b acorde a la predicción realizada para cada punto (x_i) en cada iteración hasta un máximo de iteraciones predefinido (ITER). La recta (o el plano/hiperplano) que genera el perceptrón cambia su ángulo (pendiente) con la actualización de pesos y su origen con la actualización del *bias* en el caso de los ejemplos mal clasificados (cuando el valor real y el de la predicción no coincidan, $y_i \neq \hat{y}_i$).

El algoritmo encontrará una división para ambas clases (si existe), moviendo la recta (o el plano) y ajustándola a los ejemplos mal clasificados.

El ejemplo por antonomasia de las limitaciones del modelo del perceptrón es la resolución del problema de las puertas lógicas que aparece en la figura 3.5. En esta, vemos cómo el

[7] La habilidad que tiene de ajustarse a una gran rango o familia de funciones.

[8] La explicación de esta tendencia es para no obtener cambios bruscos en la actualización de los pesos y tener más control sobre las variaciones.

Algorithm 1 Algoritmo del perceptrón simple.

Input: $X \in R^{nxd}$, $y \in \{-1, 1\}^n$, **w** inic. aleatoria, $b = 0$, $ITER \in N$
Output: **w**, b, error

for $t \leftarrow 1, ITER$ **do**
  $error(t) \leftarrow 0$
  for $i \leftarrow 1, n$ **do**
    $y_i \leftarrow f(x_i * w) + b$
    if $y_i \neq \hat{y}_i$ **then**
      $\mathbf{w} \leftarrow \mathbf{w} + y_i * x_i$
      $b \leftarrow b + y_i$
      $error(t) \leftarrow error(t) + 1$
    end if
  end for
end for

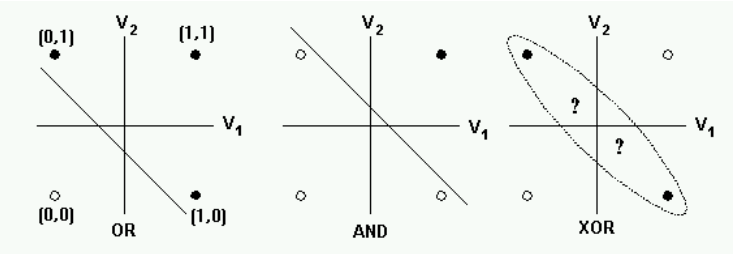


Figure 3.5: Problema de las puertas lógicas. Fuente: quora.com

perceptrón traza una línea para separar los puntos en el plano de dos dimensiones. El modelo no tiene la capacidad suficiente para separar los puntos distribuidos en la función XOR, tal y como muestran en 1969 Marvin Minsky y Seymour Papert en (Minsky & Papert, 1969), ya que con una línea es matemáticamente imposible separar esa distribución de puntos. En esta figura, las señales de entrada serían las dos coordenadas, por lo que el modelo del perceptrón recibiría 2 señales, una entrada para la primera coordenada, y otra entrada para la segunda coordenada[9]. Al tratarse de un problema de dos dimensiones, el modelo trata de dividir el espacio 2D en una línea recta que divida (discrimine) los datos en dos clases, las coordenadas que dan como resultado un 1 y un 0 acorde a las funciones de puertas lógicas que se están representando. Los puntos marcados en negro son aquellos puntos que corresponden a una salida de 1, y los puntos marcados en blanco corresponden a la salida de 0.

3.5.2 Conexionismo: Redes Neuronales y *Backpropagation*

Para poder hablar de Redes Neuronales y lo que consideramos *Deep Learning* moderno, pasamos primero por la época que se conoce como Conexionismo. Esta época está comprendida aproximadamente entre los años 1980 y 1995, donde se experimenta con el algoritmo de *Backpropagation* y se comienza a dar importancia al uso de las Redes Neuronales. Este

[9] A esto le sumamos el peso asociado al *bias* y la entrada constante de 1

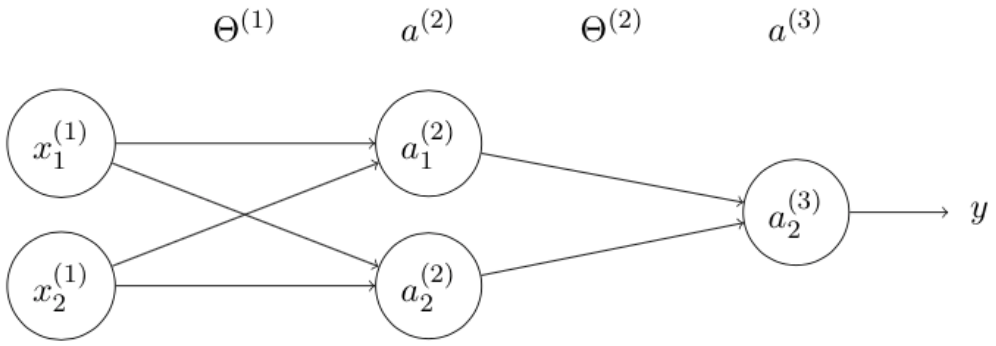


Figure 3.6: *Multilayer Perceptron* con dos neuronas por capa. Fuente: creación propia con TikZ

método de entrenamiento es en el que se basan principalmente las Redes Neuronales actuales.

A pesar de que los algoritmos de *Backpropagation* datan originalmente de la década de los 60, (Kelley, 1960) y los primeros esfuerzos en desarrollar algoritmos de *Deep Learning* con Redes Neuronales son también de la misma década (Ivakhnenko & Lapa, 1967), no es hasta 1986 con el ensayo (David E. Rumelhart, 1986) cuando se comienzan a utilizar ambos. Con esta publicación, se cubre la importancia del algoritmo de *Backpropagation* aplicado en Redes Neuronales.

3.5.2.1 *Multilayer Perceptron* (MLP)

El último modelo estudiado es en el que se fundamentan la mayoría de modelos de Redes Neuronales. La disposición resultante de unir dos o más modelos del perceptrón se denomina *Multilayer Perceptron* (MLP) o perceptrón multicapa.

El grafo[10] representado en la figura 3.6 nos da la respuesta. Los arcos del grafo representan conexiones neuronales, como veíamos en esquemas anteriores. Los nodos $a_1^{(2)}$ y $a_2^{(2)}$ representan perceptrones simples[11], teniendo como entradas x_1 y x_2 (sin contar los *biases* de cada perceptrón). Si avanzamos en la dirección natural del grafo, tenemos otro perceptrón Simple en $a_2^{(3)}$ y finalmente la salida representada como y.

Cada una de las agrupaciones verticales de nodos corresponde a una capa enumerada con el superíndice entre paréntesis. Cada capa está compuesta por las entradas que recibe (salidas de la capa anterior), las unidades de cómputo (perceptrones simples en este caso) apiladas en vertical y la salida de dichas unidades a la capa posterior, a excepción de la capa de entrada que tan solo contiene las entradas dadas y los pesos que sirven como entrada a la siguiente unidad de cómputo. Tomando como ejemplo el nodo de la capa 2 $a_1^{(2)}$, corresponde al primer perceptrón de la capa 2.

En nuestro esquema, la segunda capa recibe como entrada los datos de la primera capa (datos de entrada), mientras que el perceptrón de la 3ª capa recibe el resultado de los perceptrones de la capa anterior y realiza la inferencia, dado que su cómputo supone la salida o

[10] Conjunto de nodos o vértices (círculos en la figura 3.6) unidos por enlaces denominados aristas o arcos (si la flecha no tiene dirección o si la tiene, respectivamente)

[11] El peso que representa el *bias* así como su entrada con valor 1 se incluye de manera implícita y se suele omitir en este tipo de esquemas.

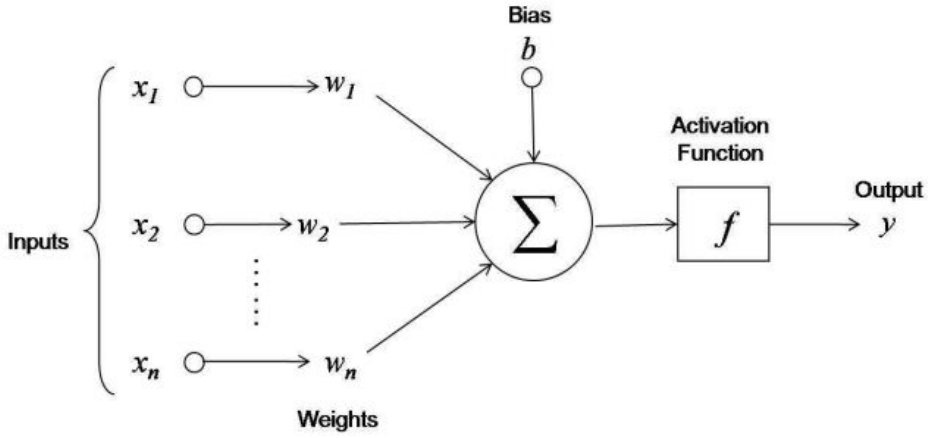


Figure 3.7: Neurona artificial con función de activación genérica. Fuente: datacamp.com

predicción. De esta forma, se van propagando los resultados entre capas hacia adelante.

Los símbolos de la figura 3.6 son ampliamente utilizados en esquemas de redes neuronales y perceptrones multicapa (abordaremos la diferencia en siguientes apartados). $\Theta^{(i)}$ representa la matriz de pesos que resulta de obtener el vector de pesos de todos los perceptrones. Siendo i la capa a la que se refiere, en nuestro ejemplo $\Theta^{(1)}$ (también se conoce como W) representa la matriz resultante de concatenar el vector de pesos del primer perceptrón y del segundo perceptrón de la segunda capa. La forma de la matriz resultante es $\Theta^{(1)} = \begin{bmatrix} w_{01} & w_{02} \\ w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}$, donde el primer subíndice representa el nodo origen de la capa anterior y el segundo el nodo al que se dirige el arco. Dicha unión supone una conexión neuronal entre los valores w_{01} y w_{02} representan la conexión de los *biases* o sesgos añadidos al esquema como una entrada más con el valor 1. El símbolo $a_j^{(i)}$ representa unidades de cómputo, perceptrones simples en nuestro caso. El superíndice i, al igual que en la matriz de pesos, representa la capa en cuestión y j el perceptrón al que se hace referencia.

3.5.2.2 Perceptrones simples vs. neuronas: rompiendo la linealidad

En el último apartado introducíamos un término nuevo: la unidad de cómputo. Hasta ahora, la unidad que hemos utilizado para computar valores de entrada mediante un umbral y obtener una salida discreta es el modelo del perceptrón (neurona simple).

En la figura 3.3 describíamos el esquema del perceptrón simple. La función de activación utilizada para este modelo era la función *threshold*. Como vimos, esta función proporcionaba valores discretos {0, 1} o {-1, 1}, según nuestra elección, si la señal resultante de computar el sumatorio de los pesos por las entradas era mayor que un cierto umbral. La figura 3.7 generaliza la función de activación dando lugar a lo que conocemos como Neurona Artificial.

La diferencia principal entre un perceptrón y una Neurona Artificial es la función de activación. Mientras que en el modelo del perceptrón la función de activación únicamente es de tipo umbral, en una neurona artificial la función de activación puede ser cualquier tipo, lo

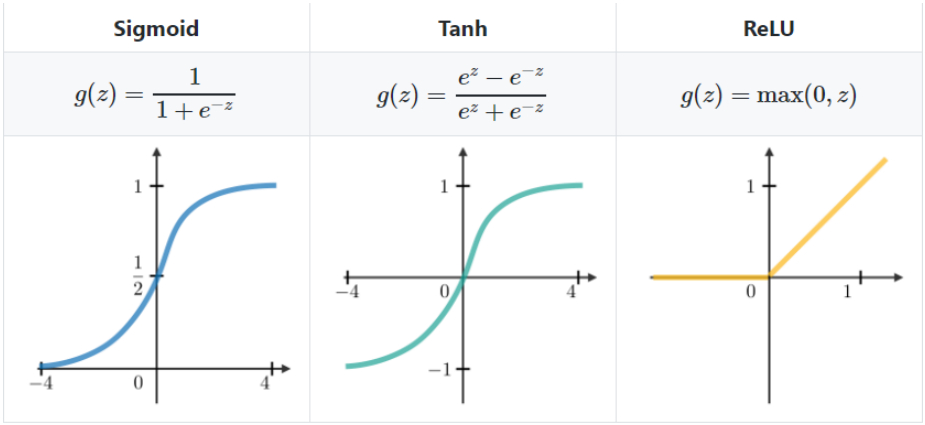


Figure 3.8: Funciones de activación comunes. Fuente: kaggle.com

cual trae consigo el uso de funciones no lineales.

Las funciones de activación no lineales proporcionan una ventaja a la hora de separar puntos en el espacio, ya que presentan una flexibilidad notable respecto a las funciones lineales. Por ello, las Neuronas Artificiales presentan una ventaja principal respecto a los perceptrones: pueden separar clases de puntos en el espacio mediante funciones no-lineales mediante las funciones de activación. En definitiva, añaden más flexibilidad a la hora de generar un modelo que represente los datos correctamente.

Las funciones más utilizadas en las neuronas artificiales se muestran en la figura 3.8, donde vemos las funciones sigmoidea, tangente hiperbólica y *Recitified Linear Unit* (ReLU), aunque existen muchas más. La elección de utilizar una función u otra no es una tarea arbitraria, ya que dicha elección es relevante para el correcto funcionamiento del modelo.

Las neuronas artificiales, agrupadas en capas como veíamos en el MLP, dan lugar a lo que conocemos como Redes Neuronales. Gracias a estas, hoy en día tenemos arquitecturas que van desde estructuras pequeñas y sencillas hasta las más complejas, con mayor número de conexiones, de capas o distintas funciones de activación en distintas capas, pero todas se basan en estos conceptos. Profundizaremos más en las Redes Neuronales en el siguiente apartado.

3.5.2.3 Redes Neuronales

Formalmente, una red neuronal (*Artificial Neural Network*, ANN) es un grafo acíclico que está compuesto por capas de neuronas (unidades de cómputo) interconectadas entre sí mediante conexiones denominadas pesos. El modelo del *Multilayer Perceptron* (MLP) es un tipo de red neuronal, en concreto una de tipo de propagación hacia adelante (*feedforward artificial neural network*). Recibe este nombre dado que la propagación de los valores desde las capas se produce siempre hacia adelante, produciendo al final una predicción (y). Como vimos, la característica principal de las Redes Neuronales es la flexibilidad que nos otorga romper la linealidad mediante las funciones de activación.

Para continuar con la notación y los conceptos sobre redes neuronales, es importante destacar que a las capas de las redes que no contienen ni las entradas a la red ni son las unidades que proporcionan las salidas, se les llama capas ocultas o *hidden layers*. La intro-

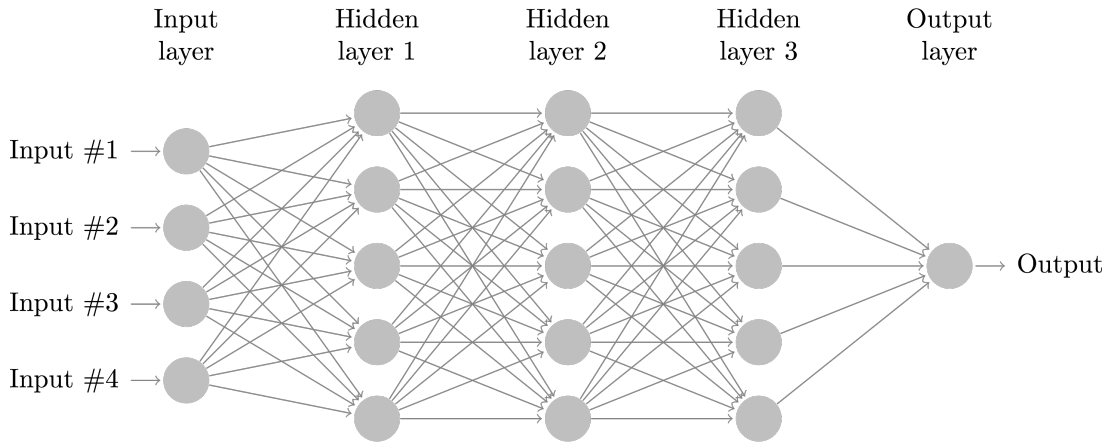


Figure 3.9: Esquema general de una red neuronal. Fuente: cs.swarthmore.edu

ducción de estas múltiples capas ocultas hace que el conjunto de métodos que utilizan las redes neuronales profundas (con varias capas) reciban el nombre de aprendizaje profundo o *Deep Learning*. La figura 3.9 muestra un esquema de una red neuronal con 3 capas ocultas.

Ahora ya conocemos cómo es una red neuronal y en qué se fundamentan la mayoría de las arquitecturas de Redes Neuronales. Sabemos qué son y cómo funcionan sus unidades de cómputo (neuronas) y cómo se propagan los resultados entre capas. La complejidad respecto al primer modelo del perceptrón es notablemente superior, pero también este tiene una capacidad mucho mayor, pudiendo ajustarse a una familia de funciones mucho más grande. Ahora bien, ¿cómo se elige la configuración de pesos adecuada para realizar tareas como la clasificación en las Redes Neuronales? En el siguiente apartado tratamos el algoritmo de *Backpropagation* basado en el método de Descenso por Gradiente, que permite que las Redes Neuronales aprendan la configuración de pesos adecuada para ajustarse a la tarea encomendada (clasificación en este caso) en el proceso que se conoce como entrenamiento.

3.5.2.4 Descenso por Gradiente y *Backpropagation*. Funciones de coste

En primer lugar, debemos saber cuánto difiere el valor predicho por la red del valor real de la etiqueta de la muestra. Para ello, necesitamos una medida cuantitativa de esta diferencia. En el Aprendizaje Automático se usa una función que se conoce como función de coste (J) (también llamada función de pérdida), que mide el error (o la pérdida *loss*) que se produce entre la salida de la red y la salida real. Hay muchos tipos de funciones de coste, y el uso de estas depende de la tarea que se está realizando. En este trabajo, nos centramos en su aplicación al campo del aprendizaje supervisado. Para este tipo de aprendizaje en el que las dos principales tareas son de regresión o clasificación, podemos listar de manera resumida los tipos de funciones de coste:

1. Funciones de coste para tareas de Regresión
2. Funciones de coste para tareas de Clasificación Binaria
3. Funciones de coste para tareas de Clasificación Multiclase

En este trabajo, nos centramos en los dos últimos tipos de funciones.

Tareas de clasificación binaria. Las funciones de coste para clasificación binaria son aquellas funciones que miden el error en tareas de clasificación cuya salida real son solo dos posibles valores, es decir, cuando la clasificación real de los datos de entrada está dividida en solo dos clases. En este caso, las salidas proporcionadas por los modelos suelen ser una probabilidad entre 0 y 1. De esta forma, necesitamos una medida del error de la probabilidad obtenida y la salida real. La principal función de coste es la función de entropía cruzada binaria (*Binary Cross Entropy Loss (BCE)*).

$$BCE = -\sum_{i=1}^{C'=2} t_i log(f(s_i)) = -t_1 log(f(s_1)) - (1-t_1) log(1-f(s_1))$$

Tareas de clasificación multiclase. Por último, las funciones de coste para tareas de clasificación multiclase son una generalización de las tareas de clasificación binaria, donde las etiquetas o valores reales de los datos de entrada pueden ser N-clases en vez de solo dos. Por tanto, la salida de la red será la probabilidad que hay de que la muestra introducida pertenezca a cada una de las clases.. En este caso, la función es una generalización de la BCE, dando lugar a la *Categorical Cross Entropy Loss (CCE)*.

$$CCE = -\sum_{i=1}^{C} t_i log(f(s_i))$$

Destacamos que, en ambas fórmulas BCE y CCE, $f(s_i)$ hace referencia al resultado de la función de activación una vez le llega la señal s_i, siendo intercambiable por $\hat{y}_i$. La notación t_i equivale al valor real o *ground truth*, y_i.

Tras conocer el objetivo de las funciones de coste y los tipos básicos que existen para el aprendizaje supervisado, ahora nos centramos en el Descenso por Gradiente y en el *Back-propagation*.

Descenso por gradiente. El algoritmo del Descenso por Gradiente es un algoritmo de optimización iterativo de primer orden que trata de buscar un mínimo local en una función diferenciable. La idea general de este algoritmo es dar pasos en la dirección opuesta al gradiente de la función, avanzando en la dirección que garantice una bajada más lejana. En el caso de las redes neuronales, hemos definido una función de coste para medir el error que se comete entre la predicción esperada y la que produce el algoritmo. El Descenso por Gradiente en las redes neuronales trata de minimizar la función de coste J, tratando de alcanzar un mínimo, que supone que el error es mínimo entre la salida real y la obtenida. De esta forma, tenemos que el objetivo es encontrar min $J(\theta)$. El algoritmo trata de obtener aquellos parámetros (pesos, θ) que hagan que esta diferencia sea mínima, de la forma:

$$\hat{\theta} = \theta - k\nabla J(\theta)$$

donde k es el *learning rate* o el tamaño de paso, que sirve para ajustar cuánto se debe descender en cada iteración del algoritmo.

Algoritmo *Backpropagation*. Como se definió en apartados previos, los pesos son los parámetros de las redes neuronales que modelan la salida deseada dadas las entradas, una por cada dimensión. En el caso de las redes neuronales de propagación hacia adelante, esto supone que el cambio en los valores de los pesos de cada capa de la red modificará la salida esperada. El algoritmo de *Backpropagation* (Dreyfus, 1962) es un algoritmo iterativo cuánto se debe descender en cada iteración del algoritmo. Este algoritmo se apoya en el concepto teórico del Descenso por Gradiente para cambiar los pesos de la red neuronal que permitan minimizar la función de coste J. El algoritmo realiza un proceso de dos pasos:

1. Propaga hacia adelante las señales desde el inicio hasta el final, obteniendo la salida de la red y el error cometido.

2. Propaga el error hacia atrás (*backpropagation*), haciendo que cada neurona reciba una parte de la señal total de error, mediante las derivadas parciales de la función de coste con respecto a los pesos implicados en cada capa.

Con este algoritmo y el desarrollo de la regla de la cadena, los parámetros de la red se actualizan de forma iterativa, como se describió en el Descenso por Gradiente. Estas variaciones permiten ajustar los pesos para minimizar el error, hasta que se obtiene punto de convergencia o se aplica una política de parada (generalmente cuando el error cometido es menor que un umbral o tras número de iteraciones[12] fijado) escogida.

Otros optimizadores para la búsqueda de mínimos. El Descenso por Gradiente estándar es uno de los algoritmos de optimización más conocidos y su variante SGD (*Stochastic Gradient Descent*) es uno de los más utilizados, junto con los basados en el momento o *momentum*. El SGD trata de actualizar los parámetros con más frecuencia. La principal diferencia con el de gradiente estándar es que a la hora de realizar la actualización los pesos no siguen un orden concreto en la selección de datos para calcular la pérdida, sino que más bien los escoge de manera aleatoria, aumentando la probabilidad de que la actualización no lleve a mínimos locales, lo cual no es deseable. Sin embargo, hay muchos más optimizadores y la selección de unos u otros tiene un largo bagaje matemático que no es objeto de este trabajo. Sin embargo, es destacable que existen cientos de ellos, aunque los más utilizados son SGD, *RMSProp*, *Adam* (*Adaptive Moment Estimation*), *Adagrad* (cambia el learning rate para cada parámetro y en cada paso), *Adadelta* (variación de Adagrad), entre muchos otros. Los basados en *momentum* tienen en cuenta las variaciones parciales en las derivadas de cada dimensión, añadiendo fracciones de as actualizaciones de cada vector a la actualización en curso. En resumen, los basados en *momentum* aceleran la convergencia debido a que el término de *momentum* aumenta para las dimensiones cuyos gradientes van en la misma dirección (se acelera la bajada) y se va reduciendo para aquellas dimensiones en las cuales el gradiente varía. Se gana rapidez y se tiene a una convergencia más rápida.

3.5.3 *Deep Learning* moderno. Aspectos específicos

Tras esta introducción a las redes neuronales profundas, nos adentramos el *Deep Learning* moderno. Esta época comienza a finales de los años 80 y sigue vigente hoy en día. En este

[12] A este número de iteraciones también se le llama épocas (*epochs*).

apartado tratamos en profundidad los temas específicos de este trabajo: las redes neuronales convolucionales (que nos sirven para los siguientes análisis teóricos) y las redes neuronales recurrentes.

3.5.3.1 Redes Neuronales Convolucionales

Las redes neuronales convolucionales (CNN), también conocidas como *ConvNets* se introdujeron en los años 80 por Yann LeCun en LeCun et al. (1989). Este investigador construyó su trabajo sobre el previo realizado por Kunihiko Fukushima, un científico japonés que diseñó el neocognitrón (Fukushima, 1980) con el objetivo de hacer reconocimiento de imágenes mediante redes neuronales. Es entonces cuando LeCun en su *paper "Backpropagation applied to handwritten zip code recognition"* publica un reconocimiento de dígitos en los códigos postales mediante redes neuronales convolucionales con *Backpropagation*. En este artículo se presenta la estructura básica de las CNN.

Las CNN son una arquitectura de redes neuronales que aplican distintas operaciones sobre los píxeles de una imagen a lo largo de distintas capas de la red. La idea general de este tipo de redes es obtener mapas de características representativas acerca de las imágenes de entrada que sean invariantes a escala, rotaciones, etc. Las CNN se utilizan, principalmente, en tareas de clasificación, detección de entidades y generación de otras imágenes a partir de una imagen en forma de entrada, entre otras. Las operaciones principales que aplican las CNN son las convoluciones y los *poolings*.

Convolución. Una convolución es una operación que se aplica sobre los píxeles de una imagen, dando como resultado otra imagen con los valores de los píxeles modificados. La operación consiste en aplicar una operación elegida (suma, resta, multiplicación[13]) sobre distintas áreas de la imagen, modificando los píxeles iniciales. Estas operaciones se realizan pasando *kernels* sobre las distintas zonas de la imagen. Un *kernel* es una matriz generalmente bidimensional (más pequeña que la imagen inicial) con unos valores (pesos) que actúan como operandos en las operaciones elegidas. Se muestra un ejemplo de una convolución en la figura 3.10. Como vemos, el *kernel* K, se desliza por la matriz inicial I, dando lugar a la matriz resultante $I * K$. Por otra parte, tenemos los filtros. Los filtros son concatenaciones de operaciones de convolución aplicadas con un número de *kernels* elegido, tal y como se ve en la figura 3.11. Como vemos, un filtro se compone de los tres kernels que se mueven por cada zona de la imagen, uno por cada canal.

Las convoluciones funcionan a modo de extractor de características y sirven para detectar líneas y distintos puntos en la imagen que pueden ser relevantes para la predicción. En la figura 3.12 se muestra un ejemplo de cómo estas operaciones de convolución pueden destacar zonas importantes de una imagen como las líneas verticales, esquinas, etc.

Poolings. La segunda operación relevante de las CNN son las operaciones de *pooling*. Uno de los problemas que sucede en el procesamiento de señales es que el mapa generado por las operaciones convolucionales es dependiente a la posición de las características que destacan. Esto supone un problema, ya que pequeños cambios en la posición de una característica generan mapas muy distintos. Las operaciones de *pooling* se centran en resumir una imagen para

[13]Por lo general, la operación elegida suele ser la de multiplicación.

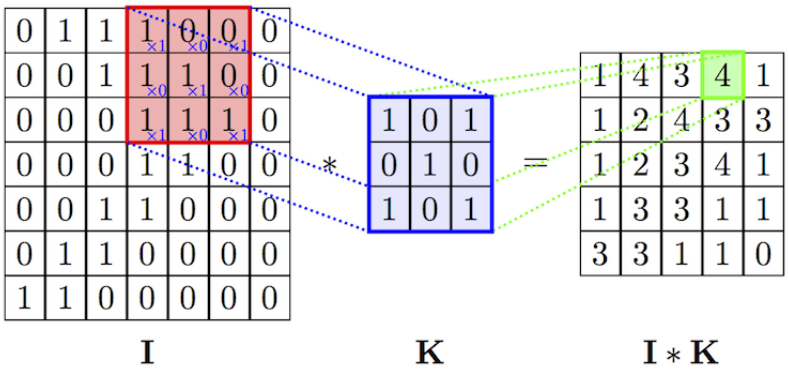


Figure 3.10: Ejemplo de operación de convolución. Fuente: researchgate.net

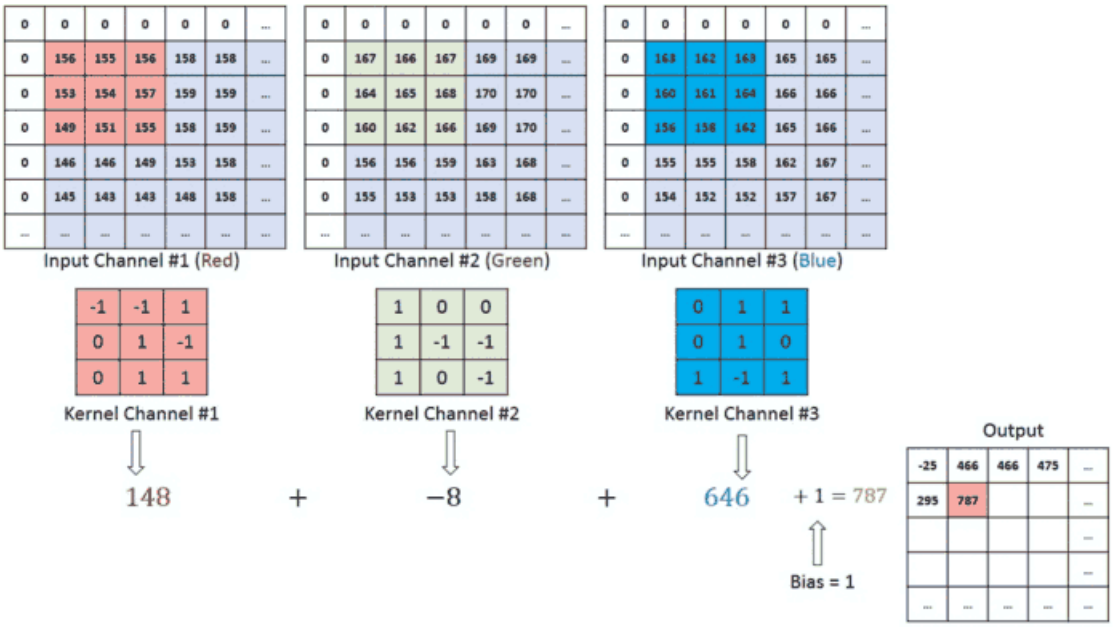


Figure 3.11: Ejemplo de operaciones de convolución con varios *kernels*. Fuente: towardsdatascience.com

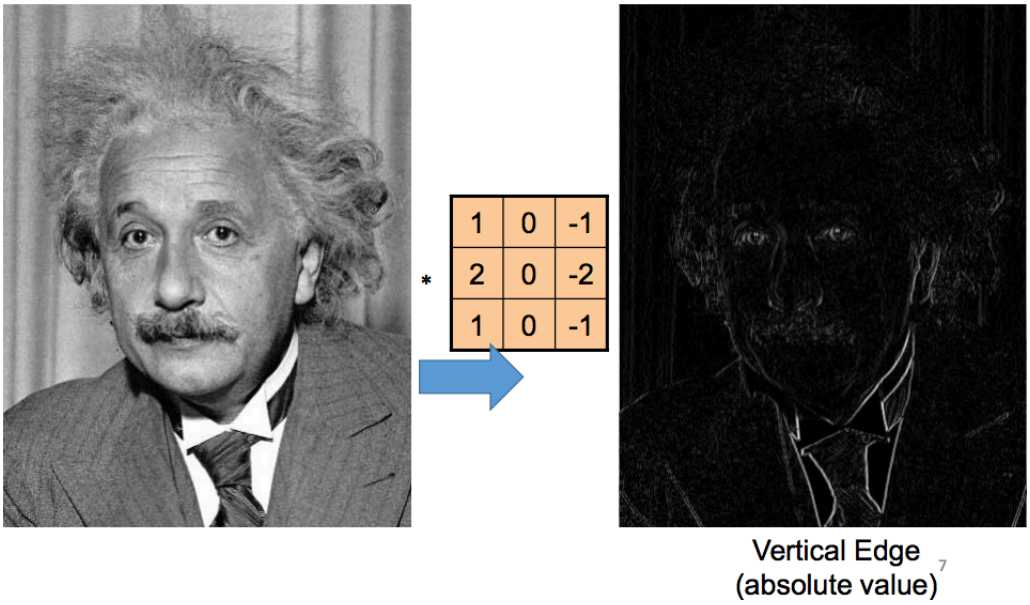


Figure 3.12: Ejemplo de extracción de características con filtro de Sobel sobre una imagen de Albert Einstein. Fuente: cs.toronto.edu

hacer estos mapas de características invariantes, en un proceso conocido como *downsampling* o muestreo. Este proceso consiste en crear una versión con menor resolución que el mapa de características generado por las operaciones de convolución, manteniendo sus partes más relevantes. Además, se resta complejidad al quitar los detalles menos relevantes.

Dentro de las operaciones de *pooling* tenemos varios tipos de operaciones, pero las más utilizadas son el *Average pooling* y el *Max Pooling*. En el caso de la media, lo que se calcula el valor medio por cada trozo del mapa de características, mientras que en la selección del máximo se selecciona el mayor valor del píxel de la zona que ocupa el kernel. La figura 3.13 muestra un ejemplo de los dos tipos de operaciones mencionadas.

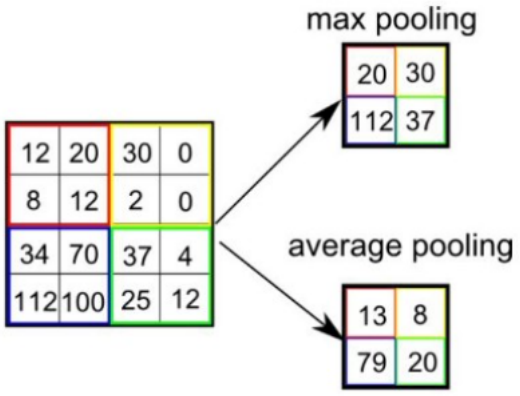


Figure 3.13: Ejemplo de *Max pooling* y *Average Pooling*. Fuente: towardsdatascience.com

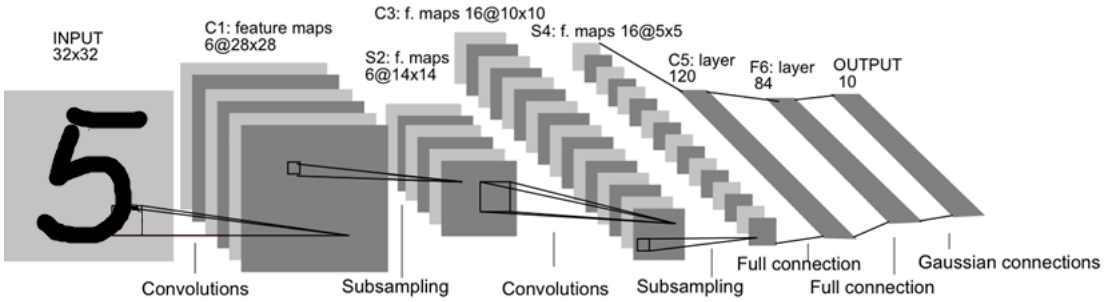


Figure 3.14: Arquitectura LeNet-5 por Yann LeCun, 1989. Fuente: researchgate.net

Figure 3.15: Ejemplo de dígitos manuscritos de la base de datos del *MNIST*.

LeNet-5. Un ejemplo de una arquitectura clásica de CNN es la arquitectura que se muestra en la figura 3.14. Como vemos, la arquitectura LeNet-5 está compuesta por capas de convolución alternadas con capas de *pooling*. En la parte final de la arquitectura se realiza una operación de *Flatten* o aplanado. Esta operación supone juntar todas las características proporcionadas por las capas anteriores. Las últimas tres capas se denominan *Fully-Connected Layers*, y son capas de la red con neuronas que están conectadas todas con todas entre capas. La salida final de esta red es un valor entre 0 y 9, dado que su objetivo es clasificar dígitos manuscritos entre esos valores. En este caso, la función utilizada es la función *softmax*, que es una generalización de la función *sigmoide* para poder dar una distribución probabilística entre todaslas clases. Dado que el número de clases para este problema era de 10, los valores oscilan entre 0 y 9, cada valor hace referencia a la clase de los dígitos. Un subconjunto del *dataset* de la base de datos *MNIST* se muestra en la figura 3.15.

Entrenamiento de una CNN. El entrenamiento de una CNN no difiere mucho de una red neuronal de las redes descritas en apartados anteriores. La principal diferencia en la disposición de las conexiones que existen entre las distintas capas. En una arquitectura de red sin convoluciones, la principal tarea del entrenamiento con *backpropagation* es la de aprender la mejor configuración de pesos entre las conexiones neuronales que minimicen el

error en la salida de la red en el proceso anteriormente descrito. En el caso de las capas de las CNN, hemos visto que los pesos se sitúan en los *kernels* que aplican operaciones de convolución sobre los píxeles. En este caso, la tarea del *backpropagation* es aprender los pesos que hacen que las convoluciones describan lo mejor posible la imagen generando estos mapas de características. En definitiva, tratan de crear el mejor mapa de características posibles para obtener finalmente el mejor descriptor de la imagen final. Los errores cometidos en las capas se retropropagan de final a inicio y, calculando los gradientes, se modifican los pesos de los kernels (filtros en su totalidad) que minimizan el error cometido por red.

3.5.3.2 Redes neuronales recurrentes

Las redes neuronales recurrentes o *Recurrent Neural Networks* (RNN) Elman (1990) son una familia de redes específicas para el procesamiento de secuencias. Una secuencia está definida por una serie de *tokens* de la forma $x^{(1)}, \ldots, x^{(N)}$, donde N suele ser el tamaño de la secuencia a procesar. De esta forma, en cada instante temporal o *timestep* t, la red neuronal procesa el token $x^{(t)}$. Para procesar este tipo de estructuras de datos, estas redes comparten pesos en distintas partes de la red. Gracias a esta característica, este tipo de redes lidian con tamaños de secuencia arbitrarios, así como compartir conocimiento estadístico en un momento determinado acerca del estado actual de la secuencia. Con una red neuronal común como el MLP, cada secuencia tendría que ser exactamente del mismo tamaño y, además, cada *token* sería procesado con unos pesos distintos, por lo que la red debería aprender todas las posibles reglas y combinaciones entre *tokens* para producir una determinada salida.

Según el tipo de tarea que se quiere realizar, ubicamos a las redes recurrentes en uno de estos tipos:

- *One-to-many*: a partir de una entrada $\mathbf{x}$, se produce una secuencia $y^{(1)}, \ldots, y^{(N)}$. Un ejemplo típico son los sistemas de *image captioning* como en Vinyals et al. (2015), en la que dada una imagen $\mathbf{x}$ se produce una secuencia que describa la imagen.
- *Many-to-one*: dada una secuencia $x^{(1)}, \ldots, x^{(N)}$, se predice la categoría a la que pertenece. Ejemplo: tareas de *sentiment analysis*, en la que una secuencia de entrada se categoriza según sea un sentimiento positivo, negativo o neutro.
- *Many-to-many*: en esta tarea, la entrada es una secuencia $x^{(1)}, \ldots, x^{(N)}$ y se trata de predecir otra secuencia $y^{(1)}, \ldots, y^{(N)}$. Un claro ejemplo es la traducción automática, en la que convierte una secuencia de entrada a otro idioma.

En general, la mayoría de las RNN se determinan por la ecuación 3.3

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta), \tag{3.3}$$

donde $h^{(t)}$ es el estado interno actual de la red en el *timestep* t tras procesar el token $x^{(t)}$ de la secuencia. Por su parte, $h^{(t-1)}$ es el estado de la red en $t-1$ y finalmente θ son los parámetros de la red. Este estado h la parte más relevante de la red, dado que en las RNN actúa a modo de resumen de lo que se ha visto y procesado hasta el momento. Como se aprecia, la ecuación 3.3 es recurrente, ya que para predecir el siguiente estado se utiliza el previo. En concreto, para realizar predicciones en una RNN genérica, las ecuaciones vienen dadas por

$$h_t = g(Uh_{t-1} + Wx_t), \tag{3.4}$$

$$\hat{y}_t = f(Vh_t), \tag{3.5}$$

donde U, W, V son matrices de pesos para proyectar linealmente las entradas, g puede ser cualquier función de activación y f suele ser la función softmax. Dadas estas ecuaciones, predice el siguiente token $\hat{y}_t$ más probable dada la entrada actual y el estado anterior de la red. El enfoque en el que se predice la siguiente palabra o *token* dada la palabra actual y estado previo se conoce como modelo de lenguaje (*language model*).

Este tipo de RNN simples han tenido históricamente varios problemas. En primer lugar, dadas las ecuaciones que expresan esos modelos, observamos que los pesos de la red, junto con el estado interno, tienen que atajar dos tareas cruciales: (1) retener y propagar información relevante para tomar decisiones futuras (como palabras clave que han aparecido anteriormente en la secuencia) y (2) mantener información relevante para la predicción actual.

Por otra parte, cuando el tamaño de secuencia es relativamente largo, esto resulta en tener muchas neuronas apiladas horizonalmente que se transfieren información de los gradientes gracias al mecanismo de *backpropagation*. Como resultado, las capas ocultas realizan muchísimas operaciones de multiplicación, lo que resulta que los gradientes de las primeras capas acaban tendiendo a 0 y aporten prácticamente información nula. Esto se conoce como el *vanishing gradient problem*, el cual es un problema común dentro del paradigma del DL.

3.5.3.3 *Long Short-Term Memory* (LSTM)

Con el objetivo de atajar los problemas mencionados de las RNN simples, aparece una extensión de estas redes conocida como *Long Short-Term Memory* (LSTM) *networks* Hochreiter & Schmidhuber (1997).

Para tratar el problema del contexto y retener información previa, estas redes se diseñaron con el objetivo de tener mecanismos para decir qué información es relevante para tomar decisiones futuras, así como de mantener información coherente y representativa de la secuencia hasta el momento t. Esta información mencionada se suele denominar contexto en la literatura.

Las LSTM añaden una capa especial para mantener este contexto (junto con las neuronas recurrentes habituales). Además, sumado a unas neuronas especiales que hacen uso de puertas para controlar el flujo de información dentro de estas neuronas, permite retener información relevante, así como tomar decisiones acertadas en cualquier instante temporal. En estas redes, existen tres tipos de puertas que componen estos mecanismos: la *forget gate*, la *add gate* y la *output gate*, que vienen dadas por las ecuaciones siguientes[14]:

$$f_t = \sigma(U_f h_{t-1} + W_f x_t) \tag{3.6}$$

$$k_t = c_{t-1} \circ f_t \tag{3.7}$$

[14]Se ha seguido la notación de Jurafsky & Martin (2009). El símbolo $\cdot$ representa el producto elemento a elemento entre dos vectores

$$g_t = \tanh(U_g h_{t-1} + W_g x_t) \tag{3.8}$$

$$i_t = \sigma(U_i h_{t-1} + W_i x_t) \tag{3.9}$$

$$j_t = g_t \circ i_t \tag{3.10}$$

$$c_t = j_t + k_t \tag{3.11}$$

$$o_t = \sigma(U_o h_{t-1} + W_o x_t) \tag{3.12}$$

$$h_t = o_t \circ \tanh(c_t) \tag{3.13}$$

En estas ecuaciones, 3.6 y 3.7 hacen referencia al cómputo del vector de información a olvidar, actuando a modo de máscara. Como se aprecia, sirve para computar qué información del vector de contexto[15] previo c_{t-1} debe combinarse con el resto de información que entra en la capa. Por otro lado, 3.8, 3.9 y 3.10 permiten obtener la información que queremos extraer a partir del estado previo de la red y de la entrada actual, dando como resultado el vector que contiene el estado de la celda actual c_t. Este vector resultante en 3.11 es la suma elemento a elemento del vector de retención de información nueva g_t junto con el de olvido k_t. Esto supone que c_t contiene el vector de contexto/memoria en t. Este estado contiene la información relevante que debe fluir al siguiente estado para predecir la salida. Finalmente, con la puerta de *output* dado por 3.12 se calcula el vector de salida o_t. Este vector, multiplicado elemento a elemento con el vector de contexto c_t calculado previamente, dan como resultado el estado interno h_t en 3.13. Es con este vector h_t con el que se realizan predicciones en cada t. La figura 3.16 muestra todo el grafo computacional de esta arquitectura.

3.5.3.4 Tareas y codificaciones. *Embeddings*

En anteriores apartados hemos definido que una tarea puede ser cualquier tipo de actividad que realice una red neuronal o algoritmos de ML. Las tareas que realice un algoritmo pueden estar o no relacionadas, siendo tareas que compartan o no características. Por su parte, un dominio es un conjunto de características que comparten una serie objetos, entidades, etc.

La codificación de tareas y dominios en el ML es una parte importante, ya que permite a nuestros algoritmos recibir información de una manera más compacta, eficiente y sencilla. Por ejemplo, si nuestro algoritmo de ML tiene que clasificar palabras de un diccionario de un idioma como positivas, negativas o neutras, disponer de una buena codificación que permita representar cada palabra será un gran avance en cuanto a recursos y representación, facilitando la convergencia del algoritmo. Para codificar distintas entidades como palabras, se suelen utilizar 4 tipos de representación: representación en crudo, representación categórica, representación con un formato *one-hot encoding* y representación en forma de *embedding*. Las dos últimas son las más utilizadas.

[15] Este vector de contexto también se denomina comúnmente carrusel de error constante (CEC)

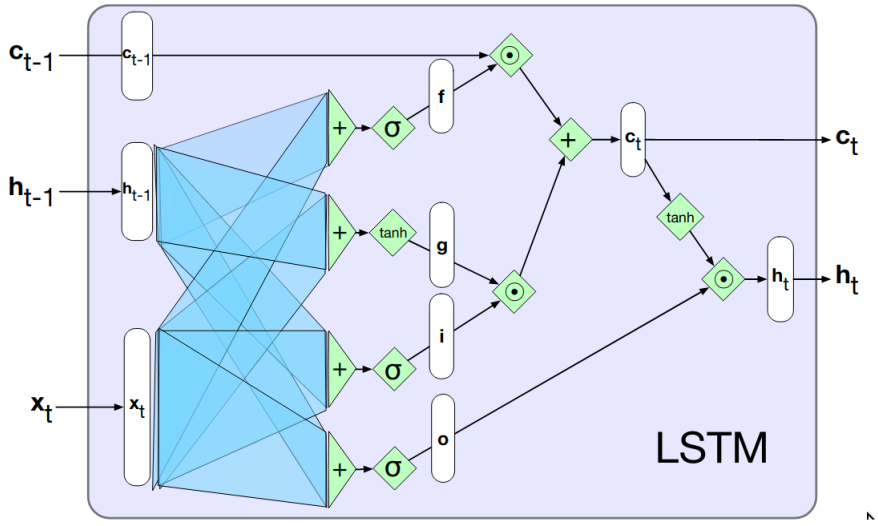


Figure 3.16: Grafo computacional completo de una unidad en una LSTM. Fuente: Jurafsky & Martin (2009)

Raw-representation. Esta representación no es realmente ningún tipo de formato concreto. En la computación y en la informática en general, todo se representa a bajo nivel con números (reales o enteros), y todo lo que salga de ahí acaba siendo una complicación que ha de solventarse. En concreto, en ML, es prácticamente imposible utilizar representaciones en crudo de valores como palabras, dado que se desperdiciaría una gran cantidad de recursos computacionales solo por el hecho de trabajar con estos datos en crudo. Lo que suele hacer es pasar a otros tipos de representaciones intermedias, de tal forma que exista una relación con una instancia o ejemplo en concreto, facilitando a los algoritmos la lectura y el procesamiento.

Categorical representation. La representación categórica (discreta, en forma numérica) representa las entidades o instancias con el conjunto de números $\mathbb{N}$. Es una representación intermedia válida y suficiente en algunos casos. Sin embargo, hay otros casos en los que no queremos forzar a nuestros algoritmos o programas a que se tenga que establecer un orden entre las distintas instancias o entidades. Por otra parte, los números naturales dan poca flexibilidad a la hora de realizar una representación de ellos y distancias muy próximas (en un mapa de coordenadas) no pueden representarse si no es con un error muy grande. Para este tipo de representaciones en las que no se quiere establecer un orden dentro de un conjunto de instancias, se suele utilizar un tipo de representación codificada llamada *one hot encoding*.

One Hot encoding. La representación *one-hot* es una representación con un formato binario, de tal forma que un bit representa una categoría o instancia posible. La tabla 3.1 muestra un ejemplo de representación de tres clases en formato *one hot*. A pesar de ser una buena representación, tiene varios problemas: en primer lugar, todas las clases están a la misma distancia en una representación espacial, lo que representa una restricción grande. Por otra parte, los recursos computacionales utilizados con este tipo de representación cuando el número de instancias distintas aumenta es casi intratable. En el ejemplo de la representación

Categoría	ID	One-hot		
Perro	0	1	0	0
Gato	1	0	1	0
Pájaro	2	0	0	1

Table 3.1: Ejemplo de una representación de tres clases en formato *one hot*.

de palabras de un diccionario, asumiendo que hay 50.000 palabras distintas en una lengua, se necesitaría un vector de 50.000 dimensiones para representar una sola palabra. Para representar todo el diccionario, haría falta unos 0,3125 *Gigabytes* solo en representación.

Embeddings. Es un tipo de representación distribuida para un tipo de datos categórico.Todas las muestras del diccionario se representan en vectores del mismo tamaño. Los valores de dichos vectores son parámetros que se aprenden o bien en el proceso de entrenamiento de la red o en tareas de aprendizaje autosupervisado (Mikolov et al., 2013). El espacio vectorial provee una proyección de las categorías, lo cual permite que estas se relacionen de manera natural. De esta forma, forzamos a la red a que las clases similares obtengan representaciones parecidas, mientras que el resto estará más alejado. Esta representación presenta los beneficios de poder ordenar relaciones a partir de los datos y provee una forma de representación vectorial para cada categoría, sin realizar grandes gastos en memoria. Además, otorga flexibilidad, ya que permite usar las dimensiones de estos vectores y agrupar según sub-categorías según relaciones sintácticas, semánticas, etc.

Como contrapunto, este sistema añade de complejidad al modelo, ya que debe aprender los pesos del vector en el proceso de aprendizaje. Sin embargo, se obtiene una separación natural de las instancias realizada en el proceso de entrenamiento, lo que otorga una mayor flexibilidad y distintas posibilidades de aplicaciones como *clustering*, clasificación de entidades a partir de esta representación, etc. Relacionando con conceptos previos, una de las ideas detrás de los *embeddings* es que la representación que se realiza de las distintas clases se represente en un formato comprimido (representación del espacio latente), destacando las características más relevantes de esa clase o instancia reduciendo su dimensionalidad. La figura 3.17 ilustra esta representación en forma de *embeddings* para películas que comparten ciertas características en dos dimensiones.

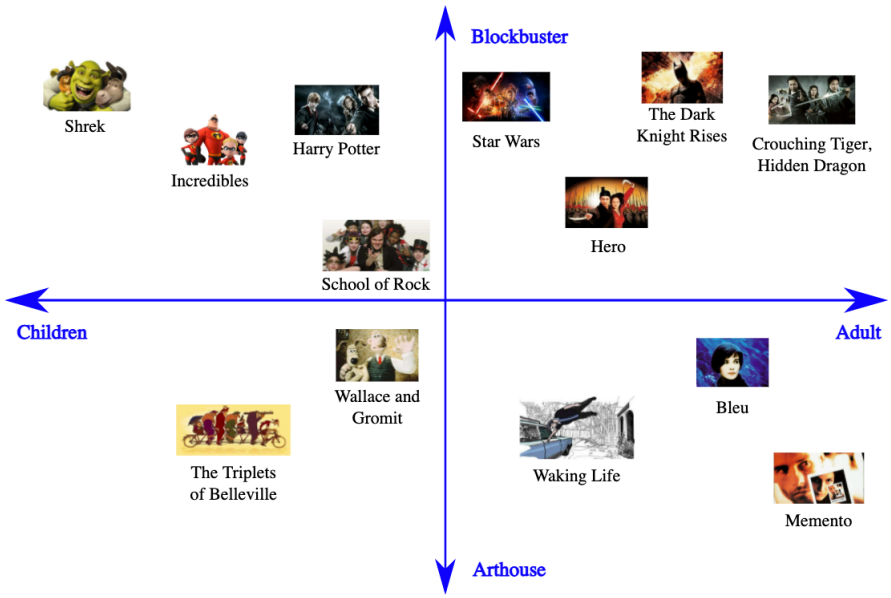


Figure 3.17: Representación en forma de *embedding* en un eje de coordenadas de dos dimensiones. Eje X: Público al que va dirigida la película. Eje Y: Representación de la película en función de lo taquillera o lo artístico de ella. Fuente: developers.google.com/machine-learning/

4 Materiales y herramientas

En este capítulo se describen los materiales y herramientas utilizadas para desarrollar el trabajo.

4.1 Datos utilizados

En este apartado se describen los datos utilizados durante el desarrollo de los tres trabajos. Es importante recalcar que el *dataset* empleado en cada uno de ellos es el mismo, y únicamente cambia la forma en la que este se utiliza.

MUSCIMA++ Jan Hajič & Pecina (2017) es un *dataset* de notación musical manuscrita que se concibió originalmente para la detección de símbolos musicales. Contiene 91255 símbolos, que consisten tanto en primitivas de notación musical como en objetos de notación de nivel superior. En el conjunto de datos hay etiquetadas 23352 notas, de las cuales 21356 tienen una cabeza de nota completa, 1648 tienen una cabeza de nota vacía y 348 son notas de gracia. Para cada objeto anotado en una imagen, este *dataset* proporciona tanto el cuadro delimitador (comúnmente llamado *bounding box*) como una máscara de píxeles que define exactamente qué zona pertenece al objeto en cuestión. Las construcciones compuestas, como las notas, se capturan a través de relaciones explícitamente anotadas de las primitivas de notación (cabezas de nota, plicas, barras...), formando así la (*Notation Graph*) MUSCIMA++, o MuNG. De este modo, la anotación proporciona un puente explícito entre los símbolos de bajo y alto nivel descritos en la literatura de OMR. Este *dataset* contiene anotaciones para 140 imágenes (partituras) del conjunto de datos CVC-MUSCIMA Fornés et al. (2012), utilizado para la identificación de escritores de notación musical manuscrita y la eliminación de pentagramas. La figura 4.1 muestra ejemplos de partituras musicales extraídas de este conjunto de datos.

En este trabajo, estamos interesados en tres tipos de datos según la tarea en cuestión: (1) la extracción de símbolos y sus relaciones para cada partitura de manera completa; (2) la extracción de nodos a partir de subgrafos (estructuras musicales); (3) la extracción de subgrafos completos (nodos y aristas a partir de estructuras musicales completas). Por tanto, utilizaremos la librería MuNG[1] descrita anteriormente. Este *dataset* contiene un grafo por cada una de las 140 partituras, en la que los nodos corresponden a las primitivas musicales (símbolos) y las aristas determinan si existe o no una relación entre cada par de símbolos.

Cada tipo de extracción realizada se describirá con más detalle en cada uno de los tres trabajos desarrollados.

[1] Para más detalles sobre la librería y las anotaciones de las partituras, consultar `https://mung.readthedocs.io/en/latest/`.

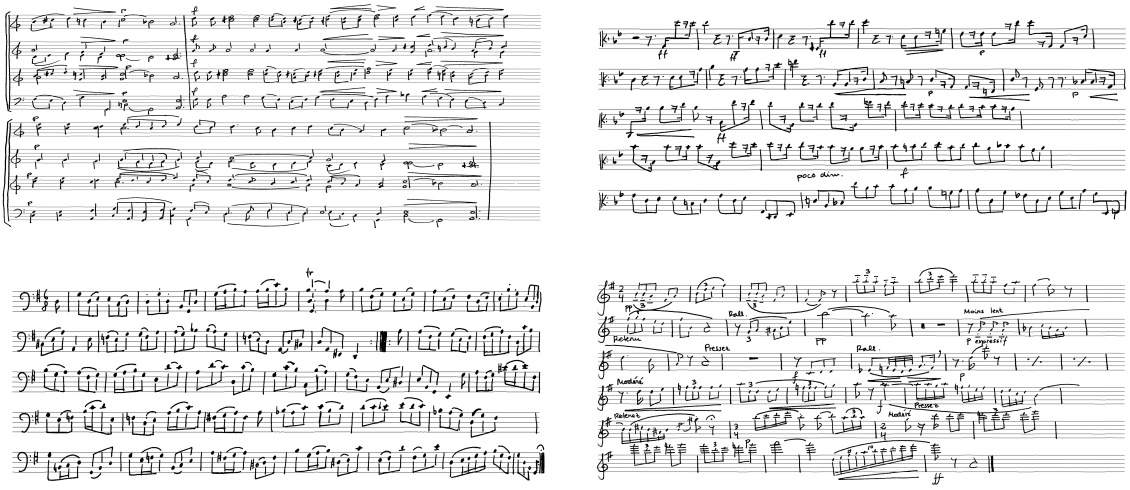


Figure 4.1: Ejemplos de distintas partituras musicales extraídas de la MUSCIMA++. Las figuras están redimensionadas para ocupar anchuras y alturas similares. Sin embargo, cada partitura tiene un tamaño distinto del resto.

4.2 Herramientas

En esta sección, describimos las herramientas utilizadas en el desarrollo del trabajo, tanto a nivel *software* como *hardware*.

4.2.1 Desarrollo del software

Python

Python[2] es un lenguaje de programación de alto nivel creado por Guido van Rossum en 1991. Es un lenguaje multiparadigma, centrado principalmente en el orientado a objetos y la programación imperativa, e incluye una gran cantidad de librerías para ampliar sus funcionalidades. Las librerías principales se centran ámbitos como la computación científica, como son las matemáticas, las de aprendizaje automático, las de visión por computador, etc. Se ha utilizado Python (versión 3.7) para todo el desarrollo del proyecto. Esto incluye el preprocesado de datos, la creación y entrenamiento de las redes neuronales, la experimentación y la visualización de resultados.

Shell Scripting

Un script de *shell* es un programa informático diseñado para ser ejecutado por el terminal de Unix, un intérprete de línea de comandos. Las operaciones típicas realizadas por los scripts del *shell* incluyen la manipulación de archivos, la ejecución de programas y la impresión de texto. En este trabajo, se ha utilizado *shell* scripting para lanzar un "meta-programa", el cual se encarga de llamar al código Python que realiza el entrenamiento o prueba de los modelos en las distintas iteraciones de la validación cruzada descrita en los experimentos.

[2] `https://www.python.org`

Librerías específicas

En el desarrollo del trabajo, se han utilizado las siguientes librerías específicas para Python:

PyTorch. PyTorch[3] es un *framework* de código abierto para implementar, entrenar y mantener algoritmos de *machine/deep Learning*. Fue desarrollado principalmente por los investigadores e ingenieros de *Facebook AI Research* (FAIR). Se lanzó en 2016, y ha adquirido gran popularidad debido a su facilidad de uso y a su orientación hacia la investigación. Permite un rápido prototipado de redes neuronales y su sencilla integración a nivel hardware es idóneo para tareas tanto genéricas como de investigación. Se ha utilizado PyTorch para todo el desarrollo y entrenamiento de los modelos en cada uno de los trabajos. La versión utilizada ha sido la 1.10.0.

PyTorch-lightning. Pytorch Lightning[4] Es una librería de código abierto escrita en Python que provee una interfaz de alto nivel para PyTorch, podría establecerse como un programa análogo a Keras y Tensorflow. Organiza el código de tal forma que desacopla al máximo las tareas de "ingeniería" de las de investigación. Permite de manera sencilla crear, monitorizar y mantener modelos de *deep learning*. Se ha utilizado PyTorch Lightning para todo el desarrollo y entrenamiento de los modelos en cada uno de los trabajos. La versión utilizada ha sido la 1.5.2.

Matplotlib. Matplotlib[5] es una librería utilizada en la creación y visualización de gráficos estáticos o interactivos en Python. Se ha utilizado la librería para la creación de gráficas sobre los resultados del trabajo. La versión utilizada de Matplotlib ha sido la 3.5.

Seaborn. Seaborn[6] es una librería de visualización de datos en Python basada en matplotlib. Proporciona una interfaz de alto nivel para dibujar gráficos atractivos e informativos. Se ha utilizado con el objetivo de crear gráficas sobre los resultados en los distintos trabajos con una estética mejorada con respecto a Matplotlib. En concreto, usamos la versión 0.11.2.

Pandas. Pandas[7] es una librería especializada en la gestión y el análisis de estructuras de datos de manera eficiente. Añade funcionalidades como lectura y escritura de distintos tipos de ficheros (CSV, Excel, SQL). Se ha utilizado en el análisis de resultados y la creación de gráficas. Se ha utilizado la versión 1.3.5 de Pandas.

OpenCV. OpenCV[8] (Open Source Computer Vision Library) es una librería de código abierto de visión artificial (*computer vision*) y ML. Se ha utilizado esta librería para el tratamiento y procesado de imágenes. La versión utilizada ha sido la 4.5.1.

[3] `https://pytorch.org/`
[4] `https://www.pytorchlightning.ai/`
[5] `https://matplotlib.org/`
[6] `https://seaborn.pydata.org/`
[7] `https://www.pandas.pydata.org`
[8] `https://opencv.org/`

Scikit-learn. Scikit-learn[9] es una librería para aprendizaje automático de código abierto para Python. Incluye multitud de algoritmos para aprendizaje supervisado como no supervisado, así como métricas básicas para clasificación, regresión, etc. Se ha utizado scikit-learn principalmente para el cálculo de métricas como el *precision*, ek *recall* y el F_1. La versión utilizada ha sido la 1.0.1.

Tensorboard. TensorBoard[10] es un kit de herramientas de código abierto que permite entender el progreso del entrenamiento y mejorar el rendimiento del modelo mediante la actualización de los hiperparámetros. El kit de herramientas TensorBoard muestra un panel de control en el que los registros pueden visualizarse como gráficas a modo de imágenes, histogramas, texto, etc. Además, ayuda a trazar información como gradientes, pérdidas, métricas y resultados intermedios.

4.2.2 Hardware utilizado

El hardware utilizado para el desarrollo del trabajo ha sido mediante un servidor local con las siguientes características:

- Procesador: Intel Core i9-11900K, Intel Z590
- RAM: 2 x 16GB DDR4 3200Mhz
- Tarjeta gráfica: Nvidia GeForce RTX 3080 10GB.

[9] `https://scikit-learn.org/stable/`
[10] `https://www.tensorflow.org/tensorboard/`

5 Metodología y experimentación

En este capítulo se describe toda la metodología y experimentación correspondientes a este trabajo. Con el objetivo de facilitar la tarea de lectura en cada una de las propuestas, cada uno de ellos se ha dividido en 3 secciones en los que se presenta la metodología; experimentación, resultados y conclusiones. En la metodología, se introduce la formulación del trabajo y se presentan los enfoques propuestos. Por su parte, en la experimentación se describe la extracción de datos, las métricas utilizadas, los distintos escenarios de evaluación y los detalles de los enfoques propuestos. Por último, en las secciones de resultados y conclusiones se describen los principales descubrimientos en cada trabajo, realizando una discusión sobre ellos y donde se exponen posibles trabajos futuros.

5.1 Recuperación de relaciones

5.1.1 Metodología

Este trabajo sigue la formulación propuesta en trabajos anteriores Jan Hajič & Pecina (2017); Pacha et al. (2019); Garrido-Munoz et al. (2022), donde se asume que la lectura computacional de una partitura, en el contexto de OMR, puede ser descrita recuperando una estructura de grafos a partir su imagen correspondiente. En este grafo, los elementos de notación (denominados "primitivas") representan los nodos, mientras que las relaciones entre ellos son las aristas. En este trabajo, nos interesa particularmente la recuperación de las aristas, una vez que los nodos han sido ya detectados (mediante, por ejemplo, enfoques de detección de objetos para OMR).

5.1.1.1 Formulación

Formalmente, un grafo es una estructura matemática que modela las relaciones por pares entre elementos (referidos como nodos o vértices) a través de sus aristas. En este caso, nuestro objetivo es recuperar las aristas (relaciones) entre cada par de nodos en las partituras, donde cada nodo representa una primitiva musical, por ejemplo, una cabeza de nota, una plica o una alteración.[1] La definición formal del problema es la siguiente.

Suponemos que para una partitura dada $\mathbf{s}$ existe un grafo $g_{\mathbf{s}}$ que representa su contenido. El grafo se define como un par (V, E), donde V representa el conjunto de nodos y E el conjunto de aristas. Dos nodos $v_i, v_j \in V$ están conectados si existe una arista $e_{i,j} = (v_i, v_j) \in E$.

En el contexto de OMR, la información sobre el conjunto de símbolos V puede obtenerse a través de las técnicas existentes (cf. Sec. 2). Esto corresponde a la etapa *detección de símbolos musicales* del *pipeline* de OMR. Por lo tanto, aquí asumimos que existe una función que mapea $\mathbf{s}$ en un conjunto V. Típicamente, cada símbolo $v_i \in V$ se representa además como

[1] A partir de ahora, utilizaremos los términos "nodo", "símbolo" y "primitiva" indistintamente: un elemento gráfico colocado en la partitura con ciertos atributos.

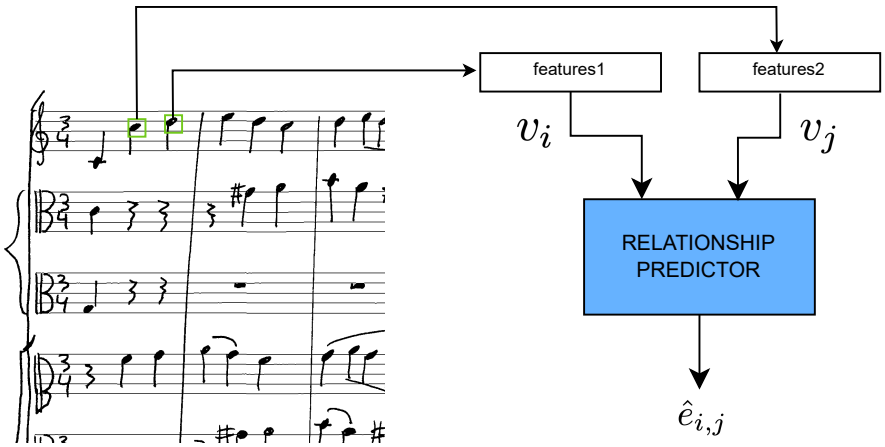


Figure 5.1: Esquema general de la metodología para recuperar las aristas del grafo de notación musical.

un conjunto de características con, al menos, la siguiente información: etiqueta de clase y coordenadas dentro de la partitura en forma de imagen. El problema que abordamos aquí es cómo obtener E dado V, que corresponde a la etapa de ensamblaje de la notación musical del pipeline de OMR.

El problema puede considerarse desde dos puntos de vista, (i) como una tarea de clasificación binaria en la que un modelo predice la clase de cada par de nodos v_i, v_j presentes en cada partitura y (ii) como una reconstrucción de la matriz de adyacencia A de un grafo dado, que representa todas las posibles relaciones entre un conjunto de nodos. Con el objetivo de simplificar, nuestro enfoque es resolver el problema (i), ya que es sencillo convertir el conjunto de relaciones recuperadas en una matriz de adyacencia.

En esta clasificación binaria, $e_{i,j}$ se etiqueta como 1 si hay relación entre v_i y v_j y 0 en caso contrario. La predicción de la relación puede representarse como una función $\varphi(v_i, v_j)$ que toma las características de los dos nodos como entrada y calcula la probabilidad de conexión. La figura 5.7 representa un esquema general de la metodología adoptada en este trabajo.

5.1.1.2 Enfoques

Es importante destacar de la formulación descrita anteriormente que el coste de predecir cada posible arista conlleva una complejidad $O(|V|^2)$. Por tanto, las aproximaciones a φ deben tener en cuenta el coste computacional para que la tarea sea factible en la práctica.

Aquí proponemos dos arquitecturas neuronales superficiales que toman un par de nodos y predicen la clase de la relación. Estas dos arquitecturas neuronales son (i) una arquitectura de Perceptrón Multicapa (MLP) que toma la entrada de las características de cada nodo concatenadas y (ii) un modelo de kernel asimétrico (*asymmetric kernel*).

Arquitectura con MLP. En este método, primero se concatenan las características–atributos– de los nodos, formando un único vector de representación que contiene toda la información de dicho par de nodos. A continuación, este vector se procesa por una serie de capas de un

MLP. La última capa implementa una función σ que modela la probabilidad de que los dos nodos de entrada se conecten:

$$\hat{e}_{i,j} = \sigma(\varphi_{MLP}([v_i, v_j]))$$

Asymmetric kernels. En este segundo esquema, nuestra arquitectura neuronal propuesta aprende una función de kernel asimétrico Wu et al. (2010). Esta función se define por $k(v_1, v_2) = (\langle \phi_{k_1}(v_1), \phi_{k_2}(v_2) \rangle)$, donde $\langle \cdot, \cdot \rangle$ es el producto punto de dos puntos N-dimensionales en dos espacios de Hilbert (espacios de características). En este trabajo, utilizamos este kernel asimétrico como una función de similitud entre las dos características mapeadas a distintos espacios de Hilbert[2].

$$\hat{e}_{i,j} = \sigma(\langle \phi_{k_1}(v_i), \phi_{k_2}(v_j) \rangle) \tag{5.1}$$

En este enfoque, $\phi_{k_1}(v_1), \phi_{k_2}(v_2)$ son kernels implementados como MLP que mapean las características del nodo inicial en dos espacios de representación diferentes (asimétricos). Se supone que este espacio representa las características apropiadas para la tarea en cuestión. Tras calcular la puntuación de similitud, se aplica una función σ para obtener probabilidades entre 0 y 1.

Lo realmente interesante de este enfoque es su eficiencia computacional: los *embeddings* se calculan independientemente para cada nodo. Luego, para cada relación posible, solo es necesario calcular el producto escalar (*dot product*) entre los *embeddings* de los nodos y aplicar la función σ. Por eso la complejidad computacional es mucho menor que la del enfoque anterior, ya que apenas hay cálculos específicos del orden $O(|V|^2)$.

Función de pérdida. En ambas arquitecturas neuronales propuestas, el objetivo es minimizar la función de pérdida binaria de entropía cruzada

$$\mathcal{L}_{\text{BCE}} = \sum_{e_{ij} \in A} e_{ij} \log(\hat{e}_{ij}) + (1 - e_{ij}) \log(1 - \hat{e}_{ij}) \tag{5.2}$$

donde $\hat{e}_{ij}$ corresponde a la probabilidad predicha por el modelo y e_{ij} es el *ground truth* (1 para una relación positiva, 0 en caso contrario).

5.1.2 Experimentos

En este apartado, describimos la configuración desarrollada para evaluar las dos arquitecturas neuronales propuestas. Describimos los datos y las métricas utilizadas para los experimentos, y luego proporcionamos algunos detalles de la implementación de las arquitecturas.

5.1.2.1 Datos y extracción

Los experimentos se han llevado a cabo utilizando el *dataset* MUSCIMA++ Jan Hajič & Pecina (2017), el cual ya hemos descrito anteriormente.

[2]Este espacio de Hilbert puede verse realmente como un espacio N-dimensional euclídeo de características

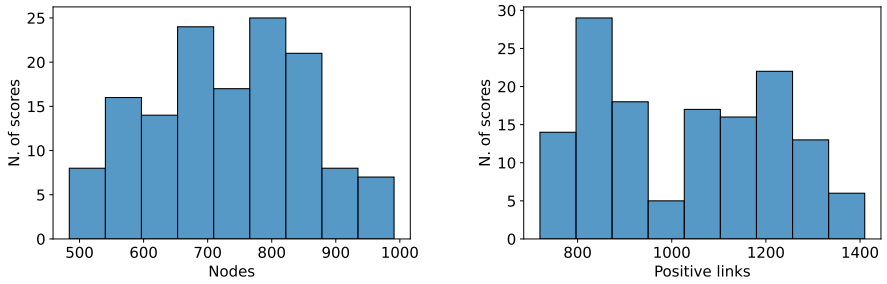


Figure 5.2: Distribución del número de nodos y de aristas positivas en el dataset MUSCIMA++.

Para esta tarea de recuperación de relaciones, utilizamos los símbolos y relaciones anotados para todas las partituras de 140 incluidas en el conjunto de datos MUSCIMA++. En este trabajo, asumimos que cada nodo está representado por un conjunto de atributos disponibles: la clase del símbolo y su información espacial -coordenadas cartesianas como *bounding boxes*- normalizadas al tamaño de la partitura de entrada.

La figura 5.2 muestra algunas estadísticas sobre la distribución de los datos en términos de nodos y aristas. Es importante destacar el bajo número de relaciones con respecto a todos los pares de nodos posibles. Las relaciones positivas representan aproximadamente el 0,2% del total de aristas posibles.

Para extraer todos los símbolos musicales y relaciones, se ha utilizado la librería MuNG comentada anteriormente. Esta librería permite procesar cada partitura, para obtener los atributos de los nodos (símbolos) anteriormente comentados y las aristas que les relacionan. Dado que queremos obtener tanto las relaciones positivas como las negativas, se ha creado un conjunto de datos añadiendo las aristas obtenidas (relaciones positivas), y posteriormente se han añadido todas las relaciones que no estuviesen en las positivas como negativas, mediante un procesando nodo a nodo. De esta forma, el *dataset* final tiene un formato tabular, en el que cada fila contiene los atributos del primer nodo (clase, *bounding boxes*), del segundo nodo y la clase de la relación (1 si es positiva, 0 si es negativa), lo que supone un total de once valores.

5.1.2.2 Métricas

Teniendo en cuenta las estadísticas presentadas anteriormente, estamos ante un problema de clasificación desequilibrado. Dado que nos interesa obtener las aristas de los grafos que representan las partituras musicales, nos centraremos en métricas orientadas a evaluar las relaciones positivas.

Por esta razón, utilizamos el *precision* (P), el *recall* (R) y el F_1 sobre las relaciones positivas para medir la calidad de la recuperación. Estas métricas se definen como

$$\mathrm{P} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}},$$
$$\mathrm{R} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}},$$
$$\mathrm{F}_1 = 2\,\frac{\mathrm{P} \cdot \mathrm{R}}{\mathrm{P} + \mathrm{R}},$$

donde TP, FP y FN son verdaderos positivos, falsos positivos y falsos negativos, respectivamente. En nuestros resultados, multiplicaremos estas métricas por un factor de 100 para que puedan interpretarse como un porcentaje.

5.1.2.3 Detalles de implementación

En este trabajo, propusimos dos métodos para clasificar las relaciones entre los símbolos musicales, las cuales se describieron conceptualmente en la sección 5.1.1.

Para la arquitectura MLP genérica, utilizamos tres capas *fully-connected* (una capa de entrada, dos capas ocultas y una capa de salida) con activaciones $ReLU$ en todas sus capas ocultas. El número de unidades en las capas ocultas es de 64 y 512, respectivamente, y una unidad en la capa de salida para calcular el *score* de la clasificación binaria.

En cuanto a los kernels asimétricos, ϕ_{k_1}, ϕ_{k_2} se implementan como un MLP diferente de cuatro capas (512, 1024, 512 y 256 unidades, respectivamente) con la función ReLU. La idea es generar dos incrustaciones de 256 dimensiones, que corresponden a dos puntos en diferentes espacios de Hilbert, y luego calcular la similitud a través del producto escalar.

En ambos casos, la última operación se implementa como una función de activación *sigmoide* para entender la salida como una probabilidad de una relación positiva. La política de decisión para este trabajo considera un umbral positivo de 0.5 en todos los casos.

Ambas arquitecturas reciben dos nodos como entrada. Cada nodo está representado inicialmente por un vector de características de cinco dimensiones: la etiqueta y cuatro valores normalizados que indican las esquinas superior izquierda e inferior derecha, que representan el cuadro delimitador dentro de la partitura de entrada. Además, ambas arquitecturas procesan la clase con una capa de *embedding* con 16 dimensiones como salida. Por lo tanto, cada nodo se representa finalmente como un vector de características de 21 dimensiones en ambas arquitecturas.

En cuanto al proceso de optimización, ambos modelos se entrenaron durante 200 épocas utilizando el optimizador Adam Kingma & Ba (2015) con un *learning rate* de 1×10^{-3}.

Por último, para proporcionar cifras de resultados más robustas, seguimos un esquema de validación cruzada de cinco iteraciones (5-CV). En cada iteración, se utiliza un 60% del conjunto de datos para el entrenamiento, 20% para la validación y 20% para test.

5.1.3 Resultados

En esta sección, presentamos y discutimos los resultados obtenidos. En concreto, estudiamos el rendimiento a dos niveles: eficacia y eficiencia. Para este estudio, incluimos los dos enfoques propuestos, desglosados además por las características del nodo consideradas: solo la etiqueta, solo las coordenadas, o ambas características. Esto nos dará una idea de qué características

Table 5.1: Media de los resultados (5-CV) en términos de F_1 con respecto al modelo y las características consideradas.

Características		Kernels Asim.			MLP		
Clase	Coordenadas	P	R	F_1	P	R	F_1
✓	✓	67.3	89.6	76.8	91.4	92.4	91.9
✓	✗	0.5	79.7	1.0	0.3	50.4	0.4
✗	✓	49.4	81.2	61.4	74.2	82.7	79.6

son importantes para la recuperación de relaciones. Además, para establecer una referencia en la efectividad que se puede obtener para esta tarea, incluimos los resultados de Pacha et al. Pacha et al. (2019), medidos en condiciones comparables. Sin embargo, su coste de tiempo es tan elevado (se mostrará más adelante) que hace inviable su integración en un sistema real. Por lo tanto, solo debe considerarse como un límite superior del rendimiento para la tarea de recuperación de relaciones.

En primer lugar, la Fig. 5.10 muestra el rendimiento medio de la validación cruzada (5-CV) en términos de F_1. Lo primero que se debe destacar es que las características juegan un papel fundamental en ambos enfoques. La etiqueta de clase, como cabe esperar, apenas es informativa por sí misma. En tal caso, los resultados son pobres (1,0 % de F_1, como mucho). Sin embargo, consigue complementar notablemente el rendimiento obtenido utilizando solo las características geométricas de los nodos (es decir, la posición en la imagen). En el caso donde se tiene en cuenta ambos tipos de características, el método MLP alcanza el 91,9 % de F_1, muy cerca del límite superior establecido por el trabajo de Pacha et al. Pacha et al. (2019). El método del kernel asimétrico, en cambio, ofrece un rendimiento correcto pero mucho más bajo (76,8 % de F_1).

Para proporcionar más información sobre nuestros resultados, desglosamos en la Tabla 5.1 nuestros resultados también en términos de P y R. Observamos una tendencia consistente en los resultados para los kernels asimétricos: el *recall* es mucho mejor que la precisión. Esto significa que el método tiene una tendencia a establecer relaciones positivas (R alta), aunque muchas de las veces se equivoca (P baja). Esto se debe probablemente a que las relaciones positivas son de naturaleza más regular y, por tanto, más fáciles de aprender, mientras que las relaciones negativas son mucho más variables e imprevisibles. El método MLP, en cambio, parece no verse afectado por este fenómeno y mantiene un buen equilibrio entre P y R.

Para complementar estos resultados, realizamos una visualización representativa de su implicación en las partituras musicales en la Fig. 5.6. Los TP, FP y FN de cada método se indican en diferentes colores. Obsérvese el gran número de relaciones debe calcular (a pesar de no destacar las aristas correctamente ignoradas), lo que demuestra la complejidad computacional de la tarea. Precisamente por esta cuestión, a continuación estudiamos el tiempo que tardan los distintos métodos en alcanzar estos resultados.

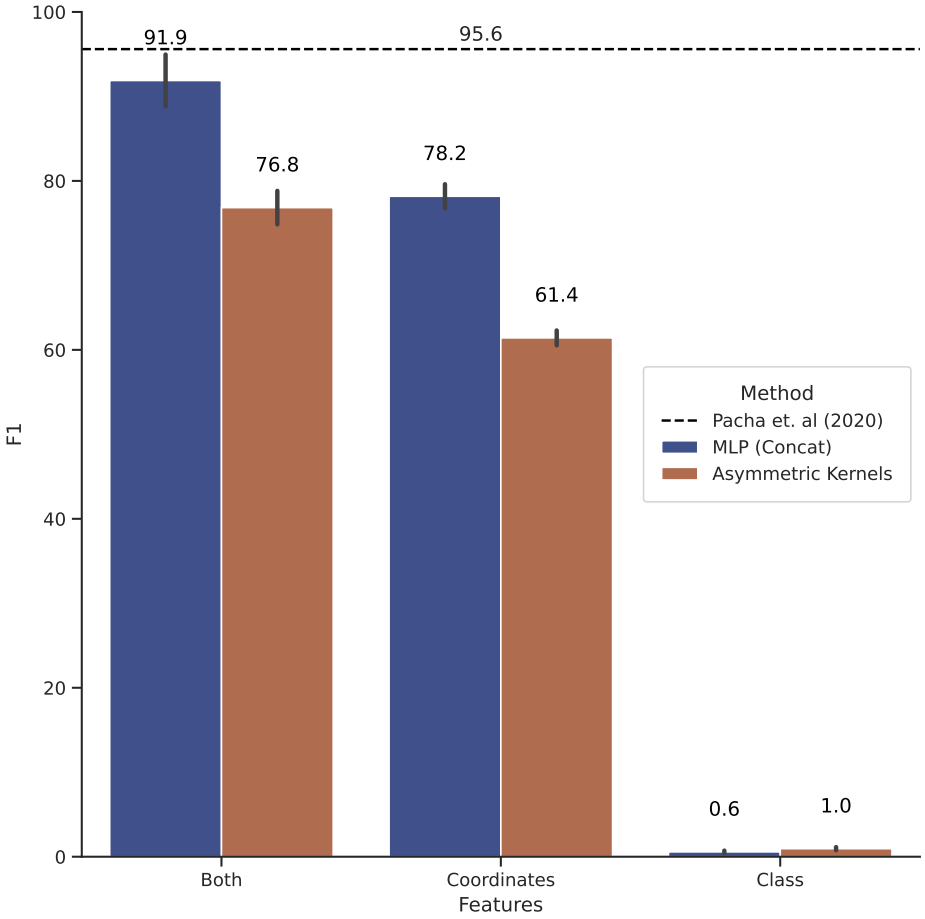


Figure 5.3: Media de los resultados (5-CV) en términos de F_1 con respecto al modelo y las características consideradas. La línea punteada representa el resultado de Pacha et al. Pacha et al. (2019) (*upper bound*).

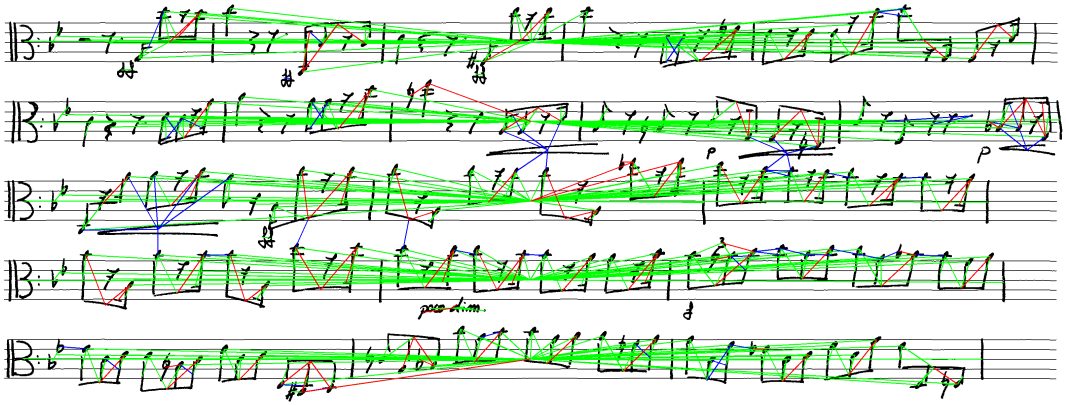


Figure 5.4: Recuperación de aristas por el método del *MLP*.

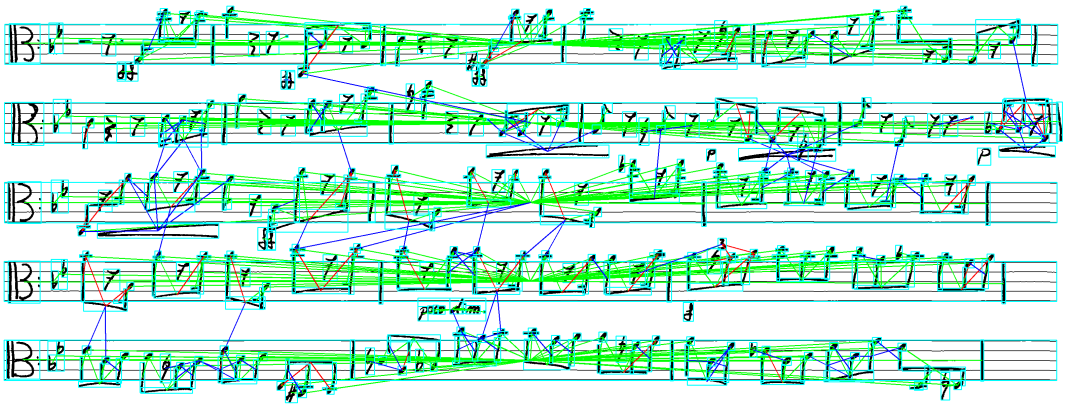


Figure 5.5: Recuperación de aristas por el método *Asymmetric Kernels*.

Figure 5.6: Visualización del rendimiento de los enfoques. Las líneas verdes representan TP, mientras que las líneas azules y rojas representan FP y FN, respectivamente. Las cajas turquesas representan los nodos.

5.1.3.1 Comparativa de tiempos

En esta sección, analizamos el rendimiento temporal en tiempo de inferencia para cada método considerado. La tabla 5.2 muestra el tiempo medio de procesamiento y las relaciones de los nodos (uno vs. todos) que pueden ser analizadas por segundo en función del número de

Table 5.2: Rendimiento temporal de cada método propuesto según el número de nodos en una partitura. Los tiempos mostrados son el resultado de una media tras 10 ejecuciones.

Nº. nodos	**Método**	**Tiempo (s)**	**Nodos/s**
500	Kernels asimétricos	3.5×10^{-4}	1,417,313.0
	MLP	9.5×10^{-2}	5,234.4
	CNN (Pacha et al., 2019)	336.2	1.5
1,000	Kernels asimétricos	3.01×10^{-4}	3,325,810.0
	MLP	1.7×10^{-1}	5,922.4
	CNN (Pacha et al., 2019)	1,346.0	0.7

nodos de la partitura.[3] Para esta comparación, consideramos dos partituras con 500 y 1.000 símbolos, a partir de los cuales tenemos que computar todas las relaciones posibles.

Como se puede observar, la CNN propuesta por Pacha et al. Pacha et al. (2019) tarda varios minutos en completar una partitura (5 en el caso de 500 elementos y 22 en el caso de 1.000 elementos). Esto significa que no podría integrarse de forma viable en ningún sistema que requiera la interacción del usuario. El MLP sí consigue una eficiencia bastante aceptable, con apenas unos milisegundos por partitura. Además, el enfoque del kernel asimétrico resulta extremadamente eficiente.

Obsérvese que el método del kernel asimétrico aprovecha la paralelización de la operación del producto escalar y el procesamiento de nodos independientes. No ocurre lo mismo con los otros métodos donde, debido a la forma en la que clasifican, requieren más cálculo. En el caso de MLP, el enfoque debe generar primero todos los posibles pares de combinaciones entre símbolos y luego predecir mediante concatenación de sus características. En el caso de Pacha et al. Pacha et al. (2019), primero hay que construir una imagen y luego procesarla con una CNN para cada par de nodos. Aunque hay cinco bloques de convoluciones, pooling, y normalización por lotes, las imágenes de entrada son muy grandes (256×512), lo que hace que el método sea excesivamente lento.

Estos resultados demuestran que no hay ningún método que supere al resto tanto en precisión como en eficiencia. Sin embargo, podríamos asumir el método MLP propuesto como el mejor candidato para la práctica hasta la fecha, dado que obtiene un muy buen F_1 (muy cercano al límite superior proporcionado por la CNN) mientras que su eficiencia es suficiente para un sistema interactivo.

5.1.4 Conclusión

En este trabajo estudiamos la recuperación de las relaciones entre símbolos de las partituras, lo que se denomina *ensamblado de notación musical* en la literatura de OMR. Abordamos el

[3]Solo por comparar los tiempos de ejecución, se ha implementado la arquitectura de CNN propuesta en Pacha et al. Pacha et al. (2019) con los detalles proporcionados en dicho artículo.

problema como una tarea de clasificación binaria en la que el objetivo es predecir si existe una relación entre cada par de nodos que han sido detectados previamente.

Propusimos dos métodos que se han demostrado empíricamente como factibles para resolver la tarea en cuestión: una arquitectura MLP que funciona con mucha precisión a costa de un coste computacional (poco) mayor y un modelo de kernel asimétrico que funciona extremadamente rápido a costa de una notable pérdida de precisión. Además, demostramos empíricamente que todas las características consideradas (etiquetas de clase y coordenadas de la caja delimitadora) son relevantes para el rendimiento.

Consideramos que este trabajo demuestra la necesidad de considerar la relevancia sobre el ensamblado de la notación en OMR, dado que no es trivial resolver la tarea si atendemos tanto a cuestiones de precisión como de eficiencia, algo que desarrollaremos en las conclusiones de este proyecto.

En este trabajo, se ha estudiado la recuperación de relaciones partiendo de una detección previa de los símbolos. Sin embargo, ¿es posible recuperar estos símbolos de manera precisa? En la siguiente sección se estudia esta recuperación de símbolos a partir de extracciones de partituras sin uso de técnicas de detección de objetos, debido al alto coste de etiquetar estos *datasets* de música.

5.2 Predicción de nodos/símbolos

5.2.1 Metodología

En esta sección, definimos formalmente el problema que aborda este trabajo y describimos la solución propuesta para afrontarlo.

5.2.1.1 Formulación

Nuestro objetivo en este trabajo es extraer el conjunto de primitivas de símbolos que conforman las estructuras musicales presentes en las partituras musicales.

La definición formal del problema es la siguiente: Sea Σ el vocabulario de primitivas de símbolos musicales (por ejemplo, cabeza de nota, plica, *flags*, ...) y sea $\mathcal{P}(\Sigma)$ el conjunto de todos los posibles multisets[4] que utilizan elementos de Σ. Dada una imagen de entrada $\mathbf{x}$, buscamos el subconjunto $\hat{S} = \{s_1, s_2, \ldots, s_n\}$ tal que

$$\hat{S} = \arg\max_{S \in \mathcal{P}(\Sigma)} P(S \mid \mathbf{x}) \tag{5.3}$$

Para resolver la ecuación 5.3, reformulamos el problema como una predicción de secuencias, dado que una secuencia se puede convertir en un multiset simplemente ignorando el orden de sus elementos. Por lo tanto, esto se reescribe como:

$$\hat{S} = \varphi\left(\arg\max_{s \in \Sigma^*} P(\mathbf{s} \mid \mathbf{x})\right) \tag{5.4}$$

donde φ es una función que convierte una secuencia en su correspondiente multiset.

La ecuación 5.4 puede ser abordada por los enfoques existentes de *image-to-sequence* de visión por computador, que se ocupan de los desafíos que se formulan de igual manera. Estos consideran el uso combinado de CNN + RNN, que representan una solución apropiada para tratar tanto la naturaleza visual y secuencial del problema. A continuación se describe el planteamiento de los modelos *image-to-sequence* considerados.

5.2.1.2 Enfoques. Modelos *Image-to-sequence*

Para implementar un enfoque de *image-to-sequence*, recurrimos a los modelos de *image captioning*, que toman una imagen como entrada y emiten una secuencia que describe los elementos que contiene.

El modelo, en cada *timestep* t, pretende maximizar la probabilidad de un símbolo (o *token*) dado un conjunto de características presentes en la imagen de entrada, junto con la secuencia generada hasta t. Esta solución se expresa como $\hat{s}_t = P(s_t \mid \mathbf{x}; \mathbf{s}_{0,t-1})$, donde $\mathbf{x}$ representa la imagen de entrada, $\mathbf{s}_{0,t-1}$ los tokens predichos anteriormente, y s_t el símbolo a predecir en el *timestep* actual.

Dado que queremos que nuestro modelo produzca una secuencia condicionada a la entrada, necesitamos introducir características de la imagen en el cálculo del modelo secuencial. Esta extracción de características puede hacerse de varias maneras, siendo la más común la que

[4] Un conjunto de elementos donde estos se pueden repetir.

utiliza redes convolucionales. A continuación, se utiliza una RNN para modelar las dependencias temporales entre las características y las salidas anteriores de la red para predecir un token determinado en un *timestep* específico. En este trabajo, implementamos este enfoque por dos medios: una CNN+RNN simple y una CNN+RNN basada en la atención visual.

CNN + RNN La primera forma de implementar el modelo de *image captioning* es combinando directamente una CNN y una RNN como en Vinyals et al. (2015). La CNN, normalmente denominada *encoder*, procesa la imagen de entrada y aprende a proyectar sus características relevantes en un mapa de características. Esta estructura de datos es entonces redimensionada y pasada como estado inicial a una RNN, conocida como el *decoder*. Esta parte también recibe las predicciones anteriores de la red como entrada. A partir de estos datos, la RNN predice el token que corresponde en cada *timestep*.

Visual-attention-based CNN + RNN Como segundo enfoque, utilizamos un modelo basado en atención visual para condicionar la RNN a la información extraída por el *encoder*, como en Xu et al. (2015). Nótese que, en este caso, las características de la imagen se extraen directamente de uno de los mapas de características generados a través de la CNN. La figura 5.7 muestra un ejemplo de cómo se hace esto.

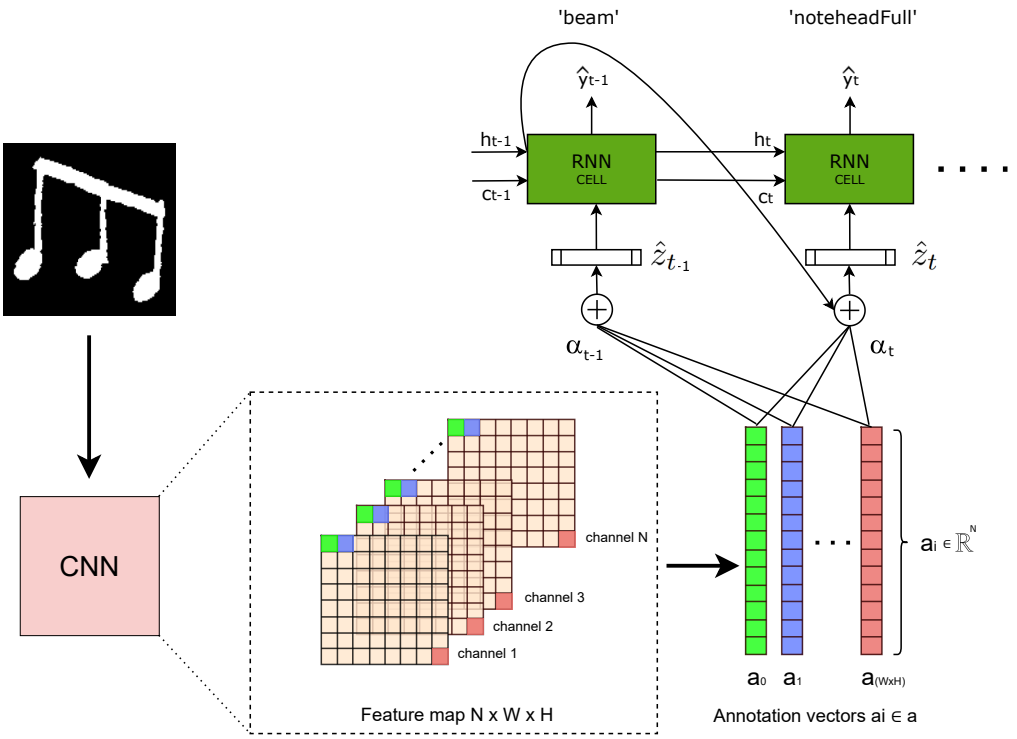


Figure 5.7: Esquema general de la metodología. El proceso de extracción de anotaciones se representa en la figura. Cada anotación a_i está compuesta por las características del píxel i a lo largo de todos los N canales extraídos del mapa de características de la CNN.

En este caso, el *decoder* recibe una entrada compuesta, junto a la salida del *encoder* como

estado inicial. Esta nueva entrada se expresa como $[\hat{y}_{t-1}, \hat{z}_t]$, donde y_{t-1} es el token previamente predicho y $\hat{z}_t$ es un vector de contexto que contiene información filtrada del *encoder* en el *timestep* actual. Este vector se produce mediante un mecanismo de atención, que filtra los mapas de características extraídos a través de una máscara de atención. Esta máscara se aprende a través de un MLP, que recibe como entrada el mapa de características concatenado junto con los *tokens* previamente predichos. En general, la red aprende a seleccionar qué información es relevante de la imagen de entrada para mejorar las predicciones del *decoder*.

Un detalle de este modelo es que su función de pérdida -la entropía cruzada- está regularizada con un término de penalización como el siguiente:

$$\mathrm{L}(\mathbf{x}, \mathbf{s}) = -\sum_{t=1}^{N} \log p_t(s_t) + \lambda \sum_{i}^{L} (1 - \sum_{t}^{c} \alpha_{ti})^2 \tag{5.5}$$

Esta pérdida puede interpretarse como una forma de "forzar" al modelo a prestar atención a cada parte de la imagen durante la inferencia de la secuencia completa.

5.2.2 Experimentos

En esta sección, definimos el entorno experimental para evaluar los modelos *image-to-sequence* en el problema propuesto.

5.2.2.1 Datos y extracción

Para estos experimentos, se extraen subgrafos del grafo total correspondiente a cada partitura en la MUSCIMA++. Estos subgrafos son lo que denominamos estructuras musicales. La figura 5.8 ilustra ejemplos de estas estructuras musicales.

En lo que respecta a los detalles de esta extracción, en la MUSCIMA++ existen dos tipos de relaciones entre nodos/símbolos: aristas de relación y aristas de precedencia. Las aristas de relación indican si dos nodos están conectados, mientras que las aristas de precedencia únicamente indican la continuidad de la música por la partitura. El último tipo de aristas sirve para conectar estas estructuras musicales. Para la extracción de estas estructuras, se ha creado un algoritmo que, dado un símbolo, se añade a los símbolos con los que se conecta (con aristas de relación) a una cola de datos, siempre que estos símbolos no estén en una lista que representa los nodos del subgrafo. Cada vez que se elimina un elemento de la cola, significa que ya se han añadido a la cola todas sus relaciones a tener en cuenta. De esta forma, se procesan todos los símbolos hasta que la cola esté vacía, ya que indica que se ha añadido todo el subgrafo. Una vez terminado, se reinician la lista del subgrafo y la cola y se pasa al siguiente símbolo, que corresponderá al inicio del siguiente subgrafo, del cual se vuelven a extraer todos sus nodos de igual forma.

De estos subgrafos obtenidos, únicamente nos quedamos con los nodos, que corresponden a las primitivas musicales en forma de multiset. Seleccionamos aquellos conjuntos que contienen más de tres primitivas, ya que suponemos que los que contienen menos elementos pueden ser reconocidos fácilmente por cualquier modelo que no sea excesivamente complejo.

Para la extracción de las imágenes correspondientes a cada subgrafo, de todos los nodos se obtienen los puntos extremos para capturar todos los nodos en la imagen, de la que se extrae el contenido con estas coordenadas.

Table 5.3: Estadísticas según el número de primitivas en las estructuras musicales seleccionadas del *dataset* MUSCIMA++. Nótese que únicamente consideramos para este problema los símbolos que aparecen en la tabla.

Primitiva	
Full notehead	14,868
Empty notehead	213
Stem	13,843
Beam	6,575
8th flag	698
16th flag	487
Total	36,684

Finalmente, lo que se obtiene es un *dataset* que contiene pares imágenes-símbolos, de igual forma que en los *corpus* de *image captioning*. Cabe destacar que toda esta información acerca de los nodos se ha extraído utilizando la librería MuNG.

La tabla 5.5 muestra del número de primitivas seleccionadas tras la extracción y selección de datos, mientras que la figura 5.9 representa la distribución de los multisets según su tamaño.

Figure 5.8: Ejemplos de diferentes estructuras musicales seleccionadas para nuestros experimentos.

5.2.2.2 Detalles de implementación

En este trabajo, implementamos tanto un modelo de *image captioning* "clásico" como uno con atención visual, descritos en la sección 5.2.1. En cuanto a la arquitectura, ambos modelos constan de las mismas partes conceptuales, un *encoder* y un *decoder*. Para el *encoder* (CNN), ambos comparten la misma arquitectura con pequeñas diferencias: cuatro capas de convoluciones y *poolings* alternos, con 64, 128 y 256 filtros respectivamente. En el caso del modelo

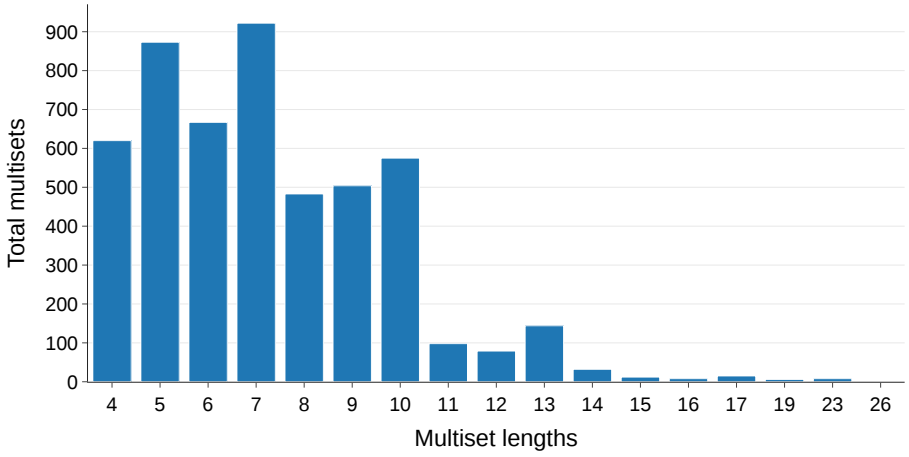


Figure 5.9: Distribución del tamaño de los multisets tras la selección de datos.

de atención visual, esta última capa produce 512 filtros. Los *poolings* son de 2x2, excepto en la última capa del modelo simple, que se aumenta a 4x4 para reducir las características eventuales antes de la RNN. Para remodelar la salida del *encoder* con el fin de alimentar un estado inicial al *decoder*, implementamos una capa de aplanamiento en el modelo "clásico", mientras que en el de atención visual se extraen las características como se describe en la Fig. 5.7. Pasamos ahora a la parte del *decoder*, que es donde se encuentran las principales diferencias.

Para el *decoder* (RNN), hacemos uso de unidades LSTM. Las diferencias más importantes radican en las dimensiones de los estados ocultos y en la entrada dada para cada *timestep*. En el caso del modelo simple, el estado oculto consta de 1024 dimensiones. La entrada en cada *timestep* es el token predicho en el anterior.[5]. Este modelo recibe las características aplanadas de la imagen codificada con un tamaño de 1024 dimensiones y se introduce como estado oculto inicial

En el modelo de atención visual, añadimos el mecanismo de atención descrito en 5.2.1.2. Este modelo recibe como entrada la concatenación del token de entrada y el vector de contexto $\hat{z}_t$. Las características del *encoder* en este caso son las anotaciones extraídas de 512 mapas de características de 8x8 píxeles, que dan un total de 64 vectores con 512 dimensiones. Las anotaciones se definen como a.

Finalmente, ambos modelos tienen una capa lineal con tantas unidades como el tamaño del vocabulario, y una operación softmax para tomar el token más probable en cada *timestep* para la predicción. Ambos modelos se entrenan con el optimizador Adam con una tasa de aprendizaje de 0.001.

5.2.2.3 Métricas

Dado que nuestro objetivo es medir la cantidad de símbolos detectados en un multiset, la principal métrica utilizada fue el *Intersection-Over-Union* (IoU). Dados dos conjuntos, A y

[5]Durante el entrenamiento, utilizamos una metodología de *teacher forcing* en ambos modelos

B, calculamos el IoU como en la ecuación (5.6).

$$\text{IoU}(A, B) = \frac{| A \cap B |}{| A \cup B |} = \frac{| A \cap B |}{| A | + | B | - | A \cap B |} \tag{5.6}$$

Obsérvese que, por definición, $0 \leqslant \text{IoU}(A, B) \leqslant 1$. Como los multiset pueden tener símbolos repetidos, primero enumeramos estos como

$$S_1 = \{a, b, c, a\}, S_2 = \{a, b, a, c\} \rightarrow S_1' = \{a_1, b_1, c_1, a_2\}, S_2' = \{a_1, b_1, a_1, c_2\}$$

y luego se calcula el IoU entre S_1', S_2'.

5.2.2.4 Escenarios de evaluación

Además de los dos enfoques para *image captioning*, incluimos algunos factores a considerar en la evaluación. En concreto, estos son:

1. **Escasez de datos**. Estudiar el rendimiento en condiciones de limitación de datos nos permite comprender los posibles límites de nuestra formulación. Esto podría ser de especial relevancia en el caso de las partituras manuscritas, las cuales son difíciles de anotar. Para simular diferentes condiciones, restringiremos los datos utilizados para el entrenamiento a porcentajes variables (1, 5, 10, 25, 50, 75 y 100).

2. **Orden en las secuencias**. Recordemos que estamos implementando modelos *image-to-sequence*, pero los datos reales vienen en forma de multiset. Por lo tanto, debemos definir un orden consistente de las primitivas de los extractos para entrenar los modelos neuronales secuenciales. En este caso, consideramos los siguientes opciones. (i) Ordenar las primitivas según la topología de la estructura de la anotación musical en la imagen; en este caso, consideramos las primitivas ordenadas de izquierda a derecha y de arriba a abajo. A partir de ahora, denominaremos a esta política "no-sort". (ii) Ordenar las primitivas alfabéticamente según el vocabulario considerado. De este modo, las primitivas de la misma categoría se colocan consecutivamente en la secuencia. Esta opción se denominará "sort".

3. **Aumentado de datos**. El uso de técnicas básicas de aumentado de datos -rotaciones de hasta 45 grados, *paddings* y flips verticales y horizontales- se incluye para analizar la respuesta de estos modelos y su relación con el rendimiento. Nótese que el hecho de evitar la localización precisa de las primitivas facilita la posibilidad de utilizar estas técnicas de aumentado de datos de forma directa (sin necesidad de modificar la anotación del *ground truth*).

La combinación de todos estos factores, junto con los dos enfoques neuronales, se estudiará empíricamente.

5.2.3 Resultados

En esta sección, presentamos y discutimos los resultados obtenidos tras realizar un experimento de validación cruzada de cinco iteraciones (5-CV). La tabla 5.7 muestra el rendimiento

Table 5.4: Resultados promedios de IoU en 5-CV con respecto al enfoque (el uso o no de mecanismos de atención, aumentado de datos, o el tipo de ordenación de la secuencia) y el porcentaje de los datos de entrenamiento utilizados. Los resultados en negrita indican el mejor IoU por % de los datos de entrenamiento.

Enfoque			% de datos						
Modelo	Aum. datos	Orden	1	5	10	25	50	75	100
CNN + RNN	No	No	68.9	67.8	80.4	89.1	92.5	94.1	95.0
	No	Yes	66.1	68.2	71.0	80.9	92.6	88.9	95.8
	Yes	No	66.6	69.7	80.9	89.4	93.0	94.3	95.0
	Yes	Yes	66.5	68.2	70.1	85.8	87.8	94.6	96.0
Visual-attention CNN + RNN	No	No	**72.4**	78.8	82.2	88.0	92.1	93.5	94.2
	No	Yes	68.6	79.7	**86.2**	90.3	**94.4**	95.2	96.2
	Yes	No	67.0	79.5	80.8	88.7	92.0	93.6	94.6
	Yes	Yes	62.3	**80.8**	85.7	**90.6**	93.7	**95.4**	**96.4**

medio en términos de IoU de cada configuración del modelo. Para observar las tendencias generales, también proporcionamos la Fig. 5.10.

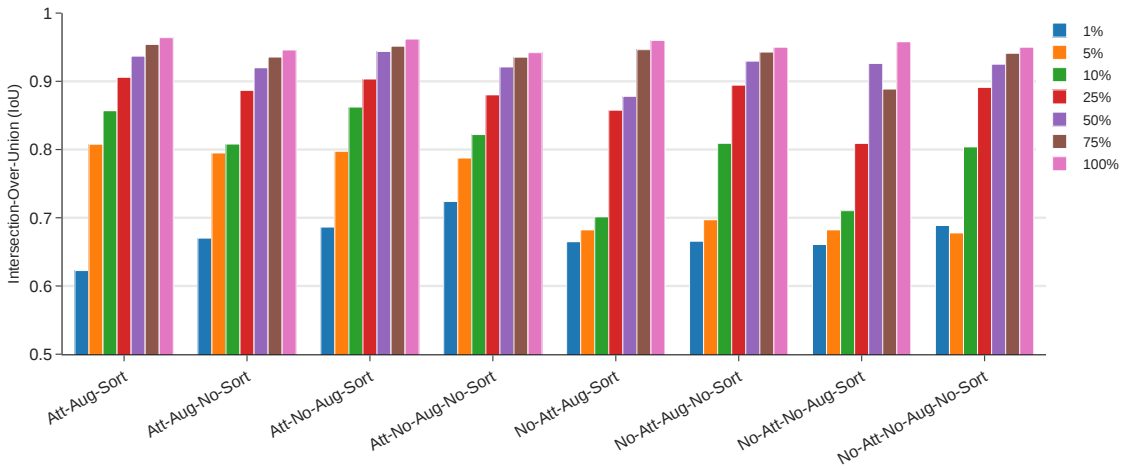


Figure 5.10: Resultados de cada modelo medidos en términos de la métrica *Intersection-Over-Union* (IoU). La gráfica muestra los resultados medios obtenidos en un 5-CV. Cada barra representa el rendimiento según el porcentaje de datos utilizados para el entrenamiento, mientras que el eje x indica la configuración específica.

En primer lugar, observamos que todos los enfoques considerados pueden considerarse exitosos, lo que indica que la tarea es factible con las formulaciones existentes. La cantidad

de datos tiene una relación directa con el rendimiento, pero solo se degrada notablemente cuando el porcentaje es muy pequeño (menos del 10%). Al inspeccionar los resultados con todos los datos disponibles, observamos que el rendimiento está siempre en torno al 95 de IoU. Para ayudar a entender el significado de estos valores, la Fig. 5.11 muestra un ejemplo de reconocimiento perfecto y otro en el que se produce un error, utilizando el mejor modelo posible (identificado más adelante). Se puede observar que, a pesar del gran número de primitivas a recuperar de la imagen, algunas de ellas (como los *beams*) con cierta complejidad gráfica, el enfoque es capaz de operar sin necesidad de especificar información espacial durante el entrenamiento.

En cuanto a los dos modelos neuronales, no se observan diferencias significativas cuando se utiliza el conjunto de datos completo. Sin embargo, el modelo que incluye la atención consigue resultados ligeramente mejores, cuya diferencia varía en función de la configuración específica. Por ejemplo, mientras que el modelo sin atención obtiene un IoU de 96 (con aumentado de datos y secuencias ordenadas alfabéticamente), el modelo con atención llega a 96,4. La diferencia entre los dos enfoques es más notable con los porcentajes más bajos de datos. Por ejemplo, con solo un 5% de datos, el modelo sin atención alcanza únicamente el 69,7 de IoU, mientras que el modelo con atención consigue llegar al 80,8 de IoU.

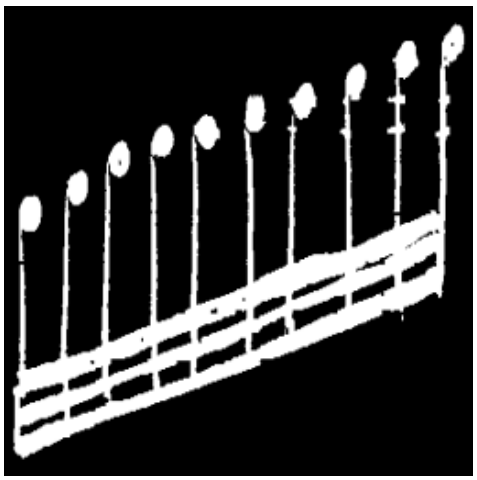
Además, los otros dos factores estudiados (el aumentado de datos y la forma de ordenar las secuencias) no parecen afectar significativamente al rendimiento. El aumentado de datos no mejora notablemente los resultados, ni siquiera en condiciones de escasez de datos. De hecho, a veces es perjudicial, como ocurre en el modelo con atención con solo 1 % de datos (de 72,4 a 67,0 y de 68,6 a 62,3 de IoU, sin y con ordenación de secuencias, respectivamente). Esto puede deberse a que el aumentado genérico de datos introduce algunas incoherencias que dificultan el proceso de aprendizaje. En cuanto a la forma de ordenar las secuencias, tampoco observamos cambios significativos, y su adecuación depende del resto de la configuración. En general, sin embargo, se observa una regularidad: el modelo sin atención funciona mejor con la ordenación topológica, mientras que el modelo con atención muestra mejores resultados cuando las primitivas aparecen alfabéticamente.

Con todo lo anterior, podemos afirmar que, en general, los modelos considerados son capaces de abordar el problema en cuestión. Si bien es cierto que hay otros factores a tener en cuenta, no parecen decisivos para el funcionamiento de los modelos. La excepción es la cantidad de datos, ya que se observa una clara correlación entre el tamaño del conjunto de entrenamiento y el rendimiento de los modelos (véase la Fig. 5.10).

5.2.4 Conclusión

En este trabajo, se ha estudiado la extracción directa de primitivas musicales a partir de diferentes estructuras musicales presentes en el conjunto de datos MUSCIMA++ sin utilizar ninguna técnica de detección de objetos. Abordamos esta tarea de recuperación como un problema de conversión de imágenes a multisets con modelos *image-to-sequence*, ya que la conversión de una secuencia a un multiset se considera trivial.

Se proponen varios escenarios experimentales para estudiar diferentes factores clave, como el uso de los mecanismos de atención, la disponibilidad de datos o la ordenación de las primitivas en la secuencia. Los resultados obtenidos demuestran que, efectivamente, es posible realizar la tarea con un gran rendimiento, independientemente del modelo específico e incluso en escenarios con grandes limitaciones de datos.



Muestra de test donde todas las primitivas se recuperan correctamente.

Muestra de test donde se pierde una primitiva.

Figure 5.11: Ejemplos de diferentes predicciones obtenidas con el mejor modelo. (a) Este ejemplo corresponde a una imagen con 23 símbolos: 10 cabezas de nota completas, 10 plicas y 3 haces. La ácrata obtenida es 100. (b) Este ejemplo contiene 16 símbolos: 6 cabezas de nota completas, 6 plicas y 4 *beams*. Al modelo le falta un *beam*. La puntuación obtenida es de 94%.

La tarea de conversión de imágenes en multisets es solo un paso inicial del problema de conversión de imágenes en grafos en OMR. Por lo tanto, en la siguiente sección avanzamos en nuestra investigación para recuperar el grafo completo (primitivas y relaciones) de manera integral.

5.3 Recuperación completa de grafos

5.3.1 Metodología

En esta sección, describimos el problema de la recuperación holística de grafos a partir de imágenes y la metodología adoptada para abordar esta tarea en el contexto de OMR.

5.3.1.1 Formulación

Como hemos definido a lo largo de este trabajo, un grafo es una estructura matemática abstracta que representa las relaciones por pares entre elementos -nodos o vértices- a través de conexiones -aristas-. El objetivo de este trabajo específico es recuperar directamente los grafos de las imágenes que contienen estructuras musicales. La definición formal del problema es la siguiente.

Un grafo puede definirse como un par (V, E) en el que V representa el conjunto de nodos y E el conjunto de aristas. Dos nodos, $v_i, v_j \in V$ están conectados si existe una arista, $e_{ij} = (v_i, v_j) \in E$. Usemos $\mathcal{G}$ para denotar el espacio de todos los grafos posibles. Dada una imagen $\mathbf{x}$, buscamos recuperar el grafo más probable $\hat{g}$:

$$\hat{g} = \arg\max_{g \in \mathcal{G}} P(g \mid \mathbf{x}). \tag{5.7}$$

Un grafo puede, en términos computacionales, ser modelado mediante un conjunto de representaciones, cada una de las cuales se identifica por un orden específico de sus nodos. En tal caso, los nodos se denotan por una secuencia en lugar de un conjunto. Sea $\mathcal{R}(g)$ el conjunto de todas las representaciones de un grafo $g \in \mathcal{G}$. Por lo tanto, la ecuación 5.7 puede reescribirse como:

$$\hat{g} = \arg\max_{g \in \mathcal{G}} P(g \mid \mathbf{x}) = \arg\max_{g \in \mathcal{G}} \sum_{r \in \mathcal{R}(g)} P(r \mid \mathbf{x}) \tag{5.8}$$

Calcular la ecuación 5.8 en la práctica resulta inviable, incluso para los grafos pequeños, dado que estas posibles representaciones crecen de forma factorial respecto al número de nodos $|V|!$. En su lugar, aproximamos la solución buscando el grafo que indica la representación más probable $\hat{r}$:

$$\hat{g} = \arg\max_{g \in \mathcal{G}} P(g \mid \mathbf{x}) \approx \arg\max_{g \in \mathcal{G}} \max_{r \in \mathcal{R}(g)} P(r \mid \mathbf{x}) \tag{5.9}$$

El objetivo de nuestro planteamiento es resolver la Ec. 5.9 mediante redes neuronales.

5.3.1.2 Enfoque

La figura 5.12 proporciona una visión general de la red neuronal propuesta para abordar la tarea en cuestión.

Para recuperar una representación del grafo, dividimos conceptualmente el problema en dos tareas: predicción de nodos y la de aristas. En la primera tarea, se recuperan los nodos de una representación específica de un grafo mediante una predicción secuencial de categorías de nodos, véase la sección 5.2.

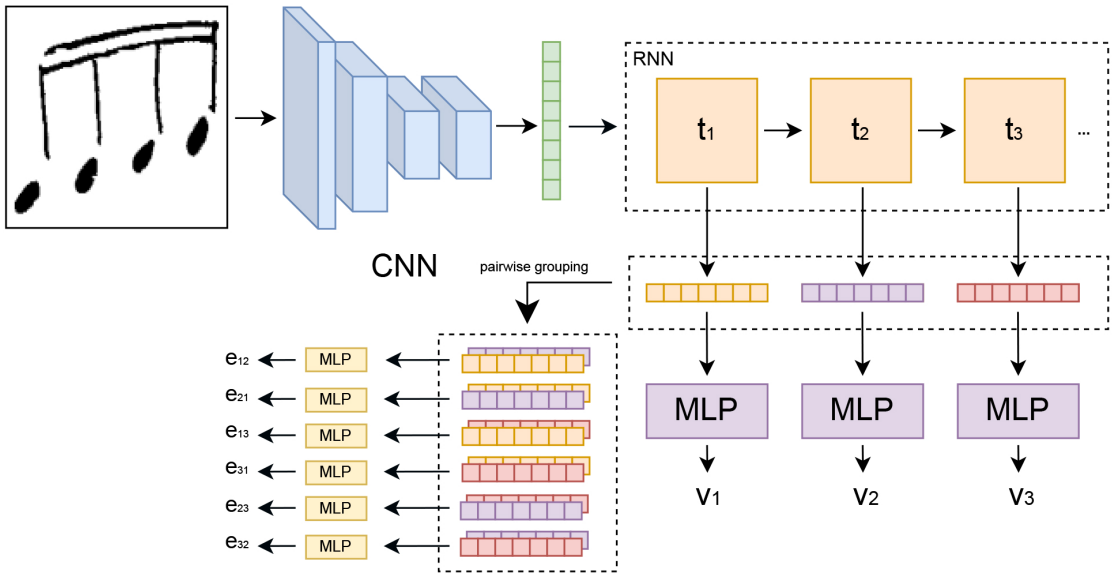


Figure 5.12: Esquema general de la metodología. Una CNN extrae la representación de la imagen, que es la entrada al *decoder* RNN. Esta RNN se entrena para producir cada símbolo en cada *timestep* t junto con la representación de los nodos. Finalmente, predecimos las aristas utilizando estas representaciones de nodos por pares.

En la segunda tarea, la red predice si un par de nodos están conectados, lo que puede verse como una predicción de la matriz de adyacencia de esa representación del grafo, de forma similar a 5.1.

En este trabajo, proponemos una arquitectura de múltiples salidas que realiza las dos tareas simultáneamente. Con respecto al objetivo de recuperar una secuencia de símbolos como nodos de una imagen, la primera etapa implica la implementación de un modelo neuronal *image-to-sequence* similar a los enfoques de *image-captioning*, en el que el objetivo es describir una imagen mediante lenguaje natural. Para ello se utiliza una arquitectura *encoder-decoder* que recupera las características de la imagen empleando una red neuronal convolucional (CNN). Estas imágenes son alimentadas a una RNN, que decodifica una primitiva y su representación para cada *timestep* hasta que se alcanza un token de fin de grafo—$< EOG >$. Los *embeddings* generados en cada *timestep* se utilizan también para la clasificación de los nodos y, al agruparlos por pares, el modelo predice si existe una arista entre ellos. El principal reto al que se enfrenta la red es el de obtener *embeddings* adecuados que representen tanto las primitivas musicales como las características presentes en la imagen, ya que son elementos clave tanto para la predicción de nodos como de aristas. Por lo tanto, con la arquitectura propuesta estamos forzando a la red a capturar la riqueza de las estructuras musicales de una manera holística.

5.3.1.3 Entrenamiento de la red

En nuestro caso, requerimos que la red neuronal se entrene sin ninguna información geométrica específica relativa a la ubicación de los vértices en la imagen de entrada. Obsérvese

que esto podría representar una ventaja competitiva a la hora de crear *datasets*, ya que es mucho menos costoso anotar las partituras de esta manera y el proceso también podría automatizarse a partir de partituras anotadas existentes.

Supongamos que nuestro conjunto de entrenamiento está formado por pares $(\mathbf{x}, g)$. Como ocurre en la Ec. 5.9, no será posible entrenar directamente la red para optimizar g dado $\mathbf{x}$ y en su lugar es necesario elegir una representación específica de $\mathcal{R}(g)$ como *ground truth* para la red neuronal. Sin embargo, esta decisión no debe ser tomada arbitrariamente en cada caso, y es, más bien, necesaria para asegurar cierta consistencia con el fin de facilitar el proceso de aprendizaje de la red. Denotamos como Φ una función que mapea consistentemente un grafo g en una de sus representaciones de la forma $r = \Phi(\mathbf{x}, g)$. Tenga en cuenta que Φ también puede ser condicionada a la imagen de entrada x. Este mapeo transforma un conjunto de nodos $\mathbf{v} = \{v_1, v_2, v_3, \dots\}$ en una secuencia ordenada $\mathbf{v}' = [v_1, v_2, v_3, \dots]$. La forma en que opera la función Φ se describirá en los detalles de la implementación y se analizará empíricamente.

La transformación de la representación del grafo en una secuencia de nodos permite tratar la primera tarea como un problema de imagen a secuencia. Sea Σ el conjunto de posibles primitivas de notación musical. Entonces es necesario buscar la secuencia de nodos $\hat{\mathbf{v}}$:

$$\hat{\mathbf{v}} = \arg\max_{\mathbf{v} \in \Sigma^*} \mathrm{P}(\mathbf{v} \mid \mathbf{x}) \tag{5.10}$$

en el que el objetivo es predecir la secuencia de tokens $\mathbf{v} = [v_1, v_2, \dots v_t]$. Esto puede ser entrenado por la minimización de la suma de la probabilidad logarítmica negativa de la probabilidad del símbolo correcto v_t para cada instante temporal t en la secuencia v:

$$\mathcal{L}_{\text{nodes}} = -\sum_{t=1}^{N} \log \mathrm{P}(\hat{v}_t = v_t | t) \tag{5.11}$$

Con respecto a las aristas, esta tarea de predicción se aborda como un problema de clasificación binaria en el que la pérdida de *Binary Cross-Entropy (BCE)* para cada par de nodos en el grafo se minimiza como

$$\mathcal{L}_{\text{BCE}} = \sum_{e_{ij} \in A} e_{ij} \log(\hat{e}_{ij}) + (1 - e_{ij}) \log(1 - \hat{e}_{ij}) \tag{5.12}$$

donde e_{ij} es la arista que empareja los nodos (v_i, v_j). Esto corresponde a una celda específica en la matriz de adyacencia A del grafo.

La suma de ambas pérdidas se expresa como la minimización de la siguiente función de pérdidas múltiples $\mathcal{L}_{total}$, dada por:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{nodes}} + \mathcal{L}_{\text{BCE}}. \tag{5.13}$$

Esta multipérdida, al ser derivable, se optimiza conjuntamente en tiempo de entrenamiento mediante *backpropagation* convencional.

5.3.1.4 Predicción

El proceso de predicción de este método sigue de cerca el flujo descrito durante el entrenamiento. En primer lugar, una imagen $\mathbf{x}$ se introduce en la CNN, que produce un mapa de

características. A continuación, el bloque recurrente recibe este mapa de características para condicionar sus estados internos y emite un *embedding* para cada *timestep* t. Posteriormente, se obtiene una secuencia de vértices $\mathbf{v}$ y sus correspondientes aristas.

Para calcular $\mathbf{v}$, pasamos el *embedding* de cada *timestep* a través de un MLP que realiza una clasificación. La clase que representa cada nodo v_i se determina siguiendo una estrategia voraz, donde se elige la primitiva con mayor valor de activación en la capa de salida. El conjunto de aristas se obtiene mediante la combinación de los *embeddings* recuperados por la RNN para predecir la matriz de adyacencia del grafo en el módulo de predicción de aristas. Para cada par, el módulo predice la probabilidad de que exista una arista entre ellos.

Una vez recuperados los nodos y las aristas, hemos reconstruido implícitamente la representación más probable de un grafo dada la imagen de entrada, que satisface la Ec. 5.9.

5.3.2 Experimentos

En esta sección, presentamos los experimentos realizados para evaluar la tarea de convertir una imagen a un grafo en el contexto de OMR. Describimos la obtención de datos y las métricas utilizadas, junto con los detalles de implementación de los modelos presentados. Finalmente, defnimos los escenarios de evaluación diseñados para este trabajo.

5.3.2.1 Datos y extracción

Para nuestra tarea, las estructuras musicales presentes en cada partitura se extrajeron de igual forma que en el trabajo anterior. Sin embargo, en este caso nos quedamos con el subgrafo completo que representa toda la estructura musical, añadiendo tanto los nodos como las relaciones entre ellos. En concreto, solo consideramos los ejemplos en los que el número de primitivas era estrictamente superior a tres, al igual que en el trabajo anterior. Finalmente, se obtiene un conjunto de datos de pares de imágenes-grafos anotados independientes que contienen los nodos - primitivas musicales-notación - y las relaciones entre ellos como una matriz de adyacencia. La figura 5.13 muestra algunos ejemplos del conjunto de datos finalmente obtenido. Para más detalles, la Tabla 5.5, 5.6 y la Fig. 5.14 presentan algunas estadísticas relativas a los símbolos y a los grafos considerados.[6]

5.3.2.2 Detalles de implementación

Implementamos nuestro enfoque neuronal con dos redes neuronales específicas, entre las cuales hay pocas diferencias. Ambas implementaciones constan de un *encoder* que extrae las características de la imagen y un *decoder* que recupera la secuencia de símbolos y construye la matriz de adyacencia. Tomando como base la técnica empleada para la parte de conexión *encoder-decoder*, denotamos como (i) el modelo de no atención y (ii) el modelo basado en la atención visual. La diferencia más importante se refiere a dichac conexión entre *encoder* y *decoder*. El primer modelo extrae las características de la imagen con cuatro capas de convolución y *poolings* alternados con 64, 128, 256 y 256 filtros, mientras que el segundo emite 512 filtros en la última capa. Los *poolings* son 2×2 en ambos modelos para cada capa,

[6] Nótese que el grado medio (por grafo) representa el grado medio de los nodos, teniendo en cuenta el tamaño del grafo a diferencia del grado medio.

Table 5.5: Estadísticas acerca de las primitivas de la notación musical (Σ) en las estructuras seleccionadas del *dataset* MUSCIMA++.

Primitivas	
Full notehead	14,868
Empty notehead	213
Stem	13,843
Beam	6,575
8th flag	698
16th flag	487
Total	36,684

Table 5.6: Estadísticas de los datos obtenidos una vez filtrada la MUSCIMA++. Avg. indica la media $\pm$ desviación típica.

Grafos	
N.º grafos	5,047
N.º nodos	36,684
N.º aristas	70,558
Media nodos	7.3 ± 2.6
Media grado	1.9 ± 1.0
Media grado (por grafo)	1.8 ± 0.3

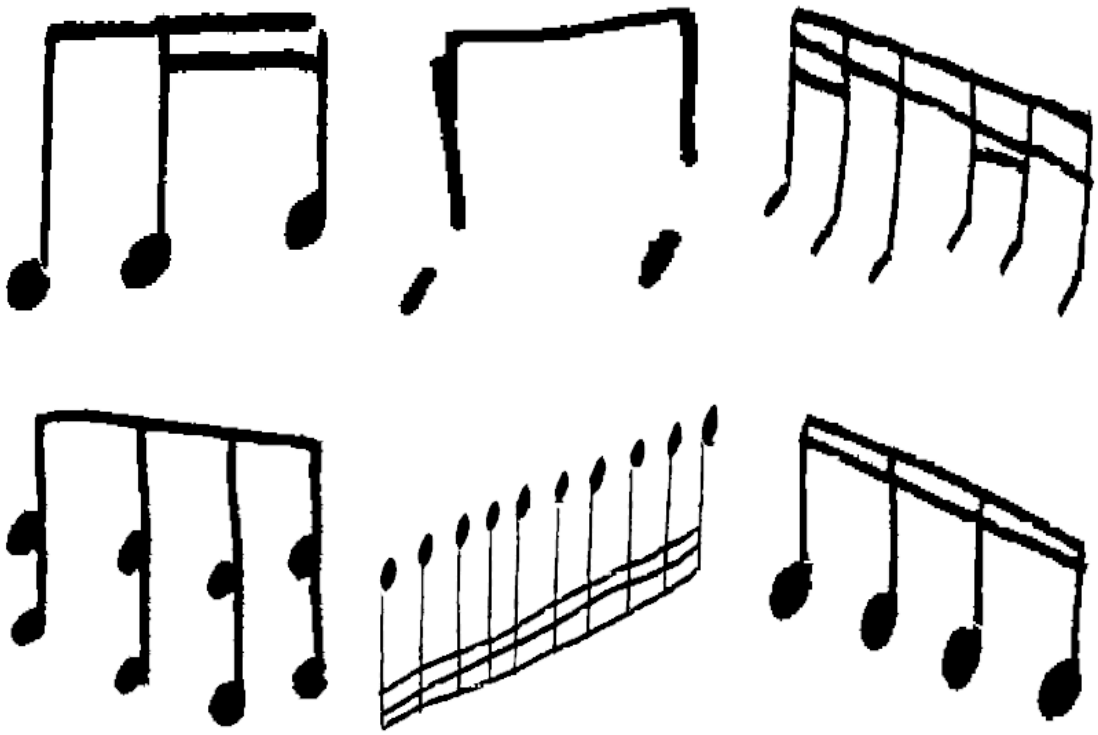


Figure 5.13: Estructuras musicales extraídas de la MUSCIMA++.

con la excepción de la última capa del primer modelo, que es 4x4 para reducir la dimensionalidad del vector de características. Todas estas capas utilizan la función de activación *Rectified Linear Unit* (ReLU).

Otra diferencia entre los modelos tiene que ver con la forma en que se remodela el mapa de características obtenido de la CNN para tratarse como una secuencia. Para el modelo sin atención visual, el mapa de características se aplana para calcular un vector representativo de 1024 dimensiones. El modelo con atención visual utiliza, en cambio, el mecanismo de atención visual, como en Xu et al. (2015). Este mecanismo se aplica al mapa de características, obteniendo 64 vectores con 512 características cada uno que luego se introducen en el *decoder* como estado inicial.

En cuanto al *decoder*, ambos modelos utilizan una capa LSTM en la parte recurrente para extraer cada símbolo para cada paso de tiempo, con 1024 unidades ocultas en (i) y 512 en (ii). En la última parte del decodificador, ambos modelos tienen una capa lineal con tantas unidades como el tamaño del vocabulario Σ y una activación softmax para la clasificación de los nodos. Para la reconstrucción de la matriz de adyacencia, concatenamos los estados ocultos por pares correspondientes a los embeddings de nodos generados por la LSTM. Para esta tarea de clasificación binaria, utilizamos un MLP de dos capas con 256 unidades en la capa oculta más una función de activación sigmoidea para predecir la probabilidad de un enlace entre dos nodos a partir de sus *embeddings*. En el momento de la predicción, establecemos un umbral de decisión de $0,5$ para determinar la existencia de un borde.

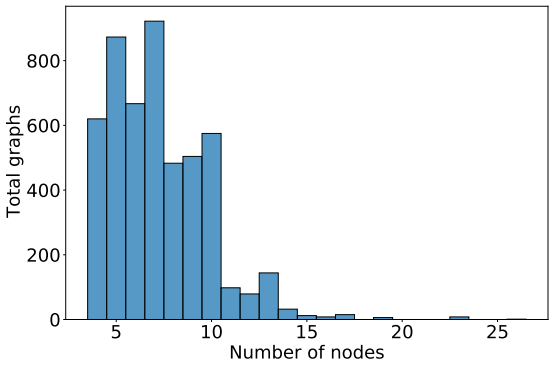


Figure 5.14: Distribución del tamaño de los grafos en el *dataset* considerado.

También aplicamos a las imágenes de entrada técnicas básicas de aumentado de datos, de la misma forma que en 5.2.

En cuanto al proceso de aprendizaje, entrenamos ambos modelos durante un máximo de 200 épocas. Se utiliza un tamaño de *batch* de 64 y se aplica un optimizador Adam Kingma & Ba (2015) con una tasa de aprendizaje de 0.001. La función de pérdida utilizada en estos modelos se define en la Ec. 5.13. También se incluye un término de atención doblemente estocástico de Xu et al. (2015) para el modelo con atención.

Por último, utilizamos dos enfoques para la ordenación de los nodos dada por la función Φ: (i) ordenar los nodos según la topología de la imagen, de izquierda a derecha y de arriba a abajo (este enfoque se denota, en adelante, como "topology"), y (ii) ordenar los nodos alfabéticamente, según el vocabulario Σ, que denotamos como "alphabetical".

5.3.2.3 Métricas

Calcular la distancia de edición de un grafo es un problema NP-completo Vento (2015), y, por lo tanto, se requieren métricas alternativas que correlacionen con el rendimiento de los modelos propuestos en esta tarea. En este caso, utilizamos dos valores para evaluar la tarea de predicción de grafos que nos ocupa: (i) la precisión de las secuencias de nodos predichas y (ii) cómo de precisas son las aristas entre estos nodos. Para el primero, consideramos el *Symbol Accuracy* (Acc), mientras que para el segundo usamos el F_1.

Symbol Accuracy. La precisión de los símbolos es una métrica basada en *Symbol Error Rate* (SER). Se trata de una métrica habitual en tareas relacionadas con el procesamiento de secuencias, como en *Handwritten Text Recognition* (Reconocimiento de Texto Escrito a Mano).[7] Este valor mide el error del modelo en la tarea de reconocimiento y se correlaciona con el esfuerzo que debería realizar un usuario para corregir manualmente los resultados. Sea H la secuencia predicha y R la secuencia de referencia. El SER se calcula midiendo la

[7] En este campo, la métrica se denomina *Word Error Rate* o *Character Error Rate*, dependiendo de la unidad considerada. Aquí empleamos el término genérico "símbolo", ya que no existe una analogía clara entre las primitivas de notación musical y las palabras/caracteres del texto.

distancia de edición entre H y R normalizada por la longitud de R. Esta métrica se convierte a la precisión de los símbolos (Acc) calculando 1 − SER.

Métrica $\mathbf{F_1}$. La métrica F_1 se define como la media armónica de la precisión (P) y el *recall* (R):

$$\mathrm{P} = \frac{TP}{TP + FP} \quad \mathrm{R} = \frac{TP}{TP + FN} \tag{5.14}$$

$$\mathrm{F_1} = 2 \cdot \frac{\mathrm{P} \cdot \mathrm{R}}{\mathrm{P} + \mathrm{R}} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \tag{5.15}$$

donde TP, FP y FN son los verdaderos positivos, los falsos positivos y los falsos negativos, respectivamente. Como deseamos medir la precisión en las aristas predichas, utilizamos las conexiones de aristas reales como la clase positiva para calcular el F_1.

5.3.2.4 Escenarios de evaluación

En este trabajo hemos considerado los siguientes escenarios de evaluación:

1. **Selección de la función** Φ. Como se ha explicado anteriormente, debemos considerar una función Φ que convierta consistentemente un grafo en una representación específica donde sus nodos se expresen como una secuencia. Analizaremos cómo la elección de Φ (topológica o alfabética) afecta al rendimiento de los modelos.

2. **Atención visual**. Las principales diferencias entre los dos modelos neuronales implementados se refieren al mecanismo de atención visual. En este trabajo se estudia el impacto del uso de este mecanismo en el rendimiento final, las diferencias entre estos mecanismos se explican en la sección 5.2.

3. **Escasez de datos**. Estudiamos el comportamiento de esta propuesta de formulación en condiciones de limitación de datos. Como ya hemos mencionado anteriormente, esto es esencial para las partituras manuscritas, donde no es fácil obtener datos etiquetados. Para simular diferentes condiciones, restringimos los datos utilizados para el entrenamiento a los siguientes porcentajes: 5%, 25%, 50% y 100%.

5.3.3 Resultados

A continuación, presentamos los resultados obtenidos para los diferentes escenarios de evaluación descritos en el apartado anterior. La tabla 5.7 y la Fig. 5.15 muestran los resultados medios en el conjunto de test, siguiendo una estrategia de validación cruzada de cinco iteraciones.

En primer lugar, hay que destacar que los enfoques propuestos cumplen con éxito sus tareas, obteniendo resultados satisfactorios en casi cualquier escenario propuesto. Obsérvese también que la precisión de los símbolos y la puntuación F_1 están estrechamente relacionadas. La correlación de estas métricas, calculada a partir de los resultados obtenidos, es concretamente de 0.998, como se muestra en la Fig. 5.16. Esta alta correlación denota que, aunque estas tareas se miden de forma independiente, su rendimiento está relacionado. Es decir, si el

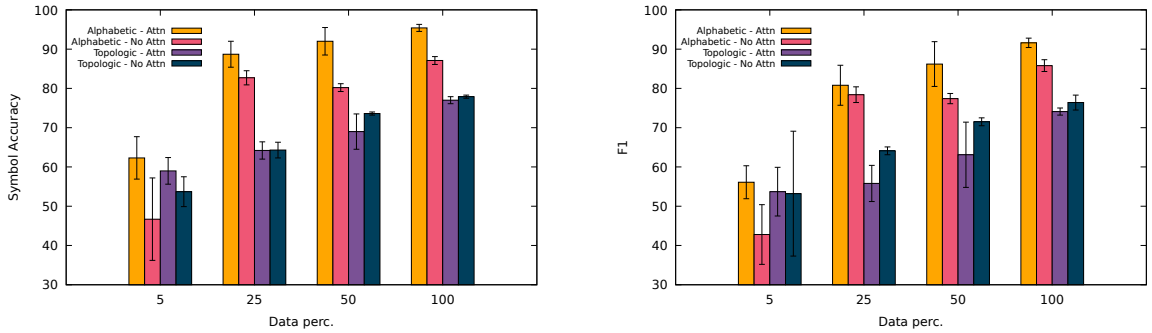


Figure 5.15: Visualización de los resultados presentados en la tabla 5.7. Se muestra la media ± std. por cada resultado de 5-CV.

Table 5.7: Resultados medios en el conjunto de test para los diferentes escenarios de evaluación en un 5-CV. Se destacan los mejores resultados para cada porcentaje según la disponibilidad de datos.

Model		% of training data							
Φ choice	Attention	5		25		50		100	
		Acc	F_1	Acc	F_1	Acc	F_1	Acc	F_1
Topology	✓	59.0	53.7	64.2	55.8	69.0	63.1	77.0	74.1
	✗	53.7	53.2	64.3	64.1	73.6	71.5	77.9	76.4
Alphabetical	✓	**62.3**	**56.1**	**88.7**	**80.8**	**92.0**	**86.2**	**95.4**	**91.6**
	✗	46.7	42.8	82.7	78.4	80.2	77.4	87.1	85.8

modelo es capaz de clasificar correctamente los símbolos, sería capaz de reconstruir las aristas del grafo con alta precisión. Este hecho demuestra que la función de pérdida propuesta en este trabajo -que une la clasificación de los nodos y la reconstrucción de las aristas- permite que la red aprenda a clasificar los nodos y a reconstruir sus aristas por igual, sin ningún sesgo.

Con respecto a la elección de la función Φ, hay diferencias significativas en el rendimiento. La elección de la Φ que ordena los símbolos alfabéticamente aumenta específicamente tanto la precisión de los símbolos como el F_1 en aproximadamente un 20%. La razón de esta diferencia viene dada por la sensibilidad a las variaciones de imagen del orden topológico y su impacto durante el entrenamiento. Cuando se presentan dos imágenes similares que tienen elementos desplazados, es probable que sus representaciones en forma de grafo correspondiente sean topológicamente diferentes, ya que los nodos aparecen en un orden diferente y esto altera la matriz de adyacencia. Esto es problemático en el entrenamiento de redes neuronales, ya que entradas similares que tienen salidas completamente diferentes -un hecho a la que la

literatura se refiere como ruido- conducen a problemas de convergencia durante el proceso de optimización y disminuyen la robustez del modelo. En nuestro caso, hay primitivas musicales con este problema, ya que las formas tienden a ser similares con ligeras variaciones. Al utilizar la ordenación topológica con la función Φ, es probable que se introduzca este tipo de fenómenos, lo que disminuye el rendimiento de la red. Además, es probable que este caso se agrave cuando se aplica el aumentado de datos. El modelo mejora significativamente cuando se utiliza una función Φ que ordena los nodos del grafo independientemente de la disposición de la imagen. Esto se debe probablemente a que muestras similares obtengan una representación gráfica parecida, lo que facilita la convergencia de la red neuronal durante el entrenamiento.

En cuanto al uso del mecanismo de atención visual, no hubo diferencias significativas en la mayoría de los escenarios. Sin embargo, en los escenarios donde la escasez de datos es más prominente, encontramos que los modelos que utilizan el mecanismo de atención visual superan claramente a los modelos que no lo utilizan. Esto se debe a que los mecanismos de atención son capaces de obtener información filtrada de representaciones específicas de la imagen, lo que permite que el modelo converja con menos muestras de entrenamiento. A la vista de los resultados, la red que utiliza este mecanismo aprovecha mejor las muestras de entrenamiento.

En relación con la escasez de datos, el rendimiento también está vinculado a la elección de la función Φ. Con la función de ordenación adecuada, el modelo se comporta bien incluso cuando la escasez de datos es extrema, como en el caso de solo el 5% de los datos disponibles. Como muestra la Tabla 5.7, los modelos con ordenación alfabética y solo el 25 % de los datos disponibles se comportan mejor que el mejor modelo con ordenación topológica utilizando el conjunto de datos completo.

Por último, para ilustrar el rendimiento, la Fig. 5.19 representa dos predicciones en el conjunto de test del mejor modelo de la Tabla 5.7. Como muestra la Fig. 5.17, el modelo etiqueta correctamente todos los nodos y recupera con éxito sus aristas correspondientes. Sin embargo, hay algunos errores en la Fig. 5.18: un nodo está mal etiquetado - predice una corchea en lugar de un *beam*, lo que puede considerarse un error razonable desde el punto de vista del reconocimiento gráfico de música, y el borde entre la cabeza de la nota inferior izquierda y el *beam* superior se pierde.

5.3.4 Conclusión

En este trabajo proponemos un nuevo modelo holístico de imagen a grafo con el que recuperar una representación en forma de grafo de los elementos presentes en una imagen de entrada. Proponemos una formulación basada en la clasificación secuencial de nodos y la reconstrucción de aristas por pares para obtener la transcripción de una partitura musical, el tema central de este trabajo.

En nuestra experimentación se proponen varios escenarios experimentales, como la conveniencia de los mecanismos de atención para el modelo recurrente, la disponibilidad de datos y los medios para mapear el conjunto de nodos del *ground truth* en una secuencia ordenada que se ciña a una representación en forma de grafo consistente.

Los resultados muestran que la metodología y la formulación propuestas tienen éxito en la resolución de la tarea de conversión de imágenes en grafos en el contexto específico de las

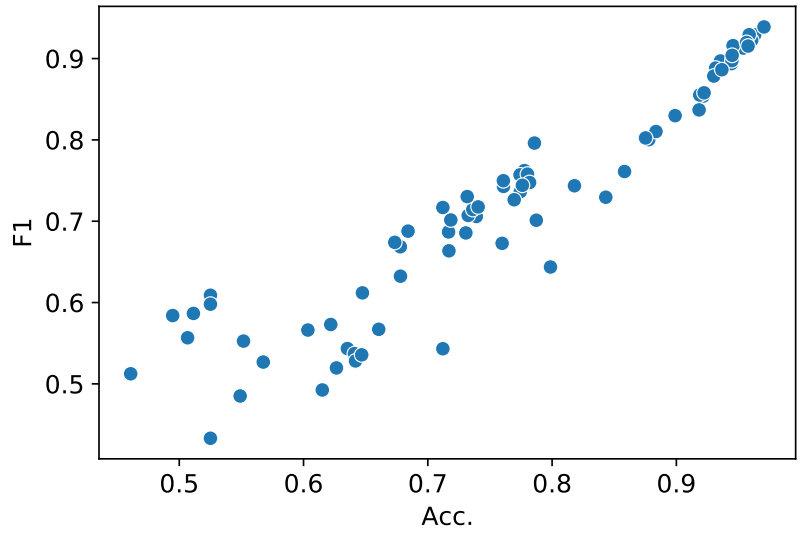


Figure 5.16: Diagrama de puntos que correlaciona los resultados en el conjunto de test obtenidos en cuanto al *Symbol Accuracy* y F_1 en cada *fold* del 5-CV para todos los modelos.

estructuras de notación musical. Nuestro método es capaz de recuperar el grafo a partir de la imagen sin necesidad de información espacial sobre los elementos de la fuente de entrada.

Este trabajo permite extraer ciertas conclusiones interesantes: en primer lugar, la selección de la representación del grafo según el ordenamiento secuencial de sus nodos parece tener una fuerte influencia en el rendimiento del modelo; en segundo lugar, el modelo necesita extraer aristas para cada par de nodos. Este cálculo puede ser costoso en escenarios de grafos grandes y requerir una cantidad de tiempo considerable para su entrenamiento.

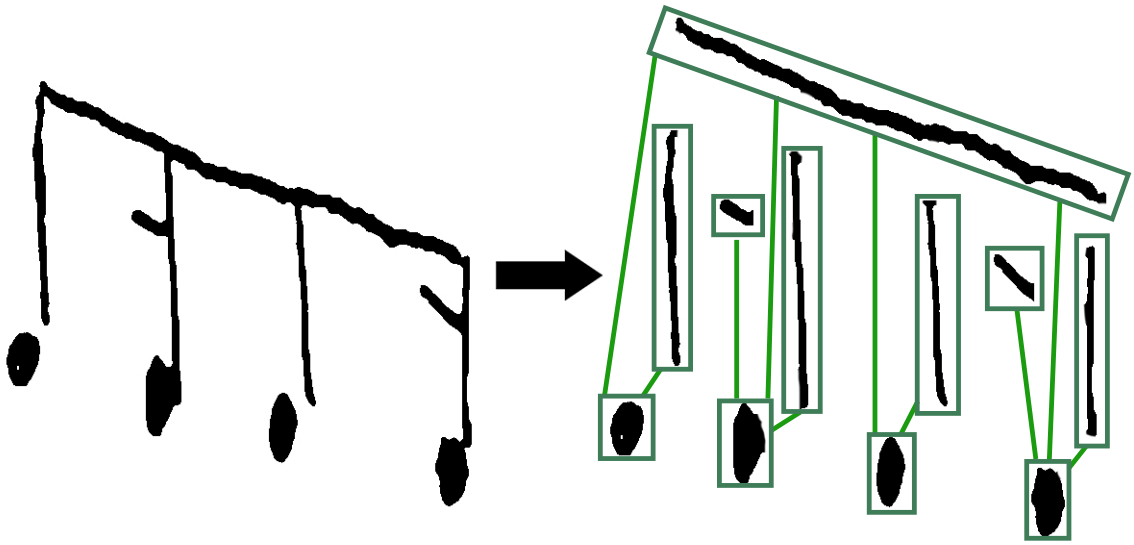


Figure 5.17: Ejemplo de una predicción correcta en la que los nodos y aristas se recuperan con éxito.

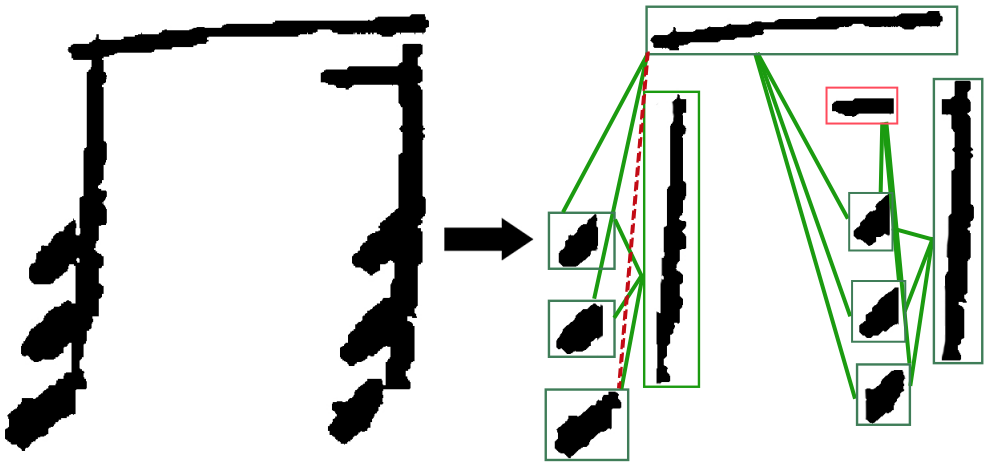


Figure 5.18: Ejemplo de predicción con errores. El recuadro rojo representa que el nodo ha sido etiquetado erróneamente, mientras que la línea roja discontinua indica que la arista entre los nodos correspondientes no ha sido predicha.

Figure 5.19: Ejemplos de predicciones del conjunto de test. Las imágenes de la izquierda representan la imagen de entrada dada al modelo, mientras que las de la derecha representan los grafos recuperados por nuestro mejor modelo. Obsérvese que los cuadros delimitadores (*bounding boxes*) representados son simplemente una ayuda visual, ya que nuestro enfoque no los recupera. El verde indica las predicciones correctas y el rojo los errores, que se describen en cada ejemplo.

6 Conclusiones

En este trabajo de fin de máster se ha abordado una brecha en la literatura del *Optical Music Recognition* (OMR): la recuperación de grafos que representen la notación musical a partir de una imagen. Para ello, se ha dividido el trabajo en tres estudios distintos. Los resultados en los experimentos realizados demuestran empíricamente la viabilidad de los tres enfoques.

En el primero, se propone una solución para recuperar las relaciones entre los distintos símbolos de la notación musical de las partituras completas, partiendo de una detección e identificación previa de los símbolos. Para ello, se establecen dos soluciones: un enfoque eficaz, con muy buenos resultados (hasta un 92 de F_1) a expensas de eficiencia, y otro muy eficiente, pero menos preciso (alrededor de 77 de F_1). Los resultados de este trabajo han sido enviados al *workshop* "*PatReCH 2022 - III International Workshop on Pattern Recognition for Cultural Heritage*", evento satélite del International Conference on Pattern Recognition 2022. Actualmente, se encuentra en estado de revisión.

En el segundo trabajo, se estudia la viabilidad de recuperar los símbolos (primitivas) musicales a partir extractos de partituras con enfoques que no utilicen técnicas de detección de objetos. Se proponen dos métodos basados en sistemas de *image captioning*, con el objetivo de estudiar la viabilidad de recuperar los símbolos musicales a partir de imágenes de extractos de partituras. Los resultados demuestran que ambos enfoques permiten capturar los símbolos de manera muy precisa (hasta 96.5% en la métrica *Intersection-Over-Union*). Estos resultados se presentaron recientemente en el "*IbPRIA 2022: 10th Iberian Conference on Pattern Recognition and Image Analysis*" (Garrido-Munoz et al., 2022), celebrado en Aveiro, Portugal.

Finalmente, en el tercer trabajo, se proponía una solución holística para recuperar los grafos completos en extracciones de partituras: los símbolos musicales (nodos) y las relaciones entre ellos (aristas). Para ello, se propuso una solución basada en redes neuronales convolucionales y redes recurrentes para obtener tanto los símbolos como las relaciones entre ellos. Los resultados demostraron que era posible recuperar de manera precisa el grafo completo, obteniendo resultados de 95.4 en la métrica *Symbol Accuracy* (1 - SER) y 91.6 en F_1 para las aristas. Este último trabajo está actualmente en una segunda fase de revisión para publicarse en la revista "*International Journal on Document Analysis and Recognition*" (IJDAR).

6.1 Trabajos futuros

A pesar de aportar avances en este hueco de la literatura científica de OMR, todavía falta mucho trabajo por hacer en este campo. A continuación, describimos las limitaciones y los trabajos futuros de cada uno de las propuestas de este trabajo de fin de máster.

6.1.1 Recuperación de relaciones

Surgen varias cuestiones con respecto a la metodología desarrollada en este proyecto. En primer lugar: ¿cuánto se consideraría este problema como "resuelto"? Actualmente, la literatura no da una definición clara de qué niveles de precisión se debería alcanzar para considerar esta tarea de recuperación de relaciones por solucionada. En general, definir cuándo una tarea se considera resuelta es un tema complejo. Podemos asumir dicha resolución cuando un usuario que revise la predicción de las relaciones por uno de estos sistemas propuestos no corrija más de N relaciones incorrectas por partitura. Queda por definir la base en la que se fundamentaría un criterio de umbral para esta área.

Por otra parte, en este trabajo hemos asumido que la tarea de recuperar relaciones entre símbolos puede partir de una detección idónea e identificación previa de los símbolos, ya que dicha tarea ha sido ampliamente tratada en la literatura de OMR. Sin embargo, es evidente que asumir una información perfecta sobre los símbolos no es viable, ya que ningún método actual de la literatura alcanza este nivel. ¿Qué impacto tendría partir de una predicción previa que detecte e identifique los símbolos con errores? ¿Qué impacto tendría en la recuperación final de relaciones?

En trabajos futuros se tratará de abordar estas dos cuestiones: (i) estudiar cuándo se consideraría resuelta esta tarea y (ii) estudiar el impacto de partir de una detección real previa en la extracción posterior de las relaciones entre símbolos.

6.1.2 Predicción de nodos/símbolos

En esta propuesta se utilizaban métodos de *image captioning* para recuperar los símbolos musicales a partir de extractos de partituras. Para ello, se formulaba el problema como una tarea *image-to-multiset*, el cual se transformaba como un problema de *image-to-sequence*. A pesar de que los enfoques propuestos funcionan bastante bien, es una primera aproximación en la que no se utiliza la partitura completa, sino que esta se va fragmentando en extractos, lo que facilita mucho la tarea y supone una limitación de las propuestas.

Como trabajos futuros, se plantea (i) estudiar el rendimiento de los métodos propuestos para partituras completas para la extracción de los símbolos y (ii) estudiar o proponer métodos que no sean dependientes del orden de la secuencia y puedan capturar conjuntos o multiconjuntos estrictamente (como los *DeepSets*, Zaheer et al. (n.d.)).

6.1.3 Recuperación completa de grafos

En esta última propuesta, se realiza una aproximación a la recuperación de manera holística de grafos a partir de extracciones de partituras. Es cierto que esto ha supuesto un nuevo *baseline* en la literatura, lo cual puede suscitar problemas comparativos con respecto a lo que ya se hace. Sin embargo, cabe destacar que esto es solo una aproximación, ya que se realiza a partir de fragmentos de la partitura y no la procesa completamente en un solo paso, lo cual supone una limitación.

Como trabajos futuros, se abordará (i) el estudio de métodos que no dependan del orden en las secuencias para sacar la representación del grafo Zaheer et al. (n.d.) o funciones de pérdidas para los mismos fines como en Carion et al. (2020) o (ii) métodos que recuperen las aristas de manera más eficiente. De igual forma, también trataremos de recuperar grafos

desde páginas completas, ya que los modelos de última generación están limitados en cuanto a la tarea de recuperar la representación de la secuencia de estos tipos de documentos. Esto abrirá un escenario interesante para el uso de la representación de grafos con el fin de abordar el problema completo de OMR. También planeamos probar nuestro enfoque en otras tareas de análisis de documentos que podrían beneficiarse de esta formulación holística de imagen a grafo.

Bibliography

Baró, A., Riba, P., Calvo-Zaragoza, J., & Fornés, A. (2019). From optical music recognition to handwritten music recognition: A baseline. *Pattern Recognition Letters*, *123*, 1–8.

Belli, D., & Kipf, T. (n.d.). Image-conditioned graph generation for road network extraction.

Byrd, D., & Simonsen, J. G. (2015). Towards a standard testbed for optical music recognition: Definitions, metrics, and page images. *Journal of New Music Research*, *44*(3), 169–195.

Calvo-Zaragoza, J., Jr., J. H., & Pacha, A. (2020). Understanding optical music recognition. *ACM Comput. Surv.*, *53*(4), 77:1–77:35.

Calvo-Zaragoza, J., Toselli, A. H., & Vidal, E. (2019). Handwritten music recognition for mensural notation with convolutional recurrent neural networks. *Pattern Recognit. Lett.*, *128*, 115–121.

Cao, N. D., & Kipf, T. (n.d.). Molgan: An implicit generative model for small molecular graphs.

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., & Zagoruyko, S. (2020). End-to-end object detection with transformers. In *European conference on computer vision* (pp. 213–229).

Chan, K.-F., & Yeung, D.-Y. (2000). Mathematical expression recognition: a survey. *International Journal on Document Analysis and Recognition*, *3*(1), 3–15.

Chang, X., Ren, P., Xu, P., Li, Z., Chen, X., & Hauptmann, A. (n.d.). Scene graphs: A survey of generations and applications.

Chu, H., Li, D., Acuna, D., Kar, A., Shugrina, M., Wei, X., … Fidler, S. (2019). Neural turtle graphics for modeling city road layouts. In *Proceedings of the ieee/cvf international conference on computer vision* (pp. 4522–4530).

Dalitz, C., Droettboom, M., Pranzas, B., & Fujinaga, I. (2008). A comparative study of staff removal algorithms. *IEEE Trans. Pattern Anal. Mach. Intell.*, *30*(5), 753–766.

David E. Rumelhart, R. J. W., Geoffrey E. Hinton. (1986). Learning representations by back-propagating errors. *Nature?*, *323*(1), 533–536.

Dreyfus, S. (1962). The numerical solution of variational problems. *Journal of Mathematical Analysis and Applications*, *5*(1), 30-45. Retrieved from `https://www.sciencedirect.com/science/article/pii/0022247X62900045` doi: https://doi.org/10.1016/0022-247X(62)90004-5

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

Fornés, A., Dutta, A., Gordo, A., & Lladós, J. (2012, September 1). Cvc-muscima: A ground truth of handwritten music score images for writer identification and staff removal. *International Journal on Document Analysis and Recognition*, *15*(3), 243–251. doi: 10.1007/s10032-011-0168-2

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. In *Biol. cybernetics* (pp. 193–202).

Garrido-Munoz, C., Ríos-Vila, A., & Calvo-Zaragoza, J. (2022). Retrieval of music-notation primitives via image-to-sequence approaches. In *Proceedings of the 10th iberian conference on pattern recognition* (Vol. 13256, pp. 482–492). Springer.

GoodFellow, I., Bengio, Y., & Courville, A. (2015). MIT. Retrieved from `https://www.deeplearningbook.org`

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Ivakhnenko, A. G., & Lapa, V. G. (1967). *Cybernetics and forecasting techniques*. New York, NY: North-Holland. Retrieved from `https://cds.cern.ch/record/209675` (Trans. from the Russian, Kiev, Naukova Dumka, 1965)

Jan Hajič, j., & Pecina, P. (2017). The MUSCIMA++ Dataset for Handwritten Optical Music Recognition. In *14th international conference on document analysis and recognition, ICDAR 2017, kyoto, japan, november 13 - 15, 2017* (pp. 39–46). New York, USA: IEEE Computer Society.

Jonas, E. (2019). Deep imitation learning for molecular inverse problems. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 32). Curran Associates, Inc.

Jurafsky, D., & Martin, J. H. (2009). *Speech and language processing (2nd edition)*. Prentice-Hall, Inc.

Kelley, H. J. (1960). Gradient theory of optimal flight paths. *ARS Journal*, *30*(10), 947-954. Retrieved from `https://doi.org/10.2514/8.5282` doi: 10.2514/8.5282

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In Y. Bengio & Y. LeCun (Eds.), *3rd international conference on learning representations, ICLR 2015, san diego, ca, usa, may 7-9, 2015, conference track proceedings.*

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989, 12). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, *1*(4), 541-551. Retrieved from `https://doi.org/10.1162/neco.1989.1.4.541` doi: 10.1162/neco.1989.1.4.541

Li, Y., Vinyals, O., Dyer, C., Pascanu, R., & Battaglia, P. W. (n.d.). Learning deep generative models of graphs.

Li, Y., Zhang, L. R., & ming Liu, Z. (2018). Multi-objective de novo drug design with conditional graph generative model. *Journal of Cheminformatics*, *10*.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. In *1st international conference on learning representations, ICLR 2013, scottsdale, arizona, usa, may 2-4, 2013, workshop track proceedings.*

Minsky, M., & Papert, S. (1969). *Perceptrons; an introduction to computational geometry.* MIT Press. Retrieved from `https://books.google.com.co/books?id=Ow1OAQAAIAAJ`

Pacha, A., Calvo-Zaragoza, J., & Jr., J. H. (2019). Learning notation graph construction for full-pipeline optical music recognition. In *Proceedings of the 20th international society for music information retrieval conference* (pp. 75–82).

Rebelo, A., Fujinaga, I., Paszkiewicz, F., Marcal, A. R., Guedes, C., & Cardoso, J. S. (2012). Optical music recognition: state-of-the-art and open issues. *International Journal of Multimedia Information Retrieval*, *1*(3), 173–190.

Rosenblatt, F. (1957, January). *The perceptron - a perceiving and recognizing automaton* (Tech. Rep. No. 85-460-1). Ithaca, New York: Cornell Aeronautical Laboratory.

Rossant, F., & Bloch, I. (2007). Robust and adaptive OMR system including fuzzy modeling, fusion of musical rules, and possible error detection. *EURASIP J. Adv. Sig. Proc.*.

Simonovsky, M., & Komodakis, N. (n.d.). Graphvae: Towards generation of small graphs using variational autoencoders.

Torras, P., Baró, A., Kang, L., & Fornés, A. (2021). On the integration of language models into sequence to sequence architectures for handwritten music recognition. In J. H. Lee et al. (Eds.), *Proceedings of the 22nd international society for music information retrieval conference, ISMIR 2021, online, november 7-12, 2021* (pp. 690–696).

Vento, M. (2015). A long trip in the charming world of graphs for pattern recognition. *Pattern Recognition*, *48*(2), 291–301.

Vinyals, O., Toshev, A., Bengio, S., & Erhan, D. (2015). Show and tell: A neural image caption generator. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3156-3164.

Wu, W., Xu, J., Li, H., & Oyama, S. (2010). Asymmetric kernel learning. *Technical Report, Microsoft Research.*

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., … Bengio, Y. (2015, 07–09 Jul). Show, attend and tell: Neural image caption generation with visual attention. In F. Bach & D. Blei (Eds.), *Proceedings of the 32nd international conference on machine learning* (Vol. 37, pp. 2048–2057). Lille, France: PMLR.

Yang, C., Zhuang, P., Shi, W., Luu, A., & Li, P. (2019). Conditional structure generation through graph variational generative adversarial nets. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 32). Curran Associates, Inc.

You, J., Ying, R., Ren, X., Hamilton, W. L., & Leskovec, J. (n.d.). Graphrnn: A deep generative model for graphs.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., & Smola, A. J. (n.d.). Deep sets. In I. Guyon et al. (Eds.), *Advances in neural information processing systems.* Curran Associates, Inc.

Zhang, J., Du, J., Zhang, S., Liu, D., Hu, Y., Hu, J., … Dai, L. (2017). Watch, attend and parse: An end-to-end neural network based approach to handwritten mathematical expression recognition. *Pattern Recognition*, *71*, 196–206.

Acrónimos y lista de abreviaciones

AE	*Autoencoder.*
AK	*Asymmetric Kernels.*
AN	*Artificial Neuron.*
ANN	*Artificial Neural Network.*
CL	*Continual Learning.*
CNN	*Convolutional Neural Network.*
CV	*Computer Vision.*
CV	*Cross Validation.*
DA	*Domain Adaptation.*
DIA	*Document Image Analysis.*
DL	*Deep Learning.*
FCNN	*Fully Convolutional Neural Network.*
FN	Falsos Negativos.
FP	Falsos Positivos.
GRFIA	Grupo de Reconocimiento de Formas e Inteligencia Artificial.
IA	Inteligencia Artificial.
IoU	*Intersection-Over-Union.*
LSTM	*Long Short-Term Memory.*
ML	*Machine Learning.*
MLP	*Multilayer Perceptron.*
NA	Neurona Artificial.
NLP	*Natural Language Processing.*
OCR	*Optical Character Recognition.*
OMR	*Optical Music Recognition.*
RL	*Reinforcement Learning.*
RNN	*Recurrent Neural Networks.*
ROC	Reconocimiento Óptico de Caracteres.
ROM	Reconocimiento Óptico de Música.
SER	*Symbol Error Rate.*
TFG	Trabajo Final de Grado.
TL	*Transfer learning.*
VN	Verdaderos Negativos.
VP	Verdaderos Positivos.