TRABAJO FIN DE GRADO

INGENIERÍA INFORMÁTICA

Etiquetado de imágenes en química

Autor
Pedro Bedmar López

Directora
Rocío Celeste Romero Zaliz

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, julio de 2022

Etiquetado de imágenes en química

Pedro Bedmar López

Palabras clave: palabra_clave1, palabra_clave2, palabra_clave3, ...

Resumen

Poner aquí el resumen.

Image labelling in chemistry

Pedro Bedmar López

Keywords: Keyword1, Keyword2, Keyword3, ...

Abstract

Write here the abstract in English.

Granada a X de mes de 201 .

Dña. **Rocío Celeste Romero Zaliz**, Profesora Titular del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado ***Etiquetado de imágenes en química***, ha sido realizado bajo su supervisión por **Pedro Bedmar López**, y autoriza la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expide y firma el presente informe en Granada a X de julio de 2022.

La directora:

Rocío Celeste Romero Zaliz

Agradecimientos

Poner aquí agradecimientos...

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Debido al desarrollo exponencial de la informática en el último siglo, todas las ramas del conocimiento se han visto afectadas. La informática está permitiendo la automatización de muchas tareas que anteriormente se realizaban de forma manual por un operario.

Con el paso de los años, esta capacidad de automatización va en aumento pudiéndose aplicar en situaciones anteriormente impensables. Los numerosos avances en hardware y la aplicación de arquitecturas GPU en el campo del Aprendizaje Automático, y concretamente del Deep Learning, han permitido crear modelos mucho más avanzados capaces de interiorizar datos más complejos.

La química es una de las ciencias que se ha impregnado de este desarrollo tecnológico y de esta combinación ha surgido lo que se conoce como cheminformatics o chemoinformatics. En este capítulo definiremos esta rama científica y describiremos sus principales actuaciones. A continuación, explicaremos distintos conceptos y tecnologías existentes relacionados con el proyecto.

1.1. Motivación del proyecto

Desde hace décadas, en el mundo de la química ha estado presente la necesidad de almacenar, gestionar y procesar la gran cantidad de información que se genera. Con el tiempo se fueron desarrollando técnicas de tratamiento de ésta, pero no fue hasta hace algunos años cuando se acuñó el nombre de cheminformatics o chemoinformatics.

En la literatura existen diferentes definiciones para este término, discutidas en [3].
"Chem(o)informatics es un término genérico que encompasa el diseño, crea-

ción, gestión, recuperación, análisis, diseminación, visualización y el uso de información química" es una de las definiciones recogidas. Otra más abierta es "La aplicación de métodos informáticos para resolver problemas de química".

Desde la Universidad de Granada, la tutora de este TFG trabaja en este ámbito. Colabora con químicos de la Universidad de Negev y es consciente de los problemas que tienen para manejar la gran cantidad de datos que aparecen en publicaciones científicas. Un tipo de datos muy valioso son las imágenes, pero clasificarlas no es trivial: pueden ser sobre cualquier temática, algunas pueden referirse a esquemas explicando cómo funciona un modelo, otras pueden contener resultados de algún experimento, pueden ser representaciones de compuestos químicos, etc. Clasificarlas manualmente por un operario no es una opción viable, ya que se consumirían demasiados recursos en una tarea que actualmente se puede automatizar.

Otra necesidad que tienen estos científicos es la de conseguir más imágenes: etiquetarlas de forma manual es una tarea tediosa.

Es por ello que en este Trabajo de Fin de Grado vamos a crear dos utilidades, un generador de imágenes sintéticas (tanto de imágenes de moléculas como de ejemplos negativos) y un modelo que permita clasificar imágenes. En concreto, aquellas que son representaciones de moléculas organometálicas, del resto.

Aunque existen diferentes opiniones sobre el alcance de las cheminformatics, se puede considerar que este proyecto está dentro de sus fronteras, ya que vamos a crear una herramienta de clasificación de imágenes químicas, es decir, una herramienta que procesa y analiza este tipo de información.

1.2. Objetivos

El proyecto tiene como fin cumplir los siguientes objetivos:

- **[OBJ1]** Crear un conjunto de datos para clasificar compuestos organometálicos.
  - **[OBJ1.1]** Generar imágenes sintéticas de compuestos químicos.
  - **[OBJ1.2]** Generar imágenes sintéticas de compuestos no químicos.
- **[OBJ2]** Clasificar imágenes que presentan compuestos organometálicos de aquellas que no.

Capítulo 2

Gestión y planificación del proyecto

2.1. Metodología

En los primeros años del desarrollo de software, este se creaba sin seguir ningún enfoque formal. Muchos de los proyectos que se iniciaban terminaban fracasando por los retrasos en la entrega, el mal funcionamiento del producto o por no cumplir los requisitos del cliente. Además, la complejidad requerida en el software iba aumentando con el tiempo.

Por ello, era necesario crear un marco que permitiera metodizar el desarrollo. Es así como surge la Ingeniería del Software, que acompaña al software durante todo su ciclo vital.

En este proyecto vamos a aplicar este enfoque a la hora de desarrollar el producto, apoyándonos sobre una metodología de desarrollo para asegurarnos que se consiguen los objetivos en el tiempo previsto, detectando posibles amenazas y problemas a tiempo.

Las metodologías se pueden clasificar en dos grandes bloques [21], tradicionales y ágiles. Las tradicionales son las que primero surgieron, se caracterizan por definir rígidamente los requisitos al inicio del proyecto. En ellas, se aplica una serie de etapas de forma lineal, y una vez alcanzada una de ellas no se puede volver atrás. Por todo esto no se adaptan bien a los cambios.

En general, los proyectos de software tienden a ser cada vez más complejos. Las metodologías ágiles surgieron con el objetivo de hacer los proyectos de desarrollo más dinámicos, de forma que se adaptaran mejor al entorno y a los cambios. Se basan en una metodología incremental, donde se van construyendo prototipos del producto poco a poco, añadiendo funcionalidades hasta obtener la aplicación final. Los equipos se reunen cada poco tiempo

para intercambiar ideas y repartir las tareas a realizar.

A la hora de elegir la metodología que vamos a usar, debemos tener en cuenta los siguientes factores relativos a la naturaleza del proyecto:

- Se trata de un proyecto de investigación, donde vamos a aplicar diferentes técnicas de Aprendizaje Automático a la resolución de un problema. Por tanto, a priori no se conoce la calidad de los resultados que se van a obtener y el número y tipo de experimentos que va a ser necesario realizar.
- La intervención del experto en química va a ser fundamental durante el desarrollo del proyecto. Aportará información y feedback esencial durante todas las fases.

Por estas razones, creemos que la metodología de desarrollo que mejor se adapta a nuestras necesidades es una metodología ágil, ya que nos provee de gran flexibilidad, permitiendo el desarrollo del proyecto de una forma incremental donde obtenemos feedback del cliente y experto en cada iteración.

Revisando las distintas metodologías [4], creemos que una buena candidata es SCRUM [2]. Es posiblemente una de las más utilizadas en la actualidad, y los proyectos que la aplican cuentan con las siguientes características:

- **Entregable flexible:** Su contenido viene dado por lo que demanda el entorno.
- **Calendario flexible:** El entregable puede ser requerido antes o después de lo previsto.
- **Equipos pequeños:** Los equipos están formados por pocas personas, de forma que la comunicación y sincronización entre sus miembros es alta.
- **Revisiones frecuentes:** El progreso del equipo se evalúa de forma periódica y frecuentemente, de forma que se ponen en común las dificultades encontradas y se intentan resolver con la ayuda de todos los miembros.
- **Colaboración:** La colaboración entre todos los miembros del equipo es muy alta, así como entre el equipo y entidades externas como el cliente.

Cuando se trabaja con esta metodología, en cada equipo existe un miembro conocido como SCRUM manager. Es el encargado de guiar al equipo en

el desarrollo y en la aplicación de la metodología. En SCRUM el equipo es muy importante y todos sus miembros participan con su opinión. Esta metodología consta de las siguientes fases:

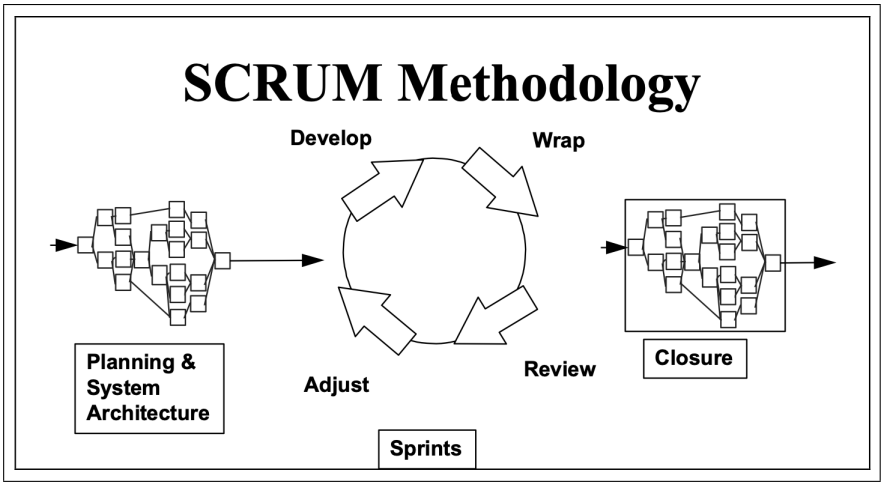


Figura 2.1: Fases de la metodología SCRUM [2]

- **Pregame:**
  - *Planificación:* En esta fase se crea la lista de tareas a realizar (backlog list), se fija la fecha de entrega del producto y su funcionalidad, se forma el equipo (o equipos) de trabajo, se valoran los riesgos que puedan surgir, los costes y finalmente se revisa todo y se aprueba el proyecto.
  - *Arquitectura/Diseño de alto nivel:* Se revisan los elementos en el backlog, se realizan cambios si es necesario para poder implementarlos, se perfila la arquitectura del sistema teniendo en cuenta estos cambios y una reunión es organizada para revisar el diseño, donde cada equipo presenta una propuesta para implementar cada backlog.

- **Game:** Corresponde con el conocido Sprint de la metodología SCRUM. Es la parte iterativa de la metodología, que se ejecuta varias veces hasta conseguir el producto final y está formada por las siguientes subtareas:
  - *Desarrollo:* Lo primero que se realiza es definir los cambios que hay que realizar en los backlogs para poder implementarlos. Se dividen las tareas presentes en el backlog en paquetes, y se completan estos paquetes diseñando, desarrollando, implementando, testeando y documentando los cambios.

  - *Envoltura:* Se cierran los paquetes, creando una ejecutable que incorpora los cambios y se explica como cumplen lo especificado en los backlogs.
  - *Revisión:* Todos los equipos se reunen para presentar el trabajo. El desarrollo obtenido se evalúa y se añaden nuevas tareas que puedan surgir al backlog. Se evalúa el riesgo y se realizan propuestas en base a este.
  - *Ajuste:* Se consolida la información recibida durante la revisión.

- **Postgame:**
  - *Cierre:* Cuando el gestor del proyecto considera que el producto está terminado y cumple con los requisitos solicitados por el cliente se entra en esta fase, donde se prepara el producto para su despliegue. Integración, generación de la documentación, testeo y marketing son algunas de las actividades que se realizan en este paso.

Estas características y fases que hemos mencionado son las especificadas en la definición original de SCRUM. Aún así, en la práctica cada proyecto las adapta a sus necesidades. En general, los sprints suelen tener una duración de 2 o 3 semanas.

2.2. Planificación

Para lograr cumplir con los objetivos, se planifican las siguientes tareas a realizar:

- [**T1**] Estudio del estado del arte del problema.
  - [**T1.1**] Lectura de publicaciones introductorias a las Cheminformatics.
  - [**T1.2**] Repaso de la literatura sobre aprendizaje profundo, comprendiendo los principales modelos de generación y clasificación de imágenes.
  - [**T1.3**] Conocimiento de las principales herramientas que se han desarrollado en ambos ámbitos.
- [**T2**] Implementación/adaptación de un modelo de síntesis de imágenes entrenado con nuestro propio conjunto de datos.
  - [**T2.1**] Sintetizar imágenes de moléculas organometálicas.

    - [**T2.2**] Sintetizar imágenes de moléculas que no son organometálicas, es decir, ejemplos negativos.

- [**T3**] Implementación de un modelo clasificador de imágenes, capaz de distinguir moléculas organometálicas de ejemplos negativos.

    - [**T3.1**] Realizar pruebas con diferentes arquitecturas e hiperparámetros para comprobar cuáles se adaptan mejor al problema.
    - [**T3.2**] Creación de ejecutable que reciba una imagen y la clasifique, utilizando la mejor configuración encontrada.

- [**T4**] Documentación del trabajo realizado.

Para el éxito del proyecto es necesario asignar una temporización razonable a cada tarea. Este se realiza en el marco de un Trabajo de Fin de Grado (TFG): en la Universidad de Granada está cuantificado en 12 créditos, lo que significan 300 horas de trabajo. Creemos que los objetivos propuestos son adecuados para ser realizados en ese número de horas, sin ser excesivos ni escasos.

Los TFGs se suelen realizar en un único cuatrimestre, que normalmente coincide con el último del grado. Esto significa cuatro meses en los que repartir las 300 horas, desde marzo hasta junio. Bien es cierto que este proyecto se empezó con algo de anterioridad, en febrero, para comenzar a profundizar en la literatura y conocer de forma más precisa las necesidades de los investigadores de Negev.

(Insertar diagrama de Gantt con las tareas/objetivos y su temporización)

2.3. Gestión de recursos

2.3.1. Recursos humanos

El proyecto cuenta con dos personas trabajando sobre él:

- **Rocío Romero Zaliz.** Profesora del Departamento de Ciencias de la Computación e Inteligencia Artificial tutora del proyecto. Encargada de supervisar al alumno, guiando su trabajo y aconsejándole cuando lo necesite. Se reunirá con él de forma periódica para comentar los avances e indicar posibles pasos a seguir.

- **Pedro Bedmar López.** Estudiante del Grado en Ingeniería Informática que realiza el TFG. Se encargará de su desarrollo y de la elaboración de la memoria.

2.3.2. Recursos materiales

Una ventaja del sector tecnológico es que muchos proyectos no necesitan una gran infraestructura para llevarse a cabo. En nuestro caso hacemos uso de lo siguiente:

- **Ordenador personal.** Equipo desde el que trabajará el estudiante. Lo utilizará para consultar la literatura, implementar el código y redactar la memoria. En concreto, se trata de un MacBook Pro 15' 2015, que monta un procesador Intel Core i7 a 2.5 GHz y 16 GB de RAM.
- **Monitor.** Utilizado como apoyo a la pantalla del portátil. El modelo utilizado ha sido fabricado por la marca Benq, y presenta una resolución 1920x1080.
- **Clúster GPU.** Hoy en día, la mayoría del entrenamiento de modelos en Aprendizaje Automático y Deep Learning no se lleva a cabo de forma local, sino en un clúster habilitado para ello. La Universidad de Granada, y en concreto el Instituto DaSCI (Instituto Andaluz Interuniversitario en Data Science and Computational Intelligence), cuenta con uno de ellos donde mediante SLURM se pueden enviar trabajos que se ejecutaran en una GPU, acelerando en gran medida los entrenamientos.

2.3.3. Recursos software

Utilizamos distintos programas que nos apoyan en distintas acciones:

- **macOS Monterey 12.3.1.** Por encima del resto de programas, se encuentra el Sistema Operativo, encargado de administrar los recursos hardware y controlar la ejecución de procesos.
- **PyCharm 2021.3.3.** IDE desarrollado por la compañía JetBrains, ideal para trabajar con proyectos en Python e incluso con Jupyter Notebooks. Además, se integra directamente mediante SSH con el clúster GPU, permitiendo la sincronización de ficheros entre el espacio de trabajo local y el remoto. Aunque es un software de pago, los estudiantes pueden descargarlo de forma gratuita.
- **Git y GitHub.** Git es un software de control de versiones que permite ir almacenando versiones intermedias del código, de forma que se puede volver hacia atrás si es necesario. También facilita el trabajo en equipo, ya que puede combinar el trabajo de múltiples personas de forma automática (siempre y cuando no hayan trabajado sobre la misma línea). GitHub es una plataforma de alojamiento de proyectos

que utilizan Git. Nos es útil, ya que actúa como almacenamiento de seguridad de nuestro trabajo.

- **Google Meet.** Producto de videoconferencias de Google. Es la plataforma oficial de comunicación síncrona en la Universidad de Granada. Será utilizado en las reuniones entre los integrantes del equipo.

- **VSCode y LaTeX.** VSCode es un editor de código desarrollado por Microsoft. Cuenta con multitud de extensiones creadas por la comunidad, entre ellas destacan algunas que permiten el formateo y compilado de ficheros LaTeX. Este software es un sistema de composición de textos utilizado para escribir esta memoria.

2.4. Presupuesto

En esta sección presentamos el presupuesto del proyecto.

2.4.1. Coste de recursos humanos

La mayor parte del coste vendrá generado por las horas de trabajo dedicadas al proyecto por el alumno.

Según la Universidad Europea, el salario medio de un informático recién egresado ronda los 18000 - 22000€ al año [14]. Si contamos el número de días hábiles para este año 2022, obtenemos 253 días [13], a los que hay que restarle 22 días de vacaciones [16]. En total trabajaría 231 días al año ocho horas por día, lo que hace un total de 1848 horas. Si tomamos como salario medio los 20000 euros anuales y los dividimos por el número de horas que se trabajan al año, obtenemos un coste por hora de 10.82€.

2.4.2. Otros costes

A la cantidad total hay que añadirle los impuestos y tasas correspondientes, en nuestro caso el IVA (21 %). En mi caso, por ser mi primer año como autónomo, el tipo de IRPF que me corresponde es del 7 %.

2.4.3. Presupuesto final

El presupuesto queda desglosado en la tabla inferior, siendo el importe total a percibir 3700.44€.

Artículo	Coste (€)	Cantidad	Importe
Hora de trabajo autónomo	10.82€	300	3246€
Base imponible:			3246€
		IVA 21 %	681.66€
		Ret. 7 % IRPF	227.22€
Total:			3700.44€

Capítulo 3

Estado del arte

Para poder organizar este proyecto debemos conocer el estado del arte en su ámbito. Hablaremos del estado de arte en cheminformatics, las herramientas que se han creado en el marco de esta ciencia y las técnicas de Deep Learning que utilizamos para desarrollar el TFG.

3.1. Cheminformatics

Como comentamos en la introducción, cheminformatics es un tema muy amplio y engloba subtópicos diferentes. Muchos de ellos surgieron en la década de 1960 y principios de 1970, y desde esa época muchos grupos de investigación siguen trabajando en ellos y nuevos grupos han surgido para aplicar nuevas tecnologías en nuevos ámbitos. A continuación discutimos algunos de los temas que históricamente han sido de interés para esta ciencia. [3]

3.1.1. Compuestos orgánicos y su representación

Un compuesto orgánico es un compuesto químico que contiene átomos de carbono, formando enlaces carbono-carbono y carbono-hidrógeno [24]. En este TFG, vamos a trabajar concretamente con la clasificación de compuestos organometálicos, donde los átomos de carbono forman enlaces covalentes con átomos metálicos [25].

En las publicaciones de química encontramos numerosas representaciones gráficas de estos compuestos, lo que se conocen como fórmulas estructurales. Éstas muestran la disposición en el espacio de los átomos que forman el compuesto. Un ejemplo es la siguiente figura:

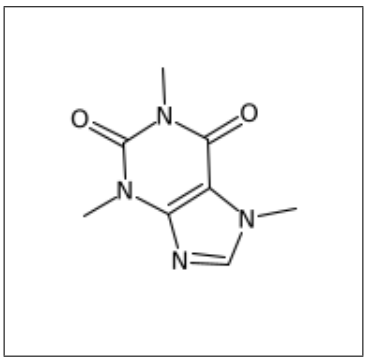


Figura 3.1: Estructura de la cafeína

En ellas pueden aparecer multitud de elementos, apareciendo siempre:

- **Átomos:** Se sitúan en los extremos de los enlaces. Representados con letras que indican el elemento químico del que se trata, tal y como aparece en la tabla periódica.
- **Enlaces:** Unen dos átomos entre sí.

En algunas ocasiones, sobre los átomos pueden mostrarse cargas positivas o negativas representando iones. Aparte, puede mostrarse lo que se conoce como información estereoquímica, que indica la disposición de los átomos en el espacio. Ésta es importante ya que afecta a las propiedades y reactividad de las moléculas [26, 22].

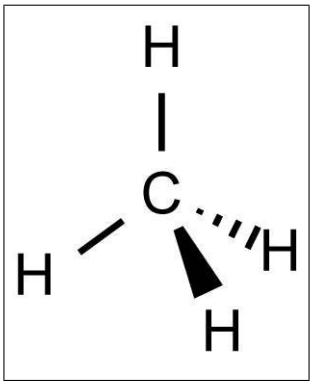


Figura 3.2: Estructura del metano

- Las líneas sólidas representan enlaces en el plano.
- Las discontinuas representan enlaces que están más alejados.

- Aquellas con forma de cuña indican que uno de los átomos se encuentra más cerca del espectador.

Además, un mismo compuesto se puede representar de distintas formas:

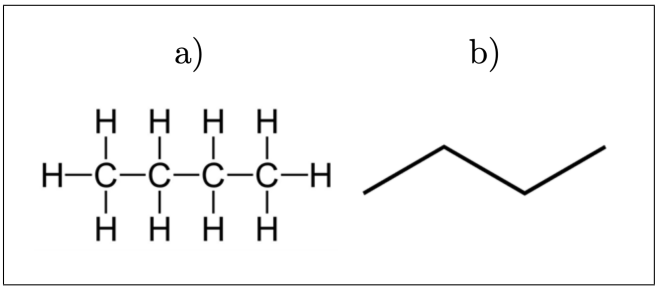


Figura 3.3: Dos formas de representar el butano

En a) se detalla la definición de todos los átomos, en cambio en b) encontramos lo que se conoce como fórmula de esqueleto, donde se omiten los átomos de carbono e hidrógeno. Se sabe que hay un átomo de carbono en los vértices que quedan libres en la intersección de dos enlaces o en las terminaciones donde no aparece ningún otro elemento. Se supone a la vez que cada átomo de carbono tiene cuatro enlaces, por tanto el número de enlaces que faltan por indicar explícitamente se corresponden con enlaces a moléculas de hidrógeno [27, 22].

3.1.2. Representación en el ordenador

El poder almacenar representaciones de compuestos químicos de manera eficiente en un ordenador requiere de la creación de métodos y formatos específicos para ello. Además, hay que tener en cuenta qué datos vamos a codificar, si solo la estructura básica del compuesto, si también queremos guardar información estereoquímica o si queremos añadir notas auxiliares sobre los compuestos. Es importante tener esto claro, ya que la complejidad de la representación influirá en la cantidad de almacenamiento que ocupe en disco y en los recursos necesarios para procesarla.

Utilizando notación lineal, representamos la estructura del compuesto como una secuencia lineal de caracteres y números. Esta es una codificación adecuada para las computadoras, ya que la pueden procesar con facilidad. Algunos formatos que utilizan esta notación son WLN (Wiswesser Line Notation), ROSDAL (Rp) o SMILES (Simplified Molecular Input Line Entry Specification). Aunque WLN y ROSDAL han quedado obsoletos, SMILES se sigue utilizando con mucha frecuencia en la actualidad. [3]

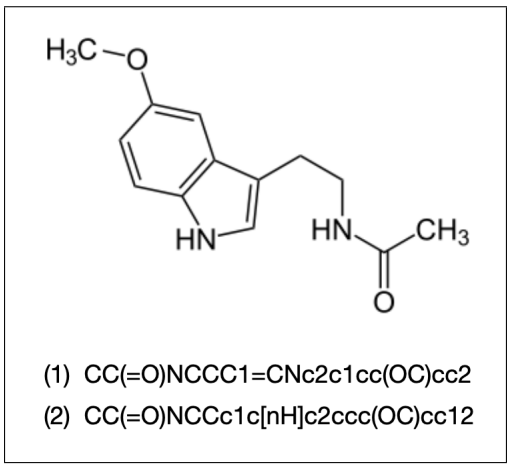


Figura 3.4: Dos posibles codificaciones de la melatonina en SMILES [28]

Aunque no vamos a entrar en detalle de cómo funciona, ya que para ello hay que tener nociones avanzadas de química, diremos que utiliza las siglas de cada elemento de la tabla periódica para representar los átomos. La primera letra del elemento se escribe en mayúscula, a no ser que se trate de un átomo perteneciente a un anillo aromático[1], ya que en ese caso se escribe en minúscula. Si el elemento tiene dos caracteres, el segundo se escribe siempre en minúscula. Además, se pueden representar cargas.

Los enlaces se representan con -, =, # y :, según el tipo. Bajo algunas circunstancias, se pueden omitir estos símbolos, ya que por su contexto se deducen. También se pueden codificar ramas, situando elementos entre paréntesis, y ciclos, utilizando un número para indicar el inicio y el fin del ciclo en la cadena de texto. [1]

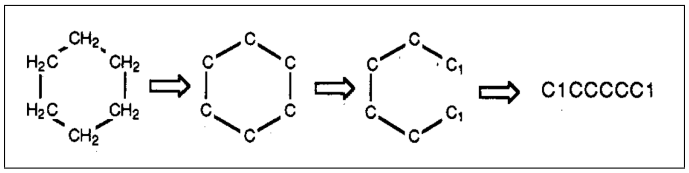


Figura 3.5: Ciclos en SMILES

Otras características como la aromaticidad o estructuras inconectas también pueden ser representadas. Entre las ventajas de esta codificación, destaca su facilidad de comprensión por los humanos. Cualquier químico puede aprender sus reglas de codificación fácilmente y diseñar sus propios com-

[1]La aromaticidad es una propiedad presente en enlaces dobles de moléculas cíclicas, donde sus electrones pueden circular libremente. Esto mejora la estabilidad del compuesto. [12, 23]

puestos. Un problema que tiene SMILES es que un mismo elemento se puede representar de diferentes formas. Pero sobre todo, el mayor problema es que un porcentaje significativo de las cadenas no se corresponden con moléculas válidas, ya sea porque son sintácticamente inválidas, no se corresponden con un grafo molecular o no cumplen reglas químicas básicas. [1]

Uno de los principales objetivos de la química computacional es el diseño de nuevas moléculas. Para ello, la utilización de modelos generativos puede ayudar a los investigadores, pero si el espacio de estados de SMILES no es completamente válido se dificulta la tarea. Para ello han surgido otras codificaciones como SELFIES con un espacio 100 % robusto. [9]

Además de estos formatos de notación lineal, es necesario mencionar otros. El lanzamiento en 1982 de MDL MOLfile llevó a su aceptación como principal formato para codificar compuestos químicos. Se han realizado distintas adaptaciones de este para añadir información extra a las moléculas, dando lugar a SDfile (puede agrupar más de un MOLfile y almacenar información estructural), RXNfile (anota los reactivos y productos de una única reacción química), etc. El formato PDB se utiliza principalmente para almacenar información 3D de macromoléculas biológicas, como son las proteínas o los polinucleótidos. CIF también es un formato para almacenar información 3D. En espectroscopia encontramos JCAMP. Finalmente, CML (chemical markup language), una extensión de XML, es una propuesta que intenta aglutinar toda la información disponible. Es compatible con moléculas, reacciones, espectroscopia y otra información. [3]

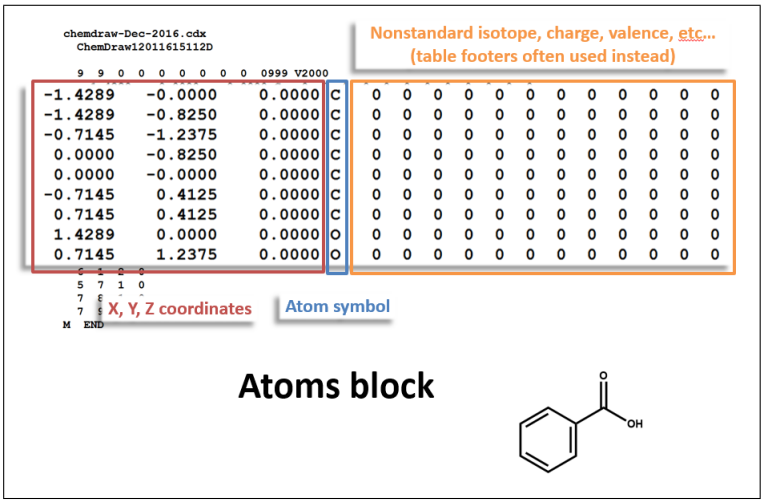


Figura 3.6: Contenido de un archivo MOL [11]

3.1.3. Fuentes y bases de datos

El gran número de facetas que presenta la información química necesita sistemas de almacenamiento a la altura. La química fue una de las primeras ramas científicas en utilizar bases de datos para almacenar: la cantidad de datos que se generaba creció rápidamente, y sigue creciendo hoy en día.

Aunque es complicado clasificarlas, vamos a separarlas en tres grandes grupos según el tipo de información que almacenan: [3]

- **Bases de datos de publicaciones:** Pueden ser bibliográficas, guardando solamente los metadatos y la referencia a la publicación, o de texto completo, donde recogen la publicación de forma íntegra.
- **Bases de datos fácticas:** Al contrario que las anteriores, que almacenan publicaciones de la literatura primaria, estas pueden guardar propiedades físicas de compuestos, información de espectroscopia, información legal, etc.
- **Bases de datos de estructuras y reacciones:** Recogen estructuras químicas, tanto individualmente como formando parte de reacciones. No se almacenan como imágenes, sino en formatos interpretables por la máquina.
- **Bases de datos de biología molecular:** Contienen secuencias de aminoácidos y nucleótidos.

Pero, ¿cómo podemos rellenarlas con datos? ¿De dónde podemos extraerlos ? Durante décadas se han publicado un gran número de artículos. Podríamos utilizarlos como una fuente muy amplia de información, ya que contienen todo el progreso científico. Específicamente para extraer datos relativos a estructuras, entran en juego utilidades conocidas como OCSR (Optical Chemical Structure Recognition).

Son capaces de transformar una imagen en un formato compatible con la máquina, como podría ser SMILES. En muchos casos incluso es posible introducirles una publicación completa y ellas mismas localizan las imágenes de moléculas. Con el paso del tiempo se han ido perfeccionando, y algunas son capaces de detectar información estereoquímica o de relacionar el compuesto con el texto de la publicación. Más adelante repasaremos algunas de estas utilidades.

Por último, mencionar una base de datos que merece la pena conocer. Creada por el National Institute of Health (NIH), PubChem es una base de datos abierta que cada mes sirve a millones de usuarios en todo el mundo. Es una base de datos de estructuras y aunque contiene mayoritariamente moléculas pequeñas, también almacena nucleótidos, carbohidratos, lípidos,

péptidos y macromoléculas modificadas químicamente. Para cada compuesto almacena su estructura, identificadores, propiedades físicas y químicas, toxicidad, patentes, etc. Los datos que aglutina provienen de diversas fuentes, como son agencias del gobierno estadounidense, editores de revistas científicas o proveedores químicos, aunque hay muchas más. [17]

3.1.4. Métodos de búsqueda

Almacenar información en las bases de datos no sirve de nada si no se desarrollan métodos eficientes para extraerla. En aquellas bases de datos donde se almacenan estructuras químicas, una de las principales formas de obtener información es buscar similitudes entre una molécula dada como entrada y otras que se encuentran almacenadas, de forma que compartan una subestructura específica o tengan otras características en común. Para ello, es clave la codificación de los compuestos.

También, si se almacenan metadatos y la base de datos está indexada sobre ellos se podría buscar por su nombre, etiquetas, etc. [3]

3.1.5. Métodos para análisis de datos

En química, grandes cantidades de datos son producidas. Una vez que hemos conseguido limpiarlos y ordenarlos, tenemos un conocimiento muy valioso en nuestras manos. La información es muy interesante en sí misma, pero también lo son las relaciones que se esconden en su interior. Para ello, se crean modelos que puedan interiorizarlas.

El análisis de datos no solo se enfrenta a la extracción de la información principal, sino que también intenta generar nueva información secundaria. [3]

Este TFG se desarrolla dentro de este ámbito, ya que, como describiremos más adelante, entrenamos un modelo capaz de detectar que imágenes contienen moléculas organometálicas. Además, creamos un modelo generativo para aumentar el tamaño del dataset en uso.

3.2. Deep learning

En las últimas décadas, la Inteligencia Artificial (I.A) ha vivido un periodo de crecimiento brutal. Dentro del ámbito de la I.A. se engloban técnicas muy diferentes que tienen como objetivo simular el comportamiento inteligente de los seres vivos, siendo uno de sus rasgos la capacidad de aprendizaje. Los sistemas expertos fueron su primer éxito comercial. En ellos se codificaban una serie de reglas manualmente, emulando el razonamiento de un experto en un problema específico.

El aprendizaje automático es un subconjunto dentro de la Inteligencia Artificial. Aglutina aquellas técnicas que permiten a un ordenador construir modelos a partir de conjuntos de datos, aprendiendo con la experiencia. No hace falta definir reglas de forma explícita. Pero tiene un inconveniente, y es que en muchos casos un especialista humano tiene que elegir manualmente que características de los datos son relevantes en el problema.

El aprendizaje profundo o deep learning va un paso más allá. Es un subconjunto del aprendizaje automático, pero en las técnicas que engloba no es necesario indicar al modelo las características útiles, ya que él mismo las descubre automáticamente. Y no se queda ahí, va un paso más allá, puede generar nuevas características a partir de las existentes.

El desarrollo temporal de estas técnicas coincide con el orden en el que las menciono. Como se puede apreciar, el proceso de aprendizaje consta de diferentes fases, de las que cada vez se encuentra automatizado un mayor número. [7]

Casi todos los algoritmos de deep learning son una adaptación particular de un proceso que se puede resumir en los siguientes cuatro pasos:

- Extraer un conjunto de datos asociado al problema (un conjunto de gran tamaño, cuanto más grande mejor).
- Diseñar una función de coste apropiada (loss function).
- Crear un modelo y definir sus hiperparámetros (tamaño, tasa de aprendizaje...), asignándole los valores más adecuados.
- Aplicar un algoritmo de optimización para minimizar la función de coste.

Siendo esta la estrategia que se utiliza en otras muchas técnicas de aprendizaje automático. La diferencia entre el deep learning y estas otras es la capacidad de abstracción que tiene el primero, que viene dada por la capacidad de jerarquizar la información en distintas capas utilizando múltiples niveles de representación. Esta abstracción permite separar la esencia de

lo prescindible, de forma que los modelos pueden aprender qué relaciones entre los datos son importantes y cuáles son simples accidentes, y consecuentemente generalizar de forma adecuada. [7]

3.2.1. Redes neuronales multicapa

Nuestro cerebro contiene un tipo especial de células llamadas neuronas. Estas reciben impulsos eléctricos que son capaces de atenuar o amplificar para posteriormente enviarlos a otras neuronas. La unión de millones de estas crea una estructura neuronal que es la que nos permite pensar y actuar como seres inteligentes.

En la informática se ha intentado emular este comportamiento, creando lo que se conoce como redes neuronales artificiales. Las neuronas artificiales omiten muchas características que definen a las biológicas, como su localización espacial, los retardos en la propagación de señales o los mecanismos químicos de sodio y potasio que permiten la transmisión de potenciales eléctricos. Al final, un modelo es una representación de la realidad, y por tanto contiene información más limitada que esta. El modelo de integración y disparo define la base de la neurona que se utiliza habitualmente en Inteligencia Artificial:

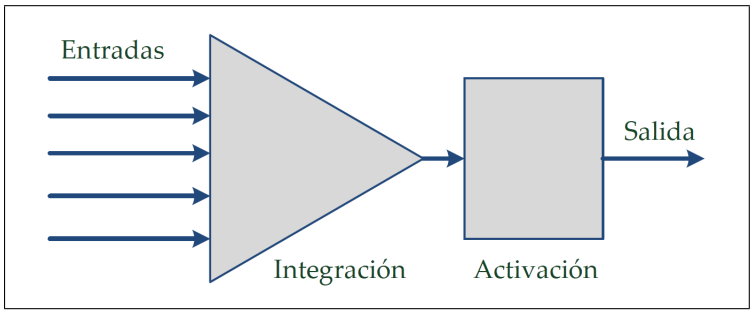


Figura 3.7: Modelo de integración y disparo en las neuronas artificiales [7]

En la fase de integración se unifican todas las señales x_i entrantes a la neurona j, cada una influyendo en mayor o menor medida según el peso w_{ij}, peso de la conexión de la neurona i hasta la neurona j:

$$z_j = b_j + \sum_{i=1}^{n} x_i w_{ij} \tag{3.1}$$

Además, en muchas ocasiones se añade la componente b_j que actúa como sesgo, permitiendo que la salida de la neurona se desplace a la izquierda o a la derecha en el eje X. Este sesgo también se puede representar de forma

implícita, de forma que i parte de 0 en vez de 1, siendo $x_0 = 1$ y $w_0 j = b_j$. [7]

Pero esta fase de integración no es suficiente para simular el comportamiento del cerebro humano, su resultado es lineal. Una red neuronal se genera encadenando capas de estas neuronas, donde la salida de las neuronas de una capa actúan de entrada a las neuronas de la siguiente capa. La combinación de funciones lineales es una función lineal, y por tanto el resultado final de la red seguiría siendo una función lineal de las entradas. En definitiva, tener n capas equivaldría a utilizar una única capa.

Es por ello que existe una segunda fase en este proceso, la fase de activación. Al resultado de integración en cada neurona se le aplica una función de activación no lineal (σ). A partir de ahora, nuestra red podrá aproximar funciones no lineales: cuantas más capas se utilicen en una red, con mayor precisión se podrá llevar a cabo esta aproximación.

$$f_j = \sigma(z_j) = \sigma(b_j + \sum_{i=1}^{n} x_i w_{ij}) \tag{3.2}$$

f_j es la salida final de la neurona. [7]

3.2.2. Redes neuronales convolutivas

Si hay una técnica del aprendizaje profundo que funcione especialmente bien en la práctica, son las redes neuronales convolutivas o convolutional neural networks (CNN). Estas se utilizan para procesar señales, como son las imágenes.

Una ventaja frente a las redes multicapa reside en que pueden trabajar con entradas y salidas estructuradas. Esto supone que, en vez de recibir un vector de entradas correspondientes a diferentes variables, pueden trabajar directamente con un vector 1D, una matriz 2D o con un tensor con múltiples dimensiones. [7]

En estas redes, las capas realizan la operación de convolución en vez de la multiplicación de las entradas por pesos. En el caso de imágenes (señal 2D), esta operación se define como:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n) K(i-m, j-n) \tag{3.3}$$

Donde I representa una imagen y K un kernel de dos dimensiones [5]. El tipo de convolución que describimos aquí es la discreta y por ello en la fórmula aparecen sumatorias, en vez de las integrales típicas de dominios

continuos. En la siguiente figura se puede observar un ejemplo de convolución:

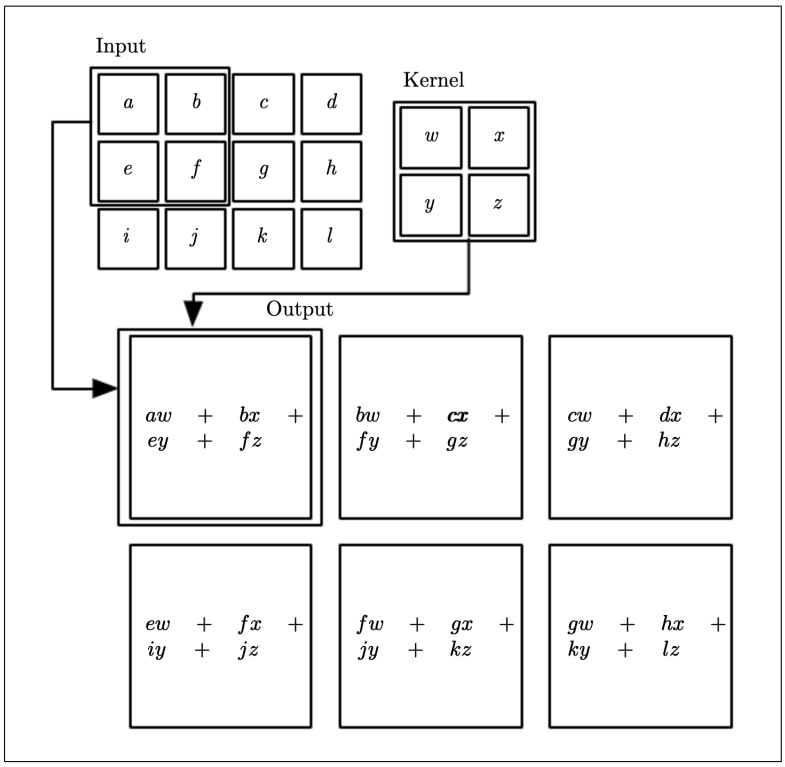


Figura 3.8: Resultado de convolucionar la entrada I con el kernel K [5]

El kernel se va desplazando por la imagen, solapándose sobre cierta área de esta. En cada una de las posiciones que toma genera un valor de salida, fruto de la multiplicación del valor de los píxeles de la imagen sobre los que se encuentra por los pesos del kernel. Todos estos valores de salida forman el resultado final, que mantiene la misma estructura 2D (aunque normalmente con un menor tamaño).

Así es como funcionan las redes neuronales convolutivas, donde cada capa de la red aplica este operador a la entrada que recibe. Con ellas:

- Se reduce el número de parámetros necesarios. Este depende del tamaño de los kernel utilizados, que suele ser muy inferior al de las imágenes de entrada.
- Se mantienen las relaciones espaciales/temporales de las señales. Muy beneficioso en análisis de imágenes, donde dos pixeles cercanos contendrán información similar. Los capas convolutivas son capaces de interpretar estas relaciones y mantenerlas para las siguientes fases del

modelo.

3.2.3. Autocodificadores

Los autocodificadores o autoencoders son un modelo de aprendizaje no supervisado que permite encontrar una representación latente no lineal para una distribución de datos dada.

Para comprenderlos correctamente, definamos primero lo que es una representación latente. Esta es una representación subyacente para una determinada distribución de datos, es decir, una forma "comprimida"de representar la distribución que siguen estos datos. En esta representación, se reduce el número de dimensiones necesarias para definir la distribución.

Existe un algoritmo conocido como análisis de componentes principales (Principal Component Analysis, PCA) que encuentra una transformación lineal de la representación original a otra latente. Pero tiene un problema, y es que como su definición indica esta transformación es lineal. [20]

Los autocodificadores realizan esta tarea pero encontrando una transformación no lineal, por lo que pueden trabajar con datos más complejos. Para ello constan de dos partes, el codificador y el decodificador:

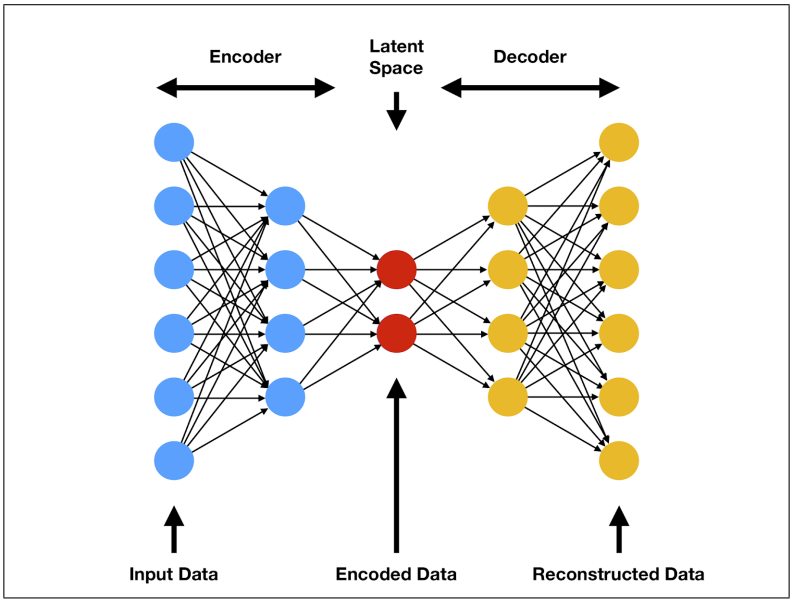


Figura 3.9: Secciones en las que se divide un autocodificador [20]

El objetivo tras el entrenamiento del modelo es que los datos reconstruidos por el decodificador sean lo más parecidos posible a los de entrada.

Un tipo especial de autocodificador es el autocodificador variacional (Variational Autoencoder, VAE). La principal diferencia viene dada por la estructura que sigue la representación latente, donde se fuerza al autocodificador a organizarla de forma que puntos de datos similares se encuentren cerca en la representación latente y puntos diferentes se encuentren separados. En un autocodificador regular esto no se exige, por lo que no podríamos desplazarnos por el espacio latente y observar cambios graduales que se corresponden con cambios graduales en la representación original. [20]

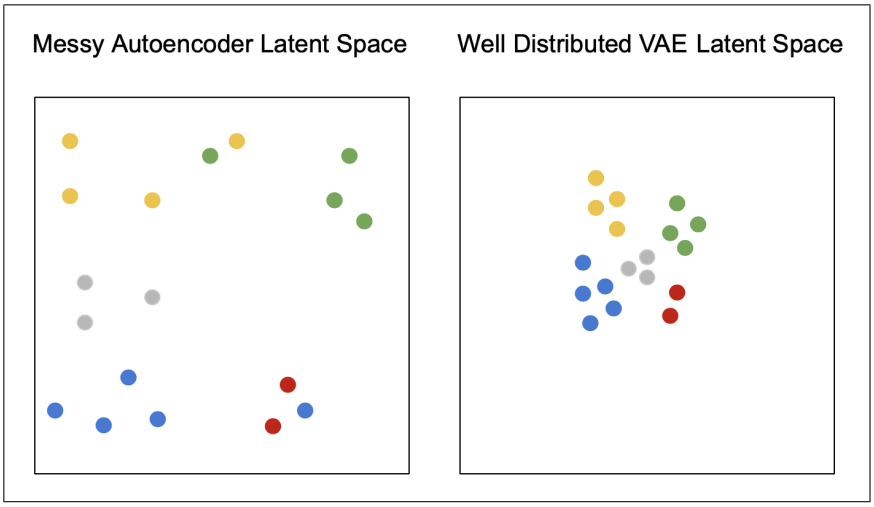


Figura 3.10: Espacio latente en un autocodificador regular vs en VAE [20]

Vector Quantized Variational Autoencoder

La representación latente por defecto en los VAE es continua. Pero existe una variante en la que se discretiza esta representación. Esto es útil cuando trabajamos con datos como imágenes: podría utilizarse una variable discreta para el color, otra para el tipo de objeto, su tamaño, etc. Muchos otros elementos del mundo real son susceptibles de representarse de forma discreta. Hay modelos como los transformers, que vamos a estudiar más adelante, que están diseñados para trabajar sobre datos con esta codificación.

Los Vector Quantized Variational Autoencoders (VQVAE) utilizan este tipo de codificación, donde la representación latente consiste en lo que se conoce como libro de código (codebook). Este codebook no deja de ser un conjunto de vectores discretos asociados a un índice.

Cuando el codificador recibe un dato, este se transforma a la representación latente. A continuación, se introduce al decodificador el vector del codebook con el que existe una menor distancia euclídea. Este vector es procesado por el decodificador devolviendo la salida final. Dicho esto, se podría

pensar que la potencia expresiva del autoencoder es muy baja, ya que el número de vectores del codebook es mucho más pequeño que el conjunto de posibles datos de entrada, y siempre que se introdujese el mismo vector al decodificador obtendríamos la misma salida.

Esto no es así, ya que normalmente no se utiliza un único vector: en datos como imágenes, es probable que se escoja un conjunto de vectores que se introducen en el decodificador. Estos son procesados conjuntamente y finalmente se devuelve la imagen sintética. De esta forma, el número de posibilidades crece enormemente. [20]

3.2.4. Redes generativas antagónicas

En inglés conocidas como Generative Adversarial Networks (GANs), son un tipo de modelos utilizados para generar datos sintéticos, que por ejemplo pueden servir para ampliar nuestro dataset. Contienen dos módulos que trabajan simultáneamente, cada uno de ellos intentando ser mejor que el otro. Estos dos módulos son el generador G y el discriminador D: el generador intenta crear datos sintéticos parecidos a los reales y el discriminador actuará de juez, tendrá que decidir si dado un dato este es real o sintético. Estamos en un juego en el que el generador intentará mejorar todo lo posible para generar imágenes indistinguibles a ojos del discriminador, desarrollándose en dos escenarios:

- Se muestrean datos x del conjunto de entrenamiento y se introducen en D. En este caso, D(x) será un valor cercano a 1, ya que las muestras son datos reales.
- Se introduce en G una muestra z generada aleatoriamente de acuerdo a una distribución de probabilidad, siendo G(z) una muestra sintética. D(G(z)) deberá tomar un valor cercano a 0, detectando que no se trata de una muestra real. Sin embargo, aunque el discriminador actúa para que D(G(z)) sea cercano a 0, el generador hace todo lo contrario y pretende que D(G(z)) se aproxime a 1.

El entrenamiento se lleva a cabo simultáneamente en G y D, actualizando los pesos de ambos en cada iteración. Este termina cuando se ha alcanzado un equilibrio en el que ni al generador ni al discriminador les interesa modificar sus pesos, llamado equilibrio de Nash.

Las GANs son muy populares actualmente y se pueden utilizar con diferentes tipos de datos, entre ellos las imágenes. Originalmente las GANs se presentaron utilizando capas completamente conectadas (fully connected layers), pero la técnica de las redes neuronales convolutivas también se aplicó a las GANs dando lugar a las Deep Convolutional GAN (DCGAN). [7]

3.2.5. Transformers

En los últimos años, se han desarrollado diferentes estrategias que permiten el modelado de secuencias. Algunas de ellas son las Redes Neuronales Recurrentes (RNN), los LSTM o modelos basados en redes convolutivas, como Wavenet. Aunque han dado buenos resultados en campos como la generación de texto, son mejorables, ya que las técnicas de memoria que implementan solo les permiten recordar los últimos datos de la secuencia. Así, en sucesiones de mayor longitud, no podrán mantener en memoria datos alejados del punto actual.

Para resolver este problema, los investigadores han creado lo que se conoce como mecanismos de atención. En ellos, el modelo es capaz de comprender qué datos de la secuencia son importantes para la tarea que estamos desarrollando. Estos datos son los que memoriza, descartando el resto. Funcionan de forma similar a nuestro cerebro: cuando vemos una imagen, aunque parece que estamos contemplando todos sus detalles, solo estamos prestando atención a fragmentos concretos de esta y obviando los detalles en el resto de áreas. Algunos autores aplicaron esta técnica a arquitecturas ya existentes.

Pero en 2017, Vaswani et al. [6] propone un modelo basado únicamente en mecanismos de atención, prescindiendo de técnicas como la recurrencia o las convoluciones: a esta arquitectura la llamó el Transformer. El modelo está formado por 6 codificadores y 6 decodificadores:

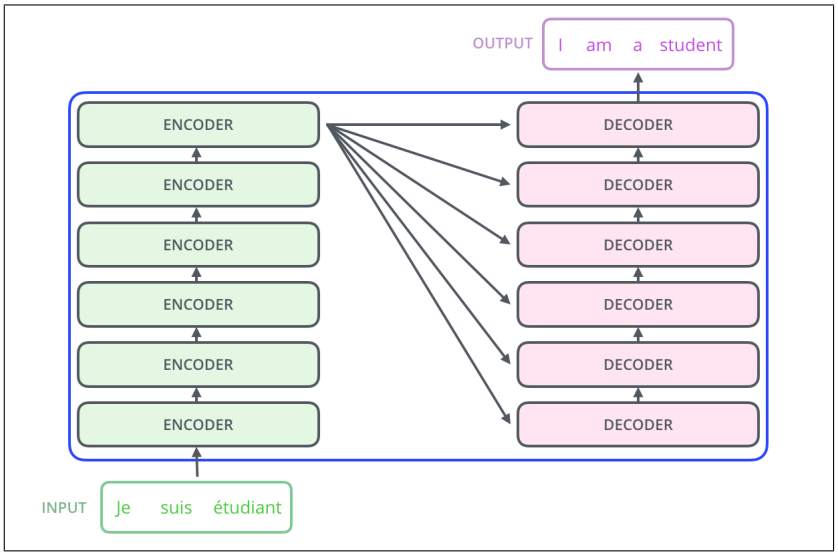
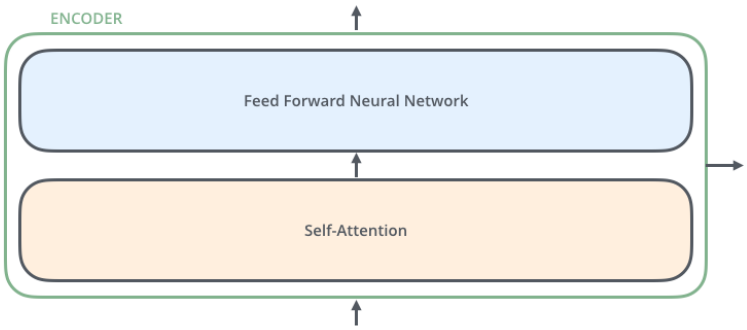


Figura 3.11: Transformer aplicado a la generación de texto [15]

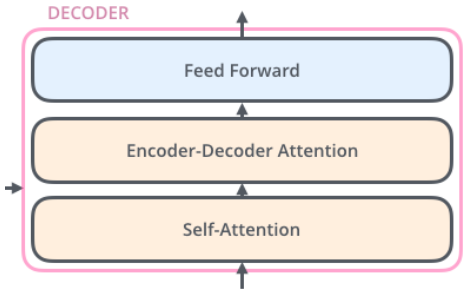
En cada codificador, las características de entrada primero pasan por una capa de auto-atención, en la que interactúan entre sí y se descubre

a cuáles hay que prestar mayor atención. La salida de esta primera capa serán los datos agregados resultantes de esas interacciones y las puntuaciones de atención. A continuación, se introduce esta salida en una red neuronal completamente conectada. [18]

Los decodificadores también poseen estas dos capas, pero entre ambas existe otra capa de atención que ayuda al decodificador a concentrarse en las características importantes de la entrada.



(a) Estructura interior del codificador



(b) Estructura exterior del decodificador

Figura 3.12: Los dos tipos de capas que forman el Transformer [15]

Los Transformer son actualmente los modelos más efectivos en tareas como la generación de lenguaje natural, y también se han aplicado a generación de imágenes. Estos son capaces de memorizar datos de cualquier punto de la secuencia, al contrario que otros modelos recurrentes o convolutivos, que sólo almacenan las interacciones locales. [15]

3.3. Bibliotecas y paquetes

En el estado del arte encontramos productos desarrollados por investigadores que nos van a facilitar cumplir nuestro objetivo.

3.3.1. Taming Transformers for High-Resolution Image Synthesis

Científicos de la Universidad de Heidelberg (Alemania) publicaron a finales de 2020 un modelo capaz de generar imágenes sintéticas de alta resolución utilizando transformers. En publicaciones previas ya se habían expuesto modelos generativos basados en transformers [8], pero estos solo podían funcionar con imágenes de pequeño tamaño. El coste computacional de un transformer aplicado directamente en imágenes crece cuadráticamente con el tamaño de esta, así que no es viable utilizarlos en imágenes de más de 100x100 píxeles, y mucho menos en aquellas del orden del megapixel.

P. Esser et al. [10] desarrolla un método capaz de combinar la potencia generativa de los transformers con la eficiencia que aportan las redes convolutivas.

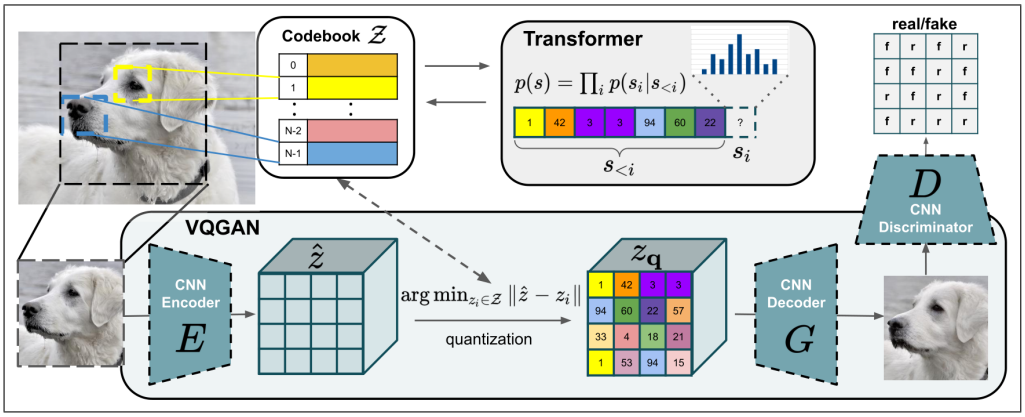


Figura 3.13: Modelo propuesto por P. Esser et al. [10]

La arquitectura convolutiva de la que hablamos es un Vector Quantized Generative Adversarial Network (VQGAN), una variante de VQVAE propuesta por los autores de la publicación donde la estructura codificador-decodificador del VAE se entrena mediante un procedimiento adversarial, utilizando un discriminador.

Tras entrenar el modelo, se habrá generado un codebook que comprime la información de las imágenes. Será sobre este codebook sobre el que se entrene el transformer que posteriormente nos permitirá sintetizar imágenes

sintéticas. Ahora tendremos un modelo que es capaz de aprovechar la gran capacidad generativa de los transformers, manteniendo tiempos de entrenamiento razonables gracias a VQGAN.

En la publicación original los autores muestran ejemplos de los resultados que se pueden obtener dada una imagen condicional de entrada:

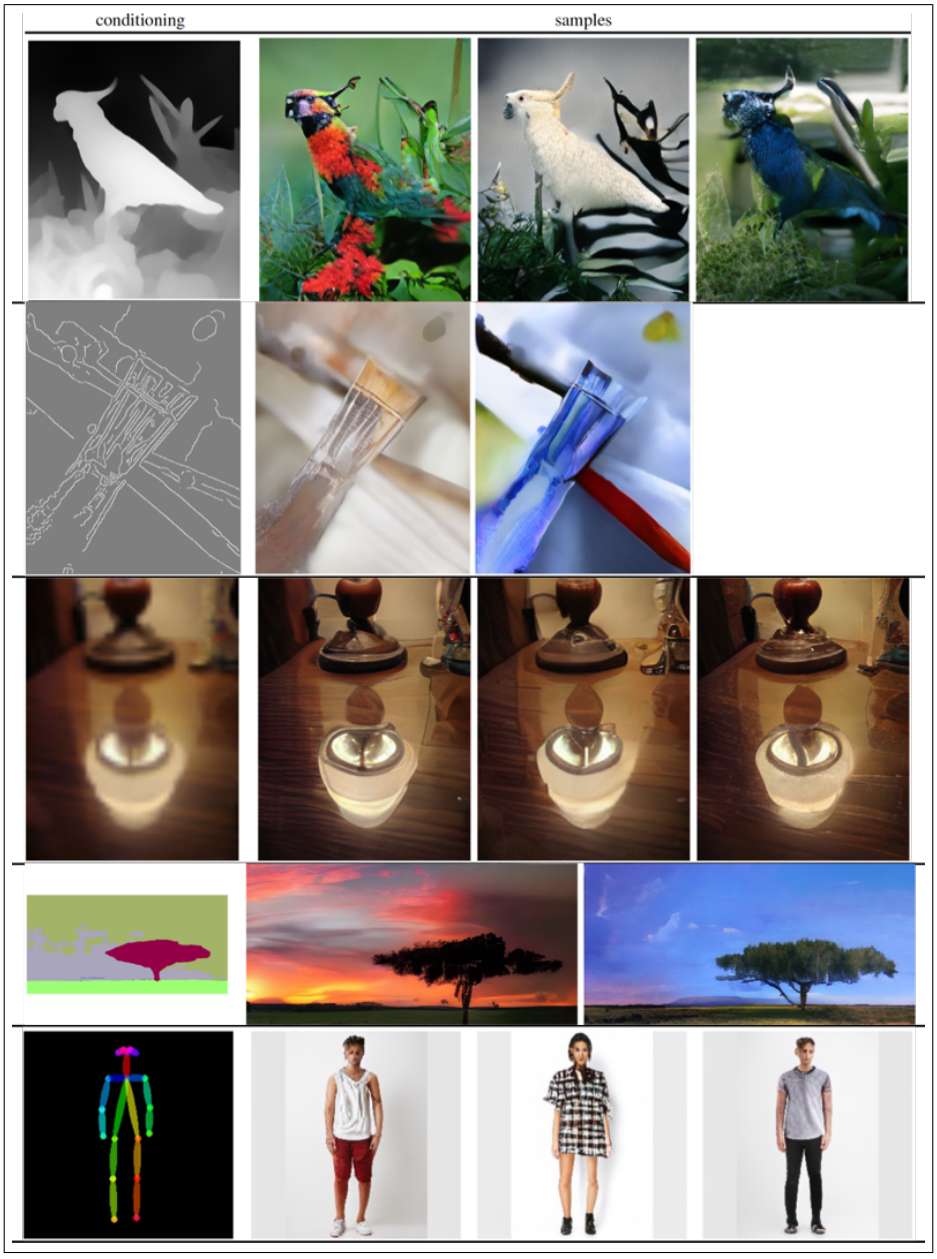


Figura 3.14: Ejemplos generados utilizando el modelo basado en transformer

Se observa que funciona correctamente en diversas situaciones, por lo que puede ser una buena opción para nuestra tarea de síntesis de imágenes.

3.3.2. Pytorch

PyTorch es una biblioteca basada en tensores optimizada para crear modelos de aprendizaje profundo que se entrenen y ejecuten tanto en GPUs como en CPUs.

Figura 3.15: Logo de PyTorch

Es adecuada para ser utilizada en proyectos de investigación, ya que convierte el desarrollar y experimentar con nuevas arquitecturas en algo relativamente sencillo. Su núcleo proporciona la capacidad de definir funciones matemáticas y calcular sus gradientes.

Al compararla con otras bibliotecas de aprendizaje profundo como Tensorflow encontramos diferencias. Esta sigue el paradigma definición-compilación-ejecución, mientras que PyTorch presenta una estructura definición-ejecución, donde no existe una etapa de compilación. El usuario puede definir una expresión matemática y calcular su gradiente directamente.

A continuación, vamos a comentar algunas características y módulos incluidos en esta biblioteca que merece la pena conocer.

Dataset y DataLoader

El código para gestionar los datos con los que vamos a trabajar se puede volver complejo y difícil de mantener. PyTorch proporciona los módulos *torch.utils.data.DataLoader* y *torch.utils.data.Dataset* para automatizar la carga y la manipulación de los datos, tanto procedentes de datasets ya existentes (por ejemplo, MNIST) como de datos propios.

torch.utils.data.Dataset es una clase heredada por una clase hija en la que se implementan ciertos métodos encargados de cargar los datos. Un da-

taset puede ser de tipo *iterable* o de tipo *map*: los primeros implementan $\_\_iter\_\_()$, y permiten que el objeto sea iterable, de forma que $iter(dataset)$ devuelve un flujo de datos que están siendo leídos de una base de datos o incluso se están produciendo en tiempo real. Por otra parte, un dataset de tipo *map* implementa los métodos $\_\_getitem\_\_()$ y $\_\_len\_\_()$. Esto permite establecer una relación clave-dato, donde si $CompoundDataset$ es un dataset con imágenes que pueden ser o no compuestos químicos, $CompoundDataset[idx]$ devolvería la imagen número idx junto a su etiqueta. Este último tipo de dataset es el que utilizamos nosotros.

El constructor de un objeto de tipo *torch.utils.data.DataLoader* recibe como principal argumento un objeto *torch.utils.data.Dataset*. También recibe otros parámetros, como el tamaño del batch o si queremos que los datos sean devueltos en un orden aleatorio. Devuelve un iterable que cumple estas características, lo que facilita en gran medida el entrenamiento de modelos.

Utilizando estos módulos separamos los datos del código de entrenamiento, obteniendo una mayor modularidad y un código más fácil de leer.

Módulos *torch.nn* y *torch.optim*

El primero contiene los principales componentes utilizados en la construcción de redes neuronales y otras arquitecturas de aprendizaje profundo. Algunos de estos son:

- **torch.nn.Module:** Es la clase padre de la que heredan todas las implementaciones de redes neuronales. En su constructor, entre otros elementos, se pueden inicializar diferentes capas que serán utilizadas en el método $forward()$, donde se especifica la estructura de la red y las conexiones entre ellas.

- **torch.nn.functional:** Recoge funciones de activación, de error así como capas de la red que no tienen un estado asociado.

- **torch.nn.Conv2d**, **torch.nn.MaxPool2d**, **torch.nn.Linear** son ejemplos de capas contenidas en este módulo. Existen diferentes tipos de capas lineales, convolutivas, recurrentes, transformer, etc. previamente implementadas y listas para ser utilizadas.

Por otro lado, *torch.optim* contiene optimizadores como son SGD o Adam, encargados de actualizar los parámetros del modelo durante el entrenamiento.

Autograd

Al entrenar redes neuronales, el método más utilizado para actualizar los parámetros del modelo es la propagación hacia atrás o back propagation. Estos se ajustan de acuerdo al gradiente de la función de error con respecto al parámetro dado.

PyTorch facilita la aplicación de la propagación hacia atrás a través del módulo *torch.autograd*. Cuando implementamos un modelo, solo es necesario declarar el método $forward()$, ya que PyTorch crea sobre la marcha el grafo de las operaciones necesarias para calcular el gradiente. Durante el entrenamiento del modelo, al llamar a $loss.backward()$ se calculará el gradiente utilizando este grafo.

Como conclusión, PyTorch aporta una gran flexibilidad para crear modelos de aprendizaje profundo, su sintaxis y paradigma de programación son relativamente fáciles de aprender y contiene un gran número de módulos que permiten crear modelos complejos. Aparte, es compatible con el entrenamiento en GPU y permite guardar los modelos entrenados. Por todas estas razones puede resultar una librería adecuada para implementar el clasificador de imágenes.

Capítulo 4

Diseño e implementación

En esta sección explicamos en detalle los productos que hemos desarrollado para los científicos Israelíes. Hemos partido de un dataset facilitado por ellos, que hemos utilizado para entrenar un modelo que genera imágenes sintéticas y para implementar un clasificador de imágenes.

Toda la implementación se encuentra en el repositorio de GitHub del proyecto [19]. La raíz del repositorio contiene los siguientes elementos:

```
/
├── datasets/
├── experiments/ ..... implementación del TFG
├── papers/ ..... algunas de las publicaciones utilizadas en su desarrollo
├── report/ ..... memoria en formato LaTeX
├── slides/
└── README.md
```

En el repositorio, el directorio datasets se encuentra vacío para no crear problemas con el sistema de control de versiones. En el README.md se indica un enlace donde se pueden descargar los datos que se deben colocar en el directorio.

4.1. Dataset utilizado

El dataset contiene ejemplos positivos y negativos. Todas las imágenes son diferentes, presentan elementos con distintos trazados de línea, tamaños, colores... Originalmente se encuentran en formato .png.

Los ejemplos positivos son imágenes de moléculas extraídas de publicaciones. La mayoría son moléculas completas, aunque algunas parecen recortes de estructuras más grandes. En total tenemos 162 imágenes de este tipo.

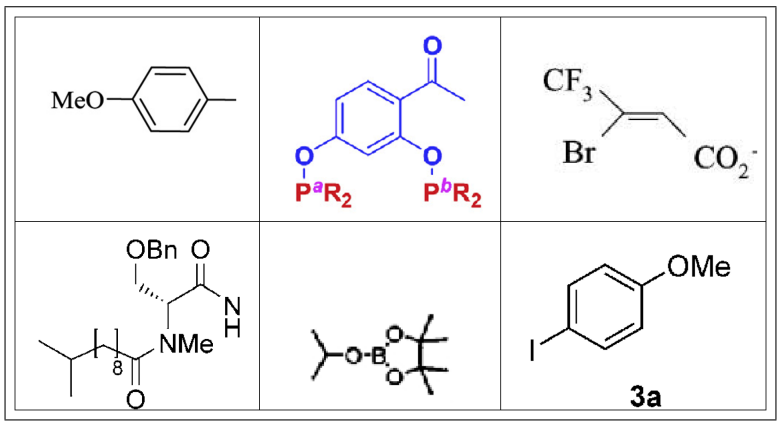


Figura 4.1: Ejemplos de muestras positivas del dataset

Los ejemplos negativos son, en cambio, imágenes que contienen rectas, curvas y otras figuras que se parecen a las formas que adquiere una molécula, pero no lo son. En esta categoría hay más imágenes, 800 en total.

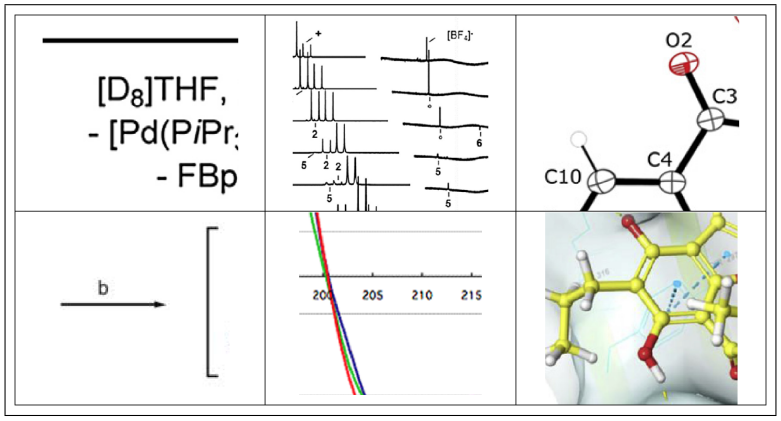


Figura 4.2: Ejemplos de muestras negativas del dataset

Estas imágenes necesitan un preprocesamiento, es necesario elegir el formato con el que se va a trabajar y dimensionarlas para que todas tengan el mismo tamaño. Debido a que el canal alfa del formato .png no se está aprovechando, decidimos convertirlas a formato .jpg y así trabajar con tres canales en nuestros modelos. Además, el modelo generador que utilizamos está diseñado para trabajar con imágenes con este número de canales, así que será lo más adecuado.

Lo más conveniente sería dimensionar todas las imágenes al tamaño de la más grande, así no se perdería información. Pero esto puede suponer un problema, ya que

También mencionar el limitado número de ejemplos positivos que tenemos en comparación con negativos: estamos ante un conjunto de datos desbalanceado. En el momento de entrenar el clasificador, tendremos que conseguir que esté balanceado, ya sea reduciendo el número de ejemplos negativos utilizado o aplicando *data augmentation* sobre los positivos. En este trabajo elegimos la segunda opción, ya que en Aprendizaje Automático cuantos más datos se utilizan en el entrenamiento, mejores modelos se obtienen y con mayor capacidad de generalización.

La técnica de *data augmentation* consiste en aumentar el tamaño del conjunto de datos completándolo con imágenes transformadas de este. Existen bibliotecas en lenguajes de programación como Python que facilitan en gran medida esta tarea, ya que permiten indicar, entre una serie de transformaciones ya implementadas, cuáles queremos aplicar. En nuestro caso vamos a crear tres *data augmentation* diferentes, y comprobaremos cuál funciona mejor en los experimentos. La primera realizará transformaciones suaves, la segunda algo más fuertes y la tercera bastante disruptivas.

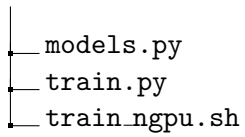
Algorithm 1 aug2: secuencia de data augmentation aplicada

Prueba

4.2. Estructura del código

Como se menciona al principio del capítulo, la implementación se sitúa en el directorio *experiments/*, dividido en dos subdirectorios. El primero contiene el código del modelo generador de imágenes, el segundo el del clasificador:

```
experiments/ ..... implementación del TFG
├── taming_transformers/ ..... generador de imágenes
│   ├── taming-transformers/
│   │   ├── configs/
│   │   ├── logs/
│   │   └── train_ngpu.sh
│   ├── data_generation_and_aug.ipynb
│   ├── sample_and_clean_molecules.ipynb
│   ├── sampling_experiment.ipynb
│   └── functions.py
└── image_classifier/ ..... clasificador de imágenes
    ├── saved_models/
    ├── datasets.py
    ├── functions.py
    └── grid_search.py
```



4.2.1. Generador de imágenes sintéticas

Uno de los objetivos del proyecto es construir un modelo que nos permita generar imágenes sintéticas de moléculas.

4.3. Ejecución y reproducibilidad

Como ejecutar los scripts y semillas utilizadas

Capítulo 5

Experimentación

Explicar las distintas ejecuciones que hemos llevado a cabo en el transformer, en el modelo de clasificación...

Capítulo 6

Conclusiones

Comentar la configuración final y si se han cumplido los objetivos del proyecto.

Bibliografía

[1] David Weininger. "SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules". *Journal of chemical information and computer sciences* 28.1 (1988), págs. 31-36.

[2] Ken Schwaber. "Scrum development process". *Business object design and implementation*. Springer, 1997, págs. 117-134.

[3] Thomas Engel. "Basic Overview of Chemoinformatics". *Journal of Chemical Information and Modeling* 46.6 (2006). PMID: 17125169, págs. 2267-2277. DOI: 10.1021/ci600234z. eprint: https://doi.org/10.1021/ci600234z. URL: https://doi.org/10.1021/ci600234z.

[4] Mihai Liviu Despa. "Comparative study on software development methodologies". *Database Systems Journal* 5.3 (2014), págs. 37-56.

[5] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[6] Ashish Vaswani y col. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[7] Fernando Berzal. *Redes neuronales & deep learning*. https://deeplearning.ikor.org. Independently published, 2018.

[8] Mark Chen y col. "Generative pretraining from pixels". *International Conference on Machine Learning*. PMLR. 2020, págs. 1691-1703.

[9] Mario Krenn y col. "Self-referencing embedded strings (SELFIES): A 100 % robust molecular string representation". *Machine Learning: Science and Technology* 1.4 (oct. de 2020), pág. 045024. DOI: 10.1088/2632-2153/aba947. URL: https://doi.org/10.1088%2F2632-2153%2Faba947.

[10] Patrick Esser, Robin Rombach y Bjorn Ommer. "Taming transformers for high-resolution image synthesis". *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, págs. 12873-12883.

Otras fuentes

[11] Robert Belford. *Anatomy of a MOL file*. URL: https://chem.libretexts.org/Courses/University_of_Arkansas_Little_Rock/ChemInformatics_(2017)%3A_Chem_4399_5399/2.2%3A_Chemical_Representations_on_Computer%3A_Part_II/2.2.2%3A_Anatomy_of_a_MOL_file (visitado 26-04-2022).

[12] curiosoando.com. *¿Qué es la aromaticidad?* URL: https://curiosoando.com/que-es-la-aromaticidad (visitado 25-04-2022).

[13] dias-laborables.es. *¿Cuántos días laborables en el año 2022?* URL: https://www.dias-laborables.es/cuantos_dias_laborables_en_ano_2022_Andaluc%C3%ADa.htm (visitado 25-05-2022).

[14] Universidad Europea. *Cuánto gana un ingeniero informático*. URL: https://universidadeuropea.com/blog/cuanto-gana-un-ingeniero-informatico/ (visitado 25-05-2022).

[15] Giuliano Giacaglia. *How Transformers Work*. URL: https://towardsdatascience.com/transformers-141e32e69591 (visitado 04-06-2022).

[16] Rocío González. *¡Comienza el verano! Cuántos días de vacaciones me corresponden*. URL: https://www.sage.com/es-es/blog/dias-vacaciones/ (visitado 25-05-2022).

[17] National Institute of Health (NIH). *PubChem Docs - About PubChem*. URL: https://pubchemdocs.ncbi.nlm.nih.gov/about (visitado 26-04-2022).

[18] Raimi Karim. *Illustrated: Self-Attention*. URL: https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a (visitado 04-06-2022).

[19] Pedro Bedmar López. *TFG GitHub repository*. URL: https://github.com/pbedmar/tfg (visitado 03-06-2022).

[20] Charlie Snell. *Understanding VQ-VAE (DALL-E Explained Pt. 1)*. URL: https://ml.berkeley.edu/blog/posts/vq-vae/ (visitado 04-06-2022).

[21] Santander Universidades. *Metodologías de desarrollo de software: ¿qué son?* URL: https://www.becas-santander.com/es/blog/metodologias-desarrollo-software.html (visitado 04-03-2022).

[22] Aditya Virani y col. *Structural Representations of Organic Compounds.* URL: https://brilliant.org/wiki/structural-representations-of-organic-compounds/ (visitado 21-04-2022).

[23] Wikipedia. *Aromaticidad.* URL: https://es.wikipedia.org/wiki/Aromaticidad (visitado 25-04-2022).

[24] Wikipedia. *Compuesto orgánico.* URL: https://es.wikipedia.org/wiki/Compuesto_org%C3%A1nico (visitado 21-04-2022).

[25] Wikipedia. *Compuesto organometálico.* URL: https://es.wikipedia.org/wiki/Compuesto_organomet%C3%A1lico (visitado 21-04-2022).

[26] Wikipedia. *Estereoquímica.* URL: https://es.wikipedia.org/wiki/Estereoqu%C3%ADmica (visitado 21-04-2022).

[27] Wikipedia. *Fórmula estructural.* URL: https://es.wikipedia.org/wiki/F%C3%B3rmula_estructural (visitado 21-04-2022).

[28] Wikipedia. *Simplified molecular-input line-entry system.* URL: https://en.wikipedia.org/wiki/Simplified_molecular-input_line-entry_system (visitado 24-04-2022).