Universidad Nacional del Centro de la Provincia de Buenos Aires

Proyecto Final

Desarrollo de una plataforma web asistida por inteligencia artificial para descarga masiva de imágenes

Autor:
Franco Javier Arellano

Supervisores:
Dr. José Ignacio Orlando
Ing. Lucas Gabriel Telesco

23 de febrero de 2024

UNIVERSIDAD NACIONAL DEL CENTRO DE LA PROVINCIA DE BUENOS AIRES

Resumen

Facultad de Ciencias Exactas

Ingeniería de Sistemas

Desarrollo de una plataforma web asistida por inteligencia artificial para descarga masiva de imágenes

por Franco Javier ARELLANO

La mejora en la eficacia de los algoritmos de aprendizaje profundo ha despertado un interés generalizado en esta tecnología. Uno de los campos donde ha tenido un impacto más significativo es la medicina, donde la capacidad de estos algoritmos para reconocer patrones se ha explotado con éxito para analizar imágenes de patologías complejas y proporcionar diagnósticos precisos.

Para obtener buenos resultados, los algoritmos de aprendizaje profundo requieren ser entrenados con una gran cantidad de datos. Esto suele ser dificultoso en medicina, por ejemplo, donde la obtención de los estudios implica colaborar con diversos profesionales y pacientes y hacer uso de equipamiento complejo y costoso.

En este trabajo se presenta el desarrollo de una plataforma diseñada para obtener imágenes mediante la extracción automática de la web, conocida como *web scraping*. Nuestro objetivo principal es recolectar un número importante de imágenes obtenidas a través del motor de búsqueda de Google Images para una consulta específica, y almacenarlas en un único repositorio para su posterior procesamiento y análisis. La solución propuesta, desarrollada íntegramente en Python, utiliza la biblioteca Selenium para interactuar con las interfaces web y realizar el *scraping* del contenido de forma eficiente. Se plantearon dos implementaciones, una secuencial y otra concurrente, y se analizaron los tiempos de descarga de cada una. Además, se propuso la integración de un módulo de detección de anomalías para el filtrado de imágenes no adecuadas al dominio esperado, a través de una API REST. A partir de esta incorporación, se plantearon dos alternativas adicionales, una basada en una regla heurística que prioriza búsquedas y recorridos en imágenes similares a las esperadas, y otra que poda las descargas de aquellas imágenes detectadas como anómalas.

El algoritmo propuesto demostró ser capaz de obtener un gran número de imágenes en tiempos cortos, lo que favorecerá su aplicación para recolectar datos para entrenamiento de algoritmos auto-supervisados. Asimismo, la implementación concurrente combinada con la estrategia de poda mostró ser lo suficientemente robusta como para recolectar imágenes de utilidad para las aplicaciones posteriores.

Agradecimientos

Agradezco a Dios, a mi familia y a mis amigos por su apoyo, no solo durante la elaboración de esta tesis, sino a lo largo de mis años de estudio.

Asimismo, doy gracias a mis directores José Ignacio Orlando y Lucas Gabriel Telesco por su constante guía y apoyo a lo largo de todo el proceso de aprendizaje que culminó en la realización de esta tesis de grado. Además, agradezco a todos los investigadores y estudiantes de doctorado del Instituto de Investigación Pladema, quienes estuvieron siempre dispuestos a brindarme su ayuda y resolver mis dudas, contribuyendo de manera significativa al desarrollo de este trabajo académico.

Índice general

Agradecimientos III

1. Introducción 1
- 1.1. Descripción del problema 3
- 1.2. Vinculación con contenidos de la carrera 5
- 1.3. Organización del trabajo 6

2. Marco teórico 8
- 2.1. Los datos en la era del aprendizaje profundo 8
- 2.2. Las páginas web como proveedoras de información 9
- 2.3. *Scraping* web para recolección de datos 11
  - 2.3.1. Historia del *web scraping* 11
  - 2.3.2. Proceso de *web scraping* 13
  - 2.3.3. Legalidad 14
- 2.4. *Web scraping* para la recolección de imágenes 15
  - 2.4.1. Herramientas para *web scraping* de imágenes 16
    - APIs de buscadores 16
    - Plataformas de descarga de imágenes gratuitas 16
    - Herramientas programáticas 17
- 2.5. Filtrado de imágenes incorrectas con aprendizaje automático . 19

3. Diseño de la solución 20
- 3.1. Descarga de imágenes en Google Images 20
- 3.2. Modelado de la solución 23
- 3.3. Diseño de solución secuencial *naive* 24
  - 3.3.1. Estructura para comunicación entre módulos 25
  - 3.3.2. El módulo Productor 26
    - Extracción de las imágenes 28
  - 3.3.3. El módulo Consumidor 30
  - 3.3.4. El módulo Filtro 31
  - 3.3.5. Control de errores 32
  - 3.3.6. Salidas del *web scrapper* 34

3.4. Diseño de solución con procesamiento concurrente 36
3.5. Diseño de solución con heurística de expansión 38
3.6. Diseño de solución con poda de expansiones 39
3.7. Detalles de implementación 40

4. Resultados **43**
4.1. Análisis de costo computacional de los distintos módulos . . . 43
4.2. Comparación del enfoque secuencial con el *multithreading* . . . 46
4.3. Análisis de grado de anomalía por imagen recorrida 47
4.4. Influencia de la consulta sobre los resultados 49
4.5. Evaluación de las estrategias de heurística y poda 50
4.6. Influencia del número de *threads* en los resultados 54

5. Conclusiones **56**

Bibliografía **58**

Índice de figuras

2.1. Estructura de código HTML 10
2.2. Fases de trabajo de un sistema de *web scraping*. 14

3.1. Ejemplo de una búsqueda de imágenes en Google Images, para la consulta `gato`. 21
3.2. Nueva sección que se abre en la página luego de clickear en un thumbnail. 22
3.3. Página obtenida luego de clickear el botón "ver más" y que contiene las imágenes relacionadas. 23
3.4. Representación en forma de árbol del proceso de recorrido de las búsquedas de imágenes. 24
3.5. Módulos principales con los que se plantea resolver el problema. 25
3.6. Interacción de Selenium con el navegador web. 27
3.7. Botón "Mostrar más resultados" ubicado al pie de la búsqueda en Google IMages. 29
3.8. Diagrama de actividades del Productor representando las diferentes tareas que realiza. 30
3.9. Comportamiento del Consumidor e interacción con las diferentes librerías y componentes necesarios. 31
3.10. Representación esquemática del flujo de nodos a lo largo del pipeline de descarga. Los círculos verdes representan nodos que se procesan correctamente y los rojos corresponden a nodos con fallos. 33
3.11. Representación del árbol de nodos en el archivo de salida. . . 35
3.12. Diseño concurrente mediante colas (*Queue*) de Python. 37
3.13. Ejemplo de búsqueda de fotos de fondo de ojo en Google Images. 38
3.14. Distintos tipos de recorrido del árbol de nodos 40
3.15. Diagrama de las clases implementadas y las relaciones entre sí. 41
3.16. Ejemplo del archivo .csv de salida. 42
3.17. Ejemplo del archivo .csv de salida con nodos expandidos . . . 42

4.1. Ejemplo de imágenes de fondo de ojo y de gatos, diferenciando entre el tipo de imagen que se busca y los posibles resultados erróneos arrojados por el motor de búsqueda. 44
4.2. Boxplots comparando la distribución de tiempos de procesamiento por nodo de cada módulo, para 3.000, 5.000 y 7.000 imágenes descargadas. 45
4.3. Comparación de la eficiencia temporal entre la versión secuencial y la versión multithreading 46
4.4. KDE (Kernel Density Estimation) representando la frecuencia relativa de imágenes descargadas según su valor de *scoring function*. 48
4.5. Distribución de los valores de *scoring function* según el tipo particular de búsqueda realizada. 49
4.6. Gráficos de barra comparando la cantidad de imágenes descargadas en un lapso de 90 minutos para cada consulta, según el tipo particular de estrategia de búsqueda. 51
4.7. Cantidad de imágenes repetidas que fueron descartadas para cada estrategia utilizada. 52
4.8. Distribución de valores de *scoring function* en los conjuntos descargados para cada consulta, según la estrategia aplicada. . 54
4.9. Relación entre la cantidad de imágenes exitosas y la cantidad de threads que se utilizan sobre el módulo Productor 55

Capítulo 1

Introducción

Durante los últimos años, hemos sido testigos de un sorprendente crecimiento en el uso de los algoritmos de aprendizaje profundo, en gran parte debido a los increíbles resultados que proporcionan, siendo capaces de resolver tareas que antes se pensaban imposibles. ChatGPT [24], DALL·E-2 [25] y Midjourney [36] son algunos de los ejemplos más conocidos, ya que pueden procesar lenguaje natural, generar imágenes realistas y/o comprender contextos complejos con una habilidad nunca antes vista. Estos modelos están transformando la forma en que interactuamos con la tecnología, abriendo nuevas posibilidades creativas y permitiendo a los usuarios resolver problemas más complejos. Su impresionante capacidad para aprender y adaptarse a partir de datos promete un futuro emocionante para la inteligencia artificial.

La fase de entrenamiento de estos modelos implica el uso de grandes volúmenes de datos sin etiquetas, haciendo uso de técnicas de aprendizaje auto-supervisado (*self-supervised learning*) [39]. Esto presenta la ventaja de que se pueden aprovechar diversas bases de datos sin necesidad de etiquetar sus muestras previamente, lo que simplifica considerablemente el proceso de entrenamiento. No obstante, la precisión de este tipo de algoritmos sigue dependiendo de la disponibilidad de un gran volumen de datos representativos [10]. Este aspecto suele ser problemático en aplicaciones específicas tales como la medicina, donde los costos de captura, la complejidad de la aparatología necesaria, o incluso el grado de especificidad de una determinada patología complejizan la acumulación de grandes volúmenes de datos [13].

Al mismo tiempo, la web constituye en sí misma un banco prácticamente insondable de información, donde millones de páginas se entrelazan unas con otras disponibilizando a la vez cantidades ingentes de texto, imágenes, sonido y demás contenido multimedia. Desde motores de búsqueda como Google o Bing es posible conectar y relacionar todas estas páginas para lograr acceder a ellas a partir de una consulta o *query*. Esto convierte a los buscadores en candidatos ideales para la recolección masiva de muestras para

entrenamiento de algoritmos de aprendizaje profundo, aunque realizar esta tarea de forma manual es un proceso tedioso y lento.

Para abordar esta cuestión, las empresas detrás de estos motores de búsqueda ofrecen interfaces abiertas o APIs que permiten automatizar el proceso, y por ende mejorar la eficiencia en la obtención de los datos [1]. De este modo, basta con proporcionar una consulta específica a través de la API, y el motor de búsqueda devolverá por ejemplo un conjunto de imágenes relacionadas con la consulta. Sin embargo, estas plataformas suelen requerir una suscripción mensual costosa, del orden de los 100 dólares aproximadamente. Como alternativa, muchas organizaciones han desarrollado sus propios motores de *web scraping*, que emplean *scripts* personalizados para analizar el código de un sitio web y extraer información relevante. Aunque estos enfoques demandan una inversión considerable en desarrollo, permiten la personalización de la búsqueda de tipos de datos específicos, y ofrecen costos significativamente inferiores en comparación con las herramientas comerciales.

El propósito principal de este trabajo final es desarrollar una plataforma de *web scraping* con la capacidad de extraer un gran volumen de imágenes a partir de consultas específicas, que puedan posteriormente utilizarse para alimentar algoritmos de aprendizaje profundo. Específicamente, se persigue la creación de una plataforma altamente personalizable que permita a los usuarios elegir la cantidad de imágenes que desean descargar y proponer diversas consultas como entrada.

Para asegurar la calidad de los datos extraídos, se plantea la integración de un algoritmo de detección de anomalías en esta plataforma, que permita filtrar aquellos resultados que no se ajustan a las expectativas de la búsqueda. Esto tiene como objetivo reducir el riesgo de descargar datos incorrectos y minimizar a la vez posibles conflictos en los algoritmos que posteriormente se entrenen con estos datos.

El capítulo actual brinda una introducción general al documento y a la problemática abordada. En particular, en la Sección 1.1 describiremos en detalle el objetivo del trabajo, mientras que en la Sección 1.2 se recorren los contenidos de la carrera vinculados con el trabajo realizado, y la Sección 1.3 brinda una guía general sobre la organización del trabajo final.

[1] `https://serpapi.com/google-images-api`

1.1. Descripción del problema

Los algoritmos de aprendizaje profundo se han convertido en una herramienta fundamental para la automatización de diversas tareas. Su capacidad única de ajustarse a partir de datos les permite abordar tareas de reconocimiento de patrones complejos con un alto nivel de eficacia [10]. Esta habilidad se logra durante la etapa de entrenamiento, en la cual el algoritmo optimiza su propio rendimiento a partir de la minimización de una función de costo [10]. Sin embargo, para asegurar su capacidad de generalización es importante contar con un extenso conjunto de datos, que permita aprender un mayor número de patrones y reducir así el riesgo de sobre-entrenamiento u *overfitting* [10].

Esta necesidad plantea a la vez un desafío significativo en campos más especializados y críticos, como la medicina [23]. La aplicación de algoritmos de aprendizaje profundo en el análisis de imágenes médicas y su integración en el ámbito clínico pueden tener un impacto substancial en la eficiencia y eficacia de los hospitales y centros de salud [13]. Sin embargo, la obtención de un gran volumen de imágenes, como radiografías, tomografías y resonancias magnéticas, a menudo implica una coordinación compleja con múltiples profesionales de la salud y pacientes.

Es esencial subrayar la relevancia crítica de la precisión en este ámbito, ya que un diagnóstico erróneo puede tener consecuencias graves. En este sentido, entonces, los márgenes de error en estos algoritmos deben mantenerse en niveles cercanos a cero, un objetivo que sólo se puede lograr mediante una fase de entrenamiento minuciosa, bien estructurada y realizada sobre un volumen de datos significativo y diverso [13].

Como se explicitó en la sección anterior, el objetivo de este trabajo es introducir un nuevo enfoque para la descarga masiva de datos, en este caso imágenes, que facilite el entrenamiento de modelos de aprendizaje profundo. Existen servicios que permiten la descarga automática de imágenes a partir de una consulta brindada por el usuario. Entre estos se incluyen la Google Search API [2], la Bing Search API [3], Pixabay [4], Unsplash [5] y Pexels [6], entre otras. Estas APIs ofrecen a los usuarios un acceso rápido a una cantidad considerable de imágenes, de forma muy sencilla. De particular relevancia son

[2] https://serpapi.com/google-images-api
[3] https://www.microsoft.com/en-us/bing/apis/bing-image-search-api
[4] pixabay.com
[5] unsplash.com
[6] pexels.com

las dos primeras, Google y Bing, ya que al ser APIs de motores de búsqueda tienen acceso a una vasta cantidad de información obtenida a través del análisis y la indexación de millones de páginas web. Sin embargo, es importante señalar que estas APIs suelen tener limitaciones en sus versiones gratuitas, por lo que para acceder a todas sus funcionalidades se requiere una suscripción de pago. Además, un desafío inherente es que la cantidad de imágenes proporcionadas por consulta es limitada, y puede resultar insuficiente, lo que implica la necesidad de realizar múltiples consultas para obtener una colección extensa de imágenes.

Por otro lado, las APIs que no dependen de motores de búsqueda, como Pixabay, Unsplash y Pexels, pueden presentar limitaciones en cuanto a la variedad y disponibilidad de imágenes, ya que sólo ofrecen contenido almacenado en su propia base de datos, lo que puede dar lugar a búsquedas sin resultados para ciertos temas.

El *web scraping* es una técnica esencial para la extracción de datos alojados en páginas web. En los últimos años, ha ganado un protagonismo significativo debido a su capacidad para automatizar la recopilación y el análisis de grandes conjuntos de datos de internet, evitando la necesidad de intervención manual. Los *web scrapers* logran esto explorando el código HTML de las páginas web para identificar y seleccionar la información de interés [20]. El web scraping se usa, por ejemplo, en los motores de búsqueda, donde sistemas informáticos analizan exhaustivamente el contenido de numerosas páginas web, entrelazando y relacionando datos para proporcionar resultados relevantes a los usuarios [14]. Otra aplicación común son aquellas páginas que funcionan como resúmenes de varias fuentes, por ejemplo aquellas que reúnen y comparan precios de hoteles o recopilan ofertas de tiendas de ropa en línea [33].

Diversas herramientas han sido desarrolladas para facilitar la implementación de *web scrapers*, entre las que destacan BeautifulSoup [31], Scrapy [17] y Selenium [2]. Éstas automatizan parte del procesamiento necesario para interactuar con las páginas web y extraer información de ellas. Cada una de ellas posee características propias, así como ventajas y desventajas. Por lo tanto, al abordar un problema específico, resulta esencial evaluarlas minuciosamente para tomar la decisión más apropiada [20]. El factor más determinante en la elección de la herramienta adecuada radica en el tipo de página web que se pretende analizar, ya que cada herramienta está diseñada para abordar contextos específicos. Por lo tanto, la selección de la más apropiada deberá basarse en una comprensión sólida de las características de la página

web en cuestión.

Lo que se propone a lo largo de este trabajo, es el desarrollo de una plataforma de *web scraping* que facilite al usuario la creación de una base de datos de imágenes relacionadas a una consulta, para su posterior uso en el entrenamiento de un algoritmo de aprendizaje profundo. Se utilizarán las herramientas adecuadas para este problema en particular y se detallarán sus ventajas y desventajas en relación a otras. El diseño será evaluado en términos de su rendimiento computacional y la capacidad para recuperar imágenes correctas. Además, se realizará una evaluación para un caso de uso específico, vinculado con la descarga masiva de fotografías de fondo de ojo.

Este trabajo final se desarrolló como parte de una Beca de Entrenamiento a la Investigación de la Comisión de Investigaciones Científicas de la Provincia de Buenos Aires, en el Grupo Yatiris del Instituto PLADEMA, y como parte de un proyecto de investigación más amplio que busca desarrollar una serie de algoritmos de inteligencia artificial para diagnóstico de enfermedades oftalmológicas.

1.2. Vinculación con contenidos de la carrera

A lo largo del desarrollo de esta tesis, se aplicaron conocimientos adquiridos a lo largo de las diversas materias de la carrera de Ingeniería en Sistemas de la Facultad de Ciencias Exactas de la UNICEN. Estos conceptos se revelaron esenciales al momento de tomar decisiones relacionadas con el desarrollo y la evaluación del proyecto. A continuación, se mencionan algunas de las materias, así como los conceptos específicos que contribuyeron al enfoque y ejecución de este trabajo:

- **Metodologías de desarrollo de software:** Se emplearon diagramas UML para estructurar y documentar parte de la solución propuesta, tanto de forma informal (durante la esquematización de la solución) como en el propio documento de la tesis.

- **Análisis y diseño de algoritmos:** Se aplicaron conceptos de heurísticas y podas, así como diversas formas de recorrer grafos y árboles, fundamentales para el diseño de los algoritmos implementados.

- **Probabilidades y estadística:** Se utilizaron conocimientos estadísticos para analizar resultados y tomar decisiones basadas en la confiabilidad de los datos obtenidos.

- **Programación orientada a objetos:** La implementación del sistema se basó en principios de programación orientada a objetos, mejorando la modularidad y mantenibilidad del código.
- **Estructuras de almacenamiento de datos:** Se exploraron diversas formas de almacenar información, optimizando la gestión de datos dentro del sistema.
- **Arquitectura de computadoras:** Para la evaluación del rendimiento de la solución, se utilizaron conceptos de paralelismo.
- **Sistemas Operativos:** Se incorporó el concepto de multithreading para optimizar el rendimiento del sistema, aprovechando eficientemente los recursos disponibles.
- **Comunicación de datos:** Se aplicaron conocimientos sobre cómo se transporta la información, especialmente a través del protocolo HTTP, esencial para las interacciones en la web.
- **Diseño de sistemas de software:** Se utilizaron patrones de diseño para planificar y estructurar la solución de manera eficaz y escalable.

La integración de estos conocimientos provenientes de diversas áreas de la carrera ha sido clave para el éxito y la coherencia en todas las etapas de este proyecto.

1.3. Organización del trabajo

Este trabajo final está organizado en una serie de capítulos que abordan distintas etapas que se llevaron a cabo a lo largo del proceso de desarrollo.

El Capítulo 2 establece las bases teóricas esenciales para comprender el desarrollo del *web scraper*, desde sus inicios hasta las herramientas contemporáneas. Explora la evolución del *web scraping* y su importancia en la recolección de datos, especialmente en el contexto del entrenamiento de algoritmos de aprendizaje profundo.

En el Capítulo 3, se detalla el diseño propuesto para abordar el problema específico planteado. Se argumenta la idoneidad de este diseño y se presentan posibles mejoras para optimizar el rendimiento del programa. Se brindan además detalles acerca de la implementación realizada.

El Capítulo 4 ofrece una visión detallada de las pruebas realizadas, brindando una comprensión profunda del comportamiento del *web scraper* en

diversas situaciones. Aquí, se ponen a prueba las estrategias propuestas en el capítulo anterior para evaluar su impacto en el rendimiento del programa.

Finalmente, en el Capítulo 5 se abordan las conclusiones finales a las que se arribó tras culminar el trabajo, y se detallan algunos potenciales trabajos futuros.

Capítulo 2

Marco teórico

2.1. Los datos en la era del aprendizaje profundo

En la actualidad se vive una revolución en lo que refiere al uso de la inteligencia artificial. Numerosas aplicaciones, desde la medicina [23] hasta la navegación autónoma de automóviles [9], están viéndose beneficiadas por el uso de este tipo de algoritmos. Los mismos se basan en redes neuronales artificiales, un tipo particular de modelo que intenta reproducir el comportamiento de las neuronas de nuestro cerebro y cómo éstas intercambian información entre sí para descubrir patrones en las entradas [8]. En contraposición con el aprendizaje automático tradicional, que requería idear formas de computar características a partir de datos crudos, los modelos de aprendizaje profundo logran reconocer por si mismos las características intrínsecas de los datos que más sirven para resolver una tarea puntual. Esta capacidad ha motivado su aplicación en áreas como asistencia sanitaria [23], detección de fraude [30], reconocimiento de rostros [1], generación automática de texto [11], entre otras. Sin embargo, esto se logra al costo de utilizar un gran volumen de datos anotados, algo que a menudo es prohibitivo en ciertos contextos, como la medicina, por ejemplo, donde conseguir una imagen médica de, por caso, una patología compleja, puede requerir un gran número de interacciones con profesionales y pacientes.

Las empresas líderes en tecnología están hoy día a la vanguardia de la innovación en inteligencia artificial con aprendizaje profundo. Debido a sus infraestructuras pre-existentes de *Big Data*, tienen acceso a cantidades ingentes de datos que les han permitido innovar en áreas tales como la predicción de tendencias y necesidades [12] o la oferta de respuestas altamente personalizadas [28], por citar algunas. Su capacidad de entrenar algoritmos de aprendizaje profundo a gran escala les permite desarrollar métodos de una eficacia sorprendente. Un ejemplo de esto es la compañía Meta, que a lo largo de los años ha acumulado una cantidad monumental de información sobre

los usuarios de Facebook e Instagram, que les permite personalizar anuncios y aumentar la eficacia de sus campañas publicitarias [29]. La gran precisión de sus resultados permite a estas compañías además ejercer una influencia significativa en la evolución de la disciplina del aprendizaje profundo, a menudo sin compartir información sobre sus algoritmos y las bases de datos utilizadas para su entrenamiento. En contraposición, la comunidad científica cuenta con dificultades para acceder al mismo poder de cómputo e información que estas empresas, lo que complejiza hoy día la capacidad de masivizar la implementación de nuevos enfoques novedosos de una forma transparente y segura [38].

2.2. Las páginas web como proveedoras de información

Cuando buscamos una fuente a partir de la cual recopilar grandes cantidades de información, Internet es el primer recurso que se nos viene a la mente. En la actualidad, la cantidad de datos que fluye a través de la red asciende a miles de millones de gigabytes diarios [1], disponibilizados mediante sitios web.

Una página web es un documento digital que aloja información en diversos formatos, incluyendo imágenes, texto y videos. Por lo general, los sitios se alojan en servidores, y un usuario accede a ellos a través de una URL (sigla de *Uniform Resource Locator*), que representa la dirección unívoca de un recurso web. Un servidor de nombres de dominio (o DNS), traduce la URL en la dirección IP del servidor que aloja el sitio. Ante la solicitud de acceso, el servidor responde entregando los archivos que constituyen la página web correspondiente, y el navegador interpreta y muestra el contenido de estos archivos para ser accedidos por el usuario. Para ello, los archivos contienen información sobre su estructura, presentación y comportamiento [34].

El código que está contenido en estos archivos se define principalmente mediante el lenguaje HTML (sigla de *Hypertext Markup Language*), que permite organizar y presentar el contenido, marcando los elementos para que se muestren y se comporten de manera específica. Los elementos en el código HTML constan de etiquetas, contenido y atributos que definen su comportamiento. La comprensión de estos componentes es fundamental para el diseño

[1] https://seedscientific.com/how-much-data-is-created-every-day/

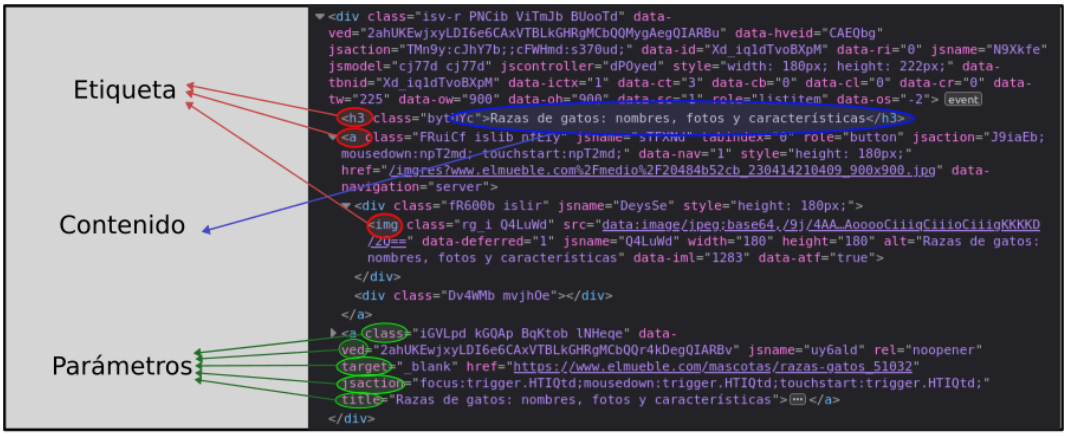


FIGURA 2.1: Código HTML de ejemplo, en este caso correspondiente al buscador de imágenes de Google.

y desarrollo efectivo de páginas web, y también para la extracción automática de información a partir de las mismas. En la Figura 2.1 se presenta un ejemplo ilustrativo que muestra estos elementos. Obsérvese que a cada uno le corresponde una etiqueta HTML, que define el tipo específico de comportamiento del mismo. Por ejemplo, a través de etiquetas, se puede indicar que un elemento es un hipervínculo a otra página web, lo que le permite al usuario ser redirigido a una URL específica simplemente haciendo clic sobre él. También se pueden incluir otros tipos de elementos como imágenes, especificando la URL de la misma para que se muestre en la página.

Los parámetros de un elemento pueden utilizarse también para identificarlo unívocamente y para clasificarlo dentro de un grupo de elementos. Para este último propósito, se emplea el parámetro `class`, que indica a qué clase pertenece el elemento. Esto facilita la organización de las páginas en secciones, haciendo que los elementos semánticamente iguales compartan la clase.

El código HTML de una página web se complementa con código CSS y JavaScript [7]. El código CSS se encarga de la presentación y el estilo de los elementos de la página, con el objetivo de crear una interfaz atractiva y amigable para el usuario. Por otro lado, JavaScript se utiliza para hacer que las páginas sean interactivas y dinámicas en respuesta a la interacción del usuario. Mediante este lenguaje, es posible manipular los elementos HTML y ocultarlos del código, haciéndolos visibles sólo cuando se produce una interacción específica por parte del usuario. Sin embargo, el dinamismo que JavaScript aporta a las páginas web puede dificultar el proceso de *web scraping*, ya que evita que obtengan todos los datos de la página únicamente al

cargarla, lo que a menudo requiere la simulación de interacciones del usuario. En algunos casos, los motores de búsqueda emplean estas técnicas para protegerse contra los algoritmos de *scraping* y garantizar un acceso controlado a su contenido.

2.3. *Scraping* web para recolección de datos

Internet alberga millones de páginas web que contienen una amplia variedad de información, desde texto hasta imágenes. Normalmente consumimos este contenido de diversas formas, ya sea para tareas rutinarias como leer las noticias o realizar compras online, o para estudiar o divertirnos.

A la tarea de extraer cantidades masivas de datos a partir de sitios de internet se la denomina *web scraping*. Realizar *web scraping* de forma manual puede ser una labor tediosa y consumir mucho tiempo. Por esta razón, el verdadero potencial de este tipo de tarea radica en su automatización, que se logra mediante programas diseñados para simular la navegación masiva de numerosos sitios web (a menudo automatizando una falsa interacción de usuario), la recolección de datos específicos y su organización para fines posteriores.

En esta sección se abordarán los detalles introductorios más importantes referentes a este tipo de estrategia de recolección de datos.

2.3.1. Historia del *web scraping*

La historia del web scraping se remonta a los primeros días de la *World Wide Web*, cuando investigadores y desarrolladores utilizaban técnicas simples para acceder y analizar el contenido web.

Según [37], el primer navegador web, creado por Tim Berners-Lee en 1991, permitía a los usuarios ver el código fuente de páginas web y guardarlas localmente. Esta es quizá la forma más primitiva y sencilla de *web scraping*, ya que permite a los usuarios copiar y pegar manualmente los datos que necesitan, pero requiere un trabajo manual profundo.

El primer bot web, *The World Wide Web Wanderer* (conocido popularmente como *The Wanderer* [19], fue creado en 1993 por Matthew Gray, y diseñado para medir el tamaño de la web visitando e indexando cada página web que pudiera encontrar. Aunque esta tarea es útil para el proceso de *scraping*, en este caso el bot no almacenaba ni procesaba ningún dato de las páginas, si no que se limitaba a recorrerlas.

El primer rastreador web que realizó *web scraping* fue JumpStation [2], lanzado en 1993 por Jonathon Fletcher. Se trata de un motor de búsqueda cuyo objetivo principal era abordar el creciente problema de la sobrecarga de información en la web y proporcionar un método más eficiente para recuperar información relevante. La característica distintiva de JumpStation era su enfoque en la indexación de palabras clave en las páginas web. A diferencia de los motores de búsqueda tradicionales de esa época, que se basaban principalmente en la coincidencia de términos de búsqueda en los metadatos de las páginas, JumpStation indexaba las palabras clave reales dentro del contenido de las páginas web, las cuales conseguía a través del *web scraping*. El autor recopiló títulos y encabezados de páginas web y los utilizó para crear un índice de búsqueda, constituyendo el primer antecedente de buscadores como Google o Bing. Al mismo tiempo, JumpStation permitía almacenar en caché las páginas para un acceso más rápido.

A medida que la web crecía en tamaño y complejidad, el *web scraping* se volvía más desafiante, requiriendo herramientas y métodos más sofisticados. Una de las herramientas más populares para este fin es BeautifulSoup[3], una biblioteca Python que puede analizar documentos HTML y XML y extraer datos de ellos. Fue lanzado por primera vez en 2004 por Leonard Richardson, y todavía se utiliza ampliamente en la actualidad (ver Sección 2.4.1).

Otro avance en la historia del *web scraping* es el auge de los *web scrapers* visuales, que permiten a los usuarios crear tareas de recolección de datos sin conocer necesariamente cómo programarlas. Estas herramientas utilizan interfaces gráficas de usuario (o GUIs) para permitir a los usuarios seleccionar los elementos que desean extraer de una página web y exportar los datos a varios formatos. Algunos ejemplos de de estas aplicaciones son Octoparse[4]), ParseHub[5] y Web Scraper[6].

En la actualidad, a medida que las tecnologías e industrias avanzan, las empresas buscan obtener una ventaja sobre su competencia. Y debido a que la cantidad de información disponible en internet está creciendo exponencialmente, el *web scraping* se está convirtiendo en uno de los métodos más prominentes y ampliamente utilizados para adquirir datos a escala en diversas industrias y esferas empresariales. Al mismo tiempo, quienes dominan los índices de información o el origen de estos datos comienzan a entender

[2] https://en.wikipedia.org/wiki/JumpStation
[3] https://www.crummy.com/software/BeautifulSoup/
[4] https://www.octoparse.com/
[5] https://www.parsehub.com/
[6] https://webscraper.io/

las desventajas de disponibilizarlo, introduciendo técnicas anti-*scraping* en sus sitios tales como *captchas*, bloqueos de IP y limitaciones de velocidad, o simplemente alterando con frecuencia el código de los sitios para evitar automatizaciones. Ya más recientemente, se han comenzado a aplicar técnicas de aprendizaje automático y procesamiento de lenguaje natural para extraer datos estructurados en estos contextos, y evitar de esa manera alguna de estas barreras.

2.3.2. Proceso de *web scraping*

El *web scraping* puede entenderse, además de como una estrategia de recolección masiva de datos, como una técnica que transforma datos no estructurados en datos estructurados, que pueden ser guardados y analizados en un único archivo centralizado o en una base de datos. Para lograr transformar la información contenida en la página web, es necesario que el *web scraper* ejecute distintas etapas[15], como se observa en la Figura 2.2.

Fase de *Fetching*: La página web deseada, que contiene la información relevante, es accedida como parte de lo que se conoce como la fase de *Fetching*. Este proceso se lleva a cabo mediante el protocolo HTTP, diseñado para enviar y recibir solicitudes de servidores web [6] (la misma que emplean los navegadores web para obtener el contenido de las páginas). En esta etapa, se utilizan bibliotecas que facilitan el envío de una solicitud `HTTP GET` a la dirección objetivo (URL) para obtener la página HTML como respuesta.

Fase de Extracción: Posterior a la obtención de la página HTML, se procede a la extracción de la información relevante. Durante esta etapa, se emplean herramientas de *scraping* 2.4.1 para extraer exclusivamente la información deseada. Previo a esta fase, se realiza un análisis de la estructura de la página web, permitiendo la identificación y separación de los elementos deseados del resto.

Fase de Transformación: Una vez obtenida únicamente la información relevante, se lleva a cabo su conversión a un formato estructurado adecuado para su presentación o almacenamiento. Con los datos almacenados, se habilita la investigación, permitiendo su utilización para diversos propósitos. En el contexto de nuestro problema planteado en la sección 1.1, estos datos almacenados pueden emplearse por ejemplo para el entrenamiento de algoritmos de aprendizaje profundo.

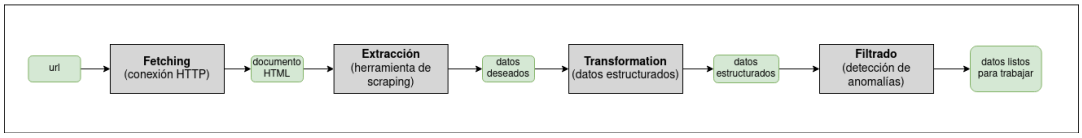


FIGURA 2.2: Fases de trabajo de un sistema de *web scraping*.

Fase de Filtrado: Finalmente, una vez formada la base de datos con la información extraída, debemos asegurarnos que la calidad de la misma es confiable. Esto es indispensable cuando el objetivo de dicho conjunto de datos es ser una fuente de entrenamiento para algoritmos de aprendizaje profundo, ya que de esto depende su correcto funcionamiento. Una posible herramienta para este paso son los algoritmos de detección de anomalías, que se detallan en la Sección 2.5.

2.3.3. Legalidad

El *web scraping* puede tener fines de interés social y alto impacto, como la investigación, el análisis de datos sociales o el establecimiento de políticas públicas. Sin embargo, también plantea cuestiones legales relacionadas con el derecho de propiedad intelectual, el derecho de la competencia, el derecho de protección de datos y los términos de servicio de los sitios web, que deben ser consideradas al momento de aplicarse [4].

En lo que se refiere al derecho de propiedad intelectual, el *web scraping* puede vulnerar los derechos de autor o de marcas registradas si se extrae información protegida sin autorización o licencia del titular [22]. Por ejemplo, si se extraen imágenes que son obras originales o que contienen signos distintivo, el *web scraper* podría incurrir en una infracción civil o penal, según el caso. No obstante, existen excepciones y límites al derecho de autor que podrían justificarlo en determinadas circunstancias, como la cita, la ilustración para la enseñanza o la investigación científica [22].

En cuanto al derecho de la competencia, el *web scraping* puede suponer una práctica desleal si se realiza con fines de competencia desleal o parasitaria [4]. Por ejemplo, si se extraen datos de precios o productos de un competidor para copiarlos o para alterar el mercado, quienes lo utilizan podrían ser sancionados por las autoridades de competencia o demandados por el afectado.

En materia del derecho de protección de datos, el *web scraping* puede implicar una violación a los datos personales si se extraen datos que permitan

identificar o hacer identificable a una persona física (por ejemplo, si se extraen nombres, correos electrónicos, direcciones o perfiles sociales) [5]. En este caso, el *web scraper* debería cumplir con las obligaciones establecidas por la normativa aplicable en el territorio de origen, como el Reglamento General de Protección de Datos (o GDPR) en Europa, por citar un caso. Entre estas obligaciones se encuentran las de informar a los interesados, obtener su consentimiento cuando sea necesario, garantizar su seguridad y respetar sus derechos [5].

Por otro lado, el *web scraping* puede contravenir las condiciones establecidas por los propietarios o gestores de los sitios web para regular su uso, violando los términos de servicio de los sitios web[4]. Por ejemplo, si se prohíbe expresamente el *web scraping* o se limita su frecuencia o alcance, el *scraper* podría ser bloqueado por los sitios web o sus usuarios recibir reclamos por incumplimiento contractual.

Otra práctica que puede hacer que el *web scraping* sea ilegal es la generación de un alto tráfico de peticiones a una página web específica, lo que puede resultar en una denegación de servicio (DDoS) al servidor objetivo. Para evitar este tipo de comportamiento, es fundamental que los web scrapers no sean intrusivos y no interfieran negativamente con el funcionamiento de los servidores de la página destino [18].

En conclusión, el *web scraping* no es ilegal en sí mismo, pero debe realizarse con cautela y respeto a las normas y derechos aplicables.

2.4. *Web scraping* para la recolección de imágenes

Retomando nuestra problemática inicial, nos proponemos abordar el desarrollo de un algoritmo de *web scraping* que permita recolectar un volumen considerable de imágenes que respondan a una determinada consulta, y que puedan luego emplearse para entrenar algoritmos de aprendizaje profundo.

La obtención de este conjunto de datos puede llevarse a cabo ya sea utilizando servicios especializados para este fin o desarrollando un *web scraper* desde cero. A continuación, procederemos a comparar las opciones comerciales disponibles, considerando cuidadosamente sus atributos y beneficios.

2.4.1. Herramientas para *web scraping* de imágenes

APIs de buscadores

Los buscadores web son la primera opción que se nos viene a la mente al pensar en extracción de datos por *web scraping*. En líneas generales, uno podría suponer la existencia de APIs asociadas a estos motores que permitan, dada una consulta, obtener las imágenes que normalmente aparecerían como resultado si la búsqueda se realizara en los motores correspondientes.

En el caso de Bing, el buscador web desarrollado por Microsoft, la misma empresa ofrece este servicio [7]. En el caso de Google Images, no existe una API oficial de la compañía, pero existen alternativas que realizan una labor similar [8].

Una de las ventajas más significativas de estas APIs es su rápida respuesta, además de permitir la inclusión de parámetros para personalizar los resultados. No obstante, éstas también presentan desventajas asociadas a su costo. En particular, su versión gratuita restringe la cantidad de solicitudes permitidas, por lo que para realizar una extracción masiva se requiere de una suscripción mensual, generando costos adicionales (del orden de los 100 dólares mensuales). Por consiguiente, incrementar el número de imágenes descargadas requiere de estrategias alternativas, como formular múltiples consultas diferentes pero asociadas al mismo concepto (lo que permitiría seguir aprovechando la versión gratuita). Enfoques de ese estilo permiten obtener mayores tamaños de muestra, aunque es factible que incluyan imágenes repetidas. Además, el proceso de ideación de términos de búsqueda alternativos complejiza notoriamente.

Plataformas de descarga de imágenes gratuitas

Explorando opciones de acceso gratuito para la descarga masiva de imágenes, nos encontramos con servicios populares como Pixabay [9], Unsplash [10] y Pexels [11].

Estas plataformas ofrecen acceso a una extensa variedad de imágenes, son fáciles de utilizar y permiten recuperar imágenes de alta calidad. Sin embargo, es importante señalar que presentan limitaciones significativas para ciertos contextos. En particular, su alcance se ve restringido a la descarga de

[7] https://www.microsoft.com/en-us/bing/apis/bing-image-search-api
[8] https://serpapi.com/
[9] https://pixabay.com/
[10] https://unsplash.com/
[11] https://www.pexels.com

imágenes contenidas en sus propias bases de datos, constituida sobre todo por fotografías e imágenes naturales. Esto es desventajoso al momento de realizar búsquedas más específicas, donde es posible que no existan resultados o la cantidad de opciones sea limitada. Si pensamos entonces en aplicaciones como la nuestra, enfocada en la recuperación de imágenes médicas desde la web, esta limitante las vuelve inefectivas.

Herramientas programáticas

Otra alternativa para implementar web scraping es desarrollar scripts propios, valiéndose de tecnologías existentes para analizar páginas web. Las mismas varían dependiendo de su enfoque de análisis, que va desde estudiar su código HTML y parsear el contenido útil a realizar interacciones directas con el navegador web, para descargar su contenido. Algunas de las librerías más populares son:

- **Beautiful Soup:**[12] Es una biblioteca de Python diseñada para extraer información de páginas web. Se sitúa sobre un analizador HTML que organiza los elementos del código en un árbol, proporcionando formas idiomáticas en Python para iterar, buscar entre sus elementos y modificarlo. Una vez que se tiene el árbol, se puede acceder al contenido mediante métodos proporcionados por la biblioteca, que devuelven uno o más elementos que coinciden con las condiciones especificadas, como el tipo de elemento, la clase, el id o el contenido del texto. Cada elemento del árbol corresponde a una etiqueta en el código HTML con su contenido y parámetros asociados. Para determinar las condiciones de los elementos que buscamos, es necesario realizar un análisis manual previo. Por ejemplo, si queremos extraer los titulares de las noticias de una página de un noticiero, primero inspeccionamos la página para entender el valor del parámetro `class` de los elementos HTML que contienen los titulares. Luego, al analizar la página con Beautiful Soup y crear el árbol, utilizamos el método `findAll`, proporcionándole el nombre de la clase como parámetro, para obtener todos los elementos que cumplen con esa condición. Beautiful Soup es una herramienta sencilla y flexible para el *web scraping* debido a su simplicidad y facilidad de uso. Sin embargo, tiene la limitación de utilizar sólo un analizador para el código HTML, lo que impide ejecutar código JavaScript. Por lo tanto, no es

[12] https://www.crummy.com/software/BeautifulSoup/bs4/doc/

adecuada para páginas con contenido dinámico que se carga después de una interacción del usuario.

- **Scrapy**[13] Es una biblioteca de Python de alto nivel, utilizada para rastrear y extraer datos estructurados de sitios web. Es una herramienta utilizada para una amplia gama de propósitos, desde la minería de datos hasta el monitoreo y las pruebas automatizadas. Su arquitectura modular permite a los usuarios definir cómo se deben rastrear los sitios web, qué datos se deben extraer y cómo se deben almacenar o procesar esos datos. Además, Scrapy es altamente personalizable y extensible, lo que permite a los desarrolladores adaptar el marco a sus necesidades específicas, haciendo que sea una herramienta popular para el *web scraping* y *crawling*. La principal desventaja que tiene es que no puede manipular e interpretar código JavaScript, por lo que no es adecuado para sitios web que dependen en gran medida de JavaScript para cargar contenido o interactuar con el usuario. Además, su curva de aprendizaje es muy pronunciada, haciendo que requiera mucho tiempo familiarizarse con su arquitectura y conceptos claves.

- **Selenium** [14] Es una herramienta de código abierto creada para la automatización de pruebas que validan aplicaciones web en diferentes navegadores y plataformas. Además, es una herramienta popular para el *web scraping* debido a su capacidad para manejar JavaScript y extraer contenido dinámico. Está disponible para varios lenguajes de programación como Java, C#, Python, etc. Esta herramienta permite interactuar con la página web simulando el comportamiento de un usuario y ejecutar código JavaScript, por lo que es la herramienta ideal para aquellas páginas que ocultan su contenido en el HTML inicial y sólo se desbloquean con alguna interacción del usuario. La desventaja que tiene es que el proceso de *scraping* comienza una vez está cargada completamente la página, por lo que es más lento.

Es importante señalar que, a pesar de existir estas librerías, el código de un *web scraper* tiene una dependencia directa con el código HTML de la página de la cual extrae información. Cualquier modificación en dicho código puede perjudicar la aplicabilidad del software, lo que implica la necesidad de realizar mantenimientos periódicos, y de personalizar las herramientas a cada sitio.

[13] https://scrapy.org/
[14] https://www.selenium.dev/documentation/webdriver/

2.5. Filtrado de imágenes incorrectas con aprendizaje automático

En búsquedas y extracciones masivas de imágenes, es de esperar que un cierto número de las descargas no se corresponda con la búsqueda original. Entrenar algoritmos con este tipo de datos puede producir sesgos en sus respuestas, o un mal uso de su capacidad [21]. Es por ello que se vuelve necesario implementar en los *web scrapers* métodos que permitan o bien refinar las búsquedas realizadas para asegurar una convergencia hacia las imágenes esperadas, o bien utilizar métodos de filtrado que permitan rechazar aquellas imágenes que no responden a los patrones esperados.

Para automatizar este proceso, una de las soluciones más útiles es emplear métodos de aprendizaje automático [26]. Los métodos supervisados requieren entrenarse sobre datos etiquetados, incluyendo en este contexto imágenes aceptables y rechazables. Sin embargo, el espacio de rechazables es prácticamente infinito, por lo que no resulta práctico entrenar este tipo de modelos en este contexto [8].

Alternativamente, los métodos de detección de anomalías se entrenan únicamente con imágenes consideradas normales, y aprenden por sí mismos a caracterizarlas para rechazar cualquier otra que no se asemeje a ellas [3]. Este tipo de enfoque es el que se utilizará en nuestro trabajo, ya que no solo permite identificar patrones y regularidades en grandes conjuntos de datos, sino que también ofrece la capacidad de reconocer anomalías y desviaciones inesperadas.

Capítulo 3

Diseño de la solución

Debido a las limitaciones de personalización y el elevado costo asociado con los servicios de *scraping* online presentados en el Capítulo 2, en este trabajo se propone desarrollar un *web scraper* para imágenes, que permita realizar extracciones masivas de datos de internet que respondan a una determinada consulta.

Aunque la implementación de un *web scraper* propio implica un esfuerzo inicial en términos de aprendizaje y desarrollo, presenta ventajas significativas, como la posibilidad de personalizar la extracción de imágenes de manera específica y la eliminación de costos económicos asociados a servicios externos.

El primer paso para el desarrollo de un *web scraper* desde cero consiste en definir la página web objetivo de la cual se extraerá la información [27]. En este contexto, Google, reconocido por su capacidad para brindar resultados precisos en respuesta a consultas específicas, se erige como la elección más correcta. En particular, la habilidad de Google Images para centralizar grandes cantidades de información visual lo posicionan como el candidato ideal para esta implementación (ver Sección 3.1).

En este capítulo se abordará en detalle el diseño propuesto para el *web scraper*. Inicialmente abordaremos el proceso de descarga de imágenes a partir de Google Images, el modelado de una solución abstracta al problema de recorrido de esta web, y luego las diferentes implementaciones alternativas planteadas como parte de este trabajo final.

3.1. Descarga de imágenes en Google Images

Google Images es uno de los buscadores de imágenes más populares. Suele ser el primer lugar al que acudimos cada vez que requerimos de imágenes

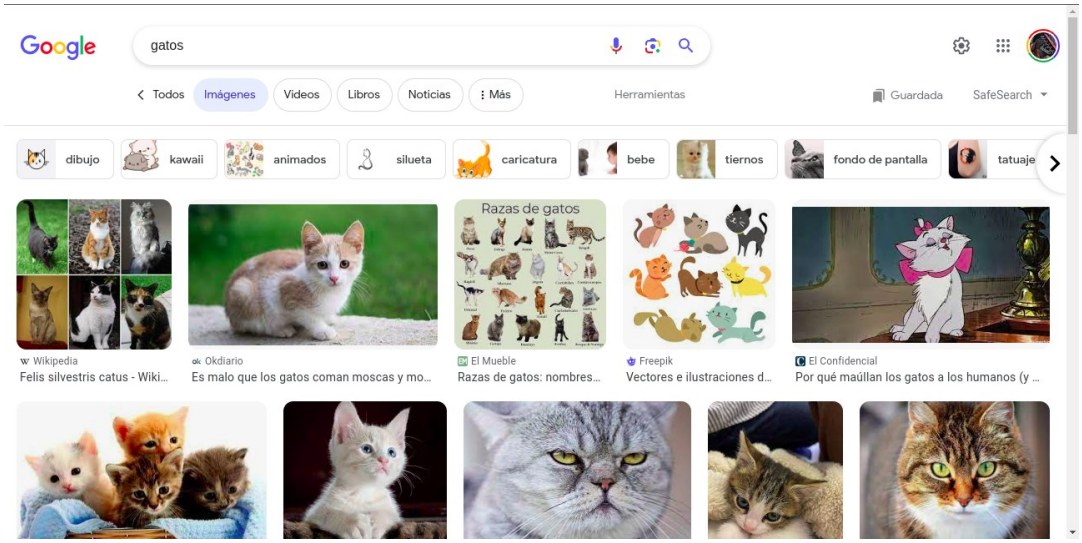


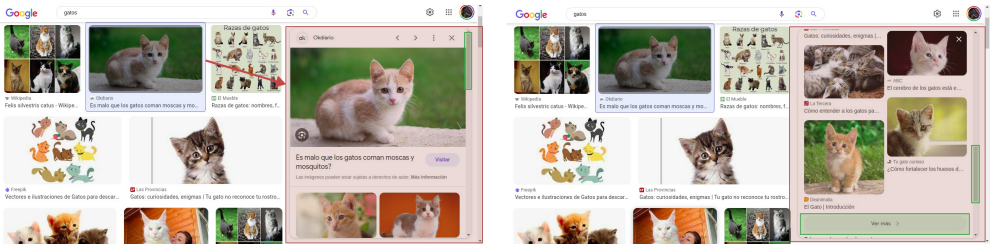
FIGURA 3.1: Ejemplo de una búsqueda de imágenes en Google Images, para la consulta `gato`.

para documentos, presentaciones, noticias, etc. Sin embargo, extraer imágenes desde esta plataforma de manera programática es desafiante, debido entre otros motivos a la ausencia de una API gratuita que nos permita automatizar el proceso, y a la complejidad del código web de la plataforma.

Para realizar una búsqueda en Google Images, es necesario introducir una descripción en la caja de texto (Figura 3.1). Tras unos instantes, la plataforma muestra una serie de imágenes asociadas a la consulta ingresada. Estas imágenes son en realidad referencias al contenido de otras páginas web, que fueron indexadas y analizadas por Google mediante *web crawling*. El motor de búsqueda simplemente recoge el enlace asociado a cada imagen, y las presenta de manera tabular, incluyendo el nombre del sitio web de origen y parte del título del mismo.

La página de resultados se puede acceder a través de una URL, por lo que podría eventualmente construirse la dirección de manera manual, sin pasar por la búsqueda en el espacio de texto. La estructura de la URL se divide en dos partes principales: la dirección de la página y los parámetros enviados a la página. Por ejemplo, para realizar la búsqueda de la palabra "gatos" ilustrada en la Figura 3.1, la URL asociada es `www.google.com/search?tbm=isch&q=gatos`. Aquí, el segmento `tbm=isch` indica que se trata de una búsqueda específica de imágenes, y `q=gatos` especifica el contenido de esta búsqueda.

Una solución trivial al *web scraping* de imágenes podría ser, entonces, la construcción de un *script* Python que simplemente construya la URL de la



(a) Imagen asociada al thumbnail con la calidad original.

(b) Botón "ver más" que aparece cuando scrolleamos en la nueva sección.

FIGURA 3.2: Nueva sección que se abre en la página luego de clickear en un thumbnail.

búsqueda, y que luego analice el contenido de la página de resultados, realizando diferentes pedidos HTTP basándose en las URLs asociadas a cada imagen. Sin embargo, para prevenir este tipo de interacciones, Google implementa restricciones en cuanto al número de solicitudes realizables. Así, el abuso de pedidos HTTP puede resultar en el bloqueo de la IP por detección como bot malicioso.

Otra limitación que presenta Google Images está relacionada con la calidad de las imágenes disponibles en esa primera ventana de resultados. En particular, las representaciones mostradas allí son imágenes miniaturizadas (*thumbnails*), no los datos originales. Para acceder a la imagen real asociada a cada miniatura, es necesario hacer clic sobre cada una de estas, lo que permite acceder a una columna adicional con más información sobre la imagen, incluyéndola en su resolución original (Figura 3.2(a)). Así, para poder descargarla, es necesario ejecutar el código JavaScript asociado a la interacción del clic.

Finalmente, existe una limitante adicional, vinculada con la cantidad de *thumbnails* visibles. En particular, Google limita explícitamente el número de miniaturas resultantes de la búsqueda. Para desbloquear más imágenes, es necesario desplazarse hasta el final de la página con el *scroll* del *mouse* o la barra de desplazamiento. Sin embargo, incluso realizando este desplazamiento existe un tope máximo preestablecido, a partir del cual ya no es posible continuar recibiendo resultados. Una solución para este inconveniente es recurrir al botón "Ver más" disponible al hacer clic sobre una de las miniaturas (Figura 3.2(b)). Al hacer clic en este botón, somos redirigidos a una nueva página que contiene imágenes relacionadas con la imagen inicial seleccionada (aquella por la cual hicimos clic en el *thumbnail*, como se muestra en la Figura 3.2(a)). Este proceso se puede repetir recursivamente, ya que cada nueva

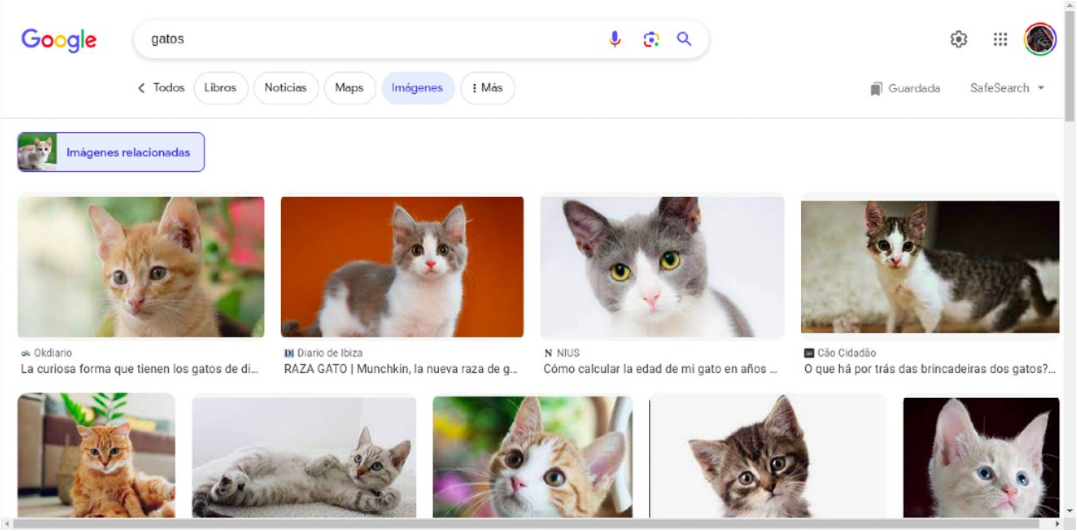


FIGURA 3.3: Página obtenida luego de clickear el botón "ver más" y que contiene las imágenes relacionadas.

imagen en las páginas resultantes también tiene un botón "Ver más", que lleva a una nueva página con imágenes adicionales relacionadas (Figura 3.3). Es importante señalar que, debido a esta funcionalidad, se puede obtener una cantidad significativamente mayor de imágenes relacionadas a partir de una sola *query*. Sin embargo, es necesario tener en cuenta que los resultados obtenidos pueden contener imágenes repetidas, lo cual debe ser tenido en cuenta para evitar caer en un ciclo recursivo, y además puede suceder que, para imágenes incorrectas o poco relacionadas con la consulta, se profundice descargando imágenes similares.

3.2. Modelado de la solución

En el proceso de búsqueda de imágenes a través del *web scraper*, podemos conceptualizar la navegación como un árbol que se ramifica a medida que exploramos las distintas páginas. Este árbol comienza con la página principal, que se genera a partir de la consulta inicial, y se expande a medida que seguimos los hipervínculos de las imágenes obtenidas.

Esta metodología de exploración, conocida como *web crawling*, simula el comportamiento de un motor de búsqueda al seguir enlaces de una página a otra. En nuestro caso, cada nodo del árbol representa una nueva página web a la cual accedemos a través del botón "ver más.ªsociado a una imagen obtenida como resultado de la búsqueda inicial. Esta estructura, representada

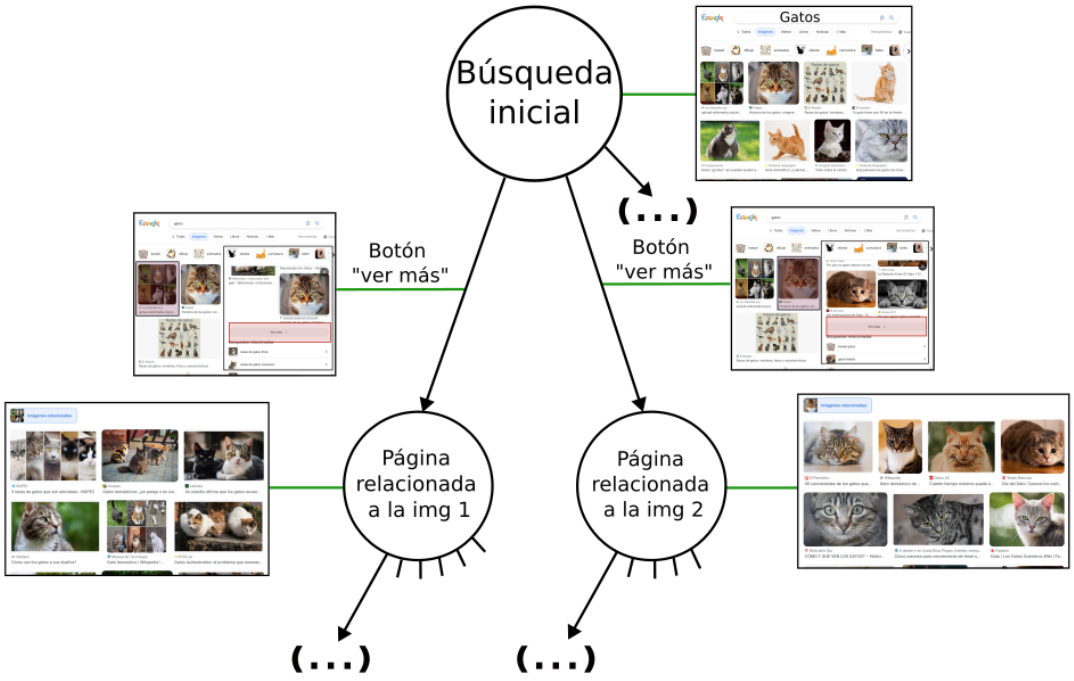


FIGURA 3.4: Representación en forma de árbol del proceso de recorrido de las búsquedas de imágenes.

gráficamente en la Figura 3.4, nos permite recorrer eficazmente los posibles caminos que podemos seguir durante la navegación.

3.3. Diseño de solución secuencial *naive*

En la sección 2.3.2 vimos que existen cuatro fases por las que debe pasar nuestro *web scraper*: *fetching*, extracción, transformación y filtrado. Las primeras dos etapas consisten en la interacción con el navegador web y la página objetivo. La tercera etapa estructura los datos extraídos. Finalmente, la última etapa consiste en la clasificación de las imágenes resultantes para establecer si corresponden a la población esperada o no.

El patrón de diseño que se corresponde a esta dinámica de trabajo es el *Pipeline*, un enfoque lineal donde cada módulo procesa la salida del módulo anterior[35]. Nuestro programa constará, entonces, de tres módulos principales, conectados de forma secuencial por la salida del módulo anterior (Figura 3.5). En primer lugar, el 'Productor' tomará las tareas de *fetching* y extracción: será responsable de abrir, analizar e interactuar con la página web de Google Images, extrayendo los enlaces de cada imagen en respuesta a una consulta. Posteriormente, el 'Consumidor' utilizará estos enlaces para descargar las

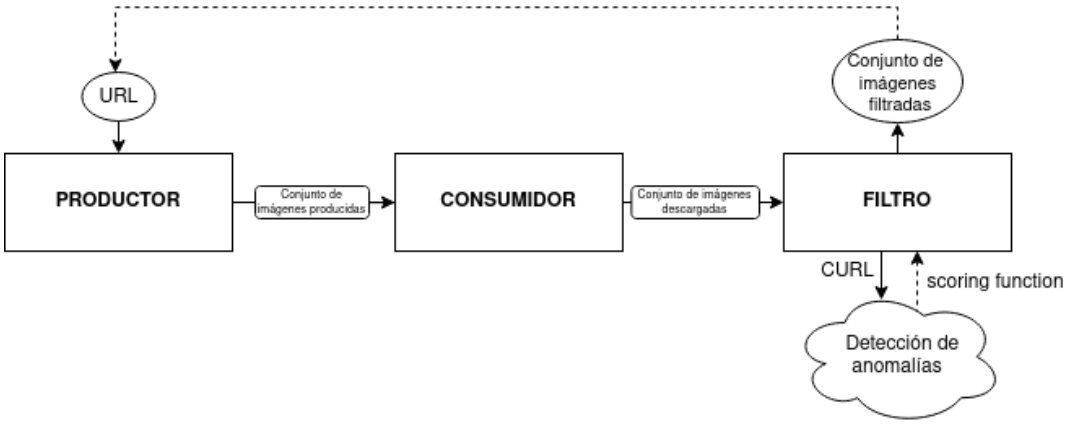


FIGURA 3.5: Módulos principales con los que plantea resolver el problema. El Productor toma una URL y genera una lista de imágenes a descargar. El Consumidor accede a cada imagen de la lista y las descarga. El Filtro se conecta con un algoritmo de detección de anomalías disponibilizado en una web, y le asigna a cada imagen un score de anomalía.

imágenes correspondientes logrando estructurar la información extraída por el Productor, y asumiendo entonces la responsabilidad de la fase de extracción. Finalmente, la fase de filtrado la hará el módulo 'Filtro', que procesará las imágenes descargadas utilizando un algoritmo de detección de anomalías basado en aprendizaje automático. Este módulo evaluará qué tan anómalas son las imágenes, permitiendo rechazar aquellas que excedan un cierto umbral, y obtener así un conjunto de imágenes filtradas más reducido, pero de mejor calidad.

En nuestro diseño, contemplamos al pipeline como una estructura cíclica. En un principio, el Productor comienza abriendo y extrayendo las imágenes de la página obtenida a partir de la búsqueda hecha en Google Images. Por cada imagen en esta ventana, genera un nodo del árbol de recorrido (ver Sección 3.2), que contiene el link de la imagen para que el consumidor pueda descargarla, y la URL a la página con las imágenes relacionadas, según se accede vía el botón "ver más" (Figura 3.2(b)). Los nodos de recorrido procesados por todos los módulos del pipeline serán luego la entrada del Productor cuando éste termine de iterar una página.

3.3.1. Estructura para comunicación entre módulos

El flujo de información entre los distintos módulos del programa requiere una estructura que defina el tipo de elementos que se transfieren de un

módulo a otro. Esta estructura debe contener la información asociada a toda imagen que vaya a descargarse y/o haya sido descargada. Cada módulo procesará estos elementos, entonces, enriqueciéndolos con nueva información.

Dado el mapeo del proceso de descarga y exploración con la idea del recorrido de un árbol que conecta imágenes (Figura 3.4), llamaremos a esta estructura "Nodo". El Nodo almacenará en este caso toda la información relacionada con una imagen, incluyendo atributos tales como el enlace para obtenerla, la consulta desde la cual se extrajo, la ruta a partir de la cual se descargó, el nivel de anomalía calculado, entre otros. Gran parte de esta información solo se obtiene cuando el Nodo es procesado por uno de los módulos, por lo que inicialmente, muchos de estos atributos pueden estar vacíos, pero se completarán paulatinamente conforme se avance en el proceso.

Los Nodos representan entonces imágenes específicas, y uno de los atributos clave que contendrán es la URL de la página que contiene los resultados relacionados con la misma imagen, como se explicó en la sección 3.1 (ver Figura 3.3). Esto posibilita que cada Nodo, después de pasar por todos los módulos, pueda servir como entrada del Productor, para que produzca imágenes relacionadas a partir de ella. De esa forma, el Nodo obedece a la idea de recorrido de un árbol.

3.3.2. El módulo Productor

A partir de una búsqueda inicial, se genera una URL que abre la página de Google Images con los resultados iniciales. La tarea del módulo Productor es abrir esa página web y obtener cada una de las imágenes resultantes. Para esta tarea existen varios desafíos, desde las dificultades que pone la misma página para evitar el *scraping* hasta el hecho de ir generando cada uno de los nodos correctamente siguiendo la lógica del árbol de recorrido 3.2.

En la sección **??** se nombraron las distintas herramientas existentes para poder extraer información de una página web, como pueden ser BeautifulSoup, Scrapy o Selenium. La decisión sobre cuál herramienta es la más adecuada para nuestro *web scraper* se baso en las características de la página objetivo. Como se mencionó en la sección 3.1, el motor de búsqueda de Google posee contenido dinámico, haciendo que parte del mismo sólo sea visible en el código HTML tras una interacción del usuario (por ejemplo, la imagen real de los resultados sólo aparece cuando el usuario hace clic en el *thumbnail* correspondiente a la misma, como se ve en la Figura 3.2(a)). Es por esta razón que descartamos BeautifulSoup y Scrapy, que no soportan JavaScript ni

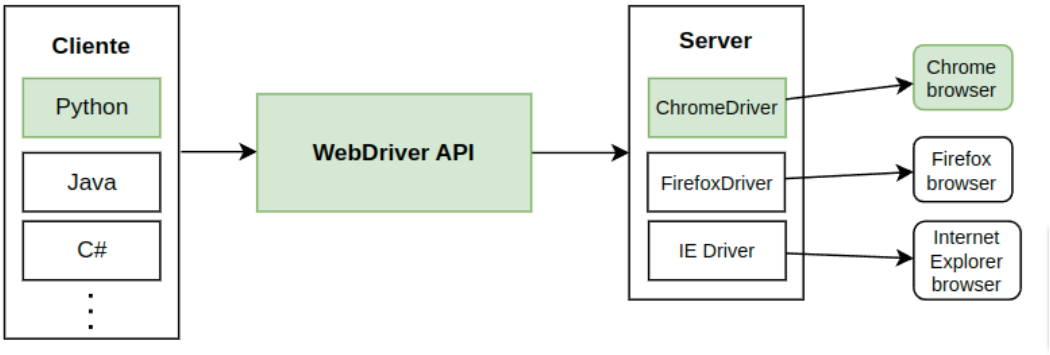


FIGURA 3.6: Interacción de Selenium con el navegador web. Las cajas marcadas en verde corresponden a los elementos utilizados en este trabajo.

permitan una interacción.

A pesar de que Selenium no fue creado como herramienta para *web scraping* sino para testear páginas web, su capacidad de simular la interacción de un usuario y su facilidad para la extracción de elementos del HTML hacen que sea una herramienta muy útil cuando se quieren *scrapear* páginas con contenido dinámico. Es por esto que nuestro Productor utilizará Selenium para interactuar con la página web y desbloquear el contenido en el cual se encuentra el link de cada una de las imágenes.

Nuestro módulo Productor recibe una URL asociada a una página web de descarga de imágenes de Google Images. El Productor debe entonces abrir la página web con las imágenes resultantes utilizando el navegador, por medio de Selenium. Para tal fin, Selenium provee una interfaz llamada `WebDriver`, que abstrae la funcionalidad necesaria para la interacción con un determinado navegador dato. En este caso, haremos uso de la clase `ChromeDriver`, que implementa la interfaz `WebDriver`, la cual permite al usuario interactuar con el navegador Chrome. Como se muestra en la Figura 3.6, el cliente (en nuestro caso, nuestros *scripts* escritos en Python) utiliza la API de `WebDriver` para conectarse con el servidor, y luego se utiliza la clase correspondiente al navegador elegido para interactuar con la página.

Una vez que Selenium se conecta con el servidor y abre el navegador con la URL indicada, el cliente puede acceder a los elementos de la página e interactuar con la misma a través de los métodos ofrecidos por la interfaz `WebDriver`. Estos métodos permiten cliquear sobre elementos, recorrer la página, recuperar el contenido de uno o más elementos, contar la cantidad de

elementos que cumplen cierta condición, etc. Con estas herramientas, el Productor extrae de cada una de las imágenes del resultado de la búsqueda tanto el link en donde se encuentra la imagen completa como la URL que apunta a sus imágenes relacionadas (que se accede como se ilustra en la Figura 3.3). El Productor guarda esta información dentro de un nuevo Nodo (Sección 3.3.1), que luego será procesado por el resto de los módulos.

Como se detalló en la Sección 3.3, cuando el Productor termine de recorrer todas las imágenes de la página, su siguiente entrada es uno de los nodos que ya fueron procesados por todos los módulos del pipeline (es decir, que ya pasaron por el módulo Filtro). De esa forma, el Productor captura el nodo, recupera la URL de su página de imágenes relacionadas, y la recorre como si se tratara del resultado de una búsqueda, generando nuevos nodos como salida.

Un detalle a considerar es que el Productor recorre primero todas las imágenes de una página, y luego solicita la ejecución del siguiente módulo. Esto se hace debido a que la tarea de abrir el navegador con la página web y esperar a que se cargue su contenido completo tiene un costo temporal alto, por lo que sería ineficiente que el Productor abra la página para solamente generar un nodo, y luego habilitara el siguiente módulo. Por esta razón, la ejecución del siguiente módulo del pipeline comienza una vez que el Productor ha recorrido todas las imágenes de la página. También es importante señalar que los nodos que forman parte de la salida del Productor son analizados buscando y descartando nodos cuyas imágenes ya hayan sido agregadas anteriormente. Esto se hace antes de poder ser consumidos por el siguiente módulo para evitar ciclos.

Extracción de las imágenes

Como se explicó en la sección 3.1, las imágenes que se presentan como resultado en el motor de búsqueda son, en realidad, miniaturas o *thumbnails*, versiones reducidas de las imágenes originales, que sólo son visibles al hacer clic sobre su versión pequeña. Por lo tanto, nuestro productor, después de abrir el navegador web con Selenium, debe hacer clic en cada uno de los *thumbnails*. Una vez que la imagen real es visible, extrae dicha imagen junto con el URL a la que nos redirige el botón "ver más", para que el Productor pueda abrir luego esa página. Es posible que, debido a problemas de red o circunstancias diversas, el tiempo de espera para que una imagen se cargue por completo sea considerable, incluso llegando al extremo de quedarse bloqueado de forma indefinida. Con el propósito de abordar esta eventualidad,

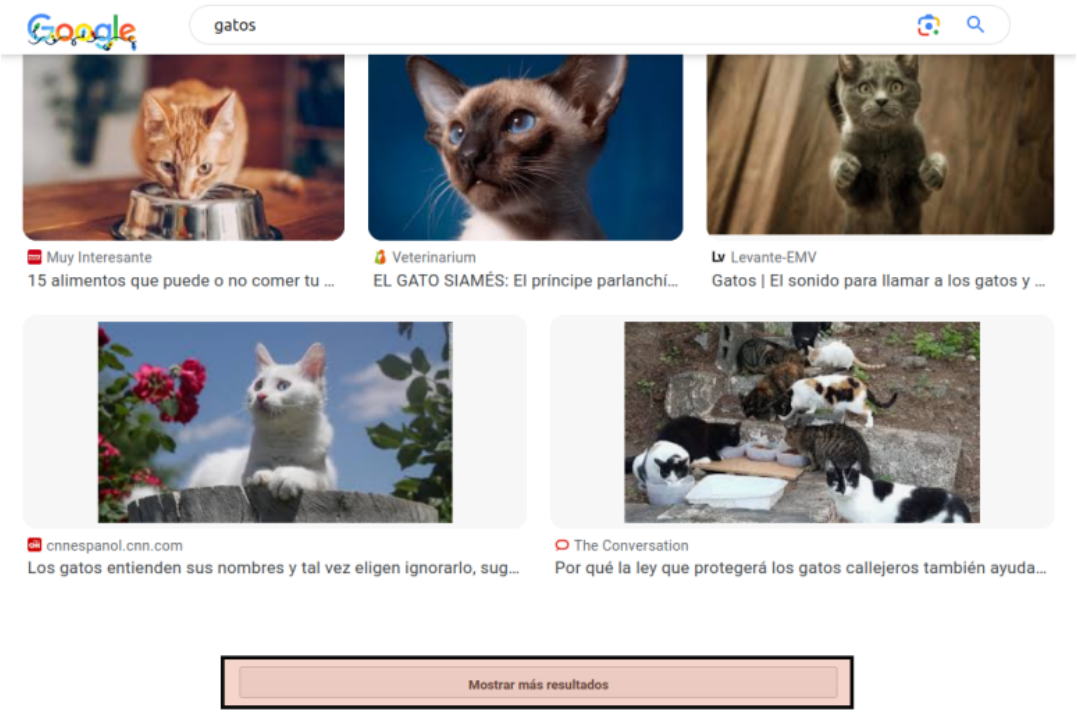


FIGURA 3.7: Botón "Mostrar más resultados" ubicado al pie de la búsqueda en Google IMages.

se decidió implementar un límite de tiempo. Este límite se establece para evitar que el módulo Productor quede inmovilizado en la espera de carga de una única imagen (Ver sección 3.3.5).

Tras hacer clic en todas las imágenes de la página, puede ocurrir en algunos casos que aparezca el botón "Mostrar más resultados", al final de la página (Figura 3.7). Al hacer clic en este botón, se desbloquean más *thumbnails* para seguir recorriendo. El funcionamiento del Productor se representa en el diagrama de actividades de la Figura 3.8. Después de hacer clic en un *thumbnail* y desbloquear la sección donde se encuentra la imagen real y el botón de "Ver más" (ver Figura 3.2(a)), se extrae la información de ambas y se verifica si se llegó al final de la página. En caso de que así sea, se verifica la existencia del botón "Mostrar más resultados". Si existe, se hace clic para continuar recorriendo; si no, se considera que se llegó al final definitivo de la página. En este último caso, el Productor cierra la página y consume el siguiente nodo como entrada para abrir una nueva página con nuevos resultados.

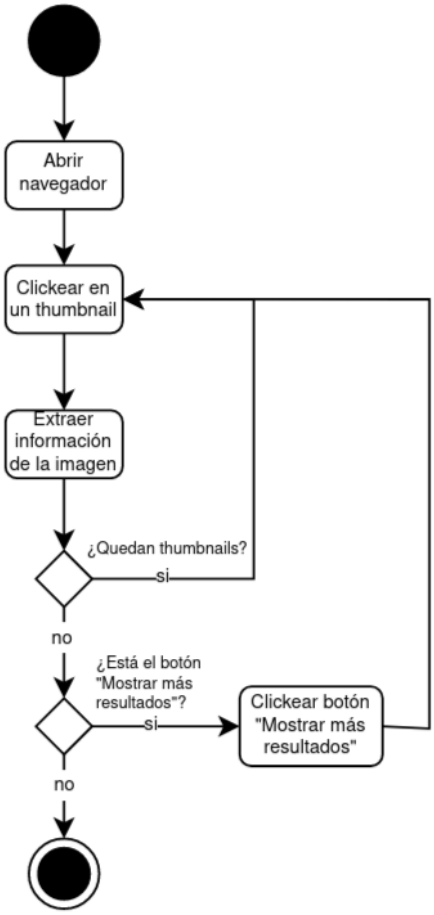


FIGURA 3.8: Diagrama de actividades del Productor representando las diferentes tareas que realiza.

3.3.3. El módulo Consumidor

El módulo "Consumidor" se encarga de descargar las imágenes obtenidas por el Productor. La entrada de este módulo consiste en un conjunto de nodos, cada uno conteniendo información de una imagen extraída (en este punto, únicamente el enlace directo a la imagen).

Por cada nodo recibido como entrada, el Consumidor ejecuta tres pasos en orden (Figura 3.9). Primero, obtiene la imagen mediante una solicitud `HTTP GET` al servidor web, al que se accede a través del enlace disponible en el nodo. La comunicación cliente-servidor se realiza a través del protocolo HTTP, y la biblioteca `requests` facilita estas solicitudes de manera efectiva.

Una vez obtenida la imagen, el segundo paso implica descargarla y almacenarla localmente. Esto se logra utilizando la función **open()** de Python, que abre un archivo en modo de escritura binaria. Dicha función toma como parámetros la ruta y nombre del archivo que se desea crear o sobrescribir.

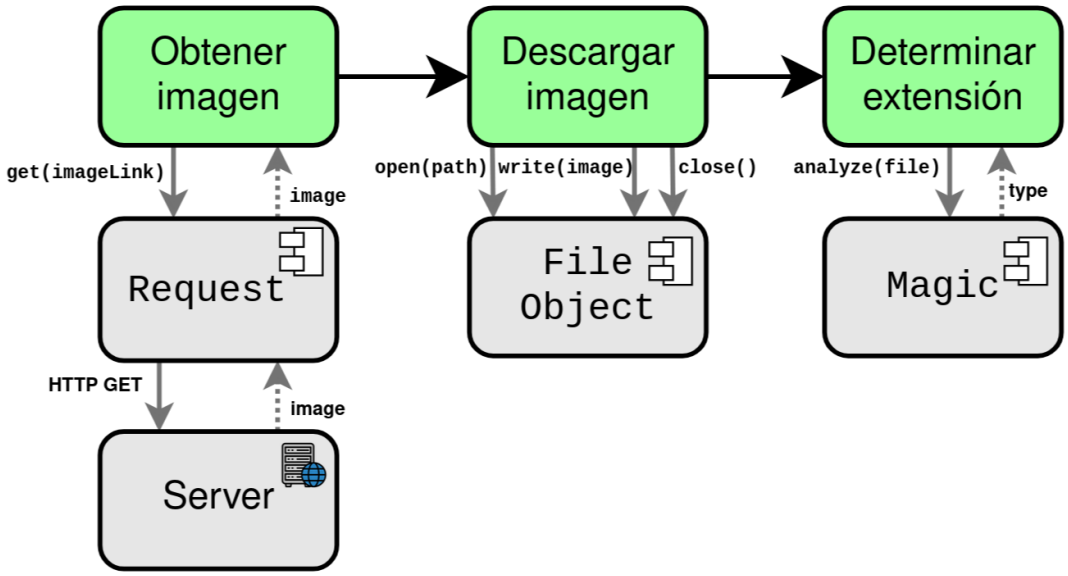


FIGURA 3.9: Comportamiento del Consumidor e interacción con las diferentes librerías y componentes necesarios.

Posteriormente, se utiliza el método **write**, que permite escribir el contenido binario de la imagen en el archivo recién creado. Finalmente, se emplea el método **close** para cerrar adecuadamente el archivo y asegurar que los cambios se guarden correctamente.

El último paso es analizar el contenido del archivo descargado para determinar su extensión, paso necesario dependiendo de las restricciones posteriores de uso de las imágenes (por ejemplo, si quisiera evitarse emplear imágenes JPEG por sus artefactos de compresión). Para realizar este análisis, se emplea la biblioteca `Magic`, que proporciona una interfaz para el análisis del tipo de un archivo específico. Para lograrlo, la librería hace una lectura del encabezado del archivo, y almacena la extensión reconocida como una de las variables internas de la estructura Nodo.

3.3.4. El módulo Filtro

Cuando realizamos búsquedas de imágenes en un motor como Google, gran parte de la efectividad de los resultados depende de la consulta realizada. Google toma la búsqueda introducida en el campo específico para este fin, y recupera imágenes comparando esta *query* con las descripciones de millones de imágenes disponibles en las páginas web de su índice. Sin embargo, es posible que algunas de estas no sean exactamente lo que el usuario busca,

si no que estén relacionadas indirectamente. Si se descargaran, el conjunto resultante contendría datos espurios, que perjudicarían la performance de los algoritmos subsecuentes. Es por esta razón que empleamos el último módulo para evaluar en qué medida cada imagen se asemeja a lo que realmente queremos encontrar, utilizando para ello un detector de anomalías.

En líneas generales, el algoritmo empleado se basa en un modelo de Máquina de Vectores de Soporte a una Clase (o One-Class SVM [16]). El mismo se entrena sobre un conjunto de datos considerados no-anómalos o normales, y aprende una función capaz de, dada una nueva muestra, reconocer si la misma pertenece o no a la población esperada. Es importante señalar que el desarrollo de este algoritmo no formó parte de esta tesis, si no que se utilizó un método desarrollado en el laboratorio de forma anterior, y que fue adaptado según fuera necesario para los experimentos.

El módulo Filtro implementado, entonces, se comunica con un servidor que alberga el modelo detector de anomalías, utilizando para ello el protocolo HTTP, mediante la biblioteca `request` de Python. Antes de esto, se abre la imagen utilizando el método `Open()` para archivos, utilizando la ruta almacenada en el nodo, que se cargó cuando el módulo Consumidor descargó la imagen. Posteriormente, se realiza una solicitud `HTTP POST` utilizando `request`, pasando la dirección del servidor y la imagen como parámetros. Al recibir esta solicitud, el servidor analiza la imagen y responde con un archivo JSON que contiene el resultado de la *scoring function* o **función de puntuación** del algoritmo, que asigna valores altos a aquellas imágenes que se asemejan a la distribución de las utilizadas para su entrenamiento [40].

Una vez que el módulo Filtro utiliza el detector de anomalías para obtener el valor de una imagen, éste se carga en una de las variables dentro del objeto Nodo. Finalmente, el conjunto de nodos procesados por este módulo pertenecerán al conjunto de soluciones del *web scraper* y estarán listos para ser consumidos nuevamente por el módulo Productor (ver Figura 3.5).

3.3.5. Control de errores

Los nodos que atraviesan cada fase del pipeline pueden experimentar diversos tipos de errores. Por ejemplo, la descarga de la imagen por parte del Consumidor puede fallar por interrupciones momentáneas del servicio de internet, o porque el tiempo de carga de la imagen al hacer clic en el *thumbnail* supere el tiempo de espera establecido. En estos casos el propio módulo debería ser lo suficientemente resiliente como para descartar el archivo que

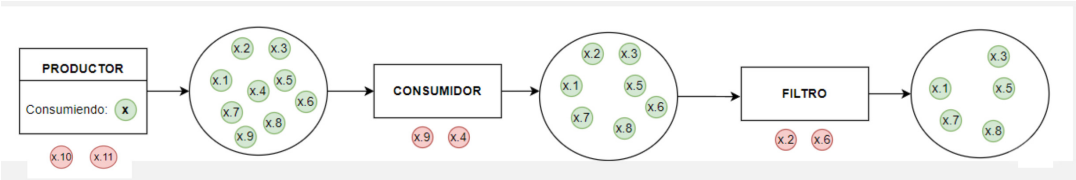


FIGURA 3.10: Representación esquemática del flujo de nodos a lo largo del pipeline de descarga. Los círculos verdes representan nodos que se procesan correctamente y los rojos corresponden a nodos con fallos.

falla, excluyéndolo de su salida, y continuar operando con el resto de imágenes.

Para poder llevar un registro de los errores ocurridos y eventualmente llevar a cabo acciones correctivas, la implementación se diseñó de forma tal que pudiera continuar de ocurrir alguna falla, y al mismo tiempo mantener un registro de las mismas para eventuales análisis estadísticos o correcciones. A tal fin, cada nodo generado almacena una referencia a su padre para que, si por alguna razón, falla en alguno de los módulos, se notifique al padre. Dentro del padre se encuentran variables que indican la cantidad de hijos generados y la cantidad de nodos que fallaron.

La Figura 3.10 representa esquemáticamente las pérdidas de nodos que puede ocurrir a lo largo de cada paso del pipeline. En el ejemplo, el Productor consume el nodo x, lo que significa que abre la página web que contiene imágenes relacionadas con la imagen de dicho nodo 3.3. La salida del Productor serán los nodos generados a partir de las imágenes iteradas de esa página, representados en verde en la figura. Además, existirán nodos que que no pudieron ser procesados (representados en rojo). Estos no pudieron ubicarse a la salida del Productor debido a que no se generaron correctamente. Algunas de las razones para que esto suceda pueden ser que la carga de la imagen real luego de hacer clic en el *thumbnail* tardó más de lo esperado o que el botón "ver más" no se cargue correctamente debido a problemas de conexión.

En la siguiente etapa, el Consumidor debe descargar las imágenes de los nodos indicados por el Productor. Aquellas que pudieron descargarse correctamente se representan en verde, mientras que los nodos en rojo son los que no pudieron ser procesados. Los errores que pueden ocurrir en este módulo incluyen la falla en la descarga de la imagen contenida en el nodo o la falta de conexión para acceder a la imagen a través del enlace.

Finalmente, en la última etapa se toman estos nodos como entrada de un

Filtro, que los procesa para determinar qué imágenes corresponden con la población esperada y cuáles no. Las fallas en este módulo ocurren cuando la solicitud a la API del detector de anomalías falla debido a la incompatibilidad en el tipo de archivo o a la falta de conexión para acceder al servidor. Los nodos que no tienen fallas al procesarse con este módulo son aquellos que sobrevivieron a cada uno de los pasos del pipeline, por lo que están listos para ser consumidos por el Productor y servir para generar nuevos nodos.

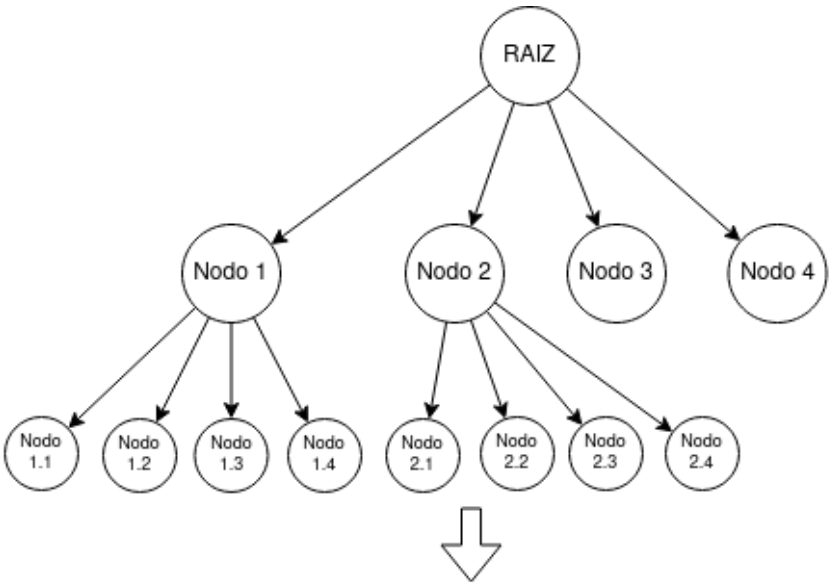
El manejo de errores proporciona robustez al sistema, ya que ante cualquier falla, los módulos cuentan con instrucciones para permitirles responder y notificar el error para continuar con su procesamiento. Cada uno de los errores que ocurren se guarda en una de las variables del nodo padre. Esto significa que el nodo padre (representado como X en la Figura 3.10) conocerá la cantidad de nodos generados y cuántos de ellos fallaron en el Productor, cuántos en el Consumidor y cuántos en el Filtro. Para tal fin, cada módulo es responsable de notificar a su padre sobre cualquier fallo, para que éste actualice la variable correspondiente y lleve un registro de los nodos que van fallando.

3.3.6. Salidas del *web scrapper*

Al finalizar la ejecución del programa, la estructura resultante es un árbol como el de la Figura 3.4, en el que cada nodo representa una imagen y guarda información referida a ésta, como por ejemplo su ubicación en el disco, la consulta de la cual se originó, su nivel de anomalía y la extensión del archivo (jpg, png, etc).

Como en todo árbol, nótese que tendremos dos tipos de nodos, los nodos hojas y los nodos que tienen uno o más hijos. Los nodos hojas son aquellos que no tienen nodos hijos; es decir, aquellos que han atravesado todos los módulos del pipeline correctamente pero no han sido consumidos nuevamente por el Productor. Aquellos que sí fueron consumidos por el Productor serán los que generen nuevos hijos, por lo que estos nodos tendrán información extra además de la relacionada a la imagen que guardan. Esta información corresponde a datos estadísticos en relación al proceso de generación de nuevos nodos, como por ejemplo la cantidad de nodos que fallaron durante el proceso (ver Sección 3.3.5)

Una vez finalizado el programa, es necesario que toda esta información se entregue al usuario. Para ello, se genera un archivo CSV que contiene la



output.csv

Query	Link	URL del ver más	Nivel	Path	Scoring value	Cantidad hijos	Repetidos	Cantidad errores
				Nodo 1 info				
				Nodo 2 info				
				Nodo 3 info				
				(...)				

FIGURA 3.11: Representación del árbol de nodos en el archivo de salida. La salida se produce en formato CSV, donde a cada nodo le corresponde una linea del archivo conteniendo toda la información del mismo.

información de todos los nodos que fueron generados y sobrevivieron a cada uno de los módulos. Este archivo servirá para analizar el porcentaje de acierto de cada una de las páginas, pudiendo entonces calcular cuántas de las imágenes recorridas pudieron ser efectivamente descargadas. Un archivo que centraliza la información de las imágenes obtenidas permite tener una organización sobre el resultado mucho más personalizada, facilitando la clasificación de las imágenes según el criterio del usuario. Por ejemplo, podría utilizarse para agrupar imágenes en grupos distintos teniendo en cuenta su grado de anomalía o su extensión. Como se puede ver en la Figura 3.11, cada uno de los nodos del árbol se proyecta en una linea del archivo de salida.

3.4. Diseño de solución con procesamiento concurrente

La ejecución del programa planteada en el enfoque naive sigue un orden secuencial, comenzando con el módulo Productor, que recorre toda una página generando un nodo por cada imagen encontrada. Luego, el conjunto de nodos, que conforma la salida del módulo Productor, se convierte en la entrada del siguiente módulo, y así sucesivamente hasta que el conjunto de nodos pasa por todos los módulos del sistema.

Gran parte del procesamiento dentro de cada módulo implica demoras por acceso a internet, como en el Productor, donde se espera a que la página web cargue completamente cada vez que se hace clic en un *thumbnail* para extraer la imagen. De manera similar, en el Consumidor se espera a que la imagen se descargue y se escriba en el disco. Finalmente, en el Filtro se aguarda la respuesta del servidor que mantiene operativo al detector de anomalías. A lo largo de los módulos, se utilizan recursos compartidos como el ancho de banda de Internet, el procesador y el disco, y una parte de la demora en la finalización del trabajo del sistema se debe a los tiempos de espera. Además, se utilizan diferentes recursos del propio hardware, lo que implica que, en muchos casos, la CPU esté ociosa.

Para aumentar la eficiencia temporal del sistema, los módulos del sistema podrían ejecutarse en paralelo. Esto es posible debido a que el diseño propuesto divide las funcionalidades en módulos completamente independientes entre sí. Nuestra hipótesis es que la ejecución simultánea de los tres módulos del sistema optimizaría la utilización de recursos y mejoraría el tiempo necesario para descargar una determinada cantidad de imágenes.

Aprovechando la independencia funcional de cada módulo, podemos implementar entonces la ejecución en paralelo mediante *multithreading*. Esta técnica permite que partes del programa se ejecuten concurrentemente, sin tener que esperar que una finalice para que comience la otra. La idea es ejecutar cada uno de los tres módulos en *threads* distintos, aprovechando los tiempos de inactividad del CPU para ejecutar otras partes del programa. Para lograr esto, debemos ajustar parte del diseño para que sea compatible con esta nueva lógica.

En la versión secuencial, el Productor recorre toda la página y genera los nodos correspondientes a cada imagen antes de que el módulo Consumidor pudiera comenzar a ejecutarse. Ahora, con cada módulo ejecutándose en un *thread* separado, el Productor puede generar nodos como salida en el

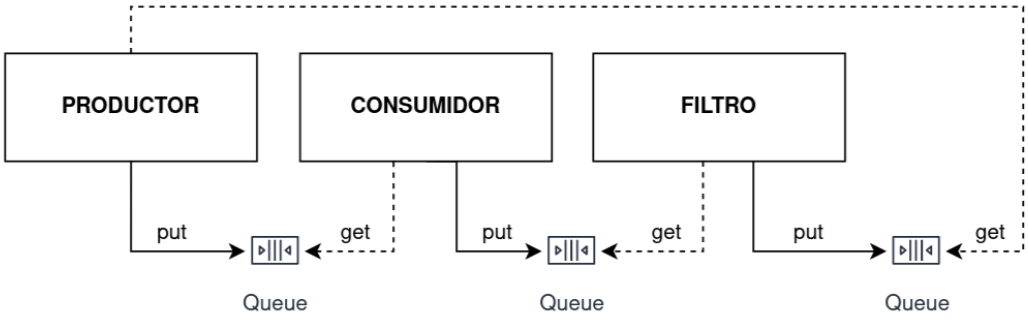


FIGURA 3.12: Diseño concurrente mediante colas (*Queue*) de Python. Cada módulo escribe o lee una cola compartida con su módulo siguiente, lo que permite evitar *race conditions*.

momento en que se producen, permitiendo que los módulos siguientes no tengan que esperar a que se recorra toda la página antes de comenzar a consumir. Lo mismo ocurre con el Consumidor, ya que a medida que procesa los nodos, los libera inmediatamente para que sean procesados por el siguiente módulo.

Para lograr esto, es crucial establecer un mecanismo de comunicación entre los *threads*, ya que queremos que un módulo pueda pasar nodos a otro sin problemas de *race condition*. Este tipo de problemas se produce cuando dos procesos intentan acceder al mismo recurso, lo que puede provocar errores de inconsistencia en el valor del recurso. La estructura **Queue** de Python es la herramienta más adecuada para lograr esta comunicación, ya que implementa automáticamente los bloqueos necesarios para el acceso simultáneo a los datos contenidos.

El diseño de la comunicación entre los módulos se muestra en la Figura 3.12, donde cada módulo consume de una **Queue** y coloca los nodos que procesa en otra, que será consumida por el siguiente módulo. Este diseño nos permite utilizar más eficientemente el tiempo del CPU, ya que el tiempo ocioso de un *thread* puede aprovecharse ejecutando otra parte del sistema en otro hilo. Además, el flujo de nodos será más estable, ya que los nodos se procesan uno a uno, a diferencia del enfoque secuencial, en el cual cada módulo procesa un conjunto de nodos.

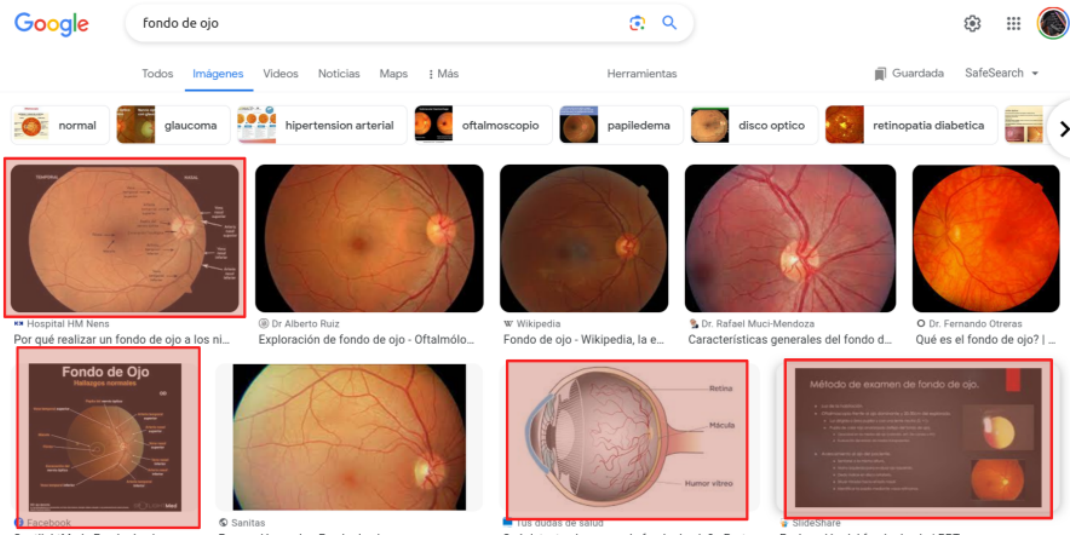


FIGURA 3.13: Ejemplo de búsqueda de fotos de fondo de ojo en Google Images. Las imágenes marcadas en rojo ejemplifican la aparición de resultados anómalos.

3.5. Diseño de solución con heurística de expansión

Cuando buscamos algo en Google Images, no siempre las imágenes resultantes corresponden con lo esperado (ver sección 3.3.4). Dicho problema da sentido al módulo Filtro, el cual se encarga calcular el nivel de anomalía de cada una de las imágenes iteradas. En la Figura 3.13 podemos ver un ejemplo de esto, en donde a la búsqueda "fondo de ojo"se le asignaron resultados esquemáticos, dibujos, o hasta diapositivas (todas marcadas en rojo).

En la implementación más básica, el Productor elige el próximo nodo a expandir simulando un recorrido BFS por el árbol de nodos que se va recorriendo. Por otro lado, cuando un nodo es expandido sabemos, que se recorre su página de imágenes relacionadas (Figura 3.3), por lo que se espera que los resultados a los que se acceda sean similares a la imagen del nodo en cuestión.

Debido a que el criterio en la elección del próximo nodo a expandir es simplemente el orden en el cual fueron agregados al árbol, puede ocurrir que muchos de los nodos que expanda correspondan a datos cuya imagen de origen difiera notoriamente de lo que se esté buscando, teniendo como resultado que se añadan aún más nodos que no tienen que ver con el tipo

de imagen buscada. En el ejemplo de la Figura 3.13, el Productor puede expandir aquellas imágenes de marcadas en rojo, lo que disminuiría significativamente en la eficacia del *web scraper* para el objetivo ulterior de entrenar algoritmos de IA.

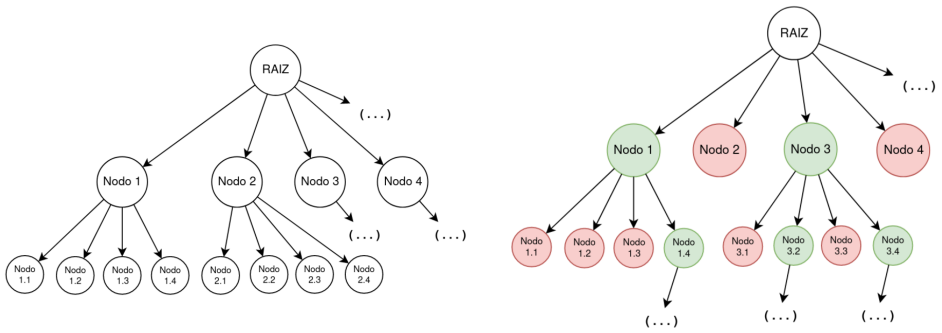
Teniendo en cuenta esta propiedad, podemos hacer uso de la funcionalidad del último módulo Filtro 3.3.4, que determina el nivel de anomalía de una imagen, para priorizar la expansión de aquellos nodos cuyo valor de *scoring function* sea mayor. Aplicando este nuevo criterio, los nodos que terminan su flujo en el pipeline y son procesados exitosamente por el Filtro, se ingresan en una cola de prioridad teniendo en cuenta el nivel de anomalía que tiene la imagen asociada al nodo. Dicha cola será accedida por el Productor cada vez que requiera un nuevo nodo para expandir, y se espera que las nuevas imágenes que se extraigan tengan un nivel de anomalía similar. Nótese, sin embargo, que este criterio no tiene en cuenta el nivel del árbol de recorrido, ya que la cola de prioridad tiene nodos mezclados de todos los niveles, vengan del padre que vengan, ordenados únicamente por su valor de *scoring function*.

3.6. Diseño de solución con poda de expansiones

Con los diseños analizados hasta el momento, dentro del conjunto de imágenes descargadas podremos encontrar imágenes con valores de *scoring function* bajos, correspondientes a anomalías. Para erradicar esto, deberíamos aplicar una estrategia que se encargue de "limpiar" los nodos que fueran generándose, según el nivel de anomalía calculado por el Filtro.

Se plantea entonces un nuevo enfoque adicional, que consiste en implementar una estrategia de poda de las expansiones cuyo nivel de anomalía sea superior a un umbral establecido previamente. De esta manera, puede evitarse que aquellos nodos cuyas imágenes no corresponden con el objetivo buscado se incluyan en el resultado final, y además se evita que sean expandidas por el Productor (Figura 3.14). Al mismo tiempo, es posible preservar el flujo por niveles del diseño original, permitiendo un recorrido más ordenado del espacio de resultados.

Para establecer un criterio objetivo sobre qué nivel de anomalía considerar aceptable, debe realizarse un análisis previo con un grupo representativo de imágenes del objetivo que se requiere, y un subconjunto de imágenes que no se correspondan con ellas. Así, se calcula el grado de anomalía para cada



(a) Recorrido sin aplicar la estrategia de poda, donde todos los nodos pueden ser expandidos.

(b) Recorrido con poda. Los nodos rojos son aquellos cuyo valor de *scoring function* fue menor al umbral establecido, y que no son expandidos. Los nodos verdes tienen un valor superior al umbral, por lo que son agregados al conjunto solución y luego expandidos.

FIGURA 3.14

una de las imágenes, y se busca el umbral que permita separar mejor ambos grupos de imágenes. Luego, este criterio puede emplearse para filtrar los casos y podarlos.

Al aplicar la poda basada en el grado de anomalía, se espera que el conjunto solución conserve su coherencia visual, ya que las imágenes restantes compartirán similitudes significativas con el objetivo deseado. Así mismo, este enfoque permitiría eliminar eficazmente aquellas imágenes que se desvían del criterio deseado, preservando la calidad visual en el conjunto final.

3.7. Detalles de implementación

Para la materialización del diseño propuesto, se eligió el lenguaje de programación Python debido a su versatilidad, extensa colección de bibliotecas y su facilidad de implementación. Python se revela como la elección idónea para nuestro *web scraper*, especialmente gracias a la implementación de bibliotecas como Selenium, que simplifican significativamente la extracción de información de páginas web dinámicas.

La implementación del diseño se fundamentó en un diagrama de clases, como se presenta en la Figura 3.15, proporcionando una representación visual de las clases y sus interrelaciones en el sistema. En este diagrama, se observa cómo la estructura destinada a almacenar la información de las imágenes, denominada **"nodo"** y explicada en la Sección 3.3.1, se traduce en una

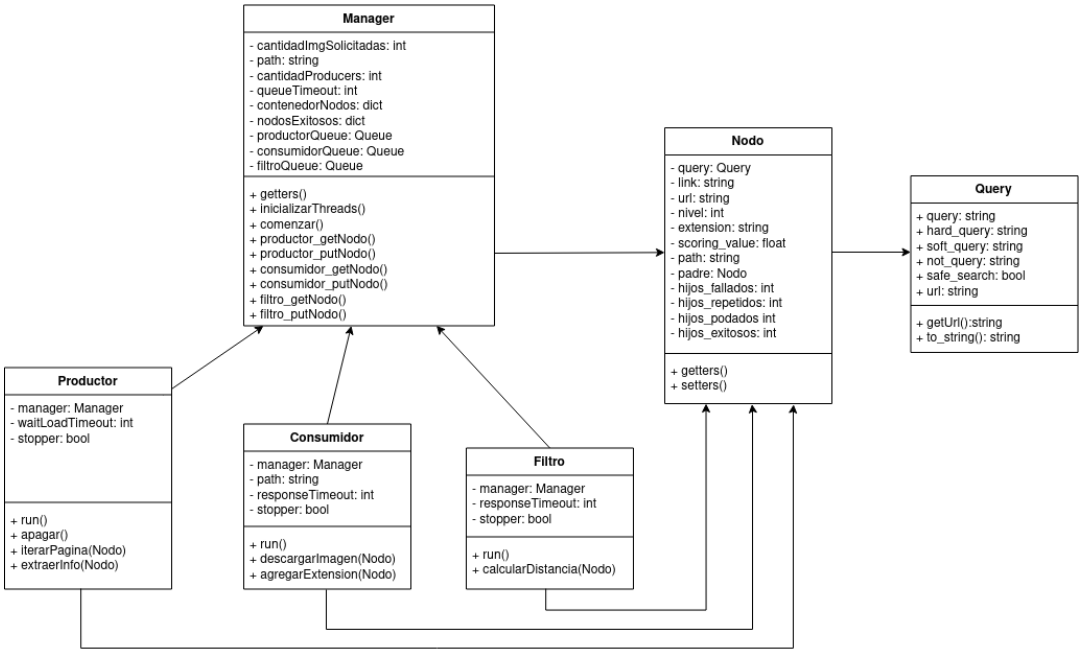


FIGURA 3.15: Diagrama de las clases implementadas y las relaciones entre sí.

clase específica que almacena de manera integral dicha información. Además, se introduce una nueva clase llamada **"Query"**, encargada de almacenar los parámetros asociados a la consulta y de disponer de un método para construir una URL relacionada con estos parámetros de búsqueda.

Los módulos Productor, Consumidor y Filtro se representan como tres clases distintas, y su intercomunicación se logra a través de la clase **"Manager"**. Esta última contiene las colas que actúan como intermediarios para los nodos que transitan entre las etapas del pipeline. En este diagrama, se visualiza cómo el diseño presentado en la Figura 3.5 se plasma de manera coherente en un diagrama de clases, facilitando la interacción entre los diversos componentes y ofreciendo una visión clara de la estructura del código.

El resultado del *web scraper* se traduce en dos archivos .csv, donde uno de ellos almacena los detalles de cada nodo generado, mientras que el otro contiene información específica sobre los nodos expandidos. Este último incluye datos significativos como la cantidad total de nodos generados, la cantidad de nodos duplicados, la cantidad de nodos que presentaron fallos, entre otros detalles relevantes. En las Figuras 3.16 y 3.17, se presentan ejemplos de estos archivos.

El código fuente y los recursos relacionados con este proyecto están disponibles en el repositorio de GitHub, accesible mediante el siguiente enlace: "`github.com/francoare/tesis-scraper`". Se alienta a los lectores a explorar

Referencia	Query	Padre	Nivel	URL	Path	Scoring Function
41d22d786ac4bfb3af3029c8cf0e8d04	[fundus picture,,,,off]	[fundus picture,,,,off]	1	https://www.google.com/search?q=fundu...	./imagen1.jpeg	0.000355980604995
57ff24efb80079ac3115065209fe95f5	[fundus picture,,,,off]	41d22d786ac4bfb3af3029c8cf0e8d04	2	https://www.google.com/search?q=fundu...	./imagen2.jpeg	0.00034161539418
53055a4ac1fe76be7778ff7222f1aa9c	[fundus picture,,,,off]	[fundus picture,,,,off]	1	https://www.google.com/search?q=fundu...	./imagen3.jpeg	0.00025974861486
5ba9a379ae2dedb48a24b9f071ee7d03	[fundus picture,,,,off]	[fundus picture,,,,off]	1	https://www.google.com/search?q=fundu...	./imagen4.jpeg	0.00029180128493
c9ffc561bee7a8e16434c2b1c9aac246	[fundus picture,,,,off]	[fundus picture,,,,off]	1	https://www.google.com/search?q=fundu...	./imagen5.jpeg	0.000480815474620
c35f6bc46b43ca8598899a3e39193ba9a	[fundus picture,,,,off]	41d22d786ac4bfb3af3029c8cf0e8d04	2	https://www.google.com/search?q=fundu...	./imagen6.jpeg	0.00029385639029
3dbdb48b80af8f575318d7736c91d27e	[fundus picture,,,,off]	5ba9a379ae2dedb48a24b9f071ee7d03	2	https://www.google.com/search?q=fundu...	./imagen7.jpeg	0.00038639669963
cfaaeb84cf01c210bbbe1b91753a1e41	[fundus picture,,,,off]	41d22d786ac4bfb3af3029c8cf0e8d04	2	https://www.google.com/search?q=fundu...	./imagen8.jpeg	0.000190410189196
97f4db2054fe36064081d9964dd2fe23	[fundus picture,,,,off]	c9ffc561bee7a8e16434c2b1c9aac246	2	https://www.google.com/search?q=fundu...	./imagen9.jpeg	0.000182801338121
70c0eeeb4a1498b1437a451a942ed8ba	[fundus picture,,,,off]	5ba9a379ae2dedb48a24b9f071ee7d03	2	https://www.google.com/search?q=fundu...	./imagen10.jpeg	0.00030480653647
3b05cf18e52a8f327c82d14d254fdbce	[fundus picture,,,,off]	41d22d786ac4bfb3af3029c8cf0e8d04	2	https://www.google.com/search?q=fundu...	./imagen11.jpeg	0.00038806429415
da6de7bba8eaa49e0f5f32a2ba337244	[fundus picture,,,,off]	41d22d786ac4bfb3af3029c8cf0e8d04	2	https://www.google.com/search?q=fundu...	./imagen12.webp	0.00035848690030
44e6199c314d3a310108dfd5b7cbfb88	[fundus picture,,,,off]	c9ffc561bee7a8e16434c2b1c9aac246	2	https://www.google.com/search?q=fundu...	./imagen13.jpeg	0.00035429344596
28370ad4b41b140621f70471514b44b3	[fundus picture,,,,off]	53055a4ac1fe76be7778ff7222f1aa9c	2	https://www.google.com/search?q=fundu...	./imagen14.png	0.000358457691780

FIGURA 3.16: Ejemplo del archivo .csv de salida en el cual se encuentran todos los nodos que se generaron exitosamente y que pasaron por todos los módulos del pipeline, junto con la información necesaria para su análisis y uso.

Referencia	Query	Padre	Nivel	Path	Scoring Function	Imagenes Exitosas	**Imagenes Recorridas**	TimeOuts	Repetidas	Descargas fallidas	Svdd Fallidas
consultaInicial	[color fundus photograph,,,,off]		0			143	**631**	487	0	1	0
52c0fa1b97eaaf309e7bc62697120db2	[color fundus photograph,,,,off]	consultaInicial	1	./52c0fa1b97eaaf309e7bc62697120db2.jpeg	0.0003568152721267	61	**100**	32	7	0	0
57ff24efb80079ac3115065209fe95f5	[color fundus photograph,,,,off]	consultaInicial	1	./57ff24efb80079ac3115065209fe95f5.jpeg	0.00034161539418 25	47	**100**	27	26	0	0
1a3214bea8fb05e0edfda f3d8cc0f530	[color fundus photograph,,,,off]	consultaInicial	1	./1a3214bea8fb05e0edfda f3d8cc0f530.jpeg	0.000137241786115 77	41	**100**	23	36	0	0
8b2ff404defcb8dc7c84f279937e7dfc	[color fundus photograph,,,,off]	consultaInicial	1	./8b2ff404defcb8dc7c84f279937e7dfc.jpeg	0.000114081943799 82	30	**100**	32	38	0	0
6c7d2d91e78cdf61d7adc9d5eb8ff7e	[color fundus photograph,,,,off]	consultaInicial	1	./6c7d2d91e78cdf61d7adc9d5eb8ff7e.jpeg	0.000256267361607 76	49	**99**	38	19	1	0
171c55234581f257d0ab17fb4920e28b	[color fundus photograph,,,,off]	consultaInicial	1	./171c55234581f257d0ab17fb4920e28b.jpeg	1.784295641785 2e-05	78	**100**	19	3	0	0
a3366f24483685b899da1ed7a8f097b6	[color fundus photograph,,,,off]	consultaInicial	1	./a3366f24483685b899da1ed7a8f097b6.jpeg	9.348245000786e-05	37	**100**	38	25	0	0
a3ca6b67a69626d109cbd75e4a665aac	[color fundus photograph,,,,off]	consultaInicial	1	./a3ca6b67a69626d109cbd75e4a665aac.jpeg	0.000314549308087 85	46	**100**	41	13	0	0
2077f70909b5ab4edf3e92e85d182afa	[color fundus photograph,,,,off]	consultaInicial	1	./2077f70909b5ab4edf3e92e85d182afa.jpeg	0.000122191046378 3	32	**100**	37	31	0	0
a93560d35cb509f776facebaab6319e4	[color fundus photograph,,,,off]	consultaInicial	1	./a93560d35cb509f776facebaab6319e4.jpeg	0.000100833440858 42	39	**100**	31	30	0	0
3ebed9e1d520d6314ca800be845afab3	[color fundus photograph,,,,off]	consultaInicial	1	./3ebed9e1d520d6314ca800be845afab3.jpeg	3.730072308074 6e-05	36	**100**	38	25	1	0
5a791a932eff34743ac4de3102a80711	[color fundus photograph,,,,off]	consultaInicial	1	./5a791a932eff34743ac4de3102a80711.jpeg	0.000334505471002 55	46	**100**	25	27	2	0
46b3eafcde9fbe4a4d9229349eb666fe	[color fundus photograph,,,,off]	consultaInicial	1	./46b3eafcde9fbe4a4d9229349eb666fe.jpeg	7.666953502623e-05	24	**99**	46	29	0	0

FIGURA 3.17: Ejemplo del archivo .csv de salida en el cual se encuentra la información de aquellos nodos que fueron expandidos.

el repositorio para obtener una comprensión más profunda de la implementación y acceder a los detalles específicos del código.

Capítulo 4

Resultados

Este capítulo reúne el análisis de los resultados obtenidos al realizar pruebas exhaustivas de las diferentes versiones diseñadas del *web scraper*, evaluando su rendimiento, eficacia y comportamiento en diversas aplicaciones. El objetivo principal es adquirir un conocimiento profundo de cada componente del *scraper* y analizar detalladamente las estrategias previamente mencionadas en el capítulo 3, a fines de identificar la más correcta.

En general, se realizaron experimentos para dos tipos de búsquedas: una enfocada a la descarga de imágenes de gatos, y otra de imágenes de fondo de ojo. A modo ilustrativo, la Figura 4.1 representa algunos ejemplos de imágenes que se espera descargar para cada una de las búsquedas, y algunas imágenes consideradas erróneas para cada descarga.

4.1. Análisis de costo computacional de los distintos módulos

Cada uno de los módulos tiene una responsabilidad específica en el sistema, lo que implica tareas distintas entre sí, algunas más complejas o costosas que otras.

Una medida útil para comparar el costo de cada módulo es el tiempo promedio que tarda en procesar una imagen o nodo. Para obtener esta medida, ejecutamos el programa y registramos los tiempos de procesamiento, o generación en el caso del Productor, de cada nodo. Para este análisis, se utilizó la biblioteca `Time` de Python, que permite calcular de manera precisa el tiempo que toma una porción específica del código en ejecución.

El primer módulo, Productor, realiza las tareas mostradas en la figura 3.8, las cuales se pueden dividir en dos partes: inicializar la página web abriendo el navegador, e iterar las imágenes para extraer el enlace de cada una. Para

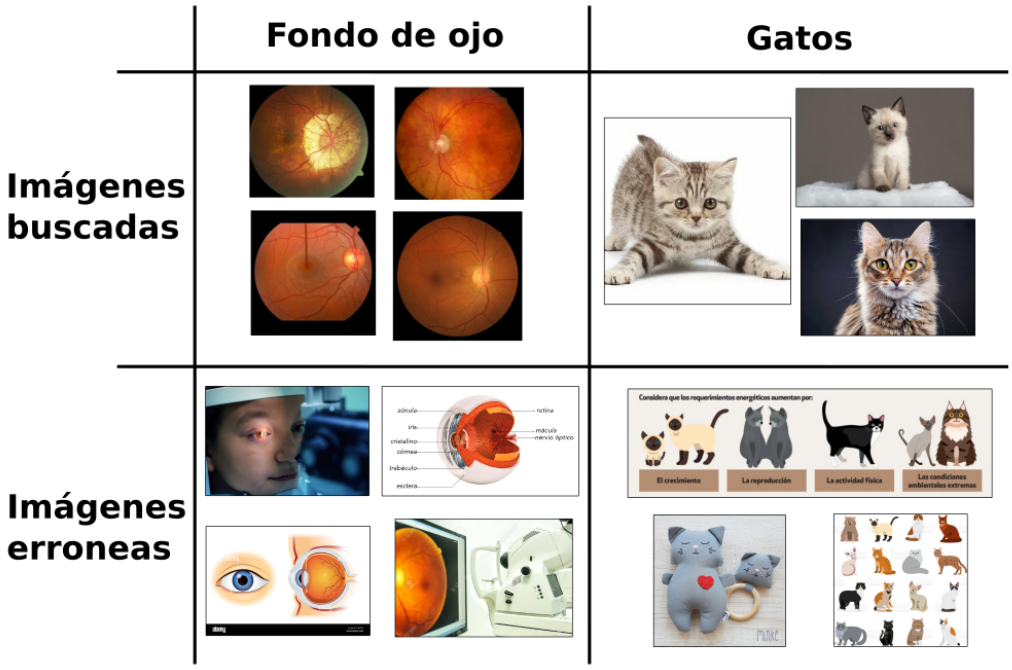


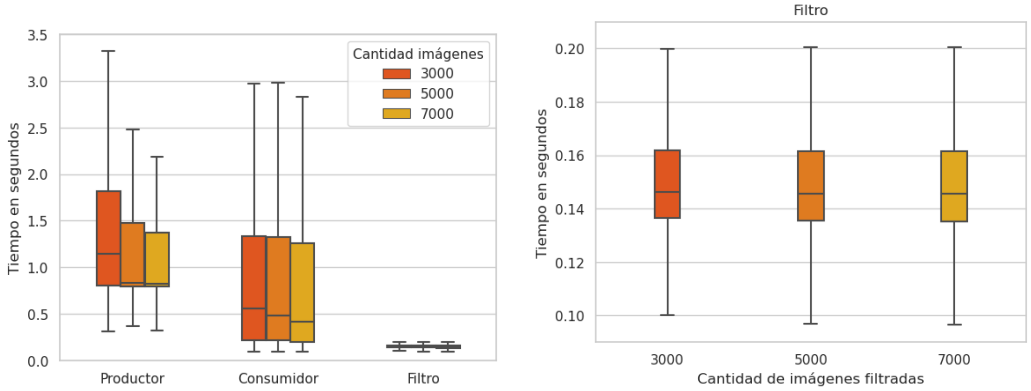
FIGURA 4.1: Ejemplo de imágenes de fondo de ojo y de gatos, diferenciando entre el tipo de imagen que se busca y los posibles resultados erróneos arrojados por el motor de búsqueda.

este análisis, medimos el tiempo de iterar sobre cada imagen, desde el clic inicial hasta la extracción completa de los datos asociados y la generación del nodo. En el módulo Consumidor, medimos el tiempo que demoran la descarga y el guardado como archivo en disco. Finalmente, medimos el tiempo que tarda cada nodo en ser procesado en el módulo Filtro, desde la solicitud al servidor con el detector de anomalías hasta la recepción del archivo JSON con el valor de la *scoring function*.

Se realizaron tres pruebas distintas utilizando cantidades diferentes de imágenes totales: 3.000, 5.000 y 7.000. Esto nos permite determinar si existe alguna variación al cambiar la cantidad total de imágenes a recorrer.

La consulta de entrada para las pruebas fue "gatos", por lo que en el módulo Filtro se utilizó un algoritmo entrenado con imágenes de gatos [1] desarrollado por un proyecto de investigación ajeno al trabajo realizado en esta tesis.

[1] En particular, se empleó una versión basada en deep Support Vector Data Description [32] entrenada con imágenes de la clase **"Gato"** provistas por el conjunto de datos **ImageNet**.



(a) Comparativa entre módulos (b) Detalle de tiempos del filtro

FIGURA 4.2: Boxplots comparando la distribución de tiempos de procesamiento por nodo de cada módulo, para 3.000, 5.000 y 7.000 imágenes descargadas.

La Figura 4.2 representa la distribución de tiempos de procesamiento para cada nodo en cada uno de los módulos, y para las diferentes cantidades de imágenes descargadas. En todos los casos, estos resultados fueron obtenidos con el enfoque secuencial *naive*. En primer lugar, se observa una notable diferencia entre los tiempos del módulo Filtro y el resto 4.2(a). Esto indica que el tiempo de procesamiento de este módulo no tiene un gran impacto en el tiempo general del *web scraper*, ya que el resto de los módulos demoran más en realizar sus tareas. Al analizar los tiempos del Productor y del Consumidor, se observa que los de este último están ligeramente por debajo del Productor, aunque también fluctúan más y sus valores están más distribuidos. En el caso del Productor, se puede observar que los valores del segundo cuartil están concentrados en un rango de valores muy pequeño, lo que *a priori* indica que en al menos el 50 % de los casos las demoras no son muy largas.

Al comparar las distintas pruebas realizadas, se aprecia que tanto el módulo Consumidor como el Filtro mantuvieron sus tiempos relativamente constantes, lo que indica una mayor estabilidad e independencia con la cantidad de nodos procesados. Esto no ocurre con el Productor, ya que se observa una diferencia en los tiempos entre las tres pruebas realizadas. Esto puede deberse a que el Productor depende en su mayoría del tiempo que tarda la página en cargar la imagen, lo que puede generar una mayor inestabilidad debido a variaciones en los tiempos de carga de internet, incluyendo la velocidad de conexión y el buen funcionamiento del servidor que contiene la página.

En resumen, no se observa un gran cambio entre las distintas pruebas, lo

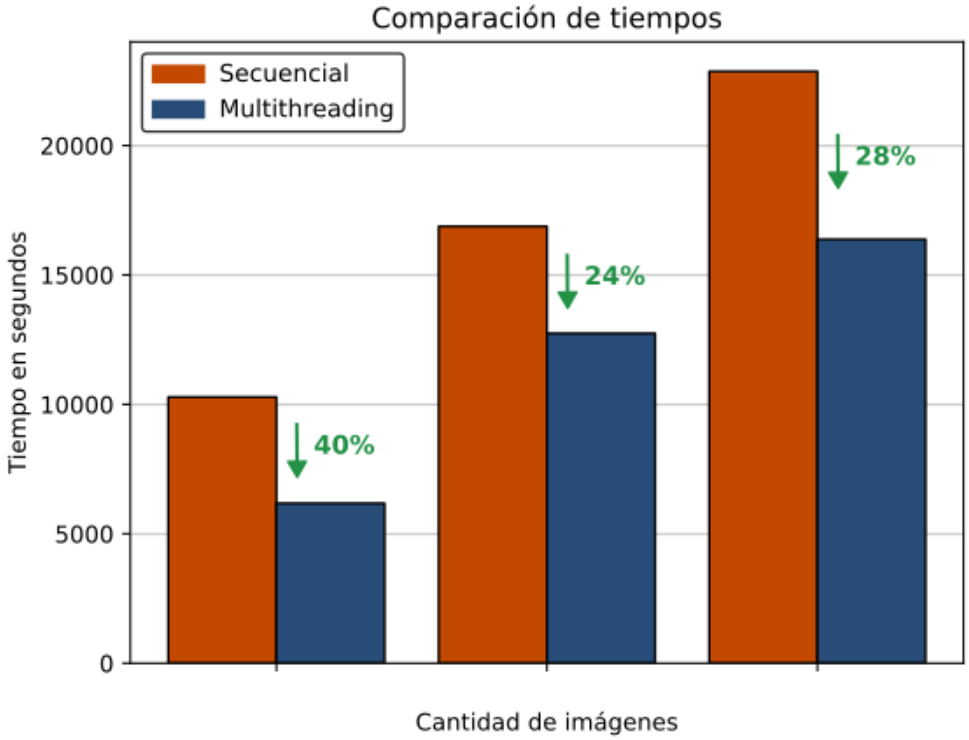


FIGURA 4.3: Comparación de la eficiencia temporal entre la versión secuencial y la versión multithreading

que sugiere que el comportamiento del *web scraper* en cuanto al tiempo consumido por cada nodo generado no depende significativamente de la cantidad total de nodos que se deben generar.

4.2. Comparación del enfoque secuencial con el *multithreading*

Con el propósito de evaluar la efectividad del enfoque *multithreading* por sobre el secuencial, se llevaron a cabo pruebas comparativas utilizando las mismas configuraciones que en ensayos anteriores, variando las cantidades totales de imágenes a descargar nuevamente, en grupos de 3.000, 5.000 y 7.000. En cada una de estas pruebas, se registró el tiempo total del programa necesario para adquirir todas las imágenes requeridas. Para la versión *multithreading*, se utilizaron un hilo para el Productor, uno para el Consumidor y uno para el Filtro.

Los resultados, plasmados en la Figura 4.3, evidencian una mejora sustancial al incorporar *threads* al proceso. Este impacto positivo valida la efectividad de la estrategia implementada. Por otro lado, se observa también que el cambio porcentual entre los tiempos de la versión secuencial y multithreading alcanza su pico máximo en la primer prueba. Esto implica una mayor notoriedad en el aumento de la eficiencia cuando se quiere descargar un conjunto pequeño de imágenes, tal vez producto del *overhead* de memoria producido por las colas. A pesar de este comportamiento, en todas las pruebas la versión multithreading mostró un rendimiento por encima del secuencial.

4.3. Análisis de grado de anomalía por imagen recorrida

En el capítulo 3.1, se detalló cómo ampliamos nuestra capacidad para obtener una mayor cantidad de imágenes más allá de las ofrecidas por el motor de búsqueda con la consulta inicial, que de por sí limita la cantidad de resultados. Aprovechando que cada imagen tiene asociado un botón de "ver más", el cual nos redirige a nuevos resultados relacionados con dicha imagen, cada nodo que completa todo el proceso del pipeline es consumido por el productor para explorar esa página y continuar generando nodos.

Dado que estas imágenes están relacionadas entre sí, es razonable esperar que el valor de la función de puntuación (*scoring function*) de cada una de estas nuevas imágenes sea similar al de la imagen principal. Para verificar esta hipótesis, realizamos una prueba con la consulta "fondo de ojo", utilizando un detector de anomalías desarrollado por un proyecto de investigación el cual fue entrenado específicamente con fotos de fondo de ojo[2]. Para esta prueba se descargaron un total de 10.000 imágenes, utilizando la versión *multithreading*.

En la Figura 4.4, se presenta un gráfico de densidad para los valores de la función de puntuación de las 10.000 imágenes. Estos valores varían desde 0 hasta aproximadamente 0.0007, siendo el valor 0.00005 el que tiene la mayor concentración de imágenes. Para comprender qué tipo de imagen se encuentra en cada rango de valores, tomamos cuatro muestras aleatorias de

[2]Esta versión del detector de anomalías consistió en una One Class SVM con una Radial Basis Function (RBF) como kernel, entrenada con un conjunto de 200 imágenes de fondo de ojo, a cada una de las cuales se les extrajo un vector de características utilizando una ResNet-18 pre-entrenada en ImageNet.

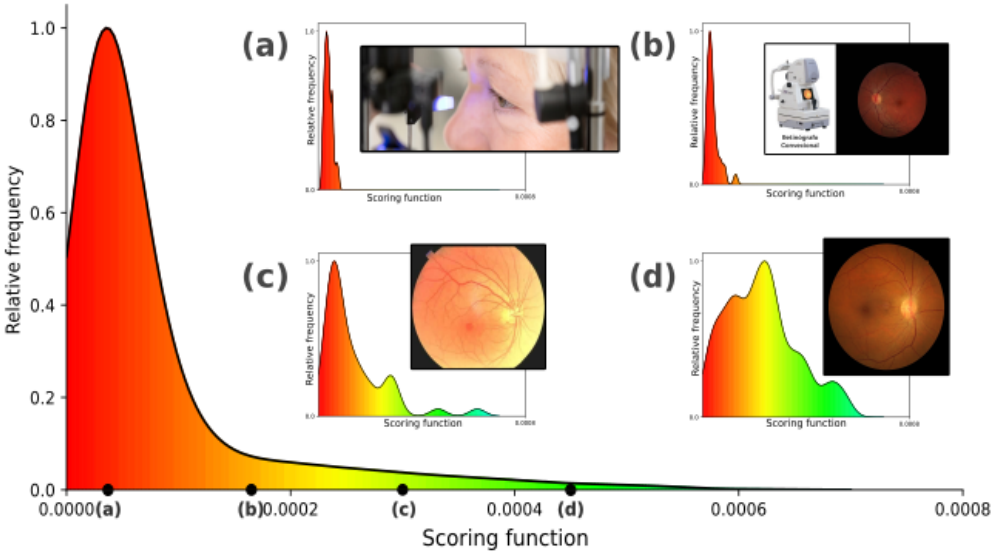


FIGURA 4.4: KDE (Kernel Density Estimation) representando la frecuencia relativa de imágenes descargadas según su valor de *scoring function*. Las imágenes (a), (b), (c) y (d) representan los KDEs asociados a las imágenes descargadas recorriendo el botón "ver más" a partir de la imagen ilustrada. Nótese que aquellas con valores de *scoring function* más bajos generan un mayor número de descargas anómalas.

imágenes que representan el tipo de imagen encontrado en ese rango (y detalladas en las Figuras 4.4 (a), (b), (c) y (d)). Nótese que a medida que las imágenes muestreadas presentan valores de la función de puntuación más alto, el tipo de imagen obtenida se aproxima más al tipo buscado, que es una foto de fondo de ojo.

Además, para cada una de las imágenes seleccionadas, la Figura 4.4 presenta el gráfico de densidad calculado con las imágenes descargadas a partir de su página asociada (una página como en la Figura 3.3). Se observa que al elegir expandir imágenes cuya función de puntuación indica una mejor calidad, obtenemos como resultado un mayor número de imágenes con valores altos de *scoring function* (y, por ende, menos anómalas).

En conclusión, podemos afirmar que es conveniente seleccionar aquellos nodos cuyas imágenes tienen un menor nivel de anomalía (con una función de puntuación más alta) para expandir, ya que los resultados obtenidos serán de mejor calidad. Esta observación se asemeja al criterio presentado en las secciones 3.5 y 3.6 para el diseño de las alternativas al modelo básico.

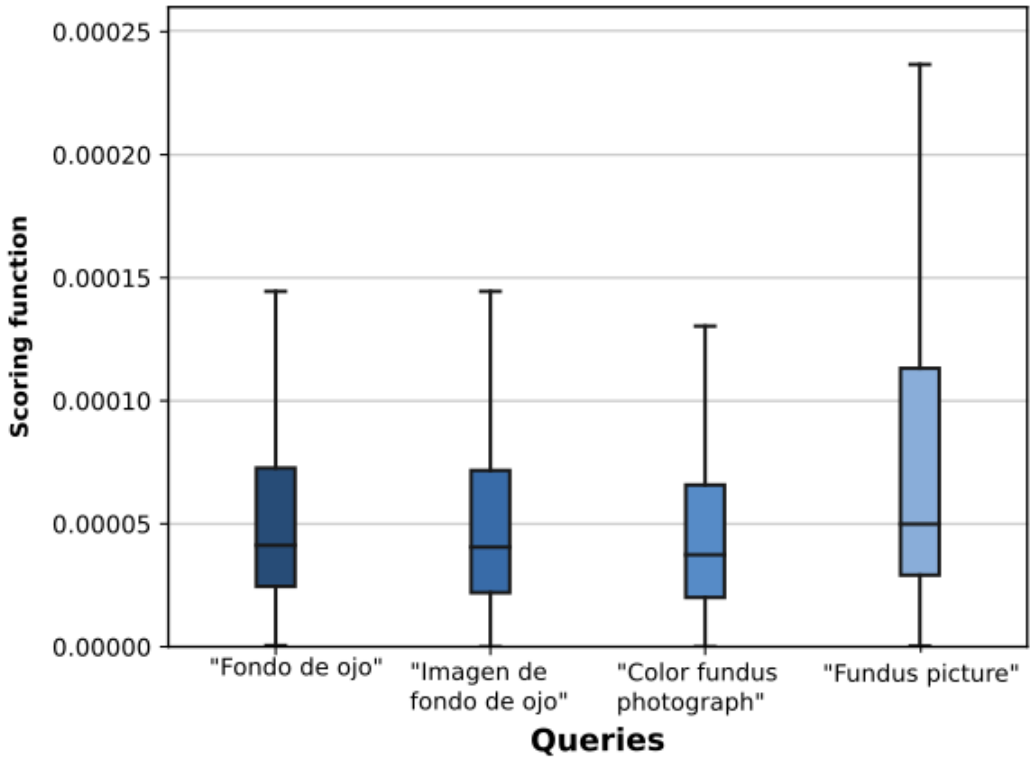


FIGURA 4.5: Distribución de los valores de *scoring function* según el tipo particular de búsqueda realizada.

4.4. Influencia de la consulta sobre los resultados

La elección de nuestra búsqueda inicial juega un papel crucial en la configuración de nuestros resultados, ya que es lo que el motor de búsqueda utiliza para seleccionar las imágenes que se mostrarán.

Para estudiar la consistencia de las imágenes descargadas con el objetivo esperado, se realizaron diferentes consultas en español y en inglés, y se capturaron los valores de la **scoring function** para todos los resultados obtenidos. Las imágenes descargadas correspondieron a aquellas asociadas con la consulta principal, sin expandir nuevos nodos; simplemente se descargaron las imágenes de la página raíz. En la figura 4.5, se presentan los boxplots que muestran la distribución de este valor para cada una de las consultas. Se observa que, en general, estos valores se distribuyen de manera muy similar para las primeras 3 búsquedas, con la excepción de la última, *"fundus picture"*, que arrojó valores algo mayores. Esta variación puede estar sugiriendo que búsquedas en inglés podrían proporcionar resultados más sólidos en este

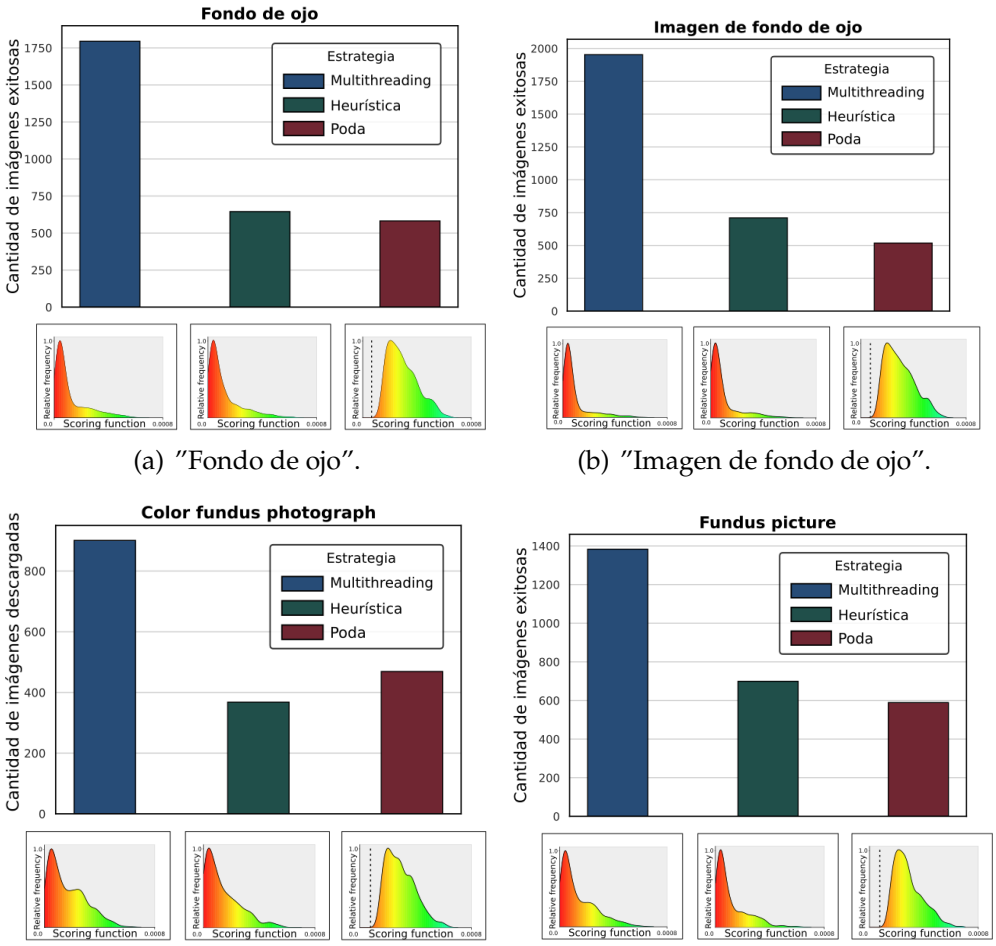
contexto específico, o que la terminología escogida es más acorde a la descarga esperada. Este hallazgo subraya la importancia de considerar el idioma y la formulación de la consulta inicial al diseñar estrategias de web scraping, ya que estos factores pueden influir significativamente en la calidad de datos obtenidos.

4.5. Evaluación de las estrategias de heurística y poda

En la sección 4.3, se destacó una posible mejora de los resultados al extender los nodos con mayor valor obtenido por la **scoring function**. En la figura 4.4, observamos que el diagrama de densidades principal exhibe una notable concentración de valores bajos, representando imágenes más anómalas (menos acordes al criterio de descarga). Dado este desafío y considerando el comportamiento descrito en la sección 4.3, evaluaremos cómo se altera esta problemática modificando la estrategia para la selección de nodos a expandir por parte del Productor.

En nuestro análisis, exploramos las dos estrategias basadas en la heurística (sección 3.5) y en poda (sección 3.6), planteadas como parte del diseño. Los resultados de ambas fueron contrastados con los de la versión *multithreading* sin estrategias adicionales. Los criterios de comparación se basaron en dos aspectos fundamentales: la cantidad de imágenes descargadas y la calidad de cada una de ellas.

La metodología de prueba se llevó a cabo ejecutando el programa durante un tiempo predeterminado, fijado en 90 minutos, utilizando cada una de las estrategias mencionadas. Al concluir este período, se analizó la cantidad de imágenes descargadas y se evaluó la calidad de las mismas, utilizando a tal fin el detector de anomalías. Cada prueba se replicó para cada una de las consultas previamente abordadas en la sección 4.3, permitiendo así una comparación exhaustiva de los resultados obtenidos en diversas situaciones. Para calcular el umbral de rechazo del método basado en poda, se utilizó un conjunto manualmente escogido de 20 imágenes de fondo de ojo. Posteriormente, se calculó el valor de **scoring function** para cada una de estas imágenes utilizando el algoritmo de detección de anomalías entrenado con imágenes de fondo de ojo. Finalmente, se eligió la imagen con el valor más bajo, tomando este como el umbral. En las pruebas realizadas, el umbral empleado fue de *0.0014*.



(a) "Fondo de ojo". (b) "Imagen de fondo de ojo".

(c) "Color fundus photograph". (d) "Fundus picture".

FIGURA 4.6: Gráficos de barra comparando la cantidad de imágenes descargadas en un lapso de 90 minutos para cada consulta, según el tipo particular de estrategia de búsqueda. Cada barra se acompaña con el KDE de frecuencias relativas según valor de *scoring function*. La línea punteada vertical en los casos con poda corresponde al valor de umbral empleado para rechazar imágenes durante la búsqueda.

En las figuras 4.6, se presentan los resultados de esta prueba, exhibiendo la cantidad de imágenes en cada caso junto con el gráfico de densidad, que permite analizar la distribución del nivel de anomalía del conjunto de imágenes obtenido.

Al comparar el método *multithreading* sin poda ni heurística contra el resto de las estrategias, puede observarse que fue capaz de descargar una mayor cantidad de imágenes, siendo **"imagen de fondo de ojo"** 4.6(b) la que obtuvo un mayor número en menor tiempo. No obstante, es importante señalar que el gráfico de densidad de los resultados indica un alto nivel de anomalía en estas imágenes. Esto sugiere que se trata de un conjunto "sucio", formado

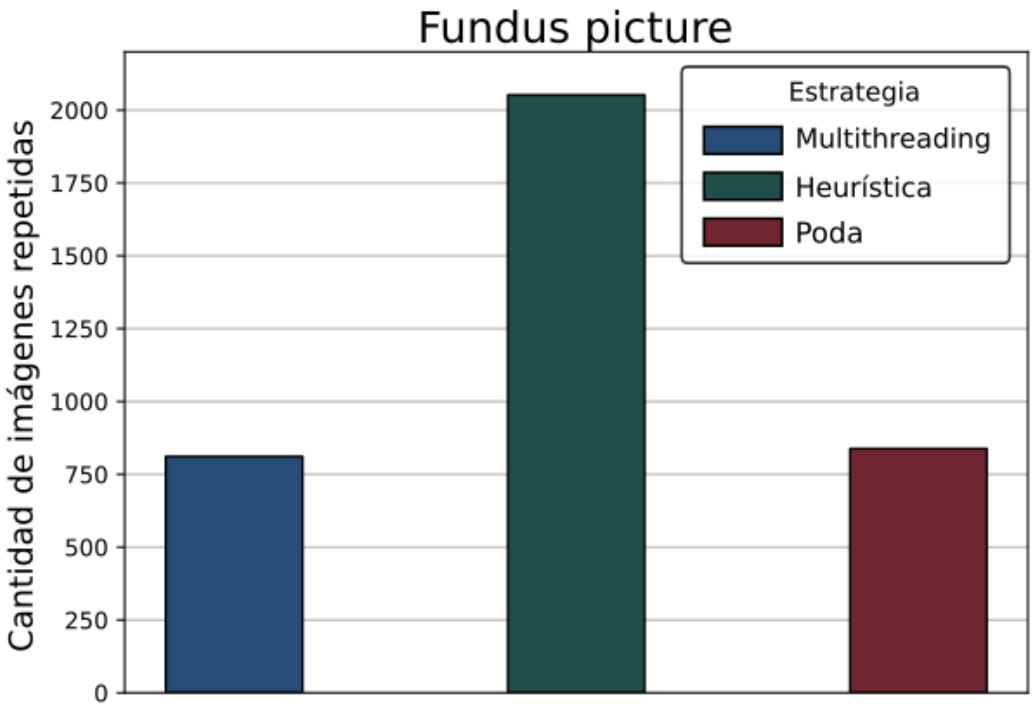


FIGURA 4.7: Cantidad de imágenes repetidas que fueron descartadas para cada estrategia utilizada.

por muchos datos espurios, no acordes con el criterio de descarga. Considerando este criterio, observamos que, aunque la consulta **"imagen de fondo de ojo"** fue la que obtuvo la mayor cantidad de imágenes, el gráfico de densidad asociado indica una mala calidad en este conjunto. En contraste, la consulta **"color fundus photograph"** (Figura 4.6(c)) fue la que logró la menor cantidad de imágenes, pero su gráfico de densidad muestra una mejora en la calidad de estas imágenes en comparación con la consulta mencionada anteriormente, en línea con lo observado en la sección anterior.

La estrategia basada en la heurística, por otro lado, presenta una disminución notoria en la cantidad de imágenes descargadas en comparación con el enfoque de *multithreading* sin estrategias. Esta reducción se evidencia consistentemente en las cuatro consultas realizadas. Uno de los motivos que provoca esta caída puede deberse al proceso mismo de búsqueda a partir de la página de imágenes relacionadas. Cuando se abre ésta mediante el botón **"ver más"** de una imagen, Google busca imágenes similares y las presenta en los resultados. Al aplicarse la técnica de heurística, siempre se priorizan imágenes poco anómalas, con alto valor en su **scoring function**. Debido a esta estrategia, las imágenes expandidas pueden ser muy similares entre sí, provocando que los resultados de sus páginas contengan las mismas imágenes

que ya fueron descargándose. Esto puede introducir demoras significativas en el proceso de descarga, debiendo procesarse muchas imágenes repetidas hasta dar con un conjunto de imágenes nuevas. Para validar esta hipótesis, se procedió a calcular la cantidad de imágenes repetidas que fueron descartadas del conjunto final mediante cada una de las estrategias para la consulta **"Fundus picture"**. En la Figura 4.7, se evidencia la notable disparidad existente entre la heurística y el resto de las estrategias en términos de la cantidad de imágenes repetidas, lo cual respalda nuestra hipótesis.

En cuanto a la calidad de las imágenes, no se observa una mejora muy notoria utilizando la heurística y en comparación con el *multithreading* básico. En todas las pruebas, persiste una tendencia hacia imágenes de baja calidad en el conjunto final, aunque, al igual que en la prueba anterior, la consulta **"Color fundus photograph"** (Figura 4.6(c)) destaca por tener datos de mayor calidad en relación con las demás consultas.

En lo que respecta al método con poda, se observa que la cantidad de imágenes obtenida mediante esta estrategia fue similar a la obtenida utilizando la heurística (Figura 4.6). No obstante, se observan diferencias notorias en la calidad de las imágenes obtenidas. Como se puede apreciar en los gráficos de densidad de la Figura 4.6, todos los valores del conjunto final están por encima del umbral, asegurando una alta calidad en cada una de estas imágenes. Por otro lado, es posible concluir que esta estrategia supera en calidad al enfoque basado en heurística, ya que logró obtener casi la misma cantidad de imágenes en cada consulta pero garantizando una mayor fiabilidad de las imágenes. En conclusión, la poda se posiciona como una estrategia efectiva para mejorar la calidad de las imágenes seleccionadas durante el proceso de *web scraping*.

En retrospectiva, cada una de las estrategias exhibió comportamientos distintos en términos de cantidad de imágenes descargadas y calidad de las mismas. Es evidente que la heurística resultó ser la opción menos favorable, ya que su implementación sólo condujo a una disminución en la cantidad de imágenes descargadas sin proporcionar una mejora en la calidad de estas. La Figura 4.8 ofrece una representación visual más clara de la distribución de los valores que indican la calidad de las imágenes en las distintas estrategias aplicadas a cada una de las consultas. Se observa una diferencia notable entre los valores de la Poda y el resto de los métodos.

Al analizar los resultados, llegamos a la conclusión de que la estrategia de Poda se destaca sobre las demás. Aunque puede descargar menos imágenes que el *multithreading* sin estrategias, el conjunto resultante de imágenes es sin

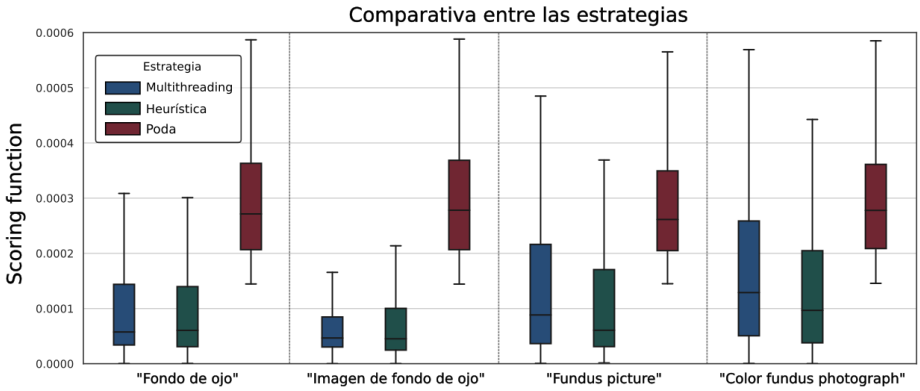


FIGURA 4.8: Distribución de valores de *scoring function* en los conjuntos descargados para cada consulta, según la estrategia aplicada.

lugar a duda de mayor calidad. La incorporación de este enfoque dentro del *web scraper* permite seleccionar un mayor número de imágenes relevantes, garantizando a la vez que las mismas se asemejen aún más al grupo esperado. Esta evidencia posiciona a la poda, entonces, como la elección preferida para el proceso de *web scraping* en el contexto de este proyecto.

4.6. Influencia del número de *threads* en los resultados

Hasta el momento, hemos logrado mejorar la velocidad de nuestro *web scraping* mediante la implementación de *multithreading*, además de obtener un conjunto de imágenes de alta calidad mediante la estrategia de poda.

En la última prueba realizada, evaluamos el rendimiento de la poda después de 90 minutos de ejecución con cuatro consultas distintas. La consulta que obtuvo la mayor cantidad de resultados utilizando esta estrategia fue **"Fundus Picture"**, con 598 imágenes descargadas exitosamente. Este enfoque evaluado utilizó un *thread* por cada módulo.

Para buscar un incremento en la cantidad de imágenes descargadas, aprovecharemos aún más el tiempo de espera para la carga de imágenes agregando más *threads* encargados del trabajo del Productor. Esto nos permitirá tener varios hilos iterando distintas páginas web al mismo tiempo, logrando así una mayor producción de imágenes.

La Figura 4.9 presenta los resultados al probar el *web scraper* con diferentes cantidades de threads sobre el módulo Productor utilizando la consulta

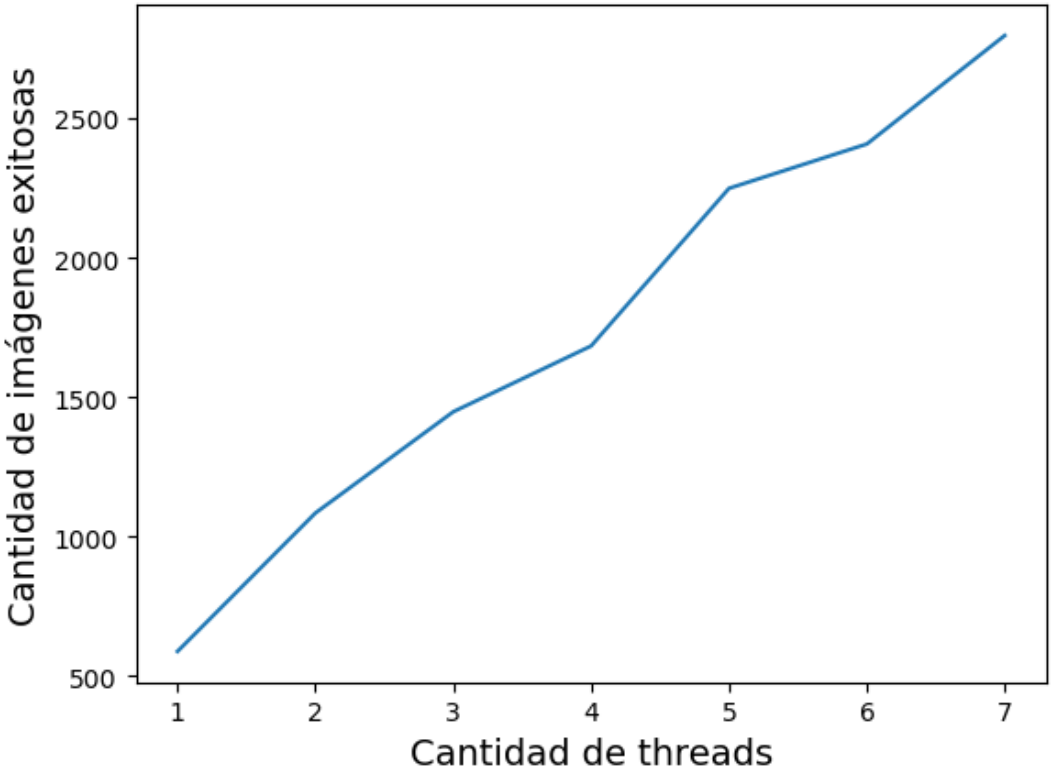


FIGURA 4.9: Relación entre la cantidad de imágenes exitosas y la cantidad de threads que se utilizan sobre el módulo Productor

"Fundus image". En el gráfico, observamos un aumento significativo en la cantidad de imágenes exitosas a medida que incrementamos la cantidad de threads, alcanzando hasta 2797 imágenes al utilizar 7 threads, más de cuatro veces la cantidad obtenida con un solo *thread* en el Productor. Debido a las limitaciones del hardware, no fue posible continuar aumentando la cantidad de threads debido al alto consumo del CPU requerido.

En conclusión, podemos multiplicar la cantidad de imágenes al agregar más *threads* y aprovechar aún más el diseño *multithreading*. En este caso, el límite fue el hardware utilizado en las pruebas, pero con equipos de mayor capacidad computacional, se podría lograr una mejora aún más significativa en relación con la cantidad de resultados obtenidos.

Capítulo 5

Conclusiones

En este trabajo final de carrera se propuso un algoritmo de *web scraping* para la descarga masiva de imágenes a partir de Google Images, de manera automática. El método presentado se basó en un recorrido en forma de árbol de los resultados de la búsqueda obtenida, complementado además por una conexión a un algoritmo de detección de anomalías, ideado para remover imágenes que no se asemejan al criterio de búsqueda. Además del enfoque modular y secuencial más tradicional, se diseñaron alternativas *multithreading* a las que se les incorporó soporte para búsqueda heurística y con criterio de poda, con el fin de optimizar tanto la cantidad como la calidad de las imágenes recopiladas.

Se realizó un análisis crítico del rendimiento del *web scraper* en diversas situaciones y consultas, considerando factores como el tiempo de procesamiento y la calidad de las imágenes obtenidas. Este análisis se llevó a cabo para cada versión diseñada, llegando a una versión óptima que maximiza tanto la cantidad como la calidad de las imágenes. Dicha optimización se logró mediante la aplicación del *multithreading* combinada con la estrategia de poda. Además, se logró incrementar la cantidad de imágenes al aumentar el número de *threads* que cumplen la tarea del Productor, iterando cada uno en una página diferente.

A pesar de haber alcanzado el objetivo inicial, el *web scraper* desarrollado presenta algunas limitaciones, ya que está directamente vinculado al código HTML de la página objetivo y, por lo tanto, sólo funciona con el motor de búsqueda de Google. Sin embargo, gracias al diseño en módulos independientes, modificar el módulo Productor para adaptarlo a una nueva página permitiría que el *web scraper* funcione con otros objetivos. Bastaría simplemente con abstraer el comportamiento común en una clase abstracta, y luego implementar cada módulo por separado, según el navegador.

Para culminar, se proponen una serie de ideas para trabajos futuros, que podrían servir para mejorar el rendimiento y el alcance del *scraper*, a saber:

- **Compatibilidad con otras páginas:** Dado que actualmente el *web scraper* está diseñado específicamente para el motor de búsqueda de Google, un trabajo futuro podría centrarse en la adaptación del módulo Productor para que sea compatible con otros motores de búsqueda. Esto ampliaría la versatilidad del *scraper* y permitiría su aplicación en diferentes contextos.

- **Sistema distribuido:** Explorar la implementación de un sistema distribuido para el *web scraper* podría ser un paso interesante. Dividir las tareas entre varios nodos o sistemas podría mejorar la eficiencia y el rendimiento del *scraper*, especialmente cuando se enfrenta a grandes volúmenes de datos.

- **Manejo de limitaciones de internet:** Dado que el rendimiento del módulo Productor está sujeto a la velocidad de carga de las imágenes en las páginas web, investigar estrategias para mitigar los efectos de la variabilidad en la velocidad de Internet podría mejorar la estabilidad y previsibilidad del *scraper*.

- **Implementación de reinicio y respaldo:** Incluir mecanismos de reinicio automático y respaldo de datos podría hacer que el *web scraper* sea más robusto frente a posibles interrupciones tales como cortes inesperados de energía.

Bibliografía

[1] Stephen Balaban. «Deep learning and face recognition: the state of the art». En: *Biometric and surveillance technology for human and activity identification XII* 9457 (2015), págs. 68-75.

[2] Andreas Bruns, Andreas Kornstadt y Dennis Wichmann. «Web application tests with selenium». En: *IEEE software* 26.5 (2009), págs. 88-91.

[3] Varun Chandola, Arindam Banerjee y Vipin Kumar. «Anomaly detection: A survey». En: *ACM computing surveys (CSUR)* 41.3 (2009), págs. 1-58.

[4] Datstrats. «¿Es legal el scraping, crawling o raspado web en España?» En: *Datstrats Blog* (2020). URL: https://datstrats.com/blog/scraping-es-legal-espana/.

[5] Eslegal. «Toda la información que necesitas sobre la legalidad del web scraping». En: *Eslegal Blog* (2020). URL: https://eslegal.info/cuando-es-legal-hacer-web-scraping/.

[6] Roy Fielding et al. *Hypertext transfer protocol–HTTP/1.1*. Inf. téc. 1999.

[7] Mario Rubiales Gómez. «HTML, CSS Y JAVASCRIPT». En: (2013).

[8] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep learning*. MIT press, 2016.

[9] Sorin Grigorescu et al. «A survey of deep learning techniques for autonomous driving». En: *Journal of Field Robotics* 37.3 (2020), págs. 362-386. DOI: https://doi.org/10.1002/rob.21918. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21918. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21918.

[10] Jie Gui et al. *A Survey on Self-supervised Learning: Algorithms, Applications, and Future Trends*. 2023. arXiv: 2301.05712 [cs.LG].

[11] Touseef Iqbal y Shaima Qureshi. «The survey: Text generation models in deep learning». En: *Journal of King Saud University-Computer and Information Sciences* 34.6 (2022), págs. 2515-2528.

[12] Yongpo Jia et al. «Fusing social networks with deep learning for volunteerism tendency prediction». En: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30. 1. 2016.

[13] Cheng Jin et al. «Label-Efficient Deep Learning in Medical Image Analysis: Challenges and Future Directions». En: (2023). arXiv: 2303.12484 [cs.CV].

[14] Junaidah Mohamed Kassim y Mahathir Rahmany. «Introduction to Semantic Search Engine». En: *2009 International Conference on Electrical Engineering and Informatics*. Vol. 02. 2009, págs. 380-386. DOI: 10.1109/ICEEI.2009.5254709.

[15] Moaiad Ahmad Khder. «Web Scraping or Web Crawling: State of Art, Techniques, Approaches and Application.» En: *International Journal of Advances in Soft Computing & Its Applications* 13.3 (2021).

[16] Minkyung Kim et al. «Unsupervised Deep One-Class Classification with Adaptive Threshold based on Training Dynamics». En: *2022 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, nov. de 2022. DOI: 10.1109/icdmw58026.2022.00014. URL: http://dx.doi.org/10.1109/ICDMW58026.2022.00014.

[17] Dimitrios Kouzis-Loukas. *Learning scrapy*. Packt Publishing Livery Place, 2016.

[18] Vlad Krotov, Leigh Johnson y Leiser Silva. «Tutorial: Legality and ethics of web scraping». En: (2020).

[19] Brian A LaMacchia. «Basis reduction algorithms and subset sum problems». En: (1991).

[20] Richard Lawson. *Web Scraping with Python*. Packt Publishing Ltd, 2015.

[21] Yann LeCun, Yoshua Bengio y Geoffrey Hinton. «Deep learning». En: *nature* 521.7553 (2015), págs. 436-444.

[22] Bamboo Legal. «¿Es legal el web scraping?» En: *Bamboo Legal Blog* (2019). URL: https://www.bamboo.legal/web-scraping-legal/.

[23] Geert Litjens et al. «A survey on deep learning in medical image analysis». En: *Medical Image Analysis* 42 (dic. de 2017), 60–88. ISSN: 1361-8415. DOI: 10.1016/j.media.2017.07.005. URL: http://dx.doi.org/10.1016/j.media.2017.07.005.

[24] Yiheng Liu et al. «Summary of ChatGPT-Related research and perspective towards the future of large language models». En: *Meta-Radiology* 1.2 (sep. de 2023), pág. 100017. ISSN: 2950-1628. DOI: 10.1016/j.metrad.2023.100017. URL: http://dx.doi.org/10.1016/j.metrad.2023.100017.

[25] Gary Marcus, Ernest Davis y Scott Aaronson. *A very preliminary analysis of DALL-E 2*. 2022. arXiv: 2204.13807 [cs.CV].

[26] Sunny Mehta y Gayatri Pandi. «An Improving Approach for Fast Web Scrapping Using Machine Learning and Selenium Automation». En: *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)* 8.10 (2019), págs. 434-438.

[27] Ryan Mitchell. *Web scraping with Python: Collecting more data from the modern web*. .º'Reilly Media, Inc.", 2018.

[28] Maxim Naumov et al. «Deep learning recommendation model for personalization and recommendation systems». En: *arXiv preprint arXiv:1906.00091* (2019).

[29] Maxim Naumov et al. «Deep learning training in facebook data centers: Design of scale-up and scale-out systems». En: *arXiv preprint arXiv:2003.09518* (2020).

[30] Pradheepan Raghavan y Neamat El Gayar. «Fraud detection using machine learning and deep learning». En: *2019 international conference on computational intelligence and knowledge economy (ICCIKE)*. IEEE. 2019, págs. 334-339.

[31] Leonard Richardson. *Beautiful soup documentation*. 2007.

[32] Lukas Ruff et al. «Deep one-class classification». En: *International conference on machine learning*. PMLR. 2018, págs. 4393-4402.

[33] Vinay Kumar Sriperambuduri, V Sireesha y Nagaratna P Hegde. «A Best Price Web Scraping Application for E-commerce Websites». En: *International Conference on Information and Management Engineering*. Springer. 2022, págs. 559-566.

[34] Andrew S Tanenbaum. *Redes de computadoras*. Pearson educación, 2003.

[35] Allan Vermeulen, Gabe Beged-Dov y Patrick Thompson. «The pipeline design pattern». En: *Proceedings of OOPSLA'95 Workshop on Design Patterns for Concurrent, Parallel, and Distributed Object-Oriented Systems*. Citeseer. 1995.

[36] Risqo Wahid, Joel Mero y Paavo Ritala. «Written by ChatGPT, illustrated by Midjourney: generative AI for content marketing». En: *Asia Pacific Journal of Marketing and Logistics* 35.8 (2023), págs. 1813-1822.

[37] WebScraper. *A Brief History of Web Scraping*. Accessed: 2023-12-14. 2021. URL: https://webscraper.io/blog/brief-history-of-web-scraping.

[38] Steven Euijong Whang et al. «Data collection and quality challenges in deep learning: A data-centric ai perspective». En: *The VLDB Journal* 32.4 (2023), págs. 791-813.

[39] Junliang Yu et al. «Self-Supervised Learning for Recommender Systems: A Survey». En: *IEEE Transactions on Knowledge and Data Engineering* (2023), págs. 1-20. DOI: 10.1109/TKDE.2023.3282907.

[40] Manqi Zhao y Venkatesh Saligrama. «Anomaly detection with score functions based on nearest neighbor graphs». En: *Advances in neural information processing systems* 22 (2009).